|  |  |  |
| --- | --- | --- |
| Numérique et Sciences Informatiques | | |
| 4h | **Types construits** |  |
| **Objectif** : savoir manipuler les tuples et les listes. | | |
| **Matériel**: feuille de papier et Python | | |

On connait les **type simples** int, float, str, bool qui permettent de contenir une valeur.

Un **type construit** permettra de contenir plusieurs valeurs dans une seule variable de type tuple ou tableau/liste ou dictionnaire.

**Les tuples**

**Testez les instructions suivantes dans la console :**

Création d'un **tuple** (= **p-uplet** en mathématique) constitué des entiers 5, 8, 6 et 9 :

mon\_tuple = (5, 8, 6, 9)

Accéder à l'élément placé à l'indice 2 dans le tuple :

mon\_tuple[2]

Impossibilité de remplacer l'élément placé à l'indice 2 par un autre (le tuple est un objet **immutable**) :

mon\_tuple[2]=3

Pour connaitre le nombre d’éléments dans le tuple :

L= len(mon\_tuple)

Le tupleest de type **tuple** :

type(mon\_tuple)

Les éléments d’un tuple de taille n sont indexés de 0 à n-1 (n = nombre d'éléments).

**Code 1**\*: Quel sera l'affichage correspondant au programme suivant ?

def add(a, b):

c = a + b

return (a, b, c)

mon\_tuple = add(5, 8)

print(mon\_tuple[0],"+",mon\_tuple[1],"=",mon\_tuple[2])

5 + 8 = 13

**Code 2**\* : Quel sera l’affichage du programme suivant ?

mon\_tuple=(1,2,3)

a,b,c=mon\_tuple

print(a,",",b,",",c)

1 , 2 , 3

**Code 3**\*\*: Ecrivez un programme qui affiche les éléments de **mon\_tuple=('Angers','Beaucouzé','Avrillé','Trélazé')** comme suit :

élément 0 : Angers

élément 1 : Beaucouzé

élément 2 : Avrillé

élément 3 : Trélazé

Le programme sera capable de s'adapter à un tuple de plus de 4 éléments.

**Code 4\*\* :** Ecrivez la fonction deux\_nombres() qui demande à l'utilisateur de saisir 2 nombres décimaux.

Cette fonction renverra un tuple contenant ces deux nombres.

Le programme appellera cette fonction et affichera le tuple créé.

