Python

Dimitri Dubois

dimitri.dubois@umontpellier.fr

Contenu

- Introduction
- Types intégrés et opérateurs de comparaison
- L'objet liste
- La chaîne de caractères
- L'objet tuple
- L'objet dictionnaire
- La structure conditionnelle et les boucles
- La list-comprehension
- Les fonctions
- La lecture et l'écriture dans un fichier
- Les modules

Introduction

- créé par Guido Van Rossum (Néerlandais)
- 1ère version date de 1991
- géré par la Python Software Fondation depuis 2001
- site web: https://www.python.org
- actuellement branche python 3
- nom vient de la troupe des "Monty Python" (troupe d'humoristes anglais)

- langage de haut niveau
- langage interprété
- dipose d'un ramasse-miettes (gestion automatique de la mémoire)
- multiplateforme
- permet de créer des scripts et des programmes complets
- de + en + utilisé pour l'analyse de données (data science, big data, machine learning ...)
- syntaxe simple, peu verbeuse
- fonctionnalités extensibles avec des packages

Environnement Python

- la console : interpréteur de code Python
- permet d'écrire et d'exécuter du code mais ne le conserve pas
- consoles évoluées: ipython, jupyter-notebook
- pour développement d'applications, utilisation d'un IDE (PyCharm par exemple)

Installation de Python

- anaconda : distribution Python
- permet d'utiliser Python sans l'installer sur le système d'exploitation
- propose de nombreuses fonctionnalités comme les environnements virtuels
- website : https://www.anaconda.com/

Règles d'écriture du code (PEP 8)

- pas d'accolades ni de point-virgule
- : et indentation pour définir les blocs d'instruction
- des underscores dans les noms de variables et fonctions
- # pour les commentaires sur une ligne
- """ pour les commentaires multilignes

The "Zen of Python", par Tim Peters

1. Préférer le beau au laid
2. l'explicite à l'implicite
3. le simple au complexe
4. le complexe au compliqué
5. le déroulé à l'imbriqué
6. l'aéré au compact
7. la lisibilité compte
8. les cas particuliers ne le sont jamais assez pour violer les règles, même s'il faut privilégier l'aspect pratique à la pureté
9. ne jamais passer les erreurs sous silence, ou les faire taire explicitement
10. face à l'ambiguité, ne pas se laisser tenter à deviner
11. il doit y avoir une - et si possible une seule - façon évidente de procéder, même si cette façon n'est pas évidente à première vue, à moins d'être Néerlandais
12. mieux vaut maintenant que jamais, même si jamais est souvent mieux qu'immédiatement
13. si l'implémentation s'explique difficilement, c'est une mauvaise idée
14. si l'implémentation s'explique facilement, c'est peut-être une bonne idée
15. les espaces de nommage sont une sacrée bonne idée, utilisons les plus souvent!

Types intégrés (built-in types) et opérateurs de comparaison

Nombres, chaînes de caractères et booléens

- int: nombre entier
- float: nombre décimal
- str: chaîne de caractères
- bool: booléen
- None: rien

- type(variable) renvoie le type
- int(), float(), str(), bool() pour changer le type (si compatible)

Opérateurs de comparaison

- + - * / % (modulo) // (division entière) et ** (puissance)
- Comparaisons (valeur): <> <=>= == et !=
- Comparaison (référence): is et is not
- and et or pour comparaisons multiples

Création d'une variable

- typage dynamique
- affectation avec le signe = (`ma_variable = 10`)
- sensible à la casse (`ma_variable != ma_Variable`)
- ne pas utiliser les mots clés du langage comme nom de variable ou de fonction

