ENSAM Bordeaux – mars 2020

Calcul Scientifique avec

python™ 3

- ▷ Introduction - Prise en main
- ▶ Les bases du langage Python
- ▷ I/O fichiers ASCII, tracés de courbes
- ▷ Les modules Numpy et Scipy
- ▷ Conception/programmation OO

Jean-Luc Charles - Éric Ducasse

Arts & Métiers, I2M

Arts et Métiers Sciences et Technologies

I2M Bordeaux

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 02'23]

Le langage Python

- Le langage Python est constitué :
  - ▷ de **mots clefs**, qui correspondent à des instructions élémentaires (`for`, `if`...) ;
  - ▷ de **littéraux** : valeurs constantes de types variés (`25`, `1.e4`, `'abc'`...) ;
  - ▷ de **types intrinsèques** (`int`, `float`, `list`, `str`...) ;
  - ▷ d'**opérateurs** (`=`, `+`, `*`, `/`, `%`...) ;
  - ▷ de **fonctions intrinsèques** (*Built-in Functions*) qui complètent le langage.

- L'utilisateur peut créer :
  - ▷ des **classes** : nouveaux types qui s'ajoutent aux types intrinsèques ;
  - ▷ des **objets** : entiers, flottants, chaînes, fonctions, programmes, modules... instances de classes définies par l'utilisateur ;
  - ▷ des **expressions** combinant identificateurs, opérateurs, fonctions...

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 04'57]

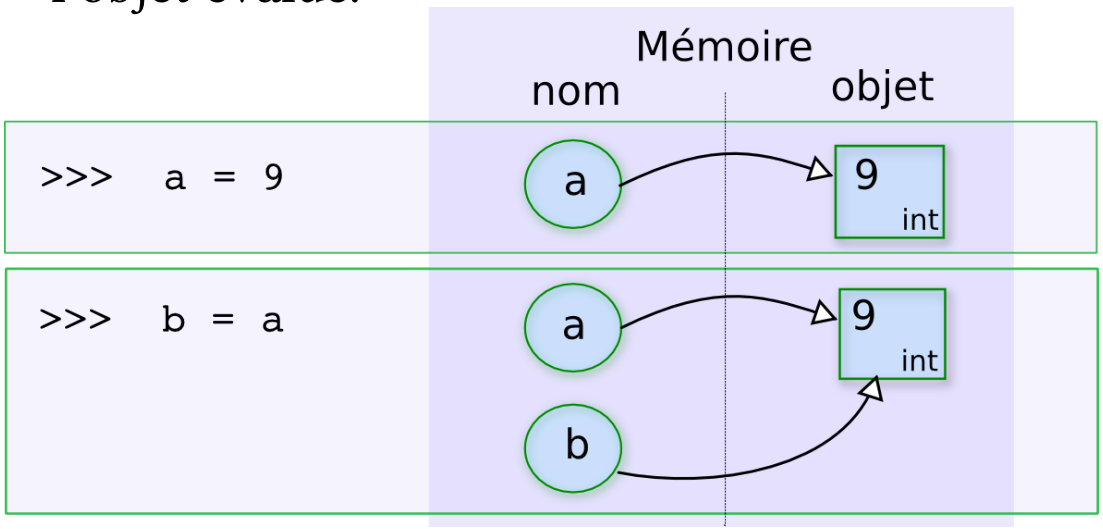
Python est un langage **Orienté Objet**

- Avec Python, tout est **objet** : données, fonctions, modules...
- Un **objet** :
  - possède une **identité** (≈ adresse mémoire) ;
  - possède un **type** : un objet est l'instanciation d'une **classe** qui définit son type (type intrinsèque : `int`, `float`, `str`... ou type utilisateur : `class xxx`) ;
  - contient des **données** (exemple : objet numérique -> sa **valeur**).
- Un objet est **référencé** par un **identificateur** :
  - **identificateur**, **référence**, **nom**, **étiquette**... sont des termes synonymes ;
  - **objet** et **variable** sont des termes synonymes (on peut préférer *objet*) ;
  - les noms au format `__xxx__` ont une signification spéciale pour l'interpréteur Python.

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 07'52]

Affectation : référence ⇝ objet

- L'affectation opère de droite à gauche :
  - le terme de droite est une **expression**, qui est **évaluée** en tant qu'objet ;
  - le terme de gauche est l'**identificateur** (**référence**, **nom**, **étiquette**) affecté à l'objet évalué.



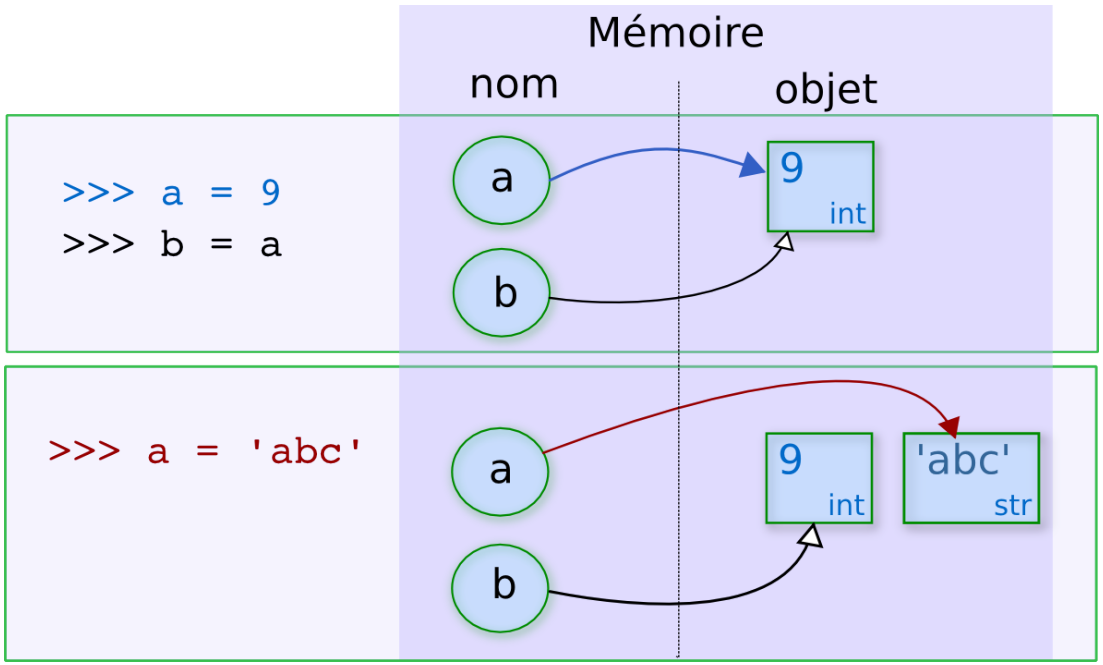
C'est l'objet qui porte le **type** et les **données** (la **valeur**, pour un objet numérique).