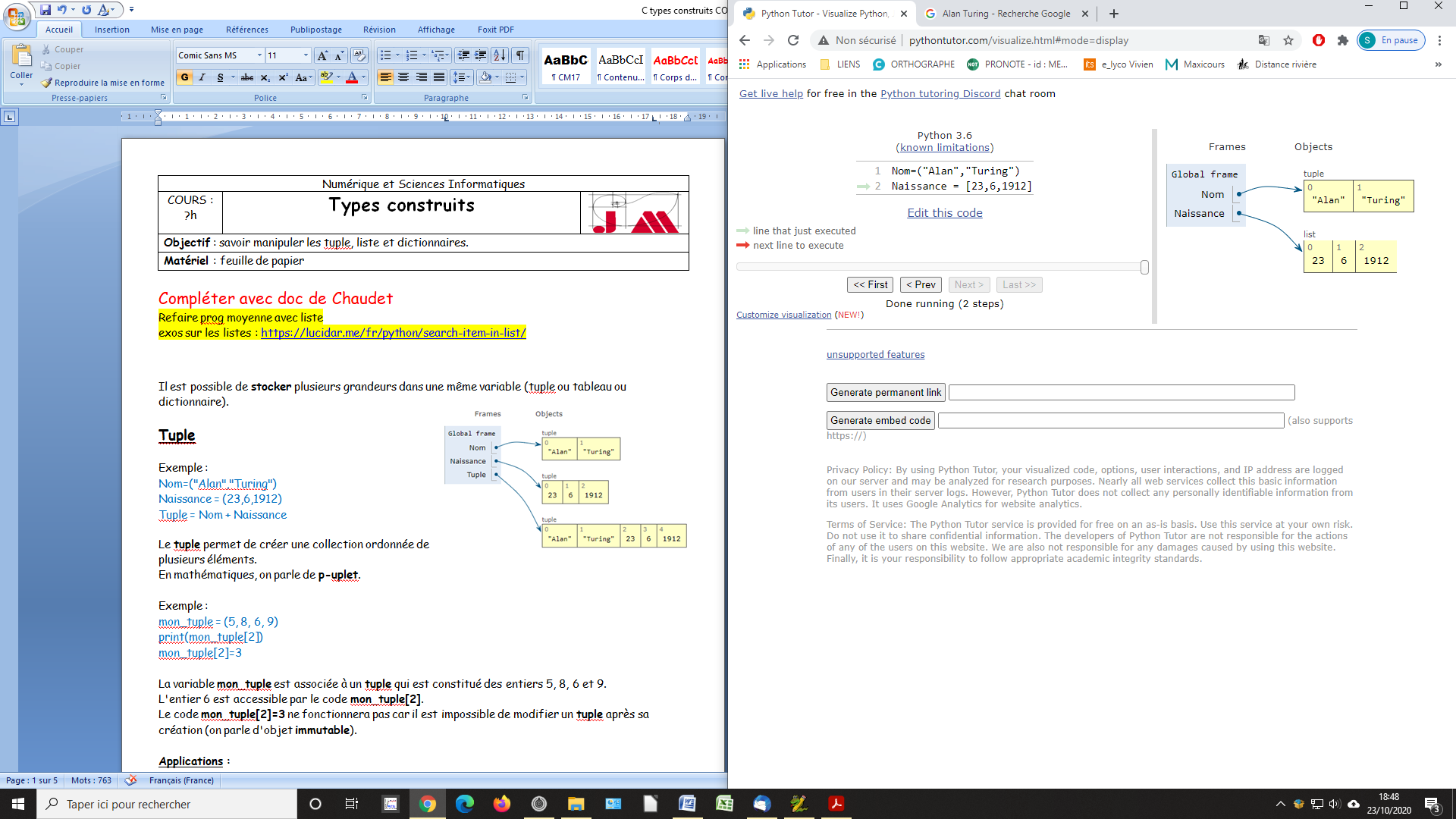
**Code 5\*\*\*** : Ecrivez la fonction milieu(A\_loc,B\_loc) qui prend en argument (paramètres) deux tuples représentant les coordonnées x/y de deux points.

Cette fonction renverra le tuple composé des coordonnées du point situé au milieu.

Le programme appellera cette fonction et affichera le milieu du segment A=(4;8) B=(-2;5).

Rappel : les coordonnées du milieu d’un segment sont ( ;).

**Les tableaux (listes)**



**Exemple :**

Nom=("Alan","Turing")

Naissance = [23,6,1912]

ma\_liste = [1.56, "tabouret", 3,'a']

Les **tableaux** (= **listes** en Python) sont des **tuples** modifiables (on parle d'objets **mutables**).

Conseil : plutôt qu'utiliser une **liste** qu'on ne modifiera pas, il sera préférable d'utiliser un **tuple** car les opérations sur les **tuples** sont plus rapidement exécutées par Python.

