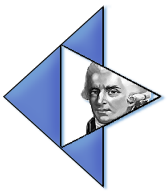
	Université de Corse - Pasquale PAOLI
	Diplôme : **Licence SPI 3ème année** — 2025-2026
	UE : Ateliers de programmation
	**Atelier 2 : Listes et boucles** - Parcours itératifs de listes : boucles for et while - Algorithmes standard (calculs, comptages, recherches) - Tests unitaires
	Enseignants : Paul-Antoine BISGAMBIGLIA, Paul PINA, Evelyne VITTORI

On s'efforcera de typer les fonctions et les paramètres, et aussi d'écrire des programmes de tests qui vérifient plusieurs cas, par exemple liste vide, liste avec un nombre paire d'éléments, liste avec un nombre impaire d'éléments.

PARTIE 1 - EXERCICES ESSENTIELS

Exercice 1 -Algorithmes classiques

1) fonction **`moyenne_dune_lst(ma_liste : list)-> float:`** qui admet en paramètre une liste d'entiers **`ma_liste`** et renvoie la moyenne des valeurs de la liste. La fonction doit renvoyer 0 si la liste est vide.
2) fonction **`nb_sup (ma_liste : list , element : int) -> list`** qui admet en paramètre une liste d'entiers **`ma_liste`** et un entier **`element`** et retourne le nombre de valeurs strictement supérieures à **`element`**
   Définissez trois versions de la fonction nb_sup :
   - une version avec une boucle for basée sur les indices (for i in range(.....))
   - une version avec une boucle for basée sur les éléments (for e in **`liste`**)
   - Une version avec une boucle while
3) fonction **`moy_sup (ma_liste : list,elt : int) -> float :`** qui admet en paramètre une liste d'entiers **`ma_liste`** et un entier elt, et retourne la moyenne des valeurs de la liste strictement supérieures à elt.
4) fonction **`val_max(lst : list) -> float :`** qui prend en paramètre une liste d'entiers et retourne la valeur maximale de cette liste.
5) fonction **`ind_max(lst) -> int :`** qui prend en paramètre une liste d'entiers et retourne l'indice de l'élément maximal de cette liste. La liste est supposée sans répétition.

Remarques:

- *vos fonctions ne doivent rien afficher mais retourner un résultat.*
- *vous prendrez soin d'éviter les répétitions de code entre vos fonctions*
- *vous favoriserez la lisibilité du code*
- *mettez les docstrings et les annotations de type*
- *Réaliser une fonction de tests qui automatise les tests.*

Exercice 2 - Recherches, fonctions booléennes, boucles while

Définissez les fonctions suivantes :

1) fonction **position(lst,elt)** qui admet en paramètres une liste lst d'entiers et un entier elt et retourne l'indice de l'entier elt dans la liste lst. Si l'entier elt n'est pas présent

dans la liste, la fonction retourne la valeur -1. Vous supposerez que tous les entiers présents dans la liste