

Mémento Python

python3 *program.py* exécuter le programme *program.py*
ipython3 console Python: utile pour tester des bouts de code, faire des calculs, ...
jupyter notebook création de documents mélangeant code Python, images, graphiques et texte (markdown, html/css, latex)

Types usuels

Exemples: (Le type des variables est détecté automatiquement).

bool	booléen	True False	objet modifiable
int	entier (sans limite de taille)	k = 1075	
float	réel en virgule flottante	12.4 x = 3.2e18	
complex	nombre complexe	2 + 3j	
str	chaîne de caractères	s = 'bon' "C'est " + s texte = ""	
list	liste d'objets	[1, 3, 3, 'plus', 5.1] + [8]	✓
tuple	liste non modifiable d'objets	(1, 3, 3, 'plus', 5.1) (8,)	✓
set	ensemble	{2, 4, 'ABC', 5.1}	✓
dict	dictionnaire (ensemble de clé → valeur)	{'A': 1, 'B': 2}	✓

objets **itérables** (objet conteneur dont on peut parcourir les éléments)

Identifiants (noms de variables, fonctions, ...), affectations (=)

Affectation: **x = 3e8/2e-5** **x, y, z = 4.5, 2.2, 'yes'** **Exemple:**
Identifiant: • sensible min/Maj **i = j = 0** affectation multiple (via tuple)
• accents acceptés **i += 1** ⇔ **i = i + 1**
• _ est un identifiant possible **j *= 2** ⇔ **j = j * 2**
L = [1, 2, 3]
M = L
L.append(4)
M et L pointent vers le même objet, la liste [1,2,3], qui peut être modifiée.

Boucles

for variable in sequence: tout objet **itérable** (list, str, tuple, set, dict, ...)
Bloc d'instructions bloc défini par l'indentation
► **Boucle sur des entiers**
range(debut, fin, pas) crée un objet itérable pour parcourir une **séquence d'entiers**. fin n'est pas inclus.
for i in range(1, 3): # pour i = 1, 2
for i in range(4): # pour i = 0, 1, 2, 3
for i in range(4, 0, -1): # pour i = 4, 3, 2, 1
Pour convertir range en liste: **list(range(debut, fin))**
► **Boucle sur des objets itérables**
for lettre in "Mamma mia"
for element in ['k', 7]
for x in reversed(sequence)
for index, element in enumerate(ma_liste)
print("L'élément à l'index", index, "est", element)
for v1, v2 in zip(sequence1, sequence2)
Parcourir plusieurs séquences en parallèle

while expression_logique: Exemple: **x = 500**
Bloc d'instructions **while x > 1.0:**
x = x/2
print(x)

continue passage immédiat au tour suivant
break sortie immédiate de la boucle

Maths

Opérateurs
+ - * / ← La division / donne toujours un float
// Division entière
** puissance **a^b signifie a XOR b**
% modulo (reste de la division entière)
@ (entre tableaux numpy) multiplication matricielle ou produit scalaire
abs(x) valeur absolue
round(x, n) arrondir à n décimales

from math import *
Définit **pi** et **e** et les fonctions:
sin(pi/2) # vaut 1.0
sqrt(2) # √2
log(e3)** # ln(e³)
log(100, 10) # log₁₀(100)
radians(90), degrees(pi)
etc...

Appliquer une fonction à une liste / tableau:
① utiliser le module **numpy**
Exemple: **2*np.sin(T)** **T**: tableau ndarray
② **générateur de séquences**
Ex: **[2*sin(x) for x in L]**
③ fonction **map()**:
Ex: **List(map(lambda x: 2*sin(x), L))**
map() crée un itérateur qui applique la fonction aux éléments de la liste.
définit une fonction "en ligne" (fct anonyme, i.e. sans identificateur)

Entrée/Sortie: écran, fichiers

Formatage de chaînes de caractères: **'%format' % (variable)** ou **f'{variable:format}'**
Exemples: **'Variable %s = %8.3f' % ('x', x)** (codes de format du langage C)
'Variable {} = {} et y = {:.4f}'.format('x', x, y) alignement (optionnel): < à gauche, > à droite, ^ centré
f'Variable x = {x} et y = {y:~10.4f}' Python ≥ 3.6 (opt.) largeur = nb caractères, (opt.) nb décimales, type: d entier (b bin, x hex.), f flottant, e format scientifique, g général (sélection auto.), s chaîne de caractères
Chaine de caractères Convertir avec **int(s)** ou **float(s)**
print('texte', variable, ...) Options suppl.: **sep=' '**, **end='\n'**, **file=f**
Fichiers: lecture
f = open('fichier.txt')
f.readline() #retourne la ligne suivante
f.readlines() #retourne une liste des lignes
f.read() #chaîne avec contenu du fichier
Fichiers: écriture
f = open('fichier.txt', 'w')
f.write('Ligne à écrire\n') w write, a append, r read/write, + read
f.close() À la fin: **with open('fichier.txt') as f:** Avec un **bloc with**, le fichier est fermé automatiquement à la fin du bloc.
for ligne in f: ← ligne contient le caractère '\n' à la fin. Remarque: Avec **readline()**, ligne = '' si fin du fichier.
print(ligne.rstrip()) ← **rstrip()** supprime les caractères blancs (\n, \t et les espaces) à la fin.