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

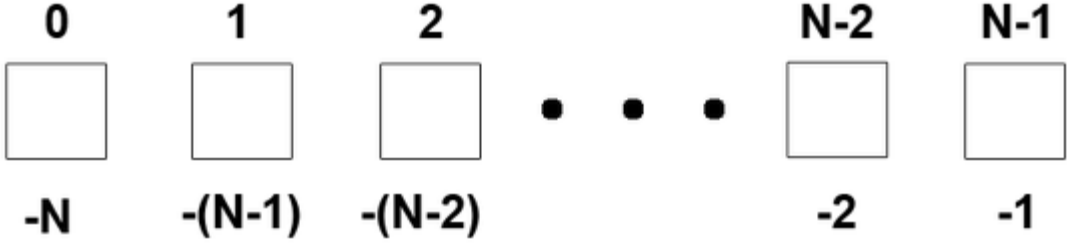
L'objet Liste

- peut stocker des éléments de type différent (int, str, bool, float, etc.)
- éléments ordonnées
- premier index à 0
- la liste et les éléments de la liste peuvent être modifiés dynamiquement
- instanciation: list() ou [] pour une liste vide
  `ma_liste = [1, "coucou", True]` pour une liste contenant déjà des éléments
- `ma_liste[0]` affiche le premier élément de la liste (ici 1)
  `ma_liste[1]` affiche "coucou"

Tranches / slices

- possible de travailler sur des tranches de liste (slice)
- permet d'afficher tout ou partie de la liste
- renvoie une nouvelle liste
- syntaxe: `ma_liste[start: stop: step]`
- `ma_liste[start: stop]` contient les éléments i tels que start <= i < stop
  donc i va de start à stop-1
- par défaut l'intervalle est de 1
- chacun des éléments du slicing est optionnel
  `ma_liste[:]` renvoie une copie de la liste
  `ma_liste[:2]` renvoie les 2 premiers éléments (index 0 et 1)
  `ma_liste[1:]` renvoie de l'index 1 à la fin
  `ma_liste[::2]` renvoie toute la liste mais par intervalles de 2

Index négatifs



- index négatifs pour partir de la fin
  `ma_liste[-1]` renvoie le dernier élément
- un intervalle négatif fait afficher dans l'ordre inverse
  `ma_liste[::-1]` renvoie [True, "coucou", 1]

Création d'une liste

```
ma_liste = ["pomme", "poire", "fraise", "frambroise"]
```

Affichage de la liste

```
ma_liste
```

```
['pomme', 'poire', 'fraise', 'frambroise']
```

Affichage de l'élément à l'index 1

```
ma_liste[1]
```

```
'poire'
```

Affichage des éléments à partir de l'index 1

```
ma_liste[1:]
```

```
['poire', 'fraise', 'frambroise']
```

Affichage des index 1 et 2

```
ma_liste[1:3]
```

```
['poire', 'fraise']
```

Affichage des deux premiers éléments

```
ma_liste[:2]
```

```
['pomme', 'poire']
```

Affichage par intervalle de 2

```
ma_liste[::2]
```

```
['pomme', 'fraise']
```

Affichage du dernier élément

```
ma_liste[-1]
```

```
'frambroise'
```

Affichage de liste à l'envers

```
ma_liste[::-1]
```

```
['frambroise', 'fraise', 'poire', 'pomme']
```

Méthodes génériques applicables l'objet liste

- `len(ma_liste)`: affiche la longueur de la liste (nombre d'éléments)
- `del ma_liste[1]`: supprime l'élément à l'index 1
- `del ma_liste[:]` vide la liste

Méthodes de l'objet liste

- `ma_liste.append(objet)`: ajoute l'objet à la fin de la liste
- `ma_liste.insert(index, objet)`: insère l'objet à l'index passé en argument
- `ma_liste.remove(objet)`: supprime la première occurence de l'objet
- `ma_liste.extend(autre_liste)`: ajoute les éléments de autre_liste à la liste
- `ma_liste.index(objet)`: renvoie le numéro d'index de la première occurence de l'objet
- `ma_liste.pop()`: renvoie le dernier élément de la liste et le supprime
- `element in ma_liste`: renvoie True si element est dans la liste et False sinon

Ajout d'un élément à la fin de la liste

```
ma_liste.append("banane")
ma_liste
```

```
['pomme', 'poire', 'fraise', 'frambroise', 'banane']
```

Suppression d'un élément

```
ma_liste.remove("banane")
ma_liste
```

```
['pomme', 'poire', 'fraise', 'frambroise']
```