- Un objet peut avoir plusieurs noms (*alias*).
- Un objet ne peut pas changer d'identité (≈ adresse mémoire), ni de type.

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 11'54]

Affectation : référence ⇝ objet

- Un identificateur associé d'abord à un objet peut ensuite référencer un nouvel objet !



- Quand un objet n'a plus de nom (nombre de références nul), il est détruit (mécanisme automatique de "ramasse-miettes", *garbage collector*).

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 13'21]

Objets

- L'affectation = permet d'affecter un **nom** à un objet **créé** ou **existant**.
- `type(•)` renvoie le type de l'objet •.
- `id(•)` renvoie l'identité de l'objet •.
- L'opérateur == teste l'égalité des **valeurs** de 2 objets.
- L'opérateur `is` compare l'**identité** de 2 objets.

```
>>> a = 999  # objet 999, int, nommé a
>>> type(a)
<class 'int'>
>>> b = a   # 2eme nom pour l'objt 999
>>> id(a), id(b)
(33724152, 33724152)
>>> type(b)
<class 'int'>
```

```
>>> c = 999.
>>> id(a), id(b),id(c)
(33724152, 33724152, 38603664)
>>> a == c     # égalité des valeurs ?
True
>>> a is c     # même identité ?
False
>>> a is b     # même identité ?
True
```

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 17'00]

Objets

- Le type d'un objet détermine :
  - ▷ les valeurs possibles de l'objet ;
  - ▷ les opérations que l'objet supporte ;
  - ▷ la signification des opérations supportées (opérations *polymorphes*).

```
>>> a = 2 + 3  # addition d'entiers
>>> a
5
>>> 2*a, 2.*a  # mult. dans N, R
(10, 10.0)
>>> 2/3, 2./3. # divis. dans  R
(0.6666666666666666, 0.666666666...)
```

```
>>>[1, 2]+[5, 6] # concat. des listes
[1, 2, 5, 6]
>>> 2*[5, 6]     # n concaténations
[5, 6, 5, 6]
>>> [5, 6]/2     # pas défini !
...unsupported operand type(s) for /
'list' and 'int'
```

- Un objet dont le contenu peut être changé est dit *mutable* (*non mutable* sinon).

```
>>> a = "Un str est non-mutable !"
>>> a[0]
'U'
>>> a[0] = "u"
...TypeError: 'str' object does
  not support item assignment
```

```
>>> a = [1, 2, 3]  # liste: mutable
>>> a[1]
2
>>> a[1] = 2.3
>>> a
[1, 2.3, 3]
```

[Audio – Python3-2-LesBases-v1.4_ObjetsClasse.mp3 : 24'15]

Classes et Objets

- `dir(•)` permet d'afficher "ce qu'il y a dans" • (classe, objet, module...).
- On peut utiliser toutes les méthodes (publiques) d'une classe sur un objet instance de la classe : `objet.methode(...)`

```
>>> L1 = [4, 2, 1, 3] # L1 est un objet liste (instance de la classe liste)
>>> type(L1)
<class 'list'>
>>> dir(L1)
['__add__',..., 'append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
>>> dir(list)
 ['__add__',..., 'append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
>>> L1.sort() ; L1          # objet.méthode_de_la_classe
[1, 2, 3, 4]
>>> L1.append(5) ; L1       # objet.méthode_de_la_classe
[1, 2, 3, 4, 5]
>>> L1.reverse() ; L1       # objet.méthode_de_la_classe
[5, 4, 3, 2, 1]
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 00'00]

Types intrinsèques (*built-in types*)

- Type `NoneType` : la seule valeur possible est `None` (≈ objet nul ou vide).

```
>>> a = None
```

▷ c'est la valeur retournée par une fonction qui "ne renvoie rien".

- Type `bool` (booléen) : les valeurs possibles sont `True` ou `False` (1, 0).

```
>>> b = True
```

▷ la valeur logique de `None` est `False`.

- Types numériques : `int`, `float`, `complex`...

```
>>> c = 123          # int : entier 32 bits, dans [-2147483648, 2147483647]
>>> c = 2**100
1267650600228229401496703205376  # entier en 'précision arbitraire'
>>> d = 1.47e-7      # float : flottants IEEE754 64 bits,
                     #   ~16 chiffres significatifs
                     #   ~10**-308 < valeur absolue < ~10**308
>>> e = 2.1 +3.45j # complex
>>> e.real, e.imag
(2.1, 3.45)
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 04'00]

Types intrinsèques (*built-in types*)

- Types `list`, `tuple`, `str` : les **séquences** (collections ordonnées) :

```
>>> f = [1.e4, 2e4, 0, 1, e1]
>>> g = (1, 2, 3, e1, f)
>>> s1 = 'l"impact'
>>> s2 = "l'impact"
>>> s3 = "Chaîne accentuée"
```

  - Objet `list` entre `[...]`
  - Objet `tuple` entre `(...)`
  - Objet `str` entre `'...'` ou `"..."`

- Type `dict` (dictionnaire) : **collection non ordonnée** de **paires** (`clef, objet`)

```
>>> d1 = {"Lundi":1, "Mardi":2}
>>> d2 = {"mean":1.2, "stdDev":9.4}
```

  - Objet `dict` : paires `key:value` entre `{...}`

- Type `set` (ensemble) : **collection non ordonnée d'items uniques**

```
>>> s = set([1,5,12,6,"a"])
>>> s = {1,5,12,6,"a"}
```

  - Un `set` est créé avec : `set(...)`
    ou avec les éléments entre `{...}`

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 08'38]

Types intrinsèques conteneurs

- Collections ordonnées : les séquences (*sequences*)
  - les listes `list`
  - les tuples `tuple`
  - les chaînes de caractères `str`
- Collections sans ordre
  - les dictionnaires `dict`
  - les ensembles `set`

Tous les conteneurs Python sont des objets **itérables** : on peut les parcourir avec une boucle `for`.

