

LES LISTES

OBJECTIFS

- savoir importer et utiliser un module
- savoir tirer des nombres aléatoires
- savoir créer une liste et manipuler ses éléments
- savoir encoder une matrice

Connaissances et savoir-faire à maîtriser

1) Les modules Python

Un module est une bibliothèque contenant un ensemble de fonctions déjà codées ayant pour objectif de simplifier votre vie de programmeur. En informatique, on dit que cela évite de « réinventer la roue ». Par exemple, pour utiliser les fonctions trigonométriques, on utilisera le module `math` via :

```
import math
print(math.cos(math.pi)) # affiche -1
```

On peut aussi importer toutes les fonctions du module et ainsi éviter de faire précéder l'appel de la fonction du nom du module via :

```
from math import *
print(cos(pi)) # affiche -1
```

Il existe une grande quantité de modules dont une liste est disponible sur le site de la doc Python à l'adresse suivante <https://docs.python.org/fr/3/py-modindex.html>

2) Le module `random`

Nous allons dans ce TP utiliser le module `random` qui permet de générer des nombres aléatoires :

```
import random
nb = random.randint(1, 10) # génère un nombre entier aléatoire entre 1 et 10
```

En informatique, on ne parle pas de nombres aléatoires mais de nombres **pseudo-aléatoires**. En effet, l'aléatoire étant par définition incompatible avec le calcul, *produire du pur hasard* sur un ordinateur qui n'est en réalité qu'une machine à calculer est un problème complexe. Toutefois, certains algorithmes sont capables de générer du *presque hasard* : on les appelle des générateurs de nombres pseudo-aléatoires. Ces générateurs ont besoin d'une **graine** (*seed* en anglais) pour être initialisé. Par défaut, Python utilise l'heure du système comme graine mais il est possible de la spécifier soi-même :

```
import random
random.seed(2020)
for i in range(0, 5):
    print(i, end = ' ') # affiche 10 10 3 8 8
```

Remarque 1 : dans l'exemple précédent, la graine étant toujours la même, la séquence de nombres générés sera toujours identique sur la machine sur laquelle le programme est exécuté.

3) Les listes

a. Création, ajout, suppression

À la différence des chaînes de caractères, les listes sont en python des **objets mutables**, autrement dit des structures de données que l'on peut modifier. On crée une liste via l'instruction `liste=[]` et on la modifie en utilisant les méthodes de listes `append` ou `remove` ou la fonction `del(liste[index])` :

```
liste = [] # création d'une liste vide
for i in range(0, 5):
    liste.append(i)
print(liste) # affiche [0, 1, 2, 3, 4]
del(liste[2]) # supprime l'élément de rang 2 dans la liste
print(liste) # affiche [0, 1, 3, 4]
liste.remove(3) # supprime la valeur 3 de la liste
print(liste) # affiche [0, 1, 4]
```

b. Les matrices

Une matrice de taille (ou de dimension) $m \times n$ est un tableau de nombres (appelés coefficients) composée de m lignes et n colonnes. Dans la matrice M de taille 4×3 ci-dessous par exemple, 17 est le coefficient située à la 3^{ième} ligne et 2^{ième} colonne :

$$M = \begin{bmatrix} 12 & 1 & 10 \\ 3 & 45 & 20 \\ 5 & 17 & 30 \\ 4 & 6 & 50 \end{bmatrix}$$

Pour représenter et manipuler ce genre d'objet sous Python, on peut utiliser **une liste de listes**. Autrement dit, une liste dont chaque élément est lui-même une liste et représente une ligne de la matrice :

```
m = [[12, 1, 10], [3, 45, 20], [5, 17, 30], [4, 6, 50]]
for lig in range(4): # parcourt ligne par ligne
    for col in range(3): # puis colonne par colonne
        print(m[lig][col], end = ' ') # évite le saut de ligne
    print("") # va à la ligne après l'affichage d'une ligne de la matrice
```

Remarque 2 : il existe en Python une bibliothèque spécifique pour effectuer du calcul matriciel, c'est le module `numpy`.