**Testez les instructions suivantes dans la console :**

Création d'une **liste** constituée des entiers 5, 8, 6 et 9 :

ma\_liste = [5, 8, 6, 9]

Accéder à l'élément placé à l'indice 2 dans la liste :

ma\_liste[2]

Remplacer l'élément placé à l'indice 2 par un autre (une liste est un objet **mutable**) :

ma\_liste[2]=3

Affichage de [5,8,3,9] :

ma\_liste

Ajout dans la liste de l'élément 7 :

ma\_liste.append(7)

Affichage de [5,8,3] étant les éléments placés de l'indice 0 à l'indice 3 non compris :

ma\_liste[0:3]

Suppression de la liste de l'élément 8 :

ma\_liste.remove(8)

Affichage de [5,3,9,7] :

ma\_liste

Affichage du dernier élément de la liste :

ma\_liste[-1] = ma\_liste[len(ma\_liste)-1]

Affichage suivant [5, 3] des éléments placés de l'indice 0 à l'indice -2 non compris :

ma\_liste[0:-2]

La listeest de type **list** :

type(ma\_liste)

**Code 1\*** : Quel sera l'affichage correspondant au programme suivant ?

groupe=["Pierre","Paul", "Jacques"]

indice = groupe.index("Paul")

groupe[indice]="Robert"

groupe.append("Jean")

groupe.remove("Jacques")

print(groupe)

[ « Pierre », « Robert », « Jean » ]

**Code 2\*** : Quel sera l'affichage correspondant au programme suivant ?

groupe=["Elodie","Pauline"]

groupe.append(groupe[0])

groupe.remove(groupe[0])

print(groupe)

[« Elodie », « Pauline »]

**Code 3\*\* :** Ecrivez les codes permettant d'effectuer les actions suivantes :

Créer une liste (ma\_liste) contenant les nombres allant de 1 à 9 \*

Afficher la taille de la liste \*

Afficher l’élément d’indice 5 \*

Ajouter le nombre 10 à la fin de la liste et afficher la nouvelle liste \*

Afficher l’avant dernier élément de la liste \*\*

Supprimer le nombre 5 de la liste et afficher la nouvelle liste \*

Supprimer l'élément d'indice 7 de la liste et afficher la nouvelle liste \*\*

**Code 4\*\* :** Ecrivez un programme qui demande à un utilisateur de saisir une liste de notes (l'arrêt de la saisie se fera avec une note non comprise entre 0 et 20). Le programme affichera la liste.

**Parcours de tableau (liste)**

Lors de l’utilisation d’une liste, on souhaite souvent parcourir celle-ci pour sélectionner, comparer ou modifier ses éléments.

**Méthode n°1 pour parcourir une liste :**

for i in range(len(ma\_liste)) :

print(ma\_liste[i])

Ici on parcourt et on effectue des instructions sur les éléments à l’aide d’un indice(=index) dans une boucle bornée.

La variable i prend pour valeur un entier entre 0 et len(ma\_liste)-1.

Avantage : cela permet de modifier les valeurs de la liste et de connaître l’indice d’un élément.

**Méthode n°2 pour parcourir une liste :**

for élément in ma\_liste :

print(élément)

Ici on parcourt et on effectue des instructions sur les éléments à l’aide d’une boucle bornée.

La variable **élément** prend pour valeur les éléments de la liste dans l’ordre de la liste.

Avantage : facile à écrire.

Inconvénient : il n’est pas possible de modifier la liste ou de connaître la position de l’élément (indice) dans la liste.

**Méthode n°3** **pour parcourir une liste :**

i=0

while (i<len(ma\_liste)) :  
 print(ma\_liste[i])  
 i=i+1

Même principe qu'avec la boucle for mais il faut initialiser l’indice et l’incrémenter nous même.

Avantage : cela permet de proposer d’autres conditions d’arrêts de la boucle, comme par exemple arrêter le parcours lorsqu'un élément de la liste à été trouvé.

**Code 5\*\* :** Ecrivez un programme qui inverse les éléments d'une liste (vous pouvez ranger ces éléments dans une nouvelle liste).

Exemple :

Liste d'entrée = [5,6,7,8,9]

Liste de sortie = [9,8,7,6,5]

**Code 6\*\* :** Ecrivez un programme qui retourne le nombre d’occurrences d’un nombre dans une liste.

Le nombre d’occurrences correspond au nombre d’apparitions d'un élément.