Extension de la liste avec une autre liste

```
ma_liste.extend(["kiwi", "orange"])
ma_liste
```

```
['pomme', 'poire', 'fraise', 'frambroise', 'kiwi', 'orange']
```

Parcourir l'objet liste

Première méthode, par les valeurs

```
for e in ma_liste:
  print(e)
```

e va successivement prendre chaque valeur de la liste

Parcours de la liste par valeur

```
for e in ma_liste:
    print(e)
```

```
pomme
poire
fraise
frambroise
kiwi
orange
```

Seconde méthode : index et valeur avec enumerate

```
for i, e in enumerate(ma_liste):
    print(i, e)
```

i prend successivement le numéro de l'index et e l'élément correspondant

Parcours de la liste avec enumerate

```
for i, e in enumerate(ma_liste):
    print(i, e)
```

```
0 pomme
1 poire
2 fraise
3 frambroise
4 kiwi
5 orange
```

Pratique

- créer un notebook
- pour chaque consigne ci-dessous mettre le code dans une nouvelle cellule

- créer une liste (pays, joueurs de foot, etc.)
- faire afficher la longueur de la liste
- faire afficher les deux derniers éléments de la liste
- ajouter un élément à la liste
- enlever le deuxième élément de la liste
- le remettre à sa place
- faire afficher chaque élément de la liste avec une boucle for
- faire afficher chaque élément de la liste et son index avec enumerate

La chaîne de caractères

- création: `ma_chaine = "le contenu"`
- est considérée comme une liste de caractères
- slices fonctionnent comme pour les listes: `ma_chaine[:2]` donne "le"

Création

```
ma_chaine = "bonjour tout le monde"
ma_chaine
```

```
'bonjour tout le monde'
```

Longueur de la chaîne

```
len(ma_chaine)
```

```
21
```

Affichage d'une partie de la chaîne

```
ma_chaine[2:10]
```

```
'njour to'
```

Méthodes de la chaîne de caractères

- capitalize(), upper(), lower(), split(separateur), count(caractere) etc.
- ne modifient pas la chaîne initiale, renvoie une nouvelle chaîne

```
ma_chaine.capitalize()
```

```
'Bonjour tout le monde'
```

```
ma_chaine.upper()
```

```
'BONJOUR TOUT LE MONDE'
```

```
ma_chaine.split("o")
```

```
['b', 'nj', 'ur t', 'ut le m', 'nde']
```

```
ma_chaine.count("o")
```

```
4
```

Formatage

- insérer des variables dans une chaîne de caractères
- faire précéder la chaîne de *f* et mettre les variables entre accolades

Formatage

```
nombre_1 = 10
nombre_2 = 20
ma_chaine_1 = f"L'addition de {nombre_1} et {nombre_2} fait {nombre_1 + nombre_2}"
print(ma_chaine_1)
```

```
L'addition de 10 et 20 fait 30
```

pour les float possibilité de spécifier le nombre de décimales

Formatage des valeurs décimales (float)

```
ma_valeur = 125.12597862
print(f"La valeur est {ma_valeur:.2f}")
```

```
La valeur est 125.13
```

Pratique

- créer une chaîne de caractères et faire afficher les 2 premières et les 2 dernières lettres de la chaîne
- `prenom, nom, age = "jean", "dupont", 35.57895`
  faire afficher (avec print et en utilisant le formatage): `"Je m'appelle Jean DUPONT, j'ai 35.58 ans."`

L'objet tuple

- comme une liste sauf que non mutable
- création avec parenthèses: `mon_tuple = (1, "A", True)`
- création d'un tuple avec un seul élément: `mon_tuple = ("A",)`
- peut-être transformé en liste: `ma_liste = list(mon_tuple)`
- peut transformer une liste: `tuple(mon_liste)`
- plus sûr qu'une liste pour stocker des éléments qui ne doivent pas être modifiés

```
ma_liste_tuple = tuple(ma_liste)
ma_liste_tuple
```

```
('pomme', 'poire', 'fraise', 'frambroise', 'kiwi', 'orange')
```

```
ma_liste_tuple[2] = "carotte"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [27], in <cell line: 1>()
----> 1 ma_liste_tuple[2] = "carotte"

TypeError: 'tuple' object does not support item assignment
```