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 09'38]

Séquences : la Classe `list`

- Collection **ordonnée** d'**objets quelconques** (conteneur hétérogène).
- Une liste n'est pas un vecteur (⇝ classe **ndarray** du module **numpy**).
- `len(•)` renvoie le nombre d'éléments de la liste •.
- **Indexation** : •[i] renvoie l'élément de rang `i` de la liste •
  - ▷ le premier élement de la liste • a le rang 0, le dernier le rang `len(•)-1`.

```
>>> L1 = [10,11,12,13]
>>> L1.append(14)
>>> L1
[10, 11, 12, 13, 14]
>>> L1[0]
10
>>> L1[-1]
14
>>> len(L1)
5
```

```
>>> L1[4]
14
>>> L1[5]
...IndexError: list index out of range
>>> L1[-4]
11
>>> L1[-5]
10
>>> L1[-6]
...IndexError: list index out of range
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 13'00]

Séquences : la Classe `list`

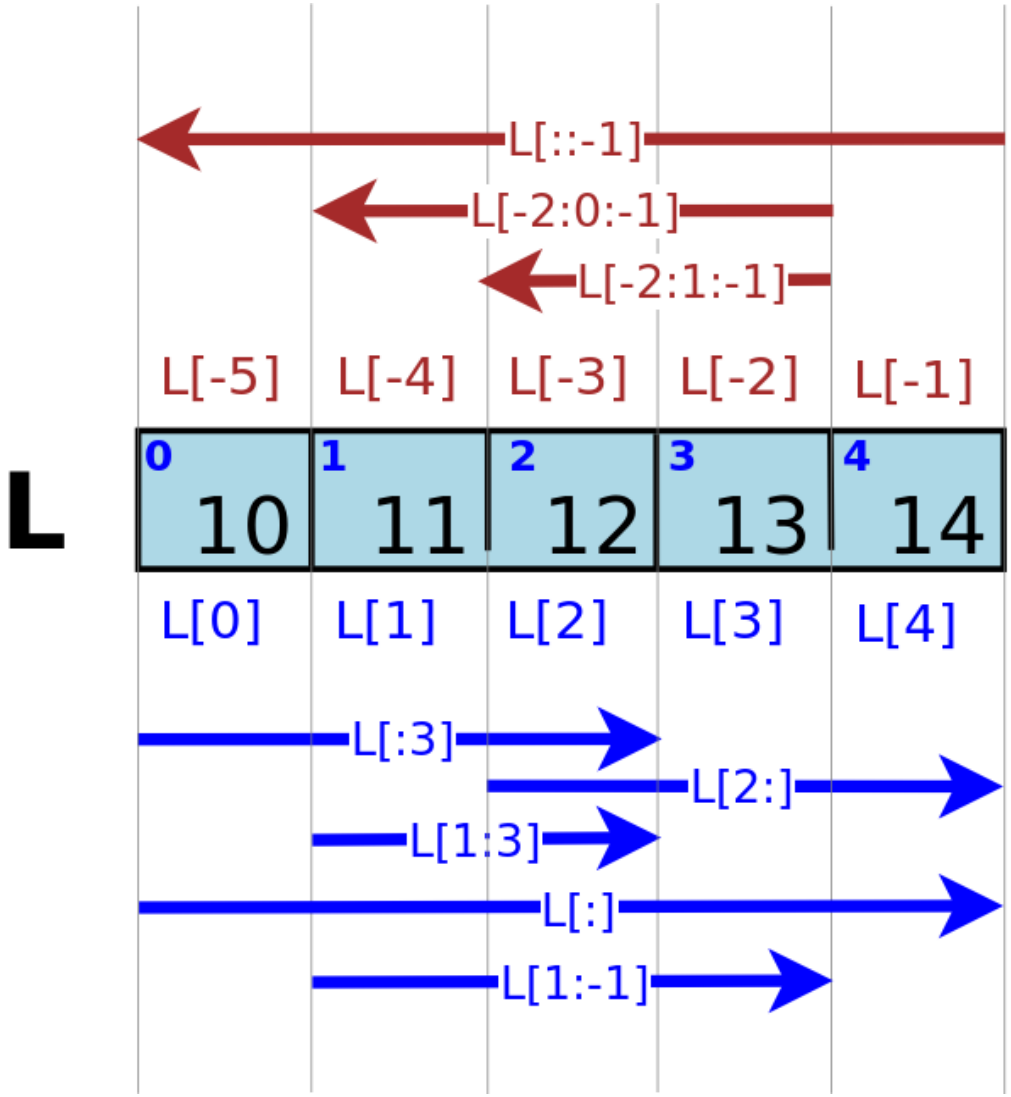
- **Sous-liste** (*Slicing*) :
  - `•[i:j]` ⇝ sous-liste de `i` inclus à `j` **exclu**
  - `•[i:j:k]` ⇝ sous-liste de `i` inclus à `j` **exclu**, par pas de `k`
- Indexation positive, **négative** :
  - `0 ⩽ i ⩽ len(•)-1` ⇝ rang dans la liste
  - `-len(•) ⩽ i ⩽ -1` ⇝ `-1` : le dernier, `-2` : l'avant dernier...

```
>>> L1 = [10,11,12,13,14]
>>> L1[1:3]
[11, 12]
>>> L1[:3]
[10, 11, 12]
>>> L1[1:]
[11, 12, 13, 14]
>>> L1[:] # toutes les valeurs de L1
[10, 11, 12, 13, 14]
```

```
>>> L1 = [10,11,12,13,14]
>>> L1[::2]
[10, 12, 14]
>>> L1[::-1]
[14, 13, 12, 11, 10]
>>> L1[-2:1]
[]
>>> L1[-2:1:-1]
[13, 12]
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 17'28]

Séquences : la Classe `list`



[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 17'44]

Séquences : la Classe `list`

Copie : Référence, copie superficielle ou copie profonde

`L2 = L1` nom supplémentaire L2 pour l'objet nommé L1

```
>>> L1 = [10, 11, 12] ; L2 = L1      # L1 et L2 : 2 noms d'un même objet
>>> id(L1), id(L2), L2 is L1
(50660560, 50660560, True)
>>> L2[0] = 0
>>> L1, L2
([0, 11, 12], [0, 11, 12])
```

Copie : Référence, copie superficielle ou copie profonde

`L3 = L1[:]` nouvel objet `list` nommé L3, initialisé par L1 (*shallow copy*)

```
>>> L1 = [10, [11, 12]] ; L3 = L1[:]     # Shallow copy !
>>> id(L1), id(L3), L3 is L1
(50660560, 51420768, False)
>>> L3[0] = 0; L3[1][0] = -1
>>> L1, L3
([10, [-1, 13]], [0, [-1, 13]])
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 17'44]

Séquences : la Classe `list`

⚠ **Copie** : **Référence**, **copie superficielle** ou **copie profonde**

```
from copy import deepcopy
L4 = deepcopy(L1)
```

nouvel objet `list` nommé L4, copie complète de l'objet L1 (*deep copy*)

```
>>> L1 = [10, [11, 12]]
>>> from copy import deepcopy
>>> L4 = deepcopy(L1)        # Deep copy !
>>> id(L1), id(L4), L4 is L1
(53980336, 53982200, False)
>>> L4[0] = 0
>>> L4[1][0] = -1
>>> L1, L4
([10, [12, 13]], [0, [-1, 13]])
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 19'27]