Remarque 3 : une image (ou bien l'écran d'un ordinateur) peut être vue comme une grille de pixel que l'on peut représenter par une matrice dont chaque coefficient représente la couleur du pixel.

➤ Exercices à réaliser

Pour chaque exercice, ouvrez le fichier mentionné dans l'énoncé et lisez les consignes qui reprennent ou complètent les notions de ce document.

EXERCICE 1

Ouvrir le fichier TP-listes-exo1.py, et répondre aux questions suivantes :

1. Créer une liste vide et écrire une fonction `remplirListe` qui remplit la liste passée en paramètre de 10 valeurs entières tirées au hasard dans l'intervalle [1 ; 100] :

```
remplirListe(liste)
|- ENTREES : une variable liste de type list
|- SORTIES : aucune
```

2. Écrire une fonction `valMax` qui renvoie le plus grand élément de la liste passée en paramètre :

```
valMax(liste)
|- ENTREES : une variable liste de type list
|- SORTIES : le plus grand élément de liste
```

3. Même question avec la fonction `valMin` :

```
valMin(liste)
|- ENTREES : une variable liste de type list
|- SORTIES : le plus petit élément de liste
```

EXERCICE 2

Ouvrir le fichier TP-listes-exo2.py, et répondre aux questions suivantes :

1. Créer une liste vide et écrire une fonction `remplirListe` qui remplit la liste passée en paramètre de 10 valeurs entières tirées au hasard dans l'intervalle [1 ; 100] :

```
remplirListe(liste)
|- ENTREES : une variable liste de type list
|- SORTIES : aucune
```

2. Écrire une fonction `permuter` qui échange les éléments de la liste de rang `i` et `j` :

```
permuter(liste, i, j)
|- ENTREES : liste est du type list, i et j sont de type int
|- SORTIES : aucune
```

```
liste = [2, 6, 1, 7]
permuter(liste, 1, 3)
print(liste) # affiche [2, 7, 1, 6]
```

3. Écrire une fonction `trierListe` qui trie les éléments d'une liste par ordre croissant :

```
trierListe(liste)
|- ENTREES : liste est du type list
|- SORTIES : aucune
```

On pourra utiliser un double parcours de type double FOR imbriqué ainsi que la fonction `permuter` définie précédemment.

EXERCICE 3

Ouvrir le fichier TP-listes-exo3.py, et répondre aux questions suivantes :

1. Écrire une fonction `create_matrix` qui crée une matrice de taille $m \times n$ de nombres entiers compris entre -10 et 10 :

```
create_matrix(m, n)
|- ENTREES : m et n sont des entiers
|- SORTIES : une liste de listes
```

2. Écrire une fonction `display_screen` qui utilise la matrice comme source de données et l’affiche sur un écran de taille $m \times n$:





```
display_screen(mat, m, n)
|- ENTREES : mat est une liste de listes, m et n sont des entiers
|- SORTIES : aucune
```

On se conformera au modèle suivant pour le formattage des données :

- si le coefficient de la matrice est strictement positif, on affiche +
- s’il est strictement négatif, on affiche -
- s’il vaut 0, on affiche la lettre 'o'

3. Utiliser les fonctions du TP précédent pour trier chaque ligne de la matrice avant de l’afficher.

🔗 Compétences à valider

LES LISTES					
MODULES	Je sais importer un module				
	Je sais appeler les fonctions d’un module				
	Je connais le module random pour tirer des nombres pseudo-aléatoires				
LISTES	Je sais créer une liste vide				
	Je sais ajouter des éléments à une liste				
	Je sais supprimer par index ou par valeur les éléments d’une liste				
	J’ai compris le principe de liste de listes qui permet d’encoder une matrice				