# Programmation avec le langage Python

*Version 0.1.263*

**Xavier Dupré**

**août 28, 2018**

# Table des matières


1 Introduction 1

2 Langage, variable et fonctions 3

3 Classes et programmation objet 95

4 Exceptions 155

5 Entrées Sorties 173

6 Modules 235

7 Expression régulières 249

8 Parallélisation 259

9 Interfaces graphiques 279

10 Données 315

11 Exemples 317

12 Précis 319

13 Index 349

CHAPITRE 1

# Introduction

Ceci est une relecture du livre que j'ai écrit en 2009 Programmation avec le langage Python [1] et disponible au format PDF [2]. Internet est le média le plus utilisé quant il s'agit d'apprendre à programmer et un module Python est plus pratique pour s'assurer qu'un code Python reste valide lors du passage à une nouvelle version du langage.

**Installation**

Les exemples de codes sont disponibles sous la forme d'un module python et des *notebooks* (page **??**) accessibles sur le site.

```
pip install teachpyx
```

1. http://www.editions-ellipses.fr/product_info.php?products_id=6891
2. http://www.xavierdupre.fr/site2013/index_documents.html

CHAPITRE 2

# Langage, variable et fonctions

## 2.1 Types et variables du langage python


— *Variables* (page 4)
— *Types immuables (ou immutable)* (page 5)
  — *Type ńărienă" ou None* (page 5)
— *Nombres réels et entiers* (page 5)
  — *Booléen* (page 7)
— *Chaîne de caractères* (page 8)
  — *Création d'une chaîne de caractères - str* (page 8)
  — *Manipulation d'une chaîne de caractères* (page 9)
  — *Formatage d'une chaîne de caractères* (page 11)
  — *tuple* (page 14)
  — *Nombres complexes* (page 15)
  — *bytes* (page 16)
— *Types modifiables ou mutable* (page 17)
  — *bytearray* (page 17)
  — *Liste* (page 17)
  — *Copie* (page 23)
  — *Dictionnaire* (page 26)
  — *Ensemble ou set* (page 31)
— *Extensions* (page 31)
  — *Fonction* `print`, `repr` *et conversion en chaîne de caractères* (page 31)
  — *Fonction* `eval` (page 32)
  — *Informations fournies par* python (page 32)
  — *Affectations multiples* (page 33)
  — *Hiérarchie des objets* (page 34)
  — *Objets internes* (page 34)
— *Commentaires accentués* (page 34)

### 2.1.1 Variables

Il est impossible d'écrire un programme sans utiliser de variable. Ce terme désigne le fait d'attribuer un nom ou identificateur à des informations : en les nommant, on peut manipuler ces informations beaucoup plus facilement. L'autre avantage est de pouvoir écrire des programmes valables pour des valeurs qui varient : on peut changer la valeur des variables, le programme s'exécutera toujours de la même manière et fera les mêmes types de calculs quelles que soient les valeurs manipulées. Les variables jouent un rôle semblable aux inconnues dans une équation mathématique.

L'ordinateur ne sait pas faire l'addition de plus de deux nombres mais cela suffit à calculer la somme de $n$ premiers nombres entiers. Pour cela, il est nécessaire de créer une variable intermédiaire qu'on appellera par exemple `somme` de manière à conserver le résultat des sommes intermédiaires.

<<<

```
n = 11
somme = 0                    # initialisation : la somme est nulle
for i in range(1, n):        # pour tous les indices de 1 à n exclu
    somme = somme + i        # on ajoute le i ème élément à somme
print(somme)
```

>>>

```
    55
```

---

**Définition D1 : variable**

Une variable est caractérisée par :

- **un identificateur :** il peut contenir des lettres, des chiffres, des blancs soulignés mais il ne peut commencer par un chiffre. Minuscules et majuscules sont différenciées. Il est aussi unique.
- **un type :** c'est une information sur le contenu de la variable qui indique à l'interpréteur *python*, la manière de manipuler cette information.

---

Comme le typage est dynamique en *python*, le type n'est pas précisé explicitement, il est implicitement liée à l'information manipulée. Par exemple, en écrivant, `x=3.4`, on ne précise pas le type de la variable `x` mais il est implicite car $x$ reçoit une valeur réelle : `x` est de type réel ou *float* en *python*. Pour leur première initialisation, une variable reçoit dans la plupart des cas une constante :

---

**Définition D2 : constante**

Les constantes sont le contraire des variables, ce sont toutes les valeurs numériques, chaînes de caractères, ..., tout ce qui n'est pas désigné par un nom. Les constantes possèdent un type mais pas d'identificateur.

---

Le langage *python* possède un certain nombre de types de variables déjà définis ou types fondamentaux à partir desquels il sera possible de définir ses propres types (voir chapitre *Classes* (page 95)). Au contraire de langages tels que le *C*, il n'est pas besoin de déclarer une variable pour signifier qu'elle existe, il suffit de lui affecter une valeur. Le type de la variable sera défini par le type de la constante qui lui est affectée. Le type d'une variable peut changer, il correspond toujours au type de la dernière affectation.

```
x   = 3.5       # création d'une variable nombre réel appelée x initialisée à 3.5
                # 3.5 est un réel, la variable est de type "float"
sc = "chaîne"   # création d'une variable chaîne de caractères appelée str
                # initialisée à "chaîne", sc est de type "str"
```

Pour tous les exemples qui suivront, le symbole # apparaîtra à maintes reprises. Il marque le début d'un commentaire que la fin de la ligne termine. Autrement dit, un commentaire est une information aidant à la compréhension du programme mais n'en faisant pas partie comme dans l'exemple qui suit.

```
x = 3           # affectation de la valeur entière 3 à la variable x
y = 3.0         # affectation de la valeur réelle 3.0 à la variable y
```

Le *python* impose une instruction par ligne. Il n'est pas possible d'utiliser deux lignes pour écrire une affectation à moins de conclure chaque ligne qui n'est pas la dernière par le symbole \ L'exemple suivant est impossible.

```
x =
    5.5
```

Il devrait être rédigé comme suit :

```
x =  \
    5.5
```

Avec ce symbole, les longues instructions peuvent être écrites sur plusieurs lignes de manière plus lisibles, de sorte qu'elles apparaissent en entier à l'écran. Si le dernier caractère est une virgule, il est implicite.

Les paragraphes suivant énumèrent les types incontournables en *python*. Ils sont classés le plus souvent en deux catégories : types *immuables* ou *modifiables*. Tous les types du langage *python* sont également des objets, c'est pourquoi on retrouve dans ce chapitre certaines formes d'écriture similaires à celles présentées plus tard dans le chapitre concernant les classes (*Classes* (page 95)).

### 2.1.2 Types immuables (ou immutable)

**Définition D3 : type immuable (ou immutable)**

Une variable de type immuable ne peut être modifiée. Une opération sur une variable de ce type entraîne nécessairement la création d'une autre variable du même type, même si cette dernière est temporaire.

Autrement dit, la simple instruction `x+=3` qui consiste à ajouter à la variable `x` la valeur `3` crée une seconde variable dont la valeur est celle de `x` augmentée de `3` puis à en recopier le contenu dans celui de la variable `x`. Les nombres sont des types immuables tout comme les chaînes de caractères et les `tuple` qui sont des tableaux d'objets. Il n'est pas possible de modifier une variable de ce type, il faut en recréer une autre du même type qui intègrera la modification.

#### Type ńărienă" ou None

*python* propose un type `None` pour signifier qu'une variable ne contient rien. La variable est de type `None` et est égale à `None`.

```
s = None
print(s)     # affiche None
```

Certaines fonctions utilisent cette convention lorsqu'il leur est impossible de retourner un résultat. Ce n'est pas la seule option pour gérer cette impossibilité : il est possible de générer une *exception* (page 155), de retourner une valeur par défaut ou encore de retourner `None`. Il n'y a pas de choix meilleur, il suffit juste de préciser la convention choisie.

Les fonctions sont définies au paragraphe *Fonctions* (page 53), plus simplement, ce sont des mini-programmes : elles permettent de découper un programme long en tâches plus petites. On les distingue des variables car leur nom est suivi d'une liste de constantes ou variables comprises entre parenthèses et séparées par une virgule.

### 2.1.3 Nombres réels et entiers

Documentation : Numeric Types ¿int, float, complex [3].

3. https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Il existe deux types de nombres en *python*, les nombres réels `float` et les nombres entiers `int`. L'instruction `x=3` crée une variable de type `int` initialisée à `3` tandis que `y=3.0` crée une variable de type `float` initialisée à `3.0`. Le programme suivant permet de vérifier cela en affichant pour les variables `x` et `y`, leurs valeurs et leurs types respectifs grâce à la fonction `type`.

<<<

```
x = 3
y = 3.0
print("x =", x, type(x))
print("y =", y, type(y))
```

>>>

```
    x = 3 <class 'int'>
    y = 3.0 <class 'float'>
```

La liste des opérateurs qui s'appliquent aux nombres réels et entiers suit. Les trois premiers résultats s'expliquent en utilisant la représentation en base deux. `8 << 1` s'écrit en base deux `100 << 1 = 1000`, ce qui vaut 16 en base décimale : les bits sont décalés vers la droite ce qui équivaut à multiplier par deux. De même, `7 & 2` s'écrit `1011 & 10 = 10`, qui vaut 2 en base décimale. Les opérateurs <<, >>, `|`, `&` sont des opérateurs bit à bit, ils se comprennent à partir de la représentation binaire des nombres entiers.

| opérateur | signification | exemple |
|---|---|---|
| `<< >>` | décalage à gauche, à droite | `x = 8 << 1` |
| `\|` | opérateur logique `ou` bit à bit | `x = 8 \| 1` |
| `&` | opérateur logique `et` bit à bit | `x = 11 & 2` |
| `+ -` | addition, soustraction | `x = y + z` |
| `+= -=` | addition ou soustraction puis affectation | `x += 3` |
| `* /` | multiplication, division | `x = y * z` |
| `//` | division entière, le résultat est de type réel si l'un des nombres est réel | `x = y // 3` |
| `%` | reste d'une division entière (modulo) | `x = y % 3` |
| `*= /=` | multiplication ou division puis affectation | `x *= 3` |
| `**` | puissance (entière ou non, racine carrée = ** 0.5) | `x = y ** 3` |

Les fonctions `int` et `float` permettent de convertir un nombre quelconque ou une chaîne de caractères respectivement en un entier (arrondi) et en un nombre réel.

<<<

```
x = int(3.5)
y = float(3)
z = int("3")
print("x:", type(x), "   y:", type(y), "   z:", type(z))
```

>>>

```
    x: <class 'int'>    y: <class 'float'>    z: <class 'int'>
```

Il peut arriver que la conversion en un nombre entier ne soit pas directe. Dans l'exemple qui suit, on cherche à convertir une chaîne de caractères (voir *Chaîne de caractères* (page 8)) en entier mais cette chaîne représente un réel. Il faut d'abord la convertir en réel puis en entier, c'est à ce moment que l'arrondi sera effectué.

```
i = int ("3.5")           # provoque une erreur
i = int (float ("3.5"))   # fonctionne
```

Les opérateurs listés par le tableau ci-dessus ont des priorités différentes, triés par ordre croissant. Toutefois, il est conseillé d'avoir recours aux parenthèses pour enlever les doutes : `3 * 2 ** 4 = 3 * (2 ** 4)`. La page Opertor Precedence [4] est plus complète à ce sujet.

*python* propose l'opérateur `//` pour les divisions entières et c'est une rare exception parmi les languages qui ne possèdent qu'un seul opérateur `/` qui retourne un entier pour une division entière excepté en *python* :

<<<

```
x = 11
y = 2
z = x // y       # le résultat est 5 et non 5.5 car la division est entière
zz = x / y       # le résultat est 5.5

print(z, zz)
```

>>>

```
    5 5.5
```

Pour éviter d'écrire le type `float`, on peut également écrire `11.0` de façon à spécifier explicitement que la valeur `11.0` est réelle et non entière. L'opérateur `//` permet d'effectuer une division entière lorsque les deux nombres à diviser sont réels, le résultat est un entier mais la variable est de type réel si l'un des nombres est de type réel.

### Booléen

Les booléens sont le résultat d'opérations logiques et ont deux valeurs possibles : `True` ou `False`. Voici la liste des opérateurs [5] qui s'appliquent aux booléens.

| opérateur | signification | exemple |
|---|---|---|
| `and or` | et, ou logique | `x = True or False` (résultat = True) |
| `not` | négation logique | `x = not x` |

<<<

```
x = 4 < 5
print(x)          # affiche True
print(not x)      # affiche False
```

>>>

```
    True
    False
```

Voici la liste des opérateurs de comparaisons [6] qui retournent des booléens. Ceux-ci s'applique à tout type, aux entiers, réels, chaînes de caractères, tuples... Une comparaison entre un entier et une chaîne de caractères est syntaxiquement correcte même si le résultat a peu d'intérêt.

| opérateur | signification | exemple |
|---|---|---|
| `< >` | inférieur, supérieur | `x = 5 < 5` |
| `<= >=` | inférieur ou égal, supérieur ou égal | `x = 5 <= 5` |
| `== !=` | égal, différent | `x = 5 == 5` |

4. https://docs.python.org/3/reference/expressions.html#operator-precedence
5. https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not
6. https://docs.python.org/3/library/stdtypes.html#comparisons

A l'instar des nombres réels, il est préférable d'utiliser les parenthèses pour éviter les problèmes de priorités d'opérateurs dans des expressions comme : `3 < x and x < 7`. Toutefois, pour cet exemple, *python* accepte l'écriture résumée qui enchaîne des comparaisons : `3 < x and x < 7` est équivalent à `3 < x < 7`. Il existe deux autres mots-clés qui retournent un résultat de type booléen :

| opérateur | signification | exemple |
|---|---|---|
| `is` | test d'identification | `"3" is str` |
| `in` | test d'appartenance | `3 in [3, 4, 5]` |

Ces deux opérateurs seront utilisés ultérieurement, `in` avec les listes, les dictionnaires, les boucles (paragraphe *Boucle for* (page 44)), `is` lors de l'étude des listes (paragraphe *Copie* (page 23) et des *classes* (page 95)). Bien souvent, les booléens sont utilisés de manière implicite lors de tests (paragraphe test_test) ce qui n'empêche pas de les déclarer explicitement.

```
x = True
y = False
```

### 2.1.4 Chaîne de caractères

#### Création d'une chaîne de caractères - str

**Définition D4 : chaîne de caractères**

Le terme ńăchaîne de caractèresă" ou *string* en anglais signifie une suite finie de caractères, autrement dit, du texte.

Ce texte est compris entre deux guillemets ou deux apostrophes, ces deux symboles sont interchangeables. Le type *python* est str[7]. L'exemple suivant montre comment créer une chaîne de caractères. Il ne faut pas confondre la partie entre guillemets ou apostrophes, qui est une constante, de la variable qui la contient.

<<<

```
t = "string = texte"
print(type(t), t)
t = 'string = texte, initialisation avec apostrophes'
print(type(t), t)

t = "morceau 1" \
    "morceau 2"    # second morceau ajouté au premier par l'ajout du symbole \,
# il ne doit rien y avoir après le symbole \,
# pas d'espace ni de commentaire
print(t)

t = """première ligne
seconde ligne"""   # chaîne de caractères qui s'étend sur deux lignes
print(t)
```

>>>

```
    <class 'str'> string = texte
    <class 'str'> string = texte, initialisation avec apostrophes
    morceau 1morceau 2
    première ligne
        seconde ligne
```

7. https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

La troisième chaîne de caractères créée lors de ce programme s'étend sur deux lignes. Il est parfois plus commode d'écrire du texte sur deux lignes plutôt que de le laisser caché par les limites de fenêtres d'affichage. *python* offre la possibilité de couper le texte en deux chaînes de caractères recollées à l'aide du symbole \ à condition que ce symbole soit le dernier de la ligne sur laquelle il apparaît. De même, lorsque le texte contient plusieurs lignes, il suffit de les encadrer entre deux symboles `"""` ou `'''` pour que l'interpréteur *python* considère l'ensemble comme une chaîne de caractères et non comme une série d'instructions.

Par défaut, le *python* ne permet pas l'insertion de caractères tels que les accents dans les chaînes de caractères, le paragraphe par_intro_accent_code explique comment résoudre ce problème. De même, pour insérer un guillemet dans une chaîne de caractères encadrée elle-même par des guillemets, il faut le faire précéder du symbole \. La séquence \ est appelée un extra-caractère (voir table extra_caractere) ou un caractère d'échappement.

| | |
|---|---|
| `"` | guillemet |
| `'` | apostrophe |
| `\n` | passage à la ligne |
| `\\` | insertion du symbole \ |
| `\%` | pourcentage, ce symbole est aussi un caractère spécial |
| `\t` | tabulation |
| `\r` | retour à la ligne, peu usité, il a surtout son importance lorsqu'on passe d'un système *Windows* à *Linux* car *Windows* l'ajoute automatiquement à tous ses fichiers textes |
| ... | Lire String and Bytes literals [8]. |

Liste des extra-caractères les plus couramment utilisés à l'intérieur d'une chaîne de caractères (voir page Lexical analysis [9]).

Il peut être fastidieux d'avoir à doubler tous les symboles \ d'un nom de fichier. Il est plus simple dans ce cas de préfixer la chaîne de caractères par `r` de façon à éviter que l'utilisation du symbole \ ne désigne un caractère spécial. Les deux lignes suivantes sont équivalentes :

```
s = "C:\\Users\\Dupre\\exemple.txt"
s = r"C:\Users\Dupre\exemple.txt"
```

Sans la lettre `"r"`, tous les \ doivent être doublés, dans le cas contraire, *python* peut avoir des effets indésirables selon le caractère qui suit ce symbole.

### Manipulation d'une chaîne de caractères

Une chaîne de caractères est semblable à un tableau et certains opérateurs qui s'appliquent aux tableaux s'appliquent également aux chaînes de caractères. Ceux-ci sont regroupés dans la table operation_string. La fonction `str` permet de convertir un nombre, un tableau, un objet (voir chapitre *Classes* (page 95)) en chaîne de caractères afin de pouvoir l'afficher. La fonction `len` retourne la longueur de la chaîne de caractères.

<<<

```
x = 5.567
s = str(x)
print(type(s), s)    # <type 'str'> 5.567
print(len(s))        # affiche 5
```

>>>

```
    <class 'str'> 5.567
    5
```

8. https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals
9. https://docs.python.org/3/reference/lexical_analysis.html

| opérateur | signification | exemple |
|---|---|---|
| `+` | concaténation de chaînes de caractères | `t = "abc" + "def"` |
| `+=` | concaténation puis affectation | `t += "abc"` |
| `in`, `not in` | une chaîne en contient-elle une autre ? | `"ed" in "med"` |
| `*` | répétition d'une chaîne de caractères | `t = "abc" * 4` |
| `[n]` | obtention du enième caractère, le premier caractère a pour indice 0 | `t = "abc"; print(t[0]) # donne a` |
| `[i:j]` | obtention des caractères compris entre les indices `i` et `j-1` inclus, le premier caractère a pour indice 0 | `t = "abc"; print(t[0:2]) # donne ab` |

Il existe d'autres fonctions qui permettent de manipuler les chaînes de caractères.

```
res = s.fonction (...)
```

Où `s` est une chaîne de caractères, `fonction` est le nom de l'opération que l'on veut appliquer à `s`, `res` est le résultat de cette manipulation.

La table string_method présente une liste non exhaustive des fonctions disponibles dont un exemple d'utilisation suit. Cette syntaxe `variable.fonction(arguments)` est celle des classes.

| | |
|---|---|
| `count( sub[, start[, end]])` | Retourne le nombre d'occurences de la chaîne de caractères `sub`, les paramètres par défaut `start` et `end` permettent de réduire la recherche entre les caractères d'indice `start` et `end` exclu. Par défaut, `start` est nul tandis que `end` correspond à la fin de la chaîne de caractères. |
| `find( sub[, start[, end]])` | Recherche une chaîne de caractères `sub`, les paramètres par défaut `start` et `end` ont la même signification que ceux de la fonction `count`. Cette fonction retourne -1 si la recherche n'a pas abouti. |
| `isalpha()` | Retourne `True` si tous les caractères sont des lettres, `False` sinon. |
| `isdigit()` | Retourne `True` si tous les caractères sont des chiffres, `False` sinon. |
| `replace( old, new[, count])` | Retourne une copie de la chaîne de caractères en remplaçant toutes les occurrences de la chaîne `old` par `new`. Si le paramètre optionnel `count` est renseigné, alors seules les `count` premières occurrences seront remplacées. |
| `split( [sep [, maxsplit]])` | Découpe la chaîne de caractères en se servant de la chaîne `sep` comme délimiteur. Si le paramètre `maxsplit` est renseigné, au plus `maxsplit` coupures seront effectuées. |
| `upper()` | Remplace les minuscules par des majuscules. |
| `lower()` | Remplace les majuscules par des minuscules. |
| `join ( li )` | `li` est une liste, cette fonction agglutine tous les éléments d'une liste séparés par `sep` dans l'expression `sep.join ( ["un", "deux"])`. |
| `startswith(prefix[, start[, end]])` | Teste si la chaîne commence par `prefix`. |
| `endswith(suffix[, start[, end]])` | Teste si la chaîne se termine par `suffix`. |
| ... | Lire String Methods [10]. |

<<<

```
st = "langage python"
st = st.upper()                 # mise en lettres majuscules
i = st.find("PYTHON")          # on cherche "PYTHON" dans st
print(i)                        # affiche 8
```

(suite sur la page suivante)

10. https://docs.python.org/3/library/stdtypes.html#string-methods

(suite de la page précédente)

```
print(st.count("PYTHON"))      # affiche 1
print(st.count("PYTHON", 9))   # affiche 0
```

>>>

```
    8
    1
    0
```

L'exemple suivant permet de retourner une chaîne de caractères contenant plusieurs éléments séparés par ";". La chaîne "un;deux;trois" doit devenir "trois;deux;un". On utilise pour cela les fonctionnalités split et join. L'exemple utilise également la fonctionnalité reverse des listes qui seront décrites plus loin dans ce chapitre. Il faut simplement retenir qu'une liste est un tableau. reverse retourne le tableau.

<<<

```
s = "un;deux;trois"
mots = s.split(";")         # mots est égal à ['un', 'deux', 'trois']
mots.reverse()              # retourne la liste, mots devient égal à
#                 ['trois', 'deux', 'un']
s2 = ";".join(mots)         # concaténation des éléments de mots séparés par ";"
print(s2)                    # affiche trois;deux;un
```

>>>

```
    trois;deux;un
```

### Formatage d'une chaîne de caractères

### Syntaxe %

*python* (printf-style String Formatting [11]) offre une manière plus concise de former une chaîne de caractères à l'aide de plusieurs types d'informations en évitant la conversion explicite de ces informations (type str) et leur concaténation. Il est particulièrement intéressant pour les nombres réels qu'il est possible d'écrire en imposant un nombre de décimales fixe. Le format est le suivant :

```
".... %c1  ....  %c2 " % (v1,v2)
```

c1 est un code choisi parmi ceux de la table format_print. Il indique le format dans lequel la variable v1 devra être transcrite. Il en est de même pour le code c2 associé à la variable v2. Les codes insérés dans la chaîne de caractères seront remplacés par les variables citées entre parenthèses après le symbole % suivant la fin de la chaîne de caractères. Il doit y avoir autant de codes que de variables, qui peuvent aussi être des constantes.

Voici concrètement l'utilisation de cette syntaxe :

<<<

```
x = 5.5
d = 7
s = "caractères"
res = "un nombre réel %f et un entier %d, une chaîne de %s, \n" \
      "un réel d'abord converti en chaîne de caractères %s" % (
          x, d, s, str(x+4))
print(res)
res = "un nombre réel " + str(x) + " et un entier " + str(d) + \
```

(suite sur la page suivante)

11. https://docs.python.org/3/library/stdtypes.html#old-string-formatting

(suite de la page précédente)

```
      ", une chaîne de " + s + \
      ",\n un réel d'abord converti en chaîne de caractères " + str(x+4)
print(res)
```

>>>

```
    un nombre réel 5.500000 et un entier 7, une chaîne de caractères,
    un réel d'abord converti en chaîne de caractères 9.5
    un nombre réel 5.5 et un entier 7, une chaîne de caractères,
     un réel d'abord converti en chaîne de caractères 9.5
```

La seconde affectation de la variable `res` propose une solution équivalente à la première en utilisant l'opérateur de concaténation `+`. Les deux solutions sont équivalentes, tout dépend des préférences de celui qui écrit le programme. La première option permet néanmoins un formatage plus précis des nombres réels en imposant par exemple un nombre défini de décimal. Le format est le suivant :

```
"%n.df" % x
```

`n` est le nombre de chiffres total et `d` est le nombre de décimales, `f` désigne un format réel indiqué par la présence du symbole `%`.

Exemple :

<<<

```
x = 0.123456789
print(x)              # affiche 0.123456789
print("%1.2f" % x)    # affiche 0.12
print("%06.2f" % x)   # affiche 000.12
```

>>>

```
    0.123456789
    0.12
    000.12
```

Il existe d'autres formats regroupés dans la table format_print. L'aide reste encore le meilleur réflexe car le langage *python* est susceptible d'évoluer et d'ajouter de nouveaux formats.

| | |
|---|---|
| `d` | entier relatif |
| `e` | nombre réel au format exponentiel |
| `f` | nombre réel au format décimal |
| `g` | nombre réel, format décimal ou exponentiel si la puissance est trop grande ou trop petite |
| `s` | chaîne de caractères |
| ... | Lire printf-style String Formatting [12]. |

### Méthode format

La méthode format [13] propose plus d'options pour formatter le texte et son usage est de plus en plus fréquent. La méthode interprète les accolades `{}` comme des codes qu'elle remplace avec les valeurs passée en argument. Le type n'importe plus. Quelques exemples :

<<<

12. https://docs.python.org/3/library/stdtypes.html#old-string-formatting
13. https://docs.python.org/3/library/stdtypes.html#str.format

```
print('{0}, {1}, {2}'.format('a', 'b', 'c'))    # le format le plus simple
print('{}, {}, {}'.format('a', 'b', 'c'))       # sans numéro
print('{2}, {1}, {0}'.format('a', 'b', 'c'))    # ordre changé
print('{0}{1}{0}'.format('abra', 'cad'))        # répétition
```

>>>

```
    a, b, c
    a, b, c
    c, b, a
    abracadabra
```

La méthode accepte aussi les paramètres nommées et des expressions.

<<<

```
print('Coordinates: {latitude}, {longitude}'.format(
    latitude='37.24N', longitude='-115.81W'))
coord = (3, 5)
print('X: {0[0]};  Y: {0[1]}'.format(coord))
```

>>>

```
    Coordinates: 37.24N, -115.81W
    X: 3;  Y: 5
```

L'alignement est plus simple :

<<<

```
print('A{:<30}B'.format('left aligned'))
print('A{:>30}B'.format('right aligned'))
print('A{:^30}B'.format('centered'))
print('A{:*^30}B'.format('centered'))
```

>>>

```
    Aleft aligned                  B
    A                 right alignedB
    A           centered           B
    A***********centered**********B
```

Format numérique :

<<<

```
print('{:.2%}'.format(19.0/22.0))
print("int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42))
print("int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin: {0:#b}".format(42))
print('{:,}'.format(1234567890))
```

>>>

```
    86.36%
    int: 42;  hex: 2a;  oct: 52;  bin: 101010
    int: 42;  hex: 0x2a;  oct: 0o52;  bin: 0b101010
    1,234,567,890
```

Dates :

<<<

```
import datetime
d = datetime.datetime.now()
print('{:%Y-%m-%d %H:%M:%S}'.format(d))
```

>>>

```
    2018-08-28 03:07:04
```

Le site pyformat [14] recense d'autres usages de la méthode format [15] comme l'affichage de chaînes de caractères tronquées.

<<<

```
print('{:10.5}'.format('formatages'))
```

>>>

```
    forma
```

### tuple

**Définition D5 : tuple**

Les tuple sont un tableau d'objets qui peuvent être de tout type. Ils ne sont pas modifiables (les tuple [16] sont *immuables* ou *immutable*).

Un tuple apparaît comme une liste d'objets comprise entre parenthèses et séparés par des virgules. Leur création reprend le même format :

```
x = (4,5)                 # création d'un tuple composé de 2 entiers
x = ("un",1,"deux",2)     # création d'un tuple composé de 2 chaînes de caractères
                          # et de 2 entiers, l'ordre d'écriture est important
x = (3,)                  # création d'un tuple d'un élément, sans la virgule,
                          # le résultat est un entier
```

Ces objets sont des vecteurs d'objets. Etant donné que les chaînes de caractères sont également des tableaux, ces opérations reprennent en partie celles des _string_paragraphe_chaine et décrites par le paragraphe Common Sequence Operations [17].

14. https://pyformat.info/
15. https://docs.python.org/3/library/stdtypes.html#str.format
16. https://docs.python.org/3/library/stdtypes.html#tuple
17. https://docs.python.org/3/library/stdtypes.html#typesseq-common

| | |
|---|---|
| `x in s` | vrai si `x` est un des éléments de `s` |
| `x not in s` | réciproque de la ligne précédente |
| `s + t` | concaténation de `s` et `t` |
| `s * n` | concatène `n` copies de `s` les unes à la suite des autres |
| `s[i]` | retourne le ième élément de `s` |
| `s[i:j]` | retourne un tuple contenant une copie des éléments de `s` d'indices *i* à *j* exclu |
| `s[i:j:k]` | retourne un tuple contenant une copie des éléments de `s` dont les indices sont compris entre *i* et *j* exclu, ces indices sont espacés de $k$ : $i, i+k, i+2k, i+3k, ...$ |
| `len(s)` | nombre d'éléments de `s` |
| `min(s)` | plus petit élément de `s`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `max(s)` | plus grand élément de `s` |
| `sum(s)` | retourne la somme de tous les éléments |

Les tuples ne sont pas **modifiables** (ou **mutable**), cela signifie qu'il est impossible de modifier un de leurs éléments. Par conséquent, la ligne d'affectation suivante n'est pas correcte :

```
a     = (4,5)
a [0] = 3       # déclenche une erreur d'exécution
```

Le message d'erreur suivant apparaît :

```
File "<pyshell#78>", line 1, in -toplevel-
    a[0]=3
TypeError: object doesn't support item assignment
```

Pour changer cet élément, il est possible de s'y prendre de la manière suivante :

```
a = (4,5)
a = (3,) + a[1:2]  # crée un tuple d'un élément concaténé
                   # avec la partie inchangée de a
```

### Nombres complexes

Il existe d'autres types comme le type complex [18] permettant de représenter les nombres complexes. Ce type numérique suit les mêmes règles et fonctionne avec les mêmes opérateurs (excepté les opérateurs de comparaisons) que ceux présentés au paragraphe type_nombre et décrivant les nombres.

<<<

```
print(complex(1, 1))
c = complex(1, 1)
print(c*c)
```

>>>

```
    (1+1j)
    2j
```

Le langage *python* offre la possibilité de créer ses propres types immuables (voir classe_slots_att) mais ils seront définis à partir des types immuables présentés jusqu'ici.

18. https://docs.python.org/3/library/functions.html#complex

### bytes

Le type bytes [19] représente un tableau d'octets. Il fonctionne quasiment pareil que le type str [20]. Les opérations qu'on peut faire dessus sont quasiment identiques :

| | |
|---|---|
| `count( sub[, start[, end]])` | Retourne le nombre d'occurences de la séquence d'octets `sub`, les paramètres par défaut `start` et `end` permettent de réduire la recherche entre les octets d'indice `start` et `end` exclu. Par défaut, `start` est nul tandis que `end` correspond à la fin de la séquence d'octets. |
| `find( sub[, start[, end]])` | Recherche une séquence d'octets `sub`, les paramètres par défaut `start` et `end` ont la même signification que ceux de la fonction `count`. Cette fonction retourne -1 si la recherche n'a pas abouti. |
| `replace( old, new[, count])` | Retourne une copie de la séquence d'octets en remplaçant toutes les occurrences de la séquence `old` par `new`. Si le paramètre optionnel `count` est renseigné, alors seules les `count` premières occurrences seront remplacées. |
| `partition( [sep [, maxsplit]])` | Découpe la séquence d'octets en se servant de la séquence `sep` comme délimiteur. Si le paramètre `maxsplit` est renseigné, au plus `maxsplit` coupures seront effectuées. |
| `join ( li )` | `li` est une liste, cette fonction agglutine tous les éléments d'une liste séparés par `sep` dans l'expression `sep.join ( ["un", "deux"])`. |
| `startswith(prefix[, start[, end]])` | Teste si la chaîne commence par `prefix`. |
| `endswith(suffix[, start[, end]])` | Teste si la chaîne se termine par `suffix`. |
| ... | Lire Bytes and Bytearray Operations [21]. |

Pour déclarer un tableau de *bytes*, il faut préfixer une chaîne de caractères par **b** :

<<<

```
b = b"345"
print(b, type(b))

b = bytes.fromhex('2Ef0 F1f2  ')
print(b, type(b))
```

>>>

```
    b'345' <class 'bytes'>
    b'.\xf0\xf1\xf2' <class 'bytes'>
```

Le type *bytes* est très utilisé quand il s'agit de convertit une chaîne de caractères d'un encoding [22] à l'autre.

<<<

```
b = "abc".encode("utf-8")
s = b.decode("ascii")
print(b, s)
print(type(b), type(s))
```

>>>

19. https://docs.python.org/3/library/stdtypes.html#bytes
20. https://docs.python.org/3/library/stdtypes.html#string-methods
21. https://docs.python.org/3/library/stdtypes.html#bytes-methods
22. https://fr.wikipedia.org/wiki/Codage_des_caract%C3%A8res

```
    b'abc' abc
    <class 'bytes'> <class 'str'>
```

Les encoding [23] sont utiles dès qu'une chaîne de caractères contient un caractère non anglais (accent, sigle...). Les bytes sont aussi très utilisés pour sérialiser [24] un objet.

### 2.1.5 Types modifiables ou mutable

Les types modifiables sont des conteneurs (ou containers en anglais) : ils contiennent d'autres objets, que ce soit des nombres, des chaînes de caractères ou des objets de type modifiable. Plutôt que d'avoir dix variables pour désigner dix objets, on en n'utilise qu'une seule qui désigne ces dix objets.

---

**Définition D6 : type modifiable (ou mutable)**

Une variable de type modifiable peut être modifiée, elle conserve le même type et le même identificateur. C'est uniquement son contenu qui évolue.

---

On pourrait penser que les types modifiables sont plus pratiques à l'usage mais ce qu'on gagne en souplesse, on le perd en taille mémoire.

<<<

```
import sys
li = [3, 4, 5, 6, 7]
tu = (3, 4, 5, 6, 7)
print(sys.getsizeof(li), sys.getsizeof(tu))
```

>>>

```
    104 88
```

#### bytearray

Le type bytearray [25] est la version *mutable* du type *bytes* (page 16).

#### Liste

#### Définition et fonctions

---

**Définition D7 : liste**

Les listes sont des collections d'objets qui peuvent être de tout type. Elles sont modifiables.

---

Une liste apparaît comme une succession d'objets compris entre crochets et séparés par des virgules. Leur création reprend le même format :

23. https://fr.wikipedia.org/wiki/Codage_des_caract%C3%A8res
24. https://fr.wikipedia.org/wiki/S%C3%A9rialisation
25. https://docs.python.org/3/library/functions.html#bytearray

```
x = [4,5]                 # création d'une liste composée de deux entiers
x = ["un",1,"deux",2]     # création d'une liste composée de
                          # deux chaînes de caractères
                          # et de deux entiers, l'ordre d'écriture est important
x = [3,]                  # création d'une liste d'un élément, sans la virgule,
                          # le résultat reste une liste
x = [ ]                   # crée une liste vide
x = list ()               # crée une liste vide
y = x [0]                 # accède au premier élément
y = x [-1]                # accède au dernier élément
```

Ces objets sont des listes chaînées d'autres objets de type quelconque (immuable ou modifiable). Il est possible d'effectuer les opérations qui suivent. Ces opérations reprennent celles des *tuple* (page 14) (voir *opération tuple* (page 15)) et incluent d'autres fonctionnalités puisque les listes sont modifiables. Il est donc possible d'insérer, de supprimer des éléments, de les trier. La syntaxe des opérations sur les listes est similaire à celle des opérations qui s'appliquent sur les chaînes de caractères, elles sont présentées par la table suivante.

| | |
|---|---|
| `x in l` | vrai si `x` est un des éléments de `l` |
| `x not in l` | réciproque de la ligne précédente |
| `l + t` | concaténation de `l` et `t` |
| `l * n` | concatène `n` copies de `l` les unes à la suite des autres |
| `l[i]` | retourne l'élément i$^text{ème}$ élément de `l`, à la différence des tuples, l'instruction `l [i] = "a"` est valide, elle remplace l'élément `i` par `"a"`. Un indice négatif correspond à la position `len(l)+i`. |
| `l[i:j]` | retourne une liste contenant les éléments de `l` d'indices *i* à *j* exclu. Il est possible de remplacer cette sous-liste par une autre en utilisant l'affectation `l[i:j] = l2` où `l2` est une autre liste (ou un tuple) de dimension différente ou égale. |
| `l[i:j:k]` | retourne une liste contenant les éléments de `l` dont les indices sont compris entre *i* et *j* exclu, ces indices sont espacés de *k* : $i, i+k, i+2k, i+3k, ...$ Ici encore, il est possible d'écrire l'affectation suivante : `l[i:j:k] = l2` mais `l2` doit être une liste (ou un tuple) de même dimension que `l[i:j:k]`. |
| `len(l)` | nombre d'éléments de `l` |
| `min(l)` | plus petit élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `max(l)` | plus grand élément de `l` |
| `sum(l)` | retourne la somme de tous les éléments |
| `del l [i:j]` | supprime les éléments d'indices entre `i` et `j` exclu. Cette instruction est équivalente à `l [i:j] = []`. |
| `list (x)` | convertit `x` en une liste quand cela est possible |
| `l.count (x)` | Retourne le nombre d'occurrences de l'élément `x`. Cette notation suit la syntaxe des classes développée au chapitre *Classes* (page 95). `count` est une méthode de la classe `list`. |
| `l.index (x)` | Retourne l'indice de la première occurrence de l'élément `x` dans la liste `l`. Si celle-ci n'existe pas, une exception est déclenchée (voir le paragraphe *Exceptions* (page 155)) |
| `l.append (x)` | Ajoute l'élément `x` à la fin de la liste `l`. Si `x` est une liste, cette fonction ajoute la liste `x` en tant qu'élément, au final, la liste `l` ne contiendra qu'un élément de plus. |
| `l.extend (k)` | Ajoute tous les éléments de la liste `k` à la liste `l`. La liste `l` aura autant d'éléments supplémentaires qu'il y en a dans la liste `k`. |
| `l.insert(i,x)` | Insère l'élément `x` à la position `i` dans la liste `l`. |
| `l.remove (x)` | Supprime la première occurrence de l'élément `x` dans la liste `l`. S'il n'y a aucune occurrence de `x`, cette méthode déclenche une exception. |
| `l.pop ([i])` | Retourne l'élément `l[i]` et le supprime de la liste. Le paramètre `i` est facultatif, s'il n'est pas précisé, c'est le dernier élément qui est retourné puis supprimé de la liste. |
| `l.reverse (x)` | Retourne la liste, le premier et dernier élément échange leurs places, le second et l'avant dernier, et ainsi de suite. |
| `l.sort ([key=None, reverse=False])` | Cette fonction trie la liste par ordre croissant. Le paramètre `key` est facultatif, il permet de préciser la fonction qui précise clé de comparaison qui doit être utilisée lors du tri. Si `reverse` est `True`, alors le tri est décroissant. Lire Sorting HOW TO [26]. |

26. https://docs.python.org/3/howto/sorting.html#sortinghowto

### Exemples

L'exemple suivant montre une utilisation de la méthode `sort`.

<<<

```
x = [9, 0, 3, 5, 4, 7, 8]           # définition d'une liste
print(x)                          # affiche cette liste
x.sort()                          # trie la liste par ordre croissant
print(x)                          # affiche la liste triée
```

>>>

```
    [9, 0, 3, 5, 4, 7, 8]
    [0, 3, 4, 5, 7, 8, 9]
```

Pour classer les objets contenus par la liste mais selon un ordre différent, il faut définir une fonction qui détermine un ordre entre deux éléments de la liste. C'est la fonction `compare` de l'exemple suivant.

<<<

```
def compare_key(x):
    return -x


x = [9, 0, 3, 5, 4, 7, 8]
x.sort(key=compare_key)      # trie la liste x à l'aide de la fonction compare
# cela revient à la trier par ordre décroissant
print(x)
```

>>>

```
    [9, 8, 7, 5, 4, 3, 0]
```

L'exemple suivant illustre un exemple dans lequel on essaye d'accéder à l'indice d'un élément qui n'existe pas dans la liste :

```
x = [9,0,3,5,0]
print(x.index(1))    # cherche la position de l'élément 1
```

Comme cet élément n'existe pas, on déclenche ce qu'on appelle une exception qui se traduit par l'affichage d'un message d'erreur. Le message indique le nom de l'exception générée (`ValueError`) ainsi qu'un message d'information permettant en règle générale de connaître l'événement qui en est la cause.

```
Traceback (most recent call last):
  File "c:/temp/temp", line 2, in -toplevel-
    print(x.index(1))
ValueError: list.index(x): x not in list
```

Pour éviter cela, on choisit d'intercepter l'exception (voir paragraphe *Exceptions* (page 155)).

<<<

```
x = [9, 0, 3, 5, 0]
try:
    print(x.index(1))
except ValueError:
    print("1 n'est pas présent dans la liste x")
else:
    print("trouvé")
```

>>>

```
    1 n'est pas présent dans la liste x
```

### Fonction range

Les listes sont souvent utilisées dans des boucles ou notamment par l'intermédiaire de la fonction range [27]. Cette fonction retourne un itérateur [28] sur des entiers. Nous verrons les itérateurs plus tard. Disons pour le moment les itérateurs ont l'apparence d'un ensembe mais ce n'en est pas un.

```
range (debut, fin [,marche])
```

Retourne une liste incluant tous les entiers compris entre `debut` et `fin` exclu. Si le paramètre facultatif `marche` est renseigné, la liste contient tous les entiers `n` compris `debut` et `fin` exclu et tels que `n - debut` soit un multiple de `marche`.

Exemple :

<<<

```
print(range(0, 10, 2))
print(list(range(0, 10, 2)))
```

>>>

```
    range(0, 10, 2)
    [0, 2, 4, 6, 8]
```

Cette fonction est souvent utilisée lors de boucles *for* (page 44) pour parcourir tous les éléments d'un tuple, d'une liste, d'un dictionnaire... Le programme suivant permet par exemple de calculer la somme de tous les entiers impairs compris entre 1 et 20 exclu.

```
s = 0
for n in range (1,20,2) :  # ce programme est équivalent à
    s += n                 # s = sum (range(1,20,2))
```

Le programme suivant permet d'afficher tous les éléments d'une liste.

<<<

```
x = ["un", 1, "deux", 2, "trois", 3]
for n in range(0, len(x)):
    print("x[%d] = %s" % (n, x[n]))
```

>>>

```
    x[0] = un
    x[1] = 1
    x[2] = deux
    x[3] = 2
    x[4] = trois
    x[5] = 3
```

27. https://docs.python.org/3/library/functions.html#func-range
28. https://fr.wikipedia.org/wiki/It%C3%A9rateur

### Boucles et listes

Il est possible aussi de ne pas se servir des indices comme intermédiaires pour accéder aux éléments d'une liste quand il s'agit d'effectuer un même traitement pour tous les éléments de la liste `x`.

<<<

```
x = ["un", 1, "deux", 2]
for el in x:
    print("la liste inclut : ", el)
```

>>>

```
    la liste inclut :  un
    la liste inclut :  1
    la liste inclut :  deux
    la liste inclut :  2
```

L'instruction `for el in x :` se traduit littéralement par : *pour tous les éléments de la liste, faire....*

Il existe également des notations abrégées lorsqu'on cherche à construire une liste à partir d'une autre. Le programme suivant construit la liste des entiers de 1 à 5 à partir du résultat retourné par la fonction `range`.

<<<

```
y = list()
for i in range(0, 5):
    y.append(i+1)
print(y)
```

>>>

```
    [1, 2, 3, 4, 5]
```

Le langage *python* offre la possibilité de résumer cette écriture en une seule ligne. Cette syntaxe sera reprise au paragraphe *Listes, boucle for, liste en extension* (page 45).

<<<

```
y = [i+1 for i in range(0, 5)]
print(y)
```

>>>

```
    [1, 2, 3, 4, 5]
```

Cette définition de liste peut également inclure des tests ou des boucles imbriquées.

<<<

```
y = [i for i in range(0, 5) if i % 2 == 0]    # sélection les éléments pairs
print(y)                                      # affiche [0,2,4]
z = [i+j for i in range(0, 5)
     for j in range(0, 5)]      # construit tous les nombres i+j possibles
print(z)
```

>>>

```
    [0, 2, 4]
    [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
```

### Collage de séquences, fonction zip

On suppose qu'on dispose de *n* séquences d'éléments (tuple, liste), toutes de longueur *l*. La fonction zip [29] permet de construire une liste de tuples qui est la juxtaposition de toutes ces séquences. Le ième tuple de la liste résultante contiendra tous les ième éléments des séquences juxtaposées. Si les longueurs des séquences sont différentes, la liste résultante aura même taille que la plus courte des séquences.

<<<

```
a = (1, 0, 7, 0, 0, 0)
b = [2, 2, 3, 5, 5, 5]
c = ["un", "deux", "trois", "quatre"]
d = zip(a, b, c)
print(d)
```

>>>

```
    <zip object at 0x7f032c687a88>
```

### Concaténation de chaîne de caractères

Il arrive fréquemment de constuire une chaîne de caractères petits bouts par petits bouts comme le montre le premier exemple ci-dessous. Cette construction peut s'avérer très lente lorsque le résultat est long. Dans ce cas, il est nettement plus rapide d'ajouter chaque morceau dans une liste puis de les concaténer en une seule fois grâce à la méthode join [30]

```
s = ""
while <condition> : s += ...
```

```
s = []
while <condition> : s.append ( ... )
s = "".join (s)
```

### Copie

A l'inverse des objets de type immuable, une affectation ne signifie pas une copie. Afin d'éviter certaines opérations superflues et parfois coûteuses en temps de traitement, on doit distinguer la variable de son contenu. Une variable désigne une liste avec un mot (ou identificateur), une affection permet de créer un second mot pour désigner la même liste. Il est alors équivalent de faire des opérations avec le premier mot ou le second comme le montre l'exemple suivant avec les listes `l` et `l2`.

<<<

```
l = [4, 5, 6]
l2 = l
print(l)            # affiche [4,5,6]
print(l2)           # affiche [4,5,6]
l2[1] = "modif"
print(l)            # affiche [4, 'modif', 6]
print(l2)           # affiche [4, 'modif', 6]
```

>>>

29. https://docs.python.org/3/library/functions.html#zip
30. https://docs.python.org/3/library/stdtypes.html#str.join

```
    [4, 5, 6]
    [4, 5, 6]
    [4, 'modif', 6]
    [4, 'modif', 6]
```

Dans cet exemple, il n'est pas utile de créer une seconde variable, dans le suivant, cela permet quelques raccourcis.

<<<

```
l = [[0, 1], [2, 3]]
l1 = l[0]
l1[0] = "modif"  # ligne équivalente à : l [0][0] = "modif"
print(l, l1)
```

>>>

```
    [['modif', 1], [2, 3]] ['modif', 1]
```

Par conséquent, lorsqu'on affecte une liste à une variable, celle-ci n'est pas recopiée, la liste reçoit seulement un nom de variable. L'affectation est en fait l'association d'un nom avec un objet (voir paragraphe *Copie d'instances* (page 123)). Pour copier une liste, il faut utiliser la fonction copy [31] du module copy [32]

<<<

```
import copy
l = [4, 5, 6]
l2 = copy.copy(l)
print(l)             # affiche [4,5,6]
print(l2)            # affiche [4,5,6]
l2[1] = "modif"
print(l)             # affiche [4,5,6]
print(l2)            # affiche [4, 'modif', 6]
```

>>>

```
    [4, 5, 6]
    [4, 5, 6]
    [4, 5, 6]
    [4, 'modif', 6]
```

Le module copy [33] est une extension interne. Cette syntaxe sera vue au chapitre *Modules* (page 235). Ce point sera rappelé au paragraphe *Listes et dictionnaires* (page 128). L'opérateur == permet de savoir si deux listes sont égales même si l'une est une copie de l'autre. Le mot-clé is [34] permet de vérifier si deux variables font référence à la même liste ou si l'une est une copie de l'autre comme le montre l'exemple suivant :

<<<

```
import copy
l = [1, 2, 3]
l2 = copy.copy(l)
l3 = l

print(l == l2)  # affiche True
print(l is l2)  # affiche False
print(l is l3)  # affiche True
```

31. https://docs.python.org/3/library/copy.html?highlight=copy#copy.copy
32. https://docs.python.org/3/library/copy.html
33. https://docs.python.org/3/library/copy.html
34. https://docs.python.org/3/library/stdtypes.html#comparisons

>>>

```
    True
    False
    True
```

**Fonction ‘‘copy‘‘ et ‘‘deepcopy‘‘**

Le comportement de la fonction copy [35] peut surprendre dans le cas où une liste contient d'autres listes. Pour être sûr que chaque élément d'une liste a été correctement recopiée, il faut utiliser la fonction deepcopy [36]. La fonction est plus longue mais elle recopie toutes les listes que ce soit une liste incluse dans une liste elle-même incluse dans une autre liste elle-même incluse...

<<<

```
import copy
l = [[1, 2, 3], [4, 5, 6]]
l2 = copy.copy(l)
l3 = copy.deepcopy(l)
l[0][0] = 1111
print(l)                # affiche [[1111, 2, 3], [4, 5, 6]]
print(l2)               # affiche [[1111, 2, 3], [4, 5, 6]]
print(l3)               # affiche [[1, 2, 3], [4, 5, 6]]
print(l is l2)          # affiche False
print(l[0] is l2[0])  # affiche True
print(l[0] is l3[0])  # affiche False
```

>>>

```
    [[1111, 2, 3], [4, 5, 6]]
    [[1111, 2, 3], [4, 5, 6]]
    [[1, 2, 3], [4, 5, 6]]
    False
    True
    False
```

La fonction deepcopy [37] est plus lente à exécuter car elle prend en compte les références récursives comme celles de l'exemple suivant où deux listes se contiennent l'une l'autre.

<<<

```
l = [1, "a"]
ll = [l, 3]    # ll contient l
l[0] = ll       # l contient ll
print(l)          # affiche [[[...], 3], 'a']
print(ll)         # affiche [[[...], 'a'], 3]

import copy
z = copy.deepcopy(l)
print(z)          # affiche [[[...], 3], 'a']
```

>>>

```
    [[[...], 3], 'a']
    [[[...], 'a'], 3]
    [[[...], 3], 'a']
```

35. https://docs.python.org/3/library/copy.html?highlight=copy#copy.copy
36. https://docs.python.org/3/library/copy.html?highlight=copy#copy.deepcopy
37. https://docs.python.org/3/library/copy.html?highlight=copy#copy.deepcopy

### Dictionnaire

Les dictionnaires sont des tableaux plus souples que les listes. Une liste référence les éléments en leur donnant une position : la liste associe à chaque élément une position entière comprise entre 0 et $n-1$ si $n$ est la longueur de la liste. Un dictionnaire permet d'associer à un élément autre chose qu'une position entière : ce peut être un entier, un réel, une chaîne de caractères, un tuple contenant des objets immuables. D'une manière générale, un dictionnaire associe à une valeur ce qu'on appelle une clé de type immuable. Cette clé permettra de retrouver la valeur associée.

L'avantage principal des dictionnaires est la recherche optimisée des clés. Par exemple, on recense les noms des employés d'une entreprise dans une liste. On souhaite ensuite savoir si une personne ayant un nom précisé à l'avance appartient à cette liste. Il faudra alors parcourir la liste jusqu'à trouver ce nom ou parcourir toute la liste si jamais celui-ci ne s'y trouve pas. Dans le cas d'un dictionnaire, cette recherche du nom sera beaucoup plus rapide à écrire et à exécuter.

### Définition et fonctions

---

**Définition D8 : dictionnaire**

Les dictionnaires sont des listes de couples. Chaque couple contient une clé et une valeur. Chaque valeur est indicée par sa clé. La valeur peut-être de tout type, la clé doit être de type immuable, ce ne peut donc être ni une liste, ni un dictionnaire. Chaque clé comme chaque valeur peut avoir un type différent des autres clés ou valeurs.

---

Un dictionnaire apparaît comme une succession de couples d'objets comprise entre accolades et séparés par des virgules. La clé et sa valeur sont séparées par le symbole `:`. Leur création reprend le même format :

```
x = { "cle1":"valeur1", "cle2":"valeur2" }
y = { }          # crée un dictionnaire vide
z = dict()       # crée aussi un dictionnaire vide
```

Les indices ne sont plus entiers mais des chaînes de caractères pour cet exemple. Pour associer la valeur à la clé ńăcle1ă", il suffit d'écrire :

<<<

```
x = {"cle1": "valeur1", "cle2": "valeur2"}
print(x["cle1"])
```

>>>

```
    valeur1
```

La plupart des fonctions disponibles pour les listes sont interdites pour les dictionnaires comme la concaténation ou l'opération de multiplication ($\star$). Il n'existe plus non plus d'indices entiers pour repérer les éléments, le seul repère est leur clé. La table suivante dresse la liste des opérations sur les dictionnaires.

| | |
|---|---|
| `x in d` | vrai si `x` est une des clés de `d` |
| `x not in d` | réciproque de la ligne précédente |
| `d[i]` | retourne l'élément associé à la clé `i` |
| `len(d)` | nombre d'éléments de `d` |
| `min(d)` | plus petite clé |
| `max(d)` | plus grande clé |
| `del d [i]` | supprime l'élément associé à la clé `i` |
| `list (d)` | retourne une liste contenant toutes les clés du dictionnaire `d` |
| `dict (x)` | convertit `x` en un dictionnaire si cela est possible, `d` est alors égal à `dict ( d.items () )` |
| `d.copy ()` | Retourne une copie de `d` |
| `d.items ()` | Retourne un itérateur sur tous les couples (clé, valeur) inclus dans le dictionnaire. |
| `d.keys ()` | Retourne un itérateur sur toutes les clés du dictionnaire `d` |
| `d.values ()` | Retourne un itérateur sur toutes les valeurs du dictionnaire `d` |
| `d.get (k[,x])` | Retourne `d[k]`, si la clé `k` est manquante, alors la valeur `None` est retournée à moins que le paramètre optionnel `x` soit renseigné, auquel cas, ce sera cette valeur qui sera retourné. |
| `d.clear ()` | Supprime tous les éléments du dictionnaire. |
| `d.update(d2)` | Le dictionnaire `d` reçoit le contenu de `d2`. |
| `d.setdefault(k[,x])` | Définit `d[k]` si la clé `k` existe, sinon, affecte `x` à `d[k]` |
| `d.pop()` | Retourne un élément et le supprime du dictionnaire. |

Contrairement à une liste, un dictionnaire ne peut être trié car sa structure interne est optimisée pour effectuer des recherches rapides parmi les éléments. On peut aussi se demander quel est l'intérêt de la méthode pop [38] qui retourne un élément puis le supprime alors qu'il existe le mot-clé `del`. Cette méthode est simplement plus rapide car elle choisit à chaque fois l'élément le moins coûteux à supprimer, surtout lorsque le dictionnaire est volumineux.

Les itérateurs sont des objets qui permettent de parcourir rapidement un dictionnaire, ils seront décrits en détail au chapitre *Classes* (page 95) sur les classes. Un exemple de leur utilisation est présenté dans le paragraphe suivant.

### Exemples

Il n'est pas possible de trier un dictionnaire. L'exemple suivant permet néanmoins d'afficher tous les éléments d'un dictionnaire selon un ordre croissant des clés. Ces exemples font appel aux paragraphes sur les boucles (voir chapitre *Syntaxe du langage Python (boucles, tests, fonctions)* (page 34)).

<<<

```
d = {"un": 1, "zéro": 0, "deux": 2, "trois": 3, "quatre": 4, "cinq": 5,
     "six": 6, "sept": 1, "huit": 8, "neuf": 9, "dix": 10}
key = list(d.keys())
key.sort()
for k in key:
    print(k, d[k])
```

>>>

```
    cinq 5
    deux 2
    dix 10
    huit 8
```

(suite sur la page suivante)

38. https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

(suite de la page précédente)

```
    neuf 9
    quatre 4
    sept 1
    six 6
    trois 3
    un 1
    zéro 0
```

L'exemple suivant montre un exemple d'utilisation des itérateurs. Il s'agit de construire un dictionnaire inversé pour lequel les valeurs seront les clés et réciproquement.

<<<

```
d = {"un": 1,   "zero": 0, "deux": 2, "trois": 3, "quatre": 4, "cinq": 5,
     "six": 6,  "sept": 1, "huit": 8, "neuf": 9,  "dix": 10}

dinv = {}                          # création d'un dictionnaire vide, on parcout
for key, value in d.items():        # les éléments du dictionnaire comme si
                                    # c'était une liste de 2-uple (clé,valeur)
    dinv[value] = key               # on retourne le dictionnaire

print(dinv)                         # affiche {0: 'zero', 1: 'un', 2: 'deux',
# 3: 'trois', 4: 'quatre', 5: 'cinq', 6: 'six',
# 8: 'huit', 9: 'neuf', 10: 'dix'}
```

>>>

```
    {1: 'sept', 0: 'zero', 2: 'deux', 3: 'trois', 4: 'quatre', 5: 'cinq', 6: 'six',␣
↪8: 'huit', 9: 'neuf', 10: 'dix'}
```

La méthode `items` retourne un itérateur. Cela permet de parcourir les éléments du dicionnaire sans créer de liste intermédiaire. Ceci explique ce qu'affiche le programme suivant :

<<<

```
d = {"un": 1,   "zero": 0, "deux": 2, "trois": 3, "quatre": 4, "cinq": 5,
     "six": 6,  "sept": 1, "huit": 8, "neuf": 9,  "dix": 10}
print(d.items())
print(list(d.items()))
```

>>>

```
    dict_items([('un', 1), ('zero', 0), ('deux', 2), ('trois', 3), ('quatre', 4), (
↪'cinq', 5), ('six', 6), ('sept', 1), ('huit', 8), ('neuf', 9), ('dix', 10)])
    [('un', 1), ('zero', 0), ('deux', 2), ('trois', 3), ('quatre', 4), ('cinq', 5), (
↪'six', 6), ('sept', 1), ('huit', 8), ('neuf', 9), ('dix', 10)]
```

D'une manière générale, il faut éviter d'ajouter ou de supprimer un élément dans une liste ou un dictionnaire qu'on est en train de parcourir au sein d'une boucle `for` ou `while`. Cela peut marcher mais cela peut aussi aboutir à des résultats imprévisibles surtout avec l'utilisation d'itérateurs (fonction `items`, `values`, `keys`). Il est préférable de terminer le parcours de la liste ou du dictionnaire puis de faire les modifications désirées une fois la boucle terminée. Dans le meilleur des cas, l'erreur suivante survient :

```
  File "essai.py", line 6, in <module>
    for k in d :
RuntimeError: dictionary changed size during iteration
```

### Copie

A l'instar des listes (voir paragraphe *Copie* (page 23)), les dictionnaires sont des objets et une affectation n'est pas équivalente à une copie comme le montre le programme suivant.

<<<

```
d = {4: 4, 5: 5, 6: 6}
d2 = d
print(d)              # affiche {4: 4, 5: 5, 6: 6}
print(d2)             # affiche {4: 4, 5: 5, 6: 6}
d2[5] = "modif"
print(d)              # affiche {4: 4, 5: 'modif', 6: 6}
print(d2)             # affiche {4: 4, 5: 'modif', 6: 6}
```

>>>

```
    {4: 4, 5: 5, 6: 6}
    {4: 4, 5: 5, 6: 6}
    {4: 4, 5: 'modif', 6: 6}
    {4: 4, 5: 'modif', 6: 6}
```

Lorsqu'on affecte un dictionnaire à une variable, celui-ci n'est pas recopié, le dictionnaire reçoit seulement un nom de variable. L'affectation est en fait l'association d'un nom avec un objet (voir paragraphe *Copie d'instances* (page 123)). Pour copier un dictionnaire, on peut utiliser la méthode copy [39].

<<<

```
d = {4: 4, 5: 5, 6: 6}
d2 = d.copy()
print(d)              # affiche {4: 4, 5: 5, 6: 6}
print(d2)             # affiche {4: 4, 5: 5, 6: 6}
d2[5] = "modif"
print(d)              # affiche {4: 4, 5: 5, 6: 6}
print(d2)             # affiche {4: 4, 5: 'modif', 6: 6}
```

>>>

```
    {4: 4, 5: 5, 6: 6}
    {4: 4, 5: 5, 6: 6}
    {4: 4, 5: 5, 6: 6}
    {4: 4, 5: 'modif', 6: 6}
```

Le mot-clé `is` a la même signification pour les dictionnaires que pour les listes, l'exemple du paragraphe *Copie* (page 23) est aussi valable pour les dictionnaires. Il en est de même pour la remarque concernant la fonction deepcopy [40]. Cette fonction recopie les listes et les dictionnaires.

### Clés de type modifiable

Ce paragraphe concerne davantage des utilisateurs avertis qui souhaitent malgré tout utiliser des clés de type modifiable. Dans l'exemple qui suit, la clé d'un dictionnaire est également un dictionnaire et cela provoque une erreur. Il en serait de même si la variable `k` utilisée comme clé était une liste.

```
k = { 1:1}
d = { }
d [k] = 0
```

39. https://docs.python.org/3/library/stdtypes.html?highlight=copy#dict.copy
40. https://docs.python.org/3/library/copy.html?highlight=copy#copy.deepcopy

La sortie :

```
Traceback (most recent call last):
  File "cledict.py", line 3, in <module>
    d [k] = 0
TypeError: dict objects are unhashable
```

Cela ne veut pas dire qu'il faille renoncer à utiliser un dictionnaire ou une liste comme clé. La fonction id [41] permet d'obtenir un entier qui identifie de manière unique tout objet. Le code suivant est parfaitement correct.

```
k = { 1:1}
d = { }
d [id (k)] = 0
```

Toutefois, ce n'est pas parce que deux dictionnaires auront des contenus identiques que leurs identifiants retournés par la fonction `id` seront égaux. C'est ce qui explique l'erreur que provoque la dernière ligne du programme suivant.

```
k = {1:1}
d = { }
d [id (k)] = 0
b = k
print(d[id(b)])  # affiche bien zéro
c = {1:1}
print(d[id(c)])  # provoque une erreur car même si k et c ont des contenus égaux,
                 # ils sont distincts, la clé id(c) n'existe pas dans d
```

Il existe un cas où on peut se passer de la fonction `id` mais il inclut la notion de classe définie au chapitre *Classes* (page 95). L'exemple suivant utilise directement l'instance `k` comme clé. En affichant le dictionnaire `d`, on vérifie que la clé est liée au résultat de l'instruction `id(k)` même si ce n'est pas la clé.

```
class A : pass

k = A ()
d = { }
d [k] = 0
print(d)                    # affiche {<__main__.A object at 0x0120DB90>: 0}
print(id(k), hex(id(k)))    # affiche 18930576, 0x120db90
print(d[id(k)])             # provoque une erreur
```

La fonction hex [42] convertit un entier en notation hexadécimale. Les nombres affichés changent à chaque exécution. Pour conclure, ce dernier exemple montre comment se passer de la fonction `id` dans le cas d'une clé de type dictionnaire.

<<<

```
class A (dict):
    def __hash__(self):
        return id(self)


k = A()
k["t"] = 4
d = {}
d[k] = 0
print(d)          # affiche {{'t': 4}: 0}
```

>>>

41. https://docs.python.org/3/library/functions.html?highlight=id#id
42. https://docs.python.org/3/library/functions.html?highlight=id#hex

```
    {{'t': 4}: 0}
```

#### Ensemble ou set

Le langage *python* définit également ce qu'on appelle un ensemble. Il est défini par les classes set [43] de type modifiable et la classe frozenset [44] de type immuable. Ils n'acceptent que des types identiques et offrent la plupart des opérations liées aux ensembles comme l'intersection, l'union.

<<<

```
print(set((1, 2, 3)) & set((2, 3, 5)))
# construit l'intersection qui est set([2, 3])
```

>>>

```
    {2, 3}
```

**A faire : Compléter le paragraphe sur les set**

set, frozen set

### 2.1.6 Extensions

#### Fonction `print`, `repr` et conversion en chaîne de caractères

La fonction `print` est déjà apparue dans les exemples présentés ci-dessus, elle permet d'afficher une ou plusieurs variables préalablement définies, séparées par des virgules. Les paragraphes qui suivent donnent quelques exemples d'utilisation. La fonction print [45] permet d'afficher n'importe quelle variable ou objet à l'écran, cet affichage suppose la conversion de cette variable ou objet en une chaîne de caractères. Deux fonctions permettent d'effectuer cette étape sans toutefois afficher le résultat à l'écran.

**Point à retenir :** la fonction `print` ne change pas le programme, elle affiche à l'écran le résultat d'une variable sans la modifier. Cela revient à écouter un programme avec un stéthoscope pour comprendre comment il fonctionne sans altérer son fonctionnement.

La fonction str [46] (voir paragraphe *Manipulation d'une chaîne de caractères* (page 9) permet de convertir toute variable en chaîne de caractères. Il existe cependant une autre fonction repr [47] qui effectue cette conversion. Dans ce cas, le résultat peut être interprété par la fonction eval [48] qui se charge de la conversion inverse. Pour les types simples comme ceux présentés dans ce chapitre, ces deux fonctions retournent des résultats identiques. Pour l'exemple, `x` désigne n'importe quelle variable.

```
x == eval (repr(x)) # est toujours vrai (True)
x == eval (str (x)) # n'est pas toujours vrai
```

43. https://docs.python.org/3/library/stdtypes.html#set
44. https://docs.python.org/3/library/stdtypes.html#frozenset
45. https://docs.python.org/3/library/functions.html?highlight=id#print
46. https://docs.python.org/3/library/functions.html?highlight=id#func-str
47. https://docs.python.org/3/library/functions.html?highlight=id#repr
48. https://docs.python.org/3/library/functions.html?highlight=id#eval

### Fonction `eval`

Comme le suggère le paragraphe précédent, la fonction eval [49] permet d'évaluer une chaîne de caractères ou plutôt de l'interpréter comme si c'était une instruction en *python*. Le petit exemple suivant permet de tester toutes les opérations de calcul possibles entre deux entiers.

<<<

```
x = 32
y = 9
op = "+ - * / % // & | and or << >>".split()
for o in op:
    s = str(x) + " " + o + "  " + str(y)
    print(s, " = ", eval(s))
```

>>>

```
    32 +  9  =  41
    32 -  9  =  23
    32 *  9  =  288
    32 /  9  =  3.5555555555555554
    32 %  9  =  5
    32 //  9  =  3
    32 &  9  =  0
    32 |  9  =  41
    32 and  9  =  9
    32 or  9  =  32
    32 <<  9  =  16384
    32 >>  9  =  0
```

Le programme va créer une chaîne de caractères pour chacune des opérations et celle-ci sera évaluée grâce à la fonction eval [50] comme si c'était une expression numérique. Il faut bien sûr que les variables que l'expression mentionne existent durant son évaluation.

### Informations fournies par *python*

Bien que les fonctions ne soient définies que plus tard (paragraphe *Fonctions* (page 53), il peut être intéressant de mentionner la fonction dir [51] qui retourne la liste de toutes les variables créées et accessibles à cet instant du programme. L'exemple suivant :

<<<

```
x = 3
print(dir())
```

>>>

```
    ['__WD__', 'x']
```

Certaines variables - des chaînes des caractères - existent déjà avant même la première instruction. Elles contiennent différentes informations concernant l'environnement dans lequel est exécuté le programme *python* :

49. https://docs.python.org/3/library/functions.html?highlight=id#eval
50. https://docs.python.org/3/library/functions.html?highlight=id#eval
51. https://docs.python.org/3/library/functions.html?highlight=id#dir

| | |
|---|---|
| `__builtins__` | Ce module contient tous les éléments présents dès le début d'un programme *python*, il contient entre autres les types présentés dans ce chapitre et des fonctions simples comme `range`. |
| `__doc__` | C'est une chaîne commentant le fichier, c'est une chaîne de caractères insérée aux premières lignes du fichiers et souvent entourée des symboles `"""` (voir chapitre *Modules* (page 235)). |
| `__file__` | Contient le nom du fichier où est écrit ce programme. |
| `__name__` | Contient le nom du module. |

La fonction dir [52] est également pratique pour afficher toutes les fonctions d'un module. L'instruction `dir(sys)` affiche la liste des fonctions du module sys [53] (voir chapitre *Modules* (page 235)).

La fonction `dir()` appelée sans argument donne la liste des fonctions et variables définies à cet endroit du programme. Ce résultat peut varier selon qu'on se trouver dans une fonction, une méthode de classe ou à l'extérieur du programme. L'instruction `dir([])` donne la liste des méthodes qui s'appliquent à une liste.

De la même manière, la fonction type [54] retourne une information concernant le type d'une variable.

<<<

```
x = 3
print(x, type(x))      # affiche 3 <type 'int'>
x = 3.5
print(x, type(x))      # affiche 3.5 <type 'float'>
```

>>>

```
    3 <class 'int'>
    3.5 <class 'float'>
```

### Affectations multiples

Il est possible d'effectuer en *python* plusieurs affectations simultanément.

```
x = 5        # affecte 5 à x
y = 6        # affecte 6 à y
x,y = 5,6    # affecte en une seule instruction 5 à x et 6 à y
```

Cette particularité reviendra lorsque les fonctions seront décrites puisqu'il est possible qu'une fonction retourne plusieurs résultats comme la fonction divmod [55] illustrée par le programme suivant.

<<<

```
x, y = divmod(17, 5)
print(x, y)                          # affiche 3 2
print("17 / 5 = 5 * ", x, " + ", y)  # affiche 17 / 5 = 5 *  3  +  2
```

>>>

```
    3 2
    17 / 5 = 5 *  3  +  2
```

Le langage *python* offre la possibilité d'effectuer plusieurs affectations sur la même ligne. Dans l'exemple qui suit, le couple $(5, 5)$ est affecté à la variable `point`, puis le couple `x, y` reçoit les deux valeurs du tuple `point`.

52. https://docs.python.org/3/library/functions.html?highlight=id#dir
53. https://docs.python.org/3/library/sys.html?highlight=sys#module-sys
54. https://docs.python.org/3/library/functions.html?highlight=id#type
55. https://docs.python.org/3/library/functions.html?highlight=divmod#divmod

```
x,y = point = 5,5
```

#### Hiérarchie des objets

La page modèle de données[56] décrit les différentes catégories d'objets du langage. Des objets de la même classe propose des fonctionnalités similaires.

#### Objets internes

Les objets objet internes[57] sont à peu près tout ce qui n'existe pas dans un langage compilé. Elles sont propres au langage et laisse transparaître des informations dont l'interpréteur a besoin pour comprendre le programme. Il est déconseillé de s'en servir si jamais on souhaite un jour traduire le même code dans un autre langage.

### 2.1.7 Commentaires accentués

Les commentaires commencent par le symbole # et se terminent par la fin de la ligne ; ils ne sont pas interprétés, ils n'ont aucune influence sur l'exécution du programme. Lorsque les commentaires incluent des symboles exclusivement français tels que les accents, le compilateur génère l'erreur suivante :

```
SyntaxError: Non-UTF-8 code starting with '\xe9' in file f.py on line 1,
but no encoding declared; see http://python.org/dev/peps/pep-0263/
for details
```

Il est néanmoins possible d'utiliser des accents dans les commentaires à condition d'insérer le commentaire suivant à la première ligne du programme. Il n'est pas nécessaire de retenir cette commande si le programme est écrit dans l'éditeur de texte fourni avec *python* car ce dernier propose automatiquement d'insérer cette ligne. Ce point est abordé au paragraphe par_intro_accent_code. Il faut inclure la placer le texte suivant en première ligne :

```
# -*- coding: utf-8 -*-
```

Ou :

```
# coding: cp1252
```

Ou encore :

```
# coding: latin-1
```

Le premier encoding `utf-8` est le plus communément utilisé dans le monde internet. Le second est utilisé par Windows.

## 2.2 Syntaxe du langage Python (boucles, tests, fonctions)


— *Les trois concepts des algorithmes* (page 35)
— *Tests* (page 38)
  — *Définition et syntaxe* (page 38)
  — *Comparaisons possibles* (page 39)
  — *Opérateurs logiques* (page 40)


56. https://docs.python.org/3/reference/datamodel.html#types
57. https://docs.python.org/3/library/stdtypes.html#internal-objects

- *Ecriture condensée* (page 40)
- *Exemple* (page 40)
- *None, True et 1* (page 42)
- *Passer, instruction `pass`* (page 42)

— *Boucles* (page 42)
- *Boucle while* (page 43)
- *Boucle for* (page 44)
- *Listes, boucle for, liste en extension* (page 45)
- *Itérateurs* (page 46)
- *Plusieurs variables de boucles* (page 47)
- *Ecriture condensée* (page 48)
- *Pilotage d'une boucle : continue* (page 49)
- *Pilotage d'une boucle : break* (page 51)
- *Fin normale d'une boucle : else* (page 51)
- *Suppression ou ajout d'éléments lors d'une boucle* (page 52)

— *Fonctions* (page 53)
- *Définition, syntaxe* (page 54)
- *Exemple* (page 54)
- *Paramètres avec des valeurs par défaut* (page 55)
- *Ordre des paramètres* (page 57)
- *Surcharge de fonction* (page 58)
- *Commentaires* (page 58)
- *Paramètres modifiables* (page 59)
- *Fonction récursive* (page 61)
- *Portée des variables, des paramètres* (page 62)
- *Portée des fonctions* (page 64)
- *Nombre de paramètres variable* (page 65)
- *Ecriture simplifiée pour des fonctions simples : lambda* (page 66)
- *Fonctions générateur* (page 68)
- *Identificateur appelable* (page 69)
- *Compilation dynamique (eval)* (page 69)
- *Compilation dynamique (compile, exec)* (page 70)

— *Indentation* (page 71)
— *Fonctions usuelles* (page 71)
— *Constructions classiques* (page 74)


Avec les variables, les boucles et les fonctions, on connaît suffisamment d'éléments pour écrire des programmes. Le plus difficile n'est pas forcément de les comprendre mais plutôt d'arriver à découper un algorithme complexe en utilisant ces briques élémentaires. C'est l'objectif des chapitres centrés autour des exercices. Toutefois, même si ce chapitre présente les composants élémentaires du langage, l'aisance qu'on peut acquérir en programmation vient à la fois de la connaissance du langage mais aussi de la connaissance d'algorithmes standards comme celui du tri ou d'une recherche dichotomique. C'est cette connaissance tout autant que la maîtrise d'un langage de programmation qui constitue l'expérience en programmation.

### 2.2.1 Les trois concepts des algorithmes

Les algorithmes sont composés d'instructions, ce sont des opérations élémentaires que le processeur exécute selon trois schémas :

| | |
|---|---|
| la séquence | enchaînement des instructions les unes à la suite des autres : passage d'une instruction à la suivante |
| le saut | passage d'une instruction à une autre qui n'est pas forcément la suivante (c'est une rupture de séquence) |
| le test | choix entre deux instructions |

Le saut n'apparaît plus de manière explicite dans les langages évolués car il est une source fréquente d'erreurs. Il intervient dorénavant de manière implicite au travers des boucles qui combinent un saut et un test. On écrit toujours ceci avec les langages les plus récents :

initialisation de la variable moy à 0
faire pour i allant de 1 à N
    moy reçoit moy + ni
moy reçoit moy / N

Et ceci est traduit par :

ligne 1 : initialisation de la variable moy à 0
ligne 2 : initialisation de la variable i à 1
ligne 3 : moy reçoit moy + ni
ligne 4 : i reçoit i + 1
ligne 5 : si i est inférieur ou égal à N alors aller à la ligne 3
ligne 6 : moy reçoit moy / N

Tout programme peut se résumer à ces trois concepts. Chaque langage les met en place avec sa propre syntaxe et parfois quelques nuances mais il est souvent facile de les reconnaître même dans des langages inconnus. Le calcul d'une somme décrit plus haut et écrit en *python* correspond à l'exemple suivant :

<<<

```
t = [0, 1, 2, 3, 4]
N = 5
moy = 0
for i in range(1, N+1):       # de 1 à N+1 exclu --> de 1 à N inclus
    moy += t[i-1]            # le premier indice est 0 et non 1
moy /= N
print(moy)
```

>>>

```
    2.0
```

Le premier élément de cette syntaxe est constituée de ses mots-clés `for` et `in` et des symboles =, +=, /=, `[`, `]`, `(`, `)`, `:`. La fonction iskeyword[58] permet de savoir si un mot-clé donné fait partie du langage *python*. Même si les modules seront décrits plus tard, la syntaxe suivante reste accessible :

<<<

```
import keyword
print(keyword.iskeyword("for"))      # affiche True
print(keyword.iskeyword("until"))    # affiche False
```

>>>

```
    True
    False
```

58. https://docs.python.org/3/library/keyword.html#keyword.iskeyword

Le programme suivant permet de récupérer la liste des mots-clés [59] du langage :

<<<

```
import keyword
print("\n".join(keyword.kwlist))
```

>>>

```
    False
    None
    True
    and
    as
    assert
    async
    await
    break
    class
    continue
    def
    del
    elif
    else
    except
    finally
    for
    from
    global
    if
    import
    in
    is
    lambda
    nonlocal
    not
    or
    pass
    raise
    return
    try
    while
    with
    yield
```

A cela s'ajoutent les symboles :

```
+   -   *   **   /   //   %
< > == <= >=   !=
<<   >>   &   |   \   ~   ^
=   +=   -=   *=   /=   //=   %=   **=
|=   &=   <<=   >>=   ~=   ^=
(   )   [   ]   {   }
"   """   '   '''
,   :   .
#
@   @=
```

59. https://docs.python.org/3/reference/lexical_analysis.html#keywords

Les espaces entre les mots-clés et les symboles ont peu d'importance, il peut n'y en avoir aucun comme dix. Les espaces servent à séparer un mot-clé, un nombre d'une variable. Les mots-clés et les symboles définissent la grammaire du langage *python*. Toutes ces règles sont décrites dans un langage un peu particulier par la page Full Grammar specification [60].

Les fonctions builtin [61] ne font pas partie de la grammaire du langage même si elles sont directement accessibles comme la fonction abs [62] qui retourne la valeur absolue d'un nombre. C'est un choix d'implémentation du programme qui interprète le langage mais absent de la grammaire.

### 2.2.2 Tests

#### Définition et syntaxe

**Définition D1 : test**

Les tests permettent d'exécuter des instructions différentes selon la valeur d'une condition logique.

Syntaxe :

**Syntaxe S1 : Tests**

```
if condition1 :
   instruction1
   instruction2
   ...
else :
   instruction3
   instruction4
   ...
```

La clause `else` est facultative. Lorsque la condition `condition1` est fausse et qu'il n'y a aucune instruction à exécuter dans ce cas, la clause `else` est inutile. La syntaxe du test devient :

```
if condition1 :
   instruction1
   instruction2
   ...
```

S'il est nécessaire d'enchaîner plusieurs tests d'affilée, il est possible de condenser l'écriture avec le mot-clé `elif` :

```
if condition1 :
   instruction1
   instruction2
   ...
elif condition2 :
   instruction3
   instruction4
   ...
elif condition3 :
   instruction5
   instruction6
```

(suite sur la page suivante)

60. https://docs.python.org/3/reference/grammar.html
61. https://docs.python.org/3/library/functions.html#built-in-functions
62. https://docs.python.org/3/library/functions.html#abs

(suite de la page précédente)

```
   ...
else :
   instruction7
   instruction8
   ...
```

Le décalage des instructions par rapport aux lignes contenant les mots-clés `if`, `elif`, `else` est très important : il fait partie de la syntaxe du langage et s'appelle l¿indentation [63]. Celle-ci permet de grouper les instructions ensemble. Le programme suivant est syntaxiquement correct même si le résultat n'est pas celui désiré.

<<<

```
x = 1
if x > 0:
    signe = 1
    print("le nombre est positif")
else:
    signe = -1
print("le nombre est négatif")  # ligne mal indentée (au sens de l'algorithme)
print("signe = ", signe)
```

>>>

```
    le nombre est positif
    le nombre est négatif
    signe =  1
```

Une ligne est mal indentée : `print("le nombre est négatif")`. Elle ne devrait être exécutée que si la condition `x>0` n'est pas vérifiée. Le fait qu'elle soit alignée avec les premières instructions du programme fait que son exécution n'a plus rien à voir avec cette condition. La programme répond de manière erronée.

Dans certains cas, l'interpréteur *python* ne sait pas à quel bloc attacher une instruction, c'est le cas de l'exemple suivant, la même ligne a été décalée de deux espaces, ce qui est différent de la ligne qui précède et de la ligne qui suit.

```
x = 1
if x > 0:
    signe = 1
    print("le nombre est positif")
else:
    signe = -1
  print("le nombre est négatif")     # ligne mal indentée (au sens de la grammaire)
print("signe = ", signe)
```

L'interpréteur retourne l'erreur suivante :

```
File "test.py", line 7
    print("le nombre est négatif")
                                  ^
IndentationError: unindent does not match any outer indentation level
```

### Comparaisons possibles

Les comparaisons possibles entre deux entités sont avant tout numériques mais ces opérateurs peuvent être définis pour tout type (voir *Classes* (page 95)), notamment sur les chaînes de caractères pour lesquelles les opérateurs de comparaison transcrivent l'ordre alphabétique.

63. https://fr.wikipedia.org/wiki/Style_d%27indentation

| | |
|---|---|
| <, > | inférieur, supérieur |
| <=, >= | inférieur ou égal, supérieur ou égal |
| ==, != | égal, différent |
| is, not is | x is y vérifie que x et y sont égaux, not is, différents, l'opérateur is est différent de l'opérateur ==, il est expliqué au paragraphe *Copie* (page 23) |
| in, not in | appartient, n'appartient pas |

### Opérateurs logiques

Il existe trois opérateurs logiques qui combinent entre eux les conditions.

| | |
|---|---|
| not | négation |
| and | et logique |
| or | ou logique |

La priorité des opérations numériques est identique à celle rencontrée en mathématiques. L'opérateur puissance vient en premier, la multiplication/division ensuite puis l'addition/soustraction. Ces opérations sont prioritaires sur les opérateurs de comparaisons (>, <, ==, ...) qui sont eux-mêmes sur les opérateurs logiques not, and, or. Il est tout de même conseillé d'ajouter des parenthèses en cas de doute. C'est ce qu décrit la page Operator precedence [64].

### Ecriture condensée

Il existe deux écritures condensées de tests. La première consiste à écrire un test et l'unique instruction qui en dépend sur une seule ligne.

**if condition :** instruction1

**else :** instruction2

Ce code peut tenir en deux lignes :

```
if condition : instruction1
else : instruction2
```

Le second cas d'écriture condensée concerne les comparaisons enchaînées. Le test `if 3 < x and x < 5 : instruction` peut être condensé par `if 3 < x < 5 : instruction`. Il est ainsi possible de juxtaposer autant de comparaisons que nécessaire : `if 3 < x < y < 5 : instruction`.

Le mot-clé `in` permet également de condenser certains tests lorsque la variable à tester est entière. `if x == 1 or x == 6 or x == 50 :` peut être résumé simplement par `if x in (1,6,50) :` ou `if x in {1,6,50}:` pour les grandes listes.

### Exemple

L'exemple suivant associe à la variable `signe` le signe de la variable `x`.

<<<

```
x = -5
if x < 0:
    signe = -1
elif x == 0:
    signe = 0
else:
```

(suite sur la page suivante)

64. https://docs.python.org/3/reference/expressions.html#operator-precedence

(suite de la page précédente)

```
    signe = 1
print(signe)
```

>>>

```
    -1
```

Son écriture condensée lorsqu'il n'y a qu'une instruction à exécuter :

<<<

```
x = -5
if x < 0:
    signe = -1
elif x == 0:
    signe = 0
else:
    signe = 1
print(signe)
```

>>>

```
    -1
```

Le programme suivant saisit une ligne au clavier et dit si c'est ńăouiă" ou ńănonă" qui a été saisi. La fonction input [65] retourne ce qui vient de l'utilisateur :

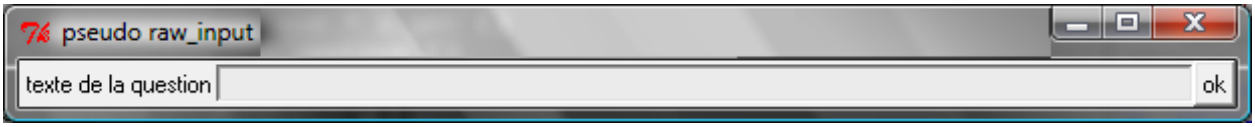
```
s = input ("dites oui : ")     # voir remarque suivante
if s == "oui" or s [0:1] == "o" or s [0:1] == "O" or s == "1" :
    print "oui"
else:
    print "non"
```

La fonction input [66] invite l'utilisateur d'un programme à saisir une réponse lors de l'exécution du programme. Tant que la touche entrée n'a pas été pressée, l'exécution du programme ne peut continuer. Cette fonction est en réalité peu utilisée. Les interfaces graphiques sont faciles d'accès en *python*, on préfère donc saisir une réponse via une fenêtre plutôt qu'en ligne de commande. L'exemple suivant montre comment remplacer cette fonction à l'aide d'une fenêtre graphique.

```
import tkinter
def question(legende):
    reponse = [""]
    root = tkinter.Tk ()
    root.title("pseudo input")
    tkinter.Label(text=legende).pack(side=tkinter.LEFT)
    s = tkinter.Entry(text="def", width=80)
    s.pack(side=tkinter.LEFT)
    def rget():
        reponse[0] = s.get ()
        root.destroy()
    tkinter.Button(text="ok", command=rget).pack(side=tkinter.LEFT)
    root.mainloop()
    return(reponse[0])

print("réponse ", question("texte de la question"))
```

65. https://docs.python.org/3/library/functions.html#input
66. https://docs.python.org/3/library/functions.html#input

On peut améliorer la fonction `question` en précisant une valeur par défaut par exemple (voir chap_interface à ce sujet). Le programme affiche la fenêtre suivante :




### None, True et 1

L'écriture de certains tests peut encore être réduite lorsqu'on cherche à comparer une variable entière, booléenne ou `None` comme le précise la table suivant :

| type | test | test équivalent |
|---|---|---|
| bool | `if v:` | `if v == True :` |
| `bool` | `if not v:` | `if v == False :` |
| `int` | `if v:` | `if v != 0:` |
| `int` | `if not v :` | `if v == 0 :` |
| `float` | `if v :` | `if v != 0.0 :` |
| `float` | `if not v :` | `if v == 0.0 :` |
| `list, dict, set` | `if v:` | `if v is not None and len(v) > 0:` |
| `list, dict, set` | `if not v:` | `if v is None or len(v) == 0:` |
| `object` | `if v :` | `if v is not None:` |
| `object` | `if not v:` | `if v is None :` |

### Passer, instruction `pass`

Dans certains cas, aucune instruction ne doit être exécutée même si un test est validé. En *python*, le corps d'un test ne peut être vide, il faut utiliser l'instruction `pass`. Lorsque celle-ci est manquante, *python* affiche un message d'erreur.

**Syntaxe S2 : Instruction pass**

```
signe = 0
x = 0
if x < 0: signe = -1
elif x == 0:
   pass           # signe est déjà égal à 0
else :
    signe = 1
```

Dans ce cas précis, si l'instruction `pass` est oubliée, l'interpréteur *python* génère l'erreur suivante :

```
File "nopass.py", line 6
    else :
    ^
IndentationError: expected an indented block
```

## 2.2.3 Boucles

Boucle B1 : test

Les boucles permettent de répéter une séquence d'instructions tant qu'une certaine condition est vérifiée.

Le langage *python* propose deux types de boucles. La boucle `while` suit scrupuleusement la définition précédent. La boucle `for` est une boucle `while` déguisée (voir *Boucle for* (page 44)), elle propose une écriture simplifiée pour répéter la même séquence d'instructions pour tous les éléments d'un ensemble.

### Boucle while

L'implémentation d'une boucle de type `while` suit le schéma d'écriture suivant :

**Syntaxe S3 : Boucle while**

```
while cond :
    instruction 1
    ...
    instruction n
```

Où `cond` est une condition qui détermine la poursuite de la répétition des instructions incluses dans la boucle. Tant que celle-ci est vraie, les instructions 1 à *n* sont exécutées.

Tout comme les tests, l'indentation joue un rôle important. Le décalage des lignes d'un cran vers la droite par rapport à l'instruction `while` permet de les inclure dans la boucle comme le montre l'exemple suivant.

<<<

```
n = 0
while n < 3:
    print("à l'intérieur ", n)
    n += 1
print("à l'extérieur ", n)
```

>>>

```
    à l'intérieur  0
    à l'intérieur  1
    à l'intérieur  2
    à l'extérieur  3
```

Les conditions suivent la même syntaxe que celles définies lors des tests (voir *Comparaisons possibles* (page 39)). A moins d'inclure l'instruction *break* (page 51) qui permet de sortir prématurément d'une boucle, la condition qui régit cette boucle doit nécessairement être modifiée à l'intérieur de celle-ci. Dans le cas contraire, on appelle une telle boucle une boucle infinie[67] puisqu'il est impossible d'en sortir.

L'exemple suivant contient une boucle infinie car le symbole = est manquant dans la dernière instruction. La variable `n` n'est jamais modifiée et la condition `n<3` toujours vraie.

```
n = 0
while n < 3 :
   print(n)
   n + 1         # n n'est jamais modifié, l'instruction correcte serait n += 1
```

67. https://fr.wikipedia.org/wiki/Boucle_infinie

### Boucle for

L'implémentation d'une boucle de type `for` suit le schéma d'écriture suivant :

**Syntaxe S4 : Boucle for**

```
for x in ensemble:
    instruction 1
    ...
    instruction n
```

Où `x` est un élément de l'ensemble `ensemble`. Les instructions 1 à *n* sont exécutées pour chaque élément `x` de l'ensemble `ensemble`. Cet ensemble peut être une chaîne de caractères, un tuple, une liste, un dictionnaire, un set ou tout autre type incluant des itérateurs qui sont présentés au chapitre *Classes* (page 95).

Tout comme les tests, l'indentation est importante. L'exemple suivant affiche tous les éléments d'un tuple à l'aide d'une boucle `for`.

<<<

```
t = (1, 2, 3, 4)
for x in t:        # affiche les nombres 1,2,3,4
    print(x)       # chacun sur une ligne différente
```

>>>

```
    1
    2
    3
    4
```

Lors de l'affichage d'un dictionnaire, les éléments n'apparaissent pas triés ni dans l'ordre dans lequel ils y ont été insérés. L'exemple suivant montre comment afficher les clés et valeurs d'un dictionnaire dans l'ordre croissant des clés.

