Aide-mémoire Python

1 — Importer une librairie

La méthode recommandée est la suivante :

```
import numpy as np
```

Ensuite toutes les commandes provenant du package **numpy** doivent être préfixées par **np** (**np.sum**...).

2 — Instructions de base

2.1 — Opérations arithmétiques

Opération	Symbole	Remarque
addition	+	
soustraction	-	
multiplication	*	
puissance	**	
Division sur les flottants	/	10/3 = 3.3333333
Quotient de la division euclidienne	//	10/3 = 3
Reste de la division euclidienne	%	10%3 = 1

Les fonctions usuelles mathématiques sont dans le package **math**, par exemple la racine carrée (**sqrt**), l'exponentielle (**exp**), etc.

Astuce Quelques opérations ont un raccourci commode

```
i += 1      # ajoute 1 à la variable i
p *= 5      # multiplie p par 5
a, b = b, a # échange les valeurs des variables de a et b
```

2.2 — Opérations conditionnelles

Outre **and**, **or** et **not**, on peut aussi utiliser

Opération	Symbole	Commentaire
strictement inférieur à	<	3 < 4 (**True**)
inférieur ou égal à	<=	13.254 <= 13.254 (**True**)
strictement supérieur à	>	
strictement ou égal à	>=	
égal à	==	Attention **a = 1** est une affectation !
n'est pas égal à	!=	1 != "1" (**True**)

2.3 — Tirer au hasard

Nécessitent d'importer le package **random** : **import random as rd**.

Opération	Fonction
tirer un flottant entre 0 et 1	**rd.random()**
tirer un entier dans $[\![i ; j]\!]$	**rd.randint(i, j)**
tirer un élément dans la liste **L**	**rd.choice(L)**
tirer n éléments dans la liste **L**, sans répétitions	**rd.sample(L, n)**

3 — Éléments de base pour la programmation

Les indentations jouent le rôle de marqueur de blocs d'instructions.

3.1 — Structure de test – `if`

```
if condition1:
    INSTRUCTION1     # si la condition1 est vérifiée
elif condition2:
    INSTRUCTION2    # si condition1 est fausse et condition2 vraie
else:
    INSTRUCTION3    # si aucune condition n'est vérifiée
POURSUITE DU PROGRAMME
```

Notez que les `elif` et `else` ne sont pas obligatoires.

3.2 — Structure de boucle – `for`

```
for i in list:
    INSTRUCTIONS
POURSUITE DU PROGRAMME
```

On peut utiliser toute sorte de listes, comme `range(n)` (liste des entiers entre 0 et $n-1$).

3.3 — Structure de boucle – `while`

```
while CONDITION:
    INSTRUCTIONS
POURSUITE DU PROGRAMME
```

Il est recommandé de coder la condition sous la forme `not(condition d'arrêt)`.
La valeur de test doit évoluer au cours de l'exécution de la boucle. Le test doit donc porter sur des variables qui seront modifiées lors de l'exécution. Ces variables doivent être initialisées avant le début de la boucle.

3.4 — Définition d'une fonction

On veut définir une fonction dont le nom est **`mafonction`** ayant **`arg1`**, **`arg2`** et **`arg3`** en arguments d'entrée et **`sortie1`** et **`sortie2`** en arguments de sortie.

```
def mafonction(arg1, arg2, arg3):
    """ Commentaire explicatif
    éventuellement sur plusieurs lignes.
    """

    INSTRUCTIONS

    return sortie1, sortie2
```

La fin de la fonction est matérialisée par le retour de l'indentation.
Pour appeler la fonction avec **`a`**, **`b`**, et **`c`** en entrée et récupérer les deux sorties dans des variables **`sort1`** et **`sort2`** :

```
sort1, sort2 = mafonction (a, b, c)
```

4 — Structure générale d'un programme

Les commentaires sont indiqués par des #. Tout ce qui suit ce caractère est ignoré jusqu'à la fin de la ligne.
Les commentaires sont essentiels pour la communication avec le lecteur.
La structure suivante est vivement recommandée :

```
import ...      # Import de toutes les librairies utiles

EPS = ...       # Définition des éventuelles constantes
                # La tradition veut que leur nom soit en majuscule

def f(...):     # Définition de toutes les fonctions
    """ Description """
    ...
    return ...

# Enfin le corps principal du programme
```

Juste après la définition d'une fonction, on décrit le fonctionnement de la fonction, en utilisant trois guillemets ouvrants et trois fermants. La description peut prendre plusieurs lignes.