Création d'une liste de tuples à partir de 2 listes de même taille avec la fonction zip

```
list(zip(["A", "B", "C"], [1, 2, 3]))
```

```
[('A', 1), ('B', 2), ('C', 3)]
```

L'objet dictionnaire

objet qui stocke sous forme clé: valeur

Instanciation

```
mon_dict_1 = dict(prenom="Jean", nom="Dupont", age=35)
mon_dict_2 = {"prenom": "Jean", "nom": "Dupont", "age": 35}
```

La première méthode n'est possible que si toutes les clés sont des mots (sans espaces)

Création

```
mon_dict = dict(python="langage de programmation", html="langage de balises")
mon_dict
```

```
{'python': 'langage de programmation', 'html': 'langage de balises'}
```

Création à partir d'une liste d'itérables de taille 2

```
dict([("A", 1), ("B", 2)])
```

```
{'A': 1, 'B': 2}
```

Ajout et suppression de paires

- pour ajouter un élément, mettre la clé entre crochets
  `mon_dict["ma_cle"] = valeur`
- si la clé existe déjà, la valeur est remplacée

- pour accéder à une valeur appellée par la clé
  `mon_dict["ma_cle"]` renvoie la valeur associée à "ma_cle"
- KeyError si on essaie d'accéder à une clé qui n'existe pas

- `del mon_dict["ma_cle"]` pour supprimer la paire "ma_cle"

Ajout d'une paire clé / valeur

```
mon_dict["javascript"] = "langage de programmation du web"
```

```
mon_dict
```

```
{'python': 'langage de programmation',
 'html': 'langage de balises',
 'javascript': 'langage de programmation du web'}
```

Accès à une clé n'existant pas

```
mon_dict["css"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [37], in <cell line: 1>()
----> 1 mon_dict["css"]

KeyError: 'css'
```

Les méthodes de l'objet dictionnaire

- `mon_dict.clear()` vide le dictionnaire
- `mon_dict.get(ma_cle, "valeur à renvoyer sinon")` renvoie "valeur à renvoyer sinon" si la clé n'existe pas (évite KeyError)

- `mon_dict.keys()` renvoie la liste des clés
- `mon_dict.values()` renvoie la liste des valeurs
- `mon_dict.items()` renvoie la liste des tuples (clé, valeur)

- pour tester l'existence d'une clé dans le dictionnaire
  `ma_cle in mon_dict.keys()` renvoie True or False

Méthode get

```
mon_dict.get("css", "langage de style pour le html")
```

```
'langage de style pour le html'
```

Méthode keys()

```
mon_dict.keys()
```

```
dict_keys(['python', 'html', 'javascript'])
```

Méthode items - Renvoie une liste de tuples (clé, valeur)

```
mon_dict.items()
```

```
dict_items([('python', 'langage de programmation'), ('html', 'langage de balises'), ('javascript', 'langage de programmation du web')])
```

Test de présence d'une clé

```
"css" in mon_dict
```

```
False
```

```
"python" in mon_dict
```

```
True
```

Parcours du dictionnaire

- `for k in mon_dict.keys():`
  `instructions avec k`

- `for v in mon_dict.values():`
  `instructions avec v`

- `for k, v in mon_dict.items():`
  `instructions avec k et/ou v`

Parcours du dictionnaire

```
for k, v in mon_dict.items():
    print(f"{k} : {v}")
```

```
python : langage de programmation
html : langage de balises
javascript : langage de programmation du web
```

Pratique

- créer un dictionnaire avec 4 paires clés / valeurs
- faire afficher une valeur en appelant une clé
- avec print, faire afficher toutes les paires clés / valeurs
- ajouter une paire clé / valeur au dictionnaire existant
- tester l'existance d'une clé dans le dictionnaire

Structure conditionnelle et boucles

Structure conditionnelle : if ... elif ... else

```
mon_nombre = 50
if nom_nombre < 100:
  print("Le nombre est inférieur à 100")
elif mon_nombre > 100:
  print("Le nombre est supérieur à 100")
else:
  print("Le nombre est égal à 100")
```