<<<

```
d = {1: 2, 3: 4, 5: 6, 7: -1, 8: -2}
# affiche le dictionnaire {8: -2, 1: 2, 3: 4, 5: 6, 7: -1}
print(d)
k = list(d.keys())
print(k)                # affiche les clés [8, 1, 3, 5, 7]
k.sort()
print(k)                # affiche les clés triées [1, 3, 5, 7, 8]
for x in k:             # affiche les éléments du dictionnaire
    print(x, ":", d[x])  # triés par clés croissantes
```

>>>

```
    {1: 2, 3: 4, 5: 6, 7: -1, 8: -2}
    [1, 3, 5, 7, 8]
    [1, 3, 5, 7, 8]
    1 : 2
    3 : 4
    5 : 6
    7 : -1
    8 : -2
```

Le langage *python* propose néanmoins la fonction `sorted` qui réduit l'exemple suivant en trois lignes :

<<<

```
d = {1: 2, 3: 4, 5: 6, 7: -1, 8: -2}
for x in sorted(d):     # pour les clés dans l'ordre croissant
    print(x, ":", d[x])
```

>>>

```
    1 : 2
    3 : 4
    5 : 6
    7 : -1
    8 : -2
```

La boucle la plus répandue est celle qui parcourt des indices entiers compris entre *0* et *n-1*. On utilise pour cela la boucle `for` et la fonction range [68] comme dans l'exemple qui suit.

<<<

```
sum = 0
N = 10
for n in range(0, N):      # va de 0 à N exclu
    sum += n                # additionne tous les entiers compris entre 0 et N-1
```

>>>

```

```

Ou encore pour une liste quelconque :

<<<

```
li = [4, 5, 3, -6, 7, 9]
sum = 0
for n in range(0, len(li)):  # va de 0 à len(li) exclu
    sum += li[n]              # additionne tous les éléments de li
```

>>>

```

```

### Listes, boucle for, liste en extension

Le paragraphe *Boucles et listes* (page 22) a montré comment le mot-clé `for` peut être utilisé pour simplifier la création d'une liste à partir d'une autre. La syntaxe d'une liste en extension [69] suit le schéma suivant :

**Syntaxe S5 : Liste en extension**

```
[ expression for x in ensemble ]
```

Où `expression` est une expression numérique incluant ou non `x`, la variable de la boucle, `ensemble` est un ensemble d'éléments, tuple, liste, dictionnaire, set ou tout autre chose qui peut être parcouru. Cette syntaxe permet de résumer en une ligne la création de la séquence `y` du programme suivant.

<<<

68. https://docs.python.org/3/library/functions.html#func-range
69. http://sametmax.com/python-love-les-listes-en-intention-partie/

```
y = list()
for i in range(0, 5):
    y.append(i+1)
print(y)                                # affiche [1,2,3,4,5]

y = [i+1 for i in range(0, 5)]      # résume trois lignes du programme précédent
print(y)                                # affiche [1,2,3,4,5]
```

>>>

```
    [1, 2, 3, 4, 5]
    [1, 2, 3, 4, 5]
```

Un autre exemple de cette syntaxe réduite a été présenté au paragraphe *Boucles et listes* (page 22). Cette écriture condensée est bien souvent plus lisible même si tout dépend des préférences de celui qui programme. Elle peut être étendue au dictionnaire.

<<<

```
y = {i: i+1 for i in range(0, 5)}
print(y)
```

>>>

```
    {0: 1, 1: 2, 2: 3, 3: 4, 4: 5}
```

### Itérateurs

Toute boucle `for` peut s'appliquer sur un objet muni d'un itérateur tels que les chaînes de caractères, tuples, les listes, les dictionnaires, les ensembles.

<<<

```
d = ["un", "deux", "trois"]
for x in d:
    print(x)            # affichage de tous les éléments de d
```

>>>

```
    un
    deux
    trois
```

Cette syntaxe réduite a déjà été introduite pour les listes et les dictionnaires au chapitre précédent. Il existe une version équivalente avec la boucle `while` utilisant de façon explicite les itérateurs. Il peut être utile de lire le chapitre suivant sur les classes et le chapitre *Exceptions* (page 155) sur les exceptions avant de revenir sur la suite de cette section qui n'est de toutes façons pas essentielle.

L'exemple précédent est convertible en une boucle `while` en faisant apparaître explicitement les itérateurs (voir *Itérateurs* (page 113)). Un itérateur est un objet qui permet de parcourir aisément un ensemble. La fonction `it = iter(e)` permet d'obtenir un itérateur `it` sur l'ensemble `e`. L'appel à l'instruction `it.next()` parcourt du premier élément jusqu'au dernier en retournant la valeur de chacun d'entre eux. Lorsqu'il n'existe plus d'élément, l'exception `StopIteration` est déclenchée (voir *Exceptions* (page 155)). Il suffit de l'intercepter pour mettre fin au parcours.

<<<

```
d = ["un", "deux", "trois"]
it = iter(d)                        # obtient un itérateur sur d
while True:
    try:
        # obtient l'élément suivant, s'il n'existe pas
        x = next(it)
    except StopIteration:
        break                       # déclenche une exception
    print(x)                        # affichage de tous les éléments de d
```

>>>

```
    un
    deux
    trois
```

### Plusieurs variables de boucles

Jusqu'à présent, la boucle `for` n'a été utilisée qu'avec une seule variable de boucle, comme dans l'exemple suivant où on parcourt une liste de tuple pour les afficher.

<<<

```
d = [(1, 0, 0), (0, 1, 0), (0, 0, 1)]
for v in d:
    print(v)
```

>>>

```
    (1, 0, 0)
    (0, 1, 0)
    (0, 0, 1)
```

Lorsque les éléments d'un ensemble sont des tuples, des listes, des dictionnaires ou des ensembles composés de taille fixe, il est possible d'utiliser une notation qui rappelle les affectations multiples (voir *Affectations multiples* (page 33)). L'exemple précédent devient dans ce cas :

<<<

```
d = [(1, 0, 0), (0, 1, 0), (0, 0, 1)]
for x, y, z in d:
    print(x, y, z)
```

>>>

```
    1 0 0
    0 1 0
    0 0 1
```

Cette écriture n'est valable que parce que chaque élément de la liste `d` est un tuple composé de trois nombres. Lorsqu'un des éléments est de taille différente à celle des autres, comme dans l'exemple suivant, une erreur survient.

<<<

```
d = [(1, 0, 0), (0, 1, 0, 6), (0, 0, 1)]  # un élément de taille quatre
for x, y, z in d:
    print(x, y, z)
```

>>>

```
    1 0 0
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 6, in <module>
      File "", line 4, in run_python_script_139651613584696
    ValueError: too many values to unpack (expected 3)
```

Cette syntaxe est très pratique associée à la fonction `zip` (voir *Collage de séquences, fonction zip* (page 23)). Il est alors possible de parcourir plusieurs séquences (tuple, liste, dictionnaire, ensemble) simultanément.

<<<

```
a = range(0, 5)
b = [x**2 for x in a]
for x, y in zip(a, b):
    print(y), " est le carré de ", x
    # affichage à droite
```

>>>

```
    0
    1
    4
    9
    16
```

### Ecriture condensée

Comme pour les tests, lorsque les boucles ne contiennent qu'une seule instruction, il est possible de l'écrire sur la même ligne que celle de la déclaration de la boucle `for` ou `while`.

<<<

```
d = ["un", "deux", "trois"]
for x in d:
    print(x)          # une seule instruction
```

>>>

```
    un
    deux
    trois
```

Il existe peu de cas où la boucle `while` s'écrit sur une ligne car elle inclut nécessairement une instruction permettant de modifier la condition d'arrêt.

<<<

```
d = ["un", "deux", "trois"]
i = 0
while d[i] != "trois":
    i += 1
print("trois a pour position ", i)
```

>>>

```
    trois a pour position  2
```

### Pilotage d'une boucle : continue

Pour certains éléments d'une boucle, lorsqu'il n'est pas nécessaire d'exécuter toutes les instructions, il est possible de passer directement à l'élément suivant ou l'itération suivante. Le programme suivant utilise le crible d'Eratosthène [70] pour dénicher tous les nombres premiers compris entre 1 et 99.

*Aparté sur le crible d'Eratosthène*

Le crible d'Eratosthène est un algorithme permettant de déterminer les nombres premiers. Pour un nombre premier $p$, il paraît plus simple de considérer tous les entiers de $p-1$ à *1* pour savoir si l'un d'eux divise $p$. C'est ce qu'on fait lorsqu'on doit vérifier le caractère premier d'un seul nombre. Pour plusieurs nombres à la fois, le crible d'Eratosthène est plus efficace : au lieu de s'intéresser aux diviseurs, on s'intéresse aux multiples d'un nombre. Pour un nombre $i$, on sait que $2i$, $3i$, ... ne sont pas premiers. On les raye de la liste. On continue avec $i+1$, $2(i+1)$, $3(i+1)$...

<<<

```
d = dict()
for i in range(1, 100):             # d [i] est vrai si i est un nombre premier
    d[i] = True                    # au début, comme on ne sait pas, on suppose
    # que tous les nombres sont premiers
for i in range(2, 100):
                                    # si d [i] est faux,
    if not d[i]:
        continue                    # les multiples de i ont déjà été cochés
        # et peut passer à l'entier suivant
    for j in range(2, 100):
        if i*j < 100:
            d[i*j] = False         # d [i*j] est faux pour tous les multiples de i
            # inférieurs à 100
print("liste des nombres premiers")
for i in d:
    if d[i]:
        print(i)
```

>>>

```
    liste des nombres premiers
    1
    2
    3
    5
    7
    11
    13
    17
    19
    23
    29
    31
    37
    41
    43
    47
    53
```

(suite sur la page suivante)

70. https://fr.wikipedia.org/wiki/Crible_d'%C3%89ratosth%C3%A8ne

(suite de la page précédente)

```
    59
    61
    67
    71
    73
    79
    83
    89
    97
```

Ce programme est équivalent au suivant :

<<<

```
d = dict()
for i in range(1, 100):
    d[i] = True

for i in range(2, 100):
    if d[i]:
        for j in range(2, 100):
            if i*j < 100:
                d[i*j] = False

print("liste des nombres premiers")
for i in d:
    if d[i]:
        print(i)
```

>>>

```
    liste des nombres premiers
    1
    2
    3
    5
    7
    11
    13
    17
    19
    23
    29
    31
    37
    41
    43
    47
    53
    59
    61
    67
    71
    73
    79
    83
    89
    97
```

Le mot-clé `continue` évite de trop nombreuses indentations et rend les programmes plus lisibles.

### Pilotage d'une boucle : break

Lors de l'écriture d'une boucle `while`, il n'est pas toujours adéquat de résumer en une seule condition toutes les raisons pour lesquelles il est nécessaire d'arrêter l'exécution de cette boucle. De même, pour une boucle `for`, il n'est pas toujours utile de visiter tous les éléments de l'ensemble à parcourir. C'est le cas par exemple lorsqu'on recherche un élément, une fois qu'il a été trouvé, il n'est pas nécessaire d'aller plus loin. L'instruction `break` permet de quitter l'exécution d'une boucle.

<<<

```
l = [6, 7, 5, 4, 3]
n = 0
c = 5
for x in l:
    if x == c:
        break   # l'élément a été trouvé, on sort de la boucle
    n += 1              # si l'élément a été trouvé, cette instruction
    # n'est pas exécutée
print("l'élément ", c, "est en position ", n)
```

>>>

```
    l'élément  5 est en position  2
```

Si deux boucles sont imbriquées, l'instruction `break` ne sort que de la boucle dans laquelle elle est insérée. L'exemple suivant vérifie si un entier est la somme des carrés de deux entiers compris entre 1 et 20.

<<<

```
ens = range(1, 21)
n = 53
for x in ens:
    for y in ens:
        c = x*x + y*y
        if c == n:
            break
    if c == n:
        break   # cette seconde instruction break est nécessaire
        # pour sortir de la seconde boucle
        # lorsque la solution a été trouvée
if c == n:
    # le symbole \ permet de passer à la ligne sans changer d'instruction
    print(n, " est la somme des carrés de deux entiers :",
          x, "*", x, "+", y, "*", y, "=", n)
else:
    print(n, " n'est pas la somme des carrés de deux entiers")
```

>>>

```
    53  est la somme des carrés de deux entiers : 2 * 2 + 7 * 7 = 53
```

### Fin normale d'une boucle : else

Le mot-clé `else` existe aussi pour les boucles et s'utilise en association avec le mot-clé `break`. L'instruction `else` est placée à la fin d'une boucle, indentée au même niveau que `for` ou `while`. Les lignes qui suivent le mot-clé `else` ne sont exécutées que si aucune instruction `break` n'a été rencontrée dans le corps de la boucle. On reprend l'exemple

du paragraphe précédent. On recherche cette fois-ci la valeur 1 qui ne se trouve pas dans la liste `L`. Les lignes suivant le test `if x == c` ne seront jamais exécutées au contraire de la dernière.

<<<

```
L = [6, 7, 5, 4, 3]
n = 0
c = 1
for x in L:
    if x == c:
        print("l'élément ", c, " est en position ", n)
        break
    n += 1
else:
    print("aucun élément ", c, " trouvé")  # affiche aucun élément  1  trouvé
```

>>>

```
    aucun élément  1  trouvé
```

Les lignes dépendant de la clause `else` seront exécutées dans tous les cas où l'exécution de la boucle n'est pas interrompue par une instruction `break` ou une instruction `return`.

### Suppression ou ajout d'éléments lors d'une boucle

En parcourant la liste en se servant des indices, il est possible de supprimer une partie de cette liste. Il faut néanmoins faire attention à ce que le code ne produise pas d'erreur comme c'est le cas pour le suivant. La boucle `for` parcourt la liste `list(range(0, len(li)))` qui n'est pas modifiée en même temps que l'instruction `del li[i:i+2]`.

<<<

```
li = list(range(0, 10))
print(li)                 # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range(0, len(li)):
    if i == 5:
        del li[i:i+2]
    print(li[i])         # affiche successivement 0, 1, 2, 3, 4, 7, 8, 9 et
    # produit une erreur
print(li)
```

>>>

```
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    0
    1
    2
    3
    4
    7
    8
    9
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 11, in <module>
      File "", line 8, in run_python_script_139651613585272
    IndexError: list index out of range
```

Le programme suivant marche parfaitement puisque cette fois-ci la boucle parcourt la liste `li`. En revanche, pour la suppression d'une partie de celle-ci, il est nécessaire de conserver en mémoire l'indice de l'élément visité. C'est le rôle de la variable `i`.

<<<

```
li = list(range(0, 10))
print(li)                # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
i = 0
for t in li:
    if i == 5:
        del li[i:i+2]
    i = i+1
    print(t)             # affiche successivement 0, 1, 2, 3, 4, 5, 8, 9
print(li)                # affiche [0, 1, 2, 3, 4, 7, 8, 9]
```

>>>

```
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    0
    1
    2
    3
    4
    5
    8
    9
    [0, 1, 2, 3, 4, 7, 8, 9]
```

Le langage *python* offre la possibilité de supprimer des éléments d'une liste alors même qu'on est en train de la parcourir. Le programme qui suit ne marche pas puisque l'instruction `del i` ne supprime pas un élément de la liste mais l'identificateur `i` qui prendra une nouvelle valeur lors du passage suivant dans la boucle.

<<<

```
li = list(range(0, 10))
print(li)                # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in li:
    if i == 5:
        del i
print(li)                # affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

>>>

```
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

On pourrait construire des exemples similaires dans le cadre de l'ajout d'un élément à la liste. Il est en règle générale déconseillé de modifier une liste, un dictionnaire pendant qu'on le parcourt. Malgré tout, si cela s'avérait indispensable, il convient de faire plus attention dans ce genre de situations.

### 2.2.4 Fonctions

Les fonctions sont des petits programmes qui effectuent des tâches plus précises que le programme entier. On peut effectivement écrire un programme sans fonction mais ils sont en général illisibles. Utiliser des fonctions implique de découper un algorithme en tâches élémentaires. Le programme final est ainsi plus facile à comprendre. Un autre avantage est de pouvoir plus facilement isoler une erreur s'il s'en produit une : il suffit de tester une à une les fonctions pour déterminer laquelle retourne un mauvais résultat. L'avantage le plus important intervient lorsqu'on doit effectuer

la même chose à deux endroits différentes d'un programme : une seule fonction suffit et elle sera appelée à ces deux endroitsfootnote{Pour les utilisateurs experts : en langage *python*, les fonctions sont également des variables, elles ont un identificateur et une valeur qui est dans ce cas un morceau de code. Cette précision explique certaines syntaxes du chapitre chap_interface sur les interfaces graphiques ou celle introduite en fin de chapitre au paragraphe paragraphe_fonction_variable.

### Définition, syntaxe

**Définition D2 : fonction**

Une fonction est une partie d'un programme - ou sous-programme - qui fonctionne indépendamment du reste du programme. Elle reçoit une liste de paramètres et retourne un résultat. Le corps de la fonction désigne toute instruction du programme qui est exécutée si la fonction est appelée.

Lorsqu'on écrit ses premiers programme, on écrit souvent des fonctions plutôt longues avant de s'apercevoir que certains parties sont identiques ailleurs. On extrait donc la partie répétée pour en faire une fonction. Avec l'habitude, on finit par écrire des fonctions plus petites et réutilisables.

**Syntaxe S6 : Déclaration d'une fonction**

```
def fonction_nom (par_1, ..., par_n) :
    instruction_1
    ...
    instruction_n
    return res_1, ..., res_n
```

`fonction_nom` est le nom de la fonction, il suit les mêmes règles que le nom des variables. `par_1` à `par_n` sont les noms des paramètres et `res_1` à `res_n` sont les résultats retournés par la fonction. Les instructions associées à une fonction doivent être indentées par rapport au mot-clé `def`.

S'il n'y a aucun résultat, l'instruction `return` est facultative ou peut être utilisée seule sans être suivie par une valeur ou une variable. Cette instruction peut apparaître plusieurs fois dans le code de la fonction mais une seule d'entre elles sera exécutée. A partir de ce moment, toute autre instruction de la fonction sera ignorée. Pour exécuter une fonction ainsi définie, il suffit de suivre la syntaxe suivante :

**Syntaxe S7 : Appel d'une fonction**

```
x_1, ..., x_n = fonction_nom (valeur_1, valeur_2, ..., valeur_n)
```

Où `fonction_nom` est le nom de la fonction, `valeur_1` à `valeur_n` sont les noms des paramètres, `x_1` à `x_n` reçoivent les résultats retournés par la fonction. Cette affectation est facultative. Si on ne souhaite pas conserver les résultats, on peut donc appeler la fonction comme suit :

```
fonction_nom (valeur_1, valeur_2, ..., valeur_n)
```

Lorsqu'on commence à programmer, il arrive parfois qu'on confonde le rôle des mots-clés `print` et `return`. Il faut se souvenir que l'instruction `print` n'a pas d'impact sur le déroulement du programme. Elle sert juste à visualiser le contenu d'une variable. Sans l'instruction `return`, toute fonction retourne `None`.

### Exemple

Le programme suivant utilise deux fonctions. La première convertit des coordonnées cartésiennes en coordonnées polaires. Elle prend deux réels en paramètres et retourne deux autres réels. La seconde fonction affiche les résultats de

la première pour tout couple de valeurs $(x, y)$. Elle ne retourne aucun résultat.

<<<

```
import math


def coordonnees_polaires(x, y):
    rho = math.sqrt(x*x+y*y)    # calcul la racine carrée de x*x+y*y
    theta = math.atan2(y, x)      # calcule l'arc tangente de y/x en tenant
    # compte des signes de x et y
    return rho, theta


def affichage(x, y):
    r, t = coordonnees_polaires(x, y)
    print("cartésien (%f,%f) --> polaire (%f,%f degrés)"
          % (x, y, r, math.degrees(t)))


affichage(1, 1)
affichage(0.5, 1)
affichage(-0.5, 1)
affichage(-0.5, -1)
affichage(0.5, -1)
```

>>>

```
    cartésien (1.000000,1.000000) --> polaire (1.414214,45.000000 degrés)
    cartésien (0.500000,1.000000) --> polaire (1.118034,63.434949 degrés)
    cartésien (-0.500000,1.000000) --> polaire (1.118034,116.565051 degrés)
    cartésien (-0.500000,-1.000000) --> polaire (1.118034,-116.565051 degrés)
    cartésien (0.500000,-1.000000) --> polaire (1.118034,-63.434949 degrés)
```

### Paramètres avec des valeurs par défaut

Lorsqu'une fonction est souvent appelée avec les mêmes valeurs pour ses paramètres, il est possible de spécifier pour ceux-ci une valeur par défaut.

**Syntaxe S8 : Valeurs par défaut**

```
def fonction_nom (param_1, param_2 = valeur_2, ..., param_n = valeur_n):
    ...
```

Où `fonction_nom` est le nom de la fonction. `param_1` à `param_n` sont les noms des paramètres, `valeur_2` à `valeur_n` sont les valeurs par défaut des paramètres `param_2` à `param_n`. La seule contrainte lors de cette définition est que si une valeur par défaut est spécifiée pour un paramètre, alors tous ceux qui suivent devront eux aussi avoir une valeur par défaut.

Exemple : %

<<<

```
def commander_carte_orange(nom, prenom, paiement="carte", nombre=1, zone=2):
    print("nom : ", nom)
    print("prénom : ", prenom)
    print("paiement : ", paiement)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    print("nombre : ", nombre)
    print("zone :", zone)


commander_carte_orange("Dupré", "Xavier", "chèque")
# les autres paramètres nombre et zone auront pour valeur
# leurs valeurs par défaut
```

>>>

```
    nom :  Dupré
    prénom :  Xavier
    paiement :  chèque
    nombre :  1
    zone : 2
```

Il est impossible qu'un paramètre sans valeur par défaut associée se situe après un paramètre dont une valeur par défaut est précisée. Le programme suivant ne pourra être exécuté.

```
def commander_carte_orange (nom, prenom, paiement="carte", nombre=1, zone):
    print("nom : ", nom)
    # ...
```

Il déclenche l'erreur suivante :

```
File "problem_zone.py", line 1
    def commander_carte_orange (nom, prenom, paiement = "carte", nombre = 1, zone):
SyntaxError: non-default argument follows default argument
```

Les valeurs par défaut de type modifiable (liste, dictionnaire, ensemble, classes) peuvent introduire des erreurs inattendues dans les programmes comme le montre l'exemple suivant :

<<<

```
def fonction(l=[0, 0]):
    l[0] += 1
    return l


print(fonction())         # affiche [1,0] : résultat attendu
print(fonction())         # affiche [2,0] : résultat surprenant
print(fonction([0, 0]))    # affiche [1,0] : résultat attendu
```

>>>

```
    [1, 0]
    [2, 0]
    [1, 0]
```

L'explication provient du fait que la valeur par défaut est une liste qui n'est pas recréée à chaque appel : c'est la même liste à chaque fois que la fonction est appelée sans paramètre. Pour remédier à cela, il faudrait écrire :

<<<

```
import copy


def fonction(l=[0, 0]):
```

(suite sur la page suivante)

(suite de la page précédente)

```
    l = copy.copy(l)
    l[0] += 1
    return l
```

>>>

```

```

L'exercice ex_hypercube propose un exemple plus complet, voire retors.

### Ordre des paramètres

Le paragraphe *Définition, syntaxe* (page 54) a présenté la syntaxe d'appel a une fonction. Lors de l'appel, le nom des paramètres n'intervient plus, supposant que chaque paramètre reçoit pour valeur celle qui a la même position que lui lors de l'appel à la fonction. Il est toutefois possible de changer cet ordre en précisant quel paramètre doit recevoir quelle valeur.

```
x_1, ..., x_n = fonction_nom (param_1 = valeur_1, ..., param_n = valeur_n)
```

Où `fonction_nom` est le nom de la fonction, `param_1` à `param_n` sont les noms des paramètres, `valeur_1` à `valeur_n` sont les valeurs que reçoivent ces paramètres. Avec cette syntaxe, l'ordre d'écriture n'importe pas. La valeur `valeur_i` sera toujours attribuée à `param_i`. Les variables `x_1` à `x_n` reçoivent les résultats retournés par la fonction. L'ordre des résultats ne peut pas être changé. S'il y a plusieurs résultats retournés, il est impossible de choisir lesquels conserver : soit tous, soit aucun.

Exemple :

<<<

```
def identite(nom, prenom):
    print("nom : ", nom, " prénom : ", prenom)


identite("Xavier", "Dupré")                  # nom :  Xavier prénom :  Dupré
identite(prenom="Xavier", nom="Dupré")   # nom :  Dupré  prénom :  Xavier
```

>>>

```
    nom :  Xavier  prénom :  Dupré
    nom :  Dupré  prénom :  Xavier
```

Cette possibilité est intéressante surtout lorsqu'il y a de nombreux paramètres par défaut et que seule la valeur d'un des derniers paramètres doit être changée.

<<<

```
def commander_carte_orange(paiement="carte", nombre=1, zone=2):
    print("paiement : ", paiement)
    print("nombre : ", nombre)
    print("zone :", zone)


commander_carte_orange(zone=5)  # seule la valeur par défaut
# du paramètre zone sera changée
```

>>>

```
    paiement :  carte
    nombre :  1
    zone : 5
```

### Surcharge de fonction

Contrairement à d'autres langages, *python* n'autorise pas la surcharge de fonction. Autrement dit, il n'est pas possible que plusieurs fonctions portent le même nom même si chacune d'entre elles a un nombre différent de paramètres.

<<<

```
def fonction(a, b):
    return a + b


def fonction(a, b, c):
    return a + b + c


print(fonction(5, 6))
print(fonction(5, 6, 7))
```

>>>

```
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 11, in <module>
      File "", line 9, in run_python_script_139651611653248
    TypeError: fonction() missing 1 required positional argument: 'c'
```

Le petit programme précédent est syntaxiquement correct mais son exécution génère une erreur parce que la seconde définition de la fonction `fonction` efface la première.

### Commentaires

Le langage *python* propose une fonction `help` qui retourne pour chaque fonction un commentaire ou mode d'emploi qui indique comment se servir de cette fonction. L'exemple suivant affiche le commentaire associé à la fonction `round`.

```
>>> help (round)

Help on built-in function round:

round(...)
    round(number[, ndigits]) -> floating point number

    Round a number to a given precision in decimal digits (default 0 digits).
    This always returns a floating point number.  Precision may be negative.
```

Lorsqu'on utilise cette fonction `help` sur la fonction `coordonnees_polaires` définie dans l'exemple du paragraphe précédent, le message affiché n'est pas des plus explicites.

```
>>> help (coordonnees_polaires)

Help on function coordonnees_polaires in module __main__:

coordonnees_polaires(x, y)
```

Pour changer ce message, il suffit d'ajouter en première ligne du code de la fonction une chaîne de caractères.

<<<

```
import math


def coordonnees_polaires(x, y):
    """
    convertit des coordonnées cartésiennes en coordonnées polaires
    (x,y) --> (pho,theta)
    """
    rho = math.sqrt(x*x+y*y)
    theta = math.atan2(y, x)
    return rho, theta


help(coordonnees_polaires)
```

>>>

```
    Help on function coordonnees_polaires in module pyquickhelper.sphinxext.sphinx_
↪runpython_extension:

    coordonnees_polaires(x, y)
        convertit des coordonnées cartésiennes en coordonnées polaires
        (x,y) --> (pho,theta)
```

Le programme affiche alors un message d'aide nettement plus explicite. Il est conseillé d'écrire ce commentaire pour toute nouvelle fonction avant même que son corps ne soit écrit. L'expérience montre qu'on oublie souvent de l'écrire après.

### Paramètres modifiables

Les paramètres de types immuables et modifiables se comportent de manières différentes à l'intérieur d'une fonction. Ces paramètres sont manipulés dans le corps de la fonction, voire modifiés parfois. Selon le type du paramètre, ces modifications ont des répercussions à l'extérieur de la fonction.

Les types immuables ne peuvent être modifiés et cela reste vrai. Lorsqu'une fonction accepte un paramètre de type immuable, elle ne reçoit qu'une copie de sa valeur. Elle peut donc modifier ce paramètre sans que la variable ou la valeur utilisée lors de l'appel de la fonction n'en soit affectée. On appelle ceci un passage de paramètre par valeur. A l'opposé, toute modification d'une variable d'un type modifiable à l'intérieur d'une fonction est répercutée à la variable qui a été utilisée lors de l'appel de cette fonction. On appelle ce second type de passage un passage par adresse.

L'exemple suivant utilise une fonction `somme_n_premier_terme` qui modifie ces deux paramètres. Le premier `n`, est immuable, sa modification n'a aucune incidence sur la variable `nb`. En revanche, le premier élément du paramètre `liste` reçoit la valeur 0. Le premier élément de la liste `l` n'a plus la même valeur après l'appel de la fonction `somme_n_premier_terme` que celle qu'il avait avant.

<<<

```
def somme_n_premier_terme(n, liste):
    """calcul la somme des n premiers termes d'une liste"""
    somme = 0
    for i in liste:
        somme += i
        n -= 1              # modification de n (type immuable)
        if n <= 0:
            break
    liste[0] = 0            # modification de liste (type modifiable)
    return somme


l = [1, 2, 3, 4]
nb = 3
# affiche   avant la fonction  3 [1, 2, 3, 4]
print("avant la fonction ", nb, l)
s = somme_n_premier_terme(nb, l)
# affiche   après la fonction  3 [0, 2, 3, 4]
print("après la fonction ", nb, l)
print("somme : ", s)                # affiche   somme :  6
```

>>>

```
    avant la fonction  3 [1, 2, 3, 4]
    après la fonction  3 [0, 2, 3, 4]
    somme :  6
```

La liste `l` est modifiée à l'intérieur de la fonction `somme_n_premier_terme` comme l'affichage suivant le montre. En fait, à l'intérieur de la fonction, la liste `l` est désignée par l'identificateur `liste`, c'est la même liste. La variable `nb` est d'un type immuable. Sa valeur a été recopiée dans le paramètre `n` de la fonction `somme_n_premier_terme`. Toute modification de `n` à l'intérieur de cette fonction n'a aucune répercussion à l'extérieur de la fonction.

*Passage par adresse*

Dans l'exemple précédent, il faut faire distinguer le fait que la liste passée en paramètre ne soit que modifiée et non changée. L'exemple suivant inclut une fonction qui affecte une nouvelle valeur au paramètre `liste` sans pour autant modifier la liste envoyée en paramètre.

```
def fonction (liste):
    liste = []

liste = [0,1,2]
print(liste)       # affiche [0,1,2]
fonction(liste)
print(liste)       # affiche [0,1,2]
```

Il faut considérer dans ce programme que la fonction `fonction` reçoit un paramètre appelé `liste` mais utilise tout de suite cet identificateur pour l'associer à un contenu différent. L'identificateur `liste` est en quelque sorte passé du statut de paramètre à celui de variable locale. La fonction associe une valeur à `liste` - ici, une liste vide - sans toucher à la valeur que cet identificateur désignait précédemment.

Le programme qui suit est différent du précédent mais produit les mêmes effets. Ceci s'explique par le fait que le mot-clé `del` ne supprime pas le contenu d'une variable mais seulement son identificateur. Le langage *python* détecte ensuite qu'un objet n'est plus désigné par aucun identificateur pour le supprimer. Cette remarque est à rapprocher de celles du paragraphe *Copie d'instances* (page 123).

<<<

```
def fonction(liste):
    del liste


liste = [0, 1, 2]
print(liste)        # affiche [0,1,2]
fonction(liste)
print(liste)        # affiche [0,1,2]
```

>>>

```
    [0, 1, 2]
    [0, 1, 2]
```

Le programme qui suit permet cette fois-ci de vider la liste `liste` passée en paramètre à la fonction `fonction`. La seule instruction de cette fonction modifie vraiment le contenu désigné par l'identificateur `liste` et cela se vérifie après l'exécution de cette fonction.

<<<

```
def fonction(liste):
    del liste[0:len(liste)]  # on peut aussi écrire : liste[:] = []


liste = [0, 1, 2]
print(liste)        # affiche [0,1,2]
fonction(liste)
print(liste)        # affiche []
```

>>>

```
    [0, 1, 2]
    []
```

### Fonction récursive

**Définition D3 : fonction récursive**

Une fonction récursive est une fonction qui s'appelle elle-même.

La fonction récursive la plus fréquemment citée en exemple est la fonction factorielle. Celle-ci met en évidence les deux composantes d'une fonction récursive, la récursion proprement dite et la condition d'arrêt.

```
def factorielle(n):
    if n == 0: return 1
    else: return n * factorielle(n-1)
```

La dernière ligne de la fonction `factorielle` est la récursion tandis que la précédente est la condition d'arrêt, sans laquelle la fonction ne cesserait de s'appeler, empêchant le programme de terminer son exécution. Si celle-ci est mal spécifiée ou absente, l'interpréteur *python* affiche une suite ininterrompue de messages. *python* n'autorise pas plus de 1000 appels récursifs : `factorielle(999)` provoque nécessairement une erreur d'exécution même si la condition d'arrêt est bien spécifiée.

```
Traceback (most recent call last):
  File "fact.py", line 5, in <module>
```

(suite sur la page suivante)

(suite de la page précédente)

```
    factorielle(999)
  File "fact.py", line 3, in factorielle
    else : return n * factorielle(n-1)
  File "fact.py", line 3, in factorielle
    else : return n * factorielle(n-1)
  ...
```

La liste des messages d'erreurs est aussi longue qu'il y a eu d'appels à la fonction récursive. Dans ce cas, il faut transformer cette fonction en une fonction non récursive équivalente, ce qui est toujours possible.

```
def factorielle_non_recursive(n):
    r = 1
    for i in range (2, n+1) :
        r *= i
    return r
```

### Portée des variables, des paramètres

Lorsqu'on définit une variable, elle n'est pas utilisable partout dans le programme. Par exemple, elle n'est pas utilisable avant d'avoir été déclarée au moyen d'une affectation. Le court programme suivant déclenche une erreur.

<<<

```
print(x)    # déclenche une erreur
```

>>>

```
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 4, in <module>
      File "", line 3, in run_python_script_139651611478272
    NameError: name 'x' is not defined
```

Il est également impossible d'utiliser une variable à l'extérieur d'une fonction où elle a été déclarée. Plusieurs fonctions peuvent ainsi utiliser le même nom de variable sans qu'à aucun moment, il n'y ait confusion. Le programme suivant déclenche une erreur identique à celle reproduite ci-dessus.

<<<

```
def portee_variable(x):
    var = x
    print(var)


portee_variable(3)
print(var)          # déclenche une erreur car var est déclarée dans
# la fonction portee_variable
```

>>>

```
    3
    [runpythonerror]

    Traceback (most recent call last):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        exec(obj, globals(), loc)
      File "", line 10, in <module>
      File "", line 8, in run_python_script_139651611653320
    NameError: name 'var' is not defined
```

**Définition D4 : portée d'un variable**

La portée d'une variable associée à un identificateur recouvre la portion du programme à l'intérieur de laquelle ce même identificateur la désigne. Ceci implique que, dans cette portion de code, aucune autre variable, aucune autre fonction, aucune autre classe, ne peut porter le même identificateur.

Une variable n'a donc d'existence que dans la fonction dans laquelle elle est déclarée. On appelle ce type de variable une variable locale. Par défaut, toute variable utilisée dans une fonction est une variable locale.

**Définition D5 : variable locale**

Une variable locale est une variable dont la portée est réduite à une fonction.

Par opposition aux variables locales, on définit les variables globales qui sont déclarées à l'extérieur de toute fonction.

**Définition D6 : variable globale**

Une variable globale est une variable dont la portée est l'ensemble du programme.

L'exemple suivant mélange variable locale et variable globale. L'identificateur `n` est utilisé à la fois pour désigner une variable globale égale à 1 et une variable locale égale à 1. A l'intérieur de la fonction, `n` désigne la variable locale égale à 2. A l'extérieur de la fonction, `n` désigne la variable globale égale à 1.

<<<

```
n = 1                    # déclaration d'une variable globale


def locale_globale():
    n = 2                # déclaration d'une variable locale
    print(n)             # affiche le contenu de la variable locale


print(n)                 # affiche 1
locale_globale()         # affiche 2
print(n)                 # affiche 1
```

>>>

```
    1
    2
    1
```

Il est possible de faire référence aux variables globales dans une fonction par l'intermédiaire du mot-clé `global`. Celui-ci indique à la fonction que l'identificateur `n` n'est plus une variable locale mais désigne une variable globale déjà déclarée.

<<<

```
n = 1                    # déclaration d'une variable globale


def locale_globale():
    global n             # cette ligne indique que n désigne la variable globale
    n = 2                # change le contenu de la variable globale
    print(n)             # affiche le contenu de la variable globale


print(n)                 # affiche 1
locale_globale()         # affiche 2
print(n)                 # affiche 2
```

>>>

```
    1
    2
    1
```

Cette possibilité est à éviter le plus possible car on peut considérer que `locale_globale` est en fait une fonction avec un paramètre caché. La fonction `locale_globale` n'est plus indépendante des autres fonctions puisqu'elle modifie une des données du programme.

### Portée des fonctions

Le langage *python* considère les fonctions également comme des variables d'un type particulier. La portée des fonctions obéit aux mêmes règles que celles des variables. Une fonction ne peut être appelée que si elle a été définie avant son appel.

<<<

```
def factorielle(n):
    # ...
    return 1


print(type(factorielle))  # affiche <type 'function'>
```

>>>

```
    <class 'function'>
```

Comme il est possible de déclarer des variables locales, il est également possible de définir des fonctions locales ou fonctions imbriquées. Une fonction locale n'est appelable qu'à l'intérieur de la fonction dans laquelle elle est définie. Dans l'exemple suivant, la fonction `affiche_pair` inclut une fonction locale qui n'est appelable que par cette fonction `affiche_pair`.

<<<

```
def affiche_pair():
    def fonction_locale(i):            # fonction locale ou imbriquée
        if i % 2 == 0:
            return True
        else:
            return False
    for i in range(0, 10):
        if fonction_locale(i):
```

(suite sur la page suivante)

(suite de la page précédente)

```
            print(i)


affiche_pair()
fonction_locale(5)      # l'appel à cette fonction locale
# déclenche une erreur d'exécution
```

>>>

```
    0
    2
    4
    6
    8
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 14, in <module>
      File "", line 12, in run_python_script_139651611429912
    NameError: name 'fonction_locale' is not defined
```

A l'intérieur d'une fonction locale, le mot-clé `global` désigne toujours les variables globales du programme et non les variables de la fonction dans laquelle cette sous-fonction est définie.

### Nombre de paramètres variable

Il est possible de définir des fonctions qui prennent un nombre indéterminé de paramètres, lorsque celui-ci n'est pas connu à l'avance. Hormis les paramètres transmis selon le mode présenté dans les paragraphes précédents, des informations peuvent être ajoutées à cette liste lors de l'appel de la fonction, ces informations sont regroupées soit dans une liste de valeurs, soit dans une liste de couples (identificateur, valeur). La déclaration d'une telle fonction obéit à la syntaxe suivante :

**Syntaxe S9 : Nombre indéfini de paramètres**

```
def fonction (param_1, ..., param_n, *liste, **dictionnaire) :
```

Où `fonction` est un nom de fonction, `param_1` à `param_n` sont des paramètres de la fonction, `liste` est le nom de la liste qui doit recevoir la liste des valeurs seules envoyées à la fonction et qui suivent les paramètres (plus précisément, c'est un tuple), `dictionnaire` reçoit la liste des couples (identificateur, valeur). L'appel à cette fonction suit quant à lui la syntaxe suivante :

```
fonction (valeur_1, ..., valeur_n, \
          liste_valeur_1, ..., liste_valeur_p, \
          nom_1 = v_1, ..., nom_q = v_q)
```

Où `fonction` est un nom de fonction, `valeur_1` à `valeur_n` sont les valeurs associées aux paramètres `param_1` à `param_n`, `liste_valeur_1` à `liste_valeur_p` formeront la liste `liste`, les couples `nom_1 : v_1` à `nom_q : v_q` formeront le dictionnaire `dictionnaire`.

Exemple :

<<<

```
def fonction(p, *l, **d):
    print("p = ", p)
    print("liste (tuple) l :", l)
    print("dictionnaire d :", d)


fonction(1, 2, 3, a=5, b=6)  # 1 est associé au paramètre p
# 2 et 3 sont insérés dans la liste l
# a=5 et b=6 sont insérés dans le dictionnaire d
```

>>>

```
    p =  1
    liste (tuple) l : (2, 3)
    dictionnaire d : {'a': 5, 'b': 6}
```

A l'instar des paramètres par défaut, la seule contrainte de cette écriture est la nécessité de respecter l'ordre dans lequel les informations doivent apparaître. Lors de l'appel, les valeurs sans précision de nom de paramètre seront placées dans une liste (ici le tuple `l`). Les valeurs associées à un nom de paramètre seront placées dans un dictionnaire (ici `d`). Les valeurs par défaut sont obligatoirement placées après les paramètres non nommés explicitement.

Une fonction qui accepte des paramètres en nombre variable peut à son tour appeler une autre fonction acceptant des paramètres en nombre variable. Il faut pour cela se servir du symbole `*` afin de transmettre à `fonction` les valeurs reçues par `fonction2`.

<<<

```
def fonction(p, *l, **d):
    print("p = ", p)
    print("liste l :", l)
    print("dictionnaire d :", d)


def fonction2(p, *l, **d):
    l += (4,)               # on ajoute une valeur au tuple
    d["c"] = 5             # on ajoute un couple (paramètre,valeur)
    fonction(p, *l, **d)  # ne pas oublier le symbole *


fonction2(1, 2, 3, a=5, b=6)
```

>>>

```
    p =  1
    liste l : (2, 3, 4)
    dictionnaire d : {'a': 5, 'b': 6, 'c': 5}
```

### Ecriture simplifiée pour des fonctions simples : lambda

Lorsque le code d'une fonction tient en une ligne et est le résultat d'une expression, il est possible de condenser son écriture à l'aide du mot-clé `lambda`.

```
nom_fonction = lambda param_1, ..., param_n : expression
```

`nom_fonction` est le nom de la fonction, `param_1` à `param_n` sont les paramètres de cette fonction (ils peuvent également recevoir des valeurs par défaut), `expression` est l'expression retournée par la fonction.

L'exemple suivant utilise cette écriture pour définir la fonction `min` retournant le plus petit entre deux nombres positifs.

<<<

```
def min(x, y): return (abs(x+y) - abs(x-y))/2


print(min(1, 2))       # affiche 1
print(min(5, 4))       # affiche 4
```

>>>

```
    1.0
    4.0
```

Cette écriture correspond à l'écriture non condensée suivante :

<<<

```
def min(x, y):
    return (abs(x+y) - abs(x-y))/2


print(min(1, 2))       # affiche 1
print(min(5, 4))       # affiche 4
```

>>>

```
    1.0
    4.0
```

La fonction `lambda` considère le contexte de fonction qui la contient comme son contexte. Il est possible de créer des fonctions `lambda` mais celle-ci utiliseront le contexte dans l'état où il est au moment de son exécution et non au moment de sa création.

<<<

```
fs = []
for a in range(0, 10):
    def f(x): return x + a
    fs.append(f)
for f in fs:
    print(f(1))    # le programme affiche 10 fois 10 de suite
    # car la variable a vaut dix à la fin de la boucle
```

>>>

```
    10
    10
    10
    10
    10
    10
    10
    10
    10
    10
```

Pour que le programme affiche les entiers de 1 à 10, il faut préciser à la fonction `lambda` une variable `y` égale à `a` au moment de la création de la fonction et qui sera intégrée au contexte de la fonction `lambda` :

<<<

```
fs = []
for a in range(0, 10):
    f = lambda x, y=a: x + y    # ligne changée
    fs.append(f)
for f in fs:
    print(f(1))
```

>>>

```
    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
```

### Fonctions générateur

Le mot-clé `yield` est un peu à part. Utilisé à l'intérieur d'une fonction, il permet d'interrompre le cours de son exécution à un endroit précis de sorte qu'au prochain appel de cette fonction, celle-ci reprendra le cours de son exécution exactement au même endroit avec des variables locales inchangées. Le mot-clé `return` ne doit pas être utilisé. Ces fonctions ou générateurs [71] sont utilisées en couple avec le mot-clé `for` pour simuler un ensemble. L'exemple suivant implémente une fonction `fonction_yield` qui simule l'ensemble des entiers compris entre 0 et *n* exclu

<<<

```
def fonction_yield(n):
    i = 0
    while i < n-1:
        print("yield 1")  # affichage : pour voir ce que fait le programme
        yield i           # arrête la fonction qui reprendra
        i = i+1           # à la ligne suivante lors du prochain appel
    print("yield 2")      # affichage : pour voir ce que fait le programme
    yield i               # arrête la fonction qui ne reprendra pas
    # lors du prochain appel car le code de la fonction
    # prend fin ici


for a in fonction_yield(2):
    print(a)                # affiche tous les éléments que retourne la
    # fonction fonction_yield, elle simule la liste
    # [0,1]
print("-----------------------------------------------")
for a in fonction_yield(3):
    print(a)                # nouvel appel, l'exécution reprend
    # au début de la fonction,
    # affiche tous les éléments que retourne la
    # fonction fonction_yield, elle simule la liste
    # [0,1,2]
```

>>>

71. https://docs.python.org/3/glossary.html#term-generator

```
    yield 1
    0
    yield 2
    1
    ------------------------------------------------
    yield 1
    0
    yield 1
    1
    yield 2
    2
```

Le programme affiche tous les entiers compris entre~0 et 4 inclus ainsi que le texte `"yield 1"` ou `"yield 2"` selon l'instruction `yield` qui a retourné le résultat. Lorsque la fonction a finalement terminé son exécution, le prochain appel agit comme si c'était la première fois qu'on l'appelait.

### Identificateur appelable

La fonction `callable` retourne un booléen permettant de savoir si un identificateur est une fonction (voir *Classes* (page 95)), de savoir par conséquent si tel identificateur est appelable comme une fonction.

<<<

```
x = 5


def y():
    return None


print(callable(x))  # affiche False car x est une variable
print(callable(y))  # affiche True car y est une fonction
```

>>>

```
    False
    True
```

### Compilation dynamique (eval)

Cette fonction a déjà été abordée lors des paragraphes :*Fonction print, repr et conversion en chaîne de caractères* (page 31) ou *Fonction eval* (page 32). Elle évalue toute chaîne de caractères contenant une expression écrite avec la syntaxe du langage *python*. Cette expression peut utiliser toute variable ou toute fonction accessible au moment où est appelée la fonction `eval`.

<<<

```
x = 3
y = 4
print(eval("x*x+y*y+2*x*y"))  # affiche 49
print((x+y)**2)                # affiche 49
```

>>>

```
    49
    49
```

Si l'expression envoyée à la fonction `eval` inclut une variable non définie, l'interpréteur *python* génère une erreur comme le montre l'exemple suivant.

<<<

```
x = 3
y = 4
print(eval("x*x+y*y+2*x*y+z"))
```

>>>

```
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 6, in <module>
      File "", line 5, in run_python_script_139651611158352
      File "<string>", line 1, in <module>
    NameError: name 'z' is not defined
```

La variable `z` n'est pas définie et l'expression n'est pas évaluable. L'erreur se produit dans une chaîne de caractères traduite en programme informatique, c'est pourquoi l'interpréteur ne peut pas situer l'erreur dans un fichier. L'erreur ne se produit dans aucun fichier, cette chaîne de caractères pourrait être définie dans un autre.

### Compilation dynamique (compile, exec)

Plus complète que la fonction eval [72], la fonction compile [73] permet d'ajouter une ou plusieurs fonctions au programme, celle-ci étant définie par une chaîne de caractères. Le code est d'abord compilé (fonction `compile`) puis incorporé au programme (fonction exec [74]) comme le montre l'exemple suivant.

<<<

```
import math
str = """def coordonnees_polaires(x,y):
    rho     = math.sqrt(x*x+y*y)
    theta   = math.atan2 (y,x)
    return rho, theta"""          # fonction définie par une chaîne de caractères

obj = compile(str, "", "exec")      # fonction compilée
exec(obj)                           # fonction incorporée au programme
# affiche (1.4142135623730951, 0.78539816339744828)
print(coordonnees_polaires(1, 1))
```

>>>

```
    (1.4142135623730951, 0.7853981633974483)
```

La fonction `compile` prend en fait trois arguments. Le premier est la chaîne de caractères contenant le code à compiler. Le second paramètre (`""` dans l'exemple) contient un nom de fichier dans lequel seront placées les erreurs de compilation. Le troisième paramètre est une chaîne de caractères à choisir parmi ńăexecă" ou ńăevală". Selon ce choix, ce sera la fonction `exec` ou `eval` qui devra être utilisée pour agréger le résultat de la fonction `compile` au programme. L'exemple suivant donne un exemple d'utilisation de la fonction `compile` avec la fonction `eval`.

<<<

72. https://docs.python.org/3/library/functions.html?highlight=eval#eval
73. https://docs.python.org/3/library/functions.html?highlight=eval#compile
74. https://docs.python.org/3/library/functions.html?highlight=eval#exec

```
import math
str = """math.sqrt(x*x+y*y)"""  # expression définie par une chaîne de caractères

obj = compile(str, "", "eval")    # expression compilée
x = 1
y = 2
print(eval(obj))                # résultat de l'expression
```

>>>

```
    2.23606797749979
```

### 2.2.5 Indentation

L'indentation est synonyme de décalage. Pour toute boucle, test, fonction, et plus tard, toute définition de classe, le fait d'indenter ou décaler les lignes permet de définir une dépendance d'un bloc de lignes par rapport à un autre. Les lignes indentées par rapport à une boucle `for` dépendent de celle-ci puisqu'elle seront exécutées à chaque passage dans la boucle. Les lignes indentées par rapport au mot-clé `def` sont considérées comme faisant partie du corps de la fonction.

IndentationError [75] est l'erreur que l'interpréteur *python* retourne en cas de mauvaise indentation (voir *tests* (page 39)).

Contrairement à d'autres langages comme le C [76] ou PERL [77], *python* n'utilise pas de délimiteurs pour regrouper les lignes. L'indentation, souvent présentée comme un moyen de rendre les programmes plus lisibles, est ici intégrée à la syntaxe du langage. Il n'y a pas non plus de délimiteurs entre deux instructions autre qu'un passage à la ligne. Le caractère \ placé à la fin d'une ligne permet de continuer l'écriture d'une instruction à la ligne suivante.

### 2.2.6 Fonctions usuelles

Certaines fonctions sont communes aux dictionnaires et aux listes, elles sont également définis pour de nombreux objets présents dans les extensions du langages. Quelque soit le contexte, le résultat attendu à la même signification. Les plus courantes sont présentées *plus bas* (page 73).

La fonction map [78] permet d'écrire des boucles de façon simplifiée. Elle est utile dans le cas où on souhaite appliquer la même fonction à tous les éléments d'un ensemble. Par exemple les deux dernières lignes du programme suivant sont équivalentes.

<<<

```
def est_pair(n):
    return n % 2 == 0


l = [0, 3, 4, 4, 5, 6]
print([est_pair(i) for i in l])  # affiche [0, 1, 0, 0, 1, 0]
print(map(est_pair, l))
print(list(map(est_pair, l)))       # affiche [0, 1, 0, 0, 1, 0]
```

>>>

```
    [True, False, True, True, False, True]
    <map object at 0x7f032cab9710>
    [True, False, True, True, False, True]
```

75. https://docs.python.org/3/library/exceptions.html?highlight=indentationerror#IndentationError
76. https://fr.wikipedia.org/wiki/C_(langage)
77. https://fr.wikipedia.org/wiki/Perl_(langage)
78. https://docs.python.org/3/library/functions.html?highlight=map#map

La fonction map [79] retourne un itérateur et non un ensemble. Cela explique le second résultat du programme précédent. Pour obtenir les résultats, il faut explicitement parcourir l'ensemble des résultats. C'est ce que fait la dernière instruction. La fonction map [80] est une *fonction générateur* (page 68). Elle peut aider à simplifier l'écriture lorsque plusieurs listes sont impliquées. Ici encore, les deux dernières lignes sont équivalentes.

<<<

```
def addition(x, y):
    return x + y


li = [0, 3, 4, 4, 5, 6]
mo = [1, 3, 4, 5, 6, 8]
print([addition(li[i], mo[i]) for i in range(0, len(li))])
print(list(map(addition, li, mo)))   # affiche [1, 6, 8, 9, 11, 14]
```

>>>

```
    [1, 6, 8, 9, 11, 14]
    [1, 6, 8, 9, 11, 14]
```

Il est possible de substituer d'utiliser la fonction `map` pour obtenir l'équivalent de la fonction zip [81].

<<<

```
li = [0, 3, 4, 4, 5, 6]
mo = [1, 3, 4, 5, 6, 8]
print(list(map((lambda x, y: (x, y)), li, mo)))
print(list(zip(li, mo)))
```

>>>

```
    [(0, 1), (3, 3), (4, 4), (4, 5), (5, 6), (6, 8)]
    [(0, 1), (3, 3), (4, 4), (4, 5), (5, 6), (6, 8)]
```

Comme pour les dictionnaires, la fonction sorted [82] permet de parcourir les éléments d'une liste de façon ordonnée. Les deux exemples qui suivent sont presque équivalents. Dans le second, la liste `li` demeure inchangée alors qu'elle est triée dans le premier programme.

<<<

```
li = [4, 5, 3, -6, 7, 9]

for n in sorted(li):     # on parcourt la liste li
    print(n)             # de façon triée
print(li)                # la liste li n'est pas triée

li.sort()                # la liste est triée
for n in li:
    print(n)
```

>>>

```
    -6
    3
```

(suite sur la page suivante)

79. https://docs.python.org/3/library/functions.html?highlight=map#map
80. https://docs.python.org/3/library/functions.html?highlight=map#map
81. https://docs.python.org/3/library/functions.html?highlight=map#zip
82. https://docs.python.org/3/library/functions.html?highlight=map#sorted

(suite de la page précédente)

```
    4
    5
    7
    9
    [4, 5, 3, -6, 7, 9]
    -6
    3
    4
    5
    7
    9
```

La fonction enumerate [83] permet d'éviter l'emploi de la fonction range [84] lorsqu'on souhaite parcourir une liste alors que l'indice et l'élément sont nécessaires.

<<<

```
li = [4, 5, 3, -6, 7, 9]

for i in range(0, len(li)):
    print(i, li[i])

print("--")

for i, v in enumerate(li):
    print(i, v)
```

>>>

```
    0 4
    1 5
    2 3
    3 -6
    4 7
    5 9
    --
    0 4
    1 5
    2 3
    3 -6
    4 7
    5 9
```

Voici la liste non exhaustive de fonctions définies par le langage *python* sans qu'aucune extension ne soit nécessaire.

83. https://docs.python.org/3/library/functions.html?highlight=map#enumerate
84. https://docs.python.org/3/library/functions.html?highlight=map#func-range

| | |
|---|---|
| `abs (x)` | Retourne la valeur absolue de `x`. |
| `callable (x)` | Dit si la variable `x` peut être appelée. |
| `chr (i)` | Retourne le caractère associé au code numérique `i`. |
| `cmp (x,y)` | Compare `x` et `y`, retourne -1 si `x<y`, 0 en cas d'égalité, 1 sinon. |
| `dir (x)` | Retourne l'ensemble des méthodes associées à `x` qui peut être un objet, un module, un variable, ... |
| `enumerate(x)` | Parcourt un ensemble itérable (voir paragraphe fonction_sorted_enumerate. |
| `help(x)` | Retourne l'aide associée à `x`. |
| `id(x)` | Retourne un identifiant unique associé à l'objet `x`. Le mot-clé `is` est relié à cet identifiant. |
| `isinstance(x, classe)` | Dit si l'objet `x` est de type `classe` (voir le chapitre *Classes* (page 95)). |
| `issubclass(cl1, cl2)` | Dit si la classe `cl1` hérite de la classe `cl2` (voir le chapitre *Classes* (page 95)). |
| `len(l)` | Retourne la longueur de `l`. |
| `map(f,l1,l2,...)` | Applique la fonction `f` sur les listes `l1`, `l2`... |
| `max(l)` | Retourne le plus grand élément de `l`. |
| `min(l)` | Retourne le plus petit élément de `l`. |
| `ord(s)` | Fonction réciproque de `chr`. |
| `range(i,j[,k])` | Construit la liste des entiers de `i` à `j`. Si `k` est précisé, va de `k` en `k` à partir de `i`. |
| `reload(module)` | Recharge un module (voir *Modules* (page 235)). |
| `repr(o)` | Retourne une chaîne de caractères qui représente l'objet `o`. |
| `round(x[,n])` | Arrondi `x` à `n` décimales près ou aucune si `n` n'est pas précisé. |
| `sorted(x [, cmp[, key[, reverse]]])` | Tri un ensemble itérable (voir paragraphe fonction_sorted_enumerate) |
| `str(o)` | Retourne une chaîne de caractères qui représente l'objet `o`. |
| `sum(l)` | Retourne la somme de l'ensemble `l`. |
| `type(o)` | Retourne le type de la variable `o`. |
| `zip(l1,l2,...)` | Construit une liste de tuples au lieu d'un tuple de listes. |

### 2.2.7 Constructions classiques

Il fait aller à *Constructions classiques* (page 74).

Ces paragraphes qui suivent décrivent des schémas qu'on retrouve dans les programmes dans de nombreuses situations. Ce sont des combinaisons simples d'une ou deux boucles, d'un test, d'une liste, d'un dictionnaire.

## 2.3 Constructions classiques


— *Constructions classiques* (page 74)
— *Constructions négatives* (page 83)


### 2.3.1 Constructions classiques

1. *calcul d'une somme* (page **??**)
2. *calcul de la somme des dix premiers entiers au carré* (page **??**)
3. *comptage* (page **??**)
4. *conversion d'un vecteur en une matrice* (page **??**)

5. *conversion d'une chaîne de caractère en datetime* (page **??**)
6. *conversion d'une chaîne de caractère en matrice* (page **??**)
7. *conversion d'une matrice en chaîne de caractères* (page **??**)
8. *conversion d'une matrice en un vecteur* (page **??**)
9. *fonction comme paramètre* (page **??**)
10. *minimum avec position* (page **??**)
11. *recherche avec index* (page **??**)
12. *recherche dichotomique* (page **??**)
13. *tri, garder les positions initiales* (page **??**)

---

**calcul d'une somme**

Le calcul d'une somme fait toujours intervenir une boucle car le langage Python [85] ne peut faire des additions qu'avec deux nombres. Le schéma est toujours le même : initialisation et boucle.

<<<

```
li = [0, 434, 43, 6456]
s = 0                          # initialisation
for l in li:                   # boucle
    s += l                      # addition
print(s)
```

>>>

```
    6933
```

Ce code est équivalent à la fonction sum [86]. Dans ce cas où la somme intègre le résultat d'une fonction (au sens mathématique) et non les éléments d'une liste, il faudrait écrire :

<<<

```
def fonction(x):
    return x


li = [0, 434, 43, 6456]
s = 0
for l in li:
    s += fonction(l)
print(s)
```

>>>

```
    6933
```

Et ces deux lignes pourraient être résumées en une seule grâce à l'une de ces instructions :

<<<

```
def fonction(x):
    return x
```

(suite sur la page suivante)

85. http://www.python.org/
86. https://docs.python.org/3/library/functions.html#sum

(suite de la page précédente)

```
li = [0, 434, 43, 6456]
s1 = sum([fonction(l) for l in li])
s2 = sum(fonction(l) for l in li)
s3 = sum(map(fonction, li))
print(s1, s2, s3)
```

>>>

```
    6933 6933 6933
```

L'avantage des deux dernières instructions est qu'elles évitent la création d'une liste intermédiaire, c'est un point à prendre en compte si la liste sur laquelle opère la somme est volumineuse.

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.somme, line 6)

---

**calcul de la somme des dix premiers entiers au carré**

Ce calcul simple peut s'écrire de diffèrentes manières.

```
s = 0
for i in range(1,11):
    s += i**2
```

D'une façon abrégée :

```
s = sum ( [ i**2 for i in range(1,11) ] )
```

---

(*entrée originale* : classiques.py :docstring of teachpyx.examples.classiques.dix_entiers_carre, line 18)

---

**comptage**

On souhaite ici compter le nombre d'occurrences de chaque élément d'un tableau. Par exemple, on pourrait connaître par ce moyen la popularité d'un mot dans un discours politique ou l'étendue du vocabulaire utilisé. L'exemple suivant compte les mots d'une liste de mots.

<<<

```
li = ["un", "deux", "un", "trois"]
d = {}
for l in li:
    if l not in d:
        d[l] = 1
    else:
        d[l] += 1
print(d)    # affiche {'un': 2, 'trois': 1, 'deux': 1}
```

>>>

```
    {'un': 2, 'deux': 1, 'trois': 1}
```

La structure la plus appropriée ici est un dictionnaire puisqu'on cherche à associer une valeur à un élément d'une liste qui peut être de tout type. Si la liste contient des éléments de type modifiable comme une liste, il faudrait convertir ceux-ci en un type immuable comme une chaîne de caractères. L'exemple suivant illustre ce cas en comptant les occurrences des lignes d'une matrice.

<<<

```
mat = [[1, 1, 1], [2, 2, 2], [1, 1, 1]]
d = {}
for l in mat:
    k = str(l)     # k = tuple (l) lorsque cela est possible
    if k not in d:
        d[k] = 1
    else:
        d[k] += 1
print(d)    # affiche {'[1, 1, 1]': 2, '[2, 2, 2]': 1}
```

>>>

```
    {'[1, 1, 1]': 2, '[2, 2, 2]': 1}
```

Les listes ne peuvent pas être les clés du dictionnaire : Why Lists Can't Be Dictionary Keys [87].

On peut également vouloir non pas compter le nombre d'occurrence mais mémoriser les positions des éléments tous identiques. On doit utiliser un dictionnaire de listes :

<<<

```
li = ["un", "deux", "un", "trois"]
d = {}
for i, v in enumerate(li):
    if v not in d:
        d[v] = [i]
    else:
        d[v].append(i)
print(d)    # affiche {'un': [0, 2], 'trois': [3], 'deux': [1]}
```

>>>

```
    {'un': [0, 2], 'deux': [1], 'trois': [3]}
```

S'il suffit juste de compter, l'écriture la plus simple est :

<<<

```
r = {}
li = ["un", "deux", "un", "trois"]
for x in li:
    r[x] = r.get(x, 0) + 1
print(r)
```

>>>

```
    {'un': 2, 'deux': 1, 'trois': 1}
```

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.compte, line 6)

---

**conversion d'un vecteur en une matrice**

Dans un langage comme le *C++*, il arrive fréquemment qu'une matrice ne soit pas représentée par une liste de listes mais par une seule liste car cette représentation est plus efficace. Il faut donc convertir un indice en deux indices ligne et colonne. Il faut bien sûr que le nombre de colonnes sur chaque ligne soit constant. Le premier programme convertit une liste de listes en une seule liste.

<<<

87. https://wiki.python.org/moin/DictionaryKeys

```
ncol = 2
vect = [0, 1, 2, 3, 4, 5]
mat = [vect[i*ncol: (i+1)*ncol] for i in range(0, len(vect)//ncol)]
print(mat)
```

>>>

```
    [[0, 1], [2, 3], [4, 5]]
```

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.vect2mat, line 8)

---

**conversion d'une chaîne de caractère en datetime**

C'est le genre de fonction qu'on n'utilise pas souvent mais qu'on peine à retrouver lorsqu'on en a besoin. Il faut utiliser la fonction strftime [88].

```
import datetime
dt = datetime.datetime.strptime ("16/01/2014", "%d/%m/%Y")
```

---

(*entrée originale* : classiques.py :docstring of teachpyx.examples.classiques.str2date, line 7)

---

**conversion d'une chaîne de caractère en matrice**

Les quelques lignes qui suivent permettent de décomposer une chaîne de caractères en matrice. Chaque ligne et chaque colonne sont séparées par des séparateurs différents. Ce procédé intervient souvent lorsqu'on récupère des informations depuis un fichier texte lui-même provenant d'un tableur.

<<<

```
s = "case11;case12;case13|case21;case22;case23"
# décomposition en matrice
ligne = s.split("|")                        # lignes
mat = [l.split(";") for l in ligne]  # colonnes

print(mat)
```

>>>

```
    [['case11', 'case12', 'case13'], ['case21', 'case22', 'case23']]
```

Comme cette opération est très fréquente lorsqu'on travaille avec les données, on ne l'implémente plus soi-même. On préfère utiliser un module comme pandas [89] qui est plus robuste et considère plus de cas. Pour écrire, utilise la méthode to_csv [90], pour lire, la fonction read_csv [91]. On peut également directement enregistrer au format Excel read_excel [92] et écrire dans ce même format to_excel [93].

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.text2mat, line 9)

---

**conversion d'une matrice en chaîne de caractères**

<<<

88. https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior
89. http://pandas.pydata.org/
90. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html
91. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.io.parsers.read_csv.html
92. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.io.excel.read_excel.html
93. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html

```
mat = [['case11', 'case12', 'case13'], ['case21', 'case22', 'case23']]
ligne = [";".join(l) for l in mat]     # colonnes
s = "|".join(ligne)                    # lignes

print(s)
```

>>>

```
    case11;case12;case13|case21;case22;case23
```

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.mat2text, line 9)

---

**conversion d'une matrice en un vecteur**

Dans un langage comme le *C++*, il arrive fréquemment qu'une matrice ne soit pas représentée par une liste de listes mais par une seule liste car cette représentation est plus efficace. Il faut donc convertir un indice en deux indices ligne et colonne. Il faut bien sûr que le nombre de colonnes sur chaque ligne soit constant. Le premier programme convertit une liste de listes en une seule liste.

<<<

```
mat = [[0, 1, 2], [3, 4, 5]]
lin = [i * len(mat[i]) + j
       for i in range(0, len(mat))
       for j in range(0, len(mat[i]))]
print(lin)
```

>>>

```
    [0, 1, 2, 3, 4, 5]
```

Vous pouvez aussi utiliser des fonctions telles que reduce [94].

<<<

```
from functools import reduce
mat = [[0, 1, 2], [3, 4, 5]]
lin = reduce(lambda x, y: x+y, mat)
print(lin)
```

>>>

```
    [0, 1, 2, 3, 4, 5]
```

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.mat2vect, line 7)

---

**fonction comme paramètre**

Une fonction peut aussi recevoir en paramètre une autre fonction. L'exemple suivant inclut la fonction `calcul_n_valeur` qui prend comme paramètres `l` et `f`. Cette fonction calcule pour toutes les valeurs `x` de la liste `l` la valeur `f(x)`. `fonction_carre` ou `fonction_cube` sont passées en paramètres à la fonction `calcul_n_valeur` qui les exécute.

<<<

94. https://docs.python.org/3/library/functools.html?highlight=reduce#module-functools

```
def fonction_carre(x):
    return x*x


def fonction_cube(x):
    return x*x*x


def calcul_n_valeur(l, f):
    res = [f(i) for i in l]
    return res


l = [0, 1, 2, 3]
print(l)   # affiche [0, 1, 2, 3]

l1 = calcul_n_valeur(l, fonction_carre)
print(l1)  # affiche [0, 1, 4, 9]

l2 = calcul_n_valeur(l, fonction_cube)
print(l2)  # affiche [0, 1, 8, 27]
```

>>>

```
    [0, 1, 2, 3]
    [0, 1, 4, 9]
    [0, 1, 8, 27]
```

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.integrale, line 10)

---

**minimum avec position**

La fonction min[95] retourne le minium d'un tableau mais pas sa position. Le premier réflexe est alors de recoder le parcours de la liste tout en conservant la position du minimum.

<<<

```
li = [0, 434, 43, 6436, 5]
m = 0
for i in range(0, len(li)):
    if li[m] < li[i]:
        m = i
print(m)
```

>>>

```
    3
```

Mais il existe une astuce pour obtenir la position sans avoir à le reprogrammer.

<<<

```
li = [0, 434, 43, 6436, 5]
k = [(v, i) for i, v in enumerate(li)]
m = min(k)
print(m)
```

95. https://docs.python.org/3/library/functions.html#min

>>>

```
    (0, 0)
```

La fonction `min` choisit l'élément minimum d'un tableau dont les éléments sont des couples (élément du premier tableau, sa position). Le minimum est choisi en comparant les éléments, et la position départegera les exaequo.

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.minindex, line 7)

---

**recherche avec index**

Lorsqu'on cherche un élément dans un tableau, on cherche plus souvent sa position que le fait que le tableau contient cet élément.

<<<

```
def recherche(li, c):
    for i, v in enumerate(li):
        if v == c:
            return i
    return -1


li = [45, 32, 43, 56]
print(recherche(li, 43))  # affiche 2
```

>>>

```
    2
```

En python, il existe un fonction simple qui permet de faire ça :

```
print(li.index(43))  # affiche 2
```

Lorsque l'élément n'y est pas, on retourne souvent la position `-1` qui ne peut être prise par aucun élément :

```
if c in li: return li.index(c)
else: return -1
```

Même si ce bout de code parcourt deux fois le tableau (une fois déterminer sa présence, une seconde fois pour sa position), ce code est souvent plus rapide que la première version et la probabilité d'y faire une erreur plus faible.

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.recherche, line 7)

---

**recherche dichotomique**

La recherche dichotomique [96] est plus rapide qu'une recherche classique mais elle suppose que celle-ci s'effectue dans un ensemble trié. L'idée est de couper en deux l'intervalle de recherche à chaque itération. Comme l'ensemble est trié, en comparant l'élément cherché à l'élément central, on peut éliminer une partie de l'ensemble : la moitié inférieure ou supérieure.

<<<

96. http://fr.wikipedia.org/wiki/Dichotomie

```
def recherche_dichotomique(li, c):
    a, b = 0, len(li)-1
    while a <= b:
        m = (a + b)//2
        if c == li[m]:
            return m
        elif c < li[m]:
            b = m-1
        else:
            a = m+1
    return -1


print(recherche_dichotomique([0, 2, 5, 7, 8], 7))
```

>>>

```
    3
```

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.recherche_dichotomique, line 7)

**tri, garder les positions initiales**

Le tri est une opération fréquente. On n'a pas toujours le temps de programmer le tri le plus efficace comme un tri quicksort[97] et un tri plus simple suffit la plupart du temps. Le tri suivant consiste à recherche le plus petit élément puis à échanger sa place avec le premier élément du tableau du tableau. On recommence la même procédure à partir de la seconde position, puis la troisième et ainsi de suite jusqu'à la fin du tableau.

<<<

```
li = [5, 6, 4, 3, 8, 2]

for i in range(0, len(li)):
    # recherche du minimum entre i et len (li) exclu
    pos = i
    for j in range(i+1, len(li)):
        if li[j] < li[pos]:
            pos = j
    # échange
    ech = li[pos]
    li[pos] = li[i]
    li[i] = ech

print(li)
```

>>>

```
    [2, 3, 4, 5, 6, 8]
```

La fonction sorted[98] trie également une liste mais selon un algorithme plus efficace que celui-ci (voir Timsort[99]). On est parfois amené à reprogrammer un tri parce qu'on veut conserver la position des éléments dans le tableau non trié. Cela arrive quand on souhaite trier un tableau et appliquer la même transformation à un second tableau. Il est toujours

97. http://fr.wikipedia.org/wiki/Tri_rapide
98. https://docs.python.org/3/library/functions.html#sorted
99. http://en.wikipedia.org/wiki/Timsort

préférable de ne pas reprogrammer un tri (moins d'erreur). Il suffit d'appliquer la même idée que pour la fonction *minindex* (page **??**).

<<<

```
tab = ["zero", "un", "deux"]                          # tableau à trier
pos = sorted((t, i) for i, t in enumerate(tab))   # tableau de couples
print(pos)                 # affiche [('deux', 2), ('un', 1), ('zero', 0)]
```

>>>

```
    [('deux', 2), ('un', 1), ('zero', 0)]
```

Si cette écriture est trop succincte, on peut la décomposer en :

<<<

```
tab = ["zero", "un", "deux"]
tab_position = [(t, i) for i, t in enumerate(tab)]
tab_position.sort()
print(tab_position)
```

>>>

```
    [('deux', 2), ('un', 1), ('zero', 0)]
```

---

(*entrée originale* : construction_classique.py :docstring of teachpyx.examples.construction_classique.triindex, line 6)

### 2.3.2 Constructions négatives

1. *Eviter d'effectuer le même appel deux fois* (page **??**)
2. *Modifier un dictionnaire en le parcourant* (page **??**)

---

**Eviter d'effectuer le même appel deux fois**

Dans cette fonction on calcule la variance d'une série d'observations.

```
def moyenne(serie):
    return sum(serie) / len(serie)

def variance_a_eviter(serie):
    s = 0
    for obs in serie :
        s += (obs-moyenne(serie))**2
    return s / len(serie)
```

La fonction `variance_a_eviter` appelle la fonction `moyenne` à chaque passage dans la boucle. Or, rien ne change d'un passage à l'autre. Il vaut mieux stocker le résultat dans une variable :

---

(*entrée originale* : classiques.py :docstring of teachpyx.examples.classiques.repetition_a_eviter, line 3)

---

**Modifier un dictionnaire en le parcourant**

Il faut éviter de modifier un container lorsqu'on le parcourt. Lorsqu'on supprime un élément d'un dictionnaire, la structure de celui-ci s'en trouve modifiée et affecte la boucle qui le parcourt. La boucle parcourt toujours l'ancienne structure du dictionnaire, celle qui existait au début au début de la boucle.

```
d = { k: k for k in range(10) }
for k, v in d.items():
    if k == 4 :
        del d[k]
```

En Python, cela produit l'erreur qui suit mais d'autres langages ne préviennent pas (C++) et cela aboutit à une erreur qui intervient plus tard dans le code (comme une valeur numérique inattendue).

```
Traceback (most recent call last):
File "session1.py", line 176, in <module>
    l = liste_modifie_dans_la_boucle()
File "session1.py", line 169, in liste_modifie_dans_la_boucle
    for k,v in d.items():
RuntimeError: dictionary changed size during iteration
```

Il faut pour éviter cela stocker les éléments qu'on veut modifier pour les supprimer ensuite.

```
d = { k:k for k in l }
rem = [ ]
for k,v in d.items():
    if k == 4 :
        rem.append(k)
for r in rem :
    del d[r]
```

Même si Python autorise cela pour les listes, il est conseillé de s'en abstenir ainsi que pour tout type d'objets qui en contient d'autres. C'est une habitude qui vous servira pour la plupart des autres langages.

---

(*entrée originale* : classiques.py :docstring of teachpyx.examples.classiques.dictionnaire_modifie_dans_la_boucle, line 3)

## 2.4 Dates

— *datetime* (page 84)
— *Autres formats de date* (page 86)

### 2.4.1 datetime

Le module datetime [100] fournit une classe **'datetime https ://docs.python.org/3/library/datetime.html ?highlight=datetime#datetime.datetime>'_** qui permet de faire des opérations et des comparaisons sur les dates et les heures. L'exemple suivant calcule l'âge d'une personne née le 11 août 1975.

<<<

```
import datetime
naissance = datetime.datetime(1975, 11, 8, 10, 0, 0)
jour = naissance.now()  # obtient l'heure et la date actuelle
print(jour)             # affiche quelque chose comme 2017-05-22 11:24:36.312000
age = jour - naissance  # calcule une différence
print(age)              # affiche 12614 days, 1:25:10.712000
```

100. https://docs.python.org/3/library/datetime.html?highlight=datetime#module-datetime

>>>

```
    2018-08-28 03:06:50.655666
    15633 days, 17:06:50.655666
```

L'objet datetime [101] autorise les soustractions et les comparaisons entre deux dates. Une soustraction retourne un objet de type timedelta [102] qui correspond à une durée qu'on peut multiplier par un réel ou ajouter à un objet de même type ou à un objet de type datetime [103]. L'utilisation de ce type d'objet évite de se pencher sur tous les problèmes de conversion.

Le module calendar [104] est assez pratique pour construire des calendriers. Le programme ci-dessous affiche une liste de t-uples incluant le jour et le jour de la semaine du mois d'août 1975. Dans cette liste, on y trouve le t-uple `(11,0)` qui signifie que le 11 août 1975 était un lundi. Cela permet de récupérer le jour de la semaine d'une date de naissance.

<<<

```
import calendar
c = calendar.Calendar()
for d in c.itermonthdays2(1975, 8):
    print(d)
```

>>>

```
    (0, 0)
    (0, 1)
    (0, 2)
    (0, 3)
    (1, 4)
    (2, 5)
    (3, 6)
    (4, 0)
    (5, 1)
    (6, 2)
    (7, 3)
    (8, 4)
    (9, 5)
    (10, 6)
    (11, 0)
    (12, 1)
    (13, 2)
    (14, 3)
    (15, 4)
    (16, 5)
    (17, 6)
    (18, 0)
    (19, 1)
    (20, 2)
    (21, 3)
    (22, 4)
    (23, 5)
    (24, 6)
    (25, 0)
    (26, 1)
    (27, 2)
    (28, 3)
```

(suite sur la page suivante)

101. https://docs.python.org/3/library/datetime.html?highlight=datetime#datetime.datetime
102. https://docs.python.org/3/library/datetime.html?highlight=timedelta#datetime.timedelta
103. https://docs.python.org/3/library/datetime.html?highlight=datetime#datetime.datetime
104. https://docs.python.org/3/library/calendar.html?highlight=calendar#module-calendar

(suite de la page précédente)

```
    (29, 4)
    (30, 5)
    (31, 6)
```

### 2.4.2 Autres formats de date

La date d'un événement est constamment utilisée, la date de modification d'un fichier par exemple pour détecter que celui-ci a été modifié depuis sa synchronisation avec un site web par exemple. On appelle ces dates associées à des événements des timestamp [105]. Ce sont des nombres entiers qui expriment le nombre de secondes qui se sont écoulées depuis une certaine origine qui dépend du système d'exploitation. C'est un nombre réel : la partie entière est un nombre de seconde, la partie décimale donne les millisecondes. La valeur n'a pas de sens exploitable à moins de la convertir en un format compréhensible. C'est ce que fait la fonction fromtimestamp [106].

<<<

```
import time
ts = time.time()
print(ts)

import datetime
st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
print(st)
```

>>>

```
    1535418410.7528732
    2018-08-28 03:06:50
```

## 2.5 Encoding et jeux de caractères


— *Jeux de caractères* (page 86)
— *Fichiers* (page 88)
— *Méthodes* (page 88)
— *Encodings par défaut* (page 88)


### 2.5.1 Jeux de caractères

La langue anglaise est la langue dominante en ce qui concerne l'informatique mais cela n'empêche pas que des programmes anglais manipulent du japonais même si le nombre de caractères est beaucoup plus grand. Les caractères les plus fréquent peuvent être représentés par un octet, soit au plus 256 caractères. Lorsque la langue en contient plus, il faut utiliser plusieurs octets pour un caractères. C'est à ça ce que servent les jeux de caractères : ils définissent la façon de passer d'une suite d'octets à une séquence de caractères et réciproquement. Pour être plus précis, le jeu de caractère désigne l'ensemble de caractères dont le programme a besoin, l'encodage décrit la manière dont on passe d'une séquence de caractères français, japonais, anglais à une séquence d'octets qui est la seule information manipulée par un ordinateur. Le décodage définit le passage inverse. Les langues latines n'ont besoin que d'un octet pour coder un caractère, les langues asiatiques en ont besoin de plusieurs. Il n'existe pas qu'un seul jeu de caractères lorsqu'on programme. Ils interviennent à plusieurs endroits différents :

105. https://fr.wikipedia.org/wiki/Horodatage
106. https://docs.python.org/3/library/datetime.html?highlight=fromtimestamp#datetime.datetime.fromtimestamp

— Le jeu de caractères utilisé par l'éditeur de texte pour afficher le programme.
— Le jeu de caractères du programme, par défaut ascii [107] mais il peut être changé en insérant une première ligne de commentaire (exemple : `# -*- coding: utf-8 -*-`). Les chaînes de caractères du programme sont codées avec ce jeu de caractères. Ce jeu devrait être identique à celui utilisé par l'éditeur de texte afin d'éviter les erreurs.
— Le jeu de caractères de la sortie, utilisé pour chaque instruction `print`, il est désigné par le code cp1252 [108] sur un système *Windows*.
— Le jeu de caractères dans lequel les chaînes de caractères sont manipulées. Un jeu standard qui permet de représenter toutes les langues est le jeu de caractères utf-8 [109]. Il peut être différent pour chaque variable.
— Le jeu de caractères d'un fichier texte. Il peut être différent pour chaque fichier.

Depuis la version 3 de *Python*, toutes les chaînes de caractères sont au format unicode [110]. C'est à dire que le jeu de caractères est le même pour toutes les chaînes de caractères. Pour en changer, il faut *encoder* le texte avec un jeu spécifique et la fonction encode [111]. Il deviendra une chaîne d'octets ou bytes [112]. La transformation inverse s'effectue avec la méthode decode [113].

<<<

```
st = "eé"
print(type(st))
print(len(st))

sb = st.encode("latin-1")
print(type(sb))
print(len(sb))
```

>>>

```
    <class 'str'>
    2
    <class 'bytes'>
    2
```

L'exemple précédent montre que la fonction len [114] retourne le nombre de caractères mais cela ne correspond pas au nombre d'octets que cette chaîne occupe en mémoire.

<<<

```
st = "eé"
print(type(st))
print(len(st))

sb = st.encode("utf-16")
print(type(sb))
print(len(sb))
```

>>>

```
    <class 'str'>
    2
    <class 'bytes'>
    6
```

107. https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange
108. https://fr.wikipedia.org/wiki/Windows-1252
109. https://fr.wikipedia.org/wiki/UTF-8
110. https://fr.wikipedia.org/wiki/Unicode
111. https://docs.python.org/3/library/stdtypes.html?highlight=encode#str.encode
112. https://docs.python.org/3/library/functions.html?highlight=bytes#bytes
113. https://docs.python.org/3/library/stdtypes.html?highlight=encode#bytes.decode
114. https://docs.python.org/3/library/functions.html?highlight=len#len

Un autre jeu, une autre longueur.

### 2.5.2 Fichiers

Comme *Python* utilise un jeu unique, il suffit de faire attention où on récupère les chaînes de caractères et au moment où on les transfère. C'est ce qu'on appelle les entrées sorties comme lire ou écrire un fichier. Par défaut un fichier est écrit avec le jeu de caractères du système d'exploitation. Pour un préciser un autre, il faut spécifier le paramètre *encoding*. Toutes les chaînes de caractères seront lues et converties au format *unicode* depuis ou vers l'encoding spécifié.

```
with open("essai.txt", "r", "utf-8") as f:
    text = f.read()
```

### 2.5.3 Méthodes

| | |
|---|---|
| `encode( [enc[,err]])` | Cette fonction permet de passer d'un jeu de caractères, celui de la variable, au jeu de caractères précisé par *enc* à moins que ce ne soit le jeu de caractères par défaut. Le paramètre *err* permet de préciser comment gérer les erreurs, doit-on interrompre le programme (valeur `'strict'` ou les ignorer (valeur `'ignore'`). La documentation *Python* recense toutes les valeurs possibles pour ces deux paramètres aux adresses |
| `decode([enc[, err]]` | Cette fonction est la fonction inverse de la fonction `encode`. Avec les mêmes paramètres, elle effectue la transformation inverse. |

### 2.5.4 Encodings par défaut

Le programme suivant permet d'obtenir le jeu de caractères par défaut et celui du système d'exploitation.

<<<

```
import sys
import locale
import platform
print(sys.platform)
print(platform.architecture)
print(sys.getdefaultencoding())
print(locale.getdefaultlocale())
```

>>>

```
    linux
    <function architecture at 0x7f0343f69950>
    utf-8
    ('en_US', 'UTF-8')
```

Les problèmes d'encoding surviennent parfois car on précise rarement l'encoding du programme *Python* ni le programmeur ne contrôle pas facilement celui de la sortie (`print`). Ces deux paramètres changent selon les éditeurs ou les systèmes d'exploitations.

## 2.6 FAQ

— *FAQ* (page 89)

### 2.6.1 FAQ

1. *A quoi sert un ‘‘StringIO‘‘ ?* (page **??**)
2. *Pourquoi l’installation de pandas (ou numpy) ne marche pas sous Windows avec pip ?* (page **??**)
3. *Python n’accepte pas les accents* (page **??**)
4. *Qu’est-ce qu’un type immuable ou immutable ?* (page **??**)
5. *Quel est l’entier le plus grand ?* (page **??**)
6. *Quelle est la différence entre / et // - division ?* (page **??**)
7. *Quelle est la différence entre return et print ?* (page **??**)
8. *Récupérer le nom du jour à partir d’une date* (page **??**)
9. *Récupérer le nom du mois à partir d’une date* (page **??**)
10. *Tabulations ou espace ?* (page **??**)

**A quoi sert un ‘‘StringIO‘‘ ?**

La plupart du temps, lorsqu’on récupère des données, elles sont sur le disque dur de votre ordinateur dans un fichier texte. Lorsqu’on souhaite automatiser un processur qu’on répète souvent avec ce fichier, on écrit une fonction qui prend le nom du fichier en entrée.

```
def processus_quotidien(nom_fichier) :
    # on compte les lignes
    nb = 0
    with open(nom_fichier,"r") as f :
        for line in f :
            nb += 1
    return nb
```

Et puis un jour, les données ne sont plus dans un fichier mais sur Internet. Le plus simple dans ce cas est de recopier ces données sur disque dur et d’appeler la même fonction. Simple. Un autre les données qu’on doit télécharger font plusieurs gigaoctets. Tout télécharger prend du temps pour finir pour s’apercevoir qu’elles sont corrompues. On a perdu plusieurs heures pour rien. On aurait bien voulu que la fonction `processus_quotidien` commence à traiter les données dès le début du téléchargement.

Pour cela, on a inventé la notion de **stream** ou **flux** qui sert d’interface entre la fonction qui traite les données et la source des données. Le flux lire les données depuis n’importe quel source (fichier, internet, mémoire), la fonction qui les traite n’a pas besoin d’en connaître la provenance.

StringIO [115] est un flux qui considère la mémoire comme source de données.

```
def processus_quotidien(data_stream):
    # on compte toujours les lignes
    nb = 0
    for line in data_stream :
        nb += 1
    return nb
```

La fonction `processus_quotidien` fonctionne pour des données en mémoire et sur un fichier.

115. https://docs.python.org/3/library/io.html#io.StringIO

```
fichier = __file__
f = open(fichier,"r")
nb = processus_quotidien(f)
print(nb)

text = "ligne1
```

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.stringio, line 6)

---

**Pourquoi l'installation de pandas (ou numpy) ne marche pas sous Windows avec pip ?**

Python est un langage très lent et c'est pourquoi la plupart des modules de calculs numériques incluent des parties implémentées en langage C++. numpy [116], pandas [117], matplotlib [118], scipy [119], scikit-learn [120], ...

Sous Linux, le compilateur est intégré au système et l'installation de ces modules via l'instruction `pip install <module>` met implicitement le compilateur à contribution. Sous Windows, il n'existe pas de compilateur C++ par défaut à moins de l'installer. Il faut faire attention alors d'utiliser exactement le même que celui utilisé pour compiler Python (voir Compiling Python on Windows [121]).

C'est pour cela qu'on préfère utiliser des distributions comme Anaconda [122] qui propose par défaut une version de Python accompagnée des modules les plus utilisés. Elle propose également une façon simple d'installer des modules précompilés avec l'instruction

```
conda install <module_compile>
```

L'autre option est d'utilser le site Unofficial Windows Binaries for Python Extension Packages [123] qui propose des versions compilées sous Windows d'un grand nombre de modules. Il faut télécharger le fichier *.whl* puis l'installer avec l'instruction `pip install <fichier.whl>`. La différence entre les deux ooptions tient aux environnements virtuels, voir Python virtual environments [124].

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.information_about_package, line 17)

---

**Python n'accepte pas les accents**

Le langage Python a été conçu en langage anglais. Dès qu'on on ajoute un caractère qui ne fait pas partie de l'alphabet anglais (ponctuation comprise), il déclenche une erreur :

```
File "faq_cvxopt.py", line 3
SyntaxError: Non-UTF-8 code starting with 'è' in file faq_cvxopt.py on line 4, but no␣
↪encoding declared;
    see http://python.org/dev/peps/pep-0263/ for details
```

Pour la résoudre, il faut dire à l'interpréteur que des caractères non anglais peuvent apparaître et écrire sur la première ligne du programme :

```
# -*- coding: latin-1 -*-
```

Ou pour tout caractère y compris chinois :

116. http://www.numpy.org/
117. http://pandas.pydata.org/
118. http://matplotlib.org/
119. http://www.scipy.org/
120. http://scikit-learn.org/stable/
121. https://docs.python.org/3/using/windows.html#compiling-python-on-windows
122. http://continuum.io/downloads#py34
123. http://www.lfd.uci.edu/~gohlke/pythonlibs/
124. http://astropy.readthedocs.org/en/stable/development/workflow/virtual_pythons.html

```
# -*- coding: utf-8 -*-
```

Si vous utilisez l'éditeur SciTE [125] sous Windows, après avoir ajouté cette ligne avec l'encoding *utf-8*, il est conseillé de fermer le fichier puis de le réouvrir. SciTE le traitera différemment.

**L'encodage ''utf-8'' est la norme sur Internet.** C'est pourquoi il est préférable d'utiliser celui-ci pour partager son code via une page Web.

---

(*entrée originale* : classiques.py :docstring of teachpyx.examples.classiques.commentaire_accentues, line 3)

---

Qu'est-ce qu'un type immuable ou immutable ?

Une variable de type *immuable* ne peut être modifiée. Cela concerne principalement :

— `int`, `float`, `str`, `tuple`

Si une variable est de type *immuable*, lorsqu'on effectue une opération, on créé implicitement une copie de l'objet.

Les dictionnaires et les listes sont *modifiables* (ou *mutable*). Pour une variable de ce type, lorsqu'on écrit `a = b`, `a` et `b` désigne le même objet même si ce sont deux noms différentes. C'est le même emplacement mémoire accessible paur deux moyens (deux identifiants).