Séquences : la Classe `list`

- Une liste est **itérable**, **on peut la parcourir avec une boucle `for`** :

```
>>> L1 = [1, 2, 3]
>>> for a in L1:
    print(a+2)

3
4
5
```

```
>>> col =["red", "green", "blue"]
>>> for i, c in enumerate(col):
    print(i, "Color: " + c)

0 Color: red
1 Color: green
2 Color: blue
```

- L'opérateur + concatène les listes :

```
>>> L1 = [1, 2, 3]
>>> L2 = [4, "a", 5]
>>> L1 + L2
[1, 2, 3, 4, 'a', 5]
```

- L'opération `list*n` ou `n*list` concatène `n` copies de la liste :

```
>>> [0]*10
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 24'22]

Séquences : la Classe `list`

- La classe `list` possède des méthodes utiles
  (*cf* `dir(list)` et `help(list.xxxxx)`...)

```
>>> help(list.remove)
Help on method_descriptor:
remove(...)
    L.remove(value) -> None -- remove first occurrence of value.
    Raises ValueError if the value is not present.
```

```
>>> L1 = [10, 11, 12 , 13, 14]
>>> del L1[2]
>>> L1
[10, 11, 13, 14]
>>> L1.remove(11)
>>> L1
[10, 13, 14]
>>> L2 = [10, 11, 10, 13, 10]
>>> L2.count(10)
3
```

```
>>> L2.remove(10)
>>> L2
[11, 10, 13, 10]
>>> L2.remove(10)
>>> L2
[11, 13, 10]
>>> L2.remove(10)
>>> L2
[11, 13]
>>> L2.remove(10)
... list.remove(x): x not in list
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 27'43]

Séquences : la Classe `tuple`

- Un tuple est une liste **non mutable** :
  - ses **éléments** ne supportent pas la **ré-affectation** ;
  - mais ses **éléments mutables** peuvent être **modifiés**.

```
>>> a  = ()      # tuple vide
>>> b  = (2,)    # 1-tuple
>>> c  = 2,      # 1-tuple
>>> d  = 2, 3, 4 # tuple implicite
>>> d
(2, 3, 4)
>>> t=(1,"non",[1,2,3])
>>> t
(1, 'non', [1, 2, 3])
>>> t[0]=2 # ré-affectation élément
...TypeError: 'tuple' object does
not support item assignment
```

```
>>> t[1]="oui"  # ré-affection élément!
...TypeError: 'tuple' object does not
support item assignment
>>> t[1][0]="N" # élément str non mutable
...TypeError: 'str' object does not
support item assignment
>>> t[2]=[3,4,5] # ré-affection élément
...TypeError: 'tuple' object does not
support item assignment
>>> t[2][0]=-3 # élément list mutable
>>> t
(1, 'non', [-3, 2, 3])
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 29'09]

Séquences : la Classe `str`

- Chaîne de caractères = **liste** de caractères...
- Même principe d'indexation que les objets `list` :

```
>>>  s = "abcdef"
>>> s[0], s[-1]
('a', 'f')
>>> s[:]
'abcdef'
>>> s[1:-1], s[1:-1:2]
('bcde', 'bd')
```

- De nombreuses méthodes utiles sont proposées par la classe `str` :

```
>>> dir(str)
[... 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum',
'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip',  'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 32'52]

Collection non-ordonnée : la Classe `dict`

- Collection sans ordre de paires `clef`, `objet` : sert à retrouver un objet par sa clef

```
>>> cyl1 = {"L": 1.2, "D": 0.5, "unit": "m"} ; cyl1
{'L': 1.2, 'unit': 'm', 'D': 0.5}
>>> cyl1["L"]
1.2
>>> cyl1["D"] = 0.6 ; cyl1
{'D': 0.6, 'L': 1.2, 'unit': 'm'}
```

- La classe `dict` possède plusieurs méthodes utiles (`dir(dict)` et `help(dict.xxx)`...)

```
>>> cyl1["d"] = 0.1 ; cyl1    # création de la clef !
{'d':0.1, 'L': 1.2, 'unit': 'm', 'D': 0.6}
>>> "mass" in cyl1, "D" in cyl1
(False, True)
>>> del cyl1["d"]
>>> list(cyl1.items())
[('L', 1.2), ('unit', 'm'), ('D', 0.6)]
>>> list(cyl1.keys())
['L', 'unit', 'D']
>>> list(cyl1.values())
[1.2, 'm', 0.6]
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 35'23]

Collection non-ordonnée : la Classe `dict`

- Un dictionnaire est **itérable** : **on peut le parcourir avec une boucle `for`.**

```
>>> for k in cyl1:
        print(k)
L
unit
D
>>> for k, v in cyl1.items():
        print(k, v)
L 1.2
unit m
D 0.6
```

- méthode `get` pour spécifier la valeur à renvoyer si la clef est absente :

```
>>>  x = cyl1['age']
Traceback (most recent call last):
...
KeyError: 'age'
>>> x = cyl1.get('age', 0)
>>> x
0
```

[Audio – Python3-2-LesBases-v1.4_Types.mp3 : 37'29]

Collection non-ordonnée : la Classe `set`

- Collection non-ordonnée d'**items uniques**

```
>>> s = set([1,5,12,6])
>>> t = set('Lettres ')
>>> t
{' ', 'e', 'L', 's', 'r', 't'}
>>> u = {"a","b","b","c"} ; u
{'b', 'a', 'c'}
```

  - Les objets `set` sont créés avec le constructeur de la classe `set` et un argument de type `list`
  - On peut aussi énumérer les éléments entre accolades.

- La classe `set` propose des opérations ensemblistes