Boucle for : répéter un ensemble d'instructions un nombre fini de fois

```
for i in iterable:
    print(i)
```

un itérable : une liste, un tuple ou un dictionnaire sont des itérables

Fonction *range*

fonction qui génère une séquence

`range(start, stop, step)`

start (inclusif): le nombre de départ (optionnel, par défaut 0)

stop (exclusif): le nombre d'arrivée sera stop - 1

step : l'intervalle entre chaque nombre (optionnel, par défaut 1)

`range(10)` génère les nombres 0 à 9

`range(0, 10, 2)` génère les nombres paires entre 0 et 9

souvent utilisée avec for :

```
for i in range(10) :
  instructions
```

répète les instructions 10 fois.

Boucle while

- tant que
- utilisée lorsque l'on ne connaît pas à l'avance le nombre de répétitions à faire
- attention à la boucle infinie

```
continuer = True
while continuer:
   nombre_aleatoire = random.random()
   print(nombre_aleatoire)
   if nombre_aleatoire < 0.25:
     continuer = False
```

L'instruction break

permet de stopper une boucle while

```
while True:
    nombre_aleatoire = random.random()
    print(nombre_aleatoire)
    if nombre_aleatoire < 0.25:
      break
```

L'instruction continue

permet de sauter un ou plusieurs éléments dans une boucle

```
for i in range(51):
   if i % 5 == 0:
     continue
   print(i)
```

Pratique

- faire afficher les nombres multiples de 10 entre 0 et 100 (compris)
- utiliser une boucle while pour afficher un nombre aléatoire. Arrêter la boucle si le nombre est supérieur à 0.75

La list-comprehension

- création d'une liste à partir d'une autre liste
  forme générale: `[expression for value in iterable]`
  `[i**2 for i in range(10)]` donne `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

- possible d'ajouter une condition
  `[i**2 for i in range(10) if i%2 == 0]` donne `[0, 4, 16, 36, 64]`

- et même plusieurs
  `[i**2 for i in range(10) if i > 2 and i%2 == 0]` donne `[16, 36, 64]`

- possible de créer un dictionnaire

  `{i: i**2 for i in range(10)}` donne

  `{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}`

- aussi avec conditions

  `{i: i**2 for i in range(10) if i > 2 and i%2 == 0}` donne

  `{4: 16, 6: 36, 8: 64}`

Pratique

créer une liste de tuples: le nombre et son carré, pour tous les nombres multiples de 4 compris entre 0 et 100

```
[(0, 0), (4, 16), (8, 64), ..., (96, 9216), (100, 10000)]
```

Les fonctions

Utilité et syntaxe

- permet de regrouper plusieurs instructions dans un bloc nommé qui peut être appelé par la suite (et à plusieurs reprises)
- déclaration avec le mot clé `def`
- l'indentation qui suit la déclaration définit le bloc d'instructions
- peut nécessiter un ou plusieurs paramètres pour fonctionner
- peut renvoyer une valeur ou un objet avec l'instruction return
  renvoie de None par défaut si pas de return
- se documente avec des docstrings (triples guillemets)
  help(fonction) affiche cette documentation

```
def agir(arguments):
    """La documentation de la fonction"""
    instructions avec les arguments
    return resultat
```

Définition d'une fonction

```
def mettre_au_carre(nombre):
    """ Renvoie nombre au carré """
    return nombre ** 2
```

Aide de la fonction

```
help(mettre_au_carre)
```

```
Help on function mettre_au_carre in module __main__:

mettre_au_carre(nombre)
    Renvoie nombre au carré
```

Appel de la fonction

```
mettre_au_carre(5)
```

```
25
```

built-in functions (fonctions intégrées)

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	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()	

documentation de ces fonctions: https://docs.python.org/3/library/functions.html

Paramètres et arguments

- paramètres: noms spécifiés dans la signature de la fonction
- arguments: objets/valeurs passés lors de l'appel de la fonction

```
def power(le_nombre, quelle_puissance):
  return le_nombre ** quelle_puissance
```

le_nombre et quelle_puissance sont les paramètres de la fonction

```
power(10, 3)
```