Exemple :

ma\_liste=[18, 13, 10, 7, 19, 19, 18, 0, 8, 12, 0, 15, 2, 8, 4, 4, 18, 5, 3, 18, 6, 12, 12, 15, 11, 8, 12, 13, 10, 20, 2, 15, 11, 11, 4, 18, 17, 0, 17, 19, 13, 7, 14, 5, 2, 13, 7, 13, 17, 9, 1, 7, 7, 3, 3, 7, 3, 4, 11, 7, 9, 2, 17, 2, 3, 5, 0, 15, 18, 0, 7, 7, 7, 15, 11, 11, 20, 8, 3, 17, 3, 18, 20, 16, 10, 18, 8, 7, 9, 17, 18, 19, 14, 11, 13, 12, 20, 12, 19, 15]

Le nombre d'occurrence de 7 dans ma\_liste est 11.

**Tableau donné en compréhension**

Nous avons vu qu'il était possible de remplir un tableau en rentrant ses éléments avec la méthode **append**.

Il est aussi possible de remplir un tableau grâce à la technique de **compréhension**.

**Exemple :**

ma\_liste = [p for p in range(200,400)]

print(ma\_liste)

**print(ma\_liste)** provoquera l'affichage suivant : [200,201,202, ... ,399].

**Code 1\*** : Quel sera l'affichage correspondant au programme suivant ?

L = [1, 7, 15, 5, 20, 10, 8]

ma\_liste = [p for p in L if p <10]

print(ma\_liste)

[1, 7, 5, 8]

**Code 2\*** : Quel sera l'affichage correspondant au programme suivant ?

L = [1, 7, 15, 5, 20, 10, 8]

ma\_liste = [p\*\*2 for p in L if p < 10]

print(ma\_liste)

[1, 49, 25, 64]

**Code 3\*\*** : Ecrivez un programme qui crée un tableau donné en compréhension contenant uniquement les nombres impairs entre 0 et 100.

**Code 4\*\*** : Ecrivez un programme qui crée un tableau donné en compréhension contenant les nombres pairs du tableau suivant T=[1,2,3,4,5,6,7,8,9].

**Structure générale d'un tableau donné en compréhension :**

[expression(element) **for** element **in** structure\_a\_parcourir **if** filtre(element)]