```
>>> t = set{[5, 6, 7]}
>>> s | t       # Union
{1, 5, 6, 7, 12}
>>> s & t       # Intersection
{5, 6}
>>> s - t       # dans s, mais pas dans t
{1, 12}
>>> s ^ t       # dans s ou (exclusif)  t
{1, 12, 7}
```

[Audio – Python3-2-LesBases-v1.4_MotsClef_Operateurs.mp3 : 00'00]

Mots clefs du langage

Doc Python > Language Reference > Identifiers and keywords

docs.python.org/reference/lexical_analysis.html#keywords

Il n'y a que 33 mots clefs (*key words*) dans le langage Python 3.6

2.3.1. Keywords

The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

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

[Audio – Python3-2-LesBases-v1.4_MotsClef_Operateurs.mp3 : 00'34]

Mots clefs du langage

Boucles	
`for`	boucle
`while`	boucle
`continue`	tour suivant
`break`	sortie de boucle

Définitions d'objets	
`def`	définition d'une fonction
`return`	fin fonction, renvoyer valeur
`class`	définition d'une classe
`nonlocal`	change la portée d'une variable
`global`	définit une variable globale

Tests	
`if`	test
`else`	alternative
`elif`	test combiné

Opérateurs logiques	
`and`	ET logique
`or`	OU logique
`not`	négation

[Audio – Python3-2-LesBases-v1.4_MotsClef_Operateurs.mp3 : 01'43]

Mots clefs du langage - suite -

importation d'un **Module**	
`import`	module entier
`from`	objets d'un module

Objets	
`del`	détruire une référence sur un objet
`in`	parcourir
`is`	comparer

gestion des **Exceptions**	
`assert`	définir une assertion
`raise`	créer une exception
`try`	traitement exception
`except`	traitement exception
`finally`	traitement exception

Autre	
`pass`	ne rien faire...
`with`	objet dans un context
`as`	utilisé avec `with`, `import`...
`yield`	fonction générateur
`lambda`	fonction *inline*

[Audio – Python3-2-LesBases-v1.4_MotsClef_Operateurs.mp3 : 03'01]

Les opérateurs docs.python.org/reference/lexical_analysis.html#operators

Principaux opérateurs

Opérateurs arithmétiques	
`+`	addition
`-`	soustraction
`*`	mutiplication
`/`	division (Python3 : toujours division flottante)
`//`	quotient de la division entière
`**`	exponentiation (0^0 donne 1)
`%`	modulo
`<< n, >> n`	décalage à gauche, à droite de `n` bits

[Audio – Python3-2-LesBases-v1.4_MotsClef_Operateurs.mp3 : 03'46]

Principaux opérateurs

Opérateurs de comparaison	
`<, <=`	inférieur strict à, inférieur ou égal à
`>, >=`	supérieur strict à, supérieur ou égal à
`==`	égal à
`!=`	différent de
Opérateurs logiques	
`and`	ET
`not`	négation
`or`	OU
`&`	ET bit à bit
`^`	XOR bit à bit
`|`	OR bit à bit
`~`	complément à 2

[Audio – Python3-2-LesBases-v1.4_MotsClef_Operateurs.mp3 : 04'43]

Principaux opérateurs

Autres Opérateurs	
`is`	même identité ?
`is not`	identité différente ?
`in`	appartient à ?
`not in`	n'appartient pas à ?

Opérateurs avec affectation	équivalence
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x //= y`	`x = x // y`
`x **= y`	`x = x ** y`
`x %= y`	`x = x % y`

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 00'00]

Les Fonctions Intrinsèques

docs.python.org/library/functions.html

... il y en a beaucoup, elles sont très utiles ☺

2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 00'17]

Fonctions intrinsèques (extraits)

- `abs` valeur absolue, module...
- `bin` convertir en base 2...
- `bool` convertir en booléen...
- `enumerate` parcourir un itérable en numérotant les éléments...
- `eval` interpréter en Python une chaîne de caractère...
- `float` convertir en float...
- `int` convertir en entier...
- `hex` convertir en base 16...
- `max` maximum d'un itérable ou de plusieurs objets...
- `map` appliquer une fonction aux éléments d'une liste...
- `min` minimum d'un itérable ou de plusieurs objets...
- `range` générer une suite de valeurs entières...
- `repr` créer la conversion d'un objet en chaîne de caractères...
- `reversed` parcourir un itérable en sens inverse...
- `str` convertir simplement en chaîne de caractères...
- `sum` somme d'un itérable ou de plusieurs objets...
- `zip` parcourir plusieurs itérables en même temps...

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 01'18]

Fonctions intrinsèques (extraits)

- `abs` : valeur absolue ou module

```
>>> abs(-6), abs(-6.7), abs(1+1j)
(6, 6.7, 1.4142135623730951)
```

- `bin`, `hex` : convertit un entier en base 2 ou 16

```
>>> bin(75), hex(64)
('0b1001011', '0x40')
```

- `bool`, `int`, `float` : convertit un objet en type `bool`, `int` ou `float`

```
>>> bool(0), bool(1), bool([]), bool([1]), bool([0]), bool("")
(False, True, False, True, True, False)
>>> int("5"), float("5"), float("1.1e2")
(5, 5.0, 110.0)
```

- `int` : convertit un nombre d'une base B (défaut : 10) en base 10

```
>>> int(2.34), int("45")
(2, 45)
>>> int("110011",2), int("a1",16)
(51, 161)
```

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 04'28]

Fonction intrinsèques (extraits)

- `enumerate` : parcourt un itérable en numérotant les éléments

```
>>> L = ["y", "yes", "o", "oui"]
>>> for i, rep in  enumerate(L):
      print((i,rep), end=" ; ")
(0, 'y') ; (1, 'yes') ; (2, 'o') ; (3, 'oui') ;
```

- `eval` : renvoie l'évaluation d'une expression contenue dans une chaîne

```
>>> x = 4
>>> rep = input("Expression en x: ")
Expression en x: x**2 - 6
>>> rep
'x**2-6'
>>> eval(rep)
10
```

- Les fonctions `min`, `max` et `sum` acceptent une liste (un itérable) en argument

```
>>> L1 = [10, 11, 12, 13, 14]
>>> sum(L1)
60
>>> min(L1), max(L1)
(10, 14)
```

```
>>> s="abcde"
>>> sum(s)
TypeError: ...
>>> min(s), max(s)
('a', 'e')
```

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 05'41]

Fonctions intrinsèques (extraits)

- `map` : applique une fonction aux éléments d'une liste (renvoie un objet `map` itérable)

```
>>> L = ["1e2", "2.3e-2", "1", "2.3"]
>>> a = map(float, L); a, type(a)
(<map object at 0x7ff5d3dff6a0>, <class 'map'>)
>>> list(a)
[100.0, 0.023, 1.0, 2.3]
```

- `range` (Python 3) : **classe** représentant une séquence d'entiers (prog. arithmétique)

```
>>> a = range(10); a, type(a)
(range(0, 10), <class 'range'>)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, -5, -1))
[0, -1, -2, -3, -4]
```

très utilisée pour construire les boucles classiques :

```
>>> for i in range(10):
        print(i, end=" ")
0 1 2 3 4 5 6 7 8 9
```

- `reversed` : construit un objet `iterator` représentant une liste en sens inverse

```
>>> L = [1, 2, 3, 4, 5]; list(reversed(L))
[5 4 3 2 1]
```

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 07'45]

Fonctions intrinsèques (extraits)

- `repr` : convertit un objet en une expression `str`

  dans la plupart des cas `eval(repr(•)) == •` est vrai !