10 et 3 sont les arguments passés lors de l'appel de la fonction

- paramètres facultatifs ont une valeur par défaut et sont placés après les paramètres obligatoires dans la signature de la fonction

```
def power(quel_nombre, quelle_puissance=2):
    return le_nombre ** quelle_puissance
```

- la fonction peut être appelée avec un seul argument (celui obligatoire)
  `power(10)` renvoie 100
- ou avec deux arguments
  `power(10, 3)` renvoie 1000

Pratique

- écrire une fonction qui prend une liste de nombres en paramètre et qui renvoie la somme des éléments de la liste
- écrire une fonction qui prend une liste en paramètre et qui renvoie une nouvelle liste avec les éléments unique de la liste initiale
- écrire une fonction qui prend 2 listes de même taille en paramètre et qui renvoie un dictionnaire dans lequel la clé est l'élément de la première liste et la valeur celui de la deuxième liste
  par exemple l'appel de la fonction avec `["A", "B", "C"], [1, 2, 3]` renvoie `{"A": 1, "B": 2, "C": 3}`

Lecture / écriture dans un fichier

La fonction open

- permet d'ouvrir un fichier s'il existe
- ou de le créer s'il n'existe pas
- plusieurs mode d'ouverture: lecture "r" (defaut), écriture "w" ou ajout "a"

```
with open("chemin", "w") as fichier:
  ecriture dans le fichier
```

Ecriture dans un fichier

- `write(contenu)` - contenu est une chaîne de caractères
  il faut ajouter des "\n" à la fin des lignes pour ajouter un retour à la ligne

```
with open("/tmp/test.txt", "w") as f:
    f.write("Bonjour tout le monde\n")
    f.write("Comment allez-vous?")
```

- `writelines(contenu)` - contenu est une liste de chaînes de caractères

```
contenu = ["Bonjour tout le monde\n", "Comment allez-vous ?"]
with open("/tmp/test_lines.txt", "w") as f:
    f.writelines(contenu)
```

Lecture d'un fichier

- `read()` lit tout le contenu et le place dans une chaîne de caractères

```
with open(chemin, "r") as f:
  contenu = f.read()
```

- `readline()` lit la ligne courante et la place dans une chaîne de caractères
  il faut donc faire une boucle pour lire tout le contenu

```
contenu = ""
with open("chemin", "r") as f:
   while True:
     line = fichier.readline()
     if line == "":
       break
     contenu += line
```

- `readlines()` lit tout le contenu et le place dans une liste de chaînes de caractères - une ligne = un élément de la liste

  ```
  contenu = f.readlines()
  ```

  contenu est une liste
- pour un fichier lourd il faut préférer readline qui est moins consommateur de mémoire (traitement ligne par ligne)

Pratique

- écrire du texte dans un fichier
- ouvrir le fichier, le lire et afficher le contenu

Les modules

- fichiers contenant des objets et fonctions que l'on peut importer
- par exemple dans le built-in il y a les fonctions sum(), max() et min()
- mais il existe aussi le module math qui contient beaucoup d'autres fonctions, objets et valeurs

  `import math` permet d'importer le module math

  `from math import pow` permet d'importer la fonction pow du module math

  `from math import pow, sqrt` permet d'importer les fonctions pow et sqrt

```
from math import pow
print(pow(10, 3))
```

- par convention les `import` se placent en haut du fichier de code
- `dir(math)` affiche la liste des objets et fonctions proposés par le module
- `help(math)` affiche la documentation du module
- `help(math.pow)` affiche la documentation de la fonction pow

```
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'fl
oor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'l
gamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tan
h', 'tau', 'trunc']
```

```
help(math.sqrt)
```

```
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.
```

Principaux modules du built-in

Nom	Utilité
os	interactions avec l'OS
sys	interactions avec l'interpréteur
datetime et time	objets et fonctions liés au temps
math	constantes et fonctions de base en maths
random	génération d'aléas (nombre, sélection etc.)

collections, copy, re ...

Pour aller plus loin ...

- *args et **kwargs dans les paramètres d'une fonction
- création d'une fonction avec lambda
- les générateurs
- les décorateurs
- sérialisation avec pickle
- ...