Fonctions

def ma_fonction(arg1, arg2, arg_opt=9):
Documentation opt. multi-ligne
Bloc d'instructions
return valeur Valeur par défaut de l'argument optionnel
Appel: **v = ma_fonction(arg1=3, arg2='texte')**
Exemple: **def norme2(x, y):**
return x2 + y**2**
n = norme2(1.5, 3.0)
Pas de restriction sur le type d'objet retourné (→ tuple pour retourner plusieurs valeurs). Pas de 'return' ⇔ return None.
Accès aux variables du prog. principal possible en **lecture**.

Séquences (seq = list, tuple, str, ...)

len(seq) nombre d'éléments
seq[i] élément d'index i
del seq[i] supprime l'élément d'index i
min(seq), max(seq), sum(seq)
seq + seq2
seq * 3
L i s t e **seq[0]** 1^{er} élément
seq[-1] dernier él.
Sous-liste: **seq[debut:fin:pas]** fin n'est pas inclus.
Exemples:
L = [1,2,3,4]
L[3] = -4 # [1,2,3,-4]
L[:3] # [1,2,3] (début → élément 3 non inclus)
L[-2:] # [3,-4] (avant-dernier → fin)
L[:] = 1 # [1,1,1,1]
M = L[:] crée une copie (superficielle) de la liste L
M = sorted(seq) crée une liste triée
filter(x<0, seq) itérateur retenant les éléments selon la condition
Convertir, au besoin, avec **list()**.

Générateur de séquence

expression for élément in itérateur if condition

Exemple: **[2*i for i in range(3)]** # liste [0, 2, 4]

Méthodes spécifiques aux listes

L.append(élément)
L.insert(index, valeur)
L.remove(valeur)
L.index(valeur)
L.pop(index) ⇔ **del L[index]** mais retourne la valeur de l'élément
L.sort() trie la liste L en place (et retourne **None**)

Conversions de type

type(variable) #retourne le type de variable
float → int **int(2.5), round(2.5), floor(-1.6), ceil(-1.6)**
str → int, float **int("5"), float("2.5")**
int, float → str **str(2.5)**
list → str **"-".join(['A', 'B', 'C']) → 'A-B-C'**
list(map(str, [1.1, 5.1])) → ['1.1', '.5']
str → list **list("Ça va") → ['Ç', 'a', ' ', 'v', 'a']**
list("Ça va".split(" ")) → ['Ça', 'va']

Dictionnaire

D = {'A': 1, 'B': 2} **D.keys()**
D[new_key] = new_value **D.values()**
for key, value in D:
print('Clé:', key, 'Valeur:', value)

Bibliothèque standard

► Arguments du prog. **import sys; liste_args = sys.argv**
► Opérations sur fichiers/dossiers **chemin/accès/fichier.txt**
import os
import shutil
import glob
liste_fichiers = glob.glob("*.jpg")
os.path.basename(path)
os.path.dirname(path)
Fichiers / dossiers **shutil.copy(src, dest)** **Dossiers**
os.rename(src, dest) **os.getcwd()**
os.remove(file_path) **os.listdir(path)**
os.mkdir(path)

► Exécuter une commande shell **import subprocess as sp**
**sp.run('ls -l'.split(' '), **
stdout=subprocess.PIPE).stdout.decode()
► Autres modules standard **random, time, datetime, re** etc...

Débogueur

import pdb; pdb.set_trace() Entrer dans le débogueur
pdb>>>
next (n) **print (p)** Affiche une expression
step in (s) **list (L)** Affiche le code source
return (r) **!commande_python_à_exécuter**
continue (c) ↩ répéter la dernière commande
quit (q)

Aide

? module (ou **type** ou autre) raccourci **ipython** pour **help(...)**
dir(module) liste le contenu d'un module ou d'une classe d'objets.
Cet aide-mémoire est un aperçu non exhaustif de commandes Python.
Documentation complète: <http://docs.python.org>
Auteur: Vincent Ballenegger (décembre 2017)

```
ax = plt.subplot(111, projection='3d')
ax.plot_surface(XX, YY, Z, cmap='jet')
```