```
>>> 0.1 + 0.2
0.30000000000000004
>>> repr(0.1 + 0.2)
'0.30000000000000004'
```

- `str` : convertit simplement un objet en chaîne de caractères

```
>>> str(12.4)
'12.4'
>>> str([1,2])
'[1, 2]'
```

utilisé pour les affichages écran simples :

```
>>> a = 12.7
>>> print("valeur de a:",a)
valeur de a: 12.7
>>> print("valeur de a:" + str(a))
valeur de a:12.7
```

[Audio – Python3-2-LesBases-v1.4_FonctionsIntrinsèques.mp3 : 08'40]

Fonctions intrinsèques (extraits)

- `zip` : renvoie un objet `iterator` combinant plusieurs objets itérables

```
>>> Z = zip([1,2,3,4,5], ["Red","Green", "Blue"], [200, 210, 220])
>>> Z
<zip object at 0x7ff5d1a7b108>
>>> list(Z)
[(1, 'Red', 200), (2, 'Green', 210), (3, 'Blue', 220)]
>>> for x, y in zip([1,2,3], [1e2, 1.1e2, 1.3e2]):
      print(x,y)

1 100.0
2 110.0
3 130.0
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 00'00]

Définition et appel des fonctions

- **Définition** : mot clef `def` ... terminé par le caractère ':'
  - le corps de la fonction est **indenté** d'un niveau (bloc indenté) ;
  - les objets définis dans le corps de la fonction sont locaux à la fonction ;
  - une fonction peut renvoyer des objets avec le mot clef `return`.
- **Appel** : nom de la fonction suivi des parenthèses `(...)`

```
>>> q = 0
>>> def divide(a,b):
    q = a // b
    r = a - q*b
    return q,r

>>> x, y = divide(5,2)
>>> q, x, y
(0, 2, 1)
```

- La variable q est définie en dehors de la fonction (valeur = 0)
- L'opérateur // effectue une division entière
- La variable q définie dans la fonction est locale !

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 02'26]

Définition et appel des fonctions

- **Passage des arguments par référence**
  - ▷ à l'appel de la fonction, les arguments (objets) sont transmis par référence ;
  - ▷ un objet mutable transmis en argument peut être modifié par la fonction ;
  - ▷ un objet non mutable transmis en argument ne peut pas être modifié par la fonction.

```
>>> def f0(a, b):
    a = 2
    b = 3
    print("f0-> a: {}, b: {}".format(a,b))
    return
>>> a=0; b=1
>>> a, b
(0, 1)
>>> f0(a, b)
f0-> a: 2, b: 3
>>> a, b
(0, 1)
```

```
>>> def f1(a):
    for i,e in enumerate(a):
        a[i] = 2.*e
    return
>>> a=[1,2,3]
>>> id(a)
140693918338312
>>> f1(a)
>>> id(a)
140693918338312
>>> print(a)
[2.0, 4.0, 6.0]
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 06'09]

Définition et appel des fonctions

- **Arguments positionnels**
  - ▷ à l'appel de la fonction, les arguments passés sont des objets
  - ▷ chaque objet correspond au paramètre de même **position** (même **rang**).

```
>>> def f2(a, b, c):
    print("a: {}, b: {}, c: {}".format(a,b,c))
    return
>>> f2(0)    ...TypeError: f2() missing 2 required positional
                           arguments: 'b' and 'c'
>>> f2(0,1) ...TypeError: f2() missing 1 required positional
                           argument: 'c'
>>> f2(0,1,2)
a: 0, b: 1, c: 2
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 07'37]

Définition et appel des fonctions

- **Arguments nommés**
  - ▷ à l'appel, les arguments passés sont des affectations des noms des arguments ;
  - ▷ l'ordre de passage des arguments nommés est indifférent ;
  - ▷ on peut mixer argument(s) positionnel(s) et argument(s) nommé(s) ;
  - ▷ souvent utilisé avec des paramètres ayant des valeurs par défaut ;
  - ▷ après un argument ayant une valeur par défaut, tous ceux qui suivent doivent aussi avoir une valeur par défaut.

```
>>> f2(c=3, a=1, b=2)
a: 1, b: 2, c: 3
>>> def f3(a, b=2, c=2):
    print("a: {}, b: {}, c: {}".format(a,b,c))
    return
>>> f3(0)
a: 0, b: 2, c: 2
```

```
>>> f3(c=1, a=2, b=3)
a: 2, b: 3, c: 1
>>> f3(0, c=5, b=4)
a: 0, b: 4, c: 5
>>> f3(0, c=5)
a: 0, b: 2, c: 5
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 09'36]

Définition et appel des fonctions

- **Arguments multiples avec le caractère ***

  Utilisé quand on ne connaît pas à l'avance le nombre d'arguments qui seront passés à une fonction.

```
def f4(*x):
    print("argument(s) reçu(s):", x)
    return
```

```
>>> f4(1,2,"a")
argument reçu: (1, 2, 'a')      # 3 arguments reçus
>>> L=[5,"b",6]
>>> f4(L)
argument reçu: ([5, 'b', 6],)   # UN seul argument : la liste L
>>> f4(*L)                      # *L permet de 'défaire' (unpacking) L
argument reçu: (5, 'b', 6)      # 3 arguments reçus
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 10'00]

Définition et appel des fonctions

- **Arguments multiples : caractères ****
  - utilisé pour transmettre un nombre quelconque d'arguments nommés ;
  - la fonction reçoit un objet de type `dict` ;
  - utilise le caractère spécial **.

```
def f5(**x):
    print("argument(s) reçu(s):", x)
    return
```

```
>>> f5(a=2, b=1, z=2)
argument(s) reçu(s): {'a': 2, 'b': 1, 'z': 2}
>>> d={"age":22, "poids":54, "yeux":"bleus"}
>>> f5(**d)
argument(s) reçu(s): {'yeux': 'bleus', 'age': 22, 'poids': 54}
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 10'24]

Les Modules

- Un **module** est un **fichier Python**, contenant constantes, fonctions, classes...
- Un module est importé avec `import nomModule`
  - création d'un **espace de nom** (*namespace*) qui contiendra (préfixe) tous les objets contenus dans le module
  - **toutes les définitions contenues dans le module sont excécutées**
  - un identifiant (`nomModule`) est associé à l'espace de nom du module, dans le programme apellant.

Fichier `test.py` :

```
a = 12
def bonjour():
    print("Hello World!")
    return
```

Import du module `test` :

```
>>> import test
>>> x = test.a; print(x)
12
>>> test.bonjour()
Hello World!
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 13'15]