Le filtre n’est pas forcément nécessaire

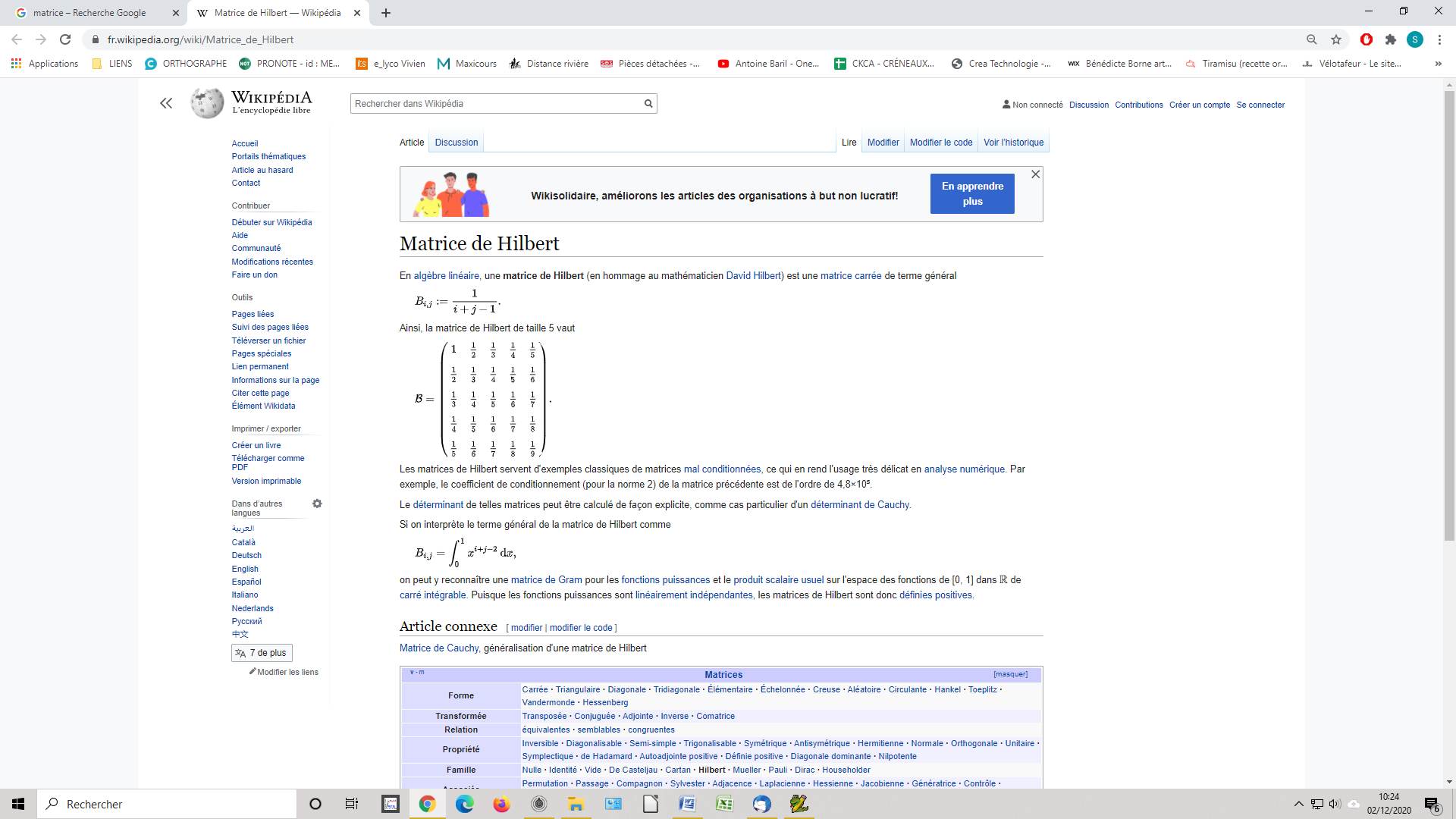
Souvent, la structure à parcourir est un intervalle de 0 à un entier n (non compris) :

[expression(i) for i in range(n)]

**Tableaux de tableaux (ou liste de liste)**

Chaque élément d'un tableau peut aussi être un tableau, on parle de **tableau de tableau.**

Cela ressemble à une **matrice** en mathématique.

**Exemple :**

m = [[1, 3, 4], [5 ,6 ,8], [2, 4, 3], [7, 8, 15]]

print(m[2])

print(m[2][1])

**print(m[2])** provoquera l'affichage suivant : [2,4,3].

**print(m[2][1])** provoquera l'affichage suivant : 4

**Code 1\*** : Quel sera l'affichage correspondant au programme suivant ?

m = [1, 2, 3]

mm = [m, m, m]

print(mm)

[[1, 2, 3], [1, 2, 3], [1, 2, 3]]

**Code 2\*\*** : Quel sera l'affichage correspondant au programme suivant ?

m = [[1, 3, 4],

[5, 6, 8],

[2, 1, 3],

[7, 8, 15]]

nb\_ligne = len(m)

nb\_colonne = len(m[0])

for i in range(nb\_ligne):

for j in range(nb\_colonne):

a = m[i][j]

print(a, end="")

1345682137815

**Code 3\*\*** : Modifiez le programme précédent pour avoir l'affichage : **48315**

**Listing des méthodes sur les objets de type liste**

Une **méthode** est une fonction qui "appartient à" un **objet** (programmation objet au programme de terminale spé NSI).