Par exemple

```
a  = (2,3)
b  = a
a += (4,5)
print( a == b ) # --> False
print(a,b)      # --> (2, 3, 4, 5) (2, 3)

a  = [2,3]
b  = a
a += [4,5]
print( a == b ) # --> True
print(a,b)      # --> [2, 3, 4, 5] [2, 3, 4, 5]
```

Dans le premier cas, le type (`tuple`) est _immutable_, l'opérateur += cache implicitement une copie. Dans le second cas, le type (`list`) est _mutable_, l'opérateur += évite la copie car la variable peut être modifiée. Même si `b=a` est exécutée avant l'instruction suivante, elle n'a **pas** pour effet de conserver l'état de `a` avant l'ajout d'élément. Un autre exemple

```
a  = [1, 2]
b  = a
a [0] = -1
print(a)        # --> [-1, 2]
print(b)        # --> [-1, 2]
```

Pour copier une liste, il faut expliciter la demander

```
a  = [1, 2]
b  = list(a)
a [0] = -1
print(a)        # --> [-1, 2]
print(b)        # --> [1, 2]
```

La page Immutable Sequence Types [126] détaille un peu plus le type qui sont *mutable* et ceux qui sont *immutable*. Parmi les types standards :

— **mutable**

125. http://www.scintilla.org/SciTE.html
126. https://docs.python.org/3/library/stdtypes.html?highlight=immutable#immutable-sequence-types

  - bool [127]
  - int [128], float [129], complex [130]
  - str [131], bytes [132]
  - None [133]
  - tuple [134], frozenset [135]
- **immutable, par défaut tous les autres types dont :**
  - list [136]
  - dict [137]
  - set [138]
  - bytearray [139]

Une instance de classe est mutable. Il est possible de la rendre immutable par quelques astuces :

- __slots__ [140]
- How to Create Immutable Classes in Python [141]
- Ways to make a class immutable in Python [142]
- freeze [143]

Enfin, pour les objects qui s'imbriquent les uns dans les autres, une liste de listes, une classe qui incluent des dictionnaires et des listes, on distingue une copie simple d'une copie intégrale (**deepcopy**). Dans le cas d'une liste de listes, la copie simple recopie uniquement la première liste

```
import copy
l1 = [ [0,1], [2,3] ]
l2 = copy.copy(l1)
l1 [0][0] = '##'
print(l1,l2)         # --> [['##', 1], [2, 3]] [['##', 1], [2, 3]]

l1 [0] = [10,10]
print(l1,l2)         # --> [[10, 10], [2, 3]] [['##', 1], [2, 3]]
```

La copie intégrale recopie également les objets inclus

```
import copy
l1 = [ [0,1], [2,3] ]
l2 = copy.deepcopy(l1)
l1 [0][0] = '##'
print(l1,l2)         # --> [['##', 1], [2, 3]] [[0, 1], [2, 3]]
```

Les deux fonctions s'appliquent à tout object Python : module copy [144].

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.same_variable, line 8)

---

**Quel est l'entier le plus grand ?**

127. https://docs.python.org/3/library/functions.html#bool
128. https://docs.python.org/3/library/functions.html#int
129. https://docs.python.org/3/library/functions.html#float
130. https://docs.python.org/3/library/functions.html#complex
131. https://docs.python.org/3/library/functions.html#func-str
132. https://docs.python.org/3/library/functions.html#bytes
133. https://docs.python.org/3/library/constants.html?highlight=none#None
134. https://docs.python.org/3/library/functions.html#func-tuple
135. https://docs.python.org/3/library/functions.html#func-frozenset
136. https://docs.python.org/3/library/functions.html#func-list
137. https://docs.python.org/3/library/functions.html#func-dict
138. https://docs.python.org/3/library/functions.html#func-set
139. https://docs.python.org/3/library/functions.html#bytearray
140. https://docs.python.org/3/reference/datamodel.html?highlight=_slots__#object.__slots__
141. http://www.blog.pythonlibrary.org/2014/01/17/how-to-create-immutable-classes-in-python/
142. http://stackoverflow.com/questions/4996815/ways-to-make-a-class-immutable-in-python
143. https://freeze.readthedocs.org/en/latest/
144. https://docs.python.org/3/library/copy.html

La version 3 du langage Python a supprimé la constante `sys.maxint` qui définissait l'entier le plus grand (voir What's New In Python 3.0 [145]). De ce fait la fonction getrandbit [146] retourne un entier aussi grand que l'on veut.

```
import random,sys
x = random.getrandbits(2048)
print(type(x),x)
```

Qui affiche

```
<class 'int'>␣
↪28821592245571075131654830983838148370214474845580101472119213042190172126736565496812698627920 29.
↪..
```

Les calculs en nombre réels se font toujours avec huit octets de précision. Au delà, il faut utiliser la librairie gmpy2 [147]. Il est également recommandé d'utiliser cette librairie pour les grands nombres entiers (entre 20 et 40 chiffres). La librairie est plus rapide que l'implémentation du langage Python (voir Overview of gmpy2 [148]).

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.entier_grande_taille, line 1)

---

**Quelle est la différence entre / et // - division ?**

Le résultat de la division avec l'opérateur `/` est toujours réel : la division de deux entiers `1/2` donne `0.5`. Le résultat de la division avec l'opérateur `//` est toujours entier. Il correspond au quotient de la division.

```
div1 = 1/2
div2 = 4/2
div3 = 1//2
div4 = 1.0//2.0
print(div1,div2,div3,div4) # affiche (0.5, 2.0, 0, 0)
```

Le reste d'une division entière est obtenue avec l'opérateur `%`.

```
print ( 5 % 2 )  # affiche 1
```

C'est uniquement vrai pour les version Python 3.x. Pour les versions 2.x, les opérateurs `/` et `//` avaient des comportements différents (voir What's New In Python 3.0 [149]).

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.difference_div, line 1)

---

**Quelle est la différence entre return et print ?**

La fonction `print` sert à afficher un résultat sur la sortie standard. Elle peut être utilisée à tout moment mais elle n'a pas d'impact sur le déroulement programme. Le mot-clé `return` n'est utilisé que dans une fonction. Lorsque le programme rencontre une instruction commençant par `return`, il quitte la fonction et transmet le résultat à l'instruction qui a appelé la fonction. La fonction `print` ne modifie pas votre algorithme. La fonction `return` spécifie le résultat de votre fonction : elle modifie l'algorithme.

---

(*entrée originale* : classiques.py :docstring of teachpyx.examples.classiques.dix_entiers_carre, line 5)

---

**Récupérer le nom du jour à partir d'une date**

145. https://docs.python.org/3.1/whatsnew/3.0.html#integers
146. https://docs.python.org/3/library/random.html#random.getrandbits
147. http://gmpy2.readthedocs.org/en/latest/
148. https://gmpy2.readthedocs.org/en/latest/overview.html
149. https://docs.python.org/3/whatsnew/3.0.html#integers

```
import datetime
dt = datetime.datetime(2016, 1, 1)
print(dt.strftime("%A"))
```

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.get_day_name, line 6)

**Récupérer le nom du mois à partir d'une date**

```
import datetime
dt = datetime.datetime(2016, 1, 1)
print(dt.strftime("%B"))
```

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.get_month_name, line 6)

**Tabulations ou espace ?**

Il est préférable de ne pas utiliser les tabulations et de les remplacer par des espaces. Lorsqu'on passe d'un Editeur à un autre, les espaces ne bougent pas. Les tabulations sont plus ou moins grandes visuellement. L'essentiel est de ne pas mélanger. Dans SciTE [150], il faut aller dans le menu Options / Change Indentation Settings... Tous les éditeurs ont une option similaire.

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.entier_grande_taille, line 27)

150. http://www.scintilla.org/SciTE.html

CHAPITRE 3

# Classes et programmation objet

## 3.1 Classes

Imaginons qu'une banque détienne un fichier contenant des informations sur ses clients et qu'il soit impossible pour un client d'avoir accès directement à ces informations. Toutefois, il lui est en théorie possible de demander à son banquier quelles sont les informations le concernant détenues par sa banque. Il est en théorie également possible de les rectifier s'il estime qu'elles sont incorrectes.

On peut comparer cette banque à un objet qui possède des informations et des moyens permettant de lire ces informations et de les modifier. Vu de l'extérieur, cette banque cache son fonctionnement interne et les informations dont elle dispose, mais propose des services à ses utilisateurs.

On peut considérer la banque comme un *objet* au sens informatique. Ce terme désigne une entité possédant des données et des méthodes permettant de les manipuler. Plus concrètement, une classe est un assemblage de variables appelées *attributs* et de fonctions appelées *méthodes*. L'ensemble des propriétés associées aux classes est regroupé sous la désignation de *programmation objet*.

### 3.1.1 Présentation des classes : méthodes et attributs

#### Définition, déclaration

**Définition D1 : classe**

Une classe est un ensemble incluant des variables ou *attributs* et des fonctions ou *méthodes*. Les attributs sont des variables accessibles depuis toute méthode de la classe où elles sont définies. En *python*, les classes sont des types modifiables.

**Syntaxe S1 : Déclaration d'une classe**

```
class nom_classe :
    # corps de la classe
    # ...
```

Le corps d'une classe peut être vide, inclure des variables ou attributs, des fonctions ou méthodes. Il est en tout cas indenté de façon à indiquer à l'interpréteur *python* les lignes qui forment le corps de la classe.

Les classes sont l'unique moyen en langage *python* de définir de nouveaux types propres à celui qui programme. Il n'existe pas de type ńămatriceă" ou de type ńăgrapheă" en langage *python* qui soit prédéfini. Il est néanmoins possible de les définir au moyen des classes. Une matrice est par exemple un objet qui inclut les attributs suivant : le nombre de lignes, le nombre de colonnes, les coefficients de la matrice. Cette matrice inclut aussi des méthodes comme des opérations entre deux matrices telles que l'addition, la soustraction, la multiplication ou des opérations sur elle-même comme l'inversion, la transposition, la diagonalisation.

Cette liste n'est pas exhaustive, elle illustre ce que peut être une classe ńămatriceă" - représentation informatique d'un objet ńămatriceă" -, un type complexe incluant des informations de types variés (entier pour les dimensions, réels pour les coefficients), et des méthodes propres à cet objet, capables de manipuler ces informations.

Il est tout-à-fait possible de se passer des classes pour rédiger un programme informatique. Leur utilisation améliore néanmoins sa présentation et la compréhension qu'on peut en avoir. Bien souvent, ceux qui passent d'un langage uniquement fonctionnel à un langage objet ne font pas marche arrière. L¿**instantiation** se fait selon le schéma suivant :

**Syntaxe S2 : Instanciation d'une classe**

```
cl = nom_classe()
```

La création d'une variable de type objet est identique à celle des types standards du langage *python* : elle passe par une simple affectation. On appelle aussi `cl` une *instance* de la classe `nom_classe`.

Cette syntaxe est identique à la syntaxe d'appel d'une fonction. La création d'une instance peut également faire intervenir des paramètres (voir paragraphe *Opérateurs* (page 107)). Le terme *instance* va de paire avec le terme *classe* :

**Définition D2 : instantiaion**

Une instance d'une classe `C` désigne une variable de type `C`. Le terme instance ne s'applique qu'aux variables dont le type est une classe.

L'exemple suivant permet de définir une classe vide. Le mot-clé `pass` permet de préciser que le corps de la classe ne contient rien.

<<<

```
class classe_vide:
    pass
```

>>>

```

```

Il est tout de même possible de définir une instance de la classe `classe_vide` simplement par l'instruction suivante :

<<<

```
class classe_vide:
    pass


cl = classe_vide()
```

>>>

Dans l'exemple précédent, la variable `cl` n'est pas de type `exemple_classe` mais de type `instance` comme le montre la ligne suivante :

<<<

```
class classe_vide:
    pass


cl = classe_vide()
print(type(cl))     # affiche <type 'instance'>
```

>>>

```
    <class 'pyquickhelper.sphinxext.sphinx_runpython_extension.run_python_script_
↪139651620500536.<locals>.classe_vide'>
```

Pour savoir si une variable est une instance d'une classe donnée, il faut utiliser la fonction `isinstance` :

<<<

```
class classe_vide:
    pass


cl = classe_vide()
print(type(cl))                     # affiche <type 'instance'>
print(isinstance(cl, classe_vide))  # affiche True
```

>>>

```
    <class 'pyquickhelper.sphinxext.sphinx_runpython_extension.run_python_script_
↪139651620447648.<locals>.classe_vide'>
    True
```

### Méthodes

**Définition D3 : méthode**

Les méthodes sont des fonctions qui sont associées de manière explicite à une classe. Elles ont comme particularité un accès privilégié aux données de la classe elle-même.

Ces données ou *attributs* sont définis plus loin. Les méthodes sont en fait des fonctions pour lesquelles la liste des paramètres contient obligatoirement un paramètre explicite qui est l'instance de la classe à laquelle cette méthode est associée. Ce paramètre est le moyen d'accéder aux données de la classe.

**Syntaxe S3 : Déclaration d'une méthode**

```
class nom_classe :
    def nom_methode (self, param_1, ..., param_n) :
        # corps de la méthode...
```

A part le premier paramètre qui doit de préférence s'appeler `self`, la syntaxe de définition d'une méthode ressemble en tout point à celle d'une fonction. Le corps de la méthode est indenté par rapport à la déclaration de la méthode, elle-même indentée par rapport à la déclaration de la classe. L'appel à cette méthode obéit à la syntaxe qui suit :

**Syntaxe S4 : Appel d'une méthode**

```
cl = nom_classe()     # variable de type nom_classe
t  = cl.nom_methode (valeur_1, ..., valeur_n)
```

L'appel d'une méthode nécessite tout d'abord la création d'une variable. Une fois cette variable créée, il suffit d'ajouter le symbole ńă`.`ă" pour exécuter la méthode. Le paramètre `self` est ici implicitement remplacé par `cl` lors de l'appel.

L'exemple suivant simule le tirage de nombres aléatoires à partir d'une suite définie par récurrence $u_{n+1} = (u_n * A) mod B$ où $A$ et $B$ sont des entiers très grands. Cette suite n'est pas aléatoire mais son comportement imite celui d'une suite aléatoire. Le terme $u_n$ est dans cet exemple contenu dans la variable globale `rnd`.

<<<

```
rnd = 42


class exemple_classe:
    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        global rnd
        rnd = 397204094 * rnd % 2147483647
        return int(rnd % n)


nb = exemple_classe()
l1 = [nb.methode1(100) for i in range(0, 10)]
print(l1)   # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]

nb2 = exemple_classe()
l2 = [nb2.methode1(100) for i in range(0, 10)]
print(l2)   # affiche [46, 42, 89, 66, 48, 12, 61, 84, 71, 41]
```

>>>

```
    [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
    [46, 42, 89, 66, 48, 12, 61, 84, 71, 41]
```

Deux instances `nb` et `nb2` de la classe `exemple_classe` sont créées, chacune d'elles est utilisée pour générer aléatoirement dix nombres entiers compris entre 0 et 99 inclus. Les deux listes sont différentes puisque l'instance `nb2` utilise la variable globale `rnd` précédemment modifiée par l'appel `nb.methode1(100)`.

Les méthodes sont des fonctions insérées à l'intérieur d'une classe. La syntaxe de la déclaration d'une méthode est identique à celle d'une fonction en tenant compte du premier paramètre qui doit impérativement être `self`. Les paramètres par défaut, l'ordre des paramètres, les nombres variables de paramètres présentés au paragraphe *Fonctions* (page 53) sont des extensions tout autant applicables aux méthodes qu'aux fonctions.

### Attributs

**Définition D4 : attribut**

Les attributs sont des variables qui sont associées de manière explicite à une classe. Les attributs de la classe se comportent comme des variables globales pour toutes les méthodes de cette classe.

---

Une classe permet en quelque sorte de regrouper ensemble des informations liées. Elles n'ont de sens qu'ensemble et les méthodes manipulent ces données liées. C'est le cas pour un segment qui est toujours défini par ces deux extrémités qui ne vont pas l'une sans l'autre.

---

**Syntaxe S5 : Déclaration d'un attribut**

```
class nom_classe :
    def nom_methode (self, param_1, ..., param_n) :
        self.nom_attribut = valeur
```

---

Le paramètre `self` n'est pas un mot-clé même si le premier paramètre est le plus souvent appelé `self`. Il désigne l'instance de la classe sur laquelle va s'appliquer la méthode. La déclaration d'une méthode inclut toujours un paramètre `self` de sorte que `self.nom_attribut` désigne un attribut de la classe. `nom_attribut` seul désignerait une variable locale sans aucun rapport avec un attribut portant le même nom. Les attributs peuvent être déclarés à l'intérieur de n'importe quelle méthode, voire à l'extérieur de la classe elle-même.

L'endroit où est déclaré un attribut a peu d'importance pourvu qu'il le soit avant sa première utilisation. Dans l'exemple qui suit, la méthode `methode1` utilise l'attribut `rnd` sans qu'il ait été créé.

<<<

```
class exemple_classe:
    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
li = [nb.methode1(100) for i in range(0, 10)]
print(li)
```

>>>

```
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 13, in <module>
      File "", line 11, in run_python_script_139651620447648
      File "", line 11, in <listcomp>
      File "", line 7, in methode1
    AttributeError: 'exemple_classe' object has no attribute 'rnd'
```

Cet exemple déclenche donc une erreur (ou exception) signifiant que l'attribut `rnd` n'a pas été créé.

Pour remédier à ce problème, il existe plusieurs endroits où il est possible de créer l'attribut `rnd`. Il est possible de créer l'attribut à l'intérieur de la méthode `methode1`. Mais le programme n'a plus le même sens puisqu'à chaque appel de la méthode `methode1`, l'attribut `rnd` reçoit la valeur 42. La liste de nombres aléatoires contient dix fois la même valeur.

<<<

```
class exemple_classe:
    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 42  # déclaration à l'intérieur de la méthode,
        # doit être précédé du mot-clé self
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
li = [nb.methode1(100) for i in range(0, 10)]
print(li)  # affiche [19, 19, 19, 19, 19, 19, 19, 19, 19, 19]
```

>>>

```
    [19, 19, 19, 19, 19, 19, 19, 19, 19, 19]
```

Il est possible de créer l'attribut `rnd` à l'extérieur de la classe. Cette écriture devrait toutefois être évitée puisque la méthode `methode1` ne peut pas être appelée sans que l'attribut `rnd` ait été ajouté.

<<<

```
class exemple_classe:
    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
           compris entre 0 et n-1 inclus"""
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
nb.rnd = 42              # déclaration à l'extérieur de la classe,
# indispensable pour utiliser la méthode methode1
li = [nb.methode1(100) for i in range(0, 10)]
print(li)  # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

>>>

```
    [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

Ceux qui découvrent la programmation se posent toujours la question de l'utilité de ce nouveau concept qui ne permet pas de faire des choses différentes, tout au plus de les faire mieux. La finalité des classes apparaît avec le concept d¿*Héritage* (page 134). L'article illustre une façon de passer progressivent des fonctions aux classes de fonctions : *C'est obligé les classes ?* (page **??**).

### 3.1.2 Constructeur

L'endroit le plus approprié pour déclarer un attribut est à l'intérieur d'une méthode appelée le *constructeur*. S'il est défini, il est implicitement exécuté lors de la création de chaque instance. Le constructeur d'une classe se présente comme une méthode et suit la même syntaxe à ceci près que son nom est imposé : `__init__`. Hormis le premier paramètre, invariablement `self`, il n'existe pas de contrainte concernant la liste des paramètres excepté que le constructeur ne doit pas retourner de résultat.

**Syntaxe S6 : Déclaration d'un constructeur**

```
class nom_classe :
    def __init__(self, param_1, ..., param_n):
        # code du constructeur
```

nom_classe est une classe, __init__ est son constructeur, sa syntaxe est la même que celle d'une méthode sauf que le constructeur ne peut employer l'instruction return. La modification des paramètres du constructeur implique également la modification de la syntaxe de création d'une instance de cette classe.

**Syntaxe S7 : Appel d'un constructeur**

```
x = nom_classe (valeur_1,...,valeur_n)
```

nom_classe est une classe, valeur_1 à valeur_n sont les valeurs associées aux paramètres param_1 à param_n du constructeur.

L'exemple suivant montre deux classes pour lesquelles un constructeur a été défini. La première n'ajoute aucun paramètre, la création d'une instance ne nécessite pas de paramètre supplémentaire. La seconde classe ajoute deux paramètres a et b. Lors de la création d'une instance de la classe classe2, il faut ajouter deux valeurs.

<<<

```
class classe1:
    def __init__(self):
        # pas de paramètre supplémentaire
        print("constructeur de la classe classe1")
        self.n = 1  # ajout de l'attribut n


x = classe1()       # affiche constructeur de la classe classe1
print(x.n)          # affiche 1


class classe2:
    def __init__(self, a, b):
        # deux paramètres supplémentaires
        print("constructeur de la classe classe2")
        self.n = (a+b)/2  # ajout de l'attribut n


x = classe2(5, 9)  # affiche constructeur de la classe classe2
print(x.n)          # affiche 7
```

>>>

```
    constructeur de la classe classe1
    1
    constructeur de la classe classe2
    7.0
```

Le constructeur autorise autant de paramètres qu'on souhaite lors de la création d'une instance et celle-ci suit la même syntaxe qu'une fonction. La création d'une instance pourrait être considérée comme l'appel à une fonction à ceci près que le type du résultat est une instance de classe.

En utilisant un constructeur, l'exemple du paragraphe précédent simulant une suite de variable aléatoire permet d'obtenir une classe autonome qui ne fait pas appel à une variable globale ni à une déclaration d'attribut extérieur à la classe.

<<<

```
class exemple_classe:
    def __init__(self):  # constructeur
        self.rnd = 42     # on crée l'attribut rnd, identique pour chaque instance
        # --> les suites générées auront toutes le même début

    def methode1(self, n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
l1 = [nb.methode1(100) for i in range(0, 10)]
print(l1)   # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]

nb2 = exemple_classe()
l2 = [nb2.methode1(100) for i in range(0, 10)]
print(l2)   # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

>>>

```
    [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
    [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

De la même manière qu'il existe un constructeur exécuté à chaque création d'instance, il existe un destructeur exécuté à chaque destruction d'instance. Il suffit pour cela de redéfinir la méthode `__del__`. A l'inverse d'autres langages comme le *C++*, cet opérateur est peu utilisé car le *python* nettoie automatiquement les objets qui ne sont plus utilisés ou plus référencés par une variable.

### 3.1.3 Apport du langage python

#### Liste des attributs

Chaque attribut d'une instance de classe est inséré dans un dictionnaire appelé __dict__ [151], attribut implicitement présent dès la création d'une instance.

<<<

```
class exemple_classe:
    def __init__(self):
        self.rnd = 42

    def methode1(self, n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
print(nb.__dict__)      # affiche {'rnd': 42}
```

>>>

```
    {'rnd': 42}
```

Ce dictionnaire offre aussi la possibilité de tester si un attribut existe ou non. Dans un des exemples du paragraphe précédent, l'attribut `rnd` était créé dans la méthode `methode1`, sa valeur était alors initialisée à chaque appel et la

151. https://docs.python.org/3/library/stdtypes.html?highlight=__dict__#object.__dict__

fonction retournait sans cesse la même valeur. En testant l'existence de l'attribut `rnd`, il est possible de le créer dans la méthode `methode1` au premier appel sans que les appels suivants ne réinitialisent sa valeur à 42.

<<<

```
class exemple_classe:
    def methode1(self, n):
        if "rnd" not in self.__dict__:  # l'attribut existe-t-il ?
            self.rnd = 42                # création de l'attribut
            self.__dict__["rnd"] = 42   # autre écriture possible
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
li = [nb.methode1(100) for i in range(0, 10)]
print(li)  # affiche [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

>>>

```
    [19, 46, 26, 88, 44, 56, 56, 26, 0, 8]
```

### Attributs implicites

Certains attributs sont créés de manière implicite lors de la création d'une instance. Ils contiennent des informations sur l'instance.

| | |
|---|---|
| `__module__` | Contient le nom du module dans lequel est incluse la classe (voir chapitre *Modules* (page 235)). |
| `__class__` | Contient le nom de la classe de l'instance. Ce nom est précédé du nom du module suivi d'un point. |
| `__dict__` | Contient la liste des attributs de l'instance (voir paragraphe *Liste des attributs* (page 102). |
| `__doc__` | Contient un commentaire associé à la classe (voir paragraphe *Commentaires, aide* (page 104). |

L'attribut `__class__` contient lui même d'autres d'attributs :

| | |
|---|---|
| `__doc__` | Contient un commentaire associé à la classe (voir paragraphe *Commentaires, aide* (page 104). |
| `__dict__` | Contient la liste des attributs statiques (définis hors d'une méthode) et des méthodes (voir paragraphe *Attributs statiques* (page 119). |
| `__name__` | Contient le nom de l'instance. |
| `__bases__` | Contient les classes dont la classe de l'instance hérite (voir paragraphe *Héritage* (page 134). |

<<<

```
class classe_vide:
    pass


cl = classe_vide()
print(cl.__module__)              # affiche __main__
print(cl.__class__)               # affiche __main__.classe_vide ()
```

(suite sur la page suivante)

(suite de la page précédente)

```
print(cl.__dict__)                # affiche {}
print(cl.__doc__)                 # affiche None  (voir paragraphe suivant)
print(cl.__class__.__doc__)       # affiche None
print(cl.__class__.__dict__)      # affiche {'__module__': '__main__',
#           '__doc__': None}
print(cl.__class__.__name__)      # affiche classe_vide
print(cl.__class__.__bases__)     # affiche ()
```

>>>

```
    pyquickhelper.sphinxext.sphinx_runpython_extension
    <class 'pyquickhelper.sphinxext.sphinx_runpython_extension.run_python_script_
↪139651621866448.<locals>.classe_vide'>
    {}
    None
    None
    {'__module__': 'pyquickhelper.sphinxext.sphinx_runpython_extension', '__dict__':
↪<attribute '__dict__' of 'classe_vide' objects>, '__weakref__': <attribute '__
↪weakref__' of 'classe_vide' objects>, '__doc__': None}
    classe_vide
    (<class 'object'>,)
```

### Commentaires, aide

Comme les fonctions et les méthodes, des commentaires peuvent être associés à une classe, ils sont affichés grâce à la fonction `help`. Cette dernière présente le commentaire associé à la classe, la liste des méthodes ainsi que chacun des commentaires qui leur sont associés. Ce commentaire est affecté à l'attribut implicite `__doc__`. L'appel à la fonction `help` rassemble le commentaire de toutes les méthodes, le résultat suit le programme ci-dessous.

<<<

```
class exemple_classe:
    """simule une suite de nombres aléatoires"""

    def __init__(self):
        """constructeur : initialisation de la première valeur"""
        self.rnd = 42

    def methode1(self, n):
        """simule la génération d'un nombre aléatoire
        compris entre 0 et n-1 inclus"""
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
help(exemple_classe)      # appelle l'aide associée à la classe
```

>>>

```
    Help on class exemple_classe in module pyquickhelper.sphinxext.sphinx_runpython_
↪extension:

    class exemple_classe(builtins.object)
     |  simule une suite de nombres aléatoires
     |
```

(suite sur la page suivante)

(suite de la page précédente)

```
     |  Methods defined here:
     |
     |  __init__(self)
     |      constructeur : initialisation de la première valeur
     |
     |  methode1(self, n)
     |      simule la génération d'un nombre aléatoire
     |      compris entre 0 et n-1 inclus
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
```

Pour obtenir seulement le commentaire associé à la classe, il suffit d'écrire l'une des trois lignes suivantes :

<<<

```
class exemple_classe:
    """simule une suite de nombres aléatoires"""
    pass


nb = exemple_classe()

print(exemple_classe.__doc__)   # affiche simule une suite de nombres aléatoires
print(nb.__doc__)               # affiche simule une suite de nombres aléatoires
print(nb.__class__.__doc__)     # affiche simule une suite de nombres aléatoires
```

>>>

```
    simule une suite de nombres aléatoires
    simule une suite de nombres aléatoires
    simule une suite de nombres aléatoires
```

La fonction `help` permet d'accéder à l'aide associée à une fonction, une classe. Il existe des outils qui permettent de collecter tous ces commentaires pour construire une documentation au format *HTML* à l'aide d'outils comme pydoc [152] Ces outils sont souvent assez simples d'utilisation. Le plus utilisé est sphinx [153].

La fonction dir [154] permet aussi d'obtenir des informations sur la classe. Cette fonction appliquée à la classe ou à une instance retourne l'ensemble de la liste des attributs et des méthodes. L'exemple suivant utilise la fonction dir [155] avant et après l'appel de la méthode `meth`. Etant donné que cette méthode ajoute un attribut, la fonction dir [156] retourne une liste plus longue après l'appel.

<<<

```
class essai_class:
    def meth(self):
        x = 6
```

(suite sur la page suivante)

152. https://docs.python.org/3/library/pydoc.html
153. http://www.sphinx-doc.org/en/
154. https://docs.python.org/3/library/functions.html?highlight=dir#dir
155. https://docs.python.org/3/library/functions.html?highlight=dir#dir
156. https://docs.python.org/3/library/functions.html?highlight=dir#dir

(suite de la page précédente)

```
        self.y = 7


a = essai_class()
print(dir(a))              # affiche ['__doc__', '__module__', 'meth']
a.meth()
print(dir(a))              # affiche ['__doc__', '__module__', 'meth', 'y']
print(dir(essai_class))    # affiche ['__doc__', '__module__', 'meth']
```

>>>

```
    ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__
↪format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_
↪subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '_
↪_reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__
↪', '__weakref__', 'meth']
    ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__
↪format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_
↪subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '_
↪_reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__
↪', '__weakref__', 'meth', 'y']
    ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__
↪format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_
↪subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '_
↪_reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__
↪', '__weakref__', 'meth']
```

La fonction dir [157] appliquée à la classe elle-même retourne une liste qui inclut les méthodes et les attributs déjà déclarés. Elle n'inclut pas ceux qui sont déclarés dans une méthode jamais exécutée jusqu'à présent.

### Classe incluse

Parfois, il arrive qu'une classe soit exclusivement utilisée en couple avec une autre, c'est par exemple le cas des itérateurs (voir paragraphe *Itérateurs* (page 113). Il est alors possible d'inclure dans la déclaration d'une classe celle d'une sous-classe.

L'exemple qui suit contient la classe `ensemble_element`. C'est un ensemble de points en trois dimensions (classe `element`) qui n'est utilisé que par cette classe. Déclarer la classe `element` à l'intérieur de la classe `ensemble_element` est un moyen de signifier ce lien.

<<<

```
class ensemble_element:

    class element:
        def __init__(self):
            self.x, self.y, self.z = 0, 0, 0

    def __init__(self):
        self.all = [ensemble_element.element() for i in range(0, 3)]

    def barycentre(self):
        b = ensemble_element.element()
        for el in self.all:
            b.x += el.x
```

(suite sur la page suivante)

157. https://docs.python.org/3/library/functions.html?highlight=dir#dir

(suite de la page précédente)

```
            b.y += el.y
            b.z += el.z
        b.x /= len(self.all)
        b.y /= len(self.all)
        b.z /= len(self.all)
        return b


f = ensemble_element()
f.all[0].x, f.all[0].y, f.all[0].z = 4.5, 1.5, 1.5
b = f.barycentre()
print(b.x, b.y, b.z)  # affiche 1.5 0.5 0.5
```

>>>

```
    1.5 0.5 0.5
```

Pour créer une instance de la classe `element`, il faut faire précéder son nom de la classe où elle est déclarée : `b = ensemble_element.element()` comme c'est le cas dans la méthode `barycentre` par exemple.

### 3.1.4 Opérateurs

Les opérateurs sont des symboles du langages comme +, −, +=, ... Au travers des opérateurs, il est possible de donner un sens à une syntaxe comme celle de l'exemple suivant :

<<<

```
class nouvelle_classe:
    pass


x = nouvelle_classe() + nouvelle_classe()
```

>>>

```
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 6, in <module>
      File "", line 5, in run_python_script_139651623048968
    TypeError: unsupported operand type(s) for +: 'nouvelle_classe' and 'nouvelle_
↪classe'
```

L'addition n'est pas le seul symbole concerné, le langage *python* permet de donner un sens à tous les opérateurs numériques et d'autres reliés à des fonctions du langage comme `len` ou `max`. Le programme suivant contient une classe définissant un nombre complexe. La méthode `ajoute` définit ce qu'est une addition entre nombres complexes.

<<<

```
import math


class nombre_complexe:
    def __init__(self, a=0, b=0):
        self.a, self.b = a, b
```

(suite sur la page suivante)

(suite de la page précédente)

```
    def get_module(self):
        return math.sqrt(self.a * self.a + self.b * self.b)

    def ajoute(self, c):
        return nombre_complexe(self.a + c.a, self.b + c.b)


c1 = nombre_complexe(0, 1)
c2 = nombre_complexe(1, 0)

c = c1.ajoute(c2)          # c = c1 + c2
print(c.a, c.b)
```

>>>

```
    1 1
```

Toutefois, on aimerait bien écrire simplement `c = c1 + c2` au lieu de `c = c1.ajoute(c2)` car cette syntaxe est plus facile à lire et surtout plus intuitive. Le langage *python* offre cette possibilité. Il existe en effet des méthodes *clés* dont l'implémentation définit ce qui doit être fait dans le cas d'une addition, d'une comparaison, d'un affichage, ... A l'instar du constructeur, toutes ces méthodes clés, qu'on appelle des *opérateurs*, sont encadrées par deux blancs soulignés, leur déclaration suit invariablement le même schéma. Voici celui de l'opérateur `__add__` qui décrit ce qu'il faut faire pour une addition.

<<<

```
class nom_class:
    def __add__(self, autre):
        # corps de l'opérateur
        return ...   # nom_classe
```

>>>

```

```

`nom_classe` est une classe. L'opérateur `__add__` définit l'addition entre l'instance `self` et l'instance `autre` et retourne une instance de la classe `nom_classe`.

Le programme suivant reprend le précédent de manière à ce que l'addition de deux nombres complexes soit dorénavant une syntaxe correcte.

<<<

```
import math


class nombre_complexe:
    def __init__(self, a=0, b=0):
        self.a, self.b = a, b

    def get_module(self):
        return math.sqrt(self.a * self.a + self.b * self.b)

    def __add__(self, c):
        return nombre_complexe(self.a + c.a, self.b + c.b)
```

(suite sur la page suivante)

(suite de la page précédente)

```
c1 = nombre_complexe(0, 1)
c2 = nombre_complexe(1, 0)
c = c1 + c2           # cette expression est maintenant syntaxiquement correcte
c = c1.__add__(c2)   # même ligne que la précédente mais écrite explicitement
print(c.a, c.b)
```

>>>

```
    1 1
```

L'avant dernière ligne appelant la méthode __add__ transcrit de façon explicite ce que le langage *python* fait lorsqu'il rencontre un opérateur + qui s'applique à des classes. Plus précisément, `c1` et `c2` pourraient être de classes différentes, l'expression serait encore valide du moment que la classe dont dépend `c1` a redéfini la méthode __add__. Chaque opérateur possède sa méthode-clé associée. L'opérateur +=, différent de + est associé à la méthode-clé __iadd__.

```
class nom_class :
    def __iadd__ (self, autre) :
        # corps de l'opérateur
        return self
```

`nom_classe` est une classe. L'opérateur __iadd__ définit l'addition entre l'instance `self` et l'instance `autre`. L'instance `self` est modifiée pour recevoir le résultat. L'opérateur retourne invariablement l'instance modifiée `self`. On étoffe la classe `nombre_complexe` à l'aide de l'opérateur __iadd__.

<<<

```
import math


class nombre_complexe:
    def __init__(self, a=0, b=0):
        self.a, self.b = a, b

    def get_module(self):
        return math.sqrt(self.a * self.a + self.b * self.b)

    def __add__(self, c):
        return nombre_complexe(self.a + c.a, self.b + c.b)

    def __iadd__(self, c):
        self.a += c.a
        self.b += c.b
        return self


c1 = nombre_complexe(0, 1)
c2 = nombre_complexe(1, 0)
c1 += c2           # utilisation de l'opérateur +=
c1.__iadd__(c2)    # c'est la transcription explicite de la ligne précédente
print(c1.a, c1.b)
```

>>>

```
    2 1
```

Un autre opérateur souvent utilisé est __str__ qui permet de redéfinir l'affichage d'un objet lors d'un appel à l'instruction `print`.

**Syntaxe S8 : Déclaration de l'opérateur __str__**

```
class nom_class :
    def __str__ (self) :
        # corps de l'opérateur
        return...
```

nom_classe est une classe. L'opérateur __str__ construit une chaîne de caractères qu'il retourne comme résultat de façon à être affiché. L'exemple suivant reprend la classe nombre_complexe pour que l'instruction print affiche un nombre complexe sous la forme $a + ib$.

<<<

```
class nombre_complexe:
    def __init__(self, a=0, b=0):
        self.a, self.b = a, b

    def __add__(self, c):
        return nombre_complexe(self.a + c.a, self.b + c.b)

    def __str__(self):
        if self.b == 0:
            return "%f" % (self.a)
        elif self.b > 0:
            return "%f + %f i" % (self.a, self.b)
        else:
            return "%f - %f i" % (self.a, -self.b)


c1 = nombre_complexe(0, 1)
c2 = nombre_complexe(1, 0)
c3 = c1 + c2
print(c3)       # affiche 1.000000 + 1.000000 i
```

>>>

```
    1.000000 + 1.000000 i
```

Il existe de nombreux opérateurs qu'il est possible de définir. La table operateur_classe présente les plus utilisés. Parmi ceux-là, on peut s'attarder sur les opérateurs __getitem__ et __setitem__, ils redéfinissent l'opérateur [] permettant d'accéder à un élément d'une liste ou d'un dictionnaire. Le premier, __getitem__ est utilisé lors d'un calcul, un affichage. Le second, __setitem__, est utilisé pour affecter une valeur.

L'exemple suivant définit un point de l'espace avec trois coordonnées. Il redéfinit ou *surcharge* les opérateurs __getitem__ et __setitem__ de manière à pouvoir accéder aux coordonnées de la classe point_espace qui définit un point dans l'espace. En règle générale, lorsque les indices ne sont pas corrects, ces deux opérateurs lèvent l'exception IndexError (voir le chapitre *Exceptions* (page 155).

<<<

```
class point_espace:
    def __init__(self, x, y, z):
        self._x, self._y, self._z = x, y, z

    def __getitem__(self, i):
        if i == 0:
            return self._x
```

(suite sur la page suivante)

(suite de la page précédente)

```
        if i == 1:
            return self._y
        if i == 2:
            return self._z
        # pour tous les autres cas --> erreur
        raise IndexError("indice impossible, 0,1,2 autorisés")

    def __setitem__(self, i, x):
        if i == 0:
            self._x = x
        elif i == 1:
            self._y = y
        elif i == 2:
            self._z = z
        # pour tous les autres cas --> erreur
        raise IndexError("indice impossible, 0,1,2 autorisés")

    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)


a = point_espace(1, -2, 3)

print(a)                      # affiche (1.000000,-2.000000,3.000000)
a[1] = -3                    # (__setitem__) affecte -3 à a.y
print("abscisse : ", a[0])   # (__getitem__) affiche abscisse :  1
print("ordonnée : ", a[1])   # (__getitem__) affiche ordonnée :  -3
print("altitude : ", a[2])   # (__getitem__) affiche altitude :  3
```

>>>

```
    (1.000000,-2.000000,3.000000)
    [runpythonerror]

    Traceback (most recent call last):
        exec(obj, globals(), loc)
      File "", line 31, in <module>
      File "", line 27, in run_python_script_139651622809744
      File "", line 16, in __setitem__
    NameError: name 'y' is not defined
```

Par le biais de l'exception `IndexError`, les expressions `a[i]` avec `i != 0,1,2` sont impossibles et arrêtent le programme par un message comme celui qui suit obtenu après l'interprétation de `print(a[4])` :

| | |
|---|---|
| `__cmp__(self,x)` | Retourne un entier égale à -1, 0, 1, chacune de ces valeurs étant associés respectivement à : `self < x`, `self == x`, `self > x`. Cet opérateur est appelé par la fonction `cmp`. |
| `__str__(self)` | Convertit un objet en une chaîne de caractère qui sera affichée par la fonction `print` ou obtenu avec la fonction `str`. |
| `__contains__(self,x)` | Retourne `True` ou `False` selon que `x` appartient à `self`. Le mot-clé `in` renvoie à cet opérateur. En d'autres termes, `if x in obj:` appelle `obj.__contains__(x)`. |
| `__len__(self)` | Retourne le nombre d'élément de `self`. Cet opérateur est appelé par la fonction `len`. |
| `__abs__(self)` | Cet opérateur est appelé par la fonction `abs`. |
| `__getitem__(self,i)` | Cet opérateur est appelé lorsqu'on cherche à accéder à un élément de l'objet `self` d'indice `i` comme si c'était une liste. Si l'indice `i` est incorrect, l'exception `IndexError` doit être levée. |
| `__setitem__(self,i,v)` | Cet opérateur est appelé lorsqu'on cherche à affecter une valeur `v` à un élément de l'objet `self` d'indice `i` comme si c'était une liste ou un dictionnaire. Si l'indice `i` est incorrect, l'exception `IndexError`. |
| `__delitem__(self,i)` | Cet opérateur est appelé lorsqu'on cherche à supprimer l'élément de l'objet `self` d'indice `i` comme si c'était une liste ou un dictionnaire. Si l'indice `i` est incorrect, l'exception `IndexError` doit être levée. |
| `__int__(self)`, `__float__(self)`, `__complex__(self)` | Ces opérateurs implémente la conversion de l'instance `self` en entier, réel ou complexe. |
| `__add__(self,x)`, `__div__(self,x)`, `__mul__(self,x)` `__sub__(self,x)`, `__pow__(self,x)`, `__lshift__(self, x)`, `__rshift__(self, x)` | Opérateurs appelés pour les opérations +, /, *, -, **, <, < |
| **`__iadd__(self,x)`, `__idiv__(self,x)`, `__imul__(self,x)`,** `__isub__(self,x)`, `__ipow__(self,x)`, `__ilshift__(self, x)` `__irshift__(self, x)` | Opérateurs appelés pour les opérations +=, /=, *=, -=, **=, <<=, >>= |

La liste complète est accessible à Operators [158]. Le langage Python [159] autorise une opération peu commune aux autres langages : des opérateurs retournés [160]. Cela permet de donner un sens à une expression du type `4 + instance d'un objet`. Le type entier ne définit pas cette opération et elle devrait normalement échoué. Comme elle n'exsite pas, il est possible de définir un opérateur retourné qui prend le relais dans ce cas. Il est recommandé d'en faire un usage modéré car c'est quelque chose peu répandu dans les langages de programmation.

<<<

```
class RightSide:

    def __init__(self, v):
        self.v = v
```

(suite sur la page suivante)

158. https://docs.python.org/3/library/operator.html
159. http://www.python.org/
160. https://docs.python.org/3/reference/datamodel.html#object.__radd__

(suite de la page précédente)

```
    def __str__(self):
        return "RS({})".format(self.v)

    def __add__(self, v):
        return RightSide('9999999999')


class LeftSide:

    def __init__(self, v):
        self.v = v

    def __str__(self):
        return "LS({})".format(self.v)

    def __add__(self, o):
        return LeftSide(self.v + o)

    def __radd__(self, o):
        return RightSide(self.v + o)

    def __lshift__(self, o):
        return self.__add__(o)

    def __rlshift__(self, o):
        return self.__radd__(o)


print(LeftSide(3) + 4)
print(4 + LeftSide(3))
print('---')
print(LeftSide(3) << 4)
print(4 << LeftSide(3))
print('---')
print(RightSide(4) + LeftSide(3))
```

>>>

```
    LS(7)
    RS(7)
    ---
    LS(7)
    RS(7)
    ---
    RS(9999999999)
```

### 3.1.5 Itérateurs

L'opérateur `__iter__` permet de définir ce qu'on appelle un itérateur. C'est un objet qui permet d'en explorer un autre, comme une liste ou un dictionnaire. Un itérateur est un objet qui désigne un élément d'un ensemble à parcourir et qui connaît l'élément suivant à visiter. Il doit pour cela contenir une référence à l'objet qu'il doit explorer et inclure une méthode `__next__` qui retourne l'élément suivant ou lève une exception si l'élément actuel est le dernier.

Par exemple, on cherche à explorer tous les éléments d'un objet de type `point_espace` défini au paragraphe précédent. Cette exploration doit s'effectuer au moyen d'une boucle `for`.

<<<

```
class point_espace:
    def __init__(self, x, y, z):
        self._x, self._y, self._z = x, y, z

    def __iter__(self):
        yield self._x
        yield self._y
        yield self._z


a = point_espace(1, -2, 3)

for x in a:
    print(x)       # affiche successivement 1,-2,3
```

>>>

```
    1
    -2
    3
```

Cette boucle cache en fait l'utilisation d'un itérateur qui apparaît explicitement dans l'exemple suivant équivalent au précédent (voir paragraphe *Itérateurs* (page 46)).

<<<

```
class point_espace:
    def __init__(self, x, y, z):
        self._x, self._y, self._z = x, y, z

    def __iter__(self):
        yield self._x
        yield self._y
        yield self._z


a = point_espace(1, -2, 3)
it = iter(a)
while True:
    try:
        print(next(it))
    except StopIteration:
        break
```

>>>

```
    1
    -2
    3
```

Afin que cet extrait de programme fonctionne, il faut définir un itérateur pour la classe `point_espace`. Cet itérateur doit inclure la méthode `__next__`. La classe `point_espace` doit quant à elle définir l'opérateur `__iter__` pour retourner l'itérateur qui permettra de l'explorer.

<<<

```
class point_espace:
    def __init__(self, x, y, z):
        self._x, self._y, self._z = x, y, z

    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)

    def __getitem__(self, i):
        if i == 0:
            return self._x
        if i == 1:
            return self._y
        if i == 2:
            return self._z
        # pour tous les autres cas --> erreur
        raise IndexError("indice impossible, 0,1,2 autorisés")

    class class_iter:
        """cette classe définit un itérateur pour point_espace"""

        def __init__(self, ins):
            """initialisation, self._ins permet de savoir quelle
               instance de point_espace on explore,
               self._n mémorise l'indice de l'élément exploré"""
            self._n = 0
            self._ins = ins

        def __iter__(self):   # le langage impose cette méthode
            return self           # dans certaines configurations

        def __next__(self):
            """retourne l'élément d'indice self._n et passe à l'élément suivant"""
            if self._n <= 2:
                v = self._ins[self._n]
                self._n += 1
                return v
            else:
                # si cet élément n'existe pas, lève une exception
                raise StopIteration

    def __iter__(self):
        """opérateur de la classe point_espace, retourne un itérateur
           permettant de l'explorer"""
        return point_espace.class_iter(self)


a = point_espace(1, -2, 3)
for x in a:
    print(x)       # affiche successivement 1,-2,3
```

>>>

```
    1
    -2
    3
```

Cette syntaxe peut paraître fastidieuse mais elle montre de manière explicite le fonctionnement des itérateurs. Cette construction est plus proche de ce que d'autres langages objets proposent. *python* offre néanmoins une syntaxe plus

courte avec le mot-clé `yield` qui permet d'éviter la création de la classe `class_iter`. Le code de la méthode `__iter__` change mais les dernières lignes du programme précédent qui affichent successivement les éléments de `point_espace` sont toujours valides.

<<<

```
class point_espace:
    def __init__(self, x, y, z):
        self._x, self._y, self._z = x, y, z

    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)

    def __getitem__(self, i):
        if i == 0:
            return self._x
        if i == 1:
            return self._y
        if i == 2:
            return self._z
        # pour tous les autres cas --> erreur
        raise IndexError("indice impossible, 0,1,2 autorisés")

    def __iter__(self):
        """itérateur avec yield (ou générateur)"""
        _n = 0
        while _n <= 2:
            yield self.__getitem__(_n)
            _n += 1


a = point_espace(1, -2, 3)
for x in a:
    print(x)       # affiche successivement 1,-2,3
```

>>>

```
    1
    -2
    3
```

### 3.1.6 Méthodes, attributs statiques et ajout de méthodes

#### Méthode statique

---

**Définition D5 : méthode statique**

Les méthodes statiques sont des méthodes qui peuvent être appelées même si aucune instance de la classe où elles sont définies n'a été créée.

---

L'exemple suivant définit une classe avec une seule méthode. Comme toutes les méthodes présentées jusqu'à présent, elle inclut le paramètre `self` qui correspond à l'instance pour laquelle elle est appelée.

<<<

```
class essai_class:
    def methode(self):
        print("méthode non statique")


x = essai_class()
x.methode()
```

>>>

```
    méthode non statique
```

Une méthode statique ne nécessite pas qu'une instance soit créée pour être appelée. C'est donc une méthode n'ayant pas besoin du paramètre `self`.

---

**Syntaxe S9 : Déclaration d'une méthode statique**

```
class nom_class :
    @staticmethod
    def nom_methode(params, ...) :
        # corps de la méthode
        ...
```

---

`nom_classe` est une classe, `nom_methode` est une méthode statique. Il faut pourtant ajouter la ligne suivante pour indiquer à la classe que cette méthode est bien statique à l'aide du mot-clé staticmethod [161]. Le programme précédent est modifié pour inclure une méthode statique. La méthode `methode` ne nécessite aucune création d'instance pour être appelée.

<<<

```
class essai_class:
    @staticmethod
    def methode():
        print("méthode statique")


essai_class.methode()
```

>>>

```
    méthode statique
```

Il est également possible de déclarer une fonction statique à l'extérieur d'une classe puis de l'ajouter en tant que méthode statique à cette classe. Le programme suivant déclare une fonction `methode` puis indique à la classe `essai_class` que la fonction est aussi une méthode statique de sa classe (avant-dernière ligne de l'exemple).

<<<

```
def methode():
    print("méthode statique")


class essai_class:
    pass
```

(suite sur la page suivante)

161. https://docs.python.org/3/library/functions.html?highlight=staticmethod#staticmethod

(suite de la page précédente)

```
essai_class.methode = staticmethod(methode)
essai_class.methode()
```

>>>

```
    méthode statique
```

Toutefois, il est conseillé de placer l'instruction qui contient `staticmethod` à l'intérieur de la classe. Elle n'y sera exécutée qu'une seule fois comme le montre l'exemple suivant :

<<<

```
def methode():
    print("méthode statique")


class classe_vide:
    print("création d'une instance de la classe classe_vide")
    methode = staticmethod(methode)


cl = classe_vide()      # affiche création d'une instance de la classe essai_class
ck = classe_vide()      # n'affiche rien
```

>>>

```
    création d'une instance de la classe classe_vide
```

Les méthodes statiques sont souvent employées pour créer des instances spécifiques d'une classe.

<<<

```
class Couleur:
    def __init__(self, r, v, b):
        self.r, self.v, self.b = r, v, b

    def __str__(self):
        return str((self.r, self.v, self.b))

    @staticmethod
    def blanc():
        return Couleur(255, 255, 255)

    @staticmethod
    def noir():
        return Couleur(0, 0, 0)


c = Couleur.blanc()
print(c)                # affiche (255, 255, 255)
c = Couleur.noir()
print(c)                # affiche (0, 0, 0)
```

>>>

```
    (255, 255, 255)
    (0, 0, 0)
```

Les méthodes sont des fonctions spécifiques à une classe sans être spécifique à une isntance.

### Attributs statiques

**Définition D6 : attribut statique**

Les attributs statiques sont des attributs qui peuvent être utilisés même si aucune instance de la classe où ils sont définis n'a été créée. Ces attributs sont partagés par toutes les instances.

**Syntaxe S10 : Déclaration d'un attribut statique**

```
class nom_class :
    attribut_statique = valeur
    def nom_methode (self,params, ...):
        nom_class.attribut_statique2 = valeur2
    @staticmethod
    def nom_methode_st (params, ...) :
        nom_class.attribut_statique3 = valeur3
```

`nom_classe` est une classe, `nom_methode` est une méthode non statique, `nom_methode_st` est une méthode statique. Les trois paramètres `attribut_statique`, `attribut_statique2`, `attribut_statique3` sont statiques, soit parce qu'ils sont déclarés en dehors d'une méthode, soit parce que leur déclaration fait intervenir le nom de la classe.

Pour le programme suivant, la méthode `meth` n'utilise pas `self.x` mais `essai_class.x`. L'attribut `x` est alors un attribut statique, partagé par toutes les instances. C'est pourquoi dans l'exemple qui suit l'instruction `z.meth()` affiche la valeur `6` puisque l'appel `y.meth()` a incrémenté la variable statique `x`.

<<<

```
class essai_class:
    x = 5

    def meth(self):
        print(essai_class.x)
        essai_class.x += 1


y = essai_class()
z = essai_class()
y.meth()    # affiche 5
z.meth()    # affiche 6
```

>>>

```
    5
    6
```

Même si un attribut est statique, il peut être utilisé avec la syntaxe `self.attribut_statique` dans une méthode non statique à condition qu'un attribut non statique ne porte pas le même nom. Si tel est pourtant le cas, certaines confusions peuvent apparaître :

<<<

```
class exemple_classe:
    rnd = 42

    def incremente_rnd(self):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        self.rnd += 1
        return self.rnd


cl = exemple_classe()

print(cl.__dict__)                          # affiche {}
print(cl.__class__.__dict__["rnd"])     # affiche 42
cl.incremente_rnd()
print(cl.__dict__)                          # affiche {'rnd': 43}
print(cl.__class__.__dict__["rnd"])     # affiche 42
```

>>>

```
    {}
    42
    {'rnd': 43}
    42
```

Dans ce cas, ce sont en fait deux attributs qui sont créés. Le premier est un attribut statique créé avec la seconde ligne de l'exemple `rnd=42`. Le second attribut n'est pas statique et apparaît dès la première exécution de l'instruction `self.rnd+=1` comme le montre son apparition dans l'attribut `__dict__` qui ne recense pas les attributs statiques.

### Ajout de méthodes

Ce point décrit une fonctionnalité du langage *python* rarement utilisée. Il offre la possibilité d'ajouter une méthode à une classe alors même que cette fonction est définie à l'extérieur de la déclaration de la classe. Cette fonction doit obligatoirement accepter un premier paramètre qui recevra l'instance de la classe. La syntaxe utilise le mot-clé classmethod [162].

<<<

```
def nom_methode(cls):
    # code de la fonction
    pass


class nom_classe:
    # code de la classe
    nom_methode = classmethod(nom_methode)          # syntaxe 1


nom_classe.nom_methode = classmethod(nom_methode)   # syntaxe 2
```

>>>

```

```

`nom_classe` est une classe, `nom_methode` est une méthode, `nom_methode` est une fonction qui est par la suite considérée comme une méthode de la classe `nom_methode` grâce à l'une ou l'autre des deux instructions incluant le mot-clé `classmethod`. Dans l'exemple qui suit, cette syntaxe est utilisée pour inclure trois méthodes à la classe `essai_class` selon que la méthode est déclarée et affectée à cette classe à l'intérieur ou à l'extérieur du corps de `essai_class`.

<<<

162. https://docs.python.org/3/library/functions.html?highlight=classmethod#classmethod

```
def meth3(cls):
    print("ok meth3", cls.x)


def meth4(cls):
    print("ok meth4", cls.x)


class essai_classe:
    x = 5

    def meth(self):
        print("ok meth", self.x)

    def meth2(cls):
        print("ok meth2", cls.x)

    meth3 = classmethod(meth3)


x = essai_classe()
x.meth()                                # affiche ok meth 5
x.meth2()                               # affiche ok meth2 5
x.meth3()                               # affiche ok meth3 5

essai_classe.meth4 = classmethod(meth4)
x.meth4()                               # affiche ok meth4 5
```

>>>

```
    ok meth 5
    ok meth2 5
    ok meth3 5
    ok meth4 5
```

### Propriétés

Cette fonctionnalité est également peu utilisée, elle permet des raccourcis d'écriture. Les propriétés permettent de faire croire à l'utilisateur d'une instance de classe qu'il utilise une variable alors qu'il utilise en réalité une ou plusieurs méthodes. A chaque fois que le programmeur utilise ce faux attribut, il appelle une méthode qui calcule sa valeur. A chaque fois que le programmeur cherche à modifier la valeur de ce faux attribut, il appelle une autre méthode qui modifie l'instance.

**Syntaxe S11 : Déclaration d'une propriété**

```
class nom_classe :
    nom_propriete = property (fget, fset, fdel, doc)
```

La documentation de la fonction property [163] propose une autre écriture plus intuitive.

**Syntaxe S12 : Déclaration d'une propriété (2)**

163. https://docs.python.org/3/library/functions.html#property

```
class nom_classe :

    @property
    def fget_variable(self):
        return self.variable

    @variable.setter
    def fset_variable(self, v):
        self.variable = v
```

Au sein de ces trois lignes, `nom_classe` est une classe, `nom_propriete` est le nom de la propriété, `fget` est la méthode qui doit retourner la valeur du pseudo-attribut `nom_propriete`, `fset` est la méthode qui doit modifier la valeur du pseudo-attribut `nom_propriete`, `fdel` est la méthode qui doit détruire le pseudo-attribut `nom_propriete`, `doc` est un commentaire qui apparaîtra lors de l'appel de la fonction `help(nom_class)` ou `help(nom_class.nom_propriete)`.

Pour illustrer l'utilisation des propriétés, on part d'une classe `nombre_complexe` qui ne contient que les parties réelle et imaginaire. Le module désigne ici le module d'un nombre complexe qui est égal à sa norme. On le note $|a+ib| = \sqrt{a^2+b^2}$. On fait appel à une méthode qui calcule ce module. Lorsqu'on cherche à modifier ce module, on fait appel à une autre méthode qui multiplie les parties réelle et imaginaire par un nombre réel positif de manière à ce que le nombre complexe ait le module demandé. On procède de même pour la propriété `arg`.

La propriété `conj` retourne quant à elle le conjugué du nombre complexe mais la réciproque n'est pas prévue. On ne peut affecter une valeur à `conj`.

<<<

```
import math


class nombre_complexe(object):            # voir remarque après l'exemple
    def __init__(self, a=0, b=0):
        self.a = a
        self.b = b

    def __str__(self):
        if self.b == 0:
            return "%f" % (self.a)
        elif self.b > 0:
            return "%f + %f i" % (self.a, self.b)
        else:
            return "%f - %f i" % (self.a, -self.b)

    def get_module(self):
        return math.sqrt(self.a * self.a + self.b * self.b)

    def set_module(self, m):
        r = self.get_module()
        if r == 0:
            self.a = m
            self.b = 0
        else:
            d = m / r
            self.a *= d
            self.b *= d

    def get_argument(self):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        r = self.get_module()
        if r == 0:
            return 0
        else:
            return math.atan2(self.b / r, self.a / r)

    def set_argument(self, arg):
        m = self.get_module()
        self.a = m * math.cos(arg)
        self.b = m * math.sin(arg)

    def get_conjugue(self):
        return nombre_complexe(self.a, -self.b)

    module = property(fget=get_module,   fset=set_module,   doc="module")
    arg = property(fget=get_argument, fset=set_argument, doc="argument")
    conj = property(fget=get_conjugue,                   doc="conjugué")


c = nombre_complexe(0.5, math.sqrt(3)/2)
print("c = ",         c)          # affiche c =  0.500000 + 0.866025 i
print("module = ",    c.module)   # affiche module =  1.0
print("argument = ",  c.arg)      # affiche argument =  1.0471975512

c = nombre_complexe()
c.module = 1
c.arg = math.pi * 2 / 3
print("c = ",         c)          # affiche c =  -0.500000 + 0.866025 i
print("module = ",    c.module)   # affiche module =  1.0
print("argument = ",  c.arg)      # affiche argument =  2.09439510239
print("conjugué = ",  c.conj)     # affiche conjugué =  -0.500000 - 0.866025 i
```

>>>

```
    c =  0.500000 + 0.866025 i
    module =  0.9999999999999999
    argument =  1.0471975511965976
    c =  -0.500000 + 0.866025 i
    module =  0.9999999999999999
    argument =  2.0943951023931953
    conjugué =  -0.500000 - 0.866025 i
```

La propriété `conj` ne possède pas de fonction qui permet de la modifier. Par conséquent, l'instruction `c.conj = nombre_complexe(0,0)` produit une erreur. Etant donné qu'une propriété porte déjà le nom de `conj`, aucun attribut du même nom ne peut être ajouté à la classe `nombre_complexe`.

Afin que la propriété fonctionne correctement, il est nécessaire que la classe hérite de la classe `object` ou une de ses descendantes (voir également *Héritage* (page 134)).

### 3.1.7 Copie d'instances

#### Copie d'instance de classe simple

Aussi étrange que cela puisse paraître, le signe = ne permet pas de recopier une instance de classe. Il permet d'obtenir deux noms différents pour désigner le même objet. Dans l'exemple qui suit, la ligne `nb2 = nb` ne fait pas de copie de l'instance `nb`, elle permet d'obtenir un second nom `nb2` pour l'instance `nb`. Vu de l'extérieur, la ligne `nb2.rnd =`

0 paraît modifier à la fois les objets nb et nb2 puisque les lignes print(nb.rnd) et print(nb2.rnd) affichent la même chose. En réalité, nb et nb2 désignent le même objet.

<<<

```
class exemple_classe:
    def __init__(self):
        self.rnd = 42

    def methode1(self, n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()
nb2 = nb
print(nb.rnd)         # affiche 42
print(nb2.rnd)        # affiche 42

nb2.rnd = 0

print(nb2.rnd)        # affiche 0, comme prévu
print(nb.rnd)         # affiche 0, si nb et nb2 étaient des objets différents,
# cette ligne devrait afficher 42
```

>>>

```
    42
    42
    0
    0
```

Pour créer une copie de l'instance nb, il faut le dire explicitement en utilisant la fonction copy du module copy (voir le chapitre *Modules* (page 235)).

```
import copy
nom_copy = copy.copy(nom_instance)
```

nom_instance est une instance à copier, nom_copy est le nom désignant la copie. L'exemple suivant applique cette copie sur la classe exemple_classe générant des nombres aléatoires.

<<<

```
class exemple_classe:
    def __init__(self):
        self.rnd = 42

    def methode1(self, n):
        self.rnd = 397204094 * self.rnd % 2147483647
        return int(self.rnd % n)


nb = exemple_classe()

import copy             # pour utiliser le module copy
nb2 = copy.copy(nb)     # copie explicite

print(nb.rnd)     # affiche 42
print(nb2.rnd)    # affiche 42
```

(suite sur la page suivante)

(suite de la page précédente)

```
nb2.rnd = 0

print(nb2.rnd)    # affiche 0
print(nb.rnd)     # affiche 42
```

>>>

```
    42
    42
    0
    42
```

Le symbole égalité ne fait donc pas de copie, ceci signifie qu'une même instance de classe peut porter plusieurs noms.

<<<

```
m = [0, 1]
m2 = m
del m2     # supprime l'identificateur mais pas la liste
print(m)   # affiche [0, 1]
```

>>>

```
    [0, 1]
```

La suppression d'un objet n'est effective que s'il ne reste aucune variable le référençant. L'exemple suivant le montre.

<<<

```
class CreationDestruction (object):

    def __init__(self):
        print("constructeur")

    def __new__(self):
        print("__new__")
        return object.__new__(self)

    def __del__(self):
        print("__del__")


print("a")
m = CreationDestruction()
print("b")
m2 = m
print("c")
del m
print("d")
del m2
print("e")
```

>>>

```
    a
    __new__
    constructeur
```

(suite sur la page suivante)

(suite de la page précédente)

```
    b
    c
    d
    __del__
    e
```

Le destructeur est appelé autant de fois que le constructeur. Il est appelé lorsque plus aucun identificateur n'est relié à l'objet. Cette configuration survient lors de l'exemple précédent car le mot-clé `del` a détruit tous les identificateurs `m` et `m2` qui étaient reliés au même objet.

### Copie d'instance de classes incluant d'autres classes

La fonction `copy` n'est pas suffisante lorsqu'une classe inclut des attributs qui sont eux-mêmes des classes incluant des attributs. Dans l'exemple qui suit, la classe `exemple_classe` inclut un attribut de type `classe_incluse` qui contient un attribut `attr`. Lors de la copie à l'aide de l'instruction `nb2 = copy.copy(nb)`, l'attribut `inclus` n'est pas copié, c'est l'instruction `nb2.inclus = nb.inclus` qui est exécutée. On se retrouve donc avec deux noms qui désignent encore le même objet.

<<<

```
class classe_incluse:
    def __init__(self):
        self.attr = 3


class exemple_classe:
    def __init__(self):
        self.inclus = classe_incluse()
        self.rnd = 42


nb = exemple_classe()

import copy              # pour utiliser le module copy
nb2 = copy.copy(nb)      # copie explicite

print(nb.inclus.attr)    # affiche 3
print(nb2.inclus.attr)   # affiche 3

nb2.inclus.attr = 0

print(nb.inclus.attr)    # affiche 0 (on voudrait 3 ici)
print(nb2.inclus.attr)   # affiche 0
```

>>>

```
    3
    3
    0
    0
```

Pour effectivement copier les attributs dont le type est une classe, la première option - la plus simple - est de remplacer la fonction `copy` par la fonction `deepcopy`. Le comportement de cette fonction dans le cas des classes est le même que dans le cas des listes comme l'explique la remarque copy_deepopy_remarque_. La seconde solution, rarement utilisée, est d'utiliser l'opérateur `__copy__` et ainsi écrire le code associé à la copie des attributs de la classe.

**Syntaxe S13 : Déclaration de l'opérateur __copy__**

```
class nom_classe :
    def __copy__ () :
        copie = nom_classe(...)
        # ...
        return copie
```

---

nom_classe est le nom d'une classe. La méthode __copy__ doit retourner une instance de la classe nom_classe, dans cet exemple, cette instance a pour nom copie.

L'exemple suivant montre un exemple d'implémentation de la classe __copy__. Cette méthode crée d'abord une autre instance copie de la classe exemple_classe puis initialise un par un ses membres. L'attribut rnd est recopié grâce à une affectation car c'est un nombre. L'attribut inclus est recopié grâce à la fonction copy du module copy car c'est une instance de classe. Après la copie, on vérifie bien que modifier l'attribut inclus.attr de l'instance nb ne modifie pas l'attribut inclus.attr de l'instance nb2.

<<<

```
import copy


class classe_incluse:
    def __init__(self): self.attr = 3


class exemple_classe:
    def __init__(self):
        self.inclus = classe_incluse()
        self.rnd = 42

    def __copy__(self):
        copie = exemple_classe()
        copie.rnd = self.rnd
        copie.inclus = copy.copy(self.inclus)
        return copie


nb = exemple_classe()

nb2 = copy.copy(nb)     # copie explicite,
# utilise l'opérateur __copy__,
# cette ligne est équivalente à
# nb2 = nb.__copy__()

print(nb.rnd)           # affiche 42
print(nb2.rnd)          # affiche 42
print(nb.inclus.attr)   # affiche 3
print(nb2.inclus.attr)  # affiche 3

nb.inclus.attr = 0
nb.rnd = 1

print(nb.rnd)           # affiche 1
print(nb2.rnd)          # affiche 42
print(nb.inclus.attr)   # affiche 0
print(nb2.inclus.attr)  # affiche 3 (c'est le résultat souhaité)
```

>>>

```
    42
    42
    3
    3
    1
    42
    0
    3
```

On peut se demander pourquoi l'affectation n'est pas équivalente à une copie. Cela tient au fait que l'affectation en langage *python* est sans cesse utilisée pour affecter le résultat d'une fonction à une variable. Lorsque ce résultat est de taille conséquente, une copie peut prendre du temps. Il est préférable que le résultat de la fonction reçoive le nom prévu pour le résultat.

<<<

```
def fonction_liste():
    return list(range(4, 7))  # retourne la liste [4,5,6]


li = fonction_liste()         # la liste [4,5,6] n'est pas recopiée,
# l'identificateur l lui est affecté
print(li)
```

>>>

```
    [4, 5, 6]
```

Lorsqu'une fonction retourne un résultat mais que celui-ci n'est pas attribué à un nom de variable. Le langage *python* détecte automatiquement que ce résultat n'est plus lié à aucune variable. Il est détruit automatiquement. *python* implémente un mécanisme de garbage collector [164].

<<<

```
def fonction_liste():
    return list(range(4, 7))


fonction_liste()  # la liste [4,5,6] n'est pas recopiée,
# elle n'est pas non plus attribuée à une variable,
# elle est alors détruite automatiquement par le langage Python
```

>>>

```

```

### Listes et dictionnaires

Les listes et les dictionnaires sont des types modifiables et aussi des classes. Par conséquent, l'affectation et la copie ont un comportement identique à celui des classes.

<<<

```
l1 = [4, 5, 6]
l2 = l1
print(l1)       # affiche [4, 5, 6]
```

(suite sur la page suivante)

164. https://fr.wikipedia.org/wiki/Ramasse-miettes_(informatique)

(suite de la page précédente)

```
print(l2)        # affiche [4, 5, 6]
l2[1] = 10
print(l1)        # affiche [4, 10, 6]
print(l2)        # affiche [4, 10, 6]
```

>>>

```
    [4, 5, 6]
    [4, 5, 6]
    [4, 10, 6]
    [4, 10, 6]
```

Pour effectuer une copie, il faut écrire le code suivant :

<<<

```
l1 = [4, 5, 6]
import copy
l2 = copy.copy(l1)
print(l1)        # affiche [4, 5, 6]
print(l2)        # affiche [4, 5, 6]
l2[1] = 10
print(l1)        # affiche [4, 5, 6]
print(l2)        # affiche [4, 10, 6]
```

>>>

```
    [4, 5, 6]
    [4, 5, 6]
    [4, 5, 6]
    [4, 10, 6]
```

La fonction copy [165] ne suffit pourtant pas lorsque l'objet à copier est par exemple une liste incluant d'autres objets. Elle copiera la liste et ne fera pas de copie des objets eux-mêmes.

<<<

```
import copy
l0 = [[i] for i in range(0, 3)]
l1 = copy.copy(l0)
print(l0, "  -  ", l1)     # affiche [[0], [1], [2]]   -   [[0], [1], [2]]
l1[0][0] = 6
print(l0, "  -  ", l1)     # affiche [[6], [1], [2]]   -   [[6], [1], [2]]
```

>>>

```
    [[0], [1], [2]]   -   [[0], [1], [2]]
    [[6], [1], [2]]   -   [[6], [1], [2]]
```

Il n'est pas possible de modifier la méthode `__copy__` d'un objet de type liste. Il existe néanmoins la fonction deepcopy [166] qui permet de faire une copie à la fois de la liste et des objets qu'elle contient.

<<<

```
import copy
l0 = [[i] for i in range(0, 3)]
```

(suite sur la page suivante)

165. https://docs.python.org/3/library/copy.html?highlight=copy#copy.copy
166. https://docs.python.org/3/library/copy.html?highlight=copy#copy.deepcopy

(suite de la page précédente)

```
ll = copy.deepcopy(l0)
print(l0, "  -  ", ll)     # affiche [[0], [1], [2]]   -   [[0], [1], [2]]
ll[0][0] = 6
print(l0, "  -  ", ll)     # affiche [[0], [1], [2]]   -   [[6], [1], [2]]
```

>>>

```
    [[0], [1], [2]]   -   [[0], [1], [2]]
    [[0], [1], [2]]   -   [[6], [1], [2]]
```

### copy et deepcopy

La fonction `copy` effectue une copie d'un objet, la fonction `deepcopy` effectue une copie d'un objet et de ceux qu'il contient. La fonction `copy` est associée à la méthode `__copy__` tandis que la fonction `deepcopy` est associée à la méthode `__deepcopy__`. Il est rare que l'une de ces deux méthodes doivent être redéfinies. L'intérêt de ce paragraphe est plus de montrer le mécanisme que cache la fonction `deepcopy` qui est la raison pour laquelle il existe deux fonctions de copie et non une seule.

```
import copy
memo = {}
nom_copy = copy.deepcopy (nom_instance [,memo])
```

`nom_instance` est une instance à copier, `nom_copy` est le nom désignant la copie. `memo` est un paramètre facultatif : s'il est envoyé à la fonction `deepcopy`, il contiendra alors la liste de toutes les copies d'objet effectuées lors de cet appel.

```
class nom_classe :
    def __deepcopy__ (self,memo) :
        copie = copy.copy (self)
        # ...
        return copie
```

`nom_classe` est le nom d'une classe. La méthode `__deepcopy__` doit retourner une instance de la classe `nom_classe`, dans cet exemple, cette instance a pour nom `copie`. Le paramètre `memo` permet de conserver la liste des copies effectuées à condition d'appeler `deepcopy` avec un dictionnaire en paramètre.

Le programme suivant reprend le second programme du paragraphe *Copie d'instance de classes incluant d'autres classes* (page 126) et modifie la classe `classe_incluse` pour distinguer copie et copie profonde. Il peut être utile de lire le paragraphe *Clés de type modifiable* (page 29) pour comprendre pourquoi un dictionnaire utilisant comme clé une instance de classe est possible.

<<<

```
import copy


class classe_incluse:
    def __init__(self):
        self.attr = 3


class exemple_classe:
    def __init__(self):
        self.inclus = classe_incluse()
        self.rnd = 42
```

(suite sur la page suivante)

(suite de la page précédente)

```
    def __copy__(self):
        copie = exemple_classe()
        copie.rnd = self.rnd
        return copie

    def __deepcopy__(self, memo):
        if self in memo:
            return memo[self]
        copie = copy.copy(self)
        memo[self] = copie     # mémorise la copie de self qui est copie
        copie.inclus = copy.deepcopy(self.inclus, memo)
        return copie


nb = exemple_classe()

nb2 = copy.deepcopy(nb)    # copie explicite à tous niveaux,
# utilise l'opérateur __copy__,
# cette ligne est équivalente à
# nb2 = nb.__deepcopy__()

print(nb.rnd)            # affiche 42
print(nb2.rnd)           # affiche 42
print(nb.inclus.attr)    # affiche 3
print(nb2.inclus.attr)   # affiche 3

nb.inclus.attr = 0
nb.rnd = 1

print(nb.rnd)            # affiche 1
print(nb2.rnd)           # affiche 42
print(nb.inclus.attr)    # affiche 0
print(nb2.inclus.attr)   # affiche 3  # résultat souhaité
```

>>>

```
    42
    42
    3
    3
    1
    42
    0
    3
```

On peut se demander quel est l'intérêt de la méthode `__deepcopy__` et surtout du paramètre `memo` modifié par la ligne `memo[self] = copie`. Ce détail est important lorsqu'un objet inclut un attribut égal à lui-même ou inclut un objet qui fait référence à l'objet de départ comme dans l'exemple qui suit.

<<<

```
import copy


class Objet1:
    def __init__(self, i):
        self.i = i
```

(suite sur la page suivante)

(suite de la page précédente)

```
    def __str__(self):
        return "o1 " + str(self.i) + " : " + str(self.o2.i)


class Objet2:
    def __init__(self, i, o):
        self.i = i
        self.o1 = o
        o.o2 = self

    def __str__(self):
        return "o2 " + str(self.i) + " : " + str(self.o1.i)

    def __deepcopy__(self, memo):
        return Objet2(self.i, self.o1)


o1 = Objet1(1)
o2 = Objet2(2, o1)
print(o1)  # affiche o1 1 : 2
print(o2)  # affiche o2 2 : 1

o3 = copy.deepcopy(o2)
o3.i = 4
print(o1)  # affiche o1 1 : 4    --> on voudrait 2
print(o2)  # affiche o2 2 : 1
print(o3)  # affiche o2 4 : 1
```

>>>

```
    o1 1 : 2
    o2 2 : 1
    o1 1 : 4
    o2 2 : 1
    o2 4 : 1
```

On modifie le programme comme suit pour obtenir une recopie d'instances de classes qui pointent les unes sur vers les autres. Le paramètre `memo` sert à savoir si la copie de l'objet a déjà été effectuée ou non. Si non, on fait une copie, si oui, on retourne la copie précédemment effectuée et conservée dans `memo`.

<<<

```
import copy


class Objet1:
    def __init__(self, i):
        self.i = i

    def __str__(self):
        return "o1 " + str(self.i) + " : " + str(self.o2.i)

    def __deepcopy__(self, memo={}):
        if self in memo:
            return memo[self]
        r = Objet1(self.i)
```

(suite sur la page suivante)

(suite de la page précédente)

```
        memo[self] = r
        r.o2 = copy.deepcopy(self.o2, memo)
        return r


class Objet2:
    def __init__(self, i, o):
        self.i = i
        self.o1 = o
        o.o2 = self

    def __str__(self):
        return "o2 " + str(self.i) + " : " + str(self.o1.i)

    def __deepcopy__(self, memo={}):
        if self in memo:
            return memo[self]
        r = Objet2(self.i, self.o1)
        memo[self] = r
        r.o1 = copy.deepcopy(self.o1, memo)
        return r


o1 = Objet1(1)
o2 = Objet2(2, o1)

print(o1)  # affiche o1 1 : 2
print(o2)  # affiche o2 2 : 1

o3 = copy.deepcopy(o2)
o3.i = 4
print(o1)  # affiche o1 1 : 2    --> on a 2 cette fois-ci
print(o2)  # affiche o2 2 : 1
print(o3)  # affiche o2 4 : 1
```

>>>

```
    o1 1 : 2
    o2 2 : 1
    o1 1 : 2
    o2 2 : 1
    o2 4 : 1
```

Ces problématiques se rencontrent souvent lorsqu'on aborde le problème de la sérialisation d'un objet qui consiste à enregistrer tout objet dans un fichier, même si cet objet inclut des références à des objets qui font référence à lui-même. C'est ce qu'on appelle des références circulaires. L'enregistrement d'un tel objet avec des références circulaires et sa relecture depuis un fichier se résolvent avec les mêmes artifices que ceux proposés ici pour la copie. L'utilisation des opérateurs `__copy__` et `__deepcopy__` est peu fréquente. Les fonctions `copy` et `deepcopy` du module `copy` suffisent dans la plupart des cas.

### 3.1.8 Attributs figés

Il arrive parfois qu'une classe contienne peu d'informations et soit utilisée pour créer un très grand nombre d'instances. Les paragraphes précédents ont montré que l'utilisation des attributs était assez souple. Il est toujours possible d'ajouter un attribut à n'importe quelle instance. En contrepartie, chaque instance conserve en mémoire un dictionnaire `__dict__` qui recense tous les attributs qui lui sont associés. Pour une classe susceptible d'être fréquemment instan-

ciée comme un point dans l'espace (voir paragraphe exemple_point_xyz), chaque instance n'a pas besoin d'avoir une liste variable d'attributs. Le langage *python* offre la possibilité de figer cette liste.

**Syntaxe S14 : Déclaration d'attributs figés**

```
class nom_classe (object) :
    __slots__ = "attribut_1", ..., "attribut_n"
```

`nom_classe` est le nom de la classe, elle doit hériter de `object` ou d'une classe qui en hérite elle-même (voir paragraphe *Héritage* (page 134)). Il faut ensuite ajouter au début du corps de la classe la ligne `__slots__= "attribut_1", ..., "attribut_n"` où `attribut_1` à `attribut_n` sont les noms des attributs de la classe. Aucun autre ne sera accepté.

L'exemple suivant utilise cette syntaxe pour définir un point avec seulement trois attributs `_x`, `_y`, `_z`.

<<<

```
class point_espace(object):
    __slots__ = "_x", "_y", "_z"

    def __init__(self, x, y, z):
        self._x, self._y, self._z = x, y, z

    def __str__(self):
        return "(%f,%f,%f)" % (self._x, self._y, self._z)


a = point_espace(1, -2, 3)
print(a)
```

>>>

```
    (1.000000,-2.000000,3.000000)
```

Etant donné que la liste des attributs est figée, l'instruction `a.j = 6` qui ajoute un attribut `j` à l'instance `a` déclenche une exception. La même erreur se déclenche si on cherche à ajouter cet attribut depuis une méthode (`self.j=6`). L'attribut `__dict__` n'existe pas non plus, par conséquent, l'expression `a.__dict__` génère la même exception. La présence de l'instruction `__slots__ = ...` n'a aucun incidence sur les attributs statiques.

## 3.1.9 Héritage

L'héritage est un des grands avantages de la programmation objet. Il permet de créer une classe à partir d'une autre en ajoutant des attributs, en modifiant ou en ajoutant des méthodes. En quelque sorte, on peut modifier des méthodes d'une classe tout en conservant la possibilité d'utiliser les anciennes versions.

### Exemple autour de pièces de monnaie

On désire réaliser une expérience à l'aide d'une pièce de monnaie. On effectue cent tirages successifs et on compte le nombre de fois où la face pile tombe. Le programme suivant implémente cette expérience sans utiliser la programmation objet.

<<<

```
import random  # extension interne incluant des fonctions
# simulant des nombres aléatoires,
# random.randint (a,b) --> retourne un nombre entier entre a et b
```

(suite sur la page suivante)

(suite de la page précédente)

```
# cette ligne doit être ajoutée à tous les exemples suivant
# même si elle n'y figure plus


def cent_tirages():
    s = 0
    for i in range(0, 100):
        s += random.randint(0, 1)
    return s


print(cent_tirages())
```

>>>

```
    51
```

On désire maintenant réaliser cette même expérience pour une pièce truquée pour laquelle la face pile sort avec une probabilité de `0,7`. Une solution consiste à réécrire la fonction `cent_tirages` pour la pièce truquée.

<<<

```
import random


def cent_tirages():
    s = 0
    for i in range(0, 100):
        t = random.randint(0, 10)
        if t >= 3:
            s += 1
    return s


print(cent_tirages())
```

>>>

```
    74
```

Toutefois cette solution n'est pas satisfaisante car il faudrait réécrire cette fonction pour chaque pièce différente pour laquelle on voudrait réaliser cette expérience. Une autre solution consiste donc à passer en paramètre de la fonction `cent_tirages` une fonction qui reproduit le comportement d'une pièce, qu'elle soit normale ou truquée.

<<<

```
import random


def piece_normale():
    return random.randint(0, 1)


def piece_truquee():
    t = random.randint(0, 10)
    if t >= 3:
        return 1
    else:
```

(suite sur la page suivante)

(suite de la page précédente)

```
        return 0


def cent_tirages(piece):
    s = 0
    for i in range(0, 100):
        s += piece()
    return s


print(cent_tirages(piece_normale))
print(cent_tirages(piece_truquee))
```

>>>

```
    49
    71
```

Mais cette solution possède toujours un inconvénient car les fonctions associées à chaque pièce n'acceptent aucun paramètre. Il n'est pas possible de définir une pièce qui est normale si la face *pile* vient de sortir et qui devient truquée si la face *face* vient de sortir. On choisit alors de représenter une pièce normale par une classe.

<<<

```
import random


class piece_normale:
    def tirage(self):
        return random.randint(0, 1)

    def cent_tirages(self):
        s = 0
        for i in range(0, 100):
            s += self.tirage()
        return s


p = piece_normale()
print(p.cent_tirages())
```

>>>

```
    52
```

On peut aisément recopier et adapter ce code pour la pièce truquée.

<<<

```
import random


class piece_normale:
    def tirage(self):
        return random.randint(0, 1)

    def cent_tirages(self):
        s = 0
```

(suite sur la page suivante)

(suite de la page précédente)

```
        for i in range(0, 100):
            s += self.tirage()
        return s


class piece_truquee:
    def tirage(self):
        t = random.randint(0, 10)
        if t >= 3:
            return 1
        else:
            return 0

    def cent_tirages(self):
        s = 0
        for i in range(0, 100):
            s += self.tirage()
        return s


p = piece_normale()
print(p.cent_tirages())
p2 = piece_truquee()
print(p2.cent_tirages())
```

>>>

```
    57
    76
```

Toutefois, pour les deux classes `piece_normale` et `piece_truquee`, la méthode `cent_tirage` est exactement la même. Il serait préférable de ne pas répéter ce code puisque si nous devions modifier la première - un nombre de tirages différent par exemple -, il faudrait également modifier la seconde. La solution passe par l'héritage. On va définir la classe `piece_truquee` à partir de la classe `piece_normale` en remplaçant seulement la méthode `tirage` puisqu'elle est la seule à changer.

On indique à la classe `piece_truquee` qu'elle hérite - ou dérive - de la classe `piece_normale` en mettant `piece_normale` entre parenthèses sur la ligne de la déclaration de la classe `piece_truquee`. Comme la méthode `cent_tirages` ne change pas, elle n'a pas besoin d'apparaître dans la définition de la nouvelle classe même si cette méthode est aussi applicable à une instance de la classe `piece_truquee`.

<<<

```
import random


class piece_normale:
    def tirage(self):
        return random.randint(0, 1)

    def cent_tirages(self):
        s = 0
        for i in range(0, 100):
            s += self.tirage()
        return s
```

(suite sur la page suivante)

(suite de la page précédente)

```
class piece_truquee (piece_normale):
    def tirage(self):
        t = random.randint(0, 10)
        if t >= 3:
            return 1
        else:
            return 0


p = piece_normale()
print(p.cent_tirages())
p2 = piece_truquee()
print(p2.cent_tirages())
```

>>>

```
    55
    70
```

Enfin, on peut définir une pièce très truquée qui devient truquée si *face* vient de sortir et qui redevient normale si *pile* vient de sortir. Cette pièce très truquée sera implémentée par la classe `piece_tres_truquee`. Elle doit contenir un attribut `avant` qui conserve la valeur du précédent tirage. Elle doit redéfinir la méthode `tirage` pour être une pièce normale ou truquée selon la valeur de l'attribut `avant`. Pour éviter de réécrire des méthodes déjà écrites, la méthode `tirage` de la classe `piece_tres_truquee` doit appeler la méthode `tirage` de la classe `piece_truquee` ou celle de la classe `piece_normale` selon la valeur de l'attribut `avant`.

<<<

```
import random


class piece_normale:
    def tirage(self):
        return random.randint(0, 1)

    def cent_tirages(self):
        s = 0
        for i in range(0, 100):
            s += self.tirage()
        return s


class piece_truquee (piece_normale):
    def tirage(self):
        t = random.randint(0, 10)
        if t >= 3:
            return 1
        else:
            return 0


class piece_tres_truquee (piece_truquee):
    def __init__(self):
        # création de l'attribut avant
        self.avant = 0

    def tirage(self):
```

(suite sur la page suivante)

(suite de la page précédente)

```
        if self.avant == 0:
            # appel de la méthode tirage de la classe piece_truquee
            self.avant = piece_truquee.tirage(self)
        else:
            # appel de la méthode tirage de la classe piece_normale
            self.avant = piece_normale.tirage(self)
        return self.avant


p = piece_normale()
print("normale ", p.cent_tirages())
p2 = piece_truquee()
print("truquee ", p2.cent_tirages())
p3 = piece_tres_truquee()
print("tres truquee ", p3.cent_tirages())
```

>>>

```
    normale  44
    truquee  77
    tres truquee  59
```

L'héritage propose donc une manière élégante d'organiser un programme. Il rend possible la modification des classes d'un programme sans pour autant les altérer.

---

**Définition D7 : héritage**

On dit qu'une classe $B$ hérite d'une autre classe $A$ si la déclaration de $B$ inclut les attributs et les méthodes de la classe $A$.

---

La surcharge est un autre concept qui va de pair avec l'héritage. Elle consiste à redéfinir des méthodes déjà définies chez l'ancêtre. Cela permet de modifier le comportement de méthodes bien que celles-ci soient utilisées par d'autres méthodes dont la définition reste inchangée.

---

**Définition D8 : surcharge**

Lorsqu'une classe *B* hérite de la classe *A* et redéfinit une méthode de la classe *A* portant le même nom, on dit qu'elle surcharge cette méthode. S'il n'est pas explicitement précisé qu'on fait appel à une méthode d'une classe donnée, c'est toujours la méthode surchargée qui est exécutée.

---

### Syntaxe

L'héritage obéit à la syntaxe suivante.

---

**Syntaxe S15 : Héritage**

```
class nom_classe (nom_ancetre) :
    # corps de la classe
    # ...
```

---

`nom_classe` désigne le nom d'une classe qui hérite ou dérive d'une autre classe `nom_ancetre`. Celle-ci `nom_ancetre` doit être une classe déjà définie.

L'utilisation de la fonction `help` permet de connaître tous les ancêtres d'une classe. On applique cette fonction à la classe `piece_tres_truquee` définie au paragraphe précédent.