Les Modules

- Variations possibles pour importer tout ou partie d'un module :
  - Module **entier** importé avec son espace de nom (*alias* possible) :
    ```
    import module [as alias]
    ```
  - Module **entier** importé, dans l'espace de nom courant :
    ```
    from module import *
    ```
  - **Sélection d'items** du module, importés dans l'espace de nom courant :
    ```
    from module import item1 [as alias], item2 [as alias]...
    ```

```
>>> import math
>>> math.sin(math.pi/4)
0.7071067811865475
>>> import math as m
>>> m.sin(m.pi/4)
0.7071067811865475
```

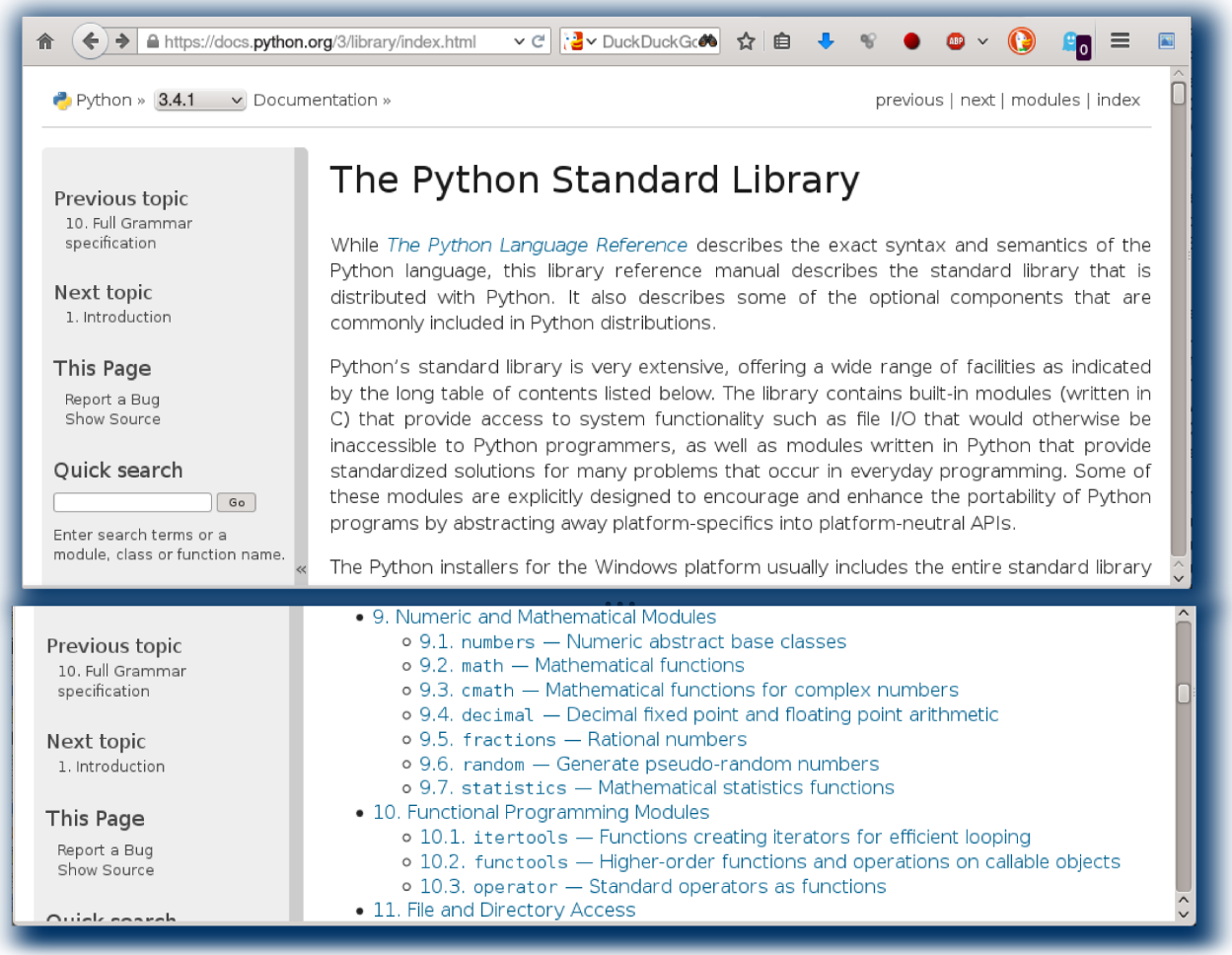
```
>>> from math import *
>>> pi
3.141592653589793
>>> sin(pi/4)
0.7071067811865475
>>> log(1)
0.0
```

```
>>> from math import pi,sin,log
>>> pi
3.141592653589793
>>> sin(pi/4)
0.7071067811865475
>>> log(1)
0.0
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 16'46]

Les Modules de la bibliothèque standard (plus de 200... ☺)

- Doc en ligne Python *Library Reference* docs.python.org/library/index.html



[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 17'15]

Module standard **math**

- Donne accès aux constantes et fonctions mathématiques usuelles (trigonométriques, hyperboliques, log, exp...)

```
>>> import math
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees',
'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 18'08]

Module standard **cmath**

- Donne accès aux fonctions mathématiques usuelles opérant sur des nombres complexes.

```
>>> import math
>>> dir(cmath)
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atanh', 'cos', 'cosh', 'e', 'exp', 'isinf', 'isnan', 'log',
'log10', 'phase', 'pi', 'polar', 'rect', 'sin', 'sinh', 'sqrt', 'tan',
'tanh']
>>> from cmath import exp as cexp
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 18'35]

Module standard os

- Propose des fonctions pour interagir avec le système d'exploitation.

  Exemple : manipulation des répertoires et des fichiers

```
>>> import os
>>> dir(os)
[... 'chdir',... 'getcwd',... 'listdir',... 'mkdir',... 'rmdir']
```

  ▷ `chdir` (*change directory*) : change le répertoire de travail

  ▷ `getcwd` (*get current directory*) : renvoit le répertoire de travail

  ▷ `listdir` (*list directory*) : liste le contenu d'un répertoire (`'.'` : le répertoire courant).