On peut appliquer les **méthodes**suivantes sur la liste **ma\_liste** qui est un **objet** pour Python :

* **ma\_liste**.append(x)
* **ma\_liste**.remove(x)
* **ma\_liste**.index(x)
* **ma\_liste**.extend([liste])
* **ma\_liste**.insert(i, x)
* **ma\_liste**.pop(i)
* **ma\_liste**.clear()
* **ma\_liste**.count(x)
* **ma\_liste**.sort()
* **ma\_liste**.reverse()
* **ma\_liste**.copy()

Explication de chaque méthode : https://docs.python.org/fr/3/tutorial/datastructures.html

Exemples de traitement de listes : https://lucidar.me/fr/python/get-list-element/

**Appel de fonction : passages de liste en paramètre**

Remarque : étant donné que dans le code de gauche liste et liste\_loc sont les mêmes listes, il n'est pas utile de passer les listes en passage de paramètre :

|  |  |
| --- | --- |
| **Passage de paramètre par référence :** | **Sans passage de paramètre :** |
| **def AjouterListe(liste\_loc, x\_loc):**  **liste\_loc.append(x\_loc)**  **return(liste\_loc)**  liste=[]  x=0  while(x!=-1):  x=int(input("Entrer un nombre :"))  liste=AjouterListe(liste,x)  print(liste) | **def AjouterListe(x\_loc):**  **liste.append(x\_loc)**  liste=[]  x=0  while(x!=-1):  x=int(input("Entrer un nombre :"))  AjouterListe(x)  print(liste) |

**Cependant il est important de donner les entrées et sorties des fonctions, on pourra donc continuer à passer les listes en paramètres pour respecter ce principe.**

**Copie de liste (utile pour le projet Mastermind)**

A tester sous **pythontutor** : <http://pythontutor.com/visualize.html#mode=edit>

NB : les Frames sont des registres du CPU qui mémorisent des pointeurs vers la mémoire vive.

Nous avons 5 joueurs pour constituer une équipe de 3 joueurs.

Le match n°1 se fera avec les joueurs 1, 2 et 3.

Pour le match n°2 on remplacera le joueur 2 par le joueur 5.

On désire garder la composition de chaque match dans des listes différentes.

|  |  |
| --- | --- |
| **Sans copy de liste :** | **Avec copy de liste :** |
| match\_1=["J1","J2","J3"]  match\_2=match\_1  match\_2[1]="J5"  print(match\_1)  print(match\_2)  #adresse mémoire de stockage :  print(id(match\_1))  print(id(match\_2)) | match\_1=["J1","J2","J3"]  **match\_2=match\_1.copy()**  match\_2[1]="J5"  print(match\_1)  print(match\_2)  #adresse mémoire de stockage :  print(id(match\_1))  print(id(match\_2)) |
|  |  |

Remarque : on peut remplacer match\_2=match\_1.copy() par match\_2=match\_1[:] ou match\_2=list(match\_1).

|  |  |
| --- | --- |
| **Passage de paramètre par référence :** | **Passage de paramètre par copie :** |
| def AjouterListe(liste\_loc, x\_loc):  liste\_loc.append(x\_loc)  liste=[]  x=0  while(x!=-1):  x=int(input("Entrer un nombre :"))  AjouterListe(liste,x)  print(liste) | def AjouterListe(liste\_loc, x\_loc):  liste\_loc.append(x\_loc)  liste=[]  x=0  while(x!=-1):  x=int(input("Entrer un nombre :"))  AjouterListe(liste.copy(),x)  print(liste) |
| Quel sera l'affichage si l'exécution du programme est la suivante :  Entrez un nombre : 4  Entrez un nombre : 5  Entrez un nombre : -1  [4, 5, -1] | Quel sera l'affichage si l'exécution du programme est la suivante :  Entrez un nombre : 4  Entrez un nombre : 5  Entrez un nombre : -1  [] |