```
help (piece_tres_truquee)
```

On obtient le résultat suivant :

```
Help on class piece_tres_truquee in module __main__:

class piece_tres_truquee(piece_truquee)
 |  Method resolution order:
 |      piece_tres_truquee
 |      piece_truquee
 |      piece_normale
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |
 |  tirage(self)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from piece_normale:
 |
 |  cent_tirages(self)
```

La rubrique Method Resolution Order[167] indique la liste des héritages successifs qui ont mené à la classe `piece_tres_truquee`. Cette rubrique indique aussi que, lorsqu'on appelle une méthode de la classe `piece_tres_truquee`, si elle n'est pas redéfinie dans cette classe, le langage *python* la cherchera chez l'ancêtre direct, ici, la classe `piece_truquee`. Si elle ne s'y trouve toujours pas, *python* ira la chercher aux niveaux précédents jusqu'à ce qu'il la trouve.

L'attribut `__bases__` d'une classe (voir paragraphe *Attributs implicites* (page 103)) contient le (ou les ancêtres, voir paragraphe *Héritage multiple* (page 144)). Il suffit d'interroger cet attribut pour savoir si une classe hérite d'une autre comme le montre l'exemple suivant.

<<<

```
class piece_normale:
    pass


class piece_truquee(piece_normale):
    pass


class piece_tres_truquee(piece_truquee):
    pass


for l in piece_tres_truquee.__bases__:
    print(l)    # affiche __main__.piece_truquee
print(piece_normale in piece_tres_truquee.__bases__)  # affiche False
print(piece_truquee in piece_tres_truquee.__bases__)  # affiche True
```

>>>

167. https://www.python.org/download/releases/2.3/mro/

```
    <class 'pyquickhelper.sphinxext.sphinx_runpython_extension.run_python_script_
↪139651626436936.<locals>.piece_truquee'>
    False
    True
```

La fonction issubclass [168] permet d'obtenir un résultat équivalent. `issubclass(A,B)` indique si la classe `A` hérite directement ou indirectement de la classe `B`. Le paragraphe *Fonctions issubclass et isinstance* (page 146) revient sur cette fonction.

<<<

```
class piece_normale:
    pass


class piece_truquee(piece_normale):
    pass


class piece_tres_truquee(piece_truquee):
    pass


print(issubclass(piece_tres_truquee, piece_normale))  # affiche True
print(issubclass(piece_truquee, piece_normale))       # affiche True
```

>>>

```
    True
    True
```

Dans les exemples précédents, `piece_normale` ne dérive d'aucune autre classe. Toutefois, le langage *python* propose une classe d'objets dont héritent toutes les autres classes définies par le langage : c'est la classe `object`. Les paragraphes précédents ont montré qu'elle offrait certains avantages (voir paragraphe *Propriétés* (page 121) sur les propriétés ou encore paragraphe *Attributs figés* (page 133) sur les attributs non liés).

Le paragraphe précédent a montré qu'il était parfois utile d'appeler dans une méthode une autre méthode appartenant explicitement à l'ancêtre direct de cette classe ou à un de ses ancêtres. La syntaxe est la suivante.

**Syntaxe S16 : Surcharge de méthodes héritées**

```
class nom_classe (nom_ancetre) :
    def nom_autre_methode (self, ...) :
        # ...
    def nom_methode (self, ...) :
        nom_ancetre.nom_methode (self, ...)
            # appel de la méthode définie chez l'ancêtre
        nom_ancetre.nom_autre_methode (self, ...)
            # appel d'une autre méthode définie chez l'ancêtre
        self.nom_autre_methode (...)
            # appel d'une méthode surchargée
```

`nom_classe` désigne le nom d'une classe, `nom_ancetre` est le nom de la classe dont `nom_classe` hérite ou dérive. `nom_methode` est une méthode surchargée qui appelle la méthode portant le même nom mais définie dans la classe `nom_ancetre` ou un de ses ancêtres. `nom_autre_methode` est une autre méthode. La mé-

168. https://docs.python.org/3/library/functions.html?highlight=issubclass#issubclass

thode nom_methode de la classe nom_classe peut faire explicitement appel à une méthode définie chez l'ancêtre nom_ancetre même si elle est également surchargée ou faire appel à la méthode surchargée.

Ces appels sont très fréquents en ce qui concerne les constructeurs qui appellent le constructeur de l'ancêtre. Il est même conseillé de le faire à chaque fois.

<<<

```
class A:
    def __init__(self):
        self.x = 0


class B (A):
    def __init__(self):
        A.__init__(self)
        self.y = 0
```

>>>

```

```

Contrairement aux méthodes, la surcharge d'attributs n'est pas possible. Si un ancêtre possède un attribut d'identificateur a, les classes dérivées le possèdent aussi et ne peuvent en déclarer un autre du même nom. Cela tient au fait que quelque soit la méthode utilisée, celle de l'ancêtre ou celle d'une classe dérivée, c'est le même dictionnaire d'attributs __dict__ qui est utilisé. En revanche, si la classe ancêtre déclare un attribut dans son constructeur, il ne faut pas oublier de l'appeler dans le constructeur de la classe fille afin que cette attribut existe pour la classe fille.

<<<

```
class ancetre:
    def __init__(self):
        self.a = 5

    def __str__(self):
        return "a = " + str(self.a)


class fille (ancetre):
    def __init__(self):
        ancetre.__init__(self)      # cette ligne est importante
        # car sans elle, l'attribut a n'existe pas
        self.a += 1

    def __str__(self):
        s = "a = " + str(self.a)
        return s


x = ancetre()
print(x)          # affiche a = 5
y = fille()
print(y)          # affiche a = 6
```

>>>

```
    a = 5
    a = 6
```

### Sens de l'héritage

Il n'est pas toujours évident de concevoir le sens d'un héritage. En mathématique, le carré est un rectangle dont les côtés sont égaux. A priori, une classe `carre` doit dériver d'une classe `rectangle`.

<<<

```
class rectangle:
    def __init__(self, a, b):
        self.a, self.b = a, b

    def __str__(self):
        return "rectangle " + str(self.a) + " x " + str(self.b)


class carre (rectangle):
    def __init__(self, a):
        rectangle.__init__(self, a, a)


r = rectangle(3, 4)
print(r)  # affiche rectangle 3 x 4
c = carre(5)
print(c)  # affiche rectangle 5 x 5
```

>>>

```
    rectangle 3 x 4
    rectangle 5 x 5
```

Toutefois, on peut aussi considérer que la classe `carre` contient une information redondante puisqu'elle possède deux attributs qui seront toujours égaux. On peut se demander s'il n'est pas préférable que la classe `rectangle` hérite de la classe `carre`.

<<<

```
class carre:
    def __init__(self, a):
        self.a = a

    def __str__(self):
        return "carre " + str(self.a)


class rectangle (carre):
    def __init__(self, a, b):
        carre.__init__(self, a)
        self.b = b

    def __str__(self):
        return "rectangle " + str(self.a) + " x " + str(self.b)


r = rectangle(3, 4)
print(r)  # affiche rectangle 3 x 4
c = carre(5)
print(c)  # affiche carre 5
```

>>>

```
    rectangle 3 x 4
    carre 5
```

Cette seconde version minimise l'information à mémoriser puisque la classe `carre` ne contient qu'un seul attribut et non deux comme dans l'exemple précédent. Néanmoins, il a fallu surcharger l'opérateur `__str__` afin d'afficher la nouvelle information.

Il n'y a pas de meilleur choix parmi ces deux solutions proposées. La première solution va dans le sens des propriétés croissantes, les méthodes implémentées pour les classes de bases restent vraies pour les suivantes. La seconde solution va dans le sens des attributs croissants, des méthodes implémentées pour les classes de bases doivent souvent être adaptées pour les héritiers. En contrepartie, il n'y a pas d'information redondante.

Ce problème d'héritage ne se pose pas à chaque fois. Dans l'exemple du paragraphe *Exemple autour de pièces de monnaie* (page 134) autour des pièces truquées, il y a moins d'ambiguïté sur le sens de l'héritage. Celui-ci est guidé par le problème à résoudre qui s'avère plus simple à concevoir dans le sens d'un héritage d'une pièce normale vers une pièce truquée.

Dans le cas des classes `carre` et `rectangle`, il n'est pas possible de déterminer la meilleure solution tant que leur usage ultérieur n'est pas connu. Ce problème revient également lorsqu'on définit des matrices et des vecteurs. Un vecteur est une matrice d'une seul colonne, il ne possède qu'une seule dimension au lieu de deux pour une matrice. L'exercice exeercie_heritage_multiple revient sur ce dilemme.

### Héritage multiple

Jusqu'à présent, tous les exemples d'héritages entre classes n'ont fait intervenir que deux classes, la classe ancêtre dont hérite la classe descendante. L'héritage multiple part du principe qu'il peut y avoir plusieurs ancêtres pour une même classe. La classe descendante hérite dans ce cas de tous les attributs et méthodes de tous ses ancêtres.

Dans l'exemple qui suit, la classe `C` hérite des classes `A` et `B`. Elle hérite donc des méthodes de `carre` et `cube`. Chacune des classes `A` et `B` contient un constructeur qui initialise l'attribut `a`. Le constructeur de la classe `C` appelle le constructeur de la classe `A` pour initialiser cet attribut.

<<<

```
class A:
    def __init__(self):
        self.a = 5

    def carre(self):
        return self.a ** 2


class B:
    def __init__(self):
        self.a = 6

    def cube(self):
        return self.a ** 3


class C (A, B):
    def __init__(self):
        A.__init__(self)


x = C()
print(x.carre())    # affiche 25
print(x.cube())     # affiche 125
```

>>>

```
    25
    125
```

Mais ces héritages multiples peuvent parfois apporter quelques ambiguïtés comme le cas où au moins deux ancêtres possèdent une méthode du même nom. Dans l'exemple qui suit, la classe `C` hérite toujours des classes `A` et `B`. Ces deux classes possèdent une méthode `calcul`. La classe `C`, qui hérite des deux, possède aussi une méthode `calcul` qui, par défaut, sera celle de la classe `A`.

<<<

```
class A:
    def __init__(self):
        self.a = 5

    def calcul(self):
        return self.a ** 2


class B:
    def __init__(self):
        self.a = 6

    def calcul(self):
        return self.a ** 3


class C (A, B):
    def __init__(self):
        A.__init__(self)


x = C()
print(x.calcul())  # affiche 25
```

>>>

```
    25
```

Cette information est disponible via la fonction `help` appliquée à la classe `C`. C'est dans ce genre de situations que l'information apportée par la section Method Resolution Order [169] est importante.

```
class C(A, B)
 |  Method resolution order:
 |      C
 |      A
 |      B
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |
 |  calcul(self)
```

Pour préciser que la méthode `calcul` de la classe `C` doit appeler la méthode `calcul` de la classe `B` et non `A`, il faut l'écrire explicitement en surchargeant cette méthode.

<<<

169. https://www.python.org/download/releases/2.3/mro/

```
class A:
    def __init__(self):
        self.a = 5

    def calcul(self):
        return self.a ** 2


class B:
    def __init__(self):
        self.a = 6

    def calcul(self):
        return self.a ** 3


class C (A, B):
    def __init__(self):
        A.__init__(self)

    def calcul(self):
        return B.calcul(self)


x = C()
print(x.calcul())  # affiche 125
```

>>>

```
    125
```

L'exemple précédent est un cas particulier où il n'est pas utile d'appeler les constructeurs des deux classes dont la classe `C` hérite mais c'est un cas particulier. Le constructeur de la classe `C` devrait être ainsi :

```
class C (A,B) :
    def __init__ (self):
        A.__init__(self)
        B.__init__ (self)
```

### Fonctions issubclass et isinstance

La fonction issubclass [170] permet de savoir si une classe hérite d'une autre.

```
issubclass (B, A)
```

Le résultat de cette fonction est vrai si la classe `B` hérite de la classe `A`, le résultat est faux dans tous les autres cas. La fonction prend comme argument des classes et non des instances de classes.

L'exemple qui suit utilise cette fonction dont le résultat est vrai même pour des classes qui n'héritent pas directement l'une de l'autre.

<<<

```
class A (object):
    pass
```

(suite sur la page suivante)

170. https://docs.python.org/3/library/functions.html?highlight=issubclass#issubclass

(suite de la page précédente)

```
class B (A):
    pass


class C (B):
    pass


print(issubclass(A, B))      # affiche False
print(issubclass(B, A))      # affiche True
print(issubclass(A, C))      # affiche False
print(issubclass(C, A))      # affiche True
```

>>>

```
    False
    True
    False
    True
```

Lorsqu'on souhaite appliquer la fonction à une instance de classe, il faut faire appel à l'attribut `__class__`. En reprenant les classes définies par l'exemple précédant cela donne :

<<<

```
class A (object):
    pass


class B (A):
    pass


class C (B):
    pass


a = A()
b = B()

print(issubclass(a.__class__, B))      # affiche False
print(issubclass(b.__class__, A))      # affiche True
print(issubclass(a.__class__, A))      # affiche True
```

>>>

```
    False
    True
    True
```

La fonction isinstance [171] permet de savoir si une instance de classe est d'une type donné. Elle est équivalente à la fonction `issubclass` à ceci près qu'elle prend comme argument une instance et une classe. L'exemple précédent devient avec la fonction `isinstance` :

<<<

171. https://docs.python.org/3/library/functions.html?highlight=isinstance#isinstance

```
class A (object):
    pass


class B (A):
    pass


class C (B):
    pass


a = A()
b = B()
print(isinstance(a, B))      # affiche False
print(isinstance(b, A))      # affiche True
print(isinstance(a, A))      # affiche True
```

>>>

```
    False
    True
    True
```

L'utilisation des fonctions `issubclass` et `isinstance` n'est pas très fréquente mais elle permet par exemple d'écrire une fonction qui peut prendre en entrée des types variables.

<<<

```
def fonction_somme_list(ens):
    r = "list "
    for e in ens:
        r += e
    return r


def fonction_somme_dict(ens):
    r = "dict "
    for k, v in ens.items():
        r += v
    return r


def fonction_somme(ens):
    if isinstance(ens, dict):
        return fonction_somme_dict(ens)
    elif isinstance(ens, list):
        return fonction_somme_list(ens)
    else:
        return "erreur"


li = ["un", "deux", "trois"]
di = {1: "un", 2: "deux", 3: "trois"}
tu = ("un", "deux", "trois")

print(fonction_somme(li))   # affiche list undeuxtrois
print(fonction_somme(di))   # affiche dict undeuxtrois
print(fonction_somme(tu))   # affiche erreur
```

>>>

```
    list undeuxtrois
    dict undeuxtrois
    erreur
```

L'avantage est d'avoir une seule fonction capable de s'adapter à différents type de variables, y compris des types créés par un programmeur en utilisant les classes.

### 3.1.10 Compilation de classes

La compilation de classe fonctionne de manière similaire à celle de la compilation de fonctions (voir paragraphe *Compilation dynamique (compile, exec)* (page 70)). Il s'agit de définir une classe sous forme de chaîne de caractères puis d'appeler la fonction `compile` pour ajouter cette classe au programme et s'en servir. L'exemple suivant reprend deux classes décrites au paragraphe *Sens de l'héritage* (page 143). Elles sont incluses dans une chaîne de caractères, compilées puis incluses au programme (fonction `exec`).

<<<

```
s = """class carre :
    def __init__( self, a) : self.a = a
    def __str__ (self)     : return "carre " + str (self.a)

class rectangle (carre):
    def __init__(self,a,b) :
        carre.__init__(self, a)
        self.b = b
    def __str__ (self) :
        return "rectangle " + str(self.a) +  " x " +  str (self.b)
"""

obj = compile(s, "", "exec")        # code à compiler
exec(obj)                           # classes incorporées au programme

r = rectangle(3, 4)
print(r)  # affiche rectangle 3 x 4
c = carre(5)
print(c)  # affiche carre 5
```

>>>

```
    [runpythonerror]
    Traceback (most recent call last):
      File "<stdin>", line 17, in <module>
      File "", line 5
        class rectangle (carre):
                                ^
    IndentationError: unindent does not match any outer indentation level
```

Comme toute fonction, la fonction `compile` génère une exception lorsque la chaîne de caractères contient une erreur. Le programme qui suit essaye de compiler une chaîne de caractères confondant `self` et `seilf`.

<<<

```
"""erreur de compilation incluses dans le code inséré dans la
chaîne de caractère s"""
s = """class carre :
```

(suite sur la page suivante)

(suite de la page précédente)

```
    def __init__( self, a) :
        seilf.a = a   # erreur de compilation
    def __str__ (self) :
        return "carre " + str (self.a) """

obj = compile(s, "variable s", "exec")    # code à compiler
exec(obj)                                 # classes incorporées au programme

c = carre(5)
print(c)      # affiche carre 5
```

>>>

```
    [runpythonerror]
    Traceback (most recent call last):
      File "<stdin>", line 16, in <module>
      File "variable s", line 3, in __init__
    NameError: name 'seilf' is not defined
```

Le message d'erreur est le même que pour un programme ne faisant pas appel à la fonction `compile` à ceci près que le fichier où a lieu l'erreur est `variable s` qui est le second paramètre envoyé à la fonction `compile`. Le numéro de ligne fait référence à la troisième ligne de la chaîne de caractères `s` et non à la troisième ligne du programme.

<<<

```
class fromage:

    def __init__(self, p, c, o):
        self.poids = p
        self.couleur = c
        self.odeur = o

    def decouper(self, nb):
        l = []
        for i in range(0, nb):
            f = fromage(self.poids/nb,
                        self.couleur, self.odeur)
            l.append(f)
        return l

    def __str__(self):
        s = "poids : " + str(self.poids)
        s += " couleur : " + str(self.couleur)
        s += " odeur : " + str(self.odeur)
        return s

    def __add__(self, f):
        print("ajout fromage")
        poids = self.poids + f.poids
        couleur = [0, 0, 0]
        for i in range(0, 3):
            couleur[i] = (self.couleur[i] * self.poids
                          + f.couleur[i] * f.poids) / poids
        odeur = (self.odeur * self.poids +
                 f.odeur * f.poids) / poids
        couleur = (couleur[0], couleur[1], couleur[2])
```

(suite sur la page suivante)

(suite de la page précédente)

```
        return fromage(poids, couleur, odeur)


class gruyere(fromage):
    def __init__(self, p):
        fromage.__init__(self, p, c=(150, 150, 0), o=0.1)

    def __str__(self):
        s = fromage.__str__(self)
        s = "gruyère, " + s
        return s

    def __add__(self, f):
        print("ajout gruyère")
        if not isinstance(f, gruyere):
            return fromage.__add__(self, f)
        else:
            r = gruyere(self.poids + f.poids)
            return r


# ------------------------------------------
fr = fromage(5.0, (255, 0, 0), 0.5)
fr2 = fromage(10.0, (0, 255, 0), 1)
fr3 = fr + fr2
print(fr)
print(fr2)
print(fr3)
print("----------------------")
g = gruyere(3.0)
g2 = gruyere(7.0)
g3 = g + g2
print(g)
print(g2)
print(g3)
print("----------------------")
print(fr2 + g)
```

>>>

```
    ajout fromage
    poids : 5.0 couleur : (255, 0, 0) odeur : 0.5
    poids : 10.0 couleur : (0, 255, 0) odeur : 1
    poids : 15.0 couleur : (85.0, 170.0, 0.0) odeur : 0.8333333333333334
    ----------------------
    ajout gruyère
    gruyère, poids : 3.0 couleur : (150, 150, 0) odeur : 0.1
    gruyère, poids : 7.0 couleur : (150, 150, 0) odeur : 0.1
    gruyère, poids : 10.0 couleur : (150, 150, 0) odeur : 0.1
    ----------------------
    ajout fromage
    poids : 13.0 couleur : (34.61538461538461, 230.76923076923077, 0.0) odeur : 0.
↪7923076923076924
```

### 3.1.11 Constructions classiques

### Héritage

Le premier exemple est classique puisqu'il reprend le programme du paragraphe paragraphe_fonction_variable pour le réécrire avec des classes et éviter de passer des fonctions comme paramètre d'une autre fonction. La première classe définit le module des suivantes. La méthode `calcul` n'accepte qu'un seul paramètre `x` mais pourrait également prendre en compte des constantes si celles-ci sont renseignées via le constructeur de la classe. C'est l'avantage de cette solution déjà illustrée par les pièces normales et truquées.

<<<

```
class Fonction:
    def calcul(self, x):
        pass

    def calcul_n_valeur(self, l):
        res = [self.calcul(i) for i in l]
        return res


class Carre (Fonction):
    def calcul(self, x):
        return x*x


class Cube (Fonction):
    def calcul(self, x):
        return x*x*x


li = [0, 1, 2, 3]
print(li)   # affiche [0, 1, 2, 3]

l1 = Carre().calcul_n_valeur(li)  # l1 vaut [0, 1, 4, 9]
l2 = Cube().calcul_n_valeur(li)   # l2 vaut [0, 1, 8, 27]
print(l1)
print(l2)
```

>>>

```
    [0, 1, 2, 3]
    [0, 1, 4, 9]
    [0, 1, 8, 27]
```

La version suivante mélange héritage et méthodes envoyées comme paramètre à une fonction. Il est préférable d'éviter cette construction même si elle est très utilisée par les interfaces graphiques. Elle n'est pas toujours transposable dans tous les langages de programmation tandis que le programme précédent aura un équivalent dans tous les langages objet.

<<<

```
class Fonction:
    def calcul(self, x):
        pass


class Carre(Fonction):
    def calcul(self, x):
        return x*x
```

(suite sur la page suivante)

(suite de la page précédente)

```
class Cube(Fonction):
    def calcul(self, x):
        return x*x*x


def calcul_n_valeur(l, f):
    res = [f(i) for i in l]
    return res


l = [0, 1, 2, 3]
l1 = calcul_n_valeur(l, Carre().calcul)  # l1 vaut [0, 1, 4, 9]
l2 = calcul_n_valeur(l, Cube().calcul)  # l2 vaut [0, 1, 8, 27]
print(l1)
print(l2)
```

>>>

```
    [0, 1, 4, 9]
    [0, 1, 8, 27]
```

### Deux lignées d'héritages

C'est une configuration qui arrive fréquemment lorsqu'on a d'un côté différentes structures pour les mêmes données et de l'autre différents algorithmes pour le même objectif. Par exemple, une matrice peut être définie comme une liste de listes ou un dictionnaire de tuples (voir exercice exercice_dame_dico_matrice). La multiplication de deux matrices peut être une multiplication classique ou celle de Strassen [172]. L'algorithme de multiplication de deux matrices de Strassen est plus rapide qu'une multiplication classique pour de grandes matrices. Son coût est en $O(n^{\frac{\ln 7}{\ln 2}}) \sim O(n^{2,807})$ au lieu de $O(n^3)$.

La question est comment modéliser ces deux structures et ces deux multiplications sachant que les quatre paires *structure - algorithme* doivent fonctionner. On pourrait simplement créer deux classes faisant référence aux deux structures différentes et à l'intérieur de chacune d'entre elles, avoir deux méthodes de multiplication. Néanmoins, si une nouvelle structure ou un nouvel algorithme apparaît, la mise à jour peut être fastidieuse.

Il est conseillé dans ce cas d'avoir quatre classes et de définir une interface d'échanges communes. L'algorithme de multiplication ne doit pas savoir quelle structure il manipule : il doit y accéder par des méthodes. De cette manière, c'est la classe qui indique l'algorithme choisi et non une méthode. Ajouter un troisième algorithme ou une troisième structure revient à ajouter une classe : l'interface d'échange ne change pas. Le programme pourrait suivre le schéma qui suit.

```
class Matrice :
    def __init__ (self,lin,col,coef):
        self.lin, self.col = lin, col

    # interface d'échange
    def get_lin () : return self.lin
    def get_col () : return self.col
    def __getitem__(self,i,j): pass
    def __setitem__(self,i,j,v): pass
    def get_submat(self, i1,j1,i2,j2): pass
    def set_submat(self, i1,j1,mat): pass
    # fin de l'interface d'échange
```

(suite sur la page suivante)

172. https://fr.wikipedia.org/wiki/Algorithme_de_Strassen

(suite de la page précédente)

```
    def trace (self) :
        t = 0
        for i in range(0, self.lin):
            t += self (i,i)
        return t

class MatriceList (Matrice) :
    def __init__ (self,lin,col,coef):
        Matrice.__init__ (self, \
                      lin, col, coef)
        #...

    def __getitem__ (self, i,j) : #...
    def __setitem__ (self, i,j, v) : #...
    def get_submat(self, i1,j1,i2,j2): #...
    def set_submat(self, i1,j1,mat): #...

class MatriceDict (Matrice) :
    def __init__ (self,lin,col,coef):
        Matrice.__init__ (self, \
                       lin, col, coef)
        #...

    def __getitem__ (self, i,j) : #...
    def __setitem__ (self, i,j, v) : #...
    def get_submat(self, i1,j1,i2,j2): #...
    def set_submat(self, i1,j1,mat): #...
```

Illustration :

```
class Produit :
    def calcul (self, mat1, mat2):
        pass

class ProduitClassique (Produit) :
    def calcul (self, mat1, mat2):
        #...
        return

class ProduitStrassen (Produit) :
    def calcul (self, mat1,mat2):
        #...
        return
```

Cette construction autorise même la multiplication de matrices de structures différentes. Très répandue, cette architecture est souvent plus coûteuse au moment de la conception car il faut bien penser l'interface d'échange mais elle l'est beaucoup moins par la suite. Il existe d'autres assemblages de classes assez fréquents, regroupés sous le terme de Design Patterns [173]. Pour peu que ceux-ci soient connus de celui qui conçoit un programme, sa relecture et sa compréhension en sont facilitées si les commentaires font mention du *pattern* utilisé.

173. https://fr.wikipedia.org/wiki/Patron_de_conception

CHAPITRE 4

# Exceptions

## 4.1 Exceptions


— *Principe des exceptions* (page 156)
  — *Attraper toutes les erreurs* (page 156)
  — *Obtenir le type d'erreur, attraper un type d'exception* (page 158)
  — *Lancer une exception* (page 160)
  — *Héritage et exception* (page 161)
  — *Instructions `try`, `except` imbriquées* (page 162)
— *Définir ses propres exceptions* (page 163)
  — *Dériver une classe d'exception* (page 163)
  — *Personnalisation d'une classe d'exception* (page 164)
— *Exemples d'utilisation des exceptions* (page 165)
  — *Les itérateurs* (page 165)
  — *Exception ou valeur aberrante* (page 165)
  — *Le piège des exceptions* (page 166)


Le petit programme suivant déclenche une erreur parce qu'il effectue une division par zéro.

```
x = 0
y = 1.0 / x
```

Il déclenche une erreur ou ce qu'on appelle une *exception* :

```
Traceback (most recent call last):
  File "cours.py", line 2, in ?
    y = 1.0 / x
ZeroDivisionError: float division
```

Le mécanisme des exceptions permet au programme de ńărattraperă" les erreurs, de détecter qu'une erreur s'est produite et d'agir en conséquence afin que le programme ne s'arrête pas.

### 4.1.1 Principe des exceptions

#### Attraper toutes les erreurs

Une exception est un objet qui indique que le programme ne peut continuer son exécution. Le type de l'exception donne une indication sur le type de l'erreur rencontrée. L'exception contient généralement un message plus détaillé. Toutes les exceptions hérite du type Exception [174].

On décide par exemple qu'on veut rattraper toutes les erreurs du programme et afficher un message d'erreur. Le programme suivant appelle la fonction qui retourne l'inverse d'un nombre.

```
def inverse (x):
    y = 1.0 / x
    return y

a = inverse(2)
print(a)
b = inverse(0)
print(b)
```

Lorsque `x == 0`, le programme effectue une division par zéro et déclenche une erreur. L'interpréteur Python affiche ce qu'on appelle la *pile d'appels* ou pile d'exécution [175]. La pile d'appel permet d'obtenir la liste de toutes les fonctions pour remonter jusqu'à celle où l'erreur s'est produite.

```
Traceback (most recent call last):
  File "cours.py", line 8, in ?
    b = inverse (0)
  File "cours.py", line 3, in inverse
    y = 1.0 / x
ZeroDivisionError: float division
```

Afin de rattraper l'erreur, on insère le code susceptible de produire une erreur entre les mots clés try [176] et except [177].

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    a = inverse(2)
    print(a)
    b = inverse(0)   # déclenche une exception
    print(b)
except:
    print("le programme a déclenché une erreur")
```

>>>

```
    0.5
    le programme a déclenché une erreur
```

Le programme essaye d'exécuter les quatre instructions incluses entre les instructions `try` et `except`. Si une erreur se produit, le programme exécute alors les lignes qui suivent l'instruction `except`. L'erreur se produit en fait à l'intérieur

174. https://docs.python.org/3/library/exceptions.html#Exception
175. https://fr.wikipedia.org/wiki/Pile_d%27ex%C3%A9cution
176. https://docs.python.org/3/reference/compound_stmts.html#try
177. https://docs.python.org/3/reference/compound_stmts.html#except

de la fonction mais celle-ci est appelée à l'intérieur d'un code ńăprotégéă" contre les erreurs. Ceci explique les lignes affichées par le programme. Il est aussi possible d'ajouter une clause qui sert de préfixe à une liste d'instructions qui ne sera exécutée que si aucune exception n'est déclenchée.

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    print(inverse(2))  # pas d'erreur
    print(inverse(1))  # pas d'erreur non plus
except:
    print("le programme a déclenché une erreur")
else:
    print("tout s'est bien passé")
```

>>>

```
    0.5
    1.0
    tout s'est bien passé
```

Ce dernier programme attrape l'erreur et affiche un message. Ce programme ne s'arrête jamais, il ne *plante* jamais. Pour résumer, la syntaxe suivante permet d'attraper toutes les erreurs qui se produisent pendant l'exécution d'une partie du programme. Cette syntaxe permet en quelque sorte de protéger cette partie du programme contre les erreurs.

```
try:
    # ... instructions à protéger
except:
    # ... que faire en cas d'erreur
else:
    # ... que faire lorsque aucune erreur n'est apparue
```

Toute erreur déclenchée alors que le programme exécute les instructions qui suivent le mot-clé `try` déclenche immédiatement l'exécution des lignes qui suivent le mot-clé `except`. Dans le cas contraire, le programme se poursuit avec l'exécution des lignes qui suivent le mot-clé `else`. Cette dernière partie est facultative, la clause `else` peut ou non être présente. Le bout de code prévoit ce qu'il faut faire dans n'importe quel cas.

Lorsqu'une section de code est protégée contre les exceptions, son exécution s'arrête à la première erreur d'exécution. Le reste du code n'est pas exécuté. Par exemple, dès la première erreur qui correspond au calcul d'une puissance non entière d'un nombre négatif, l'exécution du programme suivant est dirigée vers l'instruction qui suit le mot-clé `except`.

```
def inverse (x):
    y = 1.0 / x
    return y

try:
    print((-2.1) ** 3.1)   # première erreur
    print(inverse(2))
    print(inverse(0))      # cette ligne produirait une erreur
                           # mais le programme n'arrive jamais jusqu'ici
except:
    print("le programme a déclenché une erreur")
```

### Obtenir le type d'erreur, attraper un type d'exception

Parfois, plusieurs types d'erreurs peuvent être déclenchés à l'intérieur d'une portion de code protégée. Pour avoir une information sur ce type, il est possible de récupérer une variable de type Exception [178].

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    print(inverse(2))
    print(inverse(0))
except Exception as exc:
    print("exception de type ", exc.__class__)
    # affiche exception de type  exceptions.ZeroDivisionError
    print("message", exc)
    # affiche le message associé à l'exception
```

>>>

```
    0.5
    exception de type  <class 'ZeroDivisionError'>
    message float division by zero
```

Le programme précédent récupère une exception sous la forme d'une variable appelée . Cette variable est en fait une instance d'une classe d'erreur, `__class__` correspond au nom de cette classe. A l'aide de la fonction `isinstance`, il est possible d'exécuter des traitements différents selon le type d'erreur.

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    print((-2.1) ** 3.1)   # première erreur
    print(inverse(2))
    print(inverse(0))      # seconde erreur
except Exception as exc:
    if isinstance(exc, ZeroDivisionError):
        print("division par zéro")
    else:
        print("erreur insoupçonnée :", exc.__class__)
        print("message", exc)
```

>>>

```
    (-9.48606594010979-3.0822096637057887j)
    0.5
    division par zéro
```

L'exemple précédent affiche le message qui suit parce que la première erreur intervient lors du calcul de `(-2.1) ** 3.1`. Une autre syntaxe plus simple permet d'attraper un type d'exception donné en accolant au mot-clé `except` le type de l'exception qu'on désire attraper. L'exemple précédent est équivalent au suivant mais syntaxiquement différent.

178. https://docs.python.org/3/library/exceptions.html

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    print((-2.1) ** 3.1)
    print(inverse(2))
    print(inverse(0))
except ZeroDivisionError:
    print("division par zéro")
except Exception as exc:
    print("erreur insoupçonnée :", exc.__class__)
    print("message ", exc)
```

>>>

```
    (-9.48606594010979-3.0822096637057887j)
    0.5
    division par zéro
```

Cette syntaxe obéit au schéma qui suit.

**Syntaxe S1 : Attraper une exception**

```
try:
    # ... instructions à protéger
except type_exception_1:
    # ... que faire en cas d'erreur de type type_exception_1
except (type_exception_i, type_exception_j):
    # ... que faire en cas d'erreur de type type_exception_i ou type_exception_j
except type_exception_n:
    # ... que faire en cas d'erreur de type type_exception_n
except:
    # ... que faire en cas d'erreur d'un type différent de tous
    #     les précédents types
else:
    # ... que faire lorsque une erreur aucune erreur n'est apparue
```

Toute erreur déclenchée alors que le programme exécute les instructions qui suivent le mot-clé `try` déclenche immédiatement l'exécution des lignes qui suivent un mot-clé `except`. Le programme compare le type d'exception aux types `type_exception_1` à `type_exception_n`. S'il existe une correspondance alors ce sont les instructions de la clause `except` associée qui seront exécutées et uniquement ces instructions. La dernière clause `except` est facultative, elle est utile lorsque aucun type de ceux prévus ne correspond à l'exception générée. La clause `else` est aussi facultative. Si la dernière clause `except` n'est pas spécifiée et que l'exception déclenchée ne correspond à aucune de celle listée plus haut, le programme s'arrête sur cette erreur à moins que celle-ci ne soit attrapée plus tard.

Le langage Python propose une liste d¿exceptions standards [179]. Lorsqu'une erreur ne correspond pas à l'une de ces exceptions, il est possible de créer une exception propre à un certain type d'erreur. Lorsqu'une fonction ou une méthode déclenche une exception non standard, généralement, le commentaire qui lui est associé l'indique. Quelques types d'exception courantes :

— AttributeError [180] : Une référence à un attribut inexistant ou une affectation a échoué.

179. https://docs.python.org/3/library/exceptions.html#base-classes
180. https://docs.python.org/3/library/exceptions.html#AttributeError

— OSError [181] : Une opération concernant les entrées/sorties (Input/Output) a échoué. Cette erreur survient par exemple lorsqu'on cherche à lire un fichier qui n'existe pas.
— ImportError [182] : Cette erreur survient lorsqu'on cherche à importer un module qui n'existe pas.
— IndentationError [183] : L'interpréteur ne peut interpréter une partie du programme à cause d'un problème d'indentation. Il n'est pas possible d'exécuter un programme mal indenté mais cette erreur peut se produire lors de l'utilisation de la fonction compile [184].
— IndexError [185] : On utilise un index erroné pour accéder à un élément d'une liste, d'un dictionnaire ou de tout autre tableau.
— KeyError [186] : Une clé est utilisée pour accéder à un élément d'un dictionnaire dont elle ne fait pas partie.
— NameError [187] : On utilise une variable, une fonction, une classe qui n'existe pas.
— TypeError [188] Erreur de type, une fonction est appliquée sur un objet qu'elle n'est pas censée manipuler.
— UnicodeError [189] Erreur de conversion d'un encodage [190] de texte à un autre.
— ValueError [191] : Cette exception survient lorsqu'une valeur est inappropriée pour une certaine opération, par exemple, l'obtention du logarithme d'un nombre négatif.

### Lancer une exception

Lorsqu'une fonction détecte une erreur, il lui est possible de déclencher une exception par l'intermédiaire du mot-clé `raise`. La fonction `inverse` compare `x` à `0` et déclenche l'exception `ValueError` si `x` est nul. Cette exception est attrapée plus bas.

<<<

```
def inverse(x):
    if x == 0:
        raise ValueError
    y = 1.0 / x
    return y


try:
    print(inverse(0))  # erreur
except ValueError:
    print("erreur de type ValueError")
```

>>>

```
    erreur de type ValueError
```

Il est parfois utile d'associer un message à une exception afin que l'utilisateur ne soit pas perdu. Le programme qui suit est identique au précédent à ceci près qu'il associe à l'exception `ValueError` qui précise l'erreur et mentionne la fonction où elle s'est produite. Le message est ensuite intercepté plus bas.

<<<

```
def inverse(x):
    if x == 0:
```

(suite sur la page suivante)

181. https://docs.python.org/3/library/exceptions.html#OSError
182. https://docs.python.org/3/library/exceptions.html#ImportError
183. https://docs.python.org/3/library/exceptions.html#IndentationError
184. https://docs.python.org/3/library/functions.html?highlight=compile#compile
185. https://docs.python.org/3/library/exceptions.html#IndexError
186. https://docs.python.org/3/library/exceptions.html#KeyError
187. https://docs.python.org/3/library/exceptions.html#NameError
188. https://docs.python.org/3/library/exceptions.html#TypeError
189. https://docs.python.org/3/library/exceptions.html#UnicodeError
190. https://fr.wikipedia.org/wiki/Codage_des_caract%C3%A8res
191. https://docs.python.org/3/library/exceptions.html#ValueError

(suite de la page précédente)

```
        raise ValueError("valeur nulle interdite, fonction inverse")
    y = 1.0 / x
    return y


try:
    print(inverse(0))  # erreur
except ValueError as exc:
    print("erreur, message :", exc)
```

>>>

```
    erreur, message : valeur nulle interdite, fonction inverse
```

Le déclenchement d'une exception suit la syntaxe suivante.

**Syntaxe S2 : Lever une exception**

```
raise exception_type(message)
```

Cette instruction lance l'exception `exception_type` associée au message `message`. Le message est facultatif, lorsqu'il n'y en a pas, la syntaxe se résume à `raise exception_type`.

Et pour attraper cette exception et le message qui lui est associé, il faut utiliser la syntaxe décrite au paragraphe précédent.

### Héritage et exception

L'instruction `help(ZeroDivisionError)` retourne l'aide associée à l'exception `ZeroDivisionError`. Celle-ci indique que l'exception `ZeroDivisionError` est en fait un cas particulier de l'exception `ArithmeticError`, elle-même un cas particulier de `StandardError`.

```
class ZeroDivisionError(ArithmeticError)
    |  Second argument to a division or modulo operation was zero.
    |
    |  Method resolution order:
    |      ZeroDivisionError
    |      ArithmeticError
    |      StandardError
    |      Exception
```

Toutes les exceptions sont des cas particuliers de l'exception de type `Exception`. C'est pourquoi l'instruction `except Exception:` attrape toutes les exceptions. L'instruction `except ArithmeticError:` attrape toutes les erreurs de type `ArithmeticError`, ce qui inclut les erreurs de type `ZeroDivisionError`. Autrement dit, toute exception de type `ZeroDivisionError` est attrapée par les instructions suivantes :

```
except ZeroDivisionError:
except ArithmeticError:
except StandardError:
except Exception:
```

Plus précisément, chaque exception est une classe qui dérive directement ou indirectement de la classe `Exception`. L'instruction `except ArithmeticError :` par exemple attrape toutes les exceptions de type `ArithmeticError` et toutes celles qui en dérivent comme la classe `ZeroDivisionError`.

### Instructions `try`, `except` imbriquées

Comme pour les boucles, il est possible d'imbriquer les portions protégées de code les unes dans les autres. Dans l'exemple qui suit, la première erreur est l'appel à une fonction non définie, ce qui déclenche l'exception `NameError`.

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    try:
        print(inverses(0))   # fonction inexistante --> exception NameError
        print(inverse(0))    # division par zéro --> ZeroDivisionError
    except NameError:
        print("appel à une fonction non définie")
except ZeroDivisionError as exc:
    print("erreur", exc)
```

>>>

```
    appel à une fonction non définie
```

En revanche, dans le second exemple, les deux lignes `print(inverse(0))` et `print(inverses(0))` ont été permutées. La première exception déclenchée est la division par zéro. La première clause `except` n'interceptera pas cette erreur puisqu'elle n'est pas du type recherché.

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    try:
        print(inverse(0))    # division par zéro --> ZeroDivisionError
        print(inverses(0))   # fonction inexistante --> exception NameError
    except NameError:
        print("appel à une fonction non définie")
except ZeroDivisionError as exc:
    print("erreur", exc)
```

>>>

```
    erreur float division by zero
```

Une autre imbrication possible est l'appel à une fonction qui inclut déjà une partie de code protégée. L'exemple suivant appelle la fonction `inverse` qui intercepte les exceptions de type `ZeroDivisionError` pour retourner une grande valeur lorsque `x=0`. La seconde exception générée survient lors de l'appel à la fonction `inverses` qui déclenche l'exception `NameError`, elle aussi interceptée.

<<<

```
def inverse(x):
    try:
        y = 1.0 / x
    except ZeroDivisionError as exc:
```

(suite sur la page suivante)

(suite de la page précédente)

```
        print("erreur ", exc)
        if x > 0:
            return 1000000000
        else:
            return -1000000000
    return y


try:
    print(inverse(0))    # division par zéro    --> la fonction inverse sait gérer
    print(inverses(0))   # fonction inexistante --> exception NameError
except NameError:
    print("appel à une fonction non définie")
```

>>>

```
    erreur  float division by zero
    -1000000000
    appel à une fonction non définie
```

### 4.1.2 Définir ses propres exceptions

#### Dériver une classe d'exception

Pour définir sa propre exception, il faut créer une classe qui dérive d'une classe d'exception existante par exemple, la classe `Exception`. L'exemple suivant crée une exception `AucunChiffre` qui est lancée par la fonction `conversion` lorsque la chaîne de caractères qu'elle doit convertir ne contient pas que des chiffres.

<<<

```
class AucunChiffre(Exception):
    """
    chaîne de caractères contenant aussi autre chose que des chiffres
    """
    pass


def conversion(s):
    """
    conversion d'une chaîne de caractères en entier
    """
    if not s.isdigit():
        raise AucunChiffre(s)
    return int(s)


try:
    s = "123a"
    print(s, " = ", conversion(s))
except AucunChiffre as exc:
    # on affiche ici le commentaire associé à la classe d'exception
    # et le message associé
    print(AucunChiffre.__doc__, " : ", exc)
```

>>>

```
            chaîne de caractères contenant aussi autre chose que des chiffres
              :  123a
```

En redéfinissant l'opérateur `__str__` d'une exception, il est possible d'afficher des messages plus explicites avec la seule instruction `print`.

```
class AucunChiffre(Exception):
    """
    chaîne de caractères contenant aussi autre chose que des chiffres
    """
    def __str__ (self):
        return "{0} {1}".format(self.__doc__, Exception.__str__(self))
```

### Personnalisation d'une classe d'exception

Il est parfois utile qu'une exception contienne davantage d'informations qu'un simple message. L'exemple suivant reprend l'exemple du paragraphe précédent. L'exception `AucunChiffre` inclut cette fois-ci un paramètre supplémentaire contenant le nom de la fonction où l'erreur a été déclenchée.

La classe `AucunChiffre` possède dorénavant un constructeur qui doit recevoir deux paramètres : une valeur et un nom de fonction. L'exception est levée à l'aide de l'instruction `raise AucunChiffre(s, "conversion")` qui regroupe dans un T-uple les paramètres à envoyer à l'exception.

L'opérateur `__str__` a été modifié de façon à ajouter ces deux informations dans le message associé à l'exception. Ainsi, l'instruction `print(exc)` présente à l'avant dernière ligne de cet exemple affiche un message plus complet.

<<<

```
class AucunChiffre(Exception):
    """
    chaîne de caractères contenant aussi autre chose que des chiffres
    """

    def __init__(self, s, f=""):
        Exception.__init__(self, s)
        self.s = s
        self.f = f

    def __str__(self):
        return "exception AucunChiffre, depuis la fonction {0} avec le paramètre {1}".
↪format(self.f, self.s)


def conversion(s):
    """
    conversion d'une chaîne de caractères en entier
    """
    if not s.isdigit():
        raise AucunChiffre(s, "conversion")
    return int(s)


try:
    s = "123a"
    i = conversion(s)
    print(s, " = ", i)
```

(suite sur la page suivante)

(suite de la page précédente)

```
except AucunChiffre as exc:
    print(exc)
    print("fonction : ", exc.f)
```

>>>

```
    exception AucunChiffre, depuis la fonction conversion avec le paramètre 123a
    fonction :  conversion
```

Etant donné que le programme déclenche une exception dans la section de code protégée, les deux derniers affichages sont les seuls exécutés correctement. Ils produisent les deux lignes qui suivent. %

### 4.1.3 Exemples d'utilisation des exceptions

#### Les itérateurs

Les itérateurs sont des outils qui permettent de parcourir des objets qui sont des ensembles, comme une liste, un dictionnaire. Ils fonctionnent toujours de la même manière. L'exemple déjà présenté au chapitre *Itérateurs* (page 113) et repris en partie ici définit une classe contenant trois coordonnées, ainsi qu'un itérateur permettant de parcourir ces trois coordonnées. Arrivée à la troisième itération, l'exception StopIteration [192] est déclenchée. Cette exception indique à une boucle `for` de s'arrêter.

```
class point_espace:

    # ...

    class class_iter:
        def __init__(self, ins):
            self._n   = 0
            self._ins = ins
        def __iter__(self) :
            return self
        def next(self):
            if self._n <= 2:
                v = self._ins[self._n]
                self._n += 1
                return v
            else:
                raise StopIteration

    def __iter__(self):
        return point_espace.class_iter(self)
```

Cet exemple montre seulement que les exceptions n'interviennent pas seulement lors d'erreurs mais font parfois partie intégrante d'un algorithme.

#### Exception ou valeur aberrante

Sans exception, une solution pour indiquer un cas de mauvaise utilisation d'une fonction est de retourner une valeur aberrante. Retourner `-1` pour une fonction dont le résultat est nécessairement positif est une valeur aberrante. Cette convention permet de signifier à celui qui appelle la fonction que son appel n'a pu être traité correctement. Dans l'exemple qui suit, la fonction `racine_carree` retourne un couple de résultats, `True` ou `False` pour savoir si le calcul est possible, suivi du résultat qui n'a un sens que si `True` est retournée en première valeur.

192. https://docs.python.org/3/library/exceptions.html#StopIteration

<<<

```
def racine_carree(x):
    if x < 0:
        return False, 0
    else:
        return True, x ** 0.5


print(racine_carree(-1))   # (False, 0)
print(racine_carree(1))    # (True, 1.0)
```

>>>

```
    (False, 0)
    (True, 1.0)
```

Plutôt que de compliquer le programme avec deux résultats ou une valeur aberrante, on préfère souvent déclencher une exception, ici, `ValueError`. La plupart du temps, cette exception n'est pas déclenchée. Il est donc superflu de retourner un couple plutôt qu'une seule valeur.

```
def racine_carree(x) :
    if x < 0:
        raise ValueError("valeur négative")
    return x ** 0.5

print(racine_carree(-1))   # déclenche une exception
print(racine_carree(1))
```

### Le piège des exceptions

Ce paragraphe évoque certains problèmes lorsqu'une exception est levée. L'exemple utilise les fichiers décrits au chapitre *Fichiers* (page 173). Lorsqu'une exception est levée à l'intérieur d'une fonction, l'exécution de celle-ci s'interrompt. Si l'exception est attrapée, le programme continue sans problème ; les objets momentanément créés seront détruits par le garbage collector [193]. Il faut pourtant faire attention dans le cas par exemple où l'exception est levée alors qu'un fichier est ouvert : il ne sera pas fermé.

```
for i in range(0, 5):
    try :
        x, y = i-1, i-2
        print("{}/{}".format(x, y))
        f = open("essai.txt", "a")
        f.write("{}/{}=".format(x, y))
        f.write(str((float (x)/y)) + "\n" )      # exception si y == 0
        f.close()
    except Exception as e:
        print("erreur avec i = ", i, ",", e, f.closed)
```

Les écritures dans le fichier se font en mode ajout `"a"`, le fichier `"essai.txt"` contiendra tout ce qui aura été écrit.

193. https://docs.python.org/3/library/gc.html

| affichage | fichier |
|---|---|
| -1/-2<br>0/-1<br>1/0<br>erreur avec i = 2 , float division by zero False<br>2/1<br>3/2 | -1/-2=0.5<br>0/-1=-0.0<br>1/0=2/1=2.0<br>3/2=1.5 |

La troisième ligne du fichier est tronquée puisque l'erreur est intervenue juste avant l'affichage. On voit aussi que `f.closed` est faux. Cela signifie que le fichier n'est pas fermé. Pour se prémunir contre les exceptions lorsqu'on écrit un fichier, il faut utiliser le mot clé with [194] :

```
for i in range(0, 5):
    try :
        x, y = i-1, i-2
        print("{}/{}".format(x, y))
        with open("essai.txt", "a") as f:
            f.write("{}/{}=".format(x, y))
            f.write(str((float (x)/y)) + "\n" )      # exception si y == 0
    except Exception as e:
        print("erreur avec i = ", i, ",", e, f.closed)
```

Pour en savoir un peu plus : Les context managers et le mot clé with en Python [195].

## 4.2 Usage

### 4.2.1 Pile d'appel

La pile d'appel [196] (ou *pile d'exécution* ou *call stack*) mémorise les appels de fonctions. La premièrel ligne est le point d'entrée [197] du programme. La suivante est la seconde fonction appelée. Si celle-ci en appelle une autre, une autre ligne est ajoutée et celle-ci demeure jusqu'à ce qu'elle est terminée son exécution. A chaque instant, la dernière ligne est la fonction en train de s'exécuter, les lignes précédentes définissent le chemin que l'ordinateur a suivi pour arriver jusque là.

<<<

```
import traceback
import sys


def foncA():
    print("foncA begin")
    foncB()
    print("foncA end")


def foncB():
```

(suite sur la page suivante)

194. https://www.python.org/dev/peps/pep-0343/
195. http://sametmax.com/les-context-managers-et-le-mot-cle-with-en-python/
196. https://fr.wikipedia.org/wiki/Pile_d%27ex%C3%A9cution
197. https://fr.wikipedia.org/wiki/Point_d%27entr%C3%A9e

(suite de la page précédente)

```
    print("foncB begin")
    foncC()
    print("foncB end")


def foncC():
    print("foncC begin")
    try:
        raise Exception("erreur volontaire")
    except Exception:
        print("Erreur")
        print("\n".join(traceback.format_stack()))
    print("foncC end")


foncA()
```

>>>

```
    foncA begin
    foncB begin
    foncC begin
    Erreur
      File "/var/lib/jenkins/workspace/teachpyx/teachpyx_UT_37_std/_venv/lib/python3.
↪7/site-packages/pyquickhelper/helpgen/process_sphinx_cmd.py", line 23, in <module>
        main()

      File "/var/lib/jenkins/workspace/teachpyx/teachpyx_UT_37_std/_venv/lib/python3.
↪7/site-packages/pyquickhelper/helpgen/process_sphinx_cmd.py", line 19, in main
        run_sphinx_build(sys.argv)

      File "/var/lib/jenkins/workspace/teachpyx/teachpyx_UT_37_std/_venv/lib/python3.
↪7/site-packages/pyquickhelper/helpgen/process_sphinx_cmd.py", line 15, in run_
↪sphinx_build
        build_main(argv=argv[1:])

      File "/usr/local/lib/python3.7/site-packages/sphinx/cmd/build.py", line 38, in␣
↪main
        return build_main(argv)

      File "/usr/local/lib/python3.7/site-packages/sphinx/cmd/build.py", line 23, in␣
↪build_main
        return cmdline.main(argv)  # type: ignore

      File "/usr/local/lib/python3.7/site-packages/sphinx/cmdline.py", line 304, in␣
↪main
        app.build(args.force_all, filenames)

      File "/usr/local/lib/python3.7/site-packages/sphinx/application.py", line 331,␣
↪in build
        self.builder.build_update()

      File "/usr/local/lib/python3.7/site-packages/sphinx/builders/__init__.py", line␣
↪337, in build_update
        self.build(['__all__'], to_build)

      File "/usr/local/lib/python3.7/site-packages/sphinx/builders/__init__.py", line␣
↪355, in build
```

(suite sur la page suivante)

(suite de la page précédente)

```
        updated_docnames = set(self.env.update(self.config, self.srcdir, self.
↪doctreedir))

      File "/usr/local/lib/python3.7/site-packages/sphinx/environment/__init__.py",␣
↪line 565, in update
        self._read_serial(docnames, self.app)

      File "/usr/local/lib/python3.7/site-packages/sphinx/environment/__init__.py",␣
↪line 584, in _read_serial
        self.read_doc(docname, app)

      File "/usr/local/lib/python3.7/site-packages/sphinx/environment/__init__.py",␣
↪line 659, in read_doc
        doctree = read_doc(self.app, self, self.doc2path(docname))

      File "/usr/local/lib/python3.7/site-packages/sphinx/io.py", line 302, in read_
↪doc
        pub.publish()

      File "/usr/local/lib/python3.7/site-packages/docutils/core.py", line 217, in␣
↪publish
        self.settings)

      File "/usr/local/lib/python3.7/site-packages/docutils/readers/__init__.py",␣
↪line 72, in read
        self.parse()

      File "/usr/local/lib/python3.7/site-packages/docutils/readers/__init__.py",␣
↪line 78, in parse
        self.parser.parse(self.input, document)

      File "/usr/local/lib/python3.7/site-packages/sphinx/parsers.py", line 85, in␣
↪parse
        self.statemachine.run(inputstring, document, inliner=self.inliner)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 171, in run
        input_source=document['source'])

      File "/usr/local/lib/python3.7/site-packages/docutils/statemachine.py", line␣
↪239, in run
        context, state, transitions)

      File "/usr/local/lib/python3.7/site-packages/docutils/statemachine.py", line␣
↪460, in check_line
        return method(match, context, next_state)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 2988, in text
        self.section(title.lstrip(), source, style, lineno + 1, messages)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 327, in section
        self.new_subsection(title, lineno, messages)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 395, in new_subsection
```

(suite sur la page suivante)

(suite de la page précédente)

```
        node=section_node, match_titles=True)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 282, in nested_parse
        node=node, match_titles=match_titles)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 196, in run
        results = StateMachineWS.run(self, input_lines, input_offset)

      File "/usr/local/lib/python3.7/site-packages/docutils/statemachine.py", line␣
↪239, in run
        context, state, transitions)

      File "/usr/local/lib/python3.7/site-packages/docutils/statemachine.py", line␣
↪460, in check_line
        return method(match, context, next_state)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 2753, in underline
        self.section(title, source, style, lineno - 1, messages)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 327, in section
        self.new_subsection(title, lineno, messages)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 395, in new_subsection
        node=section_node, match_titles=True)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 282, in nested_parse
        node=node, match_titles=match_titles)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 196, in run
        results = StateMachineWS.run(self, input_lines, input_offset)

      File "/usr/local/lib/python3.7/site-packages/docutils/statemachine.py", line␣
↪239, in run
        context, state, transitions)

      File "/usr/local/lib/python3.7/site-packages/docutils/statemachine.py", line␣
↪460, in check_line
        return method(match, context, next_state)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 2326, in explicit_markup
        nodelist, blank_finish = self.explicit_construct(match)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 2338, in explicit_construct
        return method(self, expmatch)

      File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 2081, in directive
        directive_class, match, type_name, option_presets)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    File "/usr/local/lib/python3.7/site-packages/docutils/parsers/rst/states.py",␣
↪line 2130, in run_directive
      result = directive_instance.run()

    File "/var/lib/jenkins/workspace/teachpyx/teachpyx_UT_37_std/_venv/lib/python3.
↪7/site-packages/pyquickhelper/sphinxext/sphinx_runpython_extension.py", line 466,␣
↪in run
      chdir=cs_source_dir if p['current'] else None)

    File "/var/lib/jenkins/workspace/teachpyx/teachpyx_UT_37_std/_venv/lib/python3.
↪7/site-packages/pyquickhelper/sphinxext/sphinx_runpython_extension.py", line 196,␣
↪in run_python_script
      exec(obj, globals(), loc)

    File "", line 26, in <module>

    File "", line 25, in run_python_script_139651618353872

    File "", line 8, in foncA

    File "", line 13, in foncB

    File "", line 22, in foncC

  foncC end
  foncB end
  foncA end
```

### Récupération de la pile d'appel

Le module traceback [198] permet de récupérer la pile d'appels lorsqu'une exception survient.

```
def raise_exception():
    raise Exception("an error was raised")

try:
    insidefe()
except:
    exc_type, exc_value, exc_traceback = sys.exc_info()
    print("".join(traceback.format_tb(exc_traceback)))
```

Le programme affiche :

```
File "test_faq_exception.py", line 57, in raise_exception
  raise Exception("an error was raised")
```

Il est possible de récupérer la liste des appels de fonctions avec la fonction extract_tb [199]. Cette information est précieuse pour écrire un test qui vérifie qu'une erreur s'est bien produite à un endroit particulier, de détecter les cas particuliers comme les boucles infinies ou d'améliorer un message d'erreur en cas de besoin (lire How do I write Flask's excellent debug log message to a file in production ? [200]).

198. https://docs.python.org/3/library/traceback.html
199. https://docs.python.org/3/library/traceback.html#traceback.extract_tb
200. http://stackoverflow.com/questions/14037975/how-do-i-write-flasks-excellent-debug-log-message-to-a-file-in-production

#### Message d'erreur plus explicite

Lorsqu'une erreur se produit dans une librairie de Python, le message ne mentionne aucune information à propos du code qui l'a provoquée.

```
import math
ensemble = [1, 0, 2]
s = 0
for e in ensemble:
    s += math.log(e)
```

```
Traceback (most recent call last):
  File "i.py", line 5, in <module>
    s += math.log(e)
ValueError: math domain error
```

Typiquement dans ce cas précis, on ne sait pas quel est l'indice de l'élément qui a provoqué l'erreur. On utilise alors un mécanisme qui permet d'ajouter une erreur sans perdre les informations l'exception original

```
import math
ensemble = [1, 0, 2]
s = 0
for i, e in enumerate(ensemble):
    try:
        s += math.log(e)
    except Exception as exc:
        raise Exception("Issue with element {0}".format(i)) from exc
```

La dernière partie de la dernière ligne est importante : `from exc`. Le langage garde ainsi la trace de la première exception.

```
Traceback (most recent call last):
  File "i.py", line 6, in <module>
    s += math.log(e)
ValueError: math domain error

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "i.py", line 8, in <module>
    raise Exception("Issue with element {0}".format(i)) from exc
Exception: Issue with element 1
```

### 4.2.2 Conventions

#### Erreur ou code d'erreur

**A faire : terminer la section Erreur ou code d'erreur**

parler aussi de coûts d'une exception, libération des ressources

CHAPITRE 5

# Entrées Sorties

## 5.1 Fichiers


— *Format texte* (page 174)
  — *Ecriture* (page 174)
  — *Ecriture en mode ńăajoută"* (page 175)
  — *Lecture* (page 176)
  — *Encoding et les accents* (page 178)
— *Fichiers zip* (page 178)
  — *Lecture* (page 178)
  — *Ecriture* (page 178)
— *Manipulation de fichiers* (page 180)
  — *Gestion des noms de chemins* (page 180)
  — *Copie, suppression* (page 181)
  — *Liste de fichiers* (page 181)
— *Format binaire* (page 208)
  — *Ecriture dans un fichier binaire* (page 209)
  — *Lecture d'un fichier binaire* (page 209)
  — *Exemple fichier binaire* (page 209)
  — *Objets plus complexes* (page 211)


Lorsqu'un programme termine son exécution, toutes les informations stockées dans des variables sont perdues. Un moyen de les conserver est de les enregistrer dans un fichier sur disque dur. A l'intérieur de celui-ci, ces informations peuvent apparaître sous un format texte qui est lisible par n'importe quel éditeur de texte, dans un format compressé, ou sous un autre format connu par le concepteur du programme. On appelle ce dernier type un format binaire [201], il est adéquat lorsque les données à conserver sont très nombreuses ou lorsqu'on désire que celles-ci ne puissent pas être lues par une autre application que le programme lui-même. En d'autres termes, le format binaire est illisible excepté pour celui qui l'a conçu.

Ce chapitre abordera pour commencer les formats texte, binaire et compressé (*zip*) directement manipulable depuis *python*. Les manipulations de fichiers suivront pour terminer sur les expressions régulières qui sont très utilisées pour

201. https://fr.wikipedia.org/wiki/Fichier_binaire

effectuer des recherches textuelles. A l'issue de ce chapitre, on peut envisager la recherche à l'intérieur de tous les documents textes présents sur l'ordinateur, de dates particulières, de tous les numéros de téléphones commençant par 06... En utilisant des modules tels que reportlab [202] ou encore openpyxl [203], il serait possible d'étendre cette fonctionnalité aux fichiers de type pdf [204] et aux fichiers Excel [205].

### 5.1.1 Format texte

Les fichiers texte [206] sont les plus simples : ce sont des suites de caractères. Le format HTML [207] et XML [208] font partie de cette catégorie. Ils servent autant à conserver des informations qu'à en échanger comme par exemple transmettre une matrice à Excel [209].

Ce format, même s'il est simple, implique une certaine organisation dans la façon de conserver les données afin de pouvoir les récupérer. Le cas le plus fréquent est l'enregistrement d'une matrice : on choisira d'écrire les nombres les uns à la suite des autres en choisissant un séparateur de colonnes et un séparateur de lignes. Ce point sera abordé à la fin de cette section. Les fichiers texte que les programmes informatiques manipulent sont souvent structurées.

#### Ecriture

La première étape est l'écriture. Les informations sont toujours écrites sous forme de chaînes de caractères et toujours ajoutées à la fin du fichier qui s'allonge jusqu'à ce que toutes les informations y soient écrites. L'écriture s'effectue toujours selon le même schéma.

1. création ou ouverture du fichier,
2. écriture,
3. fermeture.

Lors de l'ouverture, le fichier dans lequel seront écrites les informations est créé s'il n'existe pas ou nettoyé s'il existe déjà. La fermeture permet à d'autres programmes de lire ce que vous avez placé dans ce fichier. Sans cette dernière étape, il sera impossible d'y accéder à nouveau pour le lire ou y écrire à nouveau. A l'intérieur d'un programme informatique, écrire dans un fichier suit toujours le même schéma :

```
f = open ("nom-fichier", "w")    # ouverture

f.write ( s )    # écriture de la chaîne de caractères  s
f.write ( s2 )   # écriture de la chaîne de caractères  s2
...

f.close ()  # fermeture
```

Les étapes d'ouverture et de fermeture sont toujours présentes en ce qui concerne les fichiers. Il s'agit d'indiquer au système d'exploitation que le programme souhaite accéder à un fichier et interdire à tout autre programme l'accès à ce fichier. Un autre programme qui souhaiterait créer un fichier du même nom ne le pourrait pas tant que l'étape de fermeture n'est pas exécutée. En revanche, il pourrait tout à fait le lire car la lecture ne perturbe pas l'écriture.

Il faut donc toujours fermer le fichier à la fin pour indiquer que le fichier est accessible pour un autre usage. Un fichier est une ressource [210]. Et comme ce schéma se répète toujours, le langage *Python* a prévu une syntaxe avec le mot-clé with [211].

202. https://pypi.python.org/pypi/reportlab
203. https://pypi.python.org/pypi/openpyxl
204. https://fr.wikipedia.org/wiki/Portable_Document_Format
205. https://fr.wikipedia.org/wiki/Microsoft_Excel
206. https://fr.wikipedia.org/wiki/Fichier_texte
207. https://fr.wikipedia.org/wiki/Hypertext_Markup_Language
208. https://fr.wikipedia.org/wiki/Extensible_Markup_Language
209. https://fr.wikipedia.org/wiki/Microsoft_Excel
210. https://fr.wikipedia.org/wiki/Ressource_(informatique)
211. https://docs.python.org/3/reference/compound_stmts.html#the-with-statement

```
with open ("nom-fichier", "w") as f:     # ouverture

    f.write ( s )     # écriture de la chaîne de caractères  s
    f.write ( s2 )    # écriture de la chaîne de caractères  s2
    ...
```

L'instruction `f.close()` est implicite et automatiquement exécutée dès que le programme sort de la section *with*. Lorsque que le programme se termine, même s'il reste des fichiers ńăouvertsă" pour lesquels la méthode `close` n'a pas été appelée, ils seront automatiquement fermés.

Certains caractères sont fort utiles lors de l'écriture de fichiers texte afin d'organiser les données. Le symbole `;` est très utilisé comme séparateur de colonnes pour une matrice, on utilise également le passage à la ligne ou la tabulation. Comme ce ne sont pas des caractères ńăvisiblesă", ils ont des codes :

— `\n` : passage à la ligne
— `\t` : tabulation, indique un passage à la colonne suivante dans le format tsv [212] (Tabulation-separated values).

Il existe peu de manières différentes de conserver une matrice dans un fichier, le programme ressemble dans presque tous les cas à celui qui suit :

```
mat =  ... # matrice de type liste de listes
f = open ("mat.txt", "w")
for i in range (0,len (mat)) :                  # la fonction join est aussi
    for j in range (0, len (mat [i])) :         # fréquemment utilisée
        f.write ( str (mat [i][j]) + "\t")      # pour assembler les chaînes
    f.write ("\n")                              # un une seule et réduire le
f.close ()                                      # nombre d'appels à f.write
```

Ou encore :

```
mat =  ... # matrice de type liste de listes
with open ("mat.txt", "w") as f:
    for i in range (0,len (mat)) :
        for j in range (0, len (mat [i])) :
            f.write ( str (mat [i][j]) + "\t")
        f.write ("\n")
```

La fonction open [213] accepte deux paramètres, le premier est le nom du fichier, le second définit le mode d'ouverture : `"w"` pour écrire (**w**rite), ńăaă" pour écrire et ajouter (**a**ppend), ńără" pour lire (**r**ead). Ceci signifie que la fonction `open` sert à ouvrir un fichier quelque soit l'utilisation qu'on en fait.

A la première écriture dans un fichier (premier appel à la fonction `write`, la taille du fichier créée est souvent nulle. L'écriture dans un fichier n'est pas immédiate, le langage *python* attend d'avoir reçu beaucoup d'informations avant de les écrire physiquement sur le disque dur. Les informations sont placées dans un tampon ou buffer [214]. Lorsque le tampon est plein, il est écrit sur disque dur. Pour éviter ce délai, il faut soit fermer puis réouvrir le fichier soit appeler la méthode flush [215] qui ne prend aucun paramètre. Ce mécanisme vise à réduire le nombre d'accès au disque dur car selon les technologies, il n'est pas nécessairement beaucoup plus long d'y écrire un caractère plutôt que 1000 en une fois.

### Ecriture en mode ńăajoută"

Lorsqu'on écrit des informations dans un fichier, deux cas se présentent. Le premier consiste à ne pas tenir compte du précédent contenu de ce fichier lors de son ouverture pour écriture et à l'écraser. C'est le cas traité par le précédent paragraphe. Le second cas consiste à ajouter toute nouvelle information à celles déjà présentes lors de l'ouverture du fichier. Ce second cas est presque identique au suivant hormis la première ligne qui change :

212. https://fr.wikipedia.org/wiki/Tabulation-separated_values
213. https://docs.python.org/3/library/functions.html?highlight=open#open
214. https://en.wikipedia.org/wiki/Data_buffer
215. https://docs.python.org/3.6/library/io.html#io.IOBase.flush

```
with open ("nom-fichier", "a") as f:     # ouverture en mode ajout, mode "a"
    ...
```

Pour comprendre la différence entre ces deux modes d'ouverture, voici deux programmes. Celui de gauche n'utilise pas le mode ajout tandis que celui de droite l'utilise lors de la seconde ouverture.

*Premier programme*

```
with open ("essai.txt", "w") as f:
    f.write (" premiere fois ")
    f.close ()

with f = open ("essai.txt", "w") as f:
    f.write (" seconde fois ")
    f.close ()
```

*Second programme*

```
with open ("essai.txt", "w") as f:
    f.write (" premiere fois ")
    f.close ()

with f = open ("essai.txt", "a") as f:  ###
    f.write (" seconde fois ")
    f.close ()
```

Le premier programme crée un fichier `"essai.txt"` qui ne contient que les informations écrites lors de la seconde phase d'écriture, soit `seconde fois`. Le second utilise le mode ajout lors de la seconde ouverture. Le fichier `"essai.txt"`, même s'il existait avant l'exécution de ce programme, est effacé puis rempli avec l'information `premiere fois`. Lors de la seconde ouverture, en mode ajout, une seconde chaîne de caractères est ajoutée. le fichier `"essai.txt"`, après l'exécution du programme contient donc le message : `premiere fois seconde fois`.

Un des moyens pour comprendre ou suivre l'évolution d'un programme est d'écrire des informations dans un fichier ouvert en mode ajout qui est ouvert et fermé sans cesse. Ce sont des fichiers de *traces* ou de log [216]. Ils sont souvent utilisés pour vérifier des calculs complexes. Ils permettent par exemple de comparer deux versions différentes d'un programme pour trouver à quel endroit ils commencent à diverger.

### Lecture

La lecture d'un fichier permet de retrouver les informations stockées grâce à une étape préalable d'écriture. Elle se déroule selon le même principe, à savoir :

1. ouverture du fichier en mode lecture,
2. lecture,
3. fermeture.

Une différence apparaît cependant lors de la lecture d'un fichier : celle-ci s'effectue ligne par ligne alors que l'écriture ne suit pas forcément un découpage en ligne. Les instructions à écrire pour lire un fichier diffèrent rarement du schéma qui suit où seule la ligne indiquée par `(*)` change en fonction ce qu'il faut faire avec les informations lues.

```
with f = open ("essai.txt", "r") as f:  # ouverture du fichier en mode lecture
    for ligne in f :                    # pour toutes les lignes du fichier
        print ligne                     # on affiche la ligne (*)
    # f.close ()                        # on ferme le fichier, ce qui est implicite avec with
```

216. https://fr.wikipedia.org/wiki/Historique_(informatique)

Pour des fichiers qui ne sont pas trop gros (< 100000 lignes), il est possible d'utiliser la méthode readlines [217] qui récupère toutes les lignes d'un fichier texte en une seule fois. Le programme suivant donne le même résultat que le précédent.

```
with open ("essai.txt", "r") as f: # ouverture du fichier en mode lecture
    l = f.readlines ()             # lecture de toutes les lignes, placées dans une␣
↪liste

for s in l:
    print(s)                       # on affiche les lignes à l'écran (*)
```

Lorsque le programme précédent lit une ligne dans un fichier, le résultat lu inclut le ou les caractères (`\n`, `\r` - sous Windows seulement) qui marquent la fin d'une ligne. C'est pour cela que la lecture est parfois suivie d'une étape de nettoyage.

```
with open ("essai.txt", "r") as f: # ouverture du fichier en mode lecture
    l = f.readlines ()             # lecture de toutes les lignes, placées dans une␣
↪liste

# contiendra la liste des lignes nettoyées
l_net = [ s.strip ("\n\r") for s in l ]
```

Les informations peuvent être structurées de façon plus élaborée dans un fichier texte, c'est le cas des formats HTML [218] et XML [219]. Pour ce type de format plus complexe, il est déconseillé de concevoir soi-même un programme capable de les lire, il existe presque toujours un module qui permette de le faire. C'est le cas du module html.parser [220] ou xml [221]. De plus, les modules sont régulièrement mis à jour et suivent l'évolution des formats qu'ils décryptent.

Un fichier texte est le moyen le plus simple d'échanger des matrices ou des données avec un tableur et il n'est pas besoin de modules dans ce cas. Lorsqu'on enregistre une feuille de calcul sous format texte, le fichier obtenu est organisé en colonnes : sur une même ligne, les informations sont disposées en colonne délimitées par un séparateur qui est souvent une tabulation (`\t`) ou un point virgule comme dans l'exemple suivant :

```
nom  ; prénom ; livre
Hugo  ; Victor  ; Les misérables
Kessel ; Joseph  ; Le lion
Woolf ; Virginia  ; Mrs Dalloway
Calvino ; Italo  ; Le baron perché
```

Pour lire ce fichier, il est nécessaire de scinder chaque ligne en une liste de chaînes de caractères, on utilise pour cela la méthode split [222] des chaînes de caractères.

```
mat = []                           # création d'une liste vide,
with open ("essai.txt", "r") as f: # ouverture du fichier en mode lecture
    for li in f :                  # pour toutes les lignes du fichier
        s = li.strip ("\n\r")      # on enlève les caractères de fin de ligne
        l = s.split (";")          # on découpe en colonnes
        mat.append (l)             # on ajoute la ligne à la matrice
```

Ce format de fichier texte est appelé CSV [223] (Comma Separated Value), il peut être relu depuis un programme *python* comme le montre l'exemple précédent, par *Excel* en précisant que le format du fichier est le format *CSV* et par toutes les applications ou langages traitant de données. Pour les valeurs numériques, il ne faut pas oublier de convertir en caractères lors de l'écriture et de convertir en nombres lors de la lecture.

217. https://docs.python.org/3/library/io.html?highlight=readlines#io.IOBase.readlines
218. https://fr.wikipedia.org/wiki/Hypertext_Markup_Language
219. https://fr.wikipedia.org/wiki/Extensible_Markup_Language
220. https://docs.python.org/3/library/html.parser.html
221. https://docs.python.org/3/library/xml.html?highlight=xml#module-xml
222. https://docs.python.org/3/library/stdtypes.html?highlight=split#str.split
223. https://fr.wikipedia.org/wiki/Comma-separated_values

Les nombres réels s'écrivent en anglais avec un point pour séparer la partie entière de la partie décimale. En français, il s'agit d'une virgule. Il est possible que, lors de la conversion d'une matrice, il faille remplacer les points par des virgules et réciproquement pour éviter les problèmes de conversion.

#### Encoding et les accents

Par défaut, un fichier n'accepte pas d'enregistrer des accents, uniquement les acaractères ascii [224]. C'est pourquoi il faut presque tout le temps utiliser le paramètre *encoding* de la fonction open [225] que ce soit pour écrire ou lire.

```
with open("fichier.txt", "r", encoding="utf-8") as f:
    texte = f.read()
```

L'encoding *utf-8* est une façon de représenter les caractères, les caractères ascii sur un octet, les autres sur deux ou trois octets. Cet encoding est le plus fréquent sur internet.

### 5.1.2 Fichiers zip

Les fichiers zip [226] sont très répandus de nos jours et constituent un standard de compression facile d'accès quelque soit l'ordinateur et son système d'exploitation. Le langage *python* propose quelques fonctions pour compresser et décompresser ces fichiers par l'intermédiaire du module zipfile [227]. Le format de compression *zip* est un des plus répandus bien qu'il ne soit pas le plus performant. D'autres formats proposent de meilleurs taux de compression sur les fichiers textes existent comme 7-zip [228]. Ce format n'est pas seulement utilisé pour compresser mais aussi comme un moyen de regrouper plusieurs fichiers en un seul.

#### Lecture

L'exemple suivant permet par exemple d'obtenir la liste des fichiers inclus dans un fichier *zip* :

```
import zipfile
with zipfile.ZipFile ("exemplezip.zip", "r") as fz:
    for info in fz.infolist () :
        print(info.filename, info.date_time, info.file_size)
```

Les fichiers compressés ne sont pas forcément des fichiers textes mais de tout format. Le programme suivant extrait un fichier parmi ceux qui ont été compressés puis affiche son contenu (on suppose que le fichier lu est au format texte donc lisible).

```
import zipfile
with zipfile.ZipFile ("exemplezip.zip", "r") as fz:
    data = fz.read ("informatique/testzip.py")
print(data)
```

On retrouve dans ce cas les étapes d'ouverture et de fermeture même si la première est implicitement inclus dans le constructeur de la classe ZipFile [229].

#### Ecriture

Pour créer un fichier *zip*, le procédé ressemble à la création de n'importe quel fichier. La seule différence provient du fait qu'il est possible de stocker le fichier à compresser sous un autre nom à l'intérieur du fichier *zip*, ce qui explique

224. https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange
225. https://docs.python.org/3/library/functions.html?highlight=open#open
226. https://fr.wikipedia.org/wiki/ZIP_(format_de_fichier)
227. https://docs.python.org/3/library/zipfile.html
228. http://www.7-zip.org/
229. https://docs.python.org/3/library/zipfile.html#zipfile.ZipFile

les deux premiers arguments de la méthode write [230]. Le troisième paramètre indique si le fichier doit être compressé ZIP_DEFLATED [231] ou non ZIP_STORED [232].

```
import zipfile
with zipfile.ZipFile ("test.zip", "w") as f:
    file.write ("fichier.txt", "nom_fichier_dans_zip.txt", zipfile.ZIP_DEFLATED)
```

Une utilisation possible de ce procédé serait l'envoi automatique d'un mail contenant un fichier *zip* en pièce jointe. Une requête comme *python* précédant le nom de votre serveur de mail permettra, via un moteur de recherche, de trouver des exemples sur Internet.

Selon les serveurs de mails, le programme permettant d'envoyer automatiquement un mail en *python* peut varier. L'exemple suivant permet d'envoyer un email automatiquement via un serveur de mails, il montre aussi comment attacher des pièces jointes. Il faut bien sûr être autorisé à se connecter. De plus, il est possible que l'exécution de ce programme ne soit pas toujours couronnée de succès si le mail est envoyé plusieurs fois à répétition, ce comportement est en effet proche de celui d'un spammeur.

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.utils import formatdate
from email import encoders
import os

def envoyer_mail (aqui, sujet, contenu, files = []):
    de = "email de l'auteur"
    msg = MIMEMultipart()
    msg['From'] = de
    msg['To'] = aqui
    msg['Date'] = formatdate (localtime = True)
    msg['Subject'] = sujet

    msg.attach(MIMEText(contenu))
    for file in files:
        part = MIMEBase('application', 'octet-stream')
        with open(file,'rb') as f:
            content = f.read()
        part.set_payload(content)
        encoders.encode_base64(part)
        part.add_header('Content-Disposition',  \
                        'attachment; filename="%s"' % os.path.basename(file))
        msg.attach(part)

    smtp = smtplib.SMTP("smtp.gmail.com", 587)
    smtp.ehlo()
    smtp.starttls()
    smtp.ehlo()
    smtp.login("login", "mot_de_passe")

    smtp.sendmail(de, aqui, msg.as_string())
    smtp.close()

envoyer_mail("destinataire", "sujet","contenu", ["mail.py"])
```

230. https://docs.python.org/3/library/zipfile.html#zipfile.ZipFile.write
231. https://docs.python.org/3/library/zipfile.html#zipfile.ZIP_DEFLATED
232. https://docs.python.org/3/library/zipfile.html#zipfile.ZIP_STORED

### 5.1.3 Manipulation de fichiers

Il arrive fréquemment de copier, recopier, déplacer, effacer des fichiers. Lorsqu'il s'agit de quelques fichiers, le faire manuellement ne pose pas de problème. Lorsqu'il s'agit de traiter plusieurs centaines de fichiers, il est préférable d'écrire un programme qui s'occupe de le faire automatiquement. Cela peut être la création automatique d'un fichier *zip* incluant tous les fichiers modifiés durant la journée ou la réorganisation de fichiers musicaux au format *mp3* à l'aide de modules complémentaires tel que mutagen [233].

Pour ceux qui ne sont pas familiers des systèmes d'exploitation, il faut noter que *Windows* ne fait pas de différences entre les majuscules et les minuscules à l'intérieur d'un nom de fichier. Les systèmes *Linux* et *Mac OSX* font cette différence. Ceci explique que certains programmes aient des comportements différents selon le système d'exploitation sur lequel ils sont exécutés ou encore que certains liens Internet vers des fichiers ne débouchent sur rien car ils ont été saisis avec des différences au niveau des minuscules majuscules.

#### Gestion des noms de chemins

Le module os.path [234] propose plusieurs fonctions très utiles qui permettent entre autres de tester l'existence d'un fichier, d'un répertoire, de récupérer diverses informations comme sa date de création, sa taille... La liste qui suit est loin d'être exhaustive mais elle donne une idée de ce qu'il est possible de faire.

| | |
|---|---|
| abspath(path) [235] | Retourne le chemin absolu d'un fichier ou d'un répertoire. |
| commonprefix(list) [236] | Retourne le plus grand préfixe commun à un ensemble de chemins. |
| dirname(path) [237] | Retourne le nom du répertoire. |
| exists(path) [238] | Dit si un chemin est valide ou non. |
| getatime(path) [239] | date de la dernière modification |
| getmtime(path) [240] | date de la dernière modification |
| getctime(path) [241] | date de la création |
| getsize(file) [242] | Retourne la taille d'un fichier. |
| isabs(path) [243] | Retourne `True` si le chemin est un chemin absolu. |
| isfile(path) [244] | Retourne `True` si le chemin fait référence à un fichier. |
| isdir(path) [245] | Retourne `True` si le chemin fait référence à un répertoire. |
| join(p1, p2, ...) [246] | Construit un nom de chemin étant donné une liste de répertoires. |
| split(path) [247] | Découpe un chemin, isole le nom du fichier ou le dernier répertoire des autres répertoires. |
| splitext(path) [248] | Découpe un chemin en nom + extension. |

233. https://pypi.python.org/pypi/mutagen
234. https://docs.python.org/3/library/os.path.html
235. https://docs.python.org/3/library/os.path.html#os.path.abspath
236. https://docs.python.org/3/library/os.path.html#os.path.commonprefix
237. https://docs.python.org/3/library/os.path.html#os.path.dirname
238. https://docs.python.org/3/library/os.path.html#os.path.exists
239. https://docs.python.org/3/library/os.path.html#os.path.getatime
240. https://docs.python.org/3/library/os.path.html#os.path.getmtime
241. https://docs.python.org/3/library/os.path.html#os.path.getctime
242. https://docs.python.org/3/library/os.path.html#os.path.getsize
243. https://docs.python.org/3/library/os.path.html#os.path.isabs
244. https://docs.python.org/3/library/os.path.html#os.path.isfile
245. https://docs.python.org/3/library/os.path.html#os.path.isdir
246. https://docs.python.org/3/library/os.path.html#os.path.join
247. https://docs.python.org/3/library/os.path.html#os.path.split
248. https://docs.python.org/3/library/os.path.html#os.path.splitext

### Copie, suppression

| | |
|---|---|
| copy (f1,f2) [249] | Copie le fichier `f1` vers `f2`. |
| chdir (p) [250] | Change le répertoire courant, cette fonction peut être importante lorsqu'on utilise la fonction system [251] du module os [252] pour lancer une instruction en ligne de commande ou lorsqu'on écrit un fichier sans préciser le nom du répertoire, le fichier sera écrit dans ce répertoire courant qui est par défaut le répertoire où est situé le programme *python*. C'est à partir du répertoire courant que sont définis les chemins relatifs. |
| getcwd () [253] | Retourne le répertoire courant, voir la fonction `chdir`. |
| mkdir (p) [254] | Crée le répertoire `p`. \ hline |
| makedirs (p) [255] | Crée le répertoire `p` et tous les répertoires des niveaux supérieurs s'ils n'existent pas. Dans le cas du répertoire `d:/base/repfinal`, crée d'abord `d:/base` s'il n'existe pas, puis `d:/base/repfinal`. |
| remove (f) [256] | Supprime un fichier. |
| rename (f1,f2) [257] | Renomme un fichier |
| rmdir (p) [258] | Supprime un répertoire |

### Liste de fichiers

La fonction listdir [259] permet de retourner les listes des éléments inclus dans un répertoire (fichiers et sous-répertoires). Le module glob [260] propose une fonction plus intéressante qui permet de retourner la liste des éléments d'un répertoire en appliquant un filtre. Le programme suivant permet par exemple de retourner la liste des fichiers et des répertoires inclus dans un répertoire.

<<<

```
import glob
import os.path


def liste_fichier_repertoire(folder, filter):
    # résultats
    file, fold = [], []

    # recherche des fichiers obéissant au filtre
    res = glob.glob(folder + "\\" + filter)

    # on inclut les sous-répertoires qui n'auraient pas été
    # sélectionnés par le filtre
    rep = glob.glob(folder + "\\*")
    for r in rep:
        if r not in res and os.path.isdir(r):
```

(suite sur la page suivante)

249. https://docs.python.org/3/library/shutil.html?highlight=shutil#shutil.copy
250. https://docs.python.org/3/library/os.html?highlight=chdir#os.chdir
251. https://docs.python.org/3/library/os.html?highlight=chdir#os.system
252. https://docs.python.org/3/library/os.html?highlight=chdir#os.chdir
253. https://docs.python.org/3/library/os.html?highlight=chdir#os.getcwd
254. https://docs.python.org/3/library/os.html?highlight=chdir#os.mkdir
255. https://docs.python.org/3/library/os.html?highlight=chdir#os.makedirs
256. https://docs.python.org/3/library/os.html?highlight=chdir#os.remove
257. https://docs.python.org/3/library/os.html?highlight=chdir#os.rename
258. https://docs.python.org/3/library/os.html?highlight=chdir#os.rmdir
259. https://docs.python.org/3/library/os.html?highlight=chdir#os.listdir
260. https://docs.python.org/3/library/glob.html?highlight=glob#module-glob

(suite de la page précédente)

```
            res.append(r)

    # on ajoute fichiers et répertoires aux résultats
    for r in res:
        path = r
        if os.path.isfile(path):
            # un fichier, rien à faire à part l'ajouter
            file.append(path)
        else:
            # sous-répertoire : on appelle à nouveau la fonction
            # pour retourner la liste des fichiers inclus
            fold.append(path)
            fi, fo = liste_fichier_repertoire(path, filter)
            file.extend(fi)  # on étend la liste des fichiers
            fold.extend(fo)  # on étend la liste des répertoires
    # fin
    return file, fold


folder = r"."
filter = "*.rst"
file, fold = liste_fichier_repertoire(folder, filter)

for i, f in enumerate(file):
    print("fichier ", f)
    if i >= 10:
        break
for i, f in enumerate(fold):
    print("répertoire ", f)
    if i >= 10:
        break
```

>>>

```
```

Le programme repose sur l'utilisation d'une fonction récursive qui explore d'abord le premier répertoire. Elle se contente d'ajouter à une liste les fichiers qu'elle découvre puis cette fonction s'appelle elle-même sur le premier sous-répertoire qu'elle rencontre. La fonction walk [261] permet d'obtenir la liste des fichiers et des sous-répertoire. Cette fonction parcourt automatiquement les sous-répertoires inclus, le programme est plus court mais elle ne prend pas en compte le filtre qui peut être alors pris en compte grâce aux expressions régulières (voir regex_label_chap).

<<<

```
import os


def liste_fichier_repertoire(folder):
    file, rep = [], []
    for r, d, f in os.walk(folder):
        for a in d:
            rep.append(r + "/" + a)
        for a in f:
            file.append(r + "/" + a)
    return file, rep
```

(suite sur la page suivante)

261. https://docs.python.org/3/library/os.html?highlight=walk#os.walk

(suite de la page précédente)

```
folder = r"."
file, fold = liste_fichier_repertoire(folder)

for f in file:
    print("fichier ", f)
for f in fold:
    print("répertoire ", f)
```

>>>