5 — Listes

5.1 — Créer une liste

- Création par énumération : `L=[3, 7, 42]`
- Création par concaténation : `M = A + B` (concaténation des deux listes `A` et `B`)
- Création par répétition : `L = [0, 1]*50` (liste qui répète 50 fois `0, 1`)
- Avec range : `L=list(range(1, 50))` (listes des entiers entre 1 et 49), `L=range(5, 17, 3)` (liste des entiers par pas de 3, en partant de 5, jusqu'à 17 (exclus), soit `[5, 8, 11, 14]`)
- Par extraction : `L = A[2:9]` (éléments d'indices entre 2 et 8 de la liste `A`, voir les opérations ci-dessous)
- Création par compréhension : `L=[i**2 for i in range (100)]`

5.2 — Extraction sur des listes

Opération	Code	Remarque
longueur de `L`	`len(L)`	
élément d'indice *i* de `L`	`L[i]`	le dernier élément est d'indice `len(L)-1`
élément d'indice `len(L)-i` de `L`	`L[-i]`	ainsi `L[-1]` est le dernier, `L[-2]` l'avant-dernier, etc.
sous-liste de `L`	`L[i:j]`	sous-liste de `L` entre les indices *i* inclus et *j* exclus.
	`L[i:]`	sous-liste de `L` commençant à l'indice *i* inclus allant jusqu'à la fin
	`L[:j]`	sous-liste de `L` commençant au début et allant jusqu'à *j* exclus.

5.3 — Opération sur les listes

Il est fortement recommandé d'utiliser les *méthodes* suivantes. Celles-ci transforment directement la liste *L*. Il est donc inutile de faire une affectation (on code directement `L.append(3)` et pas `L = L.append(3)`)

Opération	Code
Ajoute un élément `v` à la fin de la liste	`L.append(v)`
Ajoute les éléments de la liste `A` à la fin de la liste `L`	`L.extend(A)`
Insère l'élément `v` à l'indice *i*	`L.insert(i, v)`
Supprime le dernier élément et le renvoie	`L.pop()`
Supprime l'élément d'indice `i` et le renvoie	`L.pop(i)`
Supprime la première occurrence de `x`	`L.remove(x)`
« Renverse » la liste	`L.reverse()`
Trie la liste	`L.sort()`

La structure générale pour créer une liste est

```
L = []
for k in range(15):
    L.append(k**2)
```

5.4 — Recopier une liste

Si vous avez besoin de faire une copie d'une liste `a` de modifier cette copie de façon indépendante de la liste d'origine, une instruction du style `b=a` ne correspondra pas à ce que vous chercher. En effet, toute modification sur `a` sera effectuée aussi sur `b`, et vice-versa. Il faut donc procéder ainsi :

```
    b = list (a)
    # ou
    b = a[:]
```

Attention ! Ne marche pas avec les tableaux `numpy`.

6 — Méthode sur les fichiers

Opération	Code	Remarque
Ouvre le fichier en lecture	`f = open("nomdufichier")`	ouvert en lecture (pas de modification possible)
Ouvre le fichier en écriture	`f = open("nomdufichier", "w")`	le contenu du fichier est effacé
Ouvre le fichier en écriture pour ajout	`f = open("nomdufichier", "a")`	les écritures se feront à partir de la fin
Renvoie le contenu du fichier	`f.read()`	le résultat est une chaîne de caractère
Lit une ligne du fichier	`f.readline()`	
Liste des lignes du fichier	`f.readlines()`	à utiliser dans une boucle `for ligne in f.readlines():`
Écrit la chaîne de caractère `s` dans le fichier	`f.write(s)`	
Écrit les éléments de `L`	`f.write(s)`	chaque élément sera une ligne de `f`
Ferme le fichier `f`	`f.close()`	pour que le fichier soit complètement écrit sur le disque dur.