```
>>> os.getcwd()
'/home/jlc'
>>> os.chdir('/tmp/data')
>>> rep = os.getcwd(); rep
'/tmp/data'
>>> os.listdir(rep)
['f1.txt', 'f2.dat', 'f1.dat']
>>> 'f3.dat' in os.listdir(rep)
False
>>> 'f1.dat' in os.listdir(rep)
True
```

```
>>> os.getcwd()
'C:\\Users\\jlc'
>>> os.chdir('C:/tmp/data')
>>> rep = os.getcwd(); rep
'C:\\temp\\data'
>>> os.listdir(rep)
['f1.dat', 'f2.dat', 'f2.txt']
>>> 'f3.dat' in os.listdir(rep)
False
>>> 'f1.dat' in os.listdir(rep)
True
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 19'53]

Les Entrées/Sorties clavier, écran (*Input/Output*)

- Les Entrée/Sortie clavier/écran permettent le dialogue programme/utilisateur.
- La fonction `input('message')` est utilisée :
  - ▷ pour afficher un message à l'écran ;
  - ▷ pour **capturer** la **saisie clavier** et la renvoyer comme un `str`.

```
>>> rep = input("Entrer un nombre: ")
Entrer un nombre : 47
>>> rep
'47'
>>> rep == 47
False
>>> type(rep)
<class 'str'>
>>> x = float(rep); x
47.0
>>> type(x); x == 47
<class 'float'>
True
```

- `input` renvoit un `str`
- `float` permet de convertir un `str` en nombre flottant.

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 21'25]

Les Entrées/Sorties clavier, écran (*Input/Output*)

- Formatage des chaînes de caractères avec la méthode `str.format`

```
"chaîne de formatage contenant des {...}".format(objet1, objet2, ...)
```

```
>>> print("nom: {:s}, age:{:4d}, taille:{:5.2f} m".format("Smith", 42, 1.73)
nom: Smith, age:  42, taille: 1.73 m
>>> print("nom: {1:s}, age:{2:4d}, taille:{0:5.2f} m".format(1.73, "Smith", 42)
nom: Smith, age:  42, taille: 1.73 m
```

- Exemples de spécifications de formatage :

`.nf`	`n` décimales, format `f`lottant
`.ne`	`n` décimales, format scientifique
`n.de`	`n` caractères avec `d` décimales, format scientifique
`s`	chaîne de caractères
`d`	entier en base 10
`g`	choisit le format le plus approprié

- Le formatage "à la `printf` du `C`" peut aussi se faire avec l'opérateur `%`

```
>>> x=1.2e-3 ; print("la variable %s a pour valeur: %8.3E" % ("x", x))
la variable x a pour valeur: 1.200E-03
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 00'00]

Quelques *goodies* Pythoniques...

- **Liste par compréhension**

  Écriture *inline* d'une liste où les éléments viennent d'une expression évaluée en parcourant un objet itérable :

```
[ expr1 for e in objet_iterable ]
```

```
[ expr1 for e in objet_iterable if expr2 ]
```

```
>>> [i*i for i in range(5)]
[0, 1, 4, 9, 16]
>>> [i*i for i in range(10) if i % 2 == 0]
[0, 4, 16, 36, 64]
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
>>> [(x, x**2)  for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
>>> from math import pi
>>> [str(round(pi, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 00'00]

Quelques *goodies* Pythoniques... suite

- **Dictionnaire par compréhension**

  Écriture *inline* d'un dictionnaire où les éléments des paires (`clef` :`valeur`) viennent d'une expression évaluée en parcourant un objet itérable :

```
{ expr1 : expr2 for e in objet_iterable [ if expr3 ] }
```

```
>>> D = { chr(i):i for i in range(65,70)}; D
{'E': 69, 'A': 65, 'D': 68, 'C': 67, 'B': 66}
>>> D = {(u, v):u+v for u in range(1,4) for v in range(1,4) if u != v}; D
{(1, 2): 3, (3, 2): 5, (1, 3): 4, (2, 3): 5, (3, 1): 4, (2, 1): 3}
```

- **Ensemble par compréhension**

  Écriture *inline* d'un ensemble dont les éléments viennent d'une expression évaluée en parcourant un objet itérable :

```
{ expr1 for e in objet_iterable [ if expr2 ] }
```

```
>>> squares = { x*x for x in range(1,5)}; squares
{16, 1, 9, 4}
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 00'00]

Quelques *goodies* Pythoniques... suite

- **Fonction générateur** : fonction (`def`) contenant le mot clé `yield`.
  Elle est itérable (on peut l'utiliser dans une boucle).

```
def fibo(n):
    x, y = 0, 1
    for k in range(n+1):
        yield x
        x, y = y, x + y
```

```
>>> type(f)
<class 'generator'>
>>> f = fibo(4) # initialisation du générateur
>>> next(f) # valeur suivante
0
>>> next(f), next(f), next(f), next(f) # Les 4 valeurs suivantes...
(1, 1, 2, 3)
>>> next(f) # il b'y a que 5 valeurs !
...StopIteration
>>> [u for u in fibo(8)]
[0, 1, 1, 2, 3, 5, 8, 13, 21]
```

[Audio – Python3-2-LesBases-v1.4_FonctionsModulesIOGoodies.mp3 : 00'00]

Différences syntaxiques importantes Python 2.7 / Python 3

- Python3 : l'instruction `print` est remplacée par la fonction intrinsèque `print`

```
>>> print "a =", 45.2
a = 45.2
```

```
>>> print("a =", 45.2)
a = 45.2
```

- Python3 : l'opérateur / effectue toujours la division flottante

```
>>> 2/3
0
>>> 2./3
0.6666666666666666
```

```
>>> 2/3
0.6666666666666666
>>> 2./3
0.6666666666666666
```

- Python3 : un `set` est créé avec les accolades (sauf le `set` vide)

```
>>> s = set()  # set vide
>>> s = set(['a', 'b', 'c'])
>>> type(s)
<class 'set'>
>>> s
set(['a', 'c', 'b'])
```

```
>>> s = set() ; type(s) # set vide
<class 'set'>
>>> s = {'a', 'b', 'c'} ; s
{'a', 'c', 'b'}
>>> type(s)
<class 'set'>
```

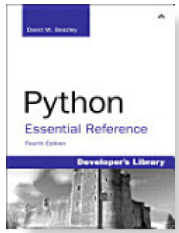
- Ligne mettre en début de fichier Pyhon 2.7 utiliser des améliorations Python 3

```
from __future__ import division, print_function, unicode_literals
```

Références bibliographiques

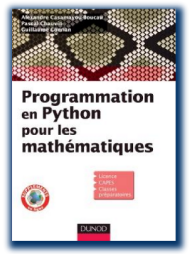
http://docs.python.org/index.html
openclassrooms.com/courses/apprenez-a-programmer-en-python

Apprenez à programmer en Python
Vincent Le Goff
Simple IT éd. (Le livre du zéro)
ISBN 979-10-90085-03-9
≈25 €



Python Essential Reference
David M. Beazley
Addison Wesley
ISBN 0-672-32978-4

Apprendre à programmer avec Python 3
Gérard Swinnen
Télécharger le PDF



Programmation en Python pour les mathématiques
A. Casamayou-Boucau
Dunod
ISBN 978-2-10-057422-3

✉ jean-luc.charles@ensam.eu

✉ eric.ducasse@ensam.eu