```
    fichier  ./info.bin
    fichier  ./thread_1.log
    fichier  ./data.bin
    fichier  ./thread_0.log
    fichier  ./build/plot_directive/filechanges-1.png
    fichier  ./build/plot_directive/filechanges-1.pdf
    fichier  ./build/plot_directive/filechanges-1.hires.png
    fichier  ./build/doctrees/environment.pickle
    fichier  ./build/doctrees/end_index2.doctree
    fichier  ./build/doctrees/i_index.doctree
    fichier  ./build/doctrees/all_report.doctree
    fichier  ./build/doctrees/generatedoc.doctree
    fichier  ./build/doctrees/end_index.doctree
    fichier  ./build/doctrees/issues_todoextlist.doctree
    fichier  ./build/doctrees/introduction.doctree
    fichier  ./build/doctrees/i_ex.doctree
    fichier  ./build/doctrees/generatesetup.doctree
    fichier  ./build/doctrees/end_index_glossaire.doctree
    fichier  ./build/doctrees/HISTORY.doctree
    fichier  ./build/doctrees/index.doctree
    fichier  ./build/doctrees/doctestunit.doctree
    fichier  ./build/doctrees/completed_todoextlist.doctree
    fichier  ./build/doctrees/end_index_changes.doctree
    fichier  ./build/doctrees/all_indexes.doctree
    fichier  ./build/doctrees/installation.doctree
    fichier  ./build/doctrees/all_notebooks.doctree
    fichier  ./build/doctrees/defthe_index.doctree
    fichier  ./build/doctrees/i_examples.doctree
    fichier  ./build/doctrees/license.doctree
    fichier  ./build/doctrees/README.doctree
    fichier  ./build/doctrees/index_module.doctree
    fichier  ./build/doctrees/all_notebooks_coverage.doctree
    fichier  ./build/doctrees/end_index_module.doctree
    fichier  ./build/doctrees/i_faq.doctree
    fichier  ./build/doctrees/glossary.doctree
    fichier  ./build/doctrees/filechanges.doctree
    fichier  ./build/doctrees/index_function.doctree
    fichier  ./build/doctrees/notebooks/numpy_tricks.doctree
    fichier  ./build/doctrees/notebooks/pandas_groupby.doctree
    fichier  ./build/doctrees/notebooks/serialisation_examples.doctree
    fichier  ./build/doctrees/notebooks/gil_example.doctree
    fichier  ./build/doctrees/notebooks/notebook_template.doctree
    fichier  ./build/doctrees/notebooks/serialisation_protobuf.doctree
    fichier  ./build/doctrees/notebooks/float_and_double_rouding.doctree
    fichier  ./build/doctrees/c_io/index.doctree
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/doctrees/c_io/files.doctree
    fichier  ./build/doctrees/c_io/serialization.doctree
    fichier  ./build/doctrees/c_parallelisation/index.doctree
    fichier  ./build/doctrees/c_parallelisation/thread.doctree
    fichier  ./build/doctrees/c_resume/index.doctree
    fichier  ./build/doctrees/c_resume/python_sheet.doctree
    fichier  ./build/doctrees/c_resume/conseil_programmes.doctree
    fichier  ./build/doctrees/c_lang/dates.doctree
    fichier  ./build/doctrees/c_lang/syntaxe.doctree
    fichier  ./build/doctrees/c_lang/index.doctree
    fichier  ./build/doctrees/c_lang/encoding.doctree
    fichier  ./build/doctrees/c_lang/constructions.doctree
    fichier  ./build/doctrees/c_lang/faq.doctree
    fichier  ./build/doctrees/c_lang/types.doctree
    fichier  ./build/doctrees/gyexamples/lambda_function.doctree
    fichier  ./build/doctrees/gyexamples/index.doctree
    fichier  ./build/doctrees/teachpyx/__init__.doctree
    fichier  ./build/doctrees/teachpyx/faq/faq_python.doctree
    fichier  ./build/doctrees/teachpyx/faq/faq_numpy.doctree
    fichier  ./build/doctrees/teachpyx/faq/faq_exception.doctree
    fichier  ./build/doctrees/teachpyx/faq/__init__.doctree
    fichier  ./build/doctrees/teachpyx/examples/numpysex.doctree
    fichier  ./build/doctrees/teachpyx/examples/classiques.doctree
    fichier  ./build/doctrees/teachpyx/examples/construction_classique.doctree
    fichier  ./build/doctrees/teachpyx/examples/__init__.doctree
    fichier  ./build/doctrees/c_data/index.doctree
    fichier  ./build/doctrees/c_module/module.doctree
    fichier  ./build/doctrees/c_module/index.doctree
    fichier  ./build/doctrees/c_module/faq.doctree
    fichier  ./build/doctrees/blog/cat-module_0000.doctree
    fichier  ./build/doctrees/blog/cat-blog_0000.doctree
    fichier  ./build/doctrees/blog/month-2018-04_0000.doctree
    fichier  ./build/doctrees/blog/blogindex.doctree
    fichier  ./build/doctrees/blog/cat-classe_0000.doctree
    fichier  ./build/doctrees/blog/cat-debug_0000.doctree
    fichier  ./build/doctrees/blog/month-2017-07_0000.doctree
    fichier  ./build/doctrees/blog/month-2018-03_0000.doctree
    fichier  ./build/doctrees/blog/main_0000.doctree
    fichier  ./build/doctrees/blog/index_blog.doctree
    fichier  ./build/doctrees/blog/month-2016-11_0000.doctree
    fichier  ./build/doctrees/blog/month-2017-01_0000.doctree
    fichier  ./build/doctrees/blog/cat-pandas_0000.doctree
    fichier  ./build/doctrees/blog/cat-weird_0000.doctree
    fichier  ./build/doctrees/blog/month-2016-09_0000.doctree
    fichier  ./build/doctrees/blog/2016/2016-11-20_groupby_nan.doctree
    fichier  ./build/doctrees/blog/2016/2016-08-18_first_blog.doctree
    fichier  ./build/doctrees/blog/2018/2018-04-05_debug.doctree
    fichier  ./build/doctrees/blog/2018/2018-03-31_classesfonctions.doctree
    fichier  ./build/doctrees/blog/2017/2017-01-06_programming.doctree
    fichier  ./build/doctrees/blog/2017/2017-07-01_cextension.doctree
    fichier  ./build/doctrees/c_exception/index.doctree
    fichier  ./build/doctrees/c_exception/exception.doctree
    fichier  ./build/doctrees/c_exception/exception_ext.doctree
    fichier  ./build/doctrees/c_gui/index.doctree
    fichier  ./build/doctrees/c_gui/tkinter.doctree
    fichier  ./build/doctrees/c_classes/classes.doctree
    fichier  ./build/doctrees/c_classes/index.doctree
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/doctrees/c_regex/index.doctree
    fichier  ./build/doctrees/c_regex/regex.doctree
    fichier  ./build/html/generatedoc.html
    fichier  ./build/html/end_index.html
    fichier  ./build/html/end_index_changes.html
    fichier  ./build/html/generatesetup.html
    fichier  ./build/html/i_examples.html
    fichier  ./build/html/all_indexes.html
    fichier  ./build/html/index.html
    fichier  ./build/html/i_index.html
    fichier  ./build/html/genindex.html
    fichier  ./build/html/index_module.html
    fichier  ./build/html/.buildinfo
    fichier  ./build/html/all_report.html
    fichier  ./build/html/all_notebooks_coverage.html
    fichier  ./build/html/py-modindex.html
    fichier  ./build/html/all_notebooks.html
    fichier  ./build/html/HISTORY.html
    fichier  ./build/html/glossary.html
    fichier  ./build/html/objects.inv
    fichier  ./build/html/introduction.html
    fichier  ./build/html/filechanges-1.png
    fichier  ./build/html/end_index2.html
    fichier  ./build/html/end_index_module.html
    fichier  ./build/html/completed_todoextlist.html
    fichier  ./build/html/doctestunit.html
    fichier  ./build/html/searchindex.js
    fichier  ./build/html/defthe_index.html
    fichier  ./build/html/filechanges-1.pdf
    fichier  ./build/html/i_ex.html
    fichier  ./build/html/README.html
    fichier  ./build/html/filechanges.html
    fichier  ./build/html/search.html
    fichier  ./build/html/end_index_glossaire.html
    fichier  ./build/html/license.html
    fichier  ./build/html/i_faq.html
    fichier  ./build/html/filechanges-1.py
    fichier  ./build/html/index_function.html
    fichier  ./build/html/issues_todoextlist.html
    fichier  ./build/html/filechanges-1.hires.png
    fichier  ./build/html/installation.html
    fichier  ./build/html/notebooks/serialisation_examples.html
    fichier  ./build/html/notebooks/float_and_double_rouding.html
    fichier  ./build/html/notebooks/notebook_template.html
    fichier  ./build/html/notebooks/pandas_groupby.html
    fichier  ./build/html/notebooks/numpy_tricks.html
    fichier  ./build/html/notebooks/serialisation_protobuf.html
    fichier  ./build/html/notebooks/gil_example.html
    fichier  ./build/html/c_io/index.html
    fichier  ./build/html/c_io/serialization.html
    fichier  ./build/html/c_io/files.html
    fichier  ./build/html/c_parallelisation/thread.html
    fichier  ./build/html/c_parallelisation/index.html
    fichier  ./build/html/c_resume/conseil_programmes.html
    fichier  ./build/html/c_resume/index.html
    fichier  ./build/html/c_resume/python_sheet.html
    fichier  ./build/html/c_lang/constructions.html
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/c_lang/syntaxe.html
    fichier  ./build/html/c_lang/index.html
    fichier  ./build/html/c_lang/dates.html
    fichier  ./build/html/c_lang/faq.html
    fichier  ./build/html/c_lang/types.html
    fichier  ./build/html/c_lang/encoding.html
    fichier  ./build/html/_static/comment.png
    fichier  ./build/html/_static/bootstrap-sphinx.css
    fichier  ./build/html/_static/project_ico_small.png
    fichier  ./build/html/_static/_stemmer.js
    fichier  ./build/html/_static/broken_example.png
    fichier  ./build/html/_static/ajax-loader.gif
    fichier  ./build/html/_static/git_logo.png
    fichier  ./build/html/_static/file.png
    fichier  ./build/html/_static/down-pressed.png
    fichier  ./build/html/_static/gallery.css
    fichier  ./build/html/_static/minus.png
    fichier  ./build/html/_static/jquery-3.2.1.js
    fichier  ./build/html/_static/websupport.js
    fichier  ./build/html/_static/basic.css
    fichier  ./build/html/_static/plus.png
    fichier  ./build/html/_static/pygments.css
    fichier  ./build/html/_static/style_notebook_snippet.css
    fichier  ./build/html/_static/underscore.js
    fichier  ./build/html/_static/up.png
    fichier  ./build/html/_static/down.png
    fichier  ./build/html/_static/up-pressed.png
    fichier  ./build/html/_static/no_image.png
    fichier  ./build/html/_static/project_ico.ico
    fichier  ./build/html/_static/comment-close.png
    fichier  ./build/html/_static/documentation_options.js
    fichier  ./build/html/_static/project_ico.png
    fichier  ./build/html/_static/bootstrap-sphinx.js
    fichier  ./build/html/_static/jquery.js
    fichier  ./build/html/_static/my-styles.css
    fichier  ./build/html/_static/jsdemo.css
    fichier  ./build/html/_static/comment-bright.png
    fichier  ./build/html/_static/searchtools.js
    fichier  ./build/html/_static/doctools.js
    fichier  ./build/html/_static/require.js
    fichier  ./build/html/_static/jsdemo.js
    fichier  ./build/html/_static/underscore-1.3.1.js
    fichier  ./build/html/_static/translations.js
    fichier  ./build/html/_static/reveal.js/package.json
    fichier  ./build/html/_static/reveal.js/revealjs.css
    fichier  ./build/html/_static/reveal.js/LICENSE
    fichier  ./build/html/_static/reveal.js/css/reveal.css
    fichier  ./build/html/_static/reveal.js/css/reveal.scss
    fichier  ./build/html/_static/reveal.js/css/theme/solarized.css
    fichier  ./build/html/_static/reveal.js/css/theme/simple.css
    fichier  ./build/html/_static/reveal.js/css/theme/black.css
    fichier  ./build/html/_static/reveal.js/css/theme/night.css
    fichier  ./build/html/_static/reveal.js/css/theme/moon.css
    fichier  ./build/html/_static/reveal.js/css/theme/white.css
    fichier  ./build/html/_static/reveal.js/css/theme/beige.css
    fichier  ./build/html/_static/reveal.js/css/theme/serif.css
    fichier  ./build/html/_static/reveal.js/css/theme/league.css
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_static/reveal.js/css/theme/sky.css
    fichier  ./build/html/_static/reveal.js/css/theme/README.md
    fichier  ./build/html/_static/reveal.js/css/theme/blood.css
    fichier  ./build/html/_static/reveal.js/css/theme/source/league.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/black.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/white.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/simple.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/beige.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/serif.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/moon.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/solarized.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/night.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/blood.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/sky.scss
    fichier  ./build/html/_static/reveal.js/css/theme/source/__pycache__/__init__.
↪cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/css/theme/template/settings.scss
    fichier  ./build/html/_static/reveal.js/css/theme/template/mixins.scss
    fichier  ./build/html/_static/reveal.js/css/theme/template/theme.scss
    fichier  ./build/html/_static/reveal.js/css/theme/template/__pycache__/__init__.
↪cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/css/theme/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./build/html/_static/reveal.js/css/__pycache__/__init__.cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/css/print/pdf.css
    fichier  ./build/html/_static/reveal.js/css/print/paper.css
    fichier  ./build/html/_static/reveal.js/css/print/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./build/html/_static/reveal.js/plugin/__pycache__/__init__.cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/js/reveal.js
    fichier  ./build/html/_static/reveal.js/js/jquery.min.js
    fichier  ./build/html/_static/reveal.js/js/__pycache__/__init__.cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/lib/css/zenburn.css
    fichier  ./build/html/_static/reveal.js/lib/css/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./build/html/_static/reveal.js/lib/js/classList.js
    fichier  ./build/html/_static/reveal.js/lib/js/html5shiv.js
    fichier  ./build/html/_static/reveal.js/lib/js/head.min.js
    fichier  ./build/html/_static/reveal.js/lib/js/__pycache__/__init__.cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.ttf
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro.
↪css
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.woff
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.ttf
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.ttf
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.ttf
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.eot
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.eot
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.woff
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.woff
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.eot
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.eot
    fichier  ./build/html/_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.woff
    fichier  ./build/html/_static/reveal.js/lib/font/league-gothic/league-gothic.css
    fichier  ./build/html/_static/reveal.js/lib/font/league-gothic/league-gothic.woff
    fichier  ./build/html/_static/reveal.js/lib/font/league-gothic/league-gothic.eot
    fichier  ./build/html/_static/reveal.js/lib/font/league-gothic/league-gothic.ttf
    fichier  ./build/html/_static/reveal.js/lib/font/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./build/html/_static/reveal.js/lib/__pycache__/__init__.cpython-37.pyc
    fichier  ./build/html/_static/reveal.js/__pycache__/__init__.cpython-37.pyc
    fichier  ./build/html/_static/bootswatch-2.3.2/journal/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/simplex/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/flatly/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/img/glyphicons-halflings-white.png
    fichier  ./build/html/_static/bootswatch-2.3.2/img/glyphicons-halflings.png
    fichier  ./build/html/_static/bootswatch-2.3.2/cyborg/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/cerulean/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/spruce/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/spacelab/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/slate/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/united/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/readable/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/amelia/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/cosmo/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-2.3.2/superhero/bootstrap.min.css
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap.min.css.map
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap-theme.css.map
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap.css
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap-theme.min.css.map
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap-theme.min.css
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap.css.map
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap-theme.css
    fichier  ./build/html/_static/bootstrap-3.3.7/css/bootstrap.min.css
    fichier  ./build/html/_static/bootstrap-3.3.7/js/bootstrap.min.js
    fichier  ./build/html/_static/bootstrap-3.3.7/js/bootstrap.js
    fichier  ./build/html/_static/bootstrap-3.3.7/js/npm.js
    fichier  ./build/html/_static/bootstrap-3.3.7/fonts/glyphicons-halflings-regular.
↪ttf
    fichier  ./build/html/_static/bootstrap-3.3.7/fonts/glyphicons-halflings-regular.
↪woff2
    fichier  ./build/html/_static/bootstrap-3.3.7/fonts/glyphicons-halflings-regular.
↪woff
    fichier  ./build/html/_static/bootstrap-3.3.7/fonts/glyphicons-halflings-regular.
↪eot
    fichier  ./build/html/_static/bootstrap-3.3.7/fonts/glyphicons-halflings-regular.
↪svg
    fichier  ./build/html/_static/bootswatch-3.3.7/darkly/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/journal/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/yeti/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/simplex/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/flatly/bootstrap.min.css
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_static/bootswatch-3.3.7/cyborg/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/cerulean/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/spacelab/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/sandstone/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/paper/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/lumen/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/slate/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/solar/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/united/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/readable/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/fonts/glyphicons-halflings-regular.
↪ttf
    fichier  ./build/html/_static/bootswatch-3.3.7/fonts/glyphicons-halflings-regular.
↪woff2
    fichier  ./build/html/_static/bootswatch-3.3.7/fonts/glyphicons-halflings-regular.
↪woff
    fichier  ./build/html/_static/bootswatch-3.3.7/fonts/glyphicons-halflings-regular.
↪eot
    fichier  ./build/html/_static/bootswatch-3.3.7/fonts/glyphicons-halflings-regular.
↪svg
    fichier  ./build/html/_static/bootswatch-3.3.7/cosmo/bootstrap.min.css
    fichier  ./build/html/_static/bootswatch-3.3.7/superhero/bootstrap.min.css
    fichier  ./build/html/_static/js/jquery-1.11.0.min.js
    fichier  ./build/html/_static/js/jquery-fix.js
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/css/lightbox.
↪css
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/img/close.png
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/img/next.png
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/img/loading.
↪gif
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/img/prev.png
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/js/jquery-1.
↪11.0.min.js
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/js/lightbox.
↪min.map
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/js/lightbox.
↪min.js
    fichier  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2_customize/
↪jquery-noconflict.js
    fichier  ./build/html/_static/bootstrap-2.3.2/css/bootstrap-responsive.min.css
    fichier  ./build/html/_static/bootstrap-2.3.2/css/bootstrap.css
    fichier  ./build/html/_static/bootstrap-2.3.2/css/bootstrap.min.css
    fichier  ./build/html/_static/bootstrap-2.3.2/css/bootstrap-responsive.css
    fichier  ./build/html/_static/bootstrap-2.3.2/img/glyphicons-halflings-white.png
    fichier  ./build/html/_static/bootstrap-2.3.2/img/glyphicons-halflings.png
    fichier  ./build/html/_static/bootstrap-2.3.2/js/bootstrap.min.js
    fichier  ./build/html/_static/bootstrap-2.3.2/js/bootstrap.js
    fichier  ./build/html/gyexamples/lambda_function.html
    fichier  ./build/html/gyexamples/index.html
    fichier  ./build/html/_downloads/pandas_groupby.ipynb
    fichier  ./build/html/_downloads/serialisation_protobuf.ipynb
    fichier  ./build/html/_downloads/gil_example.pdf
    fichier  ./build/html/_downloads/numpy_tricks.py
    fichier  ./build/html/_downloads/rss.xml
    fichier  ./build/html/_downloads/serialisation_protobuf2html.html
    fichier  ./build/html/_downloads/serialisation_examples.ipynb
    fichier  ./build/html/_downloads/numpy_tricks2html.html
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_downloads/gil_example.slides2p.html
    fichier  ./build/html/_downloads/pandas_groupby.slides.html
    fichier  ./build/html/_downloads/serialisation_examples2html.html
    fichier  ./build/html/_downloads/float_and_double_rouding.pdf
    fichier  ./build/html/_downloads/gyexamples_python.zip
    fichier  ./build/html/_downloads/numpy_tricks.ipynb
    fichier  ./build/html/_downloads/pandas_groupby.pdf
    fichier  ./build/html/_downloads/pandas_groupby2html.html
    fichier  ./build/html/_downloads/float_and_double_rouding.ipynb
    fichier  ./build/html/_downloads/float_and_double_rouding2html.html
    fichier  ./build/html/_downloads/serialisation_protobuf.slides2p.html
    fichier  ./build/html/_downloads/gyexamples_jupyter.zip
    fichier  ./build/html/_downloads/notebook_template.pdf
    fichier  ./build/html/_downloads/gil_example.ipynb
    fichier  ./build/html/_downloads/pandas_groupby.py
    fichier  ./build/html/_downloads/serialisation_protobuf.slides.html
    fichier  ./build/html/_downloads/gil_example.py
    fichier  ./build/html/_downloads/numpy_tricks.slides2p.html
    fichier  ./build/html/_downloads/serialisation_protobuf.pdf
    fichier  ./build/html/_downloads/serialisation_examples.py
    fichier  ./build/html/_downloads/serialisation_examples.slides2p.html
    fichier  ./build/html/_downloads/serialisation_examples.pdf
    fichier  ./build/html/_downloads/serialisation_examples.slides.html
    fichier  ./build/html/_downloads/notebook_template.slides2p.html
    fichier  ./build/html/_downloads/lambda_function.ipynb
    fichier  ./build/html/_downloads/notebook_template.py
    fichier  ./build/html/_downloads/float_and_double_rouding.slides2p.html
    fichier  ./build/html/_downloads/numpy_tricks.slides.html
    fichier  ./build/html/_downloads/notebook_template.ipynb
    fichier  ./build/html/_downloads/serialisation_protobuf.py
    fichier  ./build/html/_downloads/lambda_function.py
    fichier  ./build/html/_downloads/float_and_double_rouding.py
    fichier  ./build/html/_downloads/notebook_template.slides.html
    fichier  ./build/html/_downloads/gil_example2html.html
    fichier  ./build/html/_downloads/gil_example.slides.html
    fichier  ./build/html/_downloads/float_and_double_rouding.slides.html
    fichier  ./build/html/_downloads/numpy_tricks.pdf
    fichier  ./build/html/_downloads/pandas_groupby.slides2p.html
    fichier  ./build/html/_downloads/notebook_template2html.html
    fichier  ./build/html/_images/nbcov-2018-08-28.png
    fichier  ./build/html/_images/check2.png
    fichier  ./build/html/_images/menu.png
    fichier  ./build/html/_images/notebook_template.thumb.png
    fichier  ./build/html/_images/combo2.png
    fichier  ./build/html/_images/bind.png
    fichier  ./build/html/_images/numpy_tricks.thumb.png
    fichier  ./build/html/_images/float_and_double_rouding_22_1.png
    fichier  ./build/html/_images/threadim2.png
    fichier  ./build/html/_images/can.png
    fichier  ./build/html/_images/float_and_double_rouding_21_1.png
    fichier  ./build/html/_images/float_and_double_rouding_27_1.png
    fichier  ./build/html/_images/blockdiag-9b96c33ee80cd507950e8b8e98740627159a0672.
↪png
    fichier  ./build/html/_images/asyncapi.png
    fichier  ./build/html/_images/list1.png
    fichier  ./build/html/_images/bouton.png
    fichier  ./build/html/_images/pack1.png
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_images/sphx_glr_lambda_function_thumb.png
    fichier  ./build/html/_images/comm1.png
    fichier  ./build/html/_images/float_and_double_rouding_25_1.png
    fichier  ./build/html/_images/label.png
    fichier  ./build/html/_images/focus.png
    fichier  ./build/html/_images/frame.png
    fichier  ./build/html/_images/comm2.png
    fichier  ./build/html/_images/blockdiag-84aeead3c0597dcb2bce7f729f7d85ba10c5a4c4.
↪png
    fichier  ./build/html/_images/menut.png
    fichier  ./build/html/_images/bbette2.png
    fichier  ./build/html/_images/fenpers.png
    fichier  ./build/html/_images/radio.png
    fichier  ./build/html/_images/mainloop.png
    fichier  ./build/html/_images/bouton3.png
    fichier  ./build/html/_images/float_and_double_rouding_16_0.png
    fichier  ./build/html/_images/threadim1.png
    fichier  ./build/html/_images/saisie1.png
    fichier  ./build/html/_images/list2.png
    fichier  ./build/html/_images/after.png
    fichier  ./build/html/_images/entree.png
    fichier  ./build/html/_images/pool.png
    fichier  ./build/html/_images/nbcov.png
    fichier  ./build/html/_images/float_and_double_rouding_29_1.png
    fichier  ./build/html/_images/check.png
    fichier  ./build/html/_images/arbo.png
    fichier  ./build/html/_images/filechanges-1.png
    fichier  ./build/html/_images/pandas_groupby.thumb.png
    fichier  ./build/html/_images/float_and_double_rouding.thumb.png
    fichier  ./build/html/_images/gil_example.thumb.png
    fichier  ./build/html/_images/seqev.png
    fichier  ./build/html/_images/text1.png
    fichier  ./build/html/_images/grid1.png
    fichier  ./build/html/_images/saisie2.png
    fichier  ./build/html/_images/tixfile.png
    fichier  ./build/html/_images/feed-icon-16x16.png
    fichier  ./build/html/_images/float_and_double_rouding_11_0.png
    fichier  ./build/html/_images/float_and_double_rouding_31_1.png
    fichier  ./build/html/_images/bouton2.png
    fichier  ./build/html/_images/combo1.png
    fichier  ./build/html/_images/serialisation_examples.thumb.png
    fichier  ./build/html/_images/listboxs.png
    fichier  ./build/html/_images/pack2.png
    fichier  ./build/html/_images/radio2.png
    fichier  ./build/html/_images/rawinput.png
    fichier  ./build/html/_images/serialisation_protobuf.thumb.png
    fichier  ./build/html/_images/math/5de68205c0e7e79ee51438987920f671a4bb3796.png
    fichier  ./build/html/_images/math/51e6b7e101fd5bd488dfb79f9c4313ff08d906bc.png
    fichier  ./build/html/_images/math/9e6c662d530db8a324e9f2fe855eeacafbd96291.png
    fichier  ./build/html/_images/math/e8be8c687bcb1a910adeec344e2ec21d4a59545a.png
    fichier  ./build/html/_images/math/3e89908b2f485767653aa48ce3dd1e354c671c2e.png
    fichier  ./build/html/_images/math/9c82166b504284b39604901b4e93776294e12484.png
    fichier  ./build/html/_images/math/ade0048d1bebdf9519e8e643e42f9aad53caa727.png
    fichier  ./build/html/_images/math/eee72901ce86c5411c4be7b7c2ce210cf70a4523.png
    fichier  ./build/html/_images/math/ce1502643c48703f4761335aebaf764f5bdeee78.png
    fichier  ./build/html/_images/math/ee55097c892f1418de8cdd6cb44e300dbdfb8c1f.png
    fichier  ./build/html/_images/math/e39ddf05c56c382e29aa91aaceca2d5de5c4bd29.png
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_images/math/0599a39fcdcf7ae0060ae6bd0b68f92afa8ba6f3.png
    fichier  ./build/html/_images/math/935c759ee735e32aab71af3f673094e099135336.png
    fichier  ./build/html/_images/math/e43f2b63e963ef4b65f7b82b88ee5eebaaaaebeb.png
    fichier  ./build/html/_images/math/3aa4a1363ac95bf980e27b96cbc1efe4732596f6.png
    fichier  ./build/html/_images/math/a4a763b3ebb467a373431dc6238b041065b806f5.png
    fichier  ./build/html/_images/math/89861133ea22c591811f4661dc002345fb2ef8f0.png
    fichier  ./build/html/_images/math/3f2db4151922d9a1cb7e19ba0fc4a595cc18f059.png
    fichier  ./build/html/_images/math/ff4aee007b9ab35b237eb51cbfaec1f3f1cad82d.png
    fichier  ./build/html/_images/math/3d9b440fa00fb31da9880d9c34d4c1bb249c3b07.png
    fichier  ./build/html/_images/math/ec8256280af2c8dde94607bae5d74402ef7c733e.png
    fichier  ./build/html/_images/math/22a84f5db4e94d3f371b9d9c5436ad2b53bd0e6f.png
    fichier  ./build/html/_images/math/32ae8d6ee051c041f25640fa91cff6b0dc553ef4.png
    fichier  ./build/html/_images/math/619b5dfddaeded777ee2b6c7676e2caf9fcd42f6.png
    fichier  ./build/html/_images/math/d88bf614100dc9561ab4b951b22f965c77da355d.png
    fichier  ./build/html/_images/math/b92d737e3fb8e48be8a763d5e7b7ecb5965203d3.png
    fichier  ./build/html/_images/math/faff67d89fbe99cdaa566c20b52f4082d6dbe92a.png
    fichier  ./build/html/_images/math/b34f5b96a0b8962aa4f596f4ffe344f9ec8d590e.png
    fichier  ./build/html/_images/math/7fc6bffd99db3a472bf9b708854ac1959e5e606e.png
    fichier  ./build/html/_images/math/e5adc6945ddc17d0b1f0e02fa09e39f3acab61db.png
    fichier  ./build/html/_images/math/d63391906419249868edcf9ffdc408a7847b7799.png
    fichier  ./build/html/_images/math/a283b6104f42fc7a8bf845790aa022ac525329f0.png
    fichier  ./build/html/_images/math/0a1e6aabe17c6d01f7d96e8766ac8f1851079ea2.png
    fichier  ./build/html/_images/math/3ff72ae1d317644a459e73c0390a7fe6d9b835c4.png
    fichier  ./build/html/_modules/index.html
    fichier  ./build/html/_modules/teachpyx.html
    fichier  ./build/html/_modules/teachpyx/faq/faq_python.html
    fichier  ./build/html/_modules/teachpyx/faq/faq_exception.html
    fichier  ./build/html/_modules/teachpyx/faq/faq_numpy.html
    fichier  ./build/html/_modules/teachpyx/examples/construction_classique.html
    fichier  ./build/html/_modules/teachpyx/examples/numpysex.html
    fichier  ./build/html/_modules/teachpyx/examples/classiques.html
    fichier  ./build/html/_sources/end_index_glossaire.rst.txt
    fichier  ./build/html/_sources/HISTORY.rst.txt
    fichier  ./build/html/_sources/end_index.rst.txt
    fichier  ./build/html/_sources/license.rst.txt
    fichier  ./build/html/_sources/all_notebooks_coverage.rst.txt
    fichier  ./build/html/_sources/introduction.rst.txt
    fichier  ./build/html/_sources/end_index_module.rst.txt
    fichier  ./build/html/_sources/i_faq.rst.txt
    fichier  ./build/html/_sources/i_index.rst.txt
    fichier  ./build/html/_sources/end_index_changes.rst.txt
    fichier  ./build/html/_sources/issues_todoextlist.rst.txt
    fichier  ./build/html/_sources/completed_todoextlist.rst.txt
    fichier  ./build/html/_sources/defthe_index.rst.txt
    fichier  ./build/html/_sources/installation.rst.txt
    fichier  ./build/html/_sources/index_module.rst.txt
    fichier  ./build/html/_sources/filechanges.rst.txt
    fichier  ./build/html/_sources/all_indexes.rst.txt
    fichier  ./build/html/_sources/generatedoc.rst.txt
    fichier  ./build/html/_sources/index.rst.txt
    fichier  ./build/html/_sources/index_function.rst.txt
    fichier  ./build/html/_sources/end_index2.rst.txt
    fichier  ./build/html/_sources/i_examples.rst.txt
    fichier  ./build/html/_sources/all_report.rst.txt
    fichier  ./build/html/_sources/i_ex.rst.txt
    fichier  ./build/html/_sources/doctestunit.rst.txt
    fichier  ./build/html/_sources/generatesetup.rst.txt
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_sources/all_notebooks.rst.txt
    fichier  ./build/html/_sources/glossary.rst.txt
    fichier  ./build/html/_sources/README.rst.txt
    fichier  ./build/html/_sources/notebooks/float_and_double_rouding.rst.txt
    fichier  ./build/html/_sources/notebooks/pandas_groupby.rst.txt
    fichier  ./build/html/_sources/notebooks/serialisation_protobuf.rst.txt
    fichier  ./build/html/_sources/notebooks/serialisation_examples.rst.txt
    fichier  ./build/html/_sources/notebooks/numpy_tricks.rst.txt
    fichier  ./build/html/_sources/notebooks/notebook_template.rst.txt
    fichier  ./build/html/_sources/notebooks/gil_example.rst.txt
    fichier  ./build/html/_sources/c_io/serialization.rst.txt
    fichier  ./build/html/_sources/c_io/files.rst.txt
    fichier  ./build/html/_sources/c_io/index.rst.txt
    fichier  ./build/html/_sources/c_parallelisation/index.rst.txt
    fichier  ./build/html/_sources/c_parallelisation/thread.rst.txt
    fichier  ./build/html/_sources/c_resume/conseil_programmes.rst.txt
    fichier  ./build/html/_sources/c_resume/index.rst.txt
    fichier  ./build/html/_sources/c_resume/python_sheet.rst.txt
    fichier  ./build/html/_sources/c_lang/constructions.rst.txt
    fichier  ./build/html/_sources/c_lang/dates.rst.txt
    fichier  ./build/html/_sources/c_lang/syntaxe.rst.txt
    fichier  ./build/html/_sources/c_lang/encoding.rst.txt
    fichier  ./build/html/_sources/c_lang/types.rst.txt
    fichier  ./build/html/_sources/c_lang/faq.rst.txt
    fichier  ./build/html/_sources/c_lang/index.rst.txt
    fichier  ./build/html/_sources/gyexamples/lambda_function.rst.txt
    fichier  ./build/html/_sources/gyexamples/index.rst.txt
    fichier  ./build/html/_sources/teachpyx/__init__.rst.txt
    fichier  ./build/html/_sources/teachpyx/faq/faq_python.rst.txt
    fichier  ./build/html/_sources/teachpyx/faq/faq_exception.rst.txt
    fichier  ./build/html/_sources/teachpyx/faq/faq_numpy.rst.txt
    fichier  ./build/html/_sources/teachpyx/faq/__init__.rst.txt
    fichier  ./build/html/_sources/teachpyx/examples/numpysex.rst.txt
    fichier  ./build/html/_sources/teachpyx/examples/classiques.rst.txt
    fichier  ./build/html/_sources/teachpyx/examples/construction_classique.rst.txt
    fichier  ./build/html/_sources/teachpyx/examples/__init__.rst.txt
    fichier  ./build/html/_sources/c_data/index.rst.txt
    fichier  ./build/html/_sources/c_module/module.rst.txt
    fichier  ./build/html/_sources/c_module/faq.rst.txt
    fichier  ./build/html/_sources/c_module/index.rst.txt
    fichier  ./build/html/_sources/blog/cat-blog_0000.rst.txt
    fichier  ./build/html/_sources/blog/month-2018-03_0000.rst.txt
    fichier  ./build/html/_sources/blog/month-2017-07_0000.rst.txt
    fichier  ./build/html/_sources/blog/month-2016-11_0000.rst.txt
    fichier  ./build/html/_sources/blog/cat-classe_0000.rst.txt
    fichier  ./build/html/_sources/blog/month-2016-09_0000.rst.txt
    fichier  ./build/html/_sources/blog/month-2017-01_0000.rst.txt
    fichier  ./build/html/_sources/blog/main_0000.rst.txt
    fichier  ./build/html/_sources/blog/cat-debug_0000.rst.txt
    fichier  ./build/html/_sources/blog/cat-pandas_0000.rst.txt
    fichier  ./build/html/_sources/blog/blogindex.rst.txt
    fichier  ./build/html/_sources/blog/cat-module_0000.rst.txt
    fichier  ./build/html/_sources/blog/cat-weird_0000.rst.txt
    fichier  ./build/html/_sources/blog/index_blog.rst.txt
    fichier  ./build/html/_sources/blog/month-2018-04_0000.rst.txt
    fichier  ./build/html/_sources/blog/2016/2016-08-18_first_blog.rst.txt
    fichier  ./build/html/_sources/blog/2016/2016-11-20_groupby_nan.rst.txt
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./build/html/_sources/blog/2018/2018-03-31_classesfonctions.rst.txt
    fichier  ./build/html/_sources/blog/2018/2018-04-05_debug.rst.txt
    fichier  ./build/html/_sources/blog/2017/2017-07-01_cextension.rst.txt
    fichier  ./build/html/_sources/blog/2017/2017-01-06_programming.rst.txt
    fichier  ./build/html/_sources/c_exception/exception_ext.rst.txt
    fichier  ./build/html/_sources/c_exception/exception.rst.txt
    fichier  ./build/html/_sources/c_exception/index.rst.txt
    fichier  ./build/html/_sources/c_gui/index.rst.txt
    fichier  ./build/html/_sources/c_gui/tkinter.rst.txt
    fichier  ./build/html/_sources/c_classes/index.rst.txt
    fichier  ./build/html/_sources/c_classes/classes.rst.txt
    fichier  ./build/html/_sources/c_regex/index.rst.txt
    fichier  ./build/html/_sources/c_regex/regex.rst.txt
    fichier  ./build/html/teachpyx/__init__.html
    fichier  ./build/html/teachpyx/faq/faq_python.html
    fichier  ./build/html/teachpyx/faq/__init__.html
    fichier  ./build/html/teachpyx/faq/faq_exception.html
    fichier  ./build/html/teachpyx/faq/faq_numpy.html
    fichier  ./build/html/teachpyx/examples/construction_classique.html
    fichier  ./build/html/teachpyx/examples/__init__.html
    fichier  ./build/html/teachpyx/examples/numpysex.html
    fichier  ./build/html/teachpyx/examples/classiques.html
    fichier  ./build/html/c_data/index.html
    fichier  ./build/html/c_module/index.html
    fichier  ./build/html/c_module/faq.html
    fichier  ./build/html/c_module/module.html
    fichier  ./build/html/blog/blogindex.html
    fichier  ./build/html/blog/month-2016-09_0000.html
    fichier  ./build/html/blog/cat-debug_0000.html
    fichier  ./build/html/blog/month-2018-03_0000.html
    fichier  ./build/html/blog/main_0000.html
    fichier  ./build/html/blog/cat-weird_0000.html
    fichier  ./build/html/blog/cat-blog_0000.html
    fichier  ./build/html/blog/cat-classe_0000.html
    fichier  ./build/html/blog/cat-pandas_0000.html
    fichier  ./build/html/blog/month-2017-01_0000.html
    fichier  ./build/html/blog/index_blog.html
    fichier  ./build/html/blog/month-2017-07_0000.html
    fichier  ./build/html/blog/cat-module_0000.html
    fichier  ./build/html/blog/month-2018-04_0000.html
    fichier  ./build/html/blog/month-2016-11_0000.html
    fichier  ./build/html/blog/2016/2016-08-18_first_blog.html
    fichier  ./build/html/blog/2016/2016-11-20_groupby_nan.html
    fichier  ./build/html/blog/2018/2018-04-05_debug.html
    fichier  ./build/html/blog/2018/2018-03-31_classesfonctions.html
    fichier  ./build/html/blog/2017/2017-01-06_programming.html
    fichier  ./build/html/blog/2017/2017-07-01_cextension.html
    fichier  ./build/html/c_exception/exception.html
    fichier  ./build/html/c_exception/index.html
    fichier  ./build/html/c_exception/exception_ext.html
    fichier  ./build/html/c_gui/tkinter.html
    fichier  ./build/html/c_gui/index.html
    fichier  ./build/html/c_classes/index.html
    fichier  ./build/html/c_classes/classes.html
    fichier  ./build/html/c_regex/index.html
    fichier  ./build/html/c_regex/regex.html
    fichier  ./source/nbcov-2018-08-28.png
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/index_module.rst
    fichier  ./source/HISTORY.rst
    fichier  ./source/end_index2.rst
    fichier  ./source/end_index_module.rst
    fichier  ./source/doctestunit.rst
    fichier  ./source/i_examples.rst
    fichier  ./source/issues_todoextlist.rst
    fichier  ./source/all_report.rst
    fichier  ./source/completed_todoextlist.rst
    fichier  ./source/README.rst
    fichier  ./source/conf.py
    fichier  ./source/generatedoc.rst
    fichier  ./source/license.rst
    fichier  ./source/all_indexes.rst
    fichier  ./source/index.rst
    fichier  ./source/all_notebooks_coverage.rst
    fichier  ./source/all_notebooks.rst
    fichier  ./source/nbcov.png
    fichier  ./source/i_faq.rst
    fichier  ./source/i_ex.rst
    fichier  ./source/defthe_index.rst
    fichier  ./source/index_function.rst
    fichier  ./source/introduction.rst
    fichier  ./source/LICENSE.txt
    fichier  ./source/end_index_changes.rst
    fichier  ./source/glossary.rst
    fichier  ./source/i_index.rst
    fichier  ./source/filechanges.rst
    fichier  ./source/generatesetup.rst
    fichier  ./source/end_index_glossaire.rst
    fichier  ./source/end_index.rst
    fichier  ./source/installation.rst
    fichier  ./source/notebooks/pandas_groupby.ipynb
    fichier  ./source/notebooks/numpy_tricks.rst
    fichier  ./source/notebooks/serialisation_protobuf.ipynb
    fichier  ./source/notebooks/gil_example.pdf
    fichier  ./source/notebooks/notebook_template.rst
    fichier  ./source/notebooks/numpy_tricks.py
    fichier  ./source/notebooks/notebook_template.thumb.png
    fichier  ./source/notebooks/serialisation_protobuf2html.html
    fichier  ./source/notebooks/serialisation_examples.ipynb
    fichier  ./source/notebooks/numpy_tricks2html.html
    fichier  ./source/notebooks/numpy_tricks.thumb.png
    fichier  ./source/notebooks/float_and_double_rouding_22_1.png
    fichier  ./source/notebooks/gil_example.slides2p.html
    fichier  ./source/notebooks/float_and_double_rouding_21_1.png
    fichier  ./source/notebooks/pandas_groupby.slides.html
    fichier  ./source/notebooks/serialisation_examples2html.html
    fichier  ./source/notebooks/float_and_double_rouding_27_1.png
    fichier  ./source/notebooks/float_and_double_rouding.pdf
    fichier  ./source/notebooks/pandas_groupby.rst
    fichier  ./source/notebooks/numpy_tricks.ipynb
    fichier  ./source/notebooks/pandas_groupby.pdf
    fichier  ./source/notebooks/float_and_double_rouding_25_1.png
    fichier  ./source/notebooks/pandas_groupby2html.html
    fichier  ./source/notebooks/float_and_double_rouding.ipynb
    fichier  ./source/notebooks/float_and_double_rouding_16_0.png
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/notebooks/serialisation_examples.rst
    fichier  ./source/notebooks/float_and_double_rouding2html.html
    fichier  ./source/notebooks/serialisation_protobuf.slides2p.html
    fichier  ./source/notebooks/notebook_template.pdf
    fichier  ./source/notebooks/gil_example.ipynb
    fichier  ./source/notebooks/pandas_groupby.py
    fichier  ./source/notebooks/float_and_double_rouding_29_1.png
    fichier  ./source/notebooks/serialisation_protobuf.slides.html
    fichier  ./source/notebooks/gil_example.py
    fichier  ./source/notebooks/numpy_tricks.slides2p.html
    fichier  ./source/notebooks/serialisation_protobuf.pdf
    fichier  ./source/notebooks/pandas_groupby.thumb.png
    fichier  ./source/notebooks/serialisation_protobuf.rst
    fichier  ./source/notebooks/serialisation_examples.py
    fichier  ./source/notebooks/serialisation_examples.slides2p.html
    fichier  ./source/notebooks/serialisation_examples.pdf
    fichier  ./source/notebooks/float_and_double_rouding.thumb.png
    fichier  ./source/notebooks/serialisation_examples.slides.html
    fichier  ./source/notebooks/gil_example.thumb.png
    fichier  ./source/notebooks/notebook_template.slides2p.html
    fichier  ./source/notebooks/require.js
    fichier  ./source/notebooks/notebook_template.py
    fichier  ./source/notebooks/float_and_double_rouding.rst
    fichier  ./source/notebooks/float_and_double_rouding.slides2p.html
    fichier  ./source/notebooks/numpy_tricks.slides.html
    fichier  ./source/notebooks/float_and_double_rouding_11_0.png
    fichier  ./source/notebooks/notebook_template.ipynb
    fichier  ./source/notebooks/float_and_double_rouding_31_1.png
    fichier  ./source/notebooks/serialisation_protobuf.py
    fichier  ./source/notebooks/gil_example.rst
    fichier  ./source/notebooks/serialisation_examples.thumb.png
    fichier  ./source/notebooks/float_and_double_rouding.py
    fichier  ./source/notebooks/notebook_template.slides.html
    fichier  ./source/notebooks/gil_example2html.html
    fichier  ./source/notebooks/gil_example.slides.html
    fichier  ./source/notebooks/float_and_double_rouding.slides.html
    fichier  ./source/notebooks/numpy_tricks.pdf
    fichier  ./source/notebooks/pandas_groupby.slides2p.html
    fichier  ./source/notebooks/notebook_template2html.html
    fichier  ./source/notebooks/serialisation_protobuf.thumb.png
    fichier  ./source/notebooks/reveal.js/revealjs.css_t
    fichier  ./source/notebooks/reveal.js/package.json
    fichier  ./source/notebooks/reveal.js/__init__.py
    fichier  ./source/notebooks/reveal.js/LICENSE
    fichier  ./source/notebooks/reveal.js/css/__init__.py
    fichier  ./source/notebooks/reveal.js/css/reveal.css
    fichier  ./source/notebooks/reveal.js/css/reveal.scss
    fichier  ./source/notebooks/reveal.js/css/theme/solarized.css
    fichier  ./source/notebooks/reveal.js/css/theme/simple.css
    fichier  ./source/notebooks/reveal.js/css/theme/__init__.py
    fichier  ./source/notebooks/reveal.js/css/theme/black.css
    fichier  ./source/notebooks/reveal.js/css/theme/night.css
    fichier  ./source/notebooks/reveal.js/css/theme/moon.css
    fichier  ./source/notebooks/reveal.js/css/theme/white.css
    fichier  ./source/notebooks/reveal.js/css/theme/beige.css
    fichier  ./source/notebooks/reveal.js/css/theme/serif.css
    fichier  ./source/notebooks/reveal.js/css/theme/league.css
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/notebooks/reveal.js/css/theme/sky.css
    fichier  ./source/notebooks/reveal.js/css/theme/README.md
    fichier  ./source/notebooks/reveal.js/css/theme/blood.css
    fichier  ./source/notebooks/reveal.js/css/theme/source/league.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/black.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/white.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/simple.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/beige.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/__init__.py
    fichier  ./source/notebooks/reveal.js/css/theme/source/serif.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/moon.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/solarized.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/night.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/blood.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/sky.scss
    fichier  ./source/notebooks/reveal.js/css/theme/source/__pycache__/__init__.
↪cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/css/theme/template/settings.scss
    fichier  ./source/notebooks/reveal.js/css/theme/template/__init__.py
    fichier  ./source/notebooks/reveal.js/css/theme/template/mixins.scss
    fichier  ./source/notebooks/reveal.js/css/theme/template/theme.scss
    fichier  ./source/notebooks/reveal.js/css/theme/template/__pycache__/__init__.
↪cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/css/theme/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./source/notebooks/reveal.js/css/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/css/print/__init__.py
    fichier  ./source/notebooks/reveal.js/css/print/pdf.css
    fichier  ./source/notebooks/reveal.js/css/print/paper.css
    fichier  ./source/notebooks/reveal.js/css/print/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./source/notebooks/reveal.js/plugin/__init__.py
    fichier  ./source/notebooks/reveal.js/plugin/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/js/reveal.js
    fichier  ./source/notebooks/reveal.js/js/__init__.py
    fichier  ./source/notebooks/reveal.js/js/jquery.min.js
    fichier  ./source/notebooks/reveal.js/js/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/lib/__init__.py
    fichier  ./source/notebooks/reveal.js/lib/css/__init__.py
    fichier  ./source/notebooks/reveal.js/lib/css/zenburn.css
    fichier  ./source/notebooks/reveal.js/lib/css/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/lib/js/classList.js
    fichier  ./source/notebooks/reveal.js/lib/js/__init__.py
    fichier  ./source/notebooks/reveal.js/lib/js/html5shiv.js
    fichier  ./source/notebooks/reveal.js/lib/js/head.min.js
    fichier  ./source/notebooks/reveal.js/lib/js/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/lib/font/__init__.py
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.ttf
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro.css
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.woff
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.ttf
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.ttf
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.ttf
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.eot
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.eot
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.woff
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.woff
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.eot
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.eot
    fichier  ./source/notebooks/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.woff
    fichier  ./source/notebooks/reveal.js/lib/font/league-gothic/league-gothic.css
    fichier  ./source/notebooks/reveal.js/lib/font/league-gothic/league-gothic.woff
    fichier  ./source/notebooks/reveal.js/lib/font/league-gothic/league-gothic.eot
    fichier  ./source/notebooks/reveal.js/lib/font/league-gothic/league-gothic.ttf
    fichier  ./source/notebooks/reveal.js/lib/font/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/lib/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/notebooks/reveal.js/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/c_io/files.rst
    fichier  ./source/c_io/serialization.rst
    fichier  ./source/c_io/index.rst
    fichier  ./source/c_parallelisation/thread.rst
    fichier  ./source/c_parallelisation/index.rst
    fichier  ./source/c_parallelisation/images/threadim2.png
    fichier  ./source/c_parallelisation/images/asyncapi.png
    fichier  ./source/c_parallelisation/images/threadim1.png
    fichier  ./source/c_parallelisation/images/pool.png
    fichier  ./source/c_resume/conseil_programmes.rst
    fichier  ./source/c_resume/index.rst
    fichier  ./source/c_resume/python_sheet.rst
    fichier  ./source/c_lang/syntaxe.rst
    fichier  ./source/c_lang/encoding.rst
    fichier  ./source/c_lang/constructions.rst
    fichier  ./source/c_lang/index.rst
    fichier  ./source/c_lang/types.rst
    fichier  ./source/c_lang/faq.rst
    fichier  ./source/c_lang/dates.rst
    fichier  ./source/c_lang/images/rawinput.png
    fichier  ./source/gyexamples/lambda_function.rst
    fichier  ./source/gyexamples/gyexamples_python.zip
    fichier  ./source/gyexamples/gyexamples_jupyter.zip
    fichier  ./source/gyexamples/index.rst
    fichier  ./source/gyexamples/lambda_function.ipynb
    fichier  ./source/gyexamples/lambda_function.py
    fichier  ./source/gyexamples/images/thumb/sphx_glr_lambda_function_thumb.png
    fichier  ./source/teachpyx/__init__.py
    fichier  ./source/teachpyx/__init__.rst
    fichier  ./source/teachpyx/faq/faq_exception.py
    fichier  ./source/teachpyx/faq/faq_numpy.rst
    fichier  ./source/teachpyx/faq/__init__.py
    fichier  ./source/teachpyx/faq/faq_numpy.py
    fichier  ./source/teachpyx/faq/__init__.rst
    fichier  ./source/teachpyx/faq/faq_python.py
    fichier  ./source/teachpyx/faq/faq_python.rst
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/teachpyx/faq/faq_exception.rst
    fichier  ./source/teachpyx/faq/__pycache__/faq_python.cpython-37.pyc
    fichier  ./source/teachpyx/faq/__pycache__/faq_numpy.cpython-37.pyc
    fichier  ./source/teachpyx/faq/__pycache__/faq_exception.cpython-37.pyc
    fichier  ./source/teachpyx/faq/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/teachpyx/examples/classiques.py
    fichier  ./source/teachpyx/examples/__init__.py
    fichier  ./source/teachpyx/examples/classiques.rst
    fichier  ./source/teachpyx/examples/numpysex.rst
    fichier  ./source/teachpyx/examples/numpysex.py
    fichier  ./source/teachpyx/examples/__init__.rst
    fichier  ./source/teachpyx/examples/construction_classique.rst
    fichier  ./source/teachpyx/examples/construction_classique.py
    fichier  ./source/teachpyx/examples/__pycache__/classiques.cpython-37.pyc
    fichier  ./source/teachpyx/examples/__pycache__/numpysex.cpython-37.pyc
    fichier  ./source/teachpyx/examples/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/teachpyx/examples/__pycache__/construction_classique.cpython-37.
↪pyc
    fichier  ./source/teachpyx/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/c_data/index.rst
    fichier  ./source/phdoc_static/project_ico_small.png
    fichier  ./source/phdoc_static/git_logo.png
    fichier  ./source/phdoc_static/style_notebook_snippet.css
    fichier  ./source/phdoc_static/project_ico.ico
    fichier  ./source/phdoc_static/project_ico.png
    fichier  ./source/phdoc_static/my-styles.css
    fichier  ./source/phdoc_static/require.js
    fichier  ./source/phdoc_static/reveal.js/revealjs.css_t
    fichier  ./source/phdoc_static/reveal.js/package.json
    fichier  ./source/phdoc_static/reveal.js/__init__.py
    fichier  ./source/phdoc_static/reveal.js/LICENSE
    fichier  ./source/phdoc_static/reveal.js/css/__init__.py
    fichier  ./source/phdoc_static/reveal.js/css/reveal.css
    fichier  ./source/phdoc_static/reveal.js/css/reveal.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/solarized.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/simple.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/__init__.py
    fichier  ./source/phdoc_static/reveal.js/css/theme/black.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/night.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/moon.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/white.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/beige.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/serif.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/league.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/sky.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/README.md
    fichier  ./source/phdoc_static/reveal.js/css/theme/blood.css
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/league.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/black.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/white.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/simple.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/beige.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/__init__.py
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/serif.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/moon.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/solarized.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/night.scss
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/blood.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/sky.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/source/__pycache__/__init__.
↪cpython-37.pyc
    fichier  ./source/phdoc_static/reveal.js/css/theme/template/settings.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/template/__init__.py
    fichier  ./source/phdoc_static/reveal.js/css/theme/template/mixins.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/template/theme.scss
    fichier  ./source/phdoc_static/reveal.js/css/theme/template/__pycache__/__init__.
↪cpython-37.pyc
    fichier  ./source/phdoc_static/reveal.js/css/theme/__pycache__/__init__.cpython-
↪37.pyc
    fichier  ./source/phdoc_static/reveal.js/css/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/phdoc_static/reveal.js/css/print/__init__.py
    fichier  ./source/phdoc_static/reveal.js/css/print/pdf.css
    fichier  ./source/phdoc_static/reveal.js/css/print/paper.css
    fichier  ./source/phdoc_static/reveal.js/css/print/__pycache__/__init__.cpython-
↪37.pyc
    fichier  ./source/phdoc_static/reveal.js/plugin/__init__.py
    fichier  ./source/phdoc_static/reveal.js/plugin/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./source/phdoc_static/reveal.js/js/reveal.js
    fichier  ./source/phdoc_static/reveal.js/js/__init__.py
    fichier  ./source/phdoc_static/reveal.js/js/jquery.min.js
    fichier  ./source/phdoc_static/reveal.js/js/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/phdoc_static/reveal.js/lib/__init__.py
    fichier  ./source/phdoc_static/reveal.js/lib/css/__init__.py
    fichier  ./source/phdoc_static/reveal.js/lib/css/zenburn.css
    fichier  ./source/phdoc_static/reveal.js/lib/css/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./source/phdoc_static/reveal.js/lib/js/classList.js
    fichier  ./source/phdoc_static/reveal.js/lib/js/__init__.py
    fichier  ./source/phdoc_static/reveal.js/lib/js/html5shiv.js
    fichier  ./source/phdoc_static/reveal.js/lib/js/head.min.js
    fichier  ./source/phdoc_static/reveal.js/lib/js/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./source/phdoc_static/reveal.js/lib/font/__init__.py
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.ttf
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro.
↪css
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.woff
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.ttf
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.ttf
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.ttf
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.eot
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.eot
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪regular.woff
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪italic.woff
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibolditalic.eot
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.eot
    fichier  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro/source-sans-pro-
↪semibold.woff
    fichier  ./source/phdoc_static/reveal.js/lib/font/league-gothic/league-gothic.css
    fichier  ./source/phdoc_static/reveal.js/lib/font/league-gothic/league-gothic.woff
    fichier  ./source/phdoc_static/reveal.js/lib/font/league-gothic/league-gothic.eot
    fichier  ./source/phdoc_static/reveal.js/lib/font/league-gothic/league-gothic.ttf
    fichier  ./source/phdoc_static/reveal.js/lib/font/__pycache__/__init__.cpython-37.
↪pyc
    fichier  ./source/phdoc_static/reveal.js/lib/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/phdoc_static/reveal.js/__pycache__/__init__.cpython-37.pyc
    fichier  ./source/c_module/index.rst
    fichier  ./source/c_module/faq.rst
    fichier  ./source/c_module/arbo.png
    fichier  ./source/c_module/module.rst
    fichier  ./source/coverage/.coverage_old
    fichier  ./source/coverage/.coverage
    fichier  ./source/coverage/coverage_report.xml
    fichier  ./source/coverage/src_teachpyx_examples___init___py.html
    fichier  ./source/coverage/coverage_html.js
    fichier  ./source/coverage/src_teachpyx_examples_construction_classique_py.html
    fichier  ./source/coverage/src_teachpyx_faq_faq_python_py.html
    fichier  ./source/coverage/jquery.hotkeys.js
    fichier  ./source/coverage/index.html
    fichier  ./source/coverage/.coverage1~
    fichier  ./source/coverage/src_teachpyx_faq___init___py.html
    fichier  ./source/coverage/jquery.min.js
    fichier  ./source/coverage/src_teachpyx_faq_faq_exception_py.html
    fichier  ./source/coverage/style.css
    fichier  ./source/coverage/src_teachpyx_examples_numpysex_py.html
    fichier  ./source/coverage/status.dat
    fichier  ./source/coverage/src_teachpyx_faq_faq_numpy_py.html
    fichier  ./source/coverage/keybd_closed.png
    fichier  ./source/coverage/status.json
    fichier  ./source/coverage/.coverage0~
    fichier  ./source/coverage/jquery.tablesorter.min.js
    fichier  ./source/coverage/src_teachpyx___init___py.html
    fichier  ./source/coverage/covlog.txt
    fichier  ./source/coverage/src_teachpyx_examples_classiques_py.html
    fichier  ./source/coverage/jquery.isonscreen.js
    fichier  ./source/coverage/keybd_open.png
    fichier  ./source/coverage/jquery.ba-throttle-debounce.min.js
    fichier  ./source/blog/month-2017-01_0000.rst
    fichier  ./source/blog/rss.xml
    fichier  ./source/blog/cat-weird_0000.rst
    fichier  ./source/blog/month-2018-04_0000.rst
    fichier  ./source/blog/blogindex.rst
    fichier  ./source/blog/main_0000.rst
    fichier  ./source/blog/month-2016-09_0000.rst
    fichier  ./source/blog/cat-pandas_0000.rst
    fichier  ./source/blog/month-2017-07_0000.rst
    fichier  ./source/blog/cat-module_0000.rst
    fichier  ./source/blog/month-2016-11_0000.rst
    fichier  ./source/blog/cat-classe_0000.rst
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/blog/cat-debug_0000.rst
    fichier  ./source/blog/cat-blog_0000.rst
    fichier  ./source/blog/feed-icon-16x16.png
    fichier  ./source/blog/index_blog.rst
    fichier  ./source/blog/month-2018-03_0000.rst
    fichier  ./source/blog/2016/2016-08-18_first_blog.rst
    fichier  ./source/blog/2016/2016-11-20_groupby_nan.rst
    fichier  ./source/blog/2018/2018-03-31_classesfonctions.rst
    fichier  ./source/blog/2018/2018-04-05_debug.rst
    fichier  ./source/blog/2017/2017-01-06_programming.rst
    fichier  ./source/blog/2017/2017-07-01_cextension.rst
    fichier  ./source/__pycache__/conf.cpython-37.pyc
    fichier  ./source/c_exception/exception.rst
    fichier  ./source/c_exception/index.rst
    fichier  ./source/c_exception/exception_ext.rst
    fichier  ./source/c_gui/index.rst
    fichier  ./source/c_gui/tkinter.rst
    fichier  ./source/c_gui/images/essai.png
    fichier  ./source/c_gui/images/popt.png
    fichier  ./source/c_gui/images/check2.png
    fichier  ./source/c_gui/images/menu.png
    fichier  ./source/c_gui/images/wiki6.png
    fichier  ./source/c_gui/images/label2.png
    fichier  ./source/c_gui/images/hhw.png
    fichier  ./source/c_gui/images/svn9b.png
    fichier  ./source/c_gui/images/combo2.png
    fichier  ./source/c_gui/images/mysql2.png
    fichier  ./source/c_gui/images/bind.png
    fichier  ./source/c_gui/images/threadim2.png
    fichier  ./source/c_gui/images/can.png
    fichier  ./source/c_gui/images/ligne.png
    fichier  ./source/c_gui/images/list1.png
    fichier  ./source/c_gui/images/bouton.png
    fichier  ./source/c_gui/images/pack1.png
    fichier  ./source/c_gui/images/svn10a.png
    fichier  ./source/c_gui/images/apant2.png
    fichier  ./source/c_gui/images/svn4f.png
    fichier  ./source/c_gui/images/comm1.png
    fichier  ./source/c_gui/images/label.png
    fichier  ./source/c_gui/images/svn8n.png
    fichier  ./source/c_gui/images/focus.png
    fichier  ./source/c_gui/images/frame.png
    fichier  ./source/c_gui/images/comm2.png
    fichier  ./source/c_gui/images/pcmd.png
    fichier  ./source/c_gui/images/reper26.png
    fichier  ./source/c_gui/images/apant4.png
    fichier  ./source/c_gui/images/boostp.png
    fichier  ./source/c_gui/images/scite2.png
    fichier  ./source/c_gui/images/menut.png
    fichier  ./source/c_gui/images/window1.png
    fichier  ./source/c_gui/images/odbc.png
    fichier  ./source/c_gui/images/bbette2.png
    fichier  ./source/c_gui/images/fenpers.png
    fichier  ./source/c_gui/images/radio.png
    fichier  ./source/c_gui/images/inte.png
    fichier  ./source/c_gui/images/mainloop.png
    fichier  ./source/c_gui/images/bouton3.png
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/c_gui/images/svn2ex.png
    fichier  ./source/c_gui/images/pcmd26.png
    fichier  ./source/c_gui/images/pcmd2.png
    fichier  ./source/c_gui/images/threadim1.png
    fichier  ./source/c_gui/images/saisie1.png
    fichier  ./source/c_gui/images/mysql4.png
    fichier  ./source/c_gui/images/mysql8.png
    fichier  ./source/c_gui/images/mysql6.png
    fichier  ./source/c_gui/images/list2.png
    fichier  ./source/c_gui/images/ligne26.png
    fichier  ./source/c_gui/images/pydoch.png
    fichier  ./source/c_gui/images/after.png
    fichier  ./source/c_gui/images/entree.png
    fichier  ./source/c_gui/images/wiki5.png
    fichier  ./source/c_gui/images/profile.png
    fichier  ./source/c_gui/images/svn9a.png
    fichier  ./source/c_gui/images/apant5.png
    fichier  ./source/c_gui/images/bbette.png
    fichier  ./source/c_gui/images/boost.png
    fichier  ./source/c_gui/images/svn5f.png
    fichier  ./source/c_gui/images/pyform1.png
    fichier  ./source/c_gui/images/scite226.png
    fichier  ./source/c_gui/images/check.png
    fichier  ./source/c_gui/images/cmds26.png
    fichier  ./source/c_gui/images/arbo.png
    fichier  ./source/c_gui/images/wiki2.png
    fichier  ./source/c_gui/images/apant1.png
    fichier  ./source/c_gui/images/wiki3.png
    fichier  ./source/c_gui/images/apant0.png
    fichier  ./source/c_gui/images/svn3r.png
    fichier  ./source/c_gui/images/sql1.png
    fichier  ./source/c_gui/images/wiki1.png
    fichier  ./source/c_gui/images/svn12_.png
    fichier  ./source/c_gui/images/pyscripter.png
    fichier  ./source/c_gui/images/svn11a.png
    fichier  ./source/c_gui/images/svn11b.png
    fichier  ./source/c_gui/images/scite1.png
    fichier  ./source/c_gui/images/wiki4.png
    fichier  ./source/c_gui/images/pydoc_image.png
    fichier  ./source/c_gui/images/cmds.png
    fichier  ./source/c_gui/images/mysqlodbc.png
    fichier  ./source/c_gui/images/seqev.png
    fichier  ./source/c_gui/images/text1.png
    fichier  ./source/c_gui/images/grid1.png
    fichier  ./source/c_gui/images/svn72.png
    fichier  ./source/c_gui/images/svn10b.png
    fichier  ./source/c_gui/images/saisie2.png
    fichier  ./source/c_gui/images/pass.png
    fichier  ./source/c_gui/images/config.png
    fichier  ./source/c_gui/images/view.png
    fichier  ./source/c_gui/images/mysql1.png
    fichier  ./source/c_gui/images/tixfile.png
    fichier  ./source/c_gui/images/svn61.png
    fichier  ./source/c_gui/images/svn1.png
    fichier  ./source/c_gui/images/mysql5.png
    fichier  ./source/c_gui/images/svn8nn.png
    fichier  ./source/c_gui/images/graphviz.png
```

(suite sur la page suivante)

(suite de la page précédente)

```
    fichier  ./source/c_gui/images/bouton2.png
    fichier  ./source/c_gui/images/mysqlodbc2.png
    fichier  ./source/c_gui/images/runs.png
    fichier  ./source/c_gui/images/combo1.png
    fichier  ./source/c_gui/images/listboxs.png
    fichier  ./source/c_gui/images/mysql7.png
    fichier  ./source/c_gui/images/pack2.png
    fichier  ./source/c_gui/images/radio2.png
    fichier  ./source/c_gui/images/rawinput.png
    fichier  ./source/c_gui/images/reper.png
    fichier  ./source/c_gui/images/apachein.png
    fichier  ./source/c_gui/images/mysql3.png
    fichier  ./source/c_gui/images/apant3.png
    fichier  ./source/phdoc_templates/searchbox.html
    fichier  ./source/phdoc_templates/moduletoc.html
    fichier  ./source/phdoc_templates/my-styles.css
    fichier  ./source/phdoc_templates/blogtoc.html
    fichier  ./source/phdoc_templates/layout.html
    fichier  ./source/phdoc_templates/page.html
    fichier  ./source/c_classes/index.rst
    fichier  ./source/c_classes/classes.rst
    fichier  ./source/c_regex/index.rst
    fichier  ./source/c_regex/regex.rst
    répertoire  ./build
    répertoire  ./source
    répertoire  ./build/plot_directive
    répertoire  ./build/latex
    répertoire  ./build/doctrees
    répertoire  ./build/html
    répertoire  ./build/doctrees/notebooks
    répertoire  ./build/doctrees/c_io
    répertoire  ./build/doctrees/c_parallelisation
    répertoire  ./build/doctrees/c_resume
    répertoire  ./build/doctrees/c_lang
    répertoire  ./build/doctrees/gyexamples
    répertoire  ./build/doctrees/bokeh_plot
    répertoire  ./build/doctrees/teachpyx
    répertoire  ./build/doctrees/c_data
    répertoire  ./build/doctrees/c_module
    répertoire  ./build/doctrees/blog
    répertoire  ./build/doctrees/c_exception
    répertoire  ./build/doctrees/c_gui
    répertoire  ./build/doctrees/c_classes
    répertoire  ./build/doctrees/c_regex
    répertoire  ./build/doctrees/teachpyx/faq
    répertoire  ./build/doctrees/teachpyx/examples
    répertoire  ./build/doctrees/blog/2016
    répertoire  ./build/doctrees/blog/2018
    répertoire  ./build/doctrees/blog/2017
    répertoire  ./build/html/notebooks
    répertoire  ./build/html/c_io
    répertoire  ./build/html/c_parallelisation
    répertoire  ./build/html/c_resume
    répertoire  ./build/html/c_lang
    répertoire  ./build/html/_static
    répertoire  ./build/html/gyexamples
    répertoire  ./build/html/_downloads
```

(suite sur la page suivante)

(suite de la page précédente)

```
    répertoire  ./build/html/_images
    répertoire  ./build/html/_modules
    répertoire  ./build/html/_sources
    répertoire  ./build/html/teachpyx
    répertoire  ./build/html/c_data
    répertoire  ./build/html/c_module
    répertoire  ./build/html/blog
    répertoire  ./build/html/c_exception
    répertoire  ./build/html/c_gui
    répertoire  ./build/html/c_classes
    répertoire  ./build/html/c_regex
    répertoire  ./build/html/_static/reveal.js
    répertoire  ./build/html/_static/bootswatch-2.3.2
    répertoire  ./build/html/_static/bootstrap-3.3.7
    répertoire  ./build/html/_static/bootswatch-3.3.7
    répertoire  ./build/html/_static/js
    répertoire  ./build/html/_static/sphinxtrib-images
    répertoire  ./build/html/_static/bootstrap-2.3.2
    répertoire  ./build/html/_static/reveal.js/css
    répertoire  ./build/html/_static/reveal.js/plugin
    répertoire  ./build/html/_static/reveal.js/js
    répertoire  ./build/html/_static/reveal.js/lib
    répertoire  ./build/html/_static/reveal.js/__pycache__
    répertoire  ./build/html/_static/reveal.js/css/theme
    répertoire  ./build/html/_static/reveal.js/css/__pycache__
    répertoire  ./build/html/_static/reveal.js/css/print
    répertoire  ./build/html/_static/reveal.js/css/theme/source
    répertoire  ./build/html/_static/reveal.js/css/theme/template
    répertoire  ./build/html/_static/reveal.js/css/theme/__pycache__
    répertoire  ./build/html/_static/reveal.js/css/theme/source/__pycache__
    répertoire  ./build/html/_static/reveal.js/css/theme/template/__pycache__
    répertoire  ./build/html/_static/reveal.js/css/print/__pycache__
    répertoire  ./build/html/_static/reveal.js/plugin/__pycache__
    répertoire  ./build/html/_static/reveal.js/js/__pycache__
    répertoire  ./build/html/_static/reveal.js/lib/css
    répertoire  ./build/html/_static/reveal.js/lib/js
    répertoire  ./build/html/_static/reveal.js/lib/font
    répertoire  ./build/html/_static/reveal.js/lib/__pycache__
    répertoire  ./build/html/_static/reveal.js/lib/css/__pycache__
    répertoire  ./build/html/_static/reveal.js/lib/js/__pycache__
    répertoire  ./build/html/_static/reveal.js/lib/font/source-sans-pro
    répertoire  ./build/html/_static/reveal.js/lib/font/league-gothic
    répertoire  ./build/html/_static/reveal.js/lib/font/__pycache__
    répertoire  ./build/html/_static/bootswatch-2.3.2/journal
    répertoire  ./build/html/_static/bootswatch-2.3.2/simplex
    répertoire  ./build/html/_static/bootswatch-2.3.2/flatly
    répertoire  ./build/html/_static/bootswatch-2.3.2/img
    répertoire  ./build/html/_static/bootswatch-2.3.2/cyborg
    répertoire  ./build/html/_static/bootswatch-2.3.2/cerulean
    répertoire  ./build/html/_static/bootswatch-2.3.2/spruce
    répertoire  ./build/html/_static/bootswatch-2.3.2/spacelab
    répertoire  ./build/html/_static/bootswatch-2.3.2/slate
    répertoire  ./build/html/_static/bootswatch-2.3.2/united
    répertoire  ./build/html/_static/bootswatch-2.3.2/readable
    répertoire  ./build/html/_static/bootswatch-2.3.2/amelia
    répertoire  ./build/html/_static/bootswatch-2.3.2/cosmo
    répertoire  ./build/html/_static/bootswatch-2.3.2/superhero
```

(suite sur la page suivante)

(suite de la page précédente)

```
    répertoire  ./build/html/_static/bootstrap-3.3.7/css
    répertoire  ./build/html/_static/bootstrap-3.3.7/js
    répertoire  ./build/html/_static/bootstrap-3.3.7/fonts
    répertoire  ./build/html/_static/bootswatch-3.3.7/darkly
    répertoire  ./build/html/_static/bootswatch-3.3.7/journal
    répertoire  ./build/html/_static/bootswatch-3.3.7/yeti
    répertoire  ./build/html/_static/bootswatch-3.3.7/simplex
    répertoire  ./build/html/_static/bootswatch-3.3.7/flatly
    répertoire  ./build/html/_static/bootswatch-3.3.7/cyborg
    répertoire  ./build/html/_static/bootswatch-3.3.7/cerulean
    répertoire  ./build/html/_static/bootswatch-3.3.7/spacelab
    répertoire  ./build/html/_static/bootswatch-3.3.7/sandstone
    répertoire  ./build/html/_static/bootswatch-3.3.7/paper
    répertoire  ./build/html/_static/bootswatch-3.3.7/lumen
    répertoire  ./build/html/_static/bootswatch-3.3.7/slate
    répertoire  ./build/html/_static/bootswatch-3.3.7/solar
    répertoire  ./build/html/_static/bootswatch-3.3.7/united
    répertoire  ./build/html/_static/bootswatch-3.3.7/readable
    répertoire  ./build/html/_static/bootswatch-3.3.7/fonts
    répertoire  ./build/html/_static/bootswatch-3.3.7/cosmo
    répertoire  ./build/html/_static/bootswatch-3.3.7/superhero
    répertoire  ./build/html/_static/sphinxtrib-images/LightBox2
    répertoire  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2
    répertoire  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2_customize
    répertoire  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/css
    répertoire  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/img
    répertoire  ./build/html/_static/sphinxtrib-images/LightBox2/lightbox2/js
    répertoire  ./build/html/_static/bootstrap-2.3.2/css
    répertoire  ./build/html/_static/bootstrap-2.3.2/img
    répertoire  ./build/html/_static/bootstrap-2.3.2/js
    répertoire  ./build/html/_images/math
    répertoire  ./build/html/_modules/teachpyx
    répertoire  ./build/html/_modules/teachpyx/faq
    répertoire  ./build/html/_modules/teachpyx/examples
    répertoire  ./build/html/_sources/notebooks
    répertoire  ./build/html/_sources/c_io
    répertoire  ./build/html/_sources/c_parallelisation
    répertoire  ./build/html/_sources/c_resume
    répertoire  ./build/html/_sources/c_lang
    répertoire  ./build/html/_sources/gyexamples
    répertoire  ./build/html/_sources/teachpyx
    répertoire  ./build/html/_sources/c_data
    répertoire  ./build/html/_sources/c_module
    répertoire  ./build/html/_sources/blog
    répertoire  ./build/html/_sources/c_exception
    répertoire  ./build/html/_sources/c_gui
    répertoire  ./build/html/_sources/c_classes
    répertoire  ./build/html/_sources/c_regex
    répertoire  ./build/html/_sources/teachpyx/faq
    répertoire  ./build/html/_sources/teachpyx/examples
    répertoire  ./build/html/_sources/blog/2016
    répertoire  ./build/html/_sources/blog/2018
    répertoire  ./build/html/_sources/blog/2017
    répertoire  ./build/html/teachpyx/faq
    répertoire  ./build/html/teachpyx/examples
    répertoire  ./build/html/blog/2016
    répertoire  ./build/html/blog/2018
```

(suite sur la page suivante)

(suite de la page précédente)

```
    répertoire  ./build/html/blog/2017
    répertoire  ./source/notebooks
    répertoire  ./source/c_io
    répertoire  ./source/c_parallelisation
    répertoire  ./source/c_resume
    répertoire  ./source/c_lang
    répertoire  ./source/gyexamples
    répertoire  ./source/teachpyx
    répertoire  ./source/gallery
    répertoire  ./source/c_data
    répertoire  ./source/phdoc_static
    répertoire  ./source/c_module
    répertoire  ./source/coverage
    répertoire  ./source/blog
    répertoire  ./source/__pycache__
    répertoire  ./source/c_exception
    répertoire  ./source/c_gui
    répertoire  ./source/phdoc_templates
    répertoire  ./source/c_classes
    répertoire  ./source/c_regex
    répertoire  ./source/notebooks/reveal.js
    répertoire  ./source/notebooks/reveal.js/css
    répertoire  ./source/notebooks/reveal.js/plugin
    répertoire  ./source/notebooks/reveal.js/js
    répertoire  ./source/notebooks/reveal.js/lib
    répertoire  ./source/notebooks/reveal.js/__pycache__
    répertoire  ./source/notebooks/reveal.js/css/theme
    répertoire  ./source/notebooks/reveal.js/css/__pycache__
    répertoire  ./source/notebooks/reveal.js/css/print
    répertoire  ./source/notebooks/reveal.js/css/theme/source
    répertoire  ./source/notebooks/reveal.js/css/theme/template
    répertoire  ./source/notebooks/reveal.js/css/theme/__pycache__
    répertoire  ./source/notebooks/reveal.js/css/theme/source/__pycache__
    répertoire  ./source/notebooks/reveal.js/css/theme/template/__pycache__
    répertoire  ./source/notebooks/reveal.js/css/print/__pycache__
    répertoire  ./source/notebooks/reveal.js/plugin/__pycache__
    répertoire  ./source/notebooks/reveal.js/js/__pycache__
    répertoire  ./source/notebooks/reveal.js/lib/css
    répertoire  ./source/notebooks/reveal.js/lib/js
    répertoire  ./source/notebooks/reveal.js/lib/font
    répertoire  ./source/notebooks/reveal.js/lib/__pycache__
    répertoire  ./source/notebooks/reveal.js/lib/css/__pycache__
    répertoire  ./source/notebooks/reveal.js/lib/js/__pycache__
    répertoire  ./source/notebooks/reveal.js/lib/font/source-sans-pro
    répertoire  ./source/notebooks/reveal.js/lib/font/league-gothic
    répertoire  ./source/notebooks/reveal.js/lib/font/__pycache__
    répertoire  ./source/c_parallelisation/images
    répertoire  ./source/c_lang/images
    répertoire  ./source/gyexamples/images
    répertoire  ./source/gyexamples/images/thumb
    répertoire  ./source/teachpyx/faq
    répertoire  ./source/teachpyx/examples
    répertoire  ./source/teachpyx/__pycache__
    répertoire  ./source/teachpyx/faq/__pycache__
    répertoire  ./source/teachpyx/examples/__pycache__
    répertoire  ./source/phdoc_static/reveal.js
    répertoire  ./source/phdoc_static/reveal.js/css
```

(suite sur la page suivante)

(suite de la page précédente)

```
    répertoire  ./source/phdoc_static/reveal.js/plugin
    répertoire  ./source/phdoc_static/reveal.js/js
    répertoire  ./source/phdoc_static/reveal.js/lib
    répertoire  ./source/phdoc_static/reveal.js/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/css/theme
    répertoire  ./source/phdoc_static/reveal.js/css/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/css/print
    répertoire  ./source/phdoc_static/reveal.js/css/theme/source
    répertoire  ./source/phdoc_static/reveal.js/css/theme/template
    répertoire  ./source/phdoc_static/reveal.js/css/theme/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/css/theme/source/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/css/theme/template/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/css/print/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/plugin/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/js/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/lib/css
    répertoire  ./source/phdoc_static/reveal.js/lib/js
    répertoire  ./source/phdoc_static/reveal.js/lib/font
    répertoire  ./source/phdoc_static/reveal.js/lib/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/lib/css/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/lib/js/__pycache__
    répertoire  ./source/phdoc_static/reveal.js/lib/font/source-sans-pro
    répertoire  ./source/phdoc_static/reveal.js/lib/font/league-gothic
    répertoire  ./source/phdoc_static/reveal.js/lib/font/__pycache__
    répertoire  ./source/blog/2016
    répertoire  ./source/blog/2018
    répertoire  ./source/blog/2017
    répertoire  ./source/c_gui/images
```

### 5.1.4 Format binaire

Ecrire et lire des informations au travers d'un fichier texte revient à convertir les informations quel que soit leur type dans un format lisible pour tout utilisateur. Un entier est écrit sous forme de caractères décimaux alors que sa représentation en mémoire est binaire. Cette conversion dans un sens puis dans l'autre est parfois jugée coûteuse en temps de traitement et souvent plus gourmande en terme de taille de fichiers. Un fichier texte compressé, au format *zip* par exemple, est une alternative aux fichiers binaires en terme de taille mais il allonge la lecture et l'écriture par des étapes de compression et de décompression. Même si elle permet de relire les informations écrites grâce à n'importe quel éditeur de texte, il est parfois plus judicieux pour une grande masse d'informations d'utiliser directement le format binaire, c'est-à-dire celui dans lequel elles sont stockées en mémoire. Les informations apparaissent dans leur forme la plus simple pour l'ordinateur : une suite d'octets (bytes en anglais). Deux étapes vont intervenir que ce soit pour l'écriture :

1. On récupère les informations dans une suite d'octets (fonction pack [262] du module struct [263]).
2. On les écrit dans un fichier (méthode write [264] affiliée aux fichiers).

Ou la lecture :

1. On lit une suite d'octets depuis un fichier (méthode read [265] affiliée aux fichiers).
2. On transforme cette suite d'octets pour retrouver l'information qu'elle formait initialement (fonction unpack [266]).

262. https://docs.python.org/3/library/struct.html?highlight=pack#struct.pack
263. https://docs.python.org/3/library/struct.html
264. https://docs.python.org/3.6/library/io.html?highlight=readlines#io.RawIOBase.write
265. https://docs.python.org/3.6/library/io.html?highlight=readlines#io.BufferedIOBase.read
266. https://docs.python.org/3/library/struct.html?highlight=pack#struct.unpack

L'utilisation de fichiers binaires est moins évidente qu'il n'y paraît et il faut faire appel à des modules spécialisés alors que la gestion des fichiers texte ne pose aucun problème. Cela vient du fait que *python* ne donne pas directement accès à la manière dont sont stockées les informations en mémoire contrairement à des langages tels que le C++ [267]. L'intérêt de ces fichiers réside dans le fait que l'information qu'ils contiennent prend moins de place stockée en binaire plutôt que convertie en chaînes de caractères au format texte. Par exemple, un réel est toujours équivalent à huit caractères en format binaire alors que sa conversion au format texte va souvent jusqu'à quinze caractères.

L'écriture et la lecture d'un fichier binaire soulèvent les mêmes problèmes que pour un fichier texte : il faut organiser les données avant de les enregistrer pour savoir comment les retrouver. Les types immuables (réel, entier, caractère) sont assez simples à gérer dans ce format. Pour les objets complexes, *python* propose une solution grâce au module pickle [268] (voir aussi le modile dill [269] pour des types telles que des fonctions).

### Ecriture dans un fichier binaire

L'écriture d'un fichier binaire commence par l'ouverture du fichier en mode écriture par l'instruction `open("<nom_fichier>", "wb")`. C'est le code `"wb"` qui est important (*w* pour *write*, *b* pour *binary*), il spécifie le mode d'ouverture `"w"` et le format `"b"`. La fermeture est la même que pour un fichier texte.

Le module struct [270] et la fonction pack [271] permet de convertir les informations sous forme de chaîne de caractères avant de les enregistrer au format binaire. La fonction pack [272] construit une chaîne de caractères égale au contenu de la mémoire. Son affichage avec la fonction *print* produit quelque chose d'illisible le plus souvent. Le tableau suivant montre les principaux formats de conversion (liste complète [273]) :

- `c` : caractère
- `B` : caractère non signé (octet)
- `i` : entier (4 octets)
- `I` : entier non signé (4 octets)
- `d` : double (8 octets)

L'utilisation de ces codes est illustrée au paragraphe suivant.

### Lecture d'un fichier binaire

Le code associé à l'ouverture d'un fichier binaire en mode lecture est `"rb"`, cela donne : `open("<nom_fichier>", "rb")`. La lecture utilise la fonction unpack [274] pour effectuer la conversion inverse, celle d'une chaîne de caractères en entiers, réels, ... Le paragraphe suivant illustre la lecture et l'écriture au format binaire.

### Exemple fichier binaire

Cet exemple crée un fichier `"info.bin"` puis écrit des informations à l'intérieur. Il ne sera pas possible d'afficher le contenu du fichier à l'aide d'un éditeur de texte.

<<<

```
import struct
# on enregistre un entier, un réel et 4 caractères
i = 10
x = 3.1415692
```

(suite sur la page suivante)

267. https://fr.wikipedia.org/wiki/C%2B%2B
268. https://docs.python.org/3/library/pickle.html
269. https://pypi.python.org/pypi/dill
270. https://docs.python.org/3/library/struct.html
271. https://docs.python.org/3/library/struct.html?highlight=pack#struct.pack
272. https://docs.python.org/3/library/struct.html?highlight=pack#struct.pack
273. https://docs.python.org/3/library/struct.html?highlight=pack#format-characters
274. https://docs.python.org/3/library/struct.html?highlight=pack#struct.unpack

(suite de la page précédente)

```
s = "ABCD"

# écriture
with open("info.bin", "wb") as fb:
    fb.write(struct.pack("i", i))
    fb.write(struct.pack("d", x))
    # il faut convertir les caractères en bytes
    octets = s.encode("ascii")
    fb.write(struct.pack("4s", octets))

# lecture
with open("info.bin", "rb") as fb:
    i = struct.unpack("i",   fb.read(4))
    x = struct.unpack("d",   fb.read(8))
    s = struct.unpack("4s",   fb.read(4))

# affichage pour vérifier que les données ont été bien lues
print(i)
print(x)
print(s)
```

>>>

```
    (10,)
    (3.1415692,)
    (b'ABCD',)
```

Les résultats de la méthode unpack [275] apparaissent dans un tuple mais les données sont correctement récupérées. Ce programme fait aussi apparaître une des particularité du format binaire. On suppose ici que la chaîne de caractères est toujours de longueur 4. En fait, pour stocker une information de dimension variable, il faut d'abord enregistrer cette dimension puis s'en servir lors de la relecture pour connaître le nombre d'octets à lire. On modifie le programme précédent pour sauvegarder une chaîne de caractères de longueur variable.

<<<

```
import struct
# on enregistre un entier, un réel et n caractères
i = 10
x = 3.1415692
s = "ABCDEDF"

# écriture
with open("info.bin", "wb") as fb:
    fb.write(struct.pack("i", i))
    fb.write(struct.pack("d", x))
    r = s.encode("utf-8")
    fb.write(struct.pack("i", len(r)))  # on sauve la dimension de r
    fb.write(struct.pack("{0}s".format(len(r)), r))

# lecture
with open("info.bin", "rb") as fb:
    i = struct.unpack("i", fb.read(4))
    x = struct.unpack("d", fb.read(8))
    size = struct.unpack("i", fb.read(4))  # on récupère la dimension de s
    size = size[0]  # l est un tuple, on s'intéresse à son unique élément
```

(suite sur la page suivante)

275. https://docs.python.org/3/library/struct.html?highlight=pack#struct.unpack

(suite de la page précédente)

```
    s = struct.unpack("{0}s".format(size), fb.read(size))

# affichage pour contrôler
print(i)
print(x)
print(s)
```

>>>

```
    (10,)
    (3.1415692,)
    (b'ABCDEDF',)
```

Cette méthode utilisée pour les chaînes de caractères est applicable aux listes et aux dictionnaires de longueur variable : il faut d'abord stocker leur dimension. Il faut retenir également que la taille d'un réel est de huit octets, celle d'un entier de quatre octets et celle d'un caractère d'un octet. Cette règle est toujours vrai sur des ordinateurs 32 bits. Cette taille varie sur les ordinateurs 64 bits. Le programme suivant donnera la bonne réponse.

<<<

```
from struct import pack
print(len(pack('i', 0)))
print(len(pack('d', 0)))
print(len(pack('s', b'0')))
```

>>>

```
    4
    8
    1
```

Cette taille doit être passée en argument à la méthode `read`.

### Objets plus complexes

Il existe un moyen de sauvegarder dans un fichier des objets plus complexes à l'aide du module pickle [276] Celui-ci permet de stocker dans un fichier le contenu d'un dictionnaire à partir du moment où celui-ci contient des objets standard du langage *python*. Le principe pour l'écriture est le suivant :

```
import pickle

dico = {'a': [1, 2.0, 3, "e"], 'b': ('string', 2), 'c': None}
lis  = [1, 2, 3]

with open ('data.bin', 'wb') as fb:
    pickle.dump(dico, fb)
    pickle.dump(lis, fb)
```

La lecture est aussi simple :

```
with open('data.bin', 'rb') as fb:
    dico = pickle.load(fb)
    lis  = pickle.load(fb)
```

276. https://docs.python.org/3/library/pickle.html

Un des avantages du module pickle [277] est de pouvoir gérer les références circulaires : il est capable d'enregistrer et de relire une liste qui se contient elle-même, ce peut être également une liste qui en contient une autre qui contient la première... Le module *pickle* peut aussi gérer les classes définies par un programmeur à condition qu'elles puissent convertir leur contenu en un dictionnaire dans un sens et dans l'autre, ce qui correspond à la plupart des cas.

<<<

```
import pickle
import copy


class Test:
    def __init__(self):
        self.chaine = "a"
        self.entier = 5
        self.tuple = {"h": 1, 5: "j"}

    def __str__(self):
        return "c='{0}' e={1} t={2}".format(self.chaine, self.entier, self.tuple)


t = Test()

with open('data.bin', 'wb') as fb:  # lecture
    pickle.dump(t, fb)

with open('data.bin', 'rb') as fb:  # écriture
    t = pickle.load(fb)

print(t)
```

>>>

```
    c='a' e=5 t={'h': 1, 5: 'j'}
```

Lorsque la conversion nécessite un traitement spécial, il faut surcharger les opérateurs __getstate__ [278] et __setstate__ [279] Ce cas se produit par exemple lorsqu'il n'est pas nécessaire d'enregistrer tous les attributs de la classe car certains sont calculés ainsi que le montre l'exemple suivant :

<<<

```
import pickle
import copy


class Test:
    def __init__(self):
        self.x = 5
        self.y = 3
        self.calcule_norme()    # attribut calculé

    def calcule_norme(self):
        self.n = (self.x ** 2 + self.y ** 2) ** 0.5

    def __getstate__(self):
        """conversion de Test en un dictionnaire"""
```

(suite sur la page suivante)

277. https://docs.python.org/3/library/pickle.html
278. https://docs.python.org/3/library/pickle.html?highlight=__getstate__#object.__getstate__
279. https://docs.python.org/3/library/pickle.html?highlight=__setstate__#object.__setstate__

(suite de la page précédente)

```
        d = copy.copy(self.__dict__)
        del d["n"]  # attribut calculé, on le sauve pas
        return d

    def __setstate__(self, dic):
        """conversion d'un dictionnaire dic en Test"""
        self.__dict__.update(dic)
        self.calcule_norme()  # attribut calculé

    def __str__(self):
        return "x={0} y={1} n={2}".format(self.x, self.y, self.n)


t = Test()

with open('data.bin', 'wb') as fb:  # lecture
    pickle.dump(t, fb)

with open('data.bin', 'rb') as fb:  # écriture
    t = pickle.load(fb)

print(t)
```

```
>>>
```

```
    x=5 y=3 n=5.830951894845301
```

Le module pickle [280] ne permet de sérialiser tout type d'objet comme les fonctions. Il est parfois utile de sauver une fonction car c'est un paramètre du programme. Il faut dans ce cas soit le faire soi-même, soit utiliser le module dill [281].

## 5.2 Sérialisation

La sérialisation [282] est un besoin fréquent depuis l'avènement d'internet. Une appplication web est souvent un assemblage de services indépendant qui s'échangent des informations complexes. Un service doit transmettre des données à un autre. L'opération est simple lorsqu'il s'agit de transmettre un nombre réel, un entier, une chaîne de caractère. C'est déjà un peu plus compliqué lorsque les données à transmettre sont constituées de listes et de dictionnaires. Elle est complexe lorsque ce sont des instances de classes définies par le programme lui-même. Or les services s'échangent uniquement des octets (ou *byte* en anglais). Il faut donc convertir des informations complexes en une séquence d'octets puis effectuer la conversion inverse. Ces opérations sont regroupées sous le terme de *sérialisation*.


— *JSON* (page 214)
— *Binaire* (page 217)
— *Optimisation* (page 219)
— *Sérialiser autre chose que des données* (page 226)


280. https://docs.python.org/3/library/pickle.html
281. https://pypi.python.org/pypi/dill
282. https://fr.wikipedia.org/wiki/S%C3%A9rialisation

### 5.2.1 JSON

Un des premiers utilisés dans le monde internet est le format XML [283]. Le langage HTML [284] suit la même logique que le format *XML*. Il peut représenter un assemblage de liste et de dictionnaires qui contiennent des nombres et des chaînes de caractères.

```
<xml>
    <personne>                          <!-- dictionnaire -->
        <nom>Dupré</nom>                <!-- str -->
        <prenom>Xavier</prenom>         <!-- str -->
        <langages>                      <!-- liste -->
            <lang>                      <!-- dictionnaire -->
                <nom>C++</nom>          <!-- str -->
                <vitesse>rapide</nom>   <!-- str -->
                <age>40</age>           <!-- int -->
            </lang>
            <lang>                      <!-- dictionnaire -->
                <nom>python</nom>       <!-- str -->
                <vitesse>lente</nom>    <!-- str -->
                <age>20</age>           <!-- int -->
            </lang>
        </langages>
    </personnes>
</xml>
```

La sérialisation passe par une étape intermédiaire qui revient à convertir tout type de variables en une séquence de listes et de dictionnaires puis à la convertir sous la forme d'une séquence d'octets.

Le format XML [285] intervient lors de la seconde étape. Il est assez *verbeux*. La séquence d'octets ou plutôt de caractères qu'il produit est assez longue même une fois les espaces enlevés. On lui préfère le format JSON [286] qui est très utilisé pour communiquer entre les services via des API REST [287] car il est à la fois concis et lisible.

<<<

```
from json import dump
from io import StringIO

data = {'records': [{'nom': 'Xavier', 'prénom': 'Xavier',
                     'langages': [{'nom': 'C++', 'age': 40},
                                  {'nom': 'Python', 'age': 20}]}]}

buffer = StringIO()
res = dump(data, buffer)
seq = buffer.getvalue()
print(seq)
```

283. https://fr.wikipedia.org/wiki/Extensible_Markup_Language
284. https://fr.wikipedia.org/wiki/Hypertext_Markup_Language
285. https://fr.wikipedia.org/wiki/Extensible_Markup_Language
286. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
287. https://fr.wikipedia.org/wiki/Representational_state_transfer

>>>

```
    {"records": [{"nom": "Xavier", "pr\u00e9nom": "Xavier", "langages": [{"nom": "C++
↪", "age": 40}, {"nom": "Python", "age": 20}]}]}
```

Le format JSON [288] est très proche de la syntaxe du langage Python [289]. Le résultat apparaît sous la forme d'une chaîne de caractère et est facile à transmettre à n'importe quelle autre application. La conversion inverse s'effectue comme suit.

<<<

```
from json import load
from io import StringIO

seq = '{"records": [{"nom": "Xavier", "pr\\u00e9nom": "Xavier", ' + \
      '"langages": [{"nom": "C++", "age": 40}, {"nom": "Python", "age": 20}]}]}'

buffer = StringIO(seq)
read = load(buffer)
print(read)
```

>>>

```
    {'records': [{'nom': 'Xavier', 'prénom': 'Xavier', 'langages': [{'nom': 'C++',
↪'age': 40}, {'nom': 'Python', 'age': 20}]}]}
```

Tous les types ne sont pas sérialisables même si ceux-ci sont fréquemment utilisés :

<<<

```
from json import dump
from io import StringIO
import numpy

data = {'mat': numpy.array([0, 1])}

buffer = StringIO()
try:
    dump(data, buffer)
except Exception as e:
    print(e)
```

>>>

```
    Object of type ndarray is not JSON serializable
```

Les classes définies dans le programme ne sont pas non plus sérialisables par défaut.

<<<

```
from json import dump, JSONEncoder
from io import StringIO


class A:
    def __init__(self, att):
        self.att = att
```

(suite sur la page suivante)

288. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
289. http://www.python.org/

(suite de la page précédente)

```
class MyEncoder(JSONEncoder):
    def default(self, o):
        return {'classname': o.__class__.__name__, 'data': o.__dict__}


data = A('e')
buffer = StringIO()
try:
    dump(data, buffer, cls=MyEncoder)
except Exception as e:
    print(e)
```

>>>

```

```

On peut néanmoins contourner l'obstacle en indiquant à la fonction dump [290] comment convertir l'instance en un assemblage de listes et de dictionnaires avec la classe JSONEncoder [291].

<<<

```
from json import dump, JSONEncoder
from io import StringIO


class A:
    def __init__(self, att):
        self.att = att


class MyEncoder(JSONEncoder):
    def default(self, o):
        return {'classname': o.__class__.__name__, 'data': o.__dict__}


data = A('e')
buffer = StringIO()
res = dump(data, buffer, cls=MyEncoder)
res = buffer.getvalue()
print(res)
```

>>>

```
    {"classname": "A", "data": {"att": "e"}}
```

Et la relecture s'effectue comme suit :

<<<

```
from json import load, JSONDecoder
from io import StringIO


class A:
```

(suite sur la page suivante)

290. https://docs.python.org/3/library/json.html#json.dump
291. https://docs.python.org/3/library/json.html#json.JSONEncoder

(suite de la page précédente)

```
    def __init__(self, att):
        self.att = att


class MyDecoder(JSONDecoder):
    def decode(self, o):
        dec = JSONDecoder.decode(self, o)
        if isinstance(dec, dict) and dec.get('classname') == 'A':
            return A(dec['data']['att'])
        else:
            return dec


res = '{"classname": "A", "data": {"att": "e"}}'
buffer = StringIO(res)
obj = load(buffer, cls=MyDecoder)
print(obj)
```

>>>

```
    <pyquickhelper.sphinxext.sphinx_runpython_extension.run_python_script_
↪139651612975536.<locals>.A object at 0x7f032d2b86d8>
```

Il existe des alternatives plus rapides au module json [292] come le module ultrajson [293].

### 5.2.2 Binaire

Le format *binaire* n'est pas un format à proprement parler. Le problème principale lié à l'utilisation d'un format comme JSON [294] est le temps passé à convertir les informations numériques en texte et réciproquement. Il est plus efficace de conserver cette information dans un format plus proche de celui de la mémoire. Cette remarque est aussi vrai pour tous les autre types. Lorsque la sérialisation est nécessaire au sein d'une même application, il est nettement préférable de choisir ce type de format même s'il est illisible pour un autre langage voire même pour un autre programme. Le code ressemble beaucoup à ce qui a été écrit pour le format JSON [295].

<<<

```
from pickle import dump
from io import BytesIO

data = {'records': [{'nom': 'Xavier', 'prénom': 'Xavier',
                     'langages': [{'nom': 'C++', 'age': 40},
                                  {'nom': 'Python', 'age': 20}]}]}

buffer = BytesIO()
res = dump(data, buffer)
seq = buffer.getvalue()
print(seq)
```

>>>

```
    b'\x80\x03}q\x00X\x07\x00\x00\x00recordsq\x01]q\x02}
↪q\x03(X\x03\x00\x00\x00nomq\x04X\x06\x00\x00\x00Xavierq\x05X\x07\x00\x00\x00pr\xc3\xa9nomq\x06h\x05
↪q\t(h\x04X\x03\x00\x00\x00C++q\nX\x03\x00\x00\x00ageq\x0bK(u}
↪q\x0c(h\x04X\x06\x00\x00\x00Pythonq\rh\x0bK\x14ueuas.'
```

(suite sur la page suivante)

292. https://docs.python.org/3/library/json.html
293. https://github.com/esnme/ultrajson
294. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
295. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

(suite de la page précédente)

La relecture s'effectue tout aussi simplement.

<<<

```
from pickle import load, dump
from io import BytesIO

data = {'records': [{'nom': 'Xavier', 'prénom': 'Xavier',
                     'langages': [{'nom': 'C++', 'age': 40},
                                  {'nom': 'Python', 'age': 20}]}]}

# écriture
buffer = BytesIO()
res = dump(data, buffer)
seq = buffer.getvalue()

# lecture
buffer = BytesIO(seq)
read = load(buffer)
print(read)
```

>>>

```
    {'records': [{'nom': 'Xavier', 'prénom': 'Xavier', 'langages': [{'nom': 'C++',
↪'age': 40}, {'nom': 'Python', 'age': 20}]}]}
```

Certaines configurations impossible, un peu comme pour le format JSON [296], de modifier les éléments sérialisés et restaurés. Cela se fait grâce aux fonctions __getstate__ [297] et __setstate__ [298]. Dans l'exemple suivant, la classe copie un attribut deux fois. Pour réduire la taille de l'objet une fois sérialisé, on en stocke qu'un seul.

<<<

```
from io import BytesIO
from pickle import dump, load


class B:
    def __init__(self, att):
        self.att1 = att
        self.att2 = att

    def __getstate__(self):
        # On indique les attributs à converser.
        return dict(att=self.att1)

    def __setstate__(self, state):
        # On restaure l'objet à partir de ce qui a été sérialisé.
        setattr(self, 'att1', state["att"])
        setattr(self, 'att2', state["att"])


data = B('r')
buffer = BytesIO()
```

(suite sur la page suivante)

296. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
297. https://docs.python.org/3/library/pickle.html#object.__getstate__
298. https://docs.python.org/3/library/pickle.html#object.__setstate__

(suite de la page précédente)

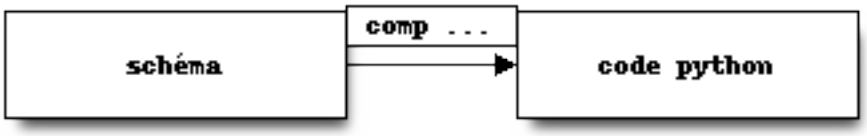
```
res = dump(data, buffer)
seq = buffer.getvalue()

buffer = BytesIO(seq)
read = load(buffer)
print(read.att1, read.att2)
```

```
>>>
```

```
    r r
```

### 5.2.3 Optimisation

La communication entre services devient parfois cruciale pour assurer un service temps réel. Le premier enjeu est d'écrire ou de lire rapidement. Le second est de pouvoir accéder à une information sans avoir à déchiffrer l'ensemble des données sérialisées. Pour ces deux objectifs, la solution la plus aboutie est la librarie protobuf [299]. L'efficacité est gagnée en imposant un format strict et figé aux données. La première étape consiste à définir un schéma de donnée, qui est convertit ensuite en un programme qui permette de sérialiser, désérialiser et accéder à une information précise dans le language de votre choix.




La suite est dans le notebook :

#### Sérialisation avec protobuf

protobuf [300] optimise la sérialisation de deux façons. Elle accélère l'écriture et la lecture des données et permet aussi un accès rapide à une information précise dans désérialiser les autres. Elle réalise cela en imposant un schéma strict de données.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


— *Schéma* (page 220)
— *Compilation* (page 220)
— *Import du module créé* (page 222)
— *Sérialisation en chaîne de caractères* (page 223)
— *Plusieurs chaînes de caractères* (page 224)
— *Sérialisation JSON* (page 225)


299. https://developers.google.com/protocol-buffers/
300. https://fr.wikipedia.org/wiki/Protocol_Buffers

### Schéma

On récupère l'exemple du tutorial [301].

```
schema = """
syntax = "proto2";

package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
"""
```

### Compilation

Il faut d'abord récupérer le compilateur. Cela peut se faire depuis le site de protobuf [302] ou sur Linux (Ubuntu/Debian) `apt-get install protobuf-compiler` pour obtenir le programme `protoc`.

```
import google.protobuf as gp
version = gp.__version__
if version == "3.5.2.post1":
    version = "3.5.1"
version
```

```
'3.5.1'
```

```
import sys, os

if sys.platform.startswith("win"):
    url = "https://github.com/google/protobuf/releases/download/v{0}/protoc-{0}-win32.
↪zip".format(version)
    name = "protoc-{0}-win32.zip".format(version)
    exe = 'protoc.exe'
```

(suite sur la page suivante)

301. https://developers.google.com/protocol-buffers/docs/pythontutorial
302. https://github.com/google/protobuf/releases

(suite de la page précédente)

```
else:
    url = "https://github.com/google/protobuf/releases/download/v{0}/protoc-{0}-linux-
↪x86_64.zip".format(version)
    exe = 'protoc'
    name = "protoc-{0}-linux-x86_64.zip".format(version)

protoc = os.path.join("bin", exe)
if not os.path.exists(name):
    from pyquickhelper.filehelper import download
    try:
        download(url)
    except Exception as e:
        raise Exception("Unable to download '{0}'\nERROR\n{1}".format(url, e))
else:
    print(name)
```

```
protoc-3.5.1-win32.zip
```

```
if not os.path.exists(protoc):
    from pyquickhelper.filehelper import unzip_files
    unzip_files(name,where_to='.')
```

```
if not os.path.exists(protoc):
    raise FileNotFoundError(protoc)
```

On écrit le format sur disque.

```
with open('schema.proto', 'w') as f:
    f.write(schema)
```

Et on peut compiler.

```
from pyquickhelper.loghelper import run_cmd
cmd = '{0} --python_out=. schema.proto'.format(protoc)
try:
    out, err = run_cmd(cmd=cmd, wait=True)
except PermissionError as e:
    # Sous Linux si ne marche pas avec bin/protoc, on utilise
    # protoc directement à supposer que le package
    # protobuf-compiler a été installé.
    if not sys.platform.startswith("win"):
        protoc = "protoc"
        cmd = '{0} --python_out=. schema.proto'.format(protoc)
        try:
            out, err = run_cmd(cmd=cmd, wait=True)
        except Exception as e:
            mes = "CMD: {0}".format(cmd)
            raise Exception("Unable to use {0}\n{1}".format(protoc, mes)) from e
    else:
        mes = "CMD: {0}".format(cmd)
        raise Exception("Unable to use {0}\n{1}".format(protoc, mes)) from e
print("\n----\n".join([out, err]))
```

```
----
```

Un fichier a été généré.

```
[_ for _ in os.listdir(".") if '.py' in _]
```

```
['schema_pb2.py']
```

```
with open('schema_pb2.py', 'r') as f:
    content = f.read()
print(content[:1000])
```

```
# Generated by the protocol buffer compiler.  DO NOT EDIT!
# source: schema.proto
import sys
_b=sys.version_info[0]<3 and (lambda x:x) or (lambda x:x.encode('latin1'))
from google.protobuf import descriptor as _descriptor
from google.protobuf import message as _message
from google.protobuf import reflection as _reflection
from google.protobuf import symbol_database as _symbol_database
from google.protobuf import descriptor_pb2
# @@protoc_insertion_point(imports)
_sym_db = _symbol_database.Default()



DESCRIPTOR = _descriptor.FileDescriptor(
  name='schema.proto',
  package='tutorial',
  syntax='proto2',
  serialized_pb=_b('nx0cschema.protox12x08tutorial"xdbx01nx06Personx12x0cnx04namex18x01␣
↪x02(tx12nnx02idx18x02 x02(x05x12rnx05x65mailx18x03 x01(tx12,
↪nx06phonesx18x04 x03(x0bx32x1c.tutorial.Person.PhoneNumberx1aMnx0bPhoneNumberx12x0enx06nu
↪x02(tx12.nx04typex18x02 x01(x0ex32x1a.tutorial.Person.PhoneType:x04HOME"
```

### Import du module créé

Pour utliser *protobuf*, il faut importer le module créé.

```
import schema_pb2
```

On créé un enregistrement.

```
person = schema_pb2.Person()
person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"
phone = person.phones.add()
phone.number = "555-4321"
phone.type = schema_pb2.Person.HOME
```

```
person
```

```
name: "John Doe"
id: 1234
email: "jdoe@example.com"
phones {
```

(suite sur la page suivante)

(suite de la page précédente)

```
  number: "555-4321"
  type: HOME
}
```

### Sérialisation en chaîne de caractères

```
res = person.SerializeToString()
type(res), res
```

```
(bytes,
 b'nx08John Doex10xd2tx1ax10jdoe@example.com"x0cnx08555-4321x10x01')
```

```
%timeit person.SerializeToString()
```

```
4.56 ţs ± 447 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
pers = schema_pb2.Person.FromString(res)
pers
```

```
name: "John Doe"
id: 1234
email: "jdoe@example.com"
phones {
  number: "555-4321"
  type: HOME
}
```

```
pers = schema_pb2.Person()
pers.ParseFromString(res)
pers
```

```
name: "John Doe"
id: 1234
email: "jdoe@example.com"
phones {
  number: "555-4321"
  type: HOME
}
```

```
%timeit schema_pb2.Person.FromString(res)
```

```
3.44 ţs ± 696 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit pers.ParseFromString(res)
```

```
3.13 ţs ± 633 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

### Plusieurs chaînes de caractères

```
db = []

person = schema_pb2.Person()
person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"
phone = person.phones.add()
phone.number = "555-4321"
phone.type = schema_pb2.Person.HOME
db.append(person)

person = schema_pb2.Person()
person.id = 5678
person.name = "Johnette Doette"
person.email = "jtdoet@example2.com"
phone = person.phones.add()
phone.number = "777-1234"
phone.type = schema_pb2.Person.MOBILE
db.append(person)
```

```
import struct
from io import BytesIO
buffer = BytesIO()
for p in db:
    size = p.ByteSize()
    buffer.write(struct.pack('i', size))
    buffer.write(p.SerializeToString())
res = buffer.getvalue()
res
```

```
b'-x00x00x00nx08John Doex10xd2tx1ax10jdoe@example.com"x0cnx08555-
↪4321x10x017x00x00x00nx0fJohnette Doettex10xae,x1ax13jtdoet@example2.
↪com"x0cnx08777-1234x10x00'
```

```
from google.protobuf.internal.decoder import _DecodeVarint32
db2 = []
buffer = BytesIO(res)
n = 0
while True:
    bsize = buffer.read(4)
    if len(bsize) == 0:
        # C'est fini.
        break
    size = struct.unpack('i', bsize)[0]
    data = buffer.read(size)
    p = schema_pb2.Person.FromString(data)
    db2.append(p)
```

```
db2[0], db2[1]
```

```
(name: "John Doe"
 id: 1234
 email: "jdoe@example.com"
 phones {
```

(suite sur la page suivante)

(suite de la page précédente)

```
  number: "555-4321"
  type: HOME
}, name: "Johnette Doette"
id: 5678
email: "jtdoet@example2.com"
phones {
  number: "777-1234"
  type: MOBILE
})
```

### Sérialisation JSON

```
from google.protobuf.json_format import MessageToJson
```

```
print(MessageToJson(pers))
```

```
{
  "name": "John Doe",
  "id": 1234,
  "email": "jdoe@example.com",
  "phones": [
    {
      "number": "555-4321",
      "type": "HOME"
    }
  ]
}
```

```
%timeit MessageToJson(pers)
```

```
76.4 µs ± 7.48 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
from google.protobuf.json_format import Parse as ParseJson
js = MessageToJson(pers)
res = ParseJson(js, message=schema_pb2.Person())
res
```

```
name: "John Doe"
id: 1234
email: "jdoe@example.com"
phones {
  number: "555-4321"
  type: HOME
}
```

```
%timeit ParseJson(js, message=schema_pb2.Person())
```

```
75 µs ± 7.77 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

### 5.2.4 Sérialiser autre chose que des données

D'une manière générale, il vaut mieux éviter de sérialiser les fonctions, itérateurs ou générateurs. Il est souvent impossible des les sérialiser proprement sans ajouter du code spécifiquement développés pour cela. Le notebook suivant étudie certains cas où cela est néanmoins possible sans trop de développement.

#### Sérialisation

Le notebook explore différentes façons de sérialiser des données et leurs limites.

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```


- *JSON* (page 226)
  - *Ecriture* (page 226)
  - *Lecture* (page 227)
  - *Limite* (page 227)
  - *Sérialisation rapide* (page 228)
- *Pickle* (page 229)
  - *Ecriture* (page 229)
  - *Lecture* (page 229)
  - *Les classes* (page 229)
  - *Réduire la taille* (page 230)
- *Cas des fonctions* (page 231)
  - *Binaire* (page 231)
  - *JSON* (page 233)
- *Cas des itérateurs* (page 233)
- *Cas des générateurs* (page 234)


#### JSON

Le format JSON [303] est le format le plus utilisé sur internet notemmant via les API REST [304].

#### Ecriture

```
data = {'records': [{'nom': 'Xavier', 'prénom': 'Xavier',
                     'langages':[{'nom':'C++', 'age':40}, {'nom':'Python', 'age': 20}
↪]}]}
```

```
from json import dump
from io import StringIO
buffer = StringIO()
res = dump(data, buffer)
seq = buffer.getvalue()
seq
```

```
'{"records": [{"nom": "Xavier", "pr\u00e9nom": "Xavier", "langages": [{"nom":␣
↪"C++", "age": 40}, {"nom": "Python", "age": 20}]}]}'
```

303. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
304. https://fr.wikipedia.org/wiki/Representational_state_transfer

### Lecture

```
from json import load
buffer = StringIO(seq)
read = load(buffer)
read
```

```
{'records': [{'nom': 'Xavier',
   'prénom': 'Xavier',
   'langages': [{'nom': 'C++', 'age': 40}, {'nom': 'Python', 'age': 20}]}]}
```

### Limite

Les matrices numpy [305] ne sont pas sérialisables facilement.

```
import numpy
data = {'mat': numpy.array([0, 1])}

buffer = StringIO()
try:
    dump(data, buffer)
except Exception as e:
    print(e)
```

```
Object of type 'ndarray' is not JSON serializable
```

Les classes ne sont pas sérialisables non plus facilement.m

```
class A:
    def __init__(self, att):
        self.att = att

data = A('e')
buffer = StringIO()
try:
    dump(data, buffer)
except Exception as e:
    print(e)
```

```
Object of type 'A' is not JSON serializable
```

Pour ce faire, il faut indiquer au module **'json <>'__** comment convertir la classe en un ensemble de listes et dictionnaires et la classe JSONEncoder [306].

```
from json import JSONEncoder
class MyEncoder(JSONEncoder):
        def default(self, o):
            return {'classname': o.__class__.__name__, 'data': o.__dict__}

data = A('e')
buffer = StringIO()
res = dump(data, buffer, cls=MyEncoder)
```

(suite sur la page suivante)

305. http://www.numpy.org/
306. https://docs.python.org/3/library/json.html#json.JSONEncoder

(suite de la page précédente)

```
res = buffer.getvalue()
res
```

```
'{"classname": "A", "data": {"att": "e"}}'
```

Et la relecture avec la classe JSONDecoder [307].

```
from json import JSONDecoder

class MyDecoder(JSONDecoder):
        def decode(self, o):
            dec = JSONDecoder.decode(self, o)
            if isinstance(dec, dict) and dec.get('classname') == 'A':
                return A(dec['data']['att'])
            else:
                return dec

buffer = StringIO(res)
obj = load(buffer, cls=MyDecoder)
obj
```

```
<__main__.A at 0x1d8a0b46e48>
```

### Sérialisation rapide

Le module **'json <>'__** est la librairie standard de Python mais comme la sérialisation au format *JSON* est un besoin très fréquent, il existe des alternative plus rapide comme ujson [308].

```
data = {'records': [{'nom': 'Xavier', 'prénom': 'Xavier',
                     'langages':[{'nom':'C++', 'age':40}, {'nom':'Python', 'age': 20}
↪]}]}
```

```
%timeit dump(data, StringIO())
```

```
29 µs ± 1.4 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
from ujson import dump as udump
%timeit udump(data, StringIO())
```

```
2.62 µs ± 80.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Ces deux lignes mesures l'écriture au format JSON mais il faut aussi mesurer la lecture.

```
buffer = StringIO()
dump(data, buffer)
res = buffer.getvalue()
%timeit load(StringIO(res))
```

```
13.4 µs ± 1.72 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

307. https://docs.python.org/3/library/json.html#json.JSONDecoder
308. https://docs.micropython.org/en/latest/pyboard/library/ujson.html

```
from ujson import load as uload
%timeit uload(StringIO(res))
```

```
5.42 µs ± 728 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

On enlève le temps passé dans la creation du buffer.

```
%timeit StringIO(res)
```

```
1.11 µs ± 56.1 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

### Pickle

Le module pickle [309] effectue la même chose mais au format binaire. Celui-ci est propre à *Python* et ne peut être lu d'autres langages, voire parfois par d'autres versions de .Python*.

### Ecriture

```
data = {'records': [{'nom': 'Xavier', 'prénom': 'Xavier',
                     'langages':[{'nom':'C++', 'age':40}, {'nom':'Python', 'age': 20}
↪]}]}
```

```
from pickle import dump
from io import BytesIO
buffer = BytesIO()
res = dump(data, buffer)
seq = buffer.getvalue()
seq
```

```
b'x80x03}qx00Xx07x00x00x00recordsqx01]qx02}qx03(Xx03x00x00x00nomqx04Xx06x00x00x00Xavierqx05
↪qt(hx04Xx03x00x00x00C++qnXx03x00x00x00ageqx0bK(u}qx0c(hx04Xx06x00x00x00Pythonqrhx0bKx14ue
↪'
```

### Lecture

```
from pickle import load
buffer = BytesIO(seq)
read = load(buffer)
read
```

```
{'records': [{'nom': 'Xavier',
   'prénom': 'Xavier',
   'langages': [{'nom': 'C++', 'age': 40}, {'nom': 'Python', 'age': 20}]}]}
```

### Les classes

A l'inverse du format *JSON*, les classes sont sérialisables avec *pickle* parce que le langage utilise un format très proche de ce qu'il a en mémoire. Il n'a pas besoin de conversion supplémentaire.

309. https://docs.python.org/3/library/pickle.html

```
data = A('r')
buffer = BytesIO()
res = dump(data, buffer)
seq = buffer.getvalue()
seq
```

```
b'x80x03c__main__nAnqx00)x81qx01}qx02Xx03x00x00x00attqx03Xx01x00x00x00rqx04sb.
↪'
```

```
buffer = BytesIO(seq)
read = load(buffer)
read
```

```
<__main__.A at 0x1d8a0b398d0>
```

### Réduire la taille

Certaines informations sont duppliquées et il est préférable de ne pas les sérialiser deux fois surtout si elles sont voluminueuses.

```
class B:
    def __init__(self, att):
        self.att1 = att
        self.att2 = att
```

```
data = B('r')
buffer = BytesIO()
res = dump(data, buffer)
seq = buffer.getvalue()
seq
```

```
b'x80x03c__main__nBnqx00)x81qx01}qx02(Xx04x00x00x00att1qx03Xx01x00x00x00rqx04Xx04x00x00x00a
↪'
```

Evitons maintenant de stocker deux fois le même attribut.

```
class B:
    def __init__(self, att):
        self.att1 = att
        self.att2 = att

    def __getstate__(self):
        return dict(att=self.att1)

data = B('r')
buffer = BytesIO()
res = dump(data, buffer)
seq = buffer.getvalue()
seq
```

```
b'x80x03c__main__nBnqx00)x81qx01}qx02Xx03x00x00x00attqx03Xx01x00x00x00rqx04sb.
↪'
```

C'est plus court mais il faut inclure maintenant la relecture.

```
class B:
    def __init__(self, att):
        self.att1 = att
        self.att2 = att

    def __getstate__(self):
        return dict(att=self.att1)

    def __setstate__(self, state):
        setattr(self, 'att1', state["att"])
        setattr(self, 'att2', state["att"])

buffer = BytesIO(seq)
read = load(buffer)
read
```

```
<__main__.B at 0x1d8a0bb4ef0>
```

```
read.att1, read.att2
```

```
('r', 'r')
```

```
data = B('r')
%timeit dump(data, BytesIO())
```

```
5.32 µs ± 542 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit load(BytesIO(seq))
```

```
7.79 µs ± 147 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

La sérialisation binaire est habituellement plus rapide dans les langages bas niveau comme C++. La même comparaison pour un langage haut niveau tel que Python n'est pas toujours prévisible. Il est possible d'accélérer un peu les choses.

```
from pickle import HIGHEST_PROTOCOL
%timeit dump(data, BytesIO(), protocol=HIGHEST_PROTOCOL)
```

```
3.37 µs ± 209 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

### Cas des fonctions

La sérialisation s'applique à des données et non à du code mais le fait de sérialiser des fonctions est tout de même tentant. La sérialisation binaire fonctionne même avec les fonctions.

### Binaire

```
def myfunc(x):
    return x + 1

data = {'x': 5, 'f': myfunc}
```

(suite sur la page suivante)

(suite de la page précédente)

```
from pickle import dump
from io import BytesIO
buffer = BytesIO()
res = dump(data, buffer)
buffer.getvalue()
```

```
b'x80x03}qx00(Xx01x00x00x00xqx01Kx05Xx01x00x00x00fqx02c__main__nmyfuncnqx03u.'
```

```
from pickle import load
res = load(BytesIO(buffer.getvalue()))
res
```

```
{'x': 5, 'f': <function __main__.myfunc(x)>}
```

```
res['f'](res['x'])
```

```
6
```

La sérialisation ne conserve pas le code de la fonction, juste son nom. Cela veut dire que si elle n'est pas disponible lorsqu'elle est appelé, il sera impossible de s'en servir.

```
del myfunc

from pickle import load
try:
    load(BytesIO(buffer.getvalue()))
except Exception as e:
    print(e)
```

```
Can't get attribute 'myfunc' on <module '__main__'>
```

Il est possible de contourner l'obstacle en utilisant le module cloudpicke [310] qui stocke le code de la fonction.

```
def myfunc(x):
    return x + 1

data = {'x': 5, 'f': myfunc}

from cloudpickle import dump
from io import BytesIO
buffer = BytesIO()
res = dump(data, buffer)
buffer.getvalue()
```

```
b'x80x04x954x01x00x00x00x00x00}x94(x8cx01xx94Kx05x8cx01fx94x8cx17cloudpickle.
 ↪cloudpicklex94x8cx0e_fill_functionx94x93x94(hx03x8cx0f_make_skel_
 ↪funcx94x93x94hx03x8cr_builtin_typex94x93x94x8cx08CodeTypex94x85x94Rx94(Kx01Kx00Kx01Kx02K
 ↪<ipython-input-171-12529e4b8824>x94x8cx06myfuncx94Kx01Cx02x00x01x94))tx94Rx94Jxffxffxffx
 ↪x94x87x94Rx94}x94(x8cx07globalsx94}x94x8cx08defaultsx94Nx8cx04dictx94}
 ↪x94x8cx06modulex94x8cx08__main__x94x8cx0eclosure_valuesx94Nx8cx08qualnamex94hx11utRu.
 ↪'
```

310. https://github.com/cloudpipe/cloudpickle

```
del myfunc

from cloudpickle import load
res = load(BytesIO(buffer.getvalue()))
res
```

```
{'x': 5, 'f': <function __main__.myfunc(x)>}
```

```
res['f'](res['x'])
```

```
6
```

### JSON

La sérialisation au format JSON ne fonctionne pas avec le module standard.

```
from json import dump
from io import StringIO
buffer = StringIO()
try:
    dump(data, buffer)
except Exception as e:
    print(e)
```

```
Object of type 'function' is not JSON serializable
```

La sérialisation avec *ujson* ne fonctionne pas non plus même si elle ne produit pas d'erreur.

```
from ujson import dump
from io import StringIO
buffer = StringIO()
res = dump(data, buffer)
buffer.getvalue()
```

```
'{"x":5,"f":{}}'
```

### Cas des itérateurs

Les itérateurs fonctionnent avec la sérialisation binaire mais ceci implique de stocker l'ensemble que l'itérateur parcourt.

```
ens = [1, 2]

data = {'x': 5, 'it': iter(ens)}

from pickle import dump
from io import BytesIO
buffer = BytesIO()
res = dump(data, buffer)
buffer.getvalue()
```

```
b'x80x03}qx00(Xx01x00x00xqx01Kx05Xx02x00x00x00itqx02cbuiltinsniternqx03]qx04(Kx01Kx02ex8
 ↪'
```

```
del ens
from pickle import load
res = load(BytesIO(buffer.getvalue()))
res
```

```
{'x': 5, 'it': <list_iterator at 0x1d8a18a9278>}
```

```
list(res["it"])
```

```
[1, 2]
```

```
list(res["it"])
```

```
[]
```

### Cas des générateurs

Ils ne peuvent être sérialisés car le langage n'a pas accès à l'ensemble des éléments que le générateur parcourt. Il n'y a aucun moyen de sérialiser un générateur mais on peut sérialiser la fonction qui crée le générateur.

```
def ensgen():
    yield 1
    yield 2

data = {'x': 5, 'it': ensgen()}

from pickle import dump
from io import BytesIO
buffer = BytesIO()
try:
    dump(data, buffer)
except Exception as e:
    print(e)
```

```
can't pickle generator objects
```

CHAPITRE 6

# Modules

## 6.1 Modules


— *Modules* (page 235)
  — *Exemple* (page 235)
  — *Importer un module* (page 238)
  — *Nom d'un module* (page 238)
— *Modules et fichiers* (page 239)
  — *Emplacement d'un module* (page 239)
  — *Attributs communs à tout module* (page 239)
— *Cas pratiques* (page 240)
  — *Ce qui ne marche pas : les import cycliques* (page 240)
  — *Ajouter un module en cours d'exécution* (page 241)
  — *Liste des modules importés* (page 241)
— *Plusieurs modules et fichiers* (page 242)
  — *Arborescence de modules, paquetage* (page 242)
  — *Import relatif* (page 243)
— *Modules internes* (page 243)
— *Modules externes* (page 246)


Il est souvent préférable de répartir le code d'un grand programme sur plusieurs fichiers. Parmi tous ces fichiers, un seul est considéré comme fichier principal, il contient son point d'entrée, les premières instructions exécutées. Les autres fichiers sont considérés comme des modules, en quelque sorte, des annexes qui contiennent tout ce dont le fichier principal a besoin.

### 6.1.1 Modules

#### Exemple

Cet exemple montre comment répartir un programme sur deux fichiers. Le premier est appelé textit{module} car il n'inclut pas le point d'entrée du programme.

**Définition D1 : point d'entrée du programme**

Le point d'entrée d'un programme est la première instruction exécutée par l'ordinateur lors de l'exécution de ce programme.

Cet exemple de module contient une fonction, une classe et une variable. Ces trois éléments peuvent être utilisés par n'importe quel fichier qui importe ce module. Le nom d'un module correspond au nom du fichier sans son extension.

<<<

```
"""
exemple de module, aide associée
"""

exemple_variable = 3


def exemple_fonction():
    """exemple de fonction"""
    return 0


class exemple_classe:
    """exemple de classe"""

    def __str__(self):
        return "exemple_classe"
```

>>>

```
```

Pour l'utiliser, il suffit de l'importer avec le mot clé import [311] :

```
import module_exemple

c = module_exemple.exemple_classe ()
print(c)
print(module_exemple.exemple_fonction())
help(module_exemple)
```

Et le résultat est :

```
    exemple_classe
    0
```

La dernière instruction affiche l'aide du module :

```
Help on module module_exemple:

NAME
    module_exemple - exemple de module, aide associée

CLASSES
    builtins.object
        exemple_classe
```

(suite sur la page suivante)

311. https://docs.python.org/3.6/library/functions.html#__import__

(suite de la page précédente)

```
    class exemple_classe(builtins.object)
     |  exemple de classe
     |
     |  Methods defined here:
     |
     |  __str__(self)
     |      Return str(self).
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

FUNCTIONS
    exemple_fonction()
        exemple de fonction

DATA
    exemple_variable = 3

FILE
    module_exemple.py
```

Pour importer un module, il suffit d'insérer l'instruction `import nom_module` avant d'utiliser une des choses qu'il définit. Ces importations sont souvent regroupées au début du programme, elles sont de cette façon mises en évidence même s'il est possible de les faire n'importe où. L'exemple ci-dessus à droite importe le module défini à gauche. Les modules commencent le plus souvent par une chaîne de caractères comme dans l'exemple précédent, celle-ci contient l'aide associée à ce module. Elle apparaît avec l'instruction `help(module_exemple)`.

Rien ne différencie les deux fichiers `module_exemple.py` et `exemple.py` excepté le fait que le second utilise des éléments définis par le premier. Dans un programme composé de plusieurs fichiers, un seul contient le point d'entrée et tous les autres sont des modules.

La syntaxe d'appel d'un élément d'un module est identique à celle d'une classe. On peut considérer un module comme une classe avec ses méthodes et ses attributs à la seule différence qu'il ne peut y avoir qu'une seule instance d'un même module. La répétition de l'instruction `import module_exemple` n'aura aucun effet : un module n'est importé que lors de la première instruction `import nom_module` rencontré lors de l'exécution du programme.

L'utilisation d'un module qu'on a soi-même conçu provoque l'apparition d'un répertoire `__pycache__` qui contient des fichiers d'extension `pyc`. Il correspond à la traduction du module en bytecode [312] plus rapidement exploitable par l'interpréteur *python*. Ce fichier est généré à chaque modification du module. Lorsqu'un module est importé, *python* vérifie la date des deux fichiers d'extension `py` et `pyc`. Si le premier est plus récent, le second est recréé. Cela permet un gain de temps lorsqu'il y a un grand nombre de modules. Il faut faire attention lorsque le fichier source d'extension `py` est supprimé, il est alors encore possible de l'importer tant que sa version d'extension `pyc` est présente.

Le module `module_exemple` contient une variable `exemple_variable` peut être modifiée au cours de l'exécution du programme. Il est possible de revenir à sa valeur initiale en forçant *python* à recharger le module grâce à la fonction import.reload [313] elle même implémentée dans le module importlib [314].

**Syntaxe S1 : importer un module (1)**

312. https://docs.python.org/3/glossary.html#term-bytecode
313. https://docs.python.org/3/library/importlib.html#importlib.reload
314. https://docs.python.org/3/library/importlib.html?highlight=reload#module-importlib

```
import importlib
import module_exemple
module_exemple.exemple_variable = 10
importlib.reload(module_exemple)
print(module_exemple.exemple_variable)       # affiche 3
```

### Importer un module

Il existe trois syntaxes différentes pour importer un module. La première est décrite au paragraphe précédent. Il en existe une autre qui permet d'affecter à un module un identificateur différent du nom du fichier dans lequel il est décrit. En ajoutant l'instruction `as` suivi d'un autre nom `alias`, le module sera désigné par la suite par l'identificateur `alias` comme le montre l'exemple suivant.

**Syntaxe S2 : importer un module (2)**

```
import module_exemple as alias

c = alias.exemple_classe()
print(c)
print(alias.exemple_fonction())
```

La syntaxe suivante n'est pas recommandée car elle masque le module d'où provient une fonction en plus de tout importer.

**Syntaxe S3 : importer un module (3)**

```
from module_exemple import *  # décommmandé
from module_exemple import exemple_classe, exemple_fonction

c = exemple_classe()
print(c)
print(exemple_fonction())
```

De plus, la partie `import *` permet d'importer toutes les classes, attributs ou fonctions d'un module mais il est possible d'écrire `from module_exemple import exemple_class` pour n'importer que cette classe. Dernier moyen, il est possible d'importer un module dont on ne connaît le nom que lors de l'exécution :

```
alias = __import__("module_exemple")

c = alias.exemple_classe()
print(c)
print(alias.exemple_fonction())
```

### Nom d'un module

Le nom d'un module est défini par le nom du fichier sous lequel il est enregistré. Dans l'exemple du paragraphe précédent, le module avait pour nom de fichier `module_exemple.py`, le nom de ce module est donc `module_exemple`.

Néanmoins, ce module peut également être exécuté comme un programme normal. Si tel est le cas, son nom devient `__main__`. C'est pourquoi, les quelques lignes qui suivent apparaissent souvent. Elles ne sont exécutées que si ce fichier a pour nom `__main__`. Un seul fichier peut porter ce nom : celui qui contient le point d'entrée.

```
if __name__ == "__main__":
    print("ce fichier est le programme principal")
```

Cette astuce est régulièrement utilisée pour tester les fonctions et classes définies dans un module. Etant donné que cette partie n'est exécutée que si ce fichier est le programme principal, ajouter du code après le test `if __name__ == "__main__":` n'a aucune incidence sur tout programme incluant ce fichier comme module.

### 6.1.2 Modules et fichiers

#### Emplacement d'un module

Lorsque le module est placé dans le même répertoire que le programme qui l'utilise, l'instruction `import nom_module_sans_extension` suffit. Cette instruction suffit également si ce module est placé dans le répertoire `site-packages` présent dans le répertoire d'installation de *python*. Si ce n'est pas le cas, il faut préciser à l'interpréteur *python* où il doit chercher ce module :

```
import sys
sys.path.append("répertoire où se trouve le module à importer")
import nom_module
```

La variable sys.path [315] contient les répertoires où *python* va chercher les modules. Le premier d'entre eux est le répertoire du programme. Il suffit d'ajouter à cette liste le répertoire désiré. Il est conseillé d'utiliser le plus souvent possible des chemins relatifs et non absolus. Il est aussi déconseillé d'utiliser cette technique. Il vaut mieux utiliser des *import* relatifs.

> *Aparté.* Depuis un répertoire, les chemins relatifs permettent de faire référence à d'autres répertoires sans avoir à prendre en compte leur emplacement sur le disque dur contrairement aux chemins absolus comme `C:\Python36_x64\python.exe` qui parte de la racine. De cette façon, on peut recopier le programme et ses modules à un autre endroit du disque dur sans altérer leur fonctionnement.

#### Attributs communs à tout module

Une fois importés, tous les modules possèdent cinq attributs qui contiennent des informations comme leur nom, le chemin du fichier correspondant, l'aide associée.

| | |
|---|---|
| `__all__` | Contient toutes les variables, fonctions, classes du module |
| `__builtins__` | Ce dictionnaire contient toutes les fonctions et classes inhérentes au langage *python* utilisées par le module. |
| `__doc__` | Contient l'aide associée au module. |
| `__file__` | Contient le nom du fichier qui définit le module. |
| `__name__` | Cette variable contient a priori le nom du module sauf si le module est le point d'entrée du programme auquel cas cette variable contient `__main__`. |

Ces attributs sont accessibles si le nom du module est utilisé comme préfixe. Sans préfixe, ce sont ceux du module lui-même.

<<<

```
import os
print(os.__name__, os.__doc__)
if __name__ == "__main__":
    print("ce fichier est le point d'entrée")
else:
    print("ce fichier est importé")
```

315. https://docs.python.org/3/library/sys.html#sys.path

```
>>>
```

```
    os OS routines for NT or Posix depending on what system we're on.

    This exports:
      - all functions from posix or nt, e.g. unlink, stat, etc.
      - os.path is either posixpath or ntpath
      - os.name is either 'posix' or 'nt'
      - os.curdir is a string representing the current directory (always '.')
      - os.pardir is a string representing the parent directory (always '..')
      - os.sep is the (or a most common) pathname separator ('/' or '\\')
      - os.extsep is the extension separator (always '.')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of being
    portable between different platforms.  Of course, they must then
    only use functions that are defined by all platforms (e.g., unlink
    and opendir), and leave all pathname manipulation to os.path
    (e.g., split and join).

    ce fichier est importé
```

### 6.1.3 Cas pratiques

#### Ce qui ne marche pas : les import cycliques

Deux modules ne peuvent s'importer l'un l'autre.

```
# module1.py
from .module2 import B
def A():
    return ...
```

```
# module2.py
from .module1 import A
def B():
    return ...
```

Le module `module1` import le module `module2` qui cherche à son tour à importer le module `module1` et c'est sans fin. Il est possible d'éviter de cela en retardant l'un des deux imports.

```
# module1.py
from .module2 import B
def A():
    return ...
```

```
# module2.py
def B():
    from .module1 import A
    return ...
```

Le second import n'a lieu qu'à la première exécution de la fonction `B`. Ca marche mais cela veut dire aussi que la fonction `B` contiendra une instruction de plus.

### Ajouter un module en cours d'exécution

De la même façon que *python* est capable d'inclure de nouvelles portions de code en cours d'exécution (fonction exec [316]), il est également capable d'inclure en cours d'exécution des modules dont on ne connaît pas le nom au début de l'exécution. Cela s'effectue grâce à la fonction `__import__` déjà présentée ci-dessus. Néanmoins, cette fonction ne peut pas importer un module si celui-ci est désigné par un nom de fichier incluant son répertoire. Il faut d'abord déterminer le répertoire où est le module grâce à la fonction split [317] du module os.path [318]. Le programme suivant illustre cette possibilité en proposant une fonction qui importe un module connaissant le nom du fichier qui le contient. Il ne faut pas oublier d'enlever l'extension et ne pas garder aucun répertoire.

```
alias = __import__("module_exemple.py".replace(".py", ""))

c = alias.exemple_classe()
print(c)
print(alias.exemple_fonction())
```

### Liste des modules importés

Le dictionnaire modules [319] du module sys [320] contient l'ensemble des modules importés. Le programme suivant affiche cette liste.

<<<

```
import sys
i = 0
for m in sys.modules:
    print(m, " " * (14 - len(str(m))), sys.modules[m])
    if i > 5:
        break
    i += 1
```

>>>

```
    sys             <module 'sys' (built-in)>
    builtins        <module 'builtins' (built-in)>
    _frozen_importlib  <module 'importlib._bootstrap' (frozen)>
    _imp            <module '_imp' (built-in)>
    _thread         <module '_thread' (built-in)>
    _warnings       <module '_warnings' (built-in)>
    _weakref        <module '_weakref' (built-in)>
```

Lorsque le programme stipule l'import d'un module, *python* vérifie s'il n'est pas déjà présent dans cette liste. Dans le cas contraire, il l'importe. Chaque module n'est importé qu'une seule fois. La première instruction `import module_exemple` rencontrée introduit une nouvelle entrée dans le dictionnaire modules [321] :

```
module_exemple  <module 'module_exemple' from 'D:\python_cours\module_exemple.py'>
```

Le dictionnaire modules [322] peut être utilisé pour vérifier la présence d'un module ou lui assigner un autre identificateur. Un module est un objet qui n'autorise qu'une seule instance.

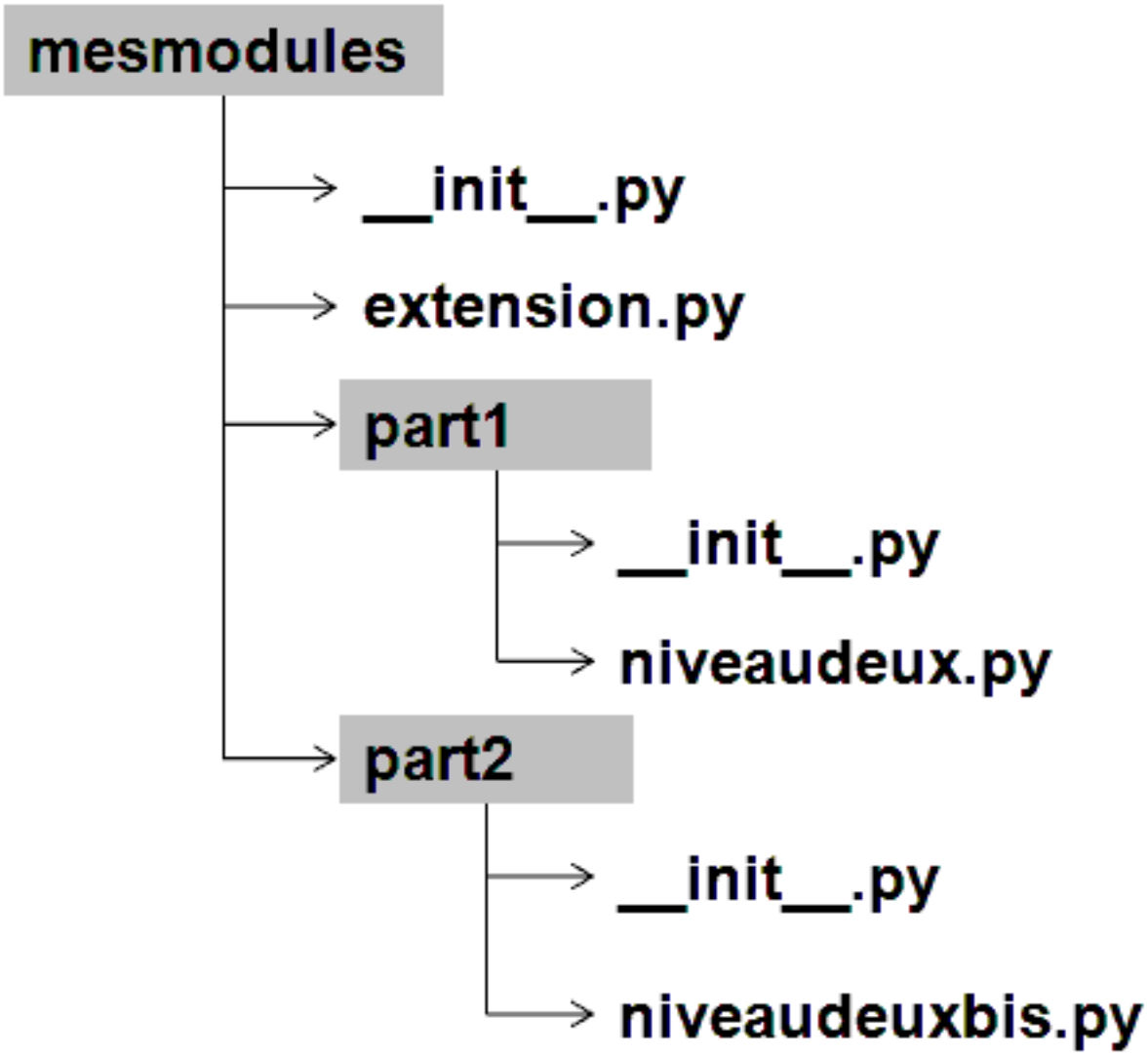
316. https://docs.python.org/3/library/functions.html#exec
317. https://docs.python.org/3.6/library/stdtypes.html#str.split
318. https://docs.python.org/3.6/library/os.path.html?highlight=os.path#module-os.path
319. https://docs.python.org/3/library/sys.html?highlight=modules#sys.modules
320. https://docs.python.org/3/library/sys.html
321. https://docs.python.org/3/library/sys.html?highlight=modules#sys.modules
322. https://docs.python.org/3/library/sys.html?highlight=modules#sys.modules

```
if "module_exemple" in sys.modules:
    m = sys.modules["module_exemple"]
    m.exemple_fonction()
```

### 6.1.4 Plusieurs modules et fichiers

#### Arborescence de modules, paquetage

Lorsque le nombre de modules devient conséquent, il est parfois souhaitable de répartir tous ces fichiers dans plusieurs répertoires. Il faudrait alors inclure tous ces répertoires dans la liste sys.path [323] ce qui paraît fastidieux. *python* propose la définition de paquetage, ce dernier englobe tous les fichiers *python* d'un répertoire à condition que celui-ci contienne un fichier `__init__.py` qui peut être vide. La figure suivante présente une telle organisation et l'exemple suivant explicite comment importer chacun de ces fichiers sans avoir à modifier les chemins d'importation.




Les répertoires sont grisées tandis que les fichiers apparaissent avec leur extension.

323. https://docs.python.org/3/library/sys.html#sys.path

```
import mesmodules.extension
import mesmodules.part1.niveaudeux
import mesmodules.part2.niveaudeuxbis
```

Lors de la première instruction `import mesmodules.extension`, le langage *python* ne s'intéresse pas qu'au seul fichier `extension.py`, il exécute également le contenu du fichier `__init__.py`. Si cela est nécessaire, c'est ici qu'il faut insérer les instructions à exécuter avant l'import de n'importe quel module du paquetage.

#### Import relatif

Les modules permettent d'écrire des programmes dans une succession de petits fichiers et c'est plus lisible comme cela. Les imports relatifs [324] Voici un ensemble de fichier avec une fonction implémentée dans chacun.

```
package/
    __init__.py        # fonction A
    subpackage1/
        __init__.py    # fonction B
        moduleX.py     # fonction C
    subpackage2/
        __init__.py    # fonction D
        moduleY.py     # fonction E
    moduleA.py         # fonction F
```

La fonction `A` peut utiliser la fonction `B` ou `C` en les important de la façon suivante :

```
from .subpackage1 import B
from .subpackage1.moduleX import C
```

La fonction `E` peut utiliser la fonction `F` ou `A` ou `C` en les important de la façon suivante :

```
from ..moduleA import F
from .. import A
from ..subpackage1.moduleX import C
```

Ce qu'il faut retenir :

— Le symbole `.` permet d'importer un module dans le même répertoire.
— Le symbole `..` permet d'importer un module dans le répertoire parent.
— Le fichier `__init__.py` est essentiel pour signifier qu'un répertoire contient des fichiers *python*.
— Il n'existe qu'une syntaxe : `from .<module> import`.

### 6.1.5 Modules internes

*python* dispose de nombreux modules préinstallés. La page Python Module Index [325] recense tous les modules disponibles par défaut avec *python*. Cette liste est trop longue pour figurer dans ce document, elle est aussi susceptible de s'allonger au fur et à mesure du développement du langage *python*. La table qui suit regroupe les modules les plus utilisés.

| | |
|---|---|
| asyncio [326] | Thread, socket, protocol. |
| calendar [327] | Gérer les calendriers, les dates. |
| cgi [328] | Utilisé dans les scripts CGI (programmation Internet). |
| cmath [329] | Fonctions mathématiques complexes. |

Suite sur la page suivante

324. http://stackoverflow.com/questions/14132789/relative-imports-for-the-billionth-time
325. https://docs.python.org/3/py-modindex.html

Tableau 1 – suite de la page précédente

| | |
|---|---|
| copy [330] | Copies d'instances de classes. |
| csv [331] | Gestion des fichiers au format CSV |
| datetime [332] | Calculs sur les dates et heures |
| gc [333] | Gestion du garbage collector |
| getopt [334] | Lire les options des paramètres passés en arguments d'un programme *python* |
| glob [335] | Chercher des fichiers |
| hashlib [336] | Fonctions de cryptage |
| htmllib [337] | Lire le format HTML |
| importlib [338] | Pour importer des modules. |
| math [339] | Fonctions mathématiques standard telles que cos [340], exp [341], log [342]... |
| os [343] | Fonctions systèmes dont certaines fonctions permettant de gérer les fichiers |
| os.path [344] | Manipulations de noms de fichiers |
| pathlib [345] | Manipulation de chemins. |
| pickle [346] | Sérialisation d'objets, la sérialisation consiste à convertir des données structurées de façon complexe en une structure linéaire facilement enregistrable dans un fichier |
| profile [347] | Etudier le temps passé dans les fonctions d'un programme |
| random [348] | Génération de nombres aléatoires |
| re [349] | Expressions régulières |
| shutil [350] | Copie de fichiers |
| sqlite3 [351] | Accès aux fonctionnalités du gestionnaire de base de données SQLite3 |
| string [352] | Manipulations des chaînes de caractères |
| sys [353] | Fonctions systèmes, fonctions liées au langage *python* |
| threading [354] | Utilisation de threads |
| time [355] | Accès à l'heure, l'heure système, l'heure d'une fichier |
| tkinter [356] | Interface graphique |
| unittest [357] | Tests unitaires (ou comment améliorer la fiabilité d'un programme) |
| urllib [358] | Pour lire le contenu de page HTML sans utiliser un navigateur |
| xml.dom [359] | Lecture du format XML. |
| xml.sax [360] | Lecture du format XML. |
| zipfile [361] | Lecture de fichiers ZIP. |

Certains de ces modules sont présentés dans les chapitres qui suivent. Le programme suivant par exemple utilise les modules random [362] math [363] pour estimer le nombre $\pi$. Pour cela, on tire aléatoirement deux nombres $x, y$ dans l'intervalle $[0, 1]$, si $\sqrt{x^2 + y^2} \leqslant 1$, on compte 1 sinon 0. Au final, $\hat{\pi} = \mathbb{E}\mathbb{1}_{\left\{\sqrt{x^2+y^2}\leqslant 1\right\}}$.

<<<

```
import random
import math

somme = 0
nb = 1000000
for i in range(0, nb):
    x = random.random()          # nombre aléatoire entre [0,1]
    y = random.random()
    r = math.sqrt(x*x + y*y)     # racine carrée
    if r <= 1:
        somme += 1

print("estimation ", 4 * float(somme) / nb)
print("PI = ", math.pi)
```

>>>

```
    estimation  3.142048
    PI =  3.141592653589793
```

326. https://docs.python.org/3/library/asyncio.html
327. https://docs.python.org/3/library/calendar.html
328. https://docs.python.org/3/library/cgi.html
329. https://docs.python.org/3/library/cmath.html
330. https://docs.python.org/3/library/copy.html
331. https://docs.python.org/3/library/csv.html
332. https://docs.python.org/3/library/datetime.html
333. https://docs.python.org/3/library/gc.html
334. https://docs.python.org/3/library/getopt.html
335. https://docs.python.org/3/library/glob.html
336. https://docs.python.org/3/library/hashlib.html
337. https://docs.python.org/3/library/htmllib.html
338. https://docs.python.org/3/library/importlib.html
339. https://docs.python.org/3/library/math.html
340. https://docs.python.org/3/library/math.html?highlight=cos#math.cos
341. https://docs.python.org/3/library/math.html?highlight=exp#math.exp
342. https://docs.python.org/3/library/math.html?highlight=exp#math.log
343. https://docs.python.org/3/library/os.html
344. https://docs.python.org/3/library/os.path.html
345. https://docs.python.org/3/library/pathlib.html
346. https://docs.python.org/3/library/pickle.html
347. https://docs.python.org/3/library/profile.html
348. https://docs.python.org/3/library/random.html
349. https://docs.python.org/3/library/re.html
350. https://docs.python.org/3/library/shutil.html
351. https://docs.python.org/3/library/sqlite3.html
352. https://docs.python.org/3/library/string.html
353. https://docs.python.org/3/library/sys.html
354. https://docs.python.org/3/library/threading.html
355. https://docs.python.org/3/library/time.html
356. https://docs.python.org/3/library/tkinter.html
357. https://docs.python.org/3/library/unittest.html
358. https://docs.python.org/3/library/urllib.html
359. https://docs.python.org/3/library/xml.dom.html
360. https://docs.python.org/3/library/xml.sax.html
361. https://docs.python.org/3/library/zipfile.html
362. https://docs.python.org/3/library/random.html
363. https://docs.python.org/3/library/math.html

Le programme suivant calcule l'intégrale de Monte Carlo [364] de la fonction $f(x) = \sqrt{x}$ qui consiste à tirer des nombres aléatoires dans l'intervalle $[a, b]$ puis à faire la moyenne des $\sqrt{x}$ obtenu.

<<<

```
import random  # import du module random : simulation du hasard
import math    # import du module math : fonctions mathématiques


def integrale_monte_carlo(a, b, f, n):
    somme = 0.0
    for i in range(0, n):
        x = random.random() * (b-a) + a
        y = f(x)
        somme += f(x)
    return somme / n


def racine(x):
    return math.sqrt(x)


print(integrale_monte_carlo(0, 1, racine, 100000))
```

>>>

```
    0.6673192323999039
```

### 6.1.6 Modules externes

Les modules externes ne sont pas fournis avec *python*, ils nécessitent une installation supplémentaire. Il serait impossible de couvrir tous les thèmes abordés par ces extensions. La simplicité d'utilisation du langage *python* et son interfaçage facile avec le langage C [365] contribue à sa popularité. Il permet de relier entre eux des projets conçus dans des environnements différents, dans des langages différents. Depuis les versions 2.3, 2.4 du langage *python*, la plupart des modules externes sont faciles à installer, faciles à utiliser d'après les exemples que fournissent de plus en plus les sites Internet qui les hébergent. De plus, il s'écoule peu de temps entre la mise à disposition d'une nouvelle version du langage *python* et la mise à jour du module pour cette version

De nombreux modules ont été conçus pour un besoin spécifique et ne sont plus maintenus. Cela convient lors de l'écriture d'un programme qui remplit un besoin ponctuel. Pour une application plus ambitieuse, il est nécessaire de vérifier quelques informations comme la date de création, celle de la dernière mise à jour, la présence d'une documentation, une prévision pour la sortie de la future version, si c'est une personne lambda qui l'a conçu ou si c'est une organisation comme celle qui fournit le module tensorflow [366]. Tout va souvent très vite. Le nombre de modifications est un critère assez simple pour s'assurer qu'un module est maintenu : commit [367]. La plupart des modules sont sur Github aujourd'hui. S'il ne l'est pas, passez votre chemin.

L'installation de modules externes n'est pas toujours simple, certains comme scipy [368] incluent des fichiers C++ qui doivent être compilés. Dans tous les cas, le code source des fichiers inclut un fichier `setup.py`. Le langage *python* fournit une procédure d'installation standard :

```
python setup.py install
```

364. https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Monte-Carlo
365. https://en.wikipedia.org/wiki/C_(programming_language)
366. https://pypi.python.org/pypi/tensorflow/
367. https://github.com/etetoolkit/ete/commits/master
368. https://pypi.python.org/pypi/scipy

Ce procédé marche la plupart du temps. Il échoue lorsque le module inclut des fichiers écrits dans un autre langage. L'installation dépend alors du système d'exploitation. Il est plus simple dans le cas d'installation des modules précmopilés. Sous Windows, beaucoup sont accessibles sur cette page : Unofficial Windows Binaries for Python Extension Packages [369].

## 6.2 FAQ

1. *Comment éviter sys.path.append... quand on développe un module ?* (page **??**)
2. *Obtenir des informations sur les packages installés* (page **??**)
3. *Récupérer la liste des modules installés* (page **??**)

---

**Comment éviter sys.path.append... quand on développe un module ?**

Lorsqu'on développe un module, on ne veut pas l'installer. On ne veut pas qu'il soit présent dans le répertoire `site-packages` de la distribution de Python car cela introduit deux versions : celle qu'on développe et celle qu'on a installer. Avant, je faisais cela pour créer un petit programme utilisant mon propre module (et on en trouve quelque trace dans mon code) :

```
import sys
sys.path.append("c:/moncode/monmodule/src")
import monmodule
```

Quand je récupère un programme utilisant ce module, il me faudrait ajouter ces petites lignes à chaque fois et c'est barbant. Pour éviter cela, il est possible de dire à l'interpréteur Python d'aller chercher ailleurs pour trouver des modules en ajoutant le chemin à la variable d'environnement [370] PYTHONPATH [371]. Sous Windows :

```
set PYTHON_PATH=%PYTHON_PATH%;c:\moncode\monmodule\src
```

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.python_path, line 1)

---

**Obtenir des informations sur les packages installés**

Le module pip [372] retourne des informations sur n'importe quel module installé, sa version, sa license

```
pip show pandas
```

On peut également l'obtenir depuis l'interpréteur python

```
import pip
pip.main(["show", "pandas"])
```

Exemple

```
Name: pandas
Version: 0.16.0
Summary: Powerful data structures for data analysis, time series,and statistics
Home-page: http://pandas.pydata.org
Author: The PyData Development Team
Author-email: pydata@googlegroups.com
```

(suite sur la page suivante)

369. http://www.lfd.uci.edu/~gohlke/pythonlibs/
370. http://fr.wikipedia.org/wiki/Variable_d'environnement
371. https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH
372. https://pip.pypa.io/en/stable/

(suite de la page précédente)

```
License: BSD
Location: c:\python35_x64\lib\site-packages
Requires: python-dateutil, pytz, numpy
```

On utilise également `pip freeze` pour répliquer l'environnement dans lequel on a développé un programme. pip freeze [373] produit la liste des modules avec la version utilisée

```
docutils==0.11
Jinja2==2.7.2
MarkupSafe==0.19
Pygments==1.6
Sphinx==1.2.2
```

Ce qu'on utilise pour répliquer l'environnement de la manière suivante

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

Cette façon de faire fonctionne très bien sous Linux mais n'est pas encore opérationnelle sous Windows à moins d'installer le compilateur C++ utilisée pour compiler Python.

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.list_of_installed_packages, line 3)

---

**Récupérer la liste des modules installés**

Le module pip [374] permet d'installer de nouveaux modules mais aussi d'obtenir la liste des packages installés

```
pip list
```

On peut également l'obtenir depuis l'interpréteur python

```
import pip
pip.main(["list"])
```

---

(*entrée originale* : faq_python.py :docstring of teachpyx.faq.faq_python.information_about_package, line 3)

373. https://pip.pypa.io/en/latest/reference/pip_freeze.html
374. https://pip.pypa.io/en/stable/

CHAPITRE 7

# Expression régulières

## 7.1 Expressions régulières


— *A quoi ça sert ?* (page 249)
— *Syntaxe* (page 250)
  — *Les ensembles de caractères* (page 251)
  — *Les multiplicateurs* (page 251)
  — *Groupes* (page 252)
  — *Assembler les caractères* (page 252)
  — *Fonctions* (page 252)
  — *Exemple plus complet* (page 254)
  — *Groupes nommés* (page 256)


### 7.1.1 A quoi ça sert ?

Chercher un mot dans un texte est une tâche facile, c'est l'objectif de la méthode find [375] attachée aux chaînes de caractères, elle suffit encore lorsqu'on cherche un mot au pluriel ou au singulier mais il faut l'appeler au moins deux fois pour chercher ces deux formes. Pour des expressions plus compliquées, il est conseillé d'utiliser les expressions régulières [376]. C'est une fonctionnalité qu'on retrouve dans beaucoup de langages. C'est une forme de grammaire qui permet de rechercher des expressions.

Lorsqu'on remplit un formulaire, on voit souvent le format `"MM/JJ/AAAA"` qui précise sous quelle forme on s'attend à ce qu'une date soit écrite. Les expressions régulières permettent de définir également ce format et de chercher dans un texte toutes les chaînes de caractères qui sont conformes à ce format.

La liste qui suit contient des dates de naissance. On cherche à obtenir toutes les dates de cet exemple sachant que les jours ou les mois contiennent un ou deux chiffres, les années deux ou quatre.

375. https://docs.python.org/3/library/stdtypes.html?highlight=find#str.find
376. https://fr.wikipedia.org/wiki/Expression_rationnelle

```
s = """date 0 : 14/9/2000
date 1 : 20/04/1971     date 2 : 14/09/1913     date 3 : 2/3/1978
date 4 : 1/7/1986     date 5 : 7/3/47     date 6 : 15/10/1914
date 7 : 08/03/1941     date 8 : 8/1/1980     date 9 : 30/6/1976"""
```

Le premier chiffre du jour est soit 0, 1, 2, ou 3 ; ceci se traduit par `[0-3]`. Le second chiffre est compris entre 0 et 9, soit `[0-9]`. Le format des jours est traduit par `[0-3][0-9]`. Mais le premier jour est facultatif, ce qu'on précise avec le symbole `?` : `[0-3]?[0-9]`. Les mois suivent le même principe : `[0-1]?[0-9]`. Pour les années, ce sont les deux premiers chiffres qui sont facultatifs, le symbole `?` s'appliquent sur les deux premiers chiffres, ce qu'on précise avec des parenthèses : `([0-2][0-9])?[0-9][0-9]`. Le format final d'une date devient :

```
[0-3]?[0-9]/[0-1]?[0-9]/([0-2][0-9])?[0-9][0-9]
```

Le module re [377] gère les expressions régulières, celui-ci traite différemment les parties de l'expression régulière qui sont entre parenthèses de celles qui ne le sont pas : c'est un moyen de dire au module re [378] que nous nous intéressons à telle partie de l'expression qui est signalée entre parenthèses. Comme la partie qui nous intéresse - une date - concerne l'intégralité de l'expression régulière, il faut insérer celle-ci entre parenthèses.

La première étape consiste à construire l'expression régulière, la seconde à rechercher toutes les fois qu'un morceau de la chaîne `s` définie plus haut correspond à l'expression régulière.

<<<

```
s = """date 0 : 14/9/2000
    date 1 : 20/04/1971     date 2 : 14/09/1913     date 3 : 2/3/1978
    date 4 : 1/7/1986     date 5 : 7/3/47     date 6 : 15/10/1914
    date 7 : 08/03/1941     date 8 : 8/1/1980     date 9 : 30/6/1976"""

import re
# première étape : construction
expression = re.compile("([0-3]?[0-9]/[0-1]?[0-9]/([0-2][0-9])?[0-9][0-9])")
# seconde étape : recherche
res = expression.findall(s)
print(res)
```

>>>

```
    [('14/9/2000', '20'), ('20/04/1971', '19'), ('14/09/1913', '19'), ('2/3/1978', '19
↪'), ('1/7/1986', '19'), ('7/3/47', ''), ('15/10/1914', '19'), ('08/03/1941', '19'),␣
↪('8/1/1980', '19'), ('30/6/1976', '19')]
```

Le résultat une liste de couples dont chaque élément correspond aux parties comprises entre parenthèses qu'on appelle des *groupes*. Lorsque les expressions régulières sont utilisées, on doit d'abord se demander comment définir ce qu'on cherche puis quelles fonctions utiliser pour obtenir les résultats de cette recherche. Les deux paragraphes qui suivent y répondent.

### 7.1.2 Syntaxe

La syntaxe des expressions régulières est décrite sur le site officiel de *python*. La page Regular Expression Syntax [379] décrit comment se servir des expressions régulières, les deux pages sont en anglais. Comme toute grammaire, celle des expressions régulières est susceptible d'évoluer au fur et à mesure des versions du langage *python*.

377. https://docs.python.org/3/library/re.html?highlight=re#module-re
378. https://docs.python.org/3/library/re.html?highlight=re#module-re
379. https://docs.python.org/3/library/re.html?highlight=re#regular-expression-syntax

### Les ensembles de caractères

Lors d'une recherche, on s'intéresse aux caractères et souvent aux classes de caractères : on cherche un chiffre, une lettre, un caractère dans un ensemble précis ou un caractère qui n'appartient pas à un ensemble précis. Certains ensembles sont prédéfinis, d'autres doivent être définis à l'aide de crochets.

Pour définir un ensemble de caractères, il faut écrire cet ensemble entre crochets : `[0123456789]` désigne un chiffre. Comme c'est une séquence de caractères consécutifs, on peut résumer cette écriture en `[0-9]`. Pour inclure les symboles -, +, il suffit d'écrire : `[-0-9+]`. Il faut penser à mettre le symbole - au début pour éviter qu'il ne désigne une séquence.

Le caractère ^ inséré au début du groupe signifie que le caractère cherché ne doit pas être un de ceux qui suivent. Le tableau suivant décrit les ensembles prédéfinis et leur équivalent en terme d'ensemble de caractères :

| | |
|---|---|
| `.` | désigne tout caractère non spécial quel qu'il soit |
| `\d` | désigne tout chiffre, est équivalent à `[0-9]` |
| `\D` | désigne tout caractère différent d'un chiffre, est équivalent à `[^0-9]` |
| `\s` | désigne tout espace ou caractère approché, est équivalent à `[\; \t\n\r\f\v]`. Ces caractères sont spéciaux, les plus utilisés sont `\t` qui est une tabulation, `\n` qui est une fin de ligne et qui `\r` qui est un retour à la ligne. |
| `\S` | désigne tout caractère différent d'un espace, est équivalent à `[^ \t\n\r\f\v]` |
| `\w` | désigne tout lettre ou chiffre, est équivalent à `[a-zA-Z0-9\_]` |
| `\W` | désigne tout caractère différent d'une lettre ou d'un chiffre, est équivalent à `[^a-zA-Z0-9\_]` |
| `^` | désigne le début d'un mot sauf s'il est placé entre crochets |
| `$` | désigne la fin d'un mot sauf s'il est placé entre crochets |

A l'instar des chaînes de caractères, comme le caractère `\` est un caractère spécial, il faut le doubler : `[\\]`. Avec ce système, le mot *ńătaxinomieă"* qui accepte deux orthographes s'écrira : `tax[io]nomie`.

Le caractère `\` est déjà un caractère spécial pour les chaînes de caractères en *python*, il faut donc le quadrupler pour l'insérer dans un expression régulière. L'expression suivante filtre toutes les images dont l'extension est *png* et qui sont enregistrées dans un répertoire `image`.

<<<

```
import re
s = "something\\support\\vba\\image/vbatd1_4.png"
print(re.compile("[\\\\/]image[\\\\/].*[.]png").search(s))   # résultat positif
print(re.compile("[\\\\/]image[\\\\/].*[.]png").search(s))   # même résultat
```

>>>

```
    <re.Match object; span=(21, 40), match='\\image/vbatd1_4.png'>
    <re.Match object; span=(21, 40), match='\\image/vbatd1_4.png'>
```

### Les multiplicateurs

Les multiplicateurs permettent de définir des expressions régulières comme : un mot entre six et huit lettres qu'on écrira `[\w]{6,8}`. Le tableau suivant donne la liste des multiplicateurs principaux :

| | |
|---|---|
| `*` | présence de l'ensemble de caractères qui précède entre 0 fois et l'infini |
| `+` | présence de l'ensemble de caractères qui précède entre 1 fois et l'infini |
| `?` | présence de l'ensemble de caractères qui précède entre 0 et 1 fois |
| `{m,n}` | présence de l'ensemble de caractères qui précède entre `m` et `n` fois, si `m=n`, cette expression peut être résumée par `{n}`. |
| `(?!(...))` | absence du groupe désigné par les points de suspensions. |

L'algorithme des expressions régulières essaye toujours de faire correspondre le plus grand morceau à l'expression régulière. Par exemple, dans la chaîne `<h1>mot</h1>`, `<.*>` correspond avec `<h1>`, `</h1>` ou encore `<h1>mot</h1>`. Par conséquent, l'expression régulière correspond à trois morceaux. Par défaut, il prendra le plus grand. Pour choisir les plus petits, il faudra écrire les multiplicateurs comme ceci : `*?`, `+?`, `??`.

<<<

```
import re
s = "<h1>mot</h1>"
print(re.compile("(<.*>)").match(s).groups())   # ('<h1>mot</h1>',)
print(re.compile("(<.*?>)").match(s).groups())   # ('<h1>',)
```

>>>

```
    ('<h1>mot</h1>',)
    ('<h1>',)
```

### Groupes

Lorsqu'un multiplicateur s'applique sur plus d'un caractère, il faut définir un groupe à l'aide de parenthèses. Par exemple, le mot `yoyo` s'écrira : `(yo){2}`. Les parenthèses jouent un rôle similaire à celui qu'elles jouent dans une expression numérique. Tout ce qui est compris entre deux parenthèses est considéré comme un groupe.

### Assembler les caractères

On peut assembler les groupes de caractères les uns à la suite des autres. Dans ce cas, il suffit de les juxtaposer comme pour trouver les mots commençant par `s` : `s[a-z]*`. On peut aussi chercher une chaîne ou une autre grâce au symbole `|`. Chercher dans un texte l'expression *Xavier Dupont* ou *M. Dupont* s'écrira : `(Xavier)|(M[.]) Dupont`.

### Fonctions

La fonction compile [380] du module re [381] permet de construire un objet ńăexpression régulièreă". A partir de cet objet, on peut vérifier la correspondance entre une expression régulière et une chaîne de caractères (méthode match [382]). On peut chercher une expression régulière (méthode search [383]). On peut aussi remplacer une expression régulière par une chaîne de caractères (méthode sub [384]).

380. https://docs.python.org/3/library/re.html?highlight=re#re.compile
381. https://docs.python.org/3/library/re.html?highlight=re#module-re
382. https://docs.python.org/3/library/re.html?highlight=re#re.match
383. https://docs.python.org/3/library/re.html?highlight=re#re.search
384. https://docs.python.org/3/library/re.html?highlight=re#re.sub

| | |
|---|---|
| `match(s[, pos[, end]])` | Vérifie la correspondance entre l'expression régulière et la chaîne `s`. Il est possible de n'effectuer cette vérification qu'entre les caractères dont les positions sont `pos` et `end`. La fonction retourne `None` s'il n'y a pas de correspondance et sinon un objet de type `Match`. |
| `search(s[, pos[, end]])` | Fonction semblable à `match`, au lieu de vérifier la correspondance entre toute la chaîne et l'expression régulière, cherche la première chaîne de caractères extraite correspondant à l'expression régulière. |
| `split(s, maxsplit = 0])` | Recherche toutes les chaînes de caractères extraites qui vérifient l'expression régulière puis découpe cette chaîne en fonction des expressions trouvées. La méthode `split` d'une chaîne de caractère permet de découper selon un séparateur. La méthode `split` d'une expression régulière permet de découper selon plusieurs séparateurs. C'est pratique pour découper une chaîne de caractères en colonnes séparées par `;` ou une tabulation. `re.compile("[\t;]").split("a;b\tc;g")` donne `["a", "b", "c", "g"]`. |
| `findall( s[, pos[, end]])` | Identique à `split` mais ne retourne pas les morceaux entre les chaînes extraites qui vérifient l'expression régulière. |
| `sub(repl, s, count = 0])` | Remplace dans la chaîne `repl` les éléments `\1`, `\2`, ... par les parties de `s` qui valident l'expression régulière. |
| `flags` | Mémorise les options de construction de l'expression régulière. C'est un attribut. |
| `pattern` & | Chaîne de caractères associée à l'expression régulière. C'est un attribut. |

Ces méthodes et attributs qui s'appliquent à un objet de type ńăexpression régulièreă" retourné par la fonction compile [385]. Les méthodes search [386] et match [387] retournent toutes des objets Match [388] :

<<<

```
s = """date 0 : 14/9/2000
date 1 : 20/04/1971     date 2 : 14/09/1913     date 3 : 2/3/1978
date 4 : 1/7/1986     date 5 : 7/3/47     date 6 : 15/10/1914
date 7 : 08/03/1941     date 8 : 8/1/1980     date 9 : 30/6/1976"""

import re
expression = re.compile(
    "([0-3]?[0-9]/[0-1]?[0-9]/([0-2][0-9])?[0-9][0-9])[^\d]")
print(expression.search(s).group(1, 2))  # affiche ('14/9/2000', '20')
c = expression.search(s).span(1)         # affiche (9, 18)
print(s[c[0]:c[1]])                      # affiche 14/9/2000
```

>>>

```
    ('14/9/2000', '20')
    14/9/2000
```

| | |
|---|---|
| `group([g1, ...])` | Retourne la chaîne de caractères validée par les groupes `g1`... |
| `groups([default])` | Retourne la liste des chaînes de caractères qui ont été validées par chacun des groupes. |
| `span([gr])` | Retourne les positions dans la chaîne originale des chaînes extraites validées le groupe `gr`. |

385. https://docs.python.org/3/library/re.html?highlight=re#re.compile
386. https://docs.python.org/3/library/re.html?highlight=re#re.search
387. https://docs.python.org/3/library/re.html?highlight=re#re.match
388. https://docs.python.org/3/library/re.html?highlight=re#re.Match

Ces méthodes qui s'appliquent à un objet de type Match [389] qui est le résultat des méthodes search [390] et match [391]. Les groupes sont des sous-parties de l'expression régulière, chacune d'entre elles incluses entre parenthèses. Le énième correspond au groupe qui suit la énième parenthèse ouvrante. Le premier groupe a pour indice 1.

### Exemple plus complet

L'exemple suivant présente trois cas d'utilisation des expressions régulières. On s'intéresse aux titres de chansons MP3 [392] stockées dans un répertoire. Le module mutagen [393] permet de récupérer certaines informations concernant un fichier *MP3* dont le titre, l'auteur et la durée.

Le premier problème consiste à retrouver les chansons sans titre ou dont le titre contient seulement son numéro : *track03*, *track - 03*, *audiotrack 03*, *track 03*, *piste 03*, *piste - 03*, *audiopiste 03*, ... Ce titre indésirable doit valider l'expression régulière suivante : `^(((audio)?track( )?( - )?[0-9]{1,2})|(piste [0-9]{1,2}))$`.

Le second problème consiste à retrouver toutes les chansons dont le titre contient le mot *heart* mais ni *heartland* ni *heartache*. Ce titre doit valider l'expression régulière : `((heart)(?!((ache)|(land))))`.

Le troisième problème consiste à compter le nombre de mots d'un titre. Les mots sont séparés par l'ensemble de caractères `[- ,;!'.?&:]`. On utilise la méthode `split` pour découper en mots. Le résultat est illustré par le programme suivant.

<<<

```
import os
import re
import warnings
import mutagen.mp3
import mutagen.easyid3


def infoMP3(file, tags):
    """retourne des informations sur un fichier MP3 sous forme de
    dictionnaire (durée, titre, artiste, ...)"""
    try:
        a = mutagen.mp3.MP3(file)
        b = mutagen.easyid3.EasyID3(file)
    except Exception as e:
        raise Exception("Unable to read file '{0}'".format(file)) from e
    info = {"minutes": a.info.length/60, "nom": file}
    for k in tags:
        try:
            info[k] = str(b[k][0])
        except KeyError:
            continue
    return info


def all_files(repertoire, tags, ext=re.compile(".mp3$")):
    """retourne les informations pour chaque fichier d'un répertoire"""
    all = []
    for r, d, f in os.walk(repertoire):
        for a in f:
            if not ext.search(a):
```

(suite sur la page suivante)

389. https://docs.python.org/3/library/re.html?highlight=re#re.Match
390. https://docs.python.org/3/library/re.html?highlight=re#re.search
391. https://docs.python.org/3/library/re.html?highlight=re#re.match
392. https://fr.wikipedia.org/wiki/MPEG-1/2_Audio_Layer_III
393. https://pypi.python.org/pypi/mutagen

(suite de la page précédente)

```
                continue
            try:
                t = infoMP3(r + "/" + a, tags)
            except Exception as e:
                warnings.warn("unable to process {0}".format(e))
                continue
            if len(t) > 0:
                all.append(t)
    return all


def heart_notitle_mots(all, avoid, sep, heart):
    """retourne trois résultats
    - les chansons dont le titre valide l'expression régulière heart
    - les chansons dont le titre valide l'expression régulière avoid
    - le nombre moyen de mots dans le titre d'une chanson"""
    liheart, notitle = [], []
    nbmot, nbsong = 0, 0
    for a in all:
        if "title" not in a:
            notitle.append(a)
            continue
        ti = a["title"].lower()
        if avoid.match(ti):
            notitle.append(a)
            continue
        if heart.search(ti):
            liheart.append(a)
        nbsong += 1
        nbmot += len([m for m in sep.split(ti) if len(m) > 0])
    nbsong = max(nbsong, 1)
    return liheart, notitle, float(nbmot)/nbsong


tags = "title album artist genre tracknumber".split()
all = all_files(r"D:\musique", tags)

avoid = re.compile(
    "^(((audio)?track( )?( - )?[0-9]{1,2})|(piste [0-9]{1,2}))$")
sep = re.compile("[- ,;!'.?&:]")
heart = re.compile("((heart)(?!((ache)|(land))))")
liheart, notitle, moymot = heart_notitle_mots(all, avoid, sep, heart)

print("nombre de mots moyen par titre ", moymot)
print("somme des durée contenant heart ", sum([s["minutes"] for s in liheart]))
print("chanson sans titre ", len(notitle))
print("liste des titres ")
for s in liheart:
    print("   ", s["title"])
```

>>>

```
    nombre de mots moyen par titre  0.0
    somme des durée contenant heart  0
    chanson sans titre  0
    liste des titres
```

### Groupes nommés

Une expression régulière ne sert pas seulement de filtre, elle permet également d'extraire le texte qui correspond à chaque groupe, à chaque expression entre parenthèses. L'exemple suivant montre comment récupérer le jour, le mois, l'année à l'intérieur d'une date.

<<<

```
import re
date = "05/22/2010"
exp = "([0-9]{1,2})/([0-9]{1,2})/(((19)|(20))[0-9]{2})"
com = re.compile(exp)
print(com.search(date).groups())     # ('05', '22', '2010', '20', None, '20')
```

>>>

```
    ('05', '22', '2010', '20', None, '20')
```

Il n'est pas toujours évident de connaître le numéro du groupe qui contient l'information à extraire. C'est pourquoi il paraît plus simple de les nommer afin de les récupérer sous la forme d'un dictionnaire et non plus sous forme de tableau. La syntaxe `(?P<nom_du_groupe>expression)` permet de nommer un groupe. Elle est appliquée à l'exemple précédent.

<<<

```
import re
date = "05/22/2010"
exp = "(?P<jj>[0-9]{1,2})/(?P<mm>[0-9]{1,2})/(?P<aa>((19)|(20))[0-9]{2})"
com = re.compile(exp)
print(com.search(date).groupdict())  # {'mm': '22', 'aa': '2010', 'jj': '05'}
```

>>>

```
    {'jj': '05', 'mm': '22', 'aa': '2010'}
```

Le programme suivant est un exemple d'utilisation des expressions régulières dont l'objectif est de détecter les fonctions définies dans un programme mais jamais utilisées. Les expressions servent à détecter les définitions de fonctions (d'après le mot-clé `def`) puis à détecter les appels. On recoupe ensuite les informations en cherchant les fonctions définies mais jamais appelées.

Il n'est pas toujours évident de construire une expression régulière qui correspondent précisément à tous les cas qu'on veut détecter. Une stratégie possible est de construire une expression régulière plus permissive puis d'éliminer les cas indésirables à l'aide d'une seconde expression régulière, c'est le cas ici pour détecter les appels.

<<<

```
"""ce programme détermine toutes les fonctions définies dans
un programme et jamais appelées"""
import glob
import os
import re


def trouve_toute_fonction(s, exp, gr, expm="^$"):
    """ à partir d'une chaîne de caractères correspondant
    à un programme Python, cette fonction retourne
    une liste de 3-uples, chacun contient :
        - le nom de la fonction
        - (debut,fin) de l'expression dans la chaîne
        - la ligne où elle a été trouvée
```

(suite sur la page suivante)

(suite de la page précédente)

```
    Paramètres:
       - s    : chaîne de caractères
       - exp  : chaîne de caractères correspond à l'expression
       - gr   : numéro de groupe correspondant au nom de la fonction
       - expm : expression négative
    """
    exp = re.compile(exp)
    res = []
    pos = 0
    r = exp.search(s, pos)    # première recherche
    while r is not None:
        temp = (r.groups()[gr], r.span(gr), r.group(gr))
        x = re.compile(expm.replace("function", temp[0]))
        if not x.match(temp[2]):
            # l'expression négative n'est pas trouvé, on peut ajouter ce résultat
            res.append(temp)
        r = exp.search(s, r.end(gr))      # recherche suivante
    return res


def get_function_list_definition(s):
    """trouve toutes les définitions de fonctions"""
    return trouve_toute_fonction(s,
                                 "\ndef[ ]+([a-zA-Z_][a-zA-Z_0-9]*)[ ]*[(].*[)][ ]*[:]
↪", 0)


def get_function_list_call(s):
    """trouve tous les appels de fonctions"""
    return trouve_toute_fonction(s,
                                 "\n.*[=(,[{ .]([a-zA-Z_][a-zA-Z_0-9]*)(?![ ]?:)[␣
↪]*[(].*[)]?", 0,
                                 "^\\n[ ]*(class|def)[ ]+function.*$")


def detection_fonction_pas_appelee(file):
    """retourne les couples de fonctions jamais appelées suivies
    du numéro de la ligne où elles sont définies"""

    f = open(file, "r")
    li = f.readlines()
    f.close()
    sfile = "".join(li)

    funcdef = get_function_list_definition(sfile)
    funccal = get_function_list_call(sfile)
    f2 = [p[0] for p in funccal]
    res = []
    for f in funcdef:
        if f[0] not in f2:
            ligne = sfile[:f[1][0]].count("\n")
            res.append((f[0], ligne+2))
    return res


def fonction_inutile():  # ligne 63
```

(suite sur la page suivante)

(suite de la page précédente)

```
    pass


file = __file__
print(detection_fonction_pas_appelee(file))
```

>>>

```
    [runpythonerror]
    Traceback (most recent call last):
      File "<stdin>", line 72, in <module>
      File "<stdin>", line 53, in detection_fonction_pas_appelee
    FileNotFoundError: [Errno 2] No such file or directory: '<stdin>'
```

CHAPITRE 8

# Parallélisation

Le langage Python propose plusieurs façons de paralléliser un traitement sans faire appel à des librairies extérieures. C'est ce que propose d'aborder les paragraphes suivants.

## 8.1 Threads


— *Premier thread* (page 260)
— *Synchronisation* (page 262)
  — *Attente* (page 262)
  — *Partage d'informations* (page 265)
— *Interface graphique* (page 266)
— *Files de messages* (page 268)
— *Multithreading plus simple* (page 269)
  — *concurrent.futures* (page 270)
  — *Notion de futures* (page 273)
  — *Un problème de blocage* (page 274)
  — *async - await - asyncio* (page 275)
— *GIL - Global Interpreter Lock* (page 276)
  — *cython* (page 276)
  — *C#, Java* (page 276)
— *Bibliographie* (page 276)


Jusqu'aux années 2003-2004, l'évolution des microprocesseurs était une course vers une augmentation de la puissance, autant en terme de nombre de transistors qu'en fréquence de fonctionnement. Arrivant aux limites de la technologie actuelle, cette évolution s'est tournée maintenant vers la construction de processeurs multicoeurs, c'est-à-dire des machines capables d'exécuter des programmes simultanément, de maintenir plusieurs *fils d'exécution* en parallèle.

Les threads [394] ou fils d'exécution ont trois usages principaux. Le premier est relié au calcul distribué [395] ou calcul parallèle. Par exemple, le calcul d'une intégrale sur un intervalle peut être effectué sur deux intervalles disjoints. Le résultat final est la somme des deux résultats sur chacun des intervalles. De plus, ces deux calculs sont indépendants

394. https://fr.wikipedia.org/wiki/Thread_(informatique)
395. https://fr.wikipedia.org/wiki/Calcul_distribu%C3%A9

et peuvent être menés de front. Le calcul intégral sera donc deux fois plus rapide puisque les deux intervalles seront traités en même temps. C'est la parallélisation des calculs : les deux calculs sur chaque intervalle seront affectés à deux threads simultanés.

Le second usage est couplé aux interfaces graphiques. Lorsque l'utilisateur entame un processus long après avoir cliqué sur un bouton, l'interface graphique ne réagit plus jusqu'à ce que ce processus s'achève. Afin d'éviter cet inconvénient, l'interface graphique va commencer un thread qui va exécuter ce processus. L'interface graphique n'a plus qu'à attendre la fin du thread, et pendant tout ce temps, elle sera également capable de traiter tout autre événement provenant de l'utilisateur.

Le dernier usage concerne la communication entre ordinateurs ou plus généralement la communication Internet. C'est une communication asynchrone [396] : l'ordinateur effectue des tâches en même temps qu'il écoute un port [397] par lequel d'autres ordinateurs communiquent avec lui. Plus précisément, le programme suit deux fils d'exécution : le fil principal et un thread qui ne fait qu'attendre et traiter les messages qu'il reçoit via un port.

La synchronisation est un point commun à ces trois usages. Ce terme désigne la dépendance entre les threads. Lors d'un calcul distribué, le résultat final dépend des résultats retournés par chaque thread, il faut donc attendre que les deux fils d'exécution aient produit le résultat attendu : il faut que les deux fils d'exécution se synchronisent.

### 8.1.1 Premier thread

Le premier exemple consiste à exécuter un thread uniquement destiné à faire des affichages. Deux fils d'exécution vont être lancés en parallèle affichant chacun un message différent. Les affichages vont s'entremêler. Il existe plusieurs manières d'exécuter un thread, une seule sera présentée en utilisant la classe `Thread` du module `threading`. Pour créer un thread, il suffit de surcharger la méthode `run` de la classe `Thread`. Si le thread a besoin de données lors de son exécution, il faut surcharger son constructeur sans oublier d'appeler le constructeur de la classe mère. L'exécution de thread commence par la création d'une instance et l'appel à la méthode `start`. En résumé, il faut retenir les éléments suivants :

1. surcharger la classe `threading.Thread`,
2. surcharger le constructeur sans oublier d'appeler le constructeur `threading.Thread.__init__`,
3. surcharger la méthode `run`, c'est le code que devra exécuter le thread,
4. créer une instance de la nouvelle classe et appeler la méthode `start` pour lancer le thread secondaire qui formera le second fil d'exécution.

Le programme principal est appelé le thread principal. Voici ce que cela donne dans un exemple :

<<<

```
import threading
import time


class MonThread (threading.Thread):
    def __init__(self, jusqua):      # jusqua = donnée supplémentaire
        threading.Thread.__init__(self)  # ne pas oublier cette ligne
        # (appel au constructeur de la classe mère)
        self.jusqua = jusqua           # donnée supplémentaire ajoutée à la classe

    def run(self):
        for i in range(0, self.jusqua):
            print("thread ", i)
            time.sleep(0.08)    # attend 100 millisecondes sans rien faire
            # facilite la lecture de l'affichage
```

(suite sur la page suivante)

396. https://fr.wikipedia.org/wiki/Asynchronisme
397. https://fr.wikipedia.org/wiki/Port_(logiciel)

(suite de la page précédente)

```
m = MonThread(10)           # crée le thread
m.start()                   # démarre le thread,
# l'instruction est exécutée en quelques millisecondes
# quelque soit la durée du thread

for i in range(0, 10):
    print("programme ", i)
    time.sleep(0.1)             # attend 100 millisecondes sans rien faire
    # facilite la lecture de l'affichage
```

>>>

```
    thread  0
    programme  0
    thread  1
    programme  1
    thread  2
    programme  2
    thread  3
    programme  3
    thread  4
    programme  4
    thread  5
    thread  6
    programme  5
    thread  7
    programme  6
    thread  8
    programme  7
    thread  9
    programme  8
    programme  9
```

Le programme affiche des lignes qui proviennent du thread principal et du thread secondaire dont les affichages diffèrent.

Le précédent programme a été adapté pour lancer deux threads secondaires en plus du thread principal. Les lignes modifiées par rapport au programme précédent sont commentées.

<<<

```
import threading
import time


class MonThread (threading.Thread):
    def __init__(self, jusqua, s):
        threading.Thread.__init__(self)
        self.jusqua = jusqua
        self.s = s

    def run(self):
        for i in range(0, self.jusqua):
            print("thread ", self.s, " : ", i)
            time.sleep(0.09)
```

(suite sur la page suivante)

(suite de la page précédente)

```
m = MonThread(10, "A")
m.start()

m2 = MonThread(10, "B")  # crée un second thread
m2.start()                 # démarre le thread,

for i in range(0, 10):
    print("programme ", i)
    time.sleep(0.1)
```

>>>

```
    thread  A  :  0
    thread  B  :  0
    programme  0
    thread  A  :  1
    thread  B  :  1
    programme  1
    thread  A  :  2
    thread  B  :  2
    programme  2
    thread  A  :  3
    thread  B  :  3
    programme  3
    thread  A  :  4
    thread  B  :  4
    programme  4
    thread  A  :  5
    thread  B  :  5
    programme  5
    thread  A  :  6
    thread  B  :  6
    programme  6
    thread  A  :  7
    thread  B  :  7
    programme  7
    thread  B  :  8
    thread  A  :  8
    programme  8
    thread  B  :  9
    thread  A  :  9
    programme  9
```

Tous les exemples présentés dans ce chapitre font souvent intervenir l'instruction time.sleep(...) [398]. A moins que ce ne soit explicitement précisé, elle sert la plupart du temps à ralentir l'exécution du programme cité en exemple afin que celle-ci soit humainement observable ou pour exagérer un défaut de synchronisation. Cette fonction est d'ordinaire beaucoup moins fréquente.

### 8.1.2 Synchronisation

#### Attente

La première situation dans laquelle on a besoin de synchroniser deux threads est l'attente d'un thread secondaire par le thread principal. Et pour ce faire, on a besoin de l'accès par les deux fils d'exécution à une même variable qui indiquera

398. https://docs.python.org/3/library/time.html#time.sleep

l'état du thread. Dans le programme suivant, on ajoute l'attribut `etat` à la classe `MonThread` qui va indiquer l'état du thread :

— `True` pour en marche
— `False` pour à l'arrêt

Le thread principal va simplement vérifier l'état du thread de temps en temps. Le premier point important est tout d'abord d'attendre que le thread se lance car sans la première boucle, le thread pourrait passer à l'état `True` après être passé dans la seconde boucle d'attente. Le second point important est de ne pas oublier d'insérer la fonction sleep [399] afin de permettre au thread principal de temporiser. Dans le cas contraire, le thread principal passe l'essentiel de son temps à vérifier l'état du thread secondaire, ce faisant, il ralentit l'ordinateur par la répétition inutile de la même action un trop grand nombre de fois. Ici, le thread principal vérifie l'état du thread secondaire tous les 100 millisecondes. Cette durée dépend de ce que fait le thread secondaire.

<<<

```
import threading
import time


class MonThread (threading.Thread):
    def __init__(self, jusqua):
        threading.Thread.__init__(self)
        self.jusqua = jusqua
        self.etat = False       # l'état du thread est soit False (à l'arrêt)
        # soit True (en marche)

    def run(self):
        self.etat = True                        # on passe en mode marche
        for i in range(0, self.jusqua):
            print("thread itération ", i)
            time.sleep(0.1)
        self.etat = False                       # on revient en mode arrêt


m = MonThread(10)           # crée un thread
m.start()                   # démarre le thread,

print("début")

while m.etat == False:
    # on attend que le thread démarre
    time.sleep(0.1)  # voir remarque ci-dessous

while m.etat == True:
    # on attend que le thread s'arrête
    # il faut introduire l'instruction time.sleep pour temporiser, il n'est pas
    # nécessaire de vérifier sans cesse que le thread est toujours en marche
    # il suffit de le vérifier tous les 100 millisecondes
    # dans le cas contraire, la machine passe son temps à vérifier au lieu
    # de se consacrer à l'exécution du thread
    time.sleep(0.1)

print("fin")
```

>>>

399. https://docs.python.org/3/library/time.html#time.sleep

```
    thread itération  0
    début
    thread itération  1
    thread itération  2
    thread itération  3
    thread itération  4
    thread itération  5
    thread itération  6
    thread itération  7
    thread itération  8
    thread itération  9
    fin
```

Ce mécanisme d'attente peut également être codé en utilisation les objets Condition [400] et Event [401]. Ces deux objets permettent d'éviter l'utilisation de la méthode sleep [402].

<<<

```
import threading
import time


class MonThread (threading.Thread):
    def __init__(self, jusqua, event):     # event = objet Event
        threading.Thread.__init__(self)   # = donnée supplémentaire
        self.jusqua = jusqua
        self.event = event                  # on garde un accès à l'objet Event

    def run(self):
        for i in range(0, self.jusqua):
            print("thread itération ", i)
            time.sleep(0.1)
        self.event.set()                    # on indique qu'on a fini :


        # on active l'object self.event
print("début")

event = threading.Event()       # on crée un objet de type Event
event.clear()                   # on désactive l'ojet Event
m = MonThread(10, event)        # crée un thread
m.start()                       # démarre le thread,
event.wait()                    # on attend jusqu'à ce que l'objet soit activé
# event.wait (0.1) : n'attend qu'un
print("fin")                     # seulement 1 dizième de seconde
```

>>>

```
    début
    thread itération  0
    thread itération  1
    thread itération  2
    thread itération  3
    thread itération  4
    thread itération  5
```

(suite sur la page suivante)

400. https://docs.python.org/3/library/threading.html?highlight=condition#threading.Condition
401. https://docs.python.org/3/library/threading.html?highlight=event#threading.Event
402. https://docs.python.org/3/library/time.html#time.sleep

(suite de la page précédente)

```
    thread itération  6
    thread itération  7
    thread itération  8
    thread itération  9
    fin
```

La méthode wait [403] de l'objet Event [404] attend que l'objet soit activé. Elle peut attendre indéfiniment ou attendre pendant une durée donnée seulement. Pour afficher la durée d'attente, on pourrait utiliser une boucle comme la suivante :

```
m.start ()
while not event.is_set():
    print("j'attends")
    event.wait (0.1)
print("fin")
```

La méthode is_set [405] permet de savoir si l'événement est bloquant ou non. Le programme affiche `"j'attends"` puis attend le thread un dixième de secondes. Au delà de cette durée, il vérifie l'état de l'événement puis recommence si le thread n'est pas fini.

Ces objets de synchronisation sont plus efficaces que le mécanisme décrit dans le premier programme car il fait appel aux fonctions du système d'exploitation.

### Partage d'informations

La seconde situation dans laquelle on a besoin de synchroniser est l'accès par deux fils d'exécution aux mêmes informations ou plutôt aux mêmes variables. Un problème survient quand parfois un thread lit ou modifie en même temps qu'un autre modifie la même variable. Le second cas de synchronisation est l'ajout de verrous qui permettent de protéger une partie du code d'un programme contre plusieurs accès simultanés.indexfrr{thread}{verrou} Ce verrou est également un objet du module `threading` : Lock [406].

Dans cet exemple, l'information partagée est la chaîne de caractères `message`, le verrou sert à protéger la fonction `ajoute` contre des ajouts simultanés. Si les deux threads veulent modifier `message` en même temps, un thread va entrer dans la fonction `ajoute` alors que l'autre n'en est pas encore sorti. Les résultats seraient imprévisibles car cette fonction modifie la variable qu'ils utilisent. On aboutit à l'exemple suivant :

```
    début
    longueur  20
    message =  11111111112222222222
```

Les trois instructions protégées pourraient être résumées en une seule : `message += c` ; le résultat resterait inchangé. En revanche, en commentant les instructions verrou.acquire() [407] et verrou.release() [408] de ce programme (marquées d'une étoile). La longueur du résultat final `message` est variable alors qu'elle devrait être de 20 puisque les deux threads appellent chacun 10 fois dans la fonction `ajoute`. Le tableau suivant montre l'évolution des variables `message`, `c`, `s` durant deux premiers appels qui s'entremêlent. Le résultat devrait être `"12"` pour message mais un caractère a été perdu. Il faut retenir que si la variable `message` est globale, les deux autres `c`, `s` sont locales et donc différentes pour les deux threads.

403. https://docs.python.org/3.5/library/threading.html?highlight=condition#threading.Event.wait
404. https://docs.python.org/3/library/threading.html?highlight=event#threading.Event
405. https://docs.python.org/3.5/library/threading.html?highlight=condition#threading.Event.is_set
406. https://docs.python.org/3/library/threading.html?highlight=condition#threading.Lock
407. https://docs.python.org/3.5/library/threading.html?highlight=condition#threading.Lock.acquire
408. https://docs.python.org/3.5/library/threading.html?highlight=condition#threading.Lock.release

| ordre | thread 1 | thread 2 | message | c | s |
|---|---|---|---|---|---|
| 1 | `s = message + c` | | `""` | `"1"` | `"1"` |
| 2 | | `s = message + c` | `""` | `"2"` | `"2"` |
| 3 | `time.sleep (0.001)` | | `""` | `"1"` | `"1"` |
| 4 | | `time.sleep (0.001)` | `""` | `"2"` | `"2"` |
| 5 | `message = s` | | `"1"` | `"1"` | `"1"` |
| 6 | | `message = s` | `"2"` | `"2"` | `"2"` |

Le verrou empêche d'exécuter une même portion de code en même temps, un code qui modifie des données partagées. C'est pourquoi le verrou est souvent déclaré au même endroit que les données qu'il protège. Le verrou de type Lock [409] n'autorise qu'un seul thread à la fois à l'intérieur de la portion de code protégée ce qui aboutit au schéma suivant :

| ordre | thread 1 | thread 2 | message | c | s |
|---|---|---|---|---|---|
| 1 | `s = message + c` | | `""` | `"1"` | `"1"` |
| 2 | `time.sleep (0.001)` | | `""` | `"1"` | `"1"` |
| 3 | `message = s` | | `"1"` | `"1"` | `"1"` |
| 4 | | `s = message + c` | `"1"` | `"2"` | `"12"` |
| 5 | | `time.sleep (0.001)` | `"1"` | `"2"` | `"12"` |
| 6 | | `message = s` | `"12"` | `"2"` | `"12"` |

Le verrou de type Semaphore [410] autorise un nombre maximal de thread à parcourir le même code. Ce procédé est parfois utile si le code en question permet d'imprimer un document. Cela permet de limiter sans interdire les accès simultanés aux ressources de l'ordinateur.

Ce mécanisme de verrou peut aboutir à des blocages avec deux threads et deux portions de code protégées. Chaque thread est ńăcoincéă" dans une section attendant que l'autre libère la sienne. Dans ce cas de figure, il est conseillé d'utiliser le même verrou pour protéger les deux sections. Ainsi, chaque thread ne pourra pas entrer dans l'une ou l'autre des portions de code protégées tant que l'une d'entre elles est visitée par l'autre thread.

### 8.1.3 Interface graphique

Un programme bâti autour d'une interface graphique inclut nécessairement une boucle de message. Celle-ci attend les messages en provenance de l'interface. Lorsqu'un de ceux-ci lui commande de lancer un traitement long, l'interface graphique n'est plus en mesure de réagir aux événements qui lui viennent pendant ce temps. Afin de remédier cela, il

409. https://docs.python.org/3/library/threading.html?highlight=condition#threading.Lock
410. https://docs.python.org/3.5/library/threading.html?highlight=condition#threading.Semaphore

suffit d'insérer le traitement dans un thread. A la fin de ce dernier, un événement sera envoyé à l'interface afin de lui signifier la fin du traitement.

Le paragraphe parag_graph_bind a montré comment associer un événement particulier à une fenêtre. La différence ici est que l'événement accroché à la fenêtre n'est pas prédéfini par le module Tkinter[411] mais par le programme lui-même - dans cet exemple <<thread_fini>> -. Les symboles <<>> au début et à la fin du nom de l'événement sont la seule contrainte. La méthode `event_generate` permet d'insérer un message dans la boucle de messages de façon à ce que celui-ci soit traité au même titre qu'un clic de souris, la pression d'une touche, ...

```
import threading, time, random, copy

# définition du thread
class MonThread (threading.Thread) :
    def __init__ (self, win, res) :
        threading.Thread.__init__ (self)
        self.win = win  # on mémorise une référence sur la fenêtre
        self.res = res

    def run (self) :
        for i in range (0, 10) :
            print("thread ", i)
            time.sleep (0.1)

          # afin que le thread retourne un résultat
          # self.res désigne thread_resultat qui reçoit un nombre de plus
        h = random.randint (0,100)
        self.res.append (h)

          # on lance un événement <<thread_fini>> à la fenêtre principale
          # pour lui dire que le thread est fini, l'événement est ensuite
          # géré par la boucle principale de messages
          # on peut transmettre également le résultat lors de l'envoi du message
          # en utilisant un attribut de la classe Event pour son propre compte
        self.win.event_generate ("<<thread_fini>>", x = h)

thread_resultat = []

def lance_thread () :
    global thread_resultat
      # fonction appelée lors de la pression du bouton
      # on change la légnde de la zone de texte
    text .config (text = "thread démarré")
    text2.config (text = "thread démarré")
      # on désactive le bouton pour éviter de lancer deux threads en même temps
    bouton.config (state = TK.DISABLED)
      # on lance le thread
    m = MonThread (root, thread_resultat)
    m.start ()

def thread_fini_fonction (e) :
    global thread_resultat
      # fonction appelée lorsque le thread est fini
    print("la fenêtre sait que le thread est fini")
      # on change la légende de la zone de texte
    text .config (text = "thread fini + résultat " + str (thread_resultat))
    text2.config (text = "thread fini + résultat (e.x) " + str (e.x))
      # on réactive le bouton de façon à pouvoir lancer un autre thread
```

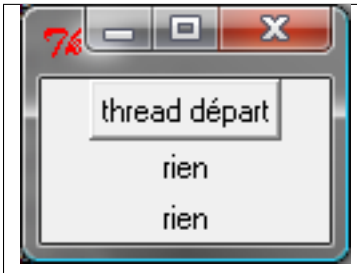
(suite sur la page suivante)

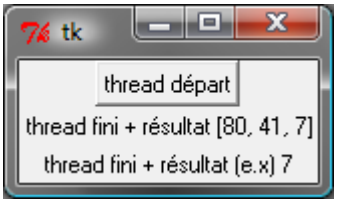
411. https://docs.python.org/3.5/library/tkinter.html?highlight=tkinter#module-tkinter

(suite de la page précédente)

```
    bouton.config (state = TK.NORMAL)

import tkinter as TK

# on crée la fenêtre
root   = TK.Tk ()
bouton = TK.Button (root, text = "thread départ", command = lance_thread)
text   = TK.Label (root, text = "rien")
text2  = TK.Label (root, text = "rien")
bouton.pack ()
text.pack ()
text2.pack ()

# on associe une fonction à un événement <<thread_fini>> propre au programme
root.bind ("<<thread_fini>>", thread_fini_fonction)

# on active la boucle principale de message
root.mainloop ()
```

L'image suivante est la fenêtre affichée par le programme lorsqu'elle attend la pression du bouton qui lance le thread et lorsqu'elle attend la fin de l'exécution de ce thread.




Le programme précédent utilise une astuce pour retourner un résultat autrement qu'un utilisant un paramètre global. On peut adjoindre lors de l'appel à la méthode `event_generate` quelques informations supplémentaires attachées à l'événement en utilisant les attributs prédéfinis de la classe Event[412]. Dans cet exemple, on utilise l'attribut `x` pour retourner le dernier entier tiré aléatoirement.

La première image est la fenêtre après trois exécutions du thread. La liste `thread_resultat` contient trois nombres. Une fois l'unique bouton pressé, la fenêtre change d'aspect pour devenir comme la seconde image. Cette transition est assurée par la fonction `lance_thread` reliée au bouton. La transition inverse est assurée par la fonction `thread_fini_fonction` qui est reliée à l'événement que génère le thread lorsqu'il a terminé.

### 8.1.4 Files de messages

Les trois usages principaux des threads sont le calcul distribué, la conception d'une interface graphique réactive et l'attente permanente d'événements. En ce qui concernent les deux premiers usages, on peut considérer qu'il existe un thread principal qui lance et attend l'exécution de threads secondaires. Les échanges d'informations ont lieu au début et à la fin de chaque thread. Il n'est pas toujours nécessaire de partager des variables en cours d'exécution : l'usage de verrous est peu fréquent pour ces deux schémas sauf pour partager des informations en cours d'exécution. Ils ralentissent considérablement l'exécution d'un programme.

En ce qui concerne le troisième usage, c'est un cas où tout au long du programme, il y a constamment au moins deux threads actifs : un thread principal et un thread qui attend. Dans ce cas, l'échange et la synchronisation d'informations

412. https://docs.python.org/3/library/threading.html?highlight=event#threading.Event

est inévitable et il est souvent fastidieux de concevoir la synchronisation. C'est pourquoi on la conçoit presque toujours sous forme de messages que les threads s'envoient.

Il existe un objet Queue [413] qui facilite cet aspect. C'est une liste qui possède son propre verrou de sorte que n'importe quel thread peut ajouter ou retirer des éléments de cette liste. Elle est utilisée principalement via quatre méthodes. Deux méthodes get [414] sont utilisées au sein du thread qui possède la pile. Deux méthodes put [415] sont appelées par des threads étrangers.

| | |
|---|---|
| `get([timeout=...]` | Retourne un élément de la liste ou attend qu'il y en ait un, le supprime si elle en trouve un. Si `timeout` est renseigné, la fonction attend au plus `timeout` secondes, sinon, elle déclenche l'exception Empty [416]. |
| `get_nowait()` | Retourne un élément de la liste s'il y a en un, dans ce cas, cet élément est supprimé. Dans le cas contraire, la méthode déclenche l'exception Empty [417]. |
| `put(e[,timeout=...])` | Ajoute l'élément `e` à la liste ou attend qu'une place se libère si la liste est pleine. Si `timeout` est renseigné, la fonction attend au plus `timeout` secondes, sinon, elle déclenche l'exception Full [418]. |
| `put_nowait(e)` | Ajoute l'élément `e` à la liste ou déclenche l'exception Full [419] si la liste est pleine. |
| `qsize()` | Retourne la taille de la pile. |

Cette pile est utilisée dans l'exemple qui suit pour simuler deux joueurs qui essaye de découvrir le nombre que l'autre joueur a tiré au hasard. A chaque essai, un joueur envoie un message de type `("essai", n)` à l'autre joueur pour dire qu'il joue le nombre `n`. Ce joueur lui répond avec des messages de type `("dessous", n)`, `("dessus", n)`, `("gagne", n)`.

```
A  : je joue ( 8 )
B  : je joue ( 9 )
A  : je tente  9  écart  10  à traiter  0
A  : je tente  7  écart  10  à traiter  0
B  : j'ai perdu après  0  essais
B  : j'arrête
A  : j'ai gagné en  2  essais, solution  9
A  : j'arrête
```

Les affichages se chevauchent parfois, il faudrait pour éviter cela synchroniser l'affichage à l'aide d'un verrou.

### 8.1.5 Multithreading plus simple

Le module threading [420] a beaucoup été utilisé mais d'autres modules ont été ajoutés à la distribution standard de python.

- concurrent.futures [421] : le module propose une interface similaire pour paralléliser avec des threads ou des processus. La création des threads s'écrit plus rapidement.
- asyncio [422] : ce module fonctionne avec les mots-clés async, await [423] et il est particulièrement adapté à la parallélisation à des accès aux ressources.

Le premier module est plus adapté à la parallélisation des calculs, le second est particulière utile pour écrire des applications non bloquantes qui gère pas mal d'accès à Internet.

413. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Queue
414. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Queue.get
415. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Queue.put
416. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Empty
417. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Empty
418. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Full
419. https://docs.python.org/3.5/library/queue.html?highlight=queu#queue.Full
420. https://docs.python.org/3/library/threading.html
421. https://docs.python.org/3/library/concurrent.futures.html#module-concurrent.futures
422. https://docs.python.org/3/library/asyncio-task.html
423. https://docs.python.org/3/library/asyncio-task.html#example-hello-world-coroutine

### concurrent.futures

Le module concurrent.futures [424] implémente une classe Executor [425] qui définit une interface pour l'exécution en parallèle. On peut soit :

— soumettre l'exécution d'une fonction avec submit [426],

— ou soumettre l'exécution de la même fonction appliquée à séquence de jeux de paramètres avec map [427].

Cette classe est dérivée en un ThreadPoolExecutor [428] dont le principal argument *max_works* définit le nombre de threads à exécuter en parallèle. Je reproduis ici l¿exemple [429] de la documentation de *Python* qui détermine si un nombre est premier.

<<<

```
import concurrent.futures
import math

PRIMES = [
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419]


def is_prime(n):
    if n % 2 == 0:
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))
    for i in range(3, sqrt_n + 1, 2):
        if n % i == 0:
            return False
    return True


def main():
    with concurrent.futures.ThreadPoolExecutor(2) as executor:
        for number, prime in zip(PRIMES, executor.map(is_prime, PRIMES)):
            print('%d is prime: %s' % (number, prime))


main()
```

>>>

```
    112272535095293 is prime: True
    112582705942171 is prime: True
    112272535095293 is prime: True
    115280095190773 is prime: True
    115797848077099 is prime: True
    1099726899285419 is prime: False
```

424. https://docs.python.org/3/library/concurrent.futures.html#module-concurrent.futures
425. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Executor
426. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Executor.submit
427. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Executor.map
428. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.ThreadPoolExecutor
429. https://docs.python.org/3/library/concurrent.futures.html#processpoolexecutor-example

Débugger un programme en parallèle n'est pas chose facile car les exécutions s'entremêlent et les instructions *print* si elles sont insérées dans la fonction parallélisée produisent des résultats indéchiffrables.

<<<

```
import concurrent.futures
import math

PRIMES = [
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419]


def is_prime(n):
    print("start", n, "*")
    if n % 2 == 0:
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))
    for i in range(3, sqrt_n + 1, 2):
        if n % i == 0:
            return False
    print("end", n, "#")
    return True


def main():
    with concurrent.futures.ThreadPoolExecutor(2) as executor:
        for number, prime in zip(PRIMES, executor.map(is_prime, PRIMES)):
            print('%d is prime: %s' % (number, prime))


main()
```

>>>

```
    start 112272535095293 *
    start 112582705942171 *
    end 112582705942171 #
    start 112272535095293 *
    end 112272535095293 #
    start 115280095190773 *
    112272535095293 is prime: True
    112582705942171 is prime: True
    end 115280095190773 #
    start 115797848077099 *
    end 112272535095293 #
    start 1099726899285419 *
    112272535095293 is prime: True
    115280095190773 is prime: True
    end 115797848077099 #
    115797848077099 is prime: True
    1099726899285419 is prime: False
```

Pour débugger, il faut utiliser le module logging [430] (voir aussi Ecrire des logs en Python [431]). L'exemple suivant construit un *logger* par thread

<<<

```
import concurrent.futures
import math
import logging
import sys
import threading

loggers = {}


def get_logger(name):
    if name in loggers:
        return loggers[name]
    logger = logging.getLogger(name)
    logger.setLevel(logging.DEBUG)
    fmt = '%(asctime)s - %(threadName)s - %(levelname)s - %(message)s'
    formatter = logging.Formatter(fmt)

    # Pour un affiche sur la sortie standard mais cela s'entremêle parfois.
    # ch = logging.StreamHandler(sys.stdout)
    # ch.setFormatter(formatter)
    # logger.addHandler(ch)

    # Pour une sortie dans un fichier.
    # Le mode "w" signifie que les logs de l'exécution précédente
    # seront effacés.
    fh = logging.FileHandler(name + ".log", "w")
    fh.setFormatter(formatter)
    logger.addHandler(fh)

    loggers[name] = logger
    return loggers[name]


PRIMES = [
    112272535095293,
    112582705942171,
    112272535095293,
    115280095190773,
    115797848077099,
    1099726899285419]


def is_prime(n):
    logger = get_logger(threading.current_thread().name)
    logger.info("start {}*".format(n))
    if n % 2 == 0:
        logger.info("end1 {}*".format(n))
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))
    for i in range(3, sqrt_n + 1, 2):
```

(suite sur la page suivante)

430. https://docs.python.org/3/library/logging.html#module-logging
431. http://sametmax.com/ecrire-des-logs-en-python/

(suite de la page précédente)

```
        if n % i == 0:
            logger.info("end2 {}*".format(n))
            return False
    logger.info("end3 {}*".format(n))
    return True


def main():
    with concurrent.futures.ThreadPoolExecutor(2, "thread") as executor:
        for number, prime in zip(PRIMES, executor.map(is_prime, PRIMES)):
            print('%d is prime: %s' % (number, prime))


main()

print("-----")

with open("thread_0.log", "r") as f:
    print(f.read())

print("-----")

with open("thread_1.log", "r") as f:
    print(f.read())
```

>>>

```
    112272535095293 is prime: True
    112582705942171 is prime: True
    112272535095293 is prime: True
    115280095190773 is prime: True
    115797848077099 is prime: True
    1099726899285419 is prime: False
    -----
    2018-08-28 03:07:51,401 - thread_0 - INFO - start 112272535095293*
    2018-08-28 03:07:55,552 - thread_0 - INFO - end3 112272535095293*
    2018-08-28 03:07:55,559 - thread_0 - INFO - start 112272535095293*
    2018-08-28 03:07:59,578 - thread_0 - INFO - end3 112272535095293*
    2018-08-28 03:07:59,583 - thread_0 - INFO - start 115797848077099*
    2018-08-28 03:08:02,270 - thread_0 - INFO - end3 115797848077099*

    -----
    2018-08-28 03:07:51,415 - thread_1 - INFO - start 112582705942171*
    2018-08-28 03:07:55,559 - thread_1 - INFO - end3 112582705942171*
    2018-08-28 03:07:55,574 - thread_1 - INFO - start 115280095190773*
    2018-08-28 03:07:59,646 - thread_1 - INFO - end3 115280095190773*
    2018-08-28 03:07:59,652 - thread_1 - INFO - start 1099726899285419*
    2018-08-28 03:08:00,899 - thread_1 - INFO - end2 1099726899285419*
```

### Notion de futures

Ce concept est apparu récemment dans les langages de programmation, non pas qu'il n'est jamais été utilisé avant l'an 2000 mais l'usage de plus en plus fréquent de la programmation parallélisée fait que certains concept sont nommés et intègres les langages. Les futures ou promesses [432] font référence à un résultat dont le calcul est géré par un autre thread ou processus. Le résultat n'est pas prêt au moment où ce second thread démarre mais il le sera bientôt d'où son

432. https://fr.wikipedia.org/wiki/Futures_(informatique)

nom. On les retrouve en C# Programmation asynchrone avec Async et Await [433] ou C++ std ::async [434]. Il y a deux objets *futures* en Python qui sont produits par différents jeux de fonctions. On ne créé jamais un *futures*, c'est toujours une fonction qui le fait.

— concurrent.futures.Future [435] : ils sont créés par le module concurrent.futures [436].
— asyncio.future [437] : ils sont créés par le module asyncio [438].

Les deux objets possèdent la même interface et sont presque compatibles. Cela dit, il vaut mieux éviter de les mélanger. Je cite la documentation :

> This class is not compatible with the wait() [439] and as_completed() [440] functions in the concurrent.futures package.

Distribuer l'exécution d'une fonction est relativement facile. Les choses se compliquent quand il s'agit de distribuer un calcul qui dépend d'un autre calcul distribué. Il faut enchaîner ces fonctions qu'on dit asynchrones [441] puisque leur exécution n'est plus inscrite dans une seule et même séquence mais dans plusieurs fils d'exécution parallèles.

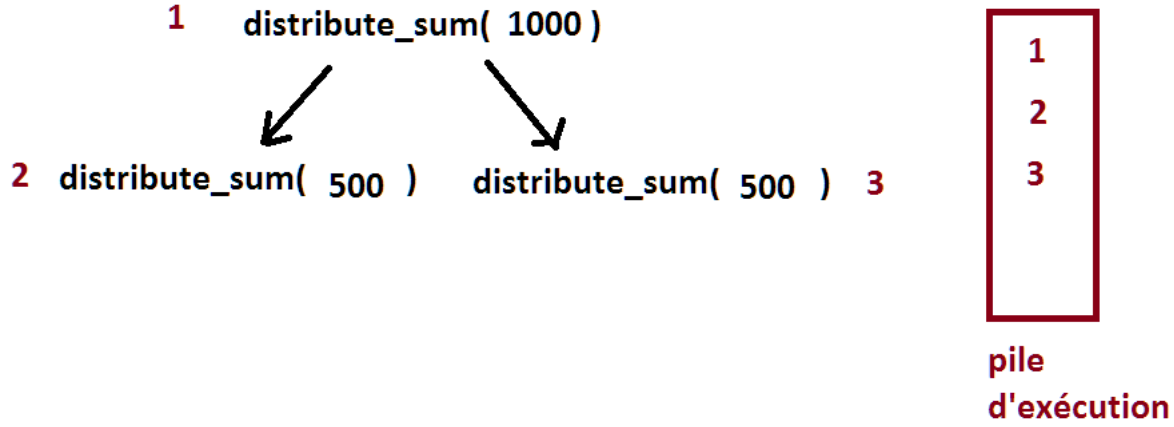
### Un problème de blocage

La fonction distribue le calcul de la somme des éléments d'un tableau. et elle est récursive.

```
def distribute_sum(executor, array, i=0, j=-1):
    if j == -1:
        j = len(array)
    if j - i == 1:
        return array[i]
    elif j - i < 10:
        return sum(array[i:j])
    else:
        m = (i + j) // 2
        # on utilise le même executor
        f1 = executor.submit( distribute_sum, executor, array, i, m)
        f2 = executor.submit( distribute_sum, executor, array, m, j)
        return f1.result() + f2.result()

import random
array = [random.random() for n in range(0, 1000)]
executor = concurrent.futures.ThreadPoolExecutor(max_workers=2)
future = executor.submit(distribute_sum, executor, array)
future.result()
```

Il faut voir l'objet *executor* comme un objet qui empile les fonctions à exécuter. Le problème dans l'exemple précédent est que la fonction *distribute_sum* est déjà dans la pile d'exécution et attend l'exécution de deux autres appels à la même fonction placée après elle dans la pile d'exécution.

433. https://msdn.microsoft.com/fr-fr/library/hh191443(v=vs.120).aspx
434. http://en.cppreference.com/w/cpp/thread/async
435. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.Future
436. https://docs.python.org/3/library/concurrent.futures.html#module-concurrent.futures
437. https://docs.python.org/3/library/asyncio-task.html#future
438. https://docs.python.org/3/library/asyncio.html
439. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.wait
440. https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.as_completed
441. https://en.wikipedia.org/wiki/Synchronous_programming_language




Si chaque appel à la fonction était effectué sur un thread différent, cela pourrait fonctionner. Mais ce n'est pas le cas pour cette implémentation. L'appel 1 attend la fin de 2 et 3 qui ne peuvent pas être exécutés tant que 1 n'est pas fini. Pour résoudre le problème dans ce cas ci, il faut remplacer le commentaire par la ligne suivante :

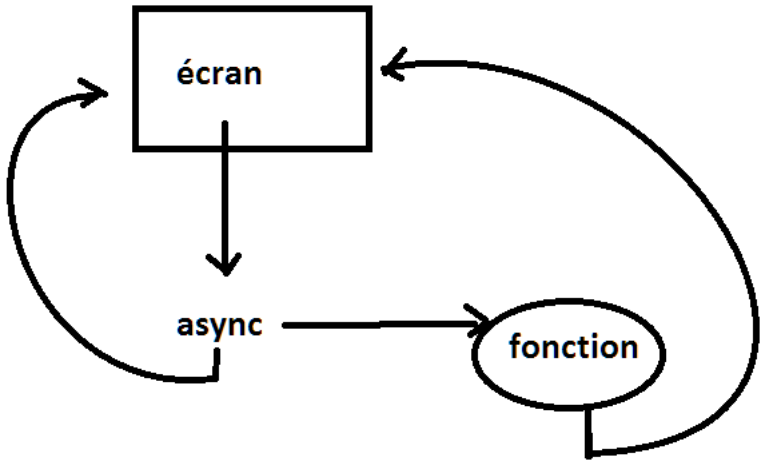
```
executor = concurrent.futures.ThreadPoolExecutor(max_workers=2)
```

### async - await - asyncio

asyncio [442] a fait émerger les mots-clés async and await [443] qui font partie du langage depuis la version 3.5 tout comme elles font partie d'autres langages comme C# [444] ou C++ [445].

Concrètement, ce n'est pas si difficile d'écrire une fonction qui a vocation à être parallélisée. Ce qui devient compliqué est d'avoir à sa disposition plein de fonctions à paralléliser. Il faut synchroniser. Ces deux mots-clés permettent de définir une fonction à vocation parallèle (`async`) et une façon d'attendre qu'elles aient retourné un résultat (`await`). Ces deux mots-clés sont une façon élégant de créer des assemblages de fonctions indépendantes et parallélisées.

Les interfaces graphiques ne contiennent qu'un seul *await* ou une seule boucle de messages qui attend inlassablement que quelque chose se passe. Le mot-clé *async* agit comme un aiguillage. Une action est enclenchée. Elle signalera sa fin et son résultat plus tard.




442. https://docs.python.org/3/library/asyncio.html
443. https://docs.python.org/3/whatsnew/3.5.html?highlight=async#whatsnew-pep-492
444. https://docs.microsoft.com/en-us/dotnet/csharp/async
445. http://www.cplusplus.com/reference/future/async/

Le mot-clés *await* sert à chaîner les fonctions parallélisées.

### 8.1.6 GIL - Global Interpreter Lock

Le Global Interpreter Lock [446] est un verrou qui évite à plusieurs threads de modifier le même objet en même temps. Dans les langages bas niveau, on fait la distrinction entre un tableau ou une liste qui supporte les accès concurrentiels ou non. Si elle ne les supporte pas, les accès sont plus rapides mais suppose que le développeur s'occupe de gérer les problèmes de synchronisation si besoin.

Le langage Python protège listes et dictionnaires par l'intermédiaire de ce verrou qui est unique pour toutes les listes afin de pouvoir gérer efficacement le garbage collector [447] (voir module gc [448]). En conséquence, si le langage Python est multithread par design, dans les faits, il ne l'est presque pas car le *GIL* est sans cesse utilisé. Le notebook *Le GIL* (page **??**) finira de vous convaincre.

#### cython

Tout est possible avec le langage C même si cython [449] est un mélange des de *python* et *C*. Le contexte nogil [450] permet de relâcher la contrainte sur GIL pour une fonction ou une partie de code. Plus de liberté veut dire aussi plus d'attention à apporter au code. La page Using parallelism [451] donne quelques exemples simples de parallélisation. Il est plus facile de paralléliser Python avec un autre langage (voir aussi Parallelizing numpy array loops with Cython and OpenMP [452]).

#### C#, Java

L'intégration de langage C est supportée par Python car l'interpéteur est implémenté en C. Utiliser d'autres langages peut se faire via une interface en C et celle-ci existe pour certains langages. C'est le cas du C# [453] qui peut être utilisé via le module pythonnet [454] ou encore Java [455] via py4j [456], pyjnius [457]. Les threads sont plus faciles à implémenter dans ces langages même si le garbage collector [458] peut nuire aux performances.

### 8.1.7 Bibliographie

*articles*

— Aynschonous API for Python [459]

Il existe d'autres librairies qui ont été développés pour des usages spécifiques telles que joblib [460] qui est utilisé par scikit-learn [461]. La librairie gevent [462] est un équivalent de la librairie asyncio [463] qui a été intégrée à Python 3.4. Plus récemment, le package uvloop [464] propose une accélération de deux à quatre fois par rapport à la librairie asyncio [465] :

446. https://en.wikipedia.org/wiki/Global_interpreter_lock
447. https://fr.wikipedia.org/wiki/Ramasse-miettes_(informatique)
448. https://docs.python.org/3/library/gc.html
449. http://cython.org/
450. http://cython.readthedocs.io/en/latest/src/userguide/external_C_code.html#releasing-the-gil
451. http://cython.readthedocs.io/en/latest/src/userguide/parallelism.html?highlight=nogil#using-parallelism
452. http://www.perrygeo.com/parallelizing-numpy-array-loops-with-cython-and-mpi.html
453. https://fr.wikipedia.org/wiki/C_sharp
454. https://github.com/pythonnet/pythonnet
455. https://fr.wikipedia.org/wiki/Java
456. https://www.py4j.org/
457. https://github.com/kivy/pyjnius
458. https://fr.wikipedia.org/wiki/Ramasse-miettes_(informatique)
459. http://stevedower.id.au/blog/async-api-for-python/
460. https://pythonhosted.org/joblib/
461. http://scikit-learn.org/stable/
462. http://www.gevent.org/
463. https://docs.python.org/3/library/asyncio.html
464. https://uvloop.readthedocs.io/user/index.html
465. https://docs.python.org/3/library/asyncio.html

uvloop : Blazing fast Python networking [466]. Cette dernière librairie est utilisée par le module sanic [467] qui implémenté un serveur web plus rapide que flask [468]. La page Parallel Processing in Python [469] passe en revue différentes stratégies de parallélisation pour l'implémentation de calculs numériques avec joblib [470], cython [471], OpenMP [472].

466. https://magic.io/blog/uvloop-blazing-fast-python-networking/
467. http://sanic.readthedocs.io/en/latest/
468. http://flask.pocoo.org/
469. https://homes.cs.washington.edu/~jmschr/lectures/Parallel_Processing_in_Python.html
470. https://pythonhosted.org/joblib/
471. http://cython.org/
472. https://fr.wikipedia.org/wiki/OpenMP

CHAPITRE 9

# Interfaces graphiques

## 9.1 tkinter


— *Introduction* (page 280)
— *Les objets* (page 281)
  — *Zone de texte* (page 281)
  — *Bouton* (page 282)
  — *Zone de saisie* (page 283)
  — *Zone de saisie à plusieurs lignes* (page 284)
  — *Case à cocher* (page 285)
  — *Case ronde ou bouton radio* (page 285)
  — *Liste* (page 286)
  — *Liste avec barre de défilement, Combobox* (page 288)
  — *Canevas* (page 289)
  — *Menus* (page 289)
  — *Méthodes communes* (page 289)
— *Disposition des objets dans une fenêtre* (page 291)
  — *Emplacements* (page 291)
  — *Sous-fenêtre* (page 293)
— *Evénements* (page 294)
  — *Fenêtre principale* (page 294)
  — *Focus* (page 294)
  — *Lancer une fonction lorsqu'un bouton est pressé* (page 294)
  — *Associer n'importe quel événement à un objet* (page 295)
  — *Menu* (page 300)
  — *Fonctions prédéfinies* (page 302)
  — *D'autres fenêtres et contrôles* (page 303)
  — *Créer une seconde boîte de dialogues* (page 303)
  — *Fenêtres standard* (page 304)
— *Constructions classiques* (page 308)
  — *Compte à rebours* (page 309)
  — *Contrôles personnalisés : utilisation des classes* (page 309)

— *Fenêtres personnalisées : utilisation des classes* (page 310)
— *Séquence d'événements* (page 312)
— *Communiquer un résultat par message* (page 314)

Les interfaces graphiques servent à rendre les programmes plus conviviaux. Elles sont pratiques à utiliser mais elles demandent un peu de temps pour les concevoir. Un programme sans interface exécute des instructions les unes à la suite des autres, le programme a un début - un point d'entrée - et une fin. Avec une interface, le programme fonctionne de manière différente. Il n'exécute plus successivement les instructions mais attend un événement - pression d'une touche du clavier, clic de souris - pour exécuter une fonction. C'est comme si le programme avait une multitude de points d'entrée.

Il existe plusieurs modules permettant d'exploiter les interfaces graphiques. Le plus simple est le module tkinter [473] présent lors de l'installation du langage *Python*. Ce module est simple mais limité. Le module wxPython [474] est une alternative plus riche mais il faut vérifier que sa dernière mise à jour est assez récente. La plus connue est PyQt5 [475] (ou PyQt4 [476]). Le module tkinter [477] fait partie de la distribution standard de *Python* et sera disponible dans toutes les verions de *Python*. Visuellement, *tkinter* est moins *joli* que d'autres extensions mais il vaut mieux vérifier la fréquence des mises à jour de leur code source avant d'en choisir une github/wxPython [478] github/PyQt5 [479]. La licence de wxPython [480] est plus souple. D'autres alternatives sont disponibles à Other Graphical User Interface Packages [481].

Le fonctionnement des interfaces graphiques sous un module ou un autre est presque identique. C'est pourquoi ce chapitre n'en présentera qu'un seul, le module tkinter [482]. Pour d'autres modules, les noms de classes changent mais la logique reste la même : il s'agit d'associer des événements à des parties du programme *Python*.

Les interfaces graphiques évoluent sans doute plus vite que les autres modules, des composantes de plus en plus complexes apparaissent régulièrement. Une excellente source de documentation sont les forums de discussion qui sont un lieu où des programmeurs échangent questions et réponses. Un message d'erreur entré sur un moteur de recherche Internet permet souvent de tomber sur des échanges de ce type, sur des problèmes résolus par d'autres. Le plus connu est stackoverflow/tkinter [483].

### 9.1.1 Introduction

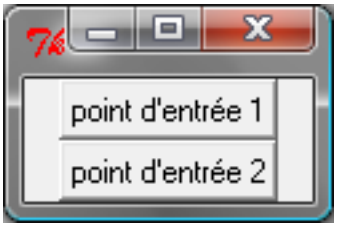
Un programme muni d'une interface graphique fonctionne différemment d'un programme classique. Un programme classique est une succession presque linéaire d'instructions. Il y a un début ou *point d'entrée* du programme et aucun événement extérieur ne vient troubler son déroulement. Avec une interface graphique, le point d'entrée du programme est masqué : il est pris en compte automatiquement. Du point de vue du programmeur, le programme a plusieurs points d'entrée : une simple fenêtre avec deux boutons propose deux façons de commencer et il faut prévoir une action associée à chaque bouton.

La conception d'une interface graphique se déroule généralement selon deux étapes. La première consiste à dessiner l'interface, c'est-à-dire choisir une position pour les objets de la fenêtre (boutons, zone de saisie, liste déroulante, ...). La seconde étape définit le fonctionnement de la fenêtre, c'est-à-dire associer à chaque objet des fonctions qui seront exécutées si un tel événement se réalise (pression d'un bouton, pression d'une touche, ...).

Pour le moment, nous allons supposer que ces deux étapes sont scindées même si elles sont parfois entremêlées lorsqu'un événement implique la modification de l'apparence de la fenêtre. La section qui suit décrit des objets que propose le module *tkinter*. La section suivante présente la manière de les disposer dans une fenêtre. La section d'après

473. https://docs.python.org/3/library/tk.html
474. https://wxpython.org/
475. http://pyqt.sourceforge.net/Docs/PyQt5/
476. http://pyqt.sourceforge.net/Docs/PyQt4/
477. https://docs.python.org/3/library/tk.html
478. https://github.com/wxWidgets/wxWidgets
479. https://github.com/pyqt/python-qt5
480. https://github.com/wxWidgets/wxWidgets
481. https://docs.python.org/3/library/othergui.html#other-gui-packages
482. https://docs.python.org/3/library/tk.html
483. http://stackoverflow.com/questions/tagged/tkinter

décrit les événements et le moyen de les relier à des fonctions du programme. Ce chapitre se termine par quelques constructions courantes à propos des interfaces graphiques.




### 9.1.2 Les objets

Les interfaces graphiques sont composées d¿*objets* ou *widgets* ou *contrôles*. Comme ils reçoivent des événements, en un sens, ce sont ces objets qui pilotent un programme ou qui le contrôlent.}. Ce paragraphe décrit les principales méthodes qui permettent de modifier le contenu et l'apparence des objets.

Les exemples de codes des paragraphes qui suivent permettent de disposer les objets dans une fenêtre qui ne s'affichera pas sans les quelques lignes de code supplémentaires présentées au paragraphe *Fenêtre principale* (page 294) et l'utilisation d'une méthode pack [484] L'exemple suivant crée un objet :

```
zone_texte = tkinter.Label (text = "zone de texte")
```

Et pour l'afficher, il faut l'enrober :

```
import tkinter          # import de tkinter
root = tkinter.Tk ()    # création de la fenêtre principale
# ...
obj = tkinter.Label (text = "zone de texte")
# ...
obj.pack ()             # on ajoute l'objet à la fenêtre principale
root.mainloop ()        # on affiche enfin la fenêtre principal et on attend
                        # les événements (souris, clic, clavier)
```

#### Zone de texte

Une zone de texte sert à insérer dans une fenêtre graphique une légende indiquant ce qu'il faut insérer dans une zone de saisie voisine comme le montre la figure qui suit. Une zone de texte correspond à la classe `Label`. Pour créer une zone de texte, il suffit d'écrire la ligne suivante :

```
zone_texte = tkinter.Label (text = "zone de texte")
```

Il est possible que le texte de cette zone de texte doive changer après quelques temps. Dans ce cas, il faut appeler la méthode `config` comme suit :

```
zone_texte = tkinter.Label (text = "premier texte")
# ...
# pour changer de texte
zone_texte.config (text = "second texte")
```

La zone de texte ou `Label` est associée à une zone de saisie. La seconde image montre une zone de texte dans l'état `DISABLED`.

484. https://docs.python.org/3/library/tkinter.html#the-packer

L'exemple suivant montre deux zones de texte. La seconde est grisée par rapport à la première. Pour obtenir cet état, il suffit d'utiliser l'instruction suivante :

```
zone_texte.config (state = tkinter.DISABLED)
```

Et pour revenir à un état normal :

```
zone_texte.config (state = tkinter.NORMAL)
```

Ces deux dernières options sont communes à tous les objets d'une interface graphique. Cette option sera rappelée au paragraphe *Méthodes communes* (page 289).

### Bouton

Un bouton a pour but de faire le lien entre une fonction et un clic de souris. Un bouton correspond à la classe Button [485] Pour créer un bouton, il suffit d'écrire la ligne suivante :

```
bouton = tkinter.Button (text = "zone de texte")
```

Il est possible que le texte de ce bouton doive changer après quelques temps. Dans ce cas, il faut appeler la méthode config [486] comme suit :

```
bouton = tkinter.Button (text = "premier texte")
# ...
# pour changer de texte
bouton.config (text = "second texte")
```

| | | |
|---|---|---|
| | | |
| Bouton non pressé | Bouton pressé | Bouton grisé |

Le troisième bouton est grisé par rapport au premier. Les boutons grisés ne peuvent pas être pressés. Pour obtenir cet état, il suffit d'utiliser l'instruction suivante :

```
bouton.config (state = tkinter.DISABLED)
```

Et pour revenir à un état normal :

```
bouton.config (state = tkinter.NORMAL)
```

C'est pour cet objet que cette option est la plus intéressante car elle permet d'interdire la possibilité pour l'utilisateur de presser le bouton tout en le laissant visible.

Il est possible également d'associer une image à un bouton. Par exemple, les trois lignes suivantes créent un bouton, charge une image au format *gif* puis l'associe au bouton `b`. Lors de l'affichage de la fenêtre, le bouton `b` ne contient pas de texte mais une image.

485. http://effbot.org/tkinterbook/button.htm
486. http://effbot.org/tkinterbook/button.htm#Tkinter.Button.config-method

```
b = tkinter.Button()
im = tkinter.PhotoImage(file="chameau.gif
↪")
b.config(image=im)
```

Les images qu'il est possible de charger sont nécessairement au format *GIF*, le seul que le module *tkinter* puisse lire.

### Zone de saisie

Une zone de saisie a pour but de recevoir une information entrée par l'utilisateur. Une zone de saisie correspond à la classe Entry [487] ; pour en créer une, il suffit d'écrire la ligne suivante :

```
saisie = tkinter.Entry ()
```

Pour modifier le contenu de la zone de saisie, il faut utiliser la méthode insert [488] qui insère un texte à une position donnée.

```
# le premier paramètre est la position
# où insérer le texte (second paramètre)
saisie.insert (pos, "contenu")
```

Pour obtenir le contenu de la zone de saisie, il faut utiliser la méthode get [489] :

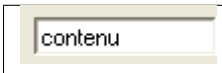
```
contenu = saisie.get ()
```

Pour supprimer le contenu de la zone de saisie, il faut utiliser la méthode delete [490]. Cette méthode supprime le texte entre deux positions.

```
# supprime le texte entre les positions pos1, pos2
saisie.delete (pos1, pos2)
```

Par exemple, pour supprimer le contenu d'une zone de saisie, on peut utiliser l'instruction suivante :

```
saisie.delete (0, len(saisie.get()))
```

|  | |
|---|---|
| Zone de saisie normale | Zone de saisie grisée |

487. http://effbot.org/tkinterbook/entry.htm
488. http://effbot.org/tkinterbook/entry.htm#Tkinter.Entry.insert-method
489. http://effbot.org/tkinterbook/entry.htm#Tkinter.Entry.get-method
490. http://effbot.org/tkinterbook/entry.htm#Tkinter.Entry.delete-method

La seconde zone de saisie est grisée par rapport à la première. Les zones de saisie grisées ne peuvent pas être modifiées. Pour obtenir cet état, il suffit d'utiliser la méthode config [491] comme pour les précédents objets. Cette option sera rappelée au paragraphe *Méthodes communes* (page 289).

### Zone de saisie à plusieurs lignes

Une zone de saisie à plusieurs lignes est identique à la précédente à ceci près qu'elle autorise la saisie d'un texte sur plusieurs lignes. Cette zone correspond à la classe Text [492]. Pour créer une telle zone, il suffit d'écrire la ligne suivante :

```
saisie = tkinter.Text ()
```

Pour modifier le contenu de la zone de saisie, il faut utiliser la méthode insert [493] qui insère un texte à une position donnée. La méthode diffère de celle de la classe *Entry* (page 283) puisque la position d'insertion est maintenant une chaîne de caractères contenant deux nombres séparés par un point : le premier nombre désigne la ligne, le second la position sur cette ligne.

```
# le premier paramètre est la position
# où insérer le texte (second paramètre)
pos = "0.0"
saisie.insert (pos, "première ligne\nseconde ligne")
```

Pour obtenir le contenu de la zone de saisie, il faut utiliser la méthode get [494] qui retourne le texte entre deux positions. La position de fin n'est pas connue, on utilise la chaîne de caractères `"end"` pour désigner la fin de la zone de saisie.

```
# retourne le texte entre deux positions
pos1 = "0.0"
pos2 = "end"  # ou tkinter.END
contenu = saisie.get (pos1, pos2)
```

Pour supprimer le contenu de la zone de saisie, il faut utiliser la méthode delete [495]. Cette méthode supprime le texte entre deux positions.

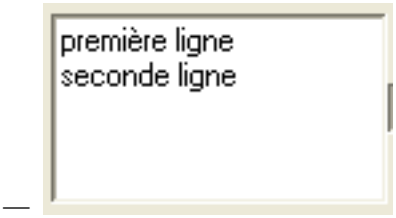
```
# supprime le texte entre les positions pos1, pos2
saisie.delete (pos1, pos2)
```

Par exemple, pour supprimer le contenu d'une zone de saisie à plusieurs lignes, on peut utiliser l'instruction suivante :

```
saisie.delete ("0.0", "end")
# on peut aussi utiliser
# saisie.delete ("0.0", tkinter.END)
```

Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
# modifie les dimensions de la zone
# width <--> largeur
# height <--> hauteur en lignes
saisie.config (width = 10, height = 5)
```




491. http://effbot.org/tkinterbook/entry.htm#Tkinter.Entry.config-method
492. http://effbot.org/tkinterbook/text.htm
493. http://effbot.org/tkinterbook/text.htm#Tkinter.Text.insert-method
494. http://effbot.org/tkinterbook/text.htm#Tkinter.Text.get-method
495. http://effbot.org/tkinterbook/text.htm#Tkinter.Text.delete-method

L'image précédente montre une zone de saisie à plusieurs lignes. Pour griser cette zone, il suffit d'utiliser la méthode config [496] rappelée au paragraphe *Méthodes communes* (page 289).

### Case à cocher

Une case à cocher correspond à la classe Checkbutton [497]. Pour créer une case à cocher, il suffit d'écrire la ligne suivante :

```
# crée un objet entier pour récupérer la valeur de la case à cocher,
# 0 pour non cochée, 1 pour cochée
v    = tkinter.IntVar ()
case = tkinter.Checkbutton (variable = v)
```

En fait, ce sont deux objets qui sont créés. Le premier, de type IntVar [498], mémorise la valeur de la case à cocher. Le second objet, de type Checkbutton [499], gère l'apparence au niveau de l'interface graphique. La raison de ces deux objets est plus évidente dans le cas de l'objet *RadioButton* (page 285) décrit au paragraphe suivant. Pour savoir si la case est cochée ou non, il suffit d'exécuter l'instruction :

```
v.get ()  # égal à 1 si la case est cochée, 0 sinon
```

Pour cocher et décocher la case, il faut utiliser les instructions suivantes :

```
case.select ()      # pour cocher
case.deselect ()    # pour décocher
```

Il est possible d'associer du texte à l'objet case à cocher :

```
case.config (text = "case à cocher")
```

| | |
|---|---|
| Cases à cocher cochée, non cochée, grisée | Case à cocher avec une légende |

La troisième est grisée par rapport à la première. Les cases grisées ne peuvent pas être cochées. Pour obtenir cet état, il suffit d'utiliser la méthode config [500] rappelée au paragraphe *Méthodes communes* (page 289).

### Case ronde ou bouton radio

Une case ronde ou *bouton radio* correspond à la classe Radiobutton [501]. Elles fonctionnent de manière semblable à des cases à cocher excepté le fait qu'elles n'apparaissent jamais seules : elles fonctionnent en groupe. Pour créer un groupe de trois cases rondes, il suffit d'écrire la ligne suivante :

```
# crée un objet entier partagé pour récupérer le numéro du bouton radio activé
v     = tkinter.IntVar ()
case1 = tkinter.Radiobutton (variable = v, value = 10)
case2 = tkinter.Radiobutton (variable = v, value = 20)
case3 = tkinter.Radiobutton (variable = v, value = 30)
```

496. http://effbot.org/tkinterbook/text.htm#Tkinter.Text.config-method
497. http://effbot.org/tkinterbook/checkbutton.htm
498. http://effbot.org/tkinterbook/variable.htm
499. http://effbot.org/tkinterbook/checkbutton.htm
500. http://effbot.org/tkinterbook/checkbutton.htm#Tkinter.Checkbutton.config-method
501. http://effbot.org/tkinterbook/radiobutton.htm

La variable `v` est partagée par les trois cases rondes. L'option `value` du constructeur permet d'associer un bouton radio à une valeur de `v`. Si `v == 10`, seul le premier bouton radio sera sélectionné. Si `v == 20`, seul le second bouton radio le sera. Si deux valeurs sont identiques pour deux boutons radio, ils seront cochés et décochés en même temps. Et pour savoir quel bouton radio est coché ou non, il suffit d'exécuter l'instruction :

```
v.get ()  #  retourne le numéro du bouton radio coché (ici, 10, 20 ou 30)
```

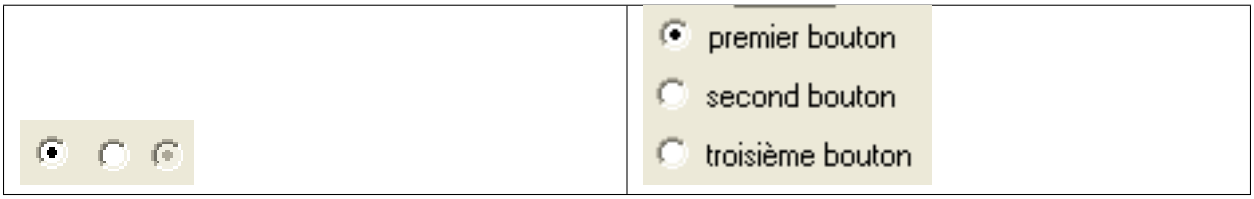
Pour cocher un des boutons radio, il faut utiliser l'instruction suivante :

```
v.set (numero)  # numéro du bouton radio à cocher
                # pour cet exemple, 10, 20 ou 30
```

Il est possible d'associer du texte à un bouton radio.

```
case1.config (text = "premier bouton")
case2.config (text = "second bouton")
case3.config (text = "troisième bouton")
```

Visuellement, cela donne :




Lorsqu'un bouton radio est grisé, son état ne peut être modifié. La seconde image présente un groupe de bouton radio. Un seul peut être sélectionné à la fois à moins que deux boutons ne soient associés à la même valeur. Dans ce cas, ils agiront de pair.

### Liste

Un objet liste contient une liste d'intitulés qu'il est possible de sélectionner. Une liste correspond à la classe ListBox [502]. Pour la créer, il suffit d'écrire la ligne suivante :

```
li  = tkinter.Listbox ()
```

Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
# modifie les dimensions de la liste
# width <--> largeur
# height <--> hauteur en lignes
li.config (width = 10, height = 5)
```

On peut insérer un élément dans la liste avec la méthode insert [503] :

```
pos = 0   # un entier, "end" ou tkinter.END pour insérer ce mot à la fin
li.insert (pos, "première ligne")
```

On peut supprimer des intitulés de cette liste avec la méthode delete [504].

502. http://effbot.org/tkinterbook/listbox.htm
503. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.insert-method
504. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.delete-method

```
pos1 = 0     # un entier
pos2 = None # un entier, "end" ou tkinter.END pour supprimer tous les éléments
            # de pos1 jusqu'au dernier
li.delete (pos1, pos2 = None)
```

Les intitulés de cette liste peuvent ou non être sélectionnés. Cliquer sur un intitulé le sélectionne mais la méthode select_set [505] permet aussi de le faire.

```
pos1 = 0
li.select_set (pos1, pos2 = None)
# sélectionne tous les éléments entre les indices pos1 et
# pos2 inclus ou seulement celui d'indice pos1 si pos2 == None
```

Réciproquement, il est possible d'enlever un intitulé de la sélection à l'aide de la méthode select_clear [506].

```
pos1 = 0
li.select_clear (pos1, pos2 = None)
# retire la sélection de tous les éléments entre les indices
# pos1 et pos2 inclus ou seulement celui d'indice pos1 si pos2 == None
```

La méthode curselection [507] permet d'obtenir la liste des indices des éléments sélectionnés.

```
sel = li.curselection ()
```

La méthode get [508] permet récupérer un élément de la liste tandis que la méthode codes{size} retourne le nombre d'éléments :indextkk{size}indextkk{get}

```
for i in range (0, li.size()):
    print(li.get (i))
```

Exemple de liste. La seconde liste est grisée et ne peut être modifiée.

Pour obtenir l'état grisé, il faut appeler la méthode config [509] et rappelée au paragraphe *Méthodes communes* (page 289). Il est possible d'adjoindre une barre de défilement verticale. Il faut pour cela inclure l'objet dans une sous-fenêtre Frame [510] qui est définie au paragraphe *Sous-fenêtre* (page 293) comme dans l'exemple suivant :

```
frame      = tkinter.Frame (parent)
scrollbar  = tkinter.Scrollbar (frame)
li         = tkinter.Listbox (frame, width = 88, height = 6, \
                              yscrollcommand = scrollbar.set)
scrollbar.config (command = li.yview)
li.pack (side = tkinter.LEFT)
scrollbar.pack (side = tkinter.RIGHT, fill = tkinter.Y)
```

505. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.select_set-method
506. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.select_clear-method
507. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.curselection-method
508. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.get-method
509. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.config-method
510. http://effbot.org/tkinterbook/frame.htm

Il suffit de transposer cet exemple pour ajouter une barre de défilement horizontale. Toutefois, il est préférable d'utiliser un objet prédéfini présent dans le module tix [511] qui est une extension du module tkinter [512]. Elle est présentée au paragraphe *Liste avec barre de défilement, Combobox* (page 288).

Lorsqu'on insère plusieurs objets ListBox [513] dans une seule fenêtre, ces objets partagent par défaut la même sélection. Autrement dit, lorsqu'on clique sur un élément de la seconde ListBox [514], l'élément sélectionné dans la première ne l'est plus. Afin de pouvoir sélectionner un élément dans chaque ListBox [515], il faut ajouter dans les paramètres du constructeur l'option `exportselection=0` comme l'illustre l'exemple suivant :

```
li = tkinter.Listbox (frame, width = 88, height = 6, exportselection=0)
```

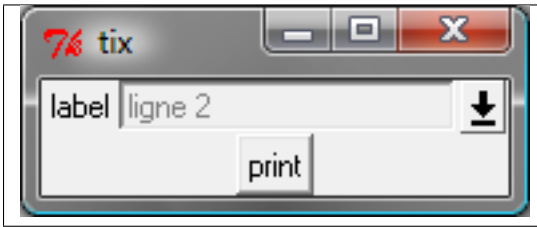
Il existe des méthodes plus avancées qui permettent de modifier l'aspect graphique d'un élément comme la méthode itemconfig [516]. Son utilisation est peu fréquente à moins de vouloir réaliser une belle interface graphique. Le paragraphe *Contrôles personnalisés : utilisation des classes* (page 309) montre l'utilisation qu'on peut en faire.

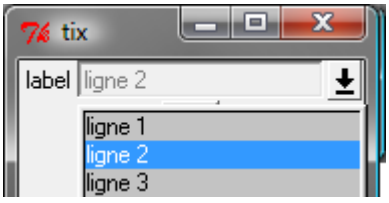
### Liste avec barre de défilement, Combobox

C'est une liste avec une barre de défilement incluse qui est présente dans l'extension ttk [517] qui étend la liste des objets proposés par tkinter [518]. C'est ce que fait l'objet ttk.Combobox [519].

```
import tkinter
import tkinter.ttk as ttk

root = tkinter.Tk()

o = ttk.Combobox(root, values=["ligne 1", "ligne 2", "ligne 3", "ligne 4"])
o.pack ()

def print_file () :                     # voir le chapitre sur les événements
    print(o.get())

b = tkinter.Button (root, text="print")
b.config (command = print_file)         # idem
b.pack ()

root.mainloop()                         # idem
```

Les extensions ttk [520] et tix [521] ne sont pas très bien documentés mais il existe de nombreuses réponses sur les forums de discussions.




511. https://docs.python.org/3/library/tkinter.tix.html
512. https://docs.python.org/3/library/tk.html
513. http://effbot.org/tkinterbook/listbox.htm
514. http://effbot.org/tkinterbook/listbox.htm
515. http://effbot.org/tkinterbook/listbox.htm
516. http://effbot.org/tkinterbook/listbox.htm#Tkinter.Listbox.itemconfig-method
517. https://docs.python.org/3/library/tkinter.ttk.html
518. https://docs.python.org/3/library/tk.html
519. https://docs.python.org/3.6/library/tkinter.ttk.html?highlight=combobox#tkinter.ttk.Combobox
520. https://docs.python.org/3/library/tkinter.ttk.html
521. https://docs.python.org/3/library/tkinter.tix.html

### Canevas

Pour dessiner, il faut utiliser un objet canevas, correspondant à la classe Canvas [522]. Pour la créer, il suffit d'écrire la ligne suivante :

```
ca = tkinter.Canvas ()
```

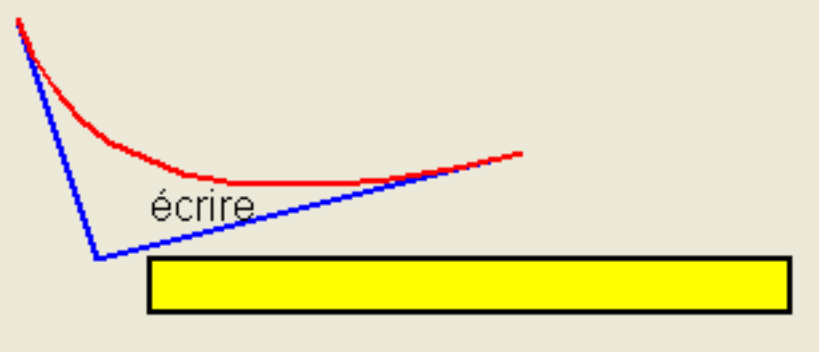
Pour modifier les dimensions de la zone de saisie à plusieurs lignes, on utilise l'instruction suivante :

```
# modifie les dimensions du canevas
# width <--> largeur en pixels
# height <--> hauteur en pixels
ca.config (width = 10, height = 5)
```

Cet objet permet de dessiner des lignes, des courbes, d'écrire du texte grâce aux méthodes create_line [523], create_rectangle [524], create_text [525].

```
  # dessine deux lignes du point 10,10 au point 40,100 et au point 200,60
  # de couleur bleue, d'épaisseur 2
ca.create_line (10,10,40,100, 200,60, fill = "blue", width = 2)
  # dessine une courbe du point 10,10 au point 200,60
  # de couleur rouge, d'épaisseur 2, c'est une courbe de Bézier
  # pour laquelle le point  40,100 sert d'assise
ca.create_line (10,10, 40,100, 200,60, smooth=1, fill = "red", width = 2)
  # dessine un rectangle plein de couleur jaune, de bord noir et d'épaisseur 2
ca.create_rectangle (300,100,60,120, fill = "gray", width = 2)
  # écrit du texte de couleur noire au point 80,80 et avec la police arial
ca.create_text (80,80, text = "écrire", fill = "black", font = "arial")
```

Visuellement, cela donne :




### Menus

Les menus apparaissent en haut des fenêtres. La plupart des applications arborent un menu commençant par *Fichier Edition Affichage...* Le paragraphe *Menu* (page 300) les décrit en détail.

### Méthodes communes

Nous avons vu que tous les objets présentés dans ce paragraphe possèdent une méthode `config` qui permet de définir l'état du widget (grisé ou normal) voire de la faire disparaître (voir paragraphe *Disposition des objets dans une fenêtre* (page 291)).

522. http://effbot.org/tkinterbook/canvas.htm
523. http://effbot.org/tkinterbook/canvas.htm#Tkinter.Canvas.create_line-method
524. http://effbot.org/tkinterbook/canvas.htm#Tkinter.Canvas.create_rectangle-method
525. http://effbot.org/tkinterbook/canvas.htm#Tkinter.Canvas.create_text-method

```
widget.config (state = tkinter.DISABLED) # grisé
widget.config (state = tkinter.NORMAL)   # aspect normal
```

Elle permet également de modifier le texte d'un objet, sa position, ... De nombreuses options sont communes à tous les objets et certaines sont spécifiques. L'aide associée à cette méthode n'est pas très explicite (par exemple `help(tkinter.Label.config)`). En fait, le constructeur et cette méthode ont les mêmes paramètres optionnels. Il est équivalent de préciser ces options lors de l'appel au constructeur :

```
l = tkinter.Label (text = "légende")
```

Ou de les modifier à l'aide de la méthode `config` :

```
l = tkinter.Label ()
l.config (text = "légende")
```

L'aide associée à la méthode codes{config} n'a pas évolué depuis la version 2.5 de *Python* :

```
Help on method configure in module tkinter:

configure(self, cnf=None, **kw) unbound tkinter.Label method
    Configure resources of a widget.

    The values for resources are specified as keyword
    arguments. To get an overview about
    the allowed keyword arguments call the method keys.
```

Tandis que l'aide associée au constructeur d'un object :

<<<

```
import tkinter
help(tkinter.Button.__init__)
```

>>>

```
    Help on function __init__ in module tkinter:

    __init__(self, master=None, cnf={}, **kw)
        Construct a button widget with the parent MASTER.

        STANDARD OPTIONS

            activebackground, activeforeground, anchor,
            background, bitmap, borderwidth, cursor,
            disabledforeground, font, foreground
            highlightbackground, highlightcolor,
            highlightthickness, image, justify,
            padx, pady, relief, repeatdelay,
            repeatinterval, takefocus, text,
            textvariable, underline, wraplength

        WIDGET-SPECIFIC OPTIONS

            command, compound, default, height,
            overrelief, state, width
```

Cette aide mentionne les options communes à tous les objets (ou widgets) et les options spécifiques à cet objet. Toutes ont une valeur par défaut qu'il est possible de changer soit dans le constructeur, soit par la méthode `config`. Quelques-unes ont été décrites, d'autres permettent de modifier entre autres la police avec laquelle est affiché le texte

de l'objet (option `font`), la couleur du fond (option `background`), l'alignement du texte, à gauche, à droite, centré (option `justify`), l'épaisseur du bord (option `borderwidth`), le fait qu'un objet reçoive le *focus* (voir paragraphe *Focus* (page 294)) après la pression de la touche tabulation (option `takefocus`).

### 9.1.3 Disposition des objets dans une fenêtre

#### Emplacements

Chacun des objets (ou widgets) présentés au paragraphe précédent possède trois méthodes qui permettent de déterminer sa position dans une fenêtre : pack [526], grid [527], place [528]. Les deux premières permettent de disposer les objets sans se soucier ni de leur dimension ni de leur position. La fenêtre gère cela automatiquement. La dernière place les objets dans une fenêtre à l'aide de coordonnées sans utiliser l'aide d'aucune grille. Dans une fenêtre, tous les objets doivent être placés avec la même méthode. Dans le cas contraire, les résultats risquent ne pas être ceux attendus.

#### Méthode pack

Cette méthode empile les objets les uns à la suite des autres. Par défaut, elle les empile les uns en dessous des autres. Par exemple, l'exemple suivant produit l'empilement des objets.

```
l = tkinter.Label (text = "première ligne")
l.pack ()
s = tkinter.Entry ()
s.pack ()
e = tkinter.Label (text = "seconde ligne")
e.pack ()
```

Les objets sont empilés à l'aide de la méthode `pack` les uns en dessous des autres pour la première image, les uns à droite des autres pour la seconde image.

On peut aussi les empiler les uns à droite des autres grâce à l'option `side`.

```
l = tkinter.Label (text = "première ligne")
l.pack (side = tkinter.RIGHT)
s = tkinter.Entry ()
s.pack (side = tkinter.RIGHT)
e = tkinter.Label (text = "seconde ligne")
e.pack (side = tkinter.RIGHT)
```

La méthode `pack` possède trois options :

- **`side`** [à choisir entre `tkinter.TOP` (valeur par défaut),] `tkinter.LEFT`, `tkinter.BOTTOM`, `tkinter.RIGHT`
- `expand` : égale à `True` ou `False` (valeur par défaut), si cette option est vraie, l'objet occupe tout l'espace.
- `fill` : égale à `None` (valeur par défaut), `X`, `Y`, `BOTH`, l'objet s'étend selon un axe (X ou Y ou les deux).

526. https://docs.python.org/3/library/tkinter.html#the-packer
527. http://effbot.org/tkinterbook/grid.htm
528. http://effbot.org/tkinterbook/place.htm

Il n'est pas toujours évident d'obtenir du premier coup le positionnement des objets souhaités au départ et il faut tâtonner pour y arriver. Lorsque un objet n'est plus nécessaire, il est possible de le faire disparaître en appelant la méthode pack_forget [529]. Le rappel de la méthode pack [530] le fera réapparaître mais rarement au même endroit.

```
s.pack_forget()    # disparition
s.pack()           # insertion à un autre endroit
```

### Méthode grid

La méthode grid [531] suppose que la fenêtre qui les contient est organisée selon une grille dont chaque case peut recevoir un objet. L'exemple suivant place trois objets dans les cases de coordonnées *(0,0)*, *(1,0)* et *(0,1)*.

```
l = tkinter.Label (text = "première ligne")
l.grid (column = 0, row = 0)
s = tkinter.Entry ()
s.grid (column = 0, row = 1)
e = tkinter.Label (text = "seconde ligne")
e.grid (column = 1, row = 0)
```

Les objets sont placés dans une grille à l'aide de la méthode `grid`. Une fois que chaque objet a reçu une position, à l'affichage, il ne sera pas tenu compte des lignes et colonnes vides.

La méthode grid [532] possède plusieurs options, en voici cinq :

- `column` : colonne dans laquelle sera placée l'objet.
- `columnspan` : nombre de colonnes que doit occuper l'objet.
- `row` : ligne dans laquelle sera placée l'objet.
- `rowspan` : nombre de lignes que doit occuper l'objet.
- `sticky` : indique ce qu'il faut faire lorsque la case est plus grande que l'objet qu'elle doit contenir. Par défaut, l'objet est centré mais il est possible d'aligner l'objet sur un ou plusieurs bords en précisant que `sticky="N"` ou `"S"` ou `"W"` ou `"E"`. Pour aligner l'objet sur un angle, il suffit de concaténer les deux lettres correspondant aux deux bords concernés. Il est aussi possible d'étendre l'objet d'un bord à l'autre en écrivant `sticky="N+S"` ou `sticky="E+W"`.

Enfin, comme pour la méthode pack [533], il existe une méthode grid_forget [534] qui permet de faire disparaître les objets.

```
s.grid_forget()  # disparition
```

### Méthode place

La méthode place [535] est sans doute la plus simple à comprendre puisqu'elle permet de placer chaque objet à une position définie par des coordonnées. Elle peut être utilisée en parallèle avec les méthodes place [536] et grid [537].

529. http://effbot.org/tkinterbook/pack.htm#Tkinter.Pack.pack_forget-method
530. http://effbot.org/tkinterbook/pack.htm#Tkinter.Pack.pack-method
531. http://effbot.org/tkinterbook/grid.htm
532. http://effbot.org/tkinterbook/grid.htm
533. https://docs.python.org/3/library/tkinter.html#the-packer
534. http://effbot.org/tkinterbook/grid.htm#Tkinter.Grid.grid_forget-method
535. http://effbot.org/tkinterbook/place.htm
536. http://effbot.org/tkinterbook/place.htm
537. http://effbot.org/tkinterbook/grid.htm

```
l = tkinter.Label(text="première ligne")
l.place (x=10, y=50)
```

La méthode place_forget [538] permet de faire disparaître un objet placer avec cette méthode. L'inconvénient de cette méthode survient lorsqu'on cherche à modifier l'emplacement d'un objet : il faut en général revoir les positions de tous les autres éléments de la fenêtre. On procède souvent par tâtonnement pour construire une fenêtre et disposer les objets. Ce travail est beaucoup plus long avec la méthode place [539].

### Sous-fenêtre

Les trois méthodes précédentes ne permettent pas toujours de placer les éléments comme on le désire. On souhaite parfois regrouper les objets dans des boîtes et placer celles-ci les unes par rapport aux autres. C'est aussi la seule façon de réutiliser un groupe de contrôle ou widgets dans plusieurs fenêtres sans avoir à dupliquer le code. La figure suivante montre deux objets regroupés dans un rectangle avec à sa gauche une zone de texte. Les boîtes sont des instances de la classe Frame [540].

Les deux premiers objets, une zone de texte au-dessus d'une zone de saisie, sont regroupés dans une boîte rectangle rouge, invisible à l'écran. A droite et centrée, une dernière zone de texte. Cet alignement est plus simple à réaliser en regroupant les deux premiers objets dans un object Frame [541]. Pour créer une boîte, il suffit d'écrire la ligne suivante :

```
f = tkinter.Frame ()
```

Ensuite, il faut pouvoir affecter un objet à cette boîte `f`. Pour cela, il suffit que `f` soit le premier paramètre du constructeur de l'objet créé :

```
l = tkinter.Label (f, text = "première ligne")
```

L'exemple qui suit correspond au code qui permet d'afficher la fenêtre de la figure ci-dessus.

```
f = tkinter.Frame ()
l = tkinter.Label (f, text = "première ligne")
l.pack ()                       # positionne l à l'intérieur de f
s = tkinter.Entry (f)
s.pack ()                       # positionne s à l'intérieur de f
f.pack (side = tkinter.LEFT)    # positionne f à l'intérieur
                                #   de la fenêtre principale
e = tkinter.Label (text = "seconde ligne")
e.pack_forget ()
e.pack (side = tkinter.RIGHT) # positionne e à l'intérieur
                                #   de la fenêtre principale
```

L'utilisation de ces blocs Frame [542] est pratique lorsque le même ensemble de contrôles apparaît dans plusieurs fenêtres différentes ou au sein de la même fenêtre. Cette possibilité est envisagée au paragraphe *Fenêtres personnalisées : utilisation des classes* (page 310).

538. http://effbot.org/tkinterbook/place.htm#Tkinter.Place.place_forget-method
539. http://effbot.org/tkinterbook/place.htm
540. http://effbot.org/tkinterbook/frame.htm
541. http://effbot.org/tkinterbook/frame.htm
542. http://effbot.org/tkinterbook/frame.htm

### 9.1.4 Evénements

#### Fenêtre principale

Tous les exemples des paragraphes précédents décrivent les différents objets disponibles et comment les disposer dans une fenêtre. Pour afficher cette fenêtre, il suffit d'ajouter au programme les deux lignes suivantes :

```
root = tkinter.Tk ()
#  ici, on trouve le code qui définit les objets
#  et leur positionnement
root.mainloop ()
```

La première ligne permet d'obtenir un identificateur relié à la fenêtre principale. La seconde ligne, outre le fait qu'elle affiche cette fenêtre, lance ce qu'on appelle une *boucle de messages*. Cette fonction récupère ou plutôt intercepte les événements comme un clic de souris, la pression d'une touche. Elle parcourt ensuite tous les objets qu'elle contient et regarde si l'un de ces objets est intéressé par cet événement. S'il est intéressé, cet objet prend l'événement et le traite. On peut revenir ensuite à la fonction `mainloop` qui attend à nouveau un événement. Cette fonction est définie par *tkinter*, il reste à lui indiquer quels événements un objet désire intercepter et ce qu'il est prévu de faire au cas où cet événement se produit.

#### Focus

Une fenêtre peut contenir plusieurs zones de saisie, toutes capables d'intercepter la pression d'une touche du clavier et d'ajouter la lettre correspondante à la zone de saisie. Or la seule qui ajoute effectivement une lettre à son contenu est celle qui a le focus [543]. La pression de la touche tabulation fait passer le focus d'un objet à l'autre. La figure ci-dessous montre un bouton qui a le focus. Lorsqu'on désire qu'un objet en particulier ait le focus, il suffit d'appeler la méthode focus_set [544].

Ce bouton est entouré d'un cercle noir en pointillé, il a le *focus*. Pour changer le *focus* :

```
e = tkinter.Entry(root)
e.pack()
e.focus_set()
```

#### Lancer une fonction lorsqu'un bouton est pressé

La plupart de temps, le seul événement qu'on souhaite attraper est la pression d'un bouton. Le code suivant permet de créer un bouton dont l'identificateur est `b`. Il a pour intitulé `fonction change_legende`. On définit ensuite une fonction `change_legende` qui change la légende de ce bouton. L'avant-dernière ligne permet d'associer au bouton `b` la fonction `change_legende` qui est alors appelée lorsque le bouton est pressé. La dernière ligne affiche la fenêtre principale et lance l'application.

```
import tkinter
root = tkinter.Tk ()
b = tkinter.Button (text = "fonction change_legende")
b.pack ()

def change_legende () :
    global b
    b.config (text = "nouvelle légende")
```

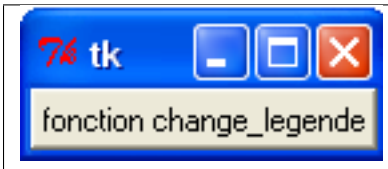
(suite sur la page suivante)

543. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
544. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

(suite de la page précédente)

```
b.config (command = change_legende)
root.mainloop ()
```

Lorsque le bouton `b` est pressé, on vérifie qu'il change bien de légende. La première fenêtre est celle qui apparaît lorsque le programme est lancé. Comme le bouton change de légende la première fois qu'il est pressé, l'apparence de la fenêtre change aussi, ce que montre la seconde image.



L'exemple précédent associe une fonction au bouton. Lorsque l'interface devient conséquente, la lisibilité du programme en est réduite car le nombre de fonctions associées à des boutons augmentent rapidement. Pour éviter cela, il est possible d'associer au bouton une méthode de classe comme le suggère l'exemple du paragraphe *Fenêtres personnalisées : utilisation des classes* (page 310). C'est même recommandé.

### Associer n'importe quel événement à un objet

Le paragraphe précédent s'est intéressé à l'association entre une fonction et la pression d'un bouton mais il est possible de faire en sorte que le programme exécute une fonction au moindre déplacement de la souris, à la pression d'une touche. Il est possible d'associer une fonction au moindre événement susceptible d'être intercepté par l'interface graphique.

On peut regrouper les événements en deux classes. La première classe regroupe les événements provenant du clavier ou de la souris. Ce sont des événements en quelque sorte *bruts*. La seconde classe regroupe des événements produit par des objets tels qu'un bouton. En effet, lorsque celui-ci détecte le clic du bouton droit de la souris, il construit un événement *ńăpression du boutonă"* et c'est celui-ci qui va déclencher l'exécution d'une fonction. Il n'est pas souvent nécessaire de revenir aux événements *bruts* car les objets proposent d'eux-mêmes de pouvoir attacher des fonctions à des événements liés à leur apparence.

Toutefois, pour un jeu par exemple, il est parfois nécessaire d'avoir accès au mouvement de la souris et il faut revenir aux événements *bruts*. Un événement est décrit par la classe Event [545] dont les attributs listés par la table suivante décrivent l'événement qui sera la plupart du temps la pression d'une touche du clavier ou le mouvement de la souris.

| | |
|---|---|
| `char` | Lorsqu'une touche a été pressée, cet attribut contient son code, il ne tient pas compte des touches dites muettes comme les touches `shift`, `ctrl`, `alt`. Il tient pas compte non plus des touches `return` ou `suppr`. |
| `keysym` | Lorsqu'une touche a été pressée, cet attribut contient son code, quelque soit la touche, muette ou non. |
| `num` | Contient un identificateur de l'objet ayant reçu l'événement. |
| x,y | Coordonnées relatives de la souris par rapport au coin supérieur gauche de l'objet ayant reçu l'événement. |
| `x_root, y_root` | Coordonnées absolues de la souris par rapport au coin supérieur gauche de l'écran. |
| `widget` | Identifiant permettant d'accéder à l'objet ayant reçu l'événement. |

La liste complète est accessible avec l'instruction suivante :

<<<

545. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm#events

```
import tkinter
help(tkinter.Event)
```

>>>

```
    Help on class Event in module tkinter:

    class Event(builtins.object)
     |  Container for the properties of an event.
     |
     |  Instances of this type are generated if one of the following events occurs:
     |
     |  KeyPress, KeyRelease - for keyboard events
     |  ButtonPress, ButtonRelease, Motion, Enter, Leave, MouseWheel - for mouse␣
↪events
     |  Visibility, Unmap, Map, Expose, FocusIn, FocusOut, Circulate,
     |  Colormap, Gravity, Reparent, Property, Destroy, Activate,
     |  Deactivate - for window events.
     |
     |  If a callback function for one of these events is registered
     |  using bind, bind_all, bind_class, or tag_bind, the callback is
     |  called with an Event as first argument. It will have the
     |  following attributes (in braces are the event types for which
     |  the attribute is valid):
     |
     |      serial - serial number of event
     |  num - mouse button pressed (ButtonPress, ButtonRelease)
     |  focus - whether the window has the focus (Enter, Leave)
     |  height - height of the exposed window (Configure, Expose)
     |  width - width of the exposed window (Configure, Expose)
     |  keycode - keycode of the pressed key (KeyPress, KeyRelease)
     |  state - state of the event as a number (ButtonPress, ButtonRelease,
     |                          Enter, KeyPress, KeyRelease,
     |                          Leave, Motion)
     |  state - state as a string (Visibility)
     |  time - when the event occurred
     |  x - x-position of the mouse
     |  y - y-position of the mouse
     |  x_root - x-position of the mouse on the screen
     |           (ButtonPress, ButtonRelease, KeyPress, KeyRelease, Motion)
     |  y_root - y-position of the mouse on the screen
     |           (ButtonPress, ButtonRelease, KeyPress, KeyRelease, Motion)
     |  char - pressed character (KeyPress, KeyRelease)
     |  send_event - see X/Windows documentation
     |  keysym - keysym of the event as a string (KeyPress, KeyRelease)
     |  keysym_num - keysym of the event as a number (KeyPress, KeyRelease)
     |  type - type of the event as a number
     |  widget - widget in which the event occurred
     |  delta - delta of wheel movement (MouseWheel)
     |
     |  Methods defined here:
     |
     |  __repr__(self)
     |      Return repr(self).
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
```

(suite sur la page suivante)

(suite de la page précédente)

```
    |  __dict__
    |      dictionary for instance variables (if defined)
    |
    |  __weakref__
    |      list of weak references to the object (if defined)
```

La méthode bind[546] permet d'exécuter une fonction lorsqu'un certain événement donné est intercepté par un objet donné. La fonction exécutée accepte un seul paramètre de type Event[547] qui est l'événement qui l'a déclenchée. Cette méthode a pour syntaxe :

```
w.bind(ev, fonction)
```

`w` est l'identificateur de l'objet devant intercepter l'événement désigné par la chaîne de caractères `ev` dont les valeurs possibles sont décrites ci-dessous. `fonction` est la fonction qui est appelée lorsque l'événement survient. Cette fonction ne prend qu'un paramètre de type Event[548].

| | |
|---|---|
| `<Key>` | Intercepter la pression de n'importe quelle touche du clavier. |
| `<Button-i>` | Intercepter la pression d'un bouton de la souris. `i` doit être remplacé par 1,2,3. |
| `<ButtonRelease-i>` | Intercepter le relâchement d'un bouton de la souris. `i` doit être remplacé par 1,2,3. |
| `<Double-Button-i>` | Intercepter la double pression d'un bouton de la souris. `i` doit être remplacé par 1,2,3. |
| `<Motion>` | Intercepter le mouvement de la souris, dès que le curseur bouge, la fonction liée à l'événement est appelée. |
| `<Enter>` | Intercepter un événement correspondant au fait que le curseur de la souris entre la zone graphique de l'objet. |
| `<Leave>` | Intercepter un événement correspondant au fait que le curseur de la souris sorte la zone graphique de l'objet. |

La liste complète est accessible avec l'instruction suivante :

<<<

```
import tkinter
help(tkinter.Label.bind)
```

>>>

```
    Help on function bind in module tkinter:

    bind(self, sequence=None, func=None, add=None)
        Bind to this widget at event SEQUENCE a call to function FUNC.

        SEQUENCE is a string of concatenated event
        patterns. An event pattern is of the form
        <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one
        of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4,
        Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3,
        B3, Alt, Button4, B4, Double, Button5, B5 Triple,
        Mod1, M1. TYPE is one of Activate, Enter, Map,
        ButtonPress, Button, Expose, Motion, ButtonRelease
```

(suite sur la page suivante)

546. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
547. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm#events
548. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm#events

(suite de la page précédente)

```
        FocusIn, MouseWheel, Circulate, FocusOut, Property,
        Colormap, Gravity Reparent, Configure, KeyPress, Key,
        Unmap, Deactivate, KeyRelease Visibility, Destroy,
        Leave and DETAIL is the button number for ButtonPress,
        ButtonRelease and DETAIL is the Keysym for KeyPress and
        KeyRelease. Examples are
        <Control-Button-1> for pressing Control and mouse button 1 or
        <Alt-A> for pressing A and the Alt key (KeyPress can be omitted).
        An event pattern can also be a virtual event of the form
        <<AString>> where AString can be arbitrary. This
        event can be generated by event_generate.
        If events are concatenated they must appear shortly
        after each other.

        FUNC will be called if the event sequence occurs with an
        instance of Event as argument. If the return value of FUNC is
        "break" no further bound function is invoked.

        An additional boolean parameter ADD specifies whether FUNC will
        be called additionally to the other bound function or whether
        it will replace the previous function.

        Bind will return an identifier to allow deletion of the bound function with
        unbind without memory leak.

        If FUNC or SEQUENCE is omitted the bound function or list
        of bound events are returned.
```

L'exemple suivant utilise la méthode bind[549] pour que le seul bouton de la fenêtre intercepte toute pression d'une touche, tout mouvement et toute pression du premier bouton de la souris lorsque le curseur est au dessus de la zone graphique du bouton.

```
import tkinter
root = tkinter.Tk()
b = tkinter.Button(text="appuyer sur une touche")
b.pack()

def affiche_touche_pressee (evt) :
    print("-------------------------- touche pressee")
    print("evt.char = ", evt.char)
    print("evt.keysym = ", evt.keysym)
    print("evt.num = ", evt.num)
    print("evt.x,evt.y = ", evt.x, ",", evt.y)
    print("evt.x_root,evt.y_root = ", evt.x_root, ",", evt.y_root)
    print("evt.widget = ", evt.widget)

b.bind ("<Key>", affiche_touche_pressee)
b.bind ("<Button-1>", affiche_touche_pressee)
b.bind ("<Motion>", affiche_touche_pressee)
b.focus_set ()

root.mainloop ()
```

Ci-dessous, la fenêtre créée par ce programme :

549. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

Et l'affichage qui en résulte :

```
evt.char =  ??
evt.keysym =  ??
evt.num =  1
evt.x,evt.y =  105 , 13
evt.x_root,evt.y_root =
               292 , 239
evt.widget =  .9261224

evt.char =
evt.keysym =  Return
evt.num =  ??
evt.x,evt.y =  105 , 13
evt.x_root,evt.y_root =
               292 , 239
evt.widget =  .9261224
```

La pression d'une touche déclenche l'affichage des caractéristiques de l'événement. La seconde colonne correspond à la pression du premier bouton de la souris. La dernière colonne correspond à la pression de la touche `Return`.

L'avant dernière ligne du programme fait intervenir la méthode `focus_set`. Elle stipule que le bouton doit recevoir le *focus*. C'est-à-dire que cet objet est celui qui peut intercepter les événements liés au clavier. Sans cette instruction, cet objet n'y a pas accès, ces événements sont dirigés vers la fenêtre principale qui ne s'en soucie pas.

Les messages d'erreur liés aux événements ne sont pas forcément très explicites. Ainsi l'instruction suivante adresse un événement inexistant.

```
b.bind ("<button-1>", affiche_touche_pressee)
```

Lors de l'exécution, le programme déclenche la succession d'exceptions suivantes qui signifie que l'événement `<button-1>` n'existe pas.

```
Traceback (most recent call last):
  File "exemple_bind.py", line 17, in ?
    b.bind ("<button-1>", affiche_touche_pressee)
  File "c:\python26\lib\lib-tk\tkinter.py", line 933, in bind
    return self._bind(('bind', self._w), sequence, func, add)
  File "c:\python26\lib\lib-tk\tkinter.py", line 888, in _bind
    self.tk.call(what + (sequence, cmd))
_tkinter.TclError: bad event type or keysym "button"
```

Il arrive parfois qu'un événement ne doive pas être associé à un seul objet mais à tous ceux que la fenêtre contient. C'est l'objectif de la méthode bind_all[550]. Sa syntaxe est exactement la même que la méthode bind[551].

```
b.bind_all ("<Button-1>", affiche_touche_pressee)
```

On utilise peu cette fonction, on préfère construire des objets propres à un programme comme suggéré au paragraphe *Contrôles personnalisés : utilisation des classes* (page 309).

De la même manière qu'il est possible d'associer un événement à un objet d'une fenêtre, il est possible d'effectuer

550. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
551. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

l'opération inverse qui consiste à supprimer cette association. La méthode unbind [552] désactive un événement associé à un objet. La méthode unbind_all [553] désactive un événement associé pour tous les objets d'une fenêtre.

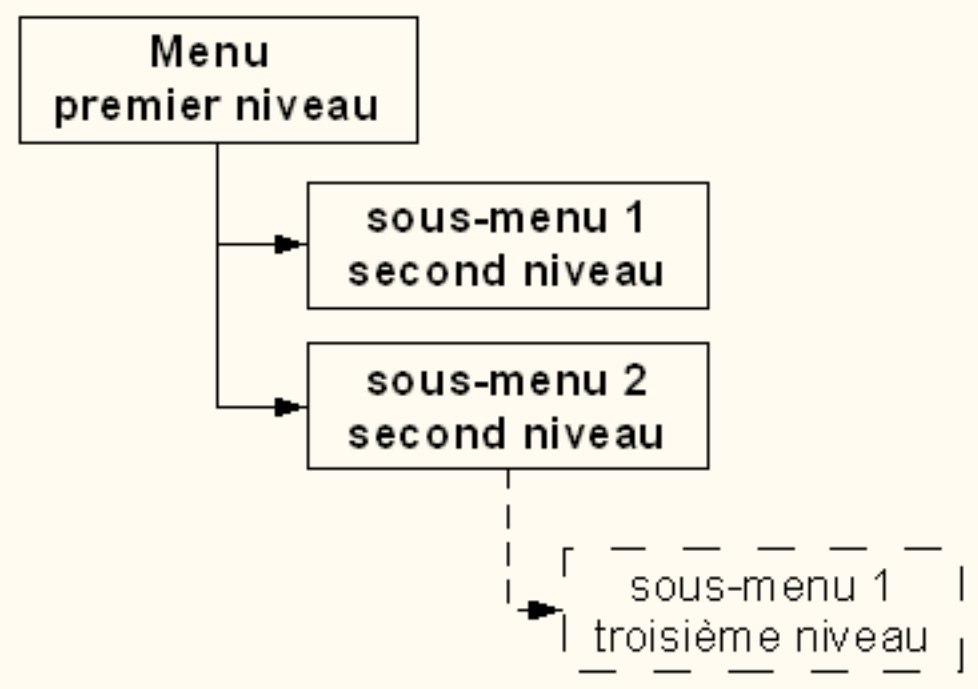
```
w.unbind(ev)
w.unbind_all(ev)
```

`w` est l'identificateur de l'objet interceptant l'événement désigné par la chaîne de caractères `ev`. Après l'appel à la méthode *unbind*, l'événement n'est plus intercepté par l'objet `w`. Après l'appel à la méthode *unbind_all*, l'événement n'est plus intercepté par aucun objet. end{xsyntax}

Il est possible de définir des événements propres aux programmes. Ceux-ci ne sont générés par aucun périphérique mais explicitement par le programme lui-même. Ce mécanisme est presque toujours couplé à l'utilisation de threads. Le paragraphe *Interface graphique* (page 266) illustre ce principe à l'aide d'un exemple à base de thread. Le paragraphe *Communiquer un résultat par message* (page 314) propose un exemple plus simple.

### Menu

Les menus fonctionnent de la même manière que les boutons. Chaque intitulé du menu est relié à une fonction qui sera exécutée à la condition que l'utilisateur sélectionne cet intitulé. L'objet Menu [554] ne désigne pas le menu dans son ensemble mais seulement un niveau. Par exemple, le menu présenté par la figure suivante est en fait un assemblage de trois menus auquel on pourrait ajouter d'autres sous-menus.




La représentation d'un menu tient plus d'un graphe que d'une liste. Chaque intitulé du menu peut être connecté à une fonction ou être le point d'entrée d'un nouveau sous-menu. Pour créer un menu ou un sous-menu, il suffit de créer un objet de type Menu [555] :

```
m = tkinter.Menu ()
```

Ce menu peut être le menu principal d'une fenêtre auquel cas, il suffit de préciser à la fenêtre en question que son menu est le suivant :

552. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
553. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm
554. http://effbot.org/tkinterbook/menu.htm
555. http://effbot.org/tkinterbook/menu.htm

```
root.config (menu = m)
```

`root` est ici la fenêtre principale mais ce pourrait être également une fenêtre de type Toplevel [556] Ce menu peut aussi être le sous-menu associé à un intitulé d'un menu existant. La méthode add_cascade [557] permet d'ajouter un sous-menu associé à un label :

```
mainmenu  = tkinter.Menu ()
msousmenu = tkinter.Menu ()
mainmenu.add_cascade (label = "sous-menu 1", menu = msousmenu)
```

En revanche, si on souhaite affecter une fonction à un menu, on utilisera la méthode add_command [558]

```
def fonction1 () :
    ....
m = tkinter.Menu ()
mainmenu.add_command (label = "fonction 1", command = fonction1)
```

L'exemple suivant regroupe les fonctionnalités présentées ci-dessus.

```
import tkinter
root = tkinter.Tk()

e = tkinter.Text(width = 50, height = 10)
e.pack()

m = tkinter.Menu(root)

sm1 = tkinter.Menu(root)
sm2 = tkinter.Menu(root)

m.add_cascade (label = "sous-menu 1", menu = sm1)
m.add_cascade (label = "sous-menu 2", menu = sm2)

nb = 0

def affiche():
    print("fonction affiche")

def calcul():
    print("fonction calcul ", 3 * 4)

def ajoute_bouton () :
    global nb
    nb += 1
    b = tkinter.Button (text = "bouton " + str (nb))
    b.pack()

sm1.add_command (label = "affiche",       command = affiche)
sm1.add_command (label = "calcul",        command = calcul)
sm2.add_command (label = "ajoute_bouton", command = ajoute_bouton)
sm2.add_command (label = "fin",           command = root.destroy)

root.config(menu = m, width = 200)
root.title("essai de menu")
root.mainloop()
```

556. http://effbot.org/tkinterbook/Toplevel.htm
557. http://effbot.org/tkinterbook/menu.htm#Tkinter.Menu.add_cascade-method
558. http://effbot.org/tkinterbook/menu.htm#Tkinter.Menu.add_command-method

Ce qui donne :

Chaque intitulé d'un menu est ajouté en fin de liste, il est possible d'en supprimer certains à partir de leur position avec la méthode delete [559].

```
m = tkinter.Menu ()
m.add_command (...)
m.delete (1, 2) # supprime le second intitulé
                # supprime les intitulés compris entre 1 et 2 exclu
```

### Fonctions prédéfinies

Il est possible de détruire la fenêtre principale, ce qui mettra fin au programme si celui-ci ne prévoit rien après la fonction `mainloop`. La destruction de la fenêtre s'effectue par la méthode `destroy`. Le programme suivant crée une fenêtre avec un seul bouton qui, s'il est pressé, mettra fin à l'application.

```
import tkinter
root = tkinter.Tk()
tkinter.Button (text = "fin", command = root.destroy).pack ()
root.mainloop()
```

La table suivante regroupe les fonctions les plus utilisées. Celles-ci s'applique à une fenêtre de type Toplevel [560] qui est aussi le type de la fenêtre principale.

| | |
|---|---|
| destroy() [561] | Détruit la fenêtre. |
| deiconify() [562] | La fenêtre reprend une taille normale. |
| geometry(s) [563] | Modifie la taille de la fenêtre. `s` est une chaîne de caractères de type `"wxh±x±y"`. `w` et `h` sont la largeur et la hauteur. `x` et `y` sont la position du coin supérieur haut à l'écran. |
| iconify() [564] | La fenêtre se réduit à un icône. |
| resizable(w,h) [565] | Spécifie si la fenêtre peut changer de taille. `w` et `h` sont des booléens. |
| title(s) [566] | Change le titre de la fenêtre, `s` est une chaîne de caractères. |
| withdraw() [567] | Fait disparaître la fenêtre. La fonction inverse est deiconify() [568]. |

559. http://effbot.org/tkinterbook/menu.htm#Tkinter.Menu.delete-method
560. http://effbot.org/tkinterbook/Toplevel.htm

### D'autres fenêtres et contrôles

*tkinter* ne propose pas beaucoup de *widgets*, pas autant que la liste étendue qu'on trouve dans la plupart des applications. Deux extensions complètent cette liste ttk [569] et tix [570]. On trouve notamment :

- Combobox [571]
- Notebook [572]
- Progressbar [573]
- Treeview [574]

tix [575] propose des widgets un peu plus complexes :

- DirTree [576]
- FileSelectBox [577]

Cette liste n'est pas exhaustive.

### Créer une seconde boîte de dialogues

Lorsqu'un programme doit utiliser plusieurs fenêtres et non pas une seule, l'emploi de l'objet Toplevel [578] est inévitable. L'instruction `root = tkinter.Tk()` crée la fenêtre principale, l'instruction `win = tkinter.Toplevel()` crée une seconde fenêtre qui fonctionne exactement comme la fenêtre principale puisqu'elle dispose elle aussi d'une boucle de messages via la méthode `mainloop`.

```
import tkinter
win = tkinter.Toplevel()
win.mainloop()
```

Un cas d'utilisation simple est par exemple un bouton pressé qui fait apparaître une fenêtre permettant de sélectionner un fichier, cette seconde fenêtre sera un objet Toplevel [579]. Il n'est pas nécessaire de s'étendre plus sur cet objet, son comportement est identique à celui de la fenêtre principale, les fonctions décrites au paragraphe *Fonctions prédéfinies* (page 302) s'appliquent également aux objets Toplevel [580]. Il reste néanmoins à préciser un dernier point. Tous les objets précédemment décrits au paragraphe *Les objets* (page 281) doivent inclure un paramètre supplémentaire dans leur constructeur pour signifier qu'ils appartiennent à un objet Toplevel [581] et non à la fenêtre principale. Par exemple, pour créer une zone de texte, la syntaxe est la suivante :

```
# zone_texte appartient à la fenêtre principale
zone_texte = tkinter.Label (text = "premier texte")
```

Pour l'inclure à une fenêtre Toplevel [582], cette syntaxe devient :

561. http://effbot.org/tkinterbook/Toplevel.htm
562. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.deiconify-method
563. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.geometry-method
564. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.iconify-method
565. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.resizable-method
566. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.title-method
567. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.withdraw-method
568. http://effbot.org/tkinterbook/wm.htm#Tkinter.Wm.deiconify-method
569. https://docs.python.org/3/library/tkinter.ttk.html
570. https://docs.python.org/3/library/tkinter.tix.html
571. https://docs.python.org/3/library/tkinter.ttk.html#ttk-combobox
572. https://docs.python.org/3/library/tkinter.ttk.html#ttk-notebook
573. https://docs.python.org/3/library/tkinter.ttk.html#ttk-progressbar
574. https://docs.python.org/3/library/tkinter.ttk.html#treeview
575. https://docs.python.org/3/library/tkinter.tix.html
576. https://docs.python.org/3/library/tkinter.tix.html#tkinter.tix.DirTree
577. https://docs.python.org/3/library/tkinter.tix.html#tkinter.tix.FileSelectBox
578. http://effbot.org/tkinterbook/Toplevel.htm
579. http://effbot.org/tkinterbook/Toplevel.htm
580. http://effbot.org/tkinterbook/Toplevel.htm
581. http://effbot.org/tkinterbook/Toplevel.htm
582. http://effbot.org/tkinterbook/Toplevel.htm

```
# zone_texte appartient à la fenêtre top
top = tkinter.Toplevel ()
zone_texte = tkinter.Label (top, text = "premier texte")
```

Lors de la définition de chaque objet ou *widget*, si le premier paramètre est de type Toplevel [583], alors ce paramètre sera affecté à la fenêtre passée en premier argument et non à la fenêtre principale. Ce principe est le même que celui de la sous-fenêtre Frame [584] (voir paragraphe *Sous-fenêtre* (page 293)). La seule différence provient du fait que l'objet Toplevel [585] est une fenêtre autonome qui peut attendre un message grâce à la méthode `mainloop`, ce n'est pas le cas de l'objet Frame [586].

Toutefois, il est possible d'afficher plusieurs fenêtres Toplevel [587] simultanément. Le programme suivant en est un exemple :

```
import tkinter

class nouvelle_fenetre :
    resultat = []
    def top (self) :
        sec = tkinter.Toplevel ()
        tkinter.Label (sec, text="entrer quelque chose").pack ()
        saisie = tkinter.Entry (sec)
        saisie.pack()
        tkinter.Button (sec, text = "valider", command = sec.quit).pack ()
        sec.mainloop ()
        nouvelle_fenetre.resultat.append ( saisie.get () )
        sec.destroy ()

root = tkinter.Tk() #fenetre principale
a = tkinter.Button (text    = "fenêtre Toplevel",
                    command = nouvelle_fenetre ().top)
a.pack()
root.mainloop()

for a in nouvelle_fenetre.resultat :
    print("contenu ", a)
```

### Fenêtres standard

Le module tix [588] propose une fenêtre de sélection de fichiers identique à celle de la figure suivante. tkinter [589] a l'avantage d'être simple et ne nécessite pas un long apprentissage pour le maîtriser mais il est limité. Pour ce type de fenêtres qu'on retrouve dans la plupart des programmes, il existe presque toujours des solutions toutes faites, via le module tix [590] par exemple. On trouve également de nombreux programmes sur Internet par le biais de moteurs de recherche. Le programme ci-dessous affiche une fenêtre qui permet de sélectionner un fichier.

```
# -*- coding: utf-8 -*-
"""module contenant une boîte de dialogue permettant
de sélectionner un fichier ou un répertoire,
il utilise l'interface Tkinter"""
```

(suite sur la page suivante)

583. http://effbot.org/tkinterbook/Toplevel.htm
584. http://effbot.org/tkinterbook/frame.htm
585. http://effbot.org/tkinterbook/Toplevel.htm
586. http://effbot.org/tkinterbook/frame.htm
587. http://effbot.org/tkinterbook/Toplevel.htm
588. https://docs.python.org/3/library/tkinter.tix.html
589. https://docs.python.org/3/library/tk.html
590. https://docs.python.org/3/library/tkinter.tix.html

(suite de la page précédente)

```
import tkinter
import os.path
import os

class FileSelection(object) :
    """classe permettant de sélectionner un fichier
    ou un répertoire à travers une boîte de dialogue"""

    def __init__(self, parent, titre = "Sélection de fichier", \
                    chemin = None, file = True, exist= True) :
        """
        initialise la classe

        @param      parent          parent
        @param      titre           titre de la fenêtre
        @param      chemin          fichier ou répertoire par défaut
        @param      file            True : fichier, False : répertoire
        @param      exist           True : le répertoire ou le fichier
                                           sélectionné doit exister"""
        self.parent = parent
        self.titre  = titre
        self.chemin = chemin
        self.file   = file
        self.exist  = exist

        if self.chemin is None:
            self.chemin = os.getcwd()

    def get_list(self) :
        """retourne la liste des fichiers et des répertoires(2 listes),
        répertoires seulement et [] si self.file == False"""
        if os.path.isdir(self.chemin):
            listf    = os.listdir(self.chemin)
        else :
            ch, fi   = os.path.split(self.chemin)
            listf    = os.listdir(ch)

        lifile  = []
        lidir   = []
        for l in listf:
            if os.path.isdir(self.chemin + "\\" + l) :
                lidir.append(l)
            elif self.file:
                lifile.append(l)

        lidir.sort()
        lifile.sort()
        return lidir, lifile

    def run(self) :
        """lance la boîte de dialogue et retourne la chaîne sélectionnée"""
        if self.parent is None:
            top         = tkinter.Toplevel()
            top.wm_title(self.titre)
        else:
            top = self.parent
        self.resultat = False
```

(suite sur la page suivante)

(suite de la page précédente)

```
        fli = tkinter.Frame(top)
        scrollbar = tkinter.Scrollbar(fli)
        li = tkinter.Listbox(fli, width = 120, height = 15, \
                             yscrollcommand = scrollbar.set)
        scrollbar.config(command = li.yview)
        ch      = tkinter.Entry(top, width = 120)
        f       = tkinter.Frame(top)
        prec    = tkinter.Button(f, text = "Précédent")
        suiv    = tkinter.Button(f, text = "Entre")
        annul   = tkinter.Button(f, text = "Annuler")
        ok      = tkinter.Button(f, text = "Ok")

        prec.grid(column = 0, row = 0)
        suiv.grid(column = 1, row = 0)
        annul.grid(column = 3, row = 0)
        ok.grid(column = 4, row = 0)
        li.pack(side = tkinter.LEFT)
        scrollbar.pack(side = tkinter.RIGHT, fill = tkinter.Y)
        fli.pack()
        ch.pack()
        f.pack()

        def update_chemin() :
            """mise à jour du chemin dans la boîte de dialogue"""
            s = ch.get()
            ch.delete(0, len(s))
            ch.insert(0, self.chemin)

        def update_list() :
            """mise à jour de la liste des fichiers et répertoires
            à partir de la chaîne dans la boîte de dialogue"""
            self.chemin     = ch.get()
            lidir, lifile   = self.get_list()
            li.delete(0, tkinter.END)
            if len(lidir) > 0 :
                for l in lidir:
                    li.insert(tkinter.END, "+ "+ l)
            if len(lifile) > 0:
                for l in lifile:
                    li.insert(tkinter.END, "  "+ l)

        def precedent() :
            """passe au répertoire précédent"""
            if os.path.isdir(self.chemin) :
                ch, last    = os.path.split(self.chemin)
                self.chemin = ch
            else :
                ch, last    = os.path.split(self.chemin)
                ch, last    = os.path.split(ch)
                self.chemin = ch
            update_chemin()
            update_list()

        def suivant() :
            """rentre dans un répertoire"""
            sel = ch.get()
```

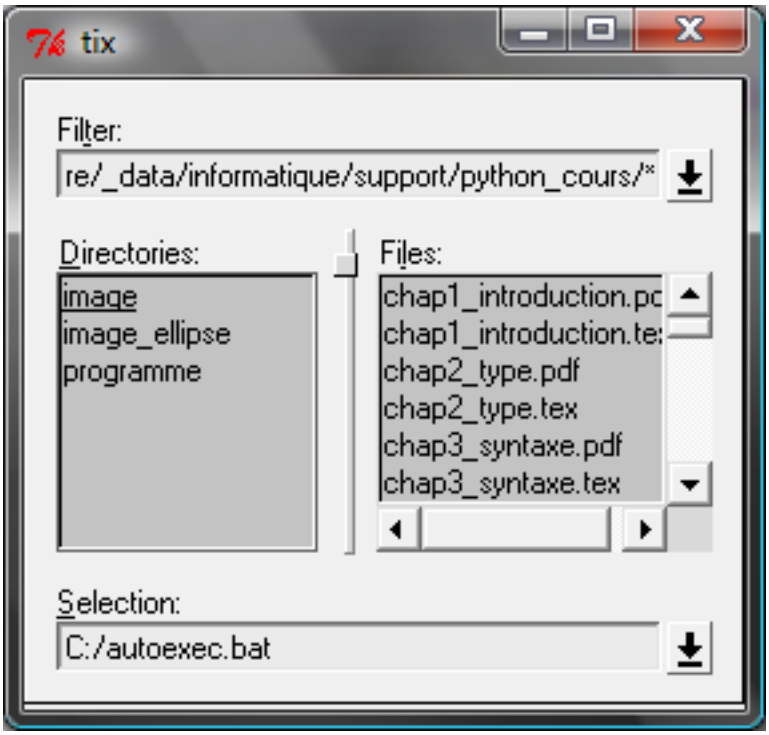
(suite sur la page suivante)

(suite de la page précédente)

```
            if os.path.isdir(sel) :
                self.chemin = sel
                update_chemin()
                update_list()

        def update_sel() :
            """mise à jour de la chaîne de caractères
            dans la boîte de dialogue à partir de la ligne
            sélectionnée dans la liste"""
            li.after(200, update_sel)
            sel = li.curselection()
            if len(sel) == 1 :
                t = li.get(sel [0])
                c = self.chemin + "\\" +  t [2:len(t)]
                s = ch.get()
                ch.delete(0, len(s))
                ch.insert(0, c)

        def annuler() :
            """annule la recherche"""
            self.resultat = False
            top.destroy()
            top.quit()

        def accepter() :
            """accepte le résultat"""
            self.resultat    = True
            self.chemin = ch.get()
            top.destroy()
            top.quit()

        prec.config(command = precedent)
        suiv.config(command = suivant)
        annul.config(command = annuler)
        ok.config(command = accepter)

        update_chemin()
        update_list()
        update_sel()
        ch.focus_set()

        if self.parent is None:
            top.mainloop()

if __name__ == "__main__" :

    def run(root) :
        r = FileSelection(root, "sélection d'un fichier", "c:\\")
        s = r.run()
        return r

    root = tkinter.Tk()
    win = run(root)
    root.mainloop()
    print("fichier sélectionné ", win.chemin)
```

Il faut comparer ce programme à celui qu'on écrirait avec l'extension tix [591] :

```
import tkinter.tix as tix
root = tix.Tk ()

o = tix.FileSelectBox (root)
o.pack ()

def print_file () :
    print(o.cget ("value"))

b = tix.Button (root, text = "print")
b.config (command = print_file)
b.pack ()

root.mainloop ()
```




### 9.1.5 Constructions classiques

L'objectif des paragraphes qui suivent est d'introduire quelques schémas de construction d'interfaces qui reviennent fréquemment. La première règle de programmation qu'il est préférable de suivre est d'isoler la partie interface du reste du programme. La gestion événementielle a pour défaut parfois de disséminer un traitement, un calcul à plusieurs endroits de l'interface. C'est le cas par exemple de longs calculs dont on souhaite connaître l'avancée. Le calcul est lancé par la pression d'un bouton puis son déroulement est ńäespionnéă" par un événement régulier comme un compte à rebours.

Le principal problème des interfaces survient lors du traitement d'un événement : pendant ce temps, l'interface n'est pas réactive et ne réagit plus aux autres événements jusqu'à ce que le traitement de l'événement en cours soit terminé. Pour contourner ce problème, il est possible soit de découper un calcul en petites fonctions chacune très rapide, cela suppose que ce calcul sera mené à bien par une succession d'événements. Il est également possible de lancer un thread, principe décrit au paragraphe *Interface graphique* (page 266).

591. https://docs.python.org/3/library/tkinter.tix.html

C'est pourquoi la première règle est de bien scinder interface et calculs scientifiques de façon à pouvoir rendre le programme plus lisible et ainsi être en mesure d'isoler plus rapidement la source d'une erreur. Les paragraphes qui suivent présentent quelques aspects récurrents qu'il est parfois utile d'avoir en tête avant de se lancer.

### Compte à rebours

Il est possible de demander à un objet d'appeler une fonction après un certains laps de temps exprimé un millisecondes. Le programme suivant crée un objet de type Label [592]. Il contient une fonction qui change son contenu et lui affecte un compte à rebours qui impose à l'objet de rappeler cette fonction 1000 millisecondes plus tard. Le résultat est un programme qui crée la fenêtre ci-dessous et change son contenu toutes les secondes.

```
import tkinter
root = tkinter.Tk()
l = tkinter.Label(text = "0 secondes")
l.pack()
sec = 0
id = None

def change_legende() :
    global l
    global sec
    global id
    sec += 1
    l.config(text = "%d secondes" % sec)
    id = l.after(1000, change_legende)

l.after(1000, change_legende)

root.mainloop()
```

L'intitulé de l'objet `Label` change toutes les secondes.

La méthode after [593] retourne un entier permettant d'identifier le compte à rebours qu'il est possible d'interrompre en utilisant la méthode after_cancel [594]. Dans l'exemple précédent, il faudrait utiliser l'instruction suivante :

```
l.after_cancel (id)
```

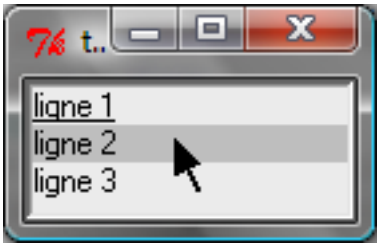
### Contrôles personnalisés : utilisation des classes

On peut personnifier un contrôle. Par exemple, on peut mettre en évidence l'intitulé d'une liste sous le curseur de la souris. Le moyen le plus simple est de créer une nouvelle classe qui se substituera au classique ListBox [595]. Il suffit que cette nouvelle classe hérite de ListBox [596] en prenant soin de lui donner un constructeur reprenant les mêmes paramètres que celui de la classe ListBox [597]. De cette façon, il suffit de remplacer ListBox [598] par `MaListbox` pour changer l'apparence d'une liste.

592. http://effbot.org/tkinterbook/label.htm
593. http://stackoverflow.com/questions/37748729/how-to-use-tkinter-after-method
594. http://effbot.org/tkinterbook/widget.htm
595. http://effbot.org/tkinterbook/listbox.htm
596. http://effbot.org/tkinterbook/listbox.htm
597. http://effbot.org/tkinterbook/listbox.htm
598. http://effbot.org/tkinterbook/listbox.htm

```
import tkinter

class MaListbox(tkinter.Listbox):
    def __init__(self, master = None, cnf=None, **kw):
        if cnf is None:
            cnf = {}
        tkinter.Listbox.__init__(self, master, cnf, **kw)
        self.bind("<Motion>", self.mouvement)
        self.pos = None  # mémoire l'ancienne position du curseur
    def mouvement(self, ev):
        pos = self.nearest(ev.y)  # nouvelle position du curseur
        if pos < 0 or pos >= self.size():
            return
        if self.pos != pos:
            if self.pos is not None:
                self.itemconfig(self.pos, bg='')
            self.itemconfigure(pos, bg='gray')
            self.pos = pos

root = tkinter.Tk()
b = MaListbox()
b.insert("end", "ligne 1")
b.insert("end", "ligne 2")
b.insert("end", "ligne 3")
b.pack()
b.focus_set()
root.mainloop()
```

Dans ce cas précis, on fait en sorte que le contrôle intercepte le mouvement du curseur. Lorsque celui-ci bouge, la méthode `mouvement` est appelée comme le constructeur de `MaListbox` l'a spécifié. La méthode `nearest` permet de définir l'intitulé le plus proche du curseur. La méthode `itemconfig` permet de changer le fond de cet intitulé en gris après avoir modifié le fond de l'intitulé précédent pour qu'il retrouve sa couleur d'avant. Le résultat est illustré la figure~ref{listbox_curseur_soiut}.




L'intitulé sous le curseur de la souris a un fond gris.

### Fenêtres personnalisées : utilisation des classes

Cet exemple prolonge l'idée du paragraphe précédent. Lorsque l'interface devient complexe, il peut être utile de créer ses propres fenêtres. Jusqu'à présent, seules des fonctions ont été attachées à événement comme la pression d'un bouton mais il est possible d'attacher la méthode d'une classe ce que développe l'exemple qui suit.

```
import tkinter

class MaFenetre :
    def __init__(self, win) :
        self.win = win
        self.creation()
```

(suite sur la page suivante)

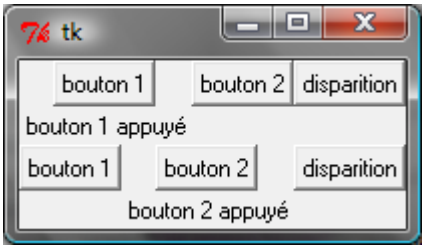
(suite de la page précédente)

```
    def creation(self) :
        b1 = tkinter.Button(self.win, text="bouton 1", command=self.commande_bouton1)
        b2 = tkinter.Button(self.win, text="bouton 2", command=self.commande_bouton2)
        b3 = tkinter.Button(self.win, text="disparition", command=self.disparition)
        b1.grid(row=0, column=0)
        b2.grid(row=0, column=1)
        b3.grid(row=0, column=2)
        self.lab = tkinter.Label(self.win, text = "-")

    def commande_bouton1(self) :
        # on déplace l'objet lab de type Label
        self.lab.configure(text = "bouton 1 appuyé")
        self.lab.grid(row = 1, column = 0)

    def commande_bouton2(self) :
        # on déplace l'objet lab de type Label
        self.lab.configure(text = "bouton 2 appuyé")
        self.lab.grid(row = 1, column = 1)

    def disparition(self) :
        # on fait disparaître l'objet lab de type Label
        self.lab.grid_forget()

if __name__ == "__main__" :
    root = tkinter.Tk()
    f = MaFenetre(root)
    root.mainloop()
```

Ce programme crée trois boutons et attache à chacun d'entre eux une méthode de la classe `MaFenetre`. Le constructeur de la classe prend comme unique paramètre un pointeur sur un objet qui peut être la fenêtre principale, un objet de type Frame [599] ou Toplevel [600]. Cette construction permet de considérer cet ensemble de trois boutons comme un objet à part entière ; de ce fait il peut être inséré plusieurs fois comme le montre l'exemple suivant illustré par la figure qui suit.

```
root = tkinter.Tk()
f = tkinter.Frame()
f.pack()
MaFenetre(f)         # première instance
g = tkinter.Frame()
g.pack()
MaFenetre(g)         # seconde instance
root.mainloop()
```

La fenêtre est composée de deux instances de `MaFenetre`.

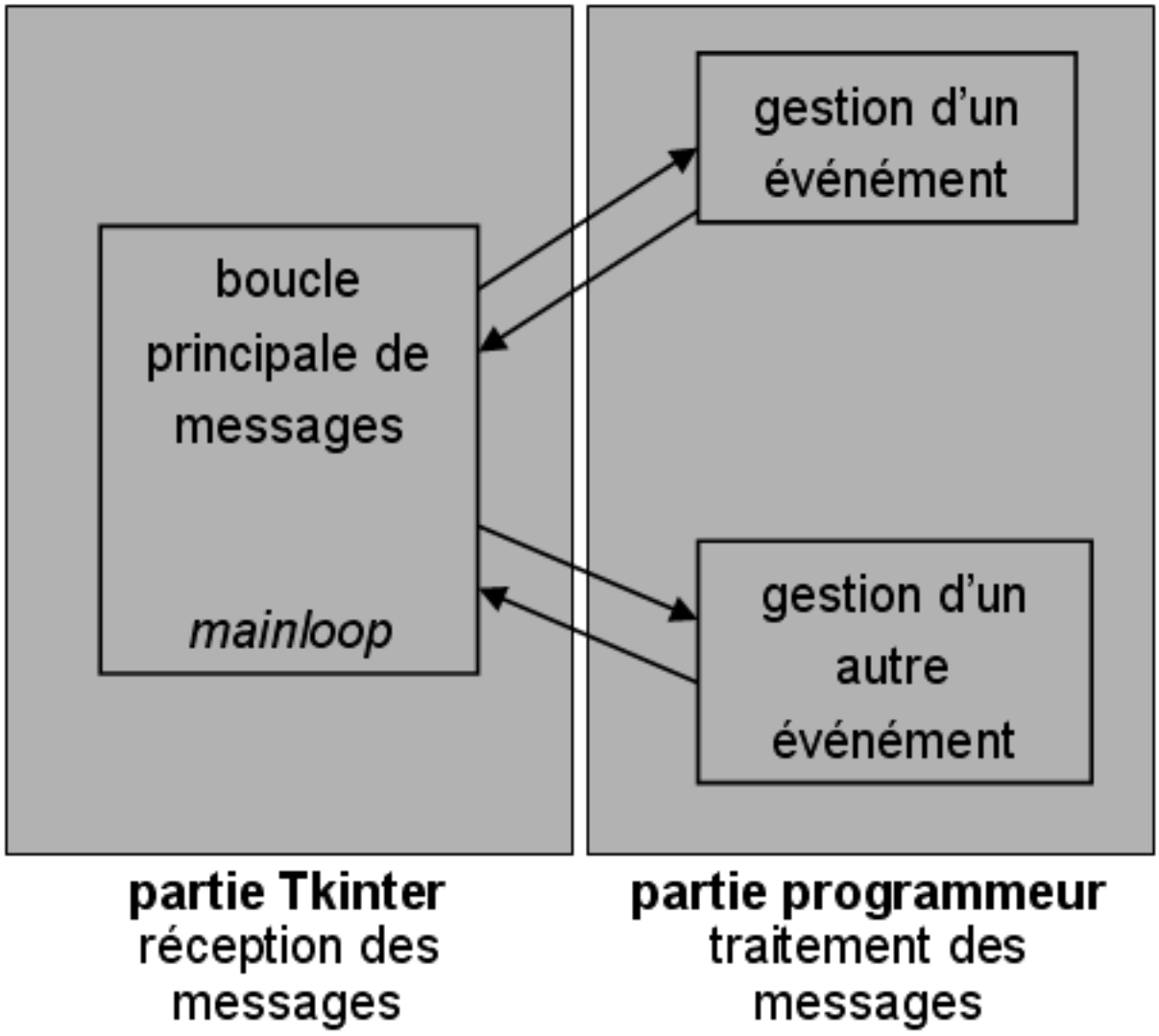



599. http://effbot.org/tkinterbook/frame.htm
600. http://effbot.org/tkinterbook/Toplevel.htm

### Séquence d'événements

Il est facile de faire réagir le programme en fonction d'un événement, il suffit d'attacher cet événement à une méthode ou une fonction. En revanche, faire réagir le programme en fonction d'une séquence d'événements est plus complexe. En effet, le premier d'événement de la séquence active une fonction, il n'est pas possible d'attendre le second événement dans cette même fonction, ce dernier ne sera observable que si on sort de cette première fonction pour revenir à la fonction `mainloop`, la seule capable de saisir le prochain événement.

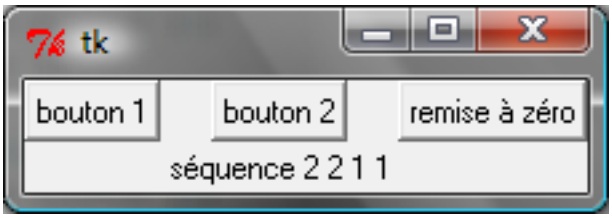
La figure qui suit précise la gestion des messages. tkinter [601] se charge de la réception des messages puis de l'appel au traitement correspondant indiqué par la méthode ou la fonction attachée à l'événement. Le programmeur peut définir les traitements associés à chaque événement. Ces deux parties sont scindées et à moins de reprogrammer sa boucle de message, il n'est pas évident de consulter les événements intervenus depuis le début du traitement de l'un d'eux.




La réception des événements est assurée par la fonction `mainloop` qui consiste à attendre le premier événement puis à appeler la fonction ou la méthode qui lui est associée si elle existe. Les classes offrent un moyen simple de gérer les séquences d'événements au sein d'une fenêtre. Celle-ci fera l'objet d'une classe qui mémorise les séquences d'événements. Tous les événements feront appel à des méthodes différentes, chacune d'elles ajoutera l'événement à une liste. Après cette ajout, une autre méthode sera appelée pour rechercher une séquence d'événements particulière.

601. https://docs.python.org/3/library/tk.html

```
import tkinter

class MaFenetreSeq:
    def __init__(self, win):
        self.win = win
        self.creation()
        self.sequence = []

    def creation(self):
        b1 = tkinter.Button(self.win, text="bouton 1", command=self.commande_bouton1)
        b2 = tkinter.Button(self.win, text="bouton 2", command=self.commande_bouton2)
        b3 = tkinter.Button(self.win, text="remise à zéro", command=self.zero)
        b1.grid(row=0, column=0)
        b2.grid(row=0, column=1)
        b3.grid(row=0, column=2)
        self.lab = tkinter.Label(self.win, text = "-")

    def commande_bouton1(self):
        # ajoute 1 à la liste self.sequence
        self.sequence.append(1)
        self.controle()

    def commande_bouton2(self):
        # ajoute 2 à la liste self.sequence
        self.sequence.append(2)
        self.controle()

    def zero(self):
        # on vide la liste self.sequence
        self.sequence = []
        self.lab.grid_forget()

    def controle(self):
        # on compare la liste sequence entre [1,2,1] et [2,2,1,1]
        # dans ce cas, on fait apparaître l'objet lab
        l = len(self.sequence)
        if l >= 3 and self.sequence [l-3:] == [1,2,1]:
            self.lab.configure(text = "séquence 1 2 1")
            self.lab.grid(row = 1, column = 0)
        elif l >= 4 and self.sequence [l-4:] == [2,2,1,1]:
            self.lab.configure(text = "séquence 2 2 1 1")
            self.lab.grid(row = 1, column = 1)

if __name__ == "__main__":
    root = tkinter.Tk()
    f = MaFenetreSeq(root)
    root.mainloop()
```



### Communiquer un résultat par message

Le module tkinter [602] permet de définir ses propres messages qui peuvent servir à communiquer des informations. Une fonction est par exemple appelée lorsqu'un bouton est pressé. Celle-ci, une fois terminée, retourne son résultat sous forme de message envoyé à l'interface graphique. Ce message sera ensuite traité comme tout autre message et pourra être intercepté ou non.

Le programme suivant utilise ce concept. La pression d'un bouton appelle une fonction `event_generate` qui génère un message personnalisé `<<perso>>` avec comme paramètre `rooty=-5`. A son tour, celui-ci est attrapé et dirigé vers la fonction `perso` qui affiche l'attribut `y_root` de la classe Event [603] qui a reçu la valeur codes{-5} lors de l'appel de la fonction `event_generate`. Ce procédé ne permet toutefois que de renvoyer que quelques résultats entiers.

```
import tkinter

def affiche_touche_pressee():
    root.event_generate("<<perso>>", rooty=-5)

def perso(evt):
    print("perso", evt.y_root)

root = tkinter.Tk()
b = tkinter.Button(text="clic", command=affiche_touche_pressee)
b.pack()
root.bind("<<perso>>", perso)  # on intercepte un événement personnalisé
root.mainloop ()
```

Ce principe est plus utilisé lorsque l'interface graphique est couplée avec les threads, l'ensemble est présenté au paragraphe *Interface graphique* (page 266).

602. https://docs.python.org/3/library/tk.html
603. http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm#events

CHAPITRE 10

# Données

CHAPITRE 11

# Exemples

Exemples, FAQ (Frequently Asked Questions), notebooks et autres petits bouts de codes qu'on espère pouvoir copier coller sans les comprendre. Parfois c'est vrai.

CHAPITRE 12

# Précis

## 12.1 Cheat sheet : Python


— *Le langage* (page 320)
— *Les variables* (page 321)
  — *Les types immuables* (page 321)
— *Les types modifiables* (page 323)
  — *Liste* (page 324)
  — *Dictionnaire* (page 325)
  — *Tableaux numériques* (page 326)
— *Tests et boucles* (page 326)
  — *Tests* (page 326)
  — *Boucles* (page 327)
  — *Fonction* `enumerate` (page 328)
  — *Fonction* `zip` (page 328)
  — *Fonction* `map` (page 328)
  — *Autres mot-clés* (page 329)
— *Fonctions* (page 329)
  — *Définition, paramètres* (page 329)
  — *mot-clé* `lambda` (page 330)
  — *mot-clé* `yield` (page 331)
— *Classes* (page 332)
  — *Déclaration d'une classe* (page 332)
  — *Attributs et méthodes* (page 333)
  — *Méthodes statiques* (page 334)
  — *Opérateurs* (page 334)
  — *Copie d'instances* (page 334)
  — *Héritage* (page 335)
  — *Décorateur* (page 336)
— *Fichiers* (page 337)
  — *Ecriture dans un fichier texte* (page 337)
  — *Lecture dans un fichier texte* (page 338)

- — *Modules* (page 339)
- — *Exceptions* (page 340)
- — *Erreurs, confusions fréquentes* (page 342)
  - — *Variables* (page 342)
    - — *Chaîne de caractères = tableau de caractères* (page 342)
    - — *Guillemets ou pas* (page 342)
  - — *Boucles* (page 343)
    - — *`range` ou pas `range`* (page 343)
    - — *Initialisation* (page 343)
  - — *Fonctions* (page 343)
    - — *Différence entre `print` et `return`* (page 343)
- — *Trois conseils pour écrire un programme* (page 344)
  - — *Des petites fonctions* (page 344)
  - — *Séparer les calculs, le chargement des données, l'interface graphique* (page 344)
  - — *Utiliser des tests unitaires* (page 345)
- — *Trucs et astuces* (page 345)
  - — *Partager du code* (page 345)
  - — *Moteurs de recherche* (page 345)


Ce document résume en quelques pages les éléments essentiels de la syntaxe du langage python. Quelques lignes suffisent pour découvrir la version installée :

```
import sys
print (sys.version)
```

Avec la version *3.x*, le langage a introduit quelques changements importants qui seront précisés. Il est préférable de choisir la version 3 plutôt que 2.7. Outre le fait qu'elle contient les dernières évolutions, elle est beaucoup plus cohérente en ce qui concerne les chaînes de caractères.

### 12.1.1 Le langage

Quelques rappels sur le langage :

- — Il n'y a pas de séparateur d'instructions, il faut écrire une instruction par ligne et décaler les lignes dans une boucle, un test, une fonction, une classe. Pour écrire une instruction sur plusieurs lignes, il faut utiliser le caractères \ sur toutes les lignes de l'instruction sauf la dernière. Il ne doit rien y avoir derrière ce caractère, ni espace, ni commentaires. Cette règle n'est nécessaire s'il s'agit d'une liste d'éléments séparés par des virgules.
- — On peut mettre autant d'espaces qu'on veut. Il peut n'y en avoir aucun sauf derrière une instruction `for`, `if`, ...
- — Les commentaires dans un programme commencent par le symbole # et vont jusqu'à la fin de la ligne.
- — L'instruction `print` permet d'afficher n'importe quelle information. Elle affiche le résultat de la méthode `__str__` pour les classes. **Version 3.x :** `print` est une fonction, tout ce qui doit être affiché doit l'être entre parenthèses.
- — L'instruction `help` affiche l'aide associée à une variable, une fonction, une classe, une méthode, un module. Pour une fonction, une classe, une méthode du programme, cette aide correspond à une chaîne de caractères encadrée par trois `"`. Ce message d'aide peut s'étaler sur plusieurs lignes.

<<<

```
def fonction():
    """fonction de
    démonstration"""
    return 0


help(fonction)
```

>>>

```
    Help on function fonction in module pyquickhelper.sphinxext.sphinx_runpython_
↪extension:

    fonction()
        fonction de
        démonstration
```

**2 astuces en cas de problème**

En cas de doute sur une partie de code, un calcul, une priorité entre opérateurs, le résultat d'une fonction, il est possible d'utiliser la fonction `print` pour afficher une valeur intermédiaire pendant un calcul. Il ne faut pas non plus hésiter à vérifier sur un petit exemple dans un petit programme que des lignes douteuses font exactement ce pour quoi elles ont été écrites.

Il est souvent utile de chercher sur Internet des exemples de programmes pour corriger une syntaxe incorrecte, utiliser un moteur de recherche en ajoutant une requête commençant par le mot *python*. Pour les erreurs, il est parfois intéressant de recopier intégralement le message d'erreur sur un moteur de recherche, les réponses obtenues sont souvent assez claires.

### 12.1.2 Les variables

Le nom d'une variable commence par une lettre ou un blanc souligné, il peut également inclure par la suite des chiffres. Python distingue les minuscules des majuscules. La portée d'une variable, c'est-à-dire la portion de code où elle définie, s'étend depuis sa déclaration (première affectation) jusqu'à la fin du programme ou de la fonction où elle est définie. Pour déclarer une variable portant le nom `va`, il suffit d'écrire :

```
va = <valeur>
```

Le type de `<valeur>` détermine le type de la variable `va`. Si une variable de même portée portait déjà ce nom-là, son contenu est écrasé (perdu aussi). L'instruction `type(x)` retourne le type de la variable `x`. Un **identificateur** qu'il désigne une variable, une fonction, une classe ou un module garde la valeur de sa dernière affectation.

#### Les types immuables

Les variables de type immuable ne peuvent pas être modifiées.

- `None`, ce type veut dire rien, il est utilisé comme convention de programmation pour dire qu'un algorithme, un calcul ne s'est pas terminé correctement ou une valeur n'a pas encore été calculée.
- `bool` : un booléen (résultat d'un test)
- `int` : un entier
- `float` : un réel
- `complex` : un complexe
- `str` : une chaîne de caractères ou string, elle apparaît entre guillemets, entre apostrophes, entre trois guillemets (`"""`) si elle s'étend sur plusieurs lignes. `s = "exemple"`. **Version 2.7 :** il existe un autre type `unicode` utilisé pour représenter des chaînes de caractères non latines (en chinois par exemple).
- `tuple` : un vecteur d'éléments de types identiques ou différents, il apparaît entre parenthèses, on accède à un de ses éléments à l'aide de crochets. Les éléments d'un t-uple `t` sont indicés de 0 à `len(t)-1` inclus.

```
t = ()          # tuple vide
t = (2, "e")    # tuple de deux éléments
print(t[0])     # affiche le premier élément
```

L'affectation d'une valeur de type immuable à une variable est une copie. On peut appliquer sur les types numériques les opérations usuelles (`+ * - / \% ** += *= -= /= \%= **=`) `**` est le symbole pour puissance : `3 ** 4` = $3^4$. On rappelle que `a += 10` est équivalent à `a = a + 10`, ceci signifie que la valeur de `a` avant le calcul

n'a plus besoin d'exister. Le *et* logique et le *ou* logique sont notés `and` et `or`. Les priorités sont celles usuellement utilisées en mathématique, en cas de doute, il faut utiliser des parenthèses.

Les opérateurs de comparaison (`< > == <= >=`) s'appliquent sur tous les types numériques ainsi que sur les chaînes de caractères. **Rappel :** les minuscules sont classées après les majuscules.

<<<

```
print('A' < 'a')
```

>>>

```
    True
```

**Fréquente source de bug :**

- **Version 2.7 :** une division entière a pour résultat le quotient et non un nombre décimal. Autrement dit : `1/2 = 0` et non `0.5`.
- **Version 3.x :** une division entière a pour résultat un réel. Autrement dit : `1/2 = 0.5`. Pour une division entière, il faut utiliser `//` : `1//2 = 0`.

Pour convertir une information d'un type à un autre, il suffit d'utiliser le nom de ce type suivi de la valeur à convertir entre parenthèses : `b = float ("2.145")` équivaut à la conversion d'une chaîne de caractères en réel.

L'addition d'un t-uple et d'une valeur retourne un t-uple incluant cette valeur à la fin (plus long d'un élément). L'addition de deux t-uples concatène les deux t-uples. L'addition de deux chaînes de caractères retourne leur concaténation.

Pour savoir si un élément `x` fait partie d'un t-uple `t`, il faut utiliser la syntaxe `x in t` dont la réciproque est `x not in t`.

La fonction len [604] retourne la longueur d'un tuple ou d'une chaîne de caractères. Les éléments ou les caractères d'un tuple ou d'une chaîne de caractères `t` sont indicés de 0 à `len (t)-1` inclus.

```
t[i:j]  # correspond à un sous-ensemble allant des indices i à j exclu
t[:j]   #  = t[0:j]
t[i:]   # = t [i: len (t)]
```

Pour les chaînes de caractères, on utilise fréquemment les méthodes suivantes :

```
st = "langage python"
st = 'langage python'           # idem
st = 'un guillement "'          # chaîne contenant un guillement
st = "un guillement \""         # chaîne contenant un guillement, il faut ajouter \
                                #     pour ne pas confondre avec l'autre guillement
st = st.upper ()                # mise en lettres majuscules
i  = st.find ("PYTHON")         # on cherche "PYTHON" dans st
print (i)                       # affiche 8  Version 3.x, écrire print (i),
                                #    pour la version 2.x, écrire print i
print (st.count ("PYTHON"))     # affiche 1  Version 3.x : idem print (...)
print (st.count ("PYTHON", 9))  # affiche 0  Version 3.x : idem print (...)
```

604. https://docs.python.org/3/library/functions.html#len

| | |
|---|---|
| `count( sub[, start[, end]])` | Retourne le nombre d'occurences de la chaîne de caractères `sub`, les paramètres par défaut `start` et `end` permettent de réduire la recherche entre les caractères d'indice `start` et `end` exclu. Par défaut, `start` est nul tandis que `end` correspond à la fin de la chaîne de caractères. |
| `find( sub[, start[, end]])` | Recherche une chaîne de caractères `sub`, les paramètres par défaut `start` et `end` ont la même signification que ceux de la fonction `count`. Cette fonction retourne -1 si la recherche n'a pas abouti. |
| `isalpha()` | Retourne `True` si tous les caractères sont des lettres, `False` sinon. |
| `isdigit()` | Retourne `True` si tous les caractères sont des chiffres, `False` sinon. |
| `replace( old, new[, count])` | Retourne une copie de la chaîne de caractères en remplaçant toutes les occurrences de la chaîne `old` par `new`. Si le paramètre optionnel `count` est renseigné, alors seules les `count` premières occurrences seront remplacées. |
| `split( [sep [, maxsplit]])` | Découpe la chaîne de caractères en se servant de la chaîne `sep` comme délimiteur. Si le paramètre `maxsplit` est renseigné, au plus `maxsplit` coupures seront effectuées. |
| `upper()` | Remplace les minuscules par des majuscules. |
| `lower()` | Remplace les majuscules par des minuscules. |
| `join ( li )` | `li` est une liste, cette fonction agglutine tous les éléments d'une liste séparés par `sep` dans l'expression `sep.join ( ["un", "deux"])`. |

Quelques fonctions s'appliquant aux chaînes de caractères, l'aide associée au langage python fournira la liste complète. Certains des paramètres sont encadrés par des crochets, ceci signifie qu'ils sont facultatifs. L'affichage de réels nécessite parfois de tronquer la partie décimale ce qui est fait grâce à la syntaxe suivante :

<<<

```
x = 0.123456789
print("%1.2f" % x)     # donne 0.12
s = "%2.2e %s" % (3.14159, "est une approximation de pi")
print(s)               # Version 2.x : print s
```

>>>

```
    0.12
    3.14e+00 est une approximation de pi
```

### 12.1.3 Les types modifiables

python fournit deux types modifiables : les listes et les dictionnaires. Pour ces deux types, **il faut faire attention à chaque affectation**.

```
a = [1,2]
b = a
```

La seconde ligne ne fait pas une copie de la première liste, elle ne fait que créer un second nom pour nommer la même liste. Pour copier une liste ou un dictionnaire, il faut utiliser :

```
a = [1,2]
import copy
b = copy.copy (a)
```

ou, si la liste inclut également d'autres listes ou dictionnaires :

```
a = [1,2]
import copy
b = copy.deepcopy (a)
```

Cette remarque s'applique à tout type modifiable, liste, dictionnaire ou tout autre classe. La suppression d'une variable n'implique pas la suppression de toutes les variables se référant à une seule et même instance de classe.

### Liste

Une liste est une sorte de tableau qui permet de mémoriser un ensemble d'éléments de types variés. C'est une sorte de t-uple modifiable.

```
x = [4,5]                # création d'une liste composée de deux entiers
x = ["un",1,"deux",2]    # création d'une liste composée deux chaînes de caractères
                         # et de deux entiers, l'ordre d'écriture est important
x = [3,]                 # création d'une liste d'un élément, sans la virgule,
                         # le résultat reste une liste
x = [ ]                  # crée une liste vide
x = list ()              # crée une liste vide
```

Opérations qu'une liste supporte.

| | |
|---|---|
| `x in s` | vrai si `x` est un des éléments de `l` |
| `x not in s` | réciproque de la ligne précédente |
| `l + t` | concaténation de `l` et `t` |
| `l * n` | concatène `n` copies de `l` les unes à la suite des autres |
| `l[i]` | retourne le ième élément de `l`, à la différence des T-uples, l'instruction `l [i] = "3"` est valide, elle remplace l'élément `i` par 3. |
| `l[i:j]` | retourne une liste contenant les éléments de `l` d'indices `i` à `j` exclu. Il est possible de remplacer cette sous-liste par une autre en utilisant l'affectation `` l[i :j] = l2 `` où `l2` est une autre liste (ou un T-uple) de dimension différente ou égale. |
| `l[i:j:k]` | retourne une liste contenant les éléments de `l` dont les indices sont compris entre `i` et `j` exclu, ces indices sont espacés de `k` (le pas) : `i, i+k, i+2k, i+3k, ...` Ici encore, il est possible d'écrire l'affectation suivante : `l[i:j:k] = l2` mais `l2` doit être une liste (ou un T-uple) de même dimension que `l[i:j:k]` |
| `len(l)` | nombre d'éléments de `l` |
| `min(l)` | plus petit élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `max(l)` | plus grand élément de `l`, résultat difficile à prévoir lorsque les types des éléments sont différents |
| `sum(l)` | retourne la somme de tous les éléments |
| `del l [i:j]` | supprime les éléments d'indices entre `i` et `j` exclu. Cette instruction est équivalente à `l [i:j] = []`. |
| `list(x)` | convertit `x` en une liste quand cela est possible |

Opérations disponibles sur les listes, identiques à celles des T-uples, on suppose que `l` et `t` sont des listes, `i` et `j` sont des entiers. `x` est quant à lui quelconque.

| | |
|---|---|
| `l.count (x)` | Retourne le nombre d'occurrences de l'élément `x`. `count` est une méthode de la classe `list`. |
| `l.index (x)` | Retourne l'indice de la première occurrence de l'élément `x` dans la liste `l`. Si celle-ci n'existe, une exception est déclenchée. |
| `l.append (x)` | Ajoute l'élément `x` à la fin de la liste `l`. Si `x` est une liste, cette fonction ajoute la liste `x` en tant qu'élément, au final, la liste `l` ne contiendra qu'un élément de plus. |
| `l.extend (k)` | Ajoute tous les éléments de la liste `k` à la liste `l`. La liste `l` aura autant d'éléments supplémentaires qu'il y en a dans la liste `k`. |
| `l.insert(i,x)` | Insère l'élément `x` à la position `i` dans la liste `l`. |
| `l.remove (x)` | Supprime la première occurence de l'élément `x` dans la liste `l`. S'il n'y a aucune occurrence de `x`, cette méthode déclenche une exception. |
| `l.pop ([i])` | Retourne l'élément `l[i]` et le supprime de la liste. Le paramètre `i` est facultatif, s'il n'est pas précisé, c'est le dernier élément dont la valeur est d'abord retournée puis il est supprimé de la liste. |
| `l.reverse (x)` | Retourne la liste, le premier et dernier élément échangent leurs places, le second et l'avant dernier, et ainsi de suite. |
| `l.sort([key=None, reverse=False])` | Cette fonction trie la liste par ordre croissant. Le paramètre `f` est facultatif, il permet de préciser une fonction qui associe une clé de tri à chaque élément. Le paramètre *reverse* permet de trier dans un sens ou dans l'autre. |

Opérations permettant de modifier une liste on suppose que `l` est une liste, `x` est quant à lui quelconque. Les listes peuvent aussi être définies à partir d'une écriture abrégée :

<<<

```
li = list(range(0, 5))                  # liste des entiers de 0 à 5 exclu
y = [i for i in li if i % 2 == 0]       # sélection des éléments pairs
print(y)
z = [i+j for i in li for j in li]       # construit tous les nombres i+j possibles
print(z)
```

>>>

```
    [0, 2, 4]
    [0, 1, 2, 3, 4, 1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8]
```

### Dictionnaire

Un dictionnaire est un tableau pour lequel les indices ou clés ne sont pas uniquement des entiers mais tout type non modifiable (le plus souvent un entier, un réel, une chaîne de caractères, un t-uple).

<<<

```
x = {"cle1": "valeur1", "cle2": "valeur2"}
print(x["cle1"])
x[(0, 1)] = "clé tuple"    # ajoute une nouvelle valeur dont la clé est (0,1)
#   les parenthèses sont superflues
print(x)
y = {}                     # crée un dictionnaire vide
z = dict()                 # crée aussi un dictionnaire vide
```

>>>

```
    valeur1
    {'cle1': 'valeur1', 'cle2': 'valeur2', (0, 1): 'clé tuple'}
```

Opérations qu'un dictionnaire supporte.

| | |
|---|---|
| `x in d` | vrai si `x` est une des clés de `d` |
| `x not in d` | réciproque de la ligne précédente |
| `l[i]` | retourne l'élément associé à la clé `i` |
| `len(d)` | nombre d'éléments de `d` |
| `min(d)` | plus petite clé |
| `max(d)` | plus grande clé |
| `del l [i]` | supprime l'élément associé à la clé `i` |
| `list (d)` | retourne une liste contenant toutes les clés du dictionnaire `d`. |
| `dict (x)` | convertit `x` en un dictionnaire si cela est possible, en particulier, `d` est égal à `dict(d.items())` |

Opérations disponibles sur les dictionnaires, `d` est un dictionnaire, `x` est quant à lui quelconque.

| | |
|---|---|
| `d.copy()` | Retourne une copie de `d`. |
| `x in d` | Retourne `True` si `x` est une clé de `d`. |
| `d.items()` | Retourne un itérateur parcourant contenant tous les couples (clé, valeur) inclus dans le dictionnaire. Pour obtenir une liste, il faut écrire `list(d.items())`. **Version 2.x :** retourne une liste. |
| `d.keys()` | Retourne un itérateur parcourant toutes les clés du dictionnaire `d`. **Version 2.x :** retourne une liste. |
| `d.values()` | Retourne un itérateur parcourant toutes les valeurs du dictionnaire `d`. **Version 2.x :** retourne une liste. |
| `d.get(k[,x])` | Retourne `d[k]`, si la clé `k` est manquante, alors la valeur `None` est retournée à moins que le paramètre optionnel `x` soit renseigné, auquel cas, ce sera ce paramètre qui sera retourné. |
| `d.clear()` | Supprime tous les éléments du dictionnaire. |
| `d.update(d2)` | Pour chaque clé de `d1`, `d[k] = d2[k]` |
| `d.setdefault(k[,x])` | Retourne `d[k]` si la clé `k` existe, sinon, affecte `x` à `d[k]`. |
| `d.popitem()` | Retourne un élément et le supprime du dictionnaire. |

Méthodes associées aux dictionnaires, `d`, `d2` sont des dictionnaires, `x` est quant à lui quelconque.

#### Tableaux numériques

Ce type ne fait pas partie du langage python standard mais il est couramment utilisé.

```
import numpy
a = numpy.array ( [0,1] )
```

Il permet de convertir des listes en une structure plus appropriée au calcul qui sont nettement plus rapides. En contrepartie, il n'est pas aussi rapide d'ajouter ou supprimer des éléments.

### 12.1.4 Tests et boucles

#### Tests

Les tests permettent d'exécuter telle ou telle instruction selon la valeur d'une condition. Le test est suivi de `:` et les instructions dépendant de ce test sont indentées (décalées vers la droite).

```
if x < 5 :
    x = x * 2
    ...
```

Il peut y avoir une contrepartie :

```
if x < 5:
    x = x * 2
    ...
else:
    x = x * 3
    ...
```

S'il n'y a qu'une seule instruction, elle peut s'écrire en bout de ligne :

```
if x < 5:
    x = x * 2
else:
    x = x * 3
```

Il peut y avoir plusieurs conditions qui s'enchaînent :

```
if x < 5:
    x = x*2
elif x > 5:
    x = x*3
else:
    x = x*6
```

Il existe un raccourci pour les intervalles :

```
if 5 < x and x < 10 :      # peut être écrit : if 5 < x < 10 :
    ...
```

### Boucles

Il y a deux types de boucles, la boucle `for` parcourt un ensemble, la boucle `while` continue tant qu'une condition est vraie. Comme pour les tests, une boucle est suivie du syumbol `:`, les lignes incluses dans cette boucle sont indentées à moins qu'il n'y en ait qu'une seule, auquel cas elle peut être écrite après le symbole `:` sur la même ligne.

```
while condition :
    # lignes décalées
    # contenu de la boucle
```

Quelques exemples de boucles `for` :

```
for i in range(0,n) :              # parcourt tous les entiers de 0 à n-1 inclus
for i in xrange(0,n) :             # même chose mais en plus rapide
                                   # Version 3.x : la fonction xrange n'existe plus,
                                   #               et range équivaut à xrange
for i in range(n,0,-1) :           # parcourt tous les entiers de n à 1 inclus
                                   #                  dans le sens décroissant
for i in range(2,1000,3) :         # parcourt tous les entiers de 2 à 1000 de 3 en 3
                                   #                                      (2,5,8,...)
for e in li :                      # parcourt tous les éléments de la liste li
for cle,valeur in di.items() :     # parcourt tous les éléments du dictionnaire di
```

Pour toutes les boucles, l'instruction `break` permet de sortir de la boucle, l'instruction `continue` passe directement à l'itération suivante sans exécuter les instructions qui suivent l'instruction `continue`.

### Fonction `enumerate`

On peut écrire :

```
l = [ 4, 5, 6 ]
s = 0
for i in range(0,len(l)) :
    s += l[i]
```

Ou utiliser la fonction enumerate [605] qui retourne chaque élément et sa position dans l'ensemble :

```
l = [ 4, 5, 6 ]
s = 0
for i,x in enumerate(l) :
    s += x
```

### Fonction `zip`

Pour faire la somme de deux listes terme à terme, on peut écrire :

```
l = [ 4, 5, 6 ]
g = [ 3,10,11 ]
s = 0
for i in range(0,len(l)) :
    s += l[i] + g[i]
```

Ou utiliser la fonction zip [606] :

```
l = [ 4, 5, 6 ]
g = [ 3,10,11 ]
s = 0
for x,y in zip(l,g) :
    s += x + y
```

### Fonction `map`

Il est possible d'éviter une fonction pour éviter d'écrire une boucle avec la fonction map [607]. Elle applique une fonction à chaque élément d'un ensemble.

<<<

```
def fonction(x):
    return x % 2


li = [3, 4, 5]
li2 = map(fonction, li)
print(list(li2))
```

>>>

605. https://docs.python.org/3/library/functions.html#enumerate
606. https://docs.python.org/3/library/functions.html#zip
607. https://docs.python.org/3/library/functions.html#map

```
    [1, 0, 1]
```

A priori, l'ensemble qui en résulte contient autant d'éléments sauf si on utilise la fonction filter [608]. L'exemple suivant affiche tous les nombres pairs.

<<<

```
def fonction(x):
    if x % 2 == 0:
        return True


li = [3, 4, 5]
li2 = filter(fonction, li)
print(list(li2))
```

>>>

```
    [4]
```

#### Autres mot-clés

Le mot-clé `with` est utile lorsqu'on une utilise une variable pour une portion réduite de code. Cette notation cache deux appels implicites à deux méthodes comme indiqué ci-dessous.

```
with random_matrix(1000,1000) as mat :
    #   appelle mat.__enter__()
    ...
    #   appelle mat.__exit__()
```

Lorsque ces méthodes sont surchargées, l'utilisation de cette syntaxe réduit la taille du code. C'est le cas des fichiers.

### 12.1.5 Fonctions

#### Définition, paramètres

Les fonctions ou sous-programmes permettent de faire la même chose sans avoir à recopier le code informatique plusieurs fois dans le programme. Elles acceptent plusieurs paramètres ou aucun, elles peuvent retourner plusieurs résultats ou aucun. Leur déclaration suit le schéma suivant :

```
def exemple_fonction (p1, p2, p3) :
    # code de la fonction
    return r1, r2

a,b = exemple_fonction (1,2,3)    # exemple d'appel de la fonction
```

L'instruction `return` n'est pas obligatoire mais si elle est présente à un ou plusieurs endroits, aucune autre instruction de la fonction ne sera exécutée après l'exécution de la première instruction `return` rencontrée lors de l'exécution de la fonction. Les fonctions peuvent être récursives et inclure des paramètres par défaut : ces paramètres reçoivent une valeur même si celle-ci n'est pas précisée lors de l'appel.

```
def exemple_fonction (p1, p2 = 4, p3 = 7) :
    # code de la fonction
    return r1, r2
```

(suite sur la page suivante)

608. https://docs.python.org/3/library/functions.html#filter

(suite de la page précédente)

```
a,b = exemple_fonction (1)          # = exemple_fonction (1,4,7)
a,b = exemple_fonction (1,2,3)      # = exemple_fonction (1,2,3)
a,b = exemple_fonction (1,2)        # = exemple_fonction (1,2,7)
a,b = exemple_fonction (1,p3 = 2)   # = exemple_fonction (1,4,2)
```

Les paramètres par défaut doivent tous être mis en fin de déclaration, l'exemple suivant n'est pas correct :

```
def exemple_fonction (p1, p2 = 4, p3) :
    # code de la fonction
    return r1, r2
# affiche le message d'erreur : SyntaxError: non-default argument follows default␣
↪argument
```

En ce qui concerne les paramètres, les paramètres de type non modifiable sont passés par valeur (une modification à l'intérieur de la fonction n'a pas de répercution à l'extérieur).

<<<

```
def exemple_fonction(p1):
    p1 = 3


a = 1
exemple_fonction(a)
print(a)
```

>>>

```
    1
```

Les paramètres de type modifiable sont passés par référence (une modification à l'intérieur de la fonction a des répercutions à l'extérieur).

<<<

```
def exemple_fonction(p1):
    p1[0] = 3


a = [1]
exemple_fonction(a)
print(a)
```

>>>

```
    [3]
```

### mot-clé `lambda`

Le mot-clé `lambda` permet de définir des fonctions au sein d'une expression.

<<<

```
def fonction(x):
    return x % 2
```

(suite sur la page suivante)

(suite de la page précédente)

```
li = [3, 4, 5]
li2 = map(fonction, li)
print(list(li2))
```

>>>

```
    [1, 0, 1]
```

Peut-être écrit comme :

<<<

```
li = [3, 4, 5]
li2 = map(lambda x: x % 2, li)
print(list(li2))
```

>>>

```
    [1, 0, 1]
```

Et si on veut ajouter un paramètre à la fonction `lambda` :

<<<

```
li = [3, 4, 5]
k = 2
li2 = map(lambda x, y=k: x % y, li)
print(list(li2))
```

>>>

```
    [1, 0, 1]
```

### mot-clé `yield`

La programmation fonctionnelle est de plus en plus utilisée. En python, elle se décline sous la forme d'itérateur.

<<<

```
def iterate_double_on_list(l):
    for x in l:
        yield x*2


print(iterate_double_on_list([4, 5, 6]))
```

>>>

```
    <generator object run_python_script_139651613789208.<locals>.iterate_double_on_
↪list at 0x7f032d085228>
```

La fonction itère sur un ensemble mais ne fait rien tant qu'on ne parcourt pas l'ensemble qu'elle génère :

<<<

```
def iterate_double_on_list(l):
    for x in l:
        yield x*2


for x in iterate_double_on_list([4, 5, 6]):
    print(x)
```

>>>

```
    8
    10
    12
```

La version 3 du langage python a changé des fonctions pour qu'elle retourne un itérateur sur un ensemble et non l'ensemble lui-même.

### 12.1.6 Classes

Les classes sont un moyen de définir de nouveaux types modifiables de variables. Peu de programmes ne les utilisent pas. Une classe est un ensemble d'attributs (ou variables) et de méthodes (ou fonctions). Un programme utilisant les classes est orienté objet. Il est possible de faire les mêmes choses avec ou sans classes mais leur utilisation rend d'ordinaire les grands programmes plus facile à comprendre et à construire.

#### Déclaration d'une classe

Pour déclarer une classe, on procède comme suit :

<<<

```
class ma_classe:
    def __init__(self, att1, att2, att3):
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3


a = ma_classe(-1, 1, 2)  # déclare une variable de type ma_classe
print(a.att1)  # affiche -1
print(a.att2)  # affiche 3
print(a.att3)  # affiche 4
print(a.att4)  # affiche -12
```

>>>

```
    -1
    1
    2
    -2
```

Lors de la déclaration de la variable `a`, le langage python exécute la méthode `__init__` aussi appelée constructeur. Elle permet de définir les attributs de la classe directement à partir des paramètres ou comme le résultat d'un calcul ou d'une fonction. Le constructeur comme toutes les autres méthodes possède comme premier paramètre `self` qui permet d'accéder aux attributs et aux méthodes de la classe. Le programme suivant est équivalent au premier.

<<<

```
class ma_classe:
    def __init__(self, att1, att2, att3):
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = self.calcule4()

    def calcule4(self):
        return self.att1 * self.att2 * self.att3


a = ma_classe(-1, 1, 2)  # déclare une variable de type ma_classe
print(a.att1)            # affiche -1
print(a.att2)            # affiche 3
print(a.att3)            # affiche 4
print(a.att4)            # affiche -12
```

>>>

```
    -1
    1
    2
    -2
```

### Attributs et méthodes

Les attributs sont déclarés le plus souvent à l'intérieur du constructeur, plus généralement à l'intérieur de toute méthode, voire à l'extérieure de la classe. Pour y faire référence à l'intérieur d'une méthode on fait précéder le nom de l'attribut de `self.`, à l'extérieur de la classe, c'est le nom de l'instance suivi d'un point `.` qui précède le nom de l'attribut comme le montre le précédent exemple.

Une méthode est déclarée à l'intérieur de la classe. Elle accepte invariablement au moins un paramètre qui est `self` comme dans le précédent exemple. Les règles d'accès sont les mêmes que pour les attributs. Elles acceptent également la récursivité et les paramètres par défaut à l'exception du premier. Chaque instance de classe est également munie d'un dictionnaire `__dict__` qui recense tous les attributs.

<<<

```
class ma_classe:
    def __init__(self, att1, att2, att3):
        self.att1 = att1                    # attribut
        self.att2 = att2                    # attribut
        self.att3 = att3                    # attribut
        self.att4 = att1 * att2 * att3      # attribut

    def calcule(self, x):                   # méthode
        return self.att1 * self.att2 * self.att3 * x


a = ma_classe(1, 2, 3)
print(a.att1)                  # affiche 1
# affiche aussi 1, ligne équivalente à la précédente
print(a.__dict__["att1"])
print(a.calcule(2))            # appel d'une méthode
```

>>>

```
1
1
12
```

### Méthodes statiques

Les méthodes statiques sont comme des fonctions : elle ne nécessite pas d'instance d'un object pour être appelée.

<<<

```
class ma_classe:
    def __init__(self, att1, att2, att3):
        # ...
        pass

    @staticmethod
    def calcule_static(x, y):          # méthode statique
        return x * y


print(ma_classe.calcule_static(2, 3))  # appel d'une méthode statique
```

>>>

```
6
```

### Opérateurs

Les opérateurs sont des méthodes qui permettent une manipulation plus simple des objets. Leur nom est fixé par convention par le langage python, ils commencent et terminent par __.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3

    def __add__ (self, a) :
         return ma_classe (self.att1 + a.att1, self.att2 + a.att2, \
                       self.att3 + a.att3, self.att4 + a.att4)

a = ma_classe (1,2,3)
b = ma_classe (4,5,6)
c = a + b              # n'a de sens que si l'opérateur __add__ a été redéfini
```

Il existe un opérateur spécifique pour chaque opération, cet opérateur permet de donner un sens à une addition, une soustraction, ..., de deux instances d'une classe. L'opérateur `__str__` retourne une chaîne de caractères et est appelé par l'instruction `print`. L'opérateur `__cmp__` retourne un entier permettant à des instances de la classe d'être comparées et triées par une liste.

### Copie d'instances

Les instances de classes sont des objets modifiables, comme pour les listes, une simple affectation ne signifie pas une copie mais un second nom pour désigner le même objet.

<<<

```
class ma_classe:
    def __init__(self, att1, att2, att3):
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3


a = ma_classe(1, 2, 3)
b = a
b.att1 = -16
print(a.att1)  # affiche -16
print(b.att1)  # affiche -16
```

>>>

```
    -16
    -16
```

Il faut donc copier explicitement l'instance pour obtenir le résultat souhaité.

<<<

```
class ma_classe:
    def __init__(self, att1, att2, att3):
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3


a = ma_classe(1, 2, 3)
import copy
b = copy.copy(a)
b.att1 = -16
print(a.att1)  # affiche 1
print(b.att1)  # affiche -16
```

>>>

```
    1
    -16
```

Lorsque une classe inclut une variable de type classe, il faut utiliser la fonction `deepcopy` et non `copy`.

### Héritage

L'héritage est l'intérêt majeur des classes et de la programmation orientée objet. Lorsqu'une classe hérite d'une autre, elle hérite de ses attributs et de ses méthodes. Le simple fait d'hériter crée donc une classe équivalente.

```
class ma_classe :
    def __init__ (self, att1, att2, att3) :
        self.att1 = att1
        self.att2 = att2
        self.att3 = att3
        self.att4 = att1 * att2 * att3
```

(suite sur la page suivante)

(suite de la page précédente)

```
class ma_classe2 (ma_classe) :      # héritage simple
    pass                            # pour dire que la classe est vide
```

Mais hériter permet de faire deux choses :

1. ajouter des attributs et ajouter des méthodes
2. modifier le comportement d'une méthode existante

<<<

```
class ma_classe:
    def __init__(self, att1):
        self.att1 = att1
        self.att2 = self.calcul()

    def calcul(self):
        return self.att1 ** 2


class ma_classe2 (ma_classe):
    def calcul(self):
        # dans cette méthode, on change le comportement
        # de la méthode calcul tout en se servant de celui
        # de la classe mère
        return ma_classe.calcul(self) * self.att1


a = ma_classe(2)
b = ma_classe2(2)
print(a.att2)    # affiche 4 = 2 * 2
print(b.att2)    # affiche 8 = (2*2) * 2
```

>>>

```
    4
    8
```

### Décorateur

Le langage python permet quelques simplifications de code avec les décorateurs comme dans l'exemple suivant :

<<<

```
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped


def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped


@makebold
@makeitalic
```

(suite sur la page suivante)

(suite de la page précédente)

```
def hello():
    return "hello world"


print(hello())  # returns <b><i>hello world</i></b>
```

>>>

```
    <b><i>hello world</i></b>
```

Il est possible aussi de définir des propriétés ou **properties**. Cela permet de séparer l'affectation de l'accès à un membre d'une classe sans changer la notation :

<<<

```
class C(object):
    def __init__(self):
        self._p = 1

    @property
    def p(self):
        return self._p

    @p.setter
    def p(self, val):
        self._p = val * 2


obj = C()
print(obj.p)  # utilise p_get, affiche 1
obj.p = 5      # utilise p_set
print(obj.p)  # utilise p_get affiche 10
```

>>>

```
    1
    10
```

### 12.1.7 Fichiers

L'écriture et la lecture dans un fichier s'effectuent toujours de la même manière. On ouvre le fichier en mode écriture ou lecture, on écrit ou on lit, puis on ferme le fichier, le laissant disponible pour une utilisation ultérieure. Ce paragraphe ne présente pas l'écriture ou la lecture dans un format binaire car celle-ci est peu utilisée dans ce langage.

#### Ecriture dans un fichier texte

L'écriture dans un fichier texte s'effectue toujours selon le même schéma :

```
f = open ("nom-fichier", "w") # ouverture en mode écriture "w" ou écriture ajout "a"

f.write (  s )                # écriture de la chaîne de caractères  s
f.write (  s2 )               # écriture de la chaîne de caractères  s2
...

f.close ()  # fermeture
```

Certains codes sont fort utiles lors de l'écriture de fichiers texte :
- `\n` : passage à la ligne \
- `\t` : insertion d'une tabulation, indique un passage à la colonne suivante dans le logiciel Excel

**Version 3.x :** une autre écriture est possible qui permet d'éviter l'appel à la méthode `close`.

```
with open ("nom-fichier", "w") as f :
    f.write (  s )
    f.write (  s2 )
```

L'usage d'un encoding est fréquent lorsqu'on manipule des fichiers issus d'Internet. Le plus répandu est `utf8`. Il est spécifié en-tête des pages web téléchargées. L'exemple qui suit n'est valable qu'avec la version 3. Il est recommandé de l'utiliser dès qu'on manipule les encodings.

```
with open ("nom-fichier", "w", encoding = "utf8") as f :
    f.write (  s )
    f.write (  s2 )
```

### Lecture dans un fichier texte

La lecture est le symétrique de l'écriture. En voici un exemple, la seule chose qui change d'un programme à l'autre est ce qu'on fait des lignes extraites.

```
f = open ("essai.txt", "r")    # ouverture du fichier en mode lecture
l = f.readlines ()             # lecture de toutes les lignes,
                               #   elles sont placées dans une liste
f.close ()                     # fermeture du fichier

for s in l : print (s)         # on affiche les lignes à l'écran
```

**Version 3.x :** la même syntaxe avec le mot-clé `with` et l'encoding existe.

Lors le programme précédent lit une ligne dans un fichier, le résultat lu inclut le ou les caractères (`\backslash n \; \backslash r` qui marquent la fin d'une ligne. C'est pour cela que la lecture est parfois suivie d'une étape de nettoyage.

```
with open ("essai.txt", "r") as f:  # ouverture du fichier en mode lecture
    l = f.readlines ()              # lecture de toutes les lignes,
                                    # elles sont placées dans une liste placées dans␣
↪une liste

l_net = []                        # contiendra la liste nettoyée des lignes du fichier
for s in l :
    s2 = s.replace ("\n", "")     # on supprime le code de fin de ligne \n
    s2 = s2.replace ("\r", "")    # on supprime le code de fin de ligne \r
                                  #                   (Windows uniquement)
      s2 = s2.strip("\r\n")         # cette ligne est équivalente aux deux précédentes
    l_net.append (s2)             # on ajoute le résultat à la liste nettoyée
```

Les fichiers textes ont de nombreux formats, on peut citer HTML ou XML qui sont des formats à balises. Leur lecture utilise des modules comme `HTMLParser` ou `xml.sax` dont la description sort du cadre de ce document. Un autre format est souvent utilisé avec le logiciel *Excel*. Lorsqu'on enregistre une feuille de calcul sous format texte, le fichier obtenu est organisé en colonnes : sur une même ligne, les informations sont disposées en colonnes délimitées par un séparateur qui est souvent une tabulation (`\t`) ou un point virgule.

```
nom  ; prénom ; livre
Hugo  ; Victor  ; Les misérables
Kessel ; Joseph  ; Le lion
```

(suite sur la page suivante)

(suite de la page précédente)

```
Woolf ; Virginia  ; Mrs Dalloway
Calvino ; Italo  ; Le baron perché
```

Pour lire ce fichier, il est nécessaire de scinder chaque ligne en une liste de chaînes de caractères. On utilise pour cela la méthode `split` des chaînes de caractères.

```
with open ("essai.txt", "r") as f:  # ouverture du fichier en mode lecture
    l = f.readlines ()              # lecture de toutes les lignes, placées dans une␣
↪liste

for s in l :
    s2 = s.replace ("\n", "")    # on supprime le code de fin de ligne \n
    s2 = s2.replace ("\r", "")  # on supprime le code de fin de ligne \r (Windows␣
↪uniquement)
    case = s2.split (";")
    if len (case) >= 3 :
        print (case [1], " ", case [0], " a écrit ", case [2])
```

### 12.1.8 Modules

Le concept de module permet de répartir différentes parties d'un programme sur plusieurs fichiers. Il existe deux types de modules : ceux disponibles sur Internet (programmés par d'autres) et ceux que l'on programme soi-même. Les premiers sont souvent fournis avec un programme d'installation automatique ou dans le cas où ils sont manquants, des instructions permettant de l'installer. Les seconds sont écrits dans le même répertoire que le fichier principal. On enregistre le module suivant sous le nom `geometrie.py`.

```
# définition du module geometrie.py

def carre (x) :
    return x ** 2

class point :
    def __init__ (self,x,y) :
        self.x, self.y = x,y

    def norme (self) :
        return (self.x ** 2 + self.y ** 2) ** 0.5
```

Pour utiliser une fonction ou une classe du module `geometrie.py`, on utilise une des syntaxes suivantes :

— Première syntaxe :

```
import geometrie
print (geometrie.carre (1.5))
p = geometrie.point (1,2)
```

— Deuxième syntaxe :

```
import geometrie as GEO  # on donne un pseudonyme au module geometrie
print (GEO.carre (1.5))
p = GEO.point (1,2)
```

— Troisième syntaxe : le module est utilisé très souvent, même un pseudonyme est trop long, il faut néanmoins s'assurer que les modules importés de cette même manière n'incluent pas des fonctions ou classes portant des noms identiques. Dans ce cas, c'est toujours le dernier qui gagne.

```
from  geometrie import *
print (carre (1.5))
p = point (1,2)
```

Dans le cas des modules installés, les trois syntaxes d'utilisation sont aussi valables. On voit aussi souvent apparaître dans un module la condition :

```
if __name__ == "__main__" :
    # quelques instructions ici
```

Ces instructions ne sont exécutées que si le module est utilisé en tant que programme principal. Lorsque ce fichier est importé, elles ne sont jamais exécutées. Cela permet d'écrire des instructions qui permettent de vérifier si le module ne contient pas d'erreurs. Une fois cette étape effectuée, il ne sert à rien de la répéter à chaque fois que le module est importé. C'est pourquoi elles ne sont exécutées que si la condition `if __name__ == "__main__" :` est vérifiée, c'est-à-dire si le module est le programme principal et non un module.

### 12.1.9 Exceptions

Le petit programme suivant déclenche une erreur parce qu'il effectue une division par zéro.

```
def inverse (x):
    y = 1.0 / x
    return y
b = inverse (0)
print (b)
```

Il déclenche une erreur ou ce qu'on appelle une *exception*.

```
Traceback (most recent call last):
  File "cours.py", line 2, in ?
    y = 1.0 / x
ZeroDivisionError: float division
```

Le mécanisme des exceptions permet au programme de ńărattraperă" les erreurs, de détecter qu'une erreur s'est produite et d'agir en conséquence afin que le programme ne s'arrête pas :

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    b = inverse(0)  # déclenche une exception
    print(b)
except:
    print("le programme a déclenché une erreur")
```

>>>

```
    le programme a déclenché une erreur
```

On protège la partie du code à l'aide des mots-clés `try` et `except`. Entre ces deux instructions, s'il se produit une erreur, le programme passe immédiatement à ce qui suit l'instruction `except`. On peut même récupérer le message d'erreur correspondant :

<<<

```
def inverse(x):
    y = 1.0 / x
    return y


try:
    print(inverse(2))
    print(inverse(0))
except Exception as exc:
    print("exception de type ", exc.__class__)
    # affiche exception de type  exceptions.ZeroDivisionError
    print("message ", exc)
    # affiche le message associé à l'exception
```

>>>

```
    0.5
    exception de type  <class 'ZeroDivisionError'>
    message  float division by zero
```

On peut aussi décider que le programme agira différemment selon l'erreur produite. Dans l'exemple suivant, le programme teste d'abord si l'erreur est de type `ZeroDivisionError` auquel cas il affiche le message *division par zéro*. Pour un autre type d'erreur, il regarde s'il y a d'autres instructions `except` qui s'y rapportent. S'il y en a une, il exécute les lignes qui la suivent, sinon, le programme s'arrête et déclenche une erreur.

```
def inverse (x):
    y = 1.0 / x
    return y
try :
    print ((-2.1) ** 3.1)
    print (inverse (2))
    print (inverse (0))
except ZeroDivisionError:
    print ("division par zéro")
except Exception as exc:
    print ("erreur insoupçonnée : ", exc.__class__)
    print ("message ", exc)
```

Les instructions `try` et `except` peuvent apparaître dans le programme principal, dans une boucle, un test, une fonction, s'imbriquer les unes dans les autres. Il est possible de déclencher soi-même une exception avec l'instruction `raise` et ou de définir ses propres exceptions en créant une classe héritant d'une classe d'exception. L'exemple suivant regroupe tous ces cas.

<<<

```
class AucunChiffre (Exception):
    """chaîne de caractères contenant
    aussi autre chose que des chiffres"""

    def __init__(self, s, f=""):
        Exception.__init__(self, s)
        self.s = s
        self.f = f

    def __str__(self):
        return """exception AucunChiffre, lancée depuis la fonction """ + self.f + \
            " avec le paramètre " + self.s
```

(suite sur la page suivante)

(suite de la page précédente)

```
def conversion(s):
    """conversion d'une chaîne de caractères en entier"""
    if not s.isdigit():
        raise AucunChiffre, (s, "conversion")
    return int(s)


try:
    s = "123a"
    i = conversion(s)
    print(s, " = ", i)
except AucunChiffre as exc:
    print(AucunChiffre.__doc__, " : ", exc)
    print("fonction : ", exc.f)
```

>>>

```
    [runpythonerror]
    File "<stdin>", line 21
        raise AucunChiffre, (s, "conversion")
                            ^
    SyntaxError: invalid syntax
```

### 12.1.10 Erreurs, confusions fréquentes

#### Variables

#### Chaîne de caractères = tableau de caractères

Une chaîne de caractères est un tableau de caractères : pour accéder à un caractère, on procède comme pour une liste.

<<<

```
s = "abcdefghijklmnopqrstuvwxyz"
print(s[4])    # affiche "e"
print(s[4:6])  # affiche "ef"
```

>>>

```
    e
    ef
```

#### Guillemets ou pas

Doit-on mettre des guillemets ou non~ ?

```
l  = [ un, deux, trois, quatre ]
up = []
for i in range (0, len (l)) :
    up.append ( l [i].upper () )
```

Le code précédent ne fonctionne pas car il n'y a pas de guillemets autour de `un`, `deux`, `trois`, `quatre`. Le langage considère alors ces quatre mots comme des variables : un identificateur qui désigne une information. Mais comme ces variables n'existent pas, ces identifiants ne sont reliés à aucun contenu et l'interpréteur python ne comprend pas.

Un mot entouré de guillemets (ou d'apostrophes) définit un contenu. Sans guillemet (ou apostrophe), il définit une variable qui permet de manipuler un contenu tout simplement en donnant la possibilité au programmeur de le nommer. Autrement dit, pour manipuler une chaîne de caractères, il faut affecter ce contenu à une variable. Les guillemets n'apparaissent plus par la suite car on doit utiliser la variable pour la manipuler.

### Boucles

#### `range` ou pas `range`

Les deux programmes suivant sont équivalents. La seule différence réside dans l'écriture dans la boucle `for` qui utilise dans le premier cas la fonction `range` et dans l'autre non.

```
l  = [ "un", "deux", "trois", "quatre" ]
up = []
for i in range (0, len(l)) :
    up.append ( l [i].upper() )
```

Lorsqu'on utilise la fonction `range`, on dispose lors de la boucle de deux informations, l'indice `i` et l'élément `l [i]`. Si l'indice n'est pas utile, il est possible de simplifier la boucle comme suit.

```
l  = [ "un", "deux", "trois", "quatre" ]
up = []
for m in l :
    up.append ( m.upper() )
```

En général, on se sert de la boucle qui utilise la fonction `range` dans deux cas :

1. On souhaite faire des opérations sur les éléments qui précèdent ou suivent l'élément en question, ce qui nécessite de connaître l'indice.
2. On parcourt deux listes de même taille à la fois : l'indice désigne la position de deux éléments, un dans chaque liste.

### Initialisation

Une boucle est souvent utilisée pour faire une somme, calculer un maximum : garder un unique résultat en parcourant une liste. Une boucle de ce type est toujours précédée d'une étape d'initialisation qui consiste à donner une valeur au résultat : celle qu'il aurait si la liste était vide.

```
l = [ "un", "deux", "trois", "quatre" ]
s = ""
for m in l :
    s += m # concaténation des mots en une seule chaîne de caractères
```

### Fonctions

#### Différence entre `print` et `return`

A la fin d'un calcul, afin de voir son résultat, on utilise souvent l'instruction `print`. On peut se demander alors si à la fin de chaque fonction, il ne faudrait pas utiliser l'instruction `print`. A quoi servirait alors l'instruction `return` ? On suppose qu'un calcul est en fait le résultat de trois calculs à la suite :

```
a = calcul1 (3)
b = calcul2 (a)
c = calcul3 (b) # c résultat souhaité et affiché
```

Chaque terme `calculx` cache une fonction or seul le résultat de la dernière nous intéresse et doit être affiché. Pour les deux premières, la seule chose importante est que leur résultat soit transmis à la fonction suivante et ceci ne peut se faire que grâce à l'instruction `return`. L'instruction `print` insérée dans le code de la fonction `calcul1` ou `calcul2` permettra d'afficher le résultat mais ne le transmettra pas : l'instruction `return` est donc indispensable, `print` facultative.

En revanche, dans la dernière fonction `calcul3`, il est possible de se passer de `return` et de se contenter uniquement d'un `print`. Cependant, il est conseillé d'utiliser quand même `return` au cas où le résultat de la fonction `calcul3` serait utilisé par une autre fonction, `calcul4` par exemple.

<<<

```
def calcul1(x):
    return x+3


y = calcul1(4)
print(y)            # affiche None
# car la fonction calcul1 ne retourne pas de résultat, elle l'affiche
```

>>>

```
    7
```

Cela peut provoquer des erreurs lorsqu'on essaye d'utiliser ce résultat dans un calcul par la suite.

```
def calcul1(x):
    print (x+3)
def calcul2(x):
    return calcul1(x) + 5
y = calcul2(4)      # affiche l'erreur
```

```
ported operand type(s) for +: 'NoneType' and 'int'
```

Il faut retenir que l'instruction `print` n'a aucun impact sur le résultat d'un programme.

### 12.1.11 Trois conseils pour écrire un programme

#### Des petites fonctions

Pour plusieurs raisons :

1. Il est plus facile de corriger un programme qui est constitué de petites fonctions plutôt que de quelques grandes. Chaque fonction peut être vérifiée séparément.
2. Il est plus facile de réutiliser des petites fonctions.
3. Il est plus facile de répartir le travail sur plusieurs personnes.

Il vaut mieux éviter les variables globales qui sont considérées que comme des paramètres cachés.

#### Séparer les calculs, le chargement des données, l'interface graphique

Pour plusieurs raisons :

1. Il est plus facile de vérifier un calcul s'il est dans une fonction indépendante plutôt que caché dans le code d'une interface graphique.
2. C'est facile de faire un calcul une fois lorsqu'un utilisateur appuie sur un bouton, si on veut faire ce calcul cent fois, on ne peut pas lui demander d'appuyer cent fois sur le même bouton.
3. Les calculs ou le chargement des données peuvent être utilisés dans d'autres programmes.

#### Utiliser des tests unitaires

Ces fonctions peuvent être exécutées au début du programme pour vérifier que certaines parties du programme fonctionnent toujours même après les avoir modifiées.

L'exemple suivant considère une fonction qui doit retourner une somme réelle même si les éléments de la liste sont entiers. On écrit la fonction qui vérifie cela.

```
def somme_double(liste):
    return 1.0 * sum(liste)

def test_somme_double():
    y = somme_double([ 1 ]) / 2
    if y == 0:
        raise Exception ("valeur > 0 attendue")

if __name__ == "__main__" :
    test_somme_double()
```

Si plus tard, quelqu'un modifie la fonction `somme_double` en enlevant la multiplication parce qu'il considère cela inutile. La fonction de test provoquera une erreur. Elle est là pour rappeler que la fonction a été programmée pour retourner un nombre réel et que quiconque l'utilise s'attend à ce qu'elle retourne ce type de résultat.

```
Traceback (most recent call last):
  File "conseil.py", line 10, in <module>
    test_somme_double()
  File "conseil.py", line 7, in test_somme_double
    if y == 0 : raise Exception ("valeur > 0 attendue")
Exception: valeur > 0 attendue
```

### 12.1.12 Trucs et astuces

#### Partager du code

Il existe aujourd'hui des solutions qui permettent d'éviter les envois de programme par email. Des outil comme *DropBox*, *SkyDrive*, *GoogleDrive* permettent de partager un répertoire. Un même répertoire peut être partagé sur plusieurs ordinateurs et plusieurs personnes. Une modification (y compris une suppression) sur l'une des répliques sera propagée sur tous les ordinateurs dès qu'ils sont connectés à Internet.

Il est possible de coupler cette solution avec *SVN* ou *TortoiseSVN* qui sont des logiciels de suivis de source. On garde à la fois la dernière version et l'historique des modifications.

#### Moteurs de recherche

Lorsqu'on ne comprend un message d'erreur, il est souvent utile de recopier le texte dans un moteur de recherche (Google, Bing, ...). Il est très rare de ne pas réussir à trouver d'indices.

## 12.2 Quelques conseils pour écrire un programme

### 12.2.1 Des petites fonctions

Pour plusieurs raisons :

— Il est plus facile de corriger un programme qui est constitué de petites fonctions plutôt que de quelques grandes. Chaque fonction peut être vérifiée séparément.
— Il est plus facile de réutiliser des petites fonctions.
— Il est plus facile de répartir le travail sur plusieurs personnes.

### 12.2.2 Pas de variables globales

Il vaut mieux éviter les variables globales qui sont considérées que comme des paramètres cachés. Il faut réécrire le programme dès qu'on veut répéter le même code avec différentes valeurs pour cette variable globale.

```
vglob = 0

def un_calcul(a=0):
    global vglob
    return a + vglob

loops = [un_calcul(a) for a in range(0, 10)]
```

Comment faire une boucle sur la valeur de `vglob` ? Quelque soit la solution choisie, faire une boucle sur la variable globale requiert quelques contorsions. Les variables globales sont à éviter avec les threads.

### 12.2.3 Séparer les calculs, le chargement des données, l'interface graphique

Pour plusieurs raisons :

— Il est plus facile de vérifier un calcul s'il est dans une fonction indépendante plutôt que caché dans le code d'une interface graphique.
— C'est facile de faire un calcul une fois lorsqu'un utilisateur appuie sur un bouton, si on veut faire ce calcul cent fois, on ne peut pas lui demander d'appuyer cent fois sur le même bouton.
— Les calculs ou le chargement des données peuvent être utilisés dans d'autres programmes.

### 12.2.4 Utiliser des fonctions de tests

Ces fonctions peuvent être exécutées au début du programme pour vérifier que certaines parties du programme fonctionnent toujours même après les avoir modifiées.

L'exemple suivant considère une fonction qui doit retourner une somme réelle même si les éléments de la liste sont entiers. On écrit la fonction qui vérifie cela.

```
def somme_double (liste) :
    return 1.0 * sum(liste)

def test_somme_double () :
    y = somme_double([ 1 ]) / 2
    if y == 0 : raise Exception ("valeur > 0 attendue")

if __name__ == "__main__" :
    test_somme_double()
```

Si plus tard, quelqu'un modifie la fonction `somme_double` en enlevant la multiplication parce qu'il considère cela inutile. La fonction de test provoquera une erreur. Elle est là pour rappeler que la fonction a été programmée pour retourner un nombre réel et que quiconque l'utilise s'attend à ce qu'elle retourne ce type de résultat.

```
Traceback (most recent call last):
  File "conseil.py", line 10, in <module>
    test_somme_double()
  File "conseil.py", line 7, in test_somme_double
    if y == 0 : raise Exception ("valeur > 0 attendue")
Exception: valeur > 0 attendue
```

### 12.2.5 Trucs et astuces

**Moteurs de recherche**

Lorsqu'on ne comprend un message d'erreur, il est souvent utile de recopier le texte dans un moteur de recherche (Google, Bing, ...). Il est très rare de ne pas réussir à trouver d'indices.

**Partager du code**

Il existe aujourd'hui des solutions qui permettent d'éviter les envois de programme par email. Le plus utilisé pour les librairies open source est GitHub [609].

*Autres liens*

- Rappel de programmation [610]
- Cheat Sheets [611]
- Python Numpy Tutorial [612]
- 100 numpy exercises [613]
- Matplotlib tutorial [614]
- Scipy Lecture Notes [615]

*Autres langages*

- C++ crash course for C programmers [616]

609. https://github.com/
610. http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx3/td_2a.html#rappels-de-programmation
611. http://www.xavierdupre.fr/app/ensae_projects/helpsphinx/cheat_sheets.html
612. http://cs231n.github.io/python-numpy-tutorial/
613. http://www.labri.fr/perso/nrougier/teaching/numpy.100/
614. http://www.labri.fr/perso/nrougier/teaching/matplotlib/matplotlib.html
615. http://www.scipy-lectures.org/
616. http://www.labri.fr/perso/nrougier/teaching/c++-crash-course/index.html

CHAPITRE 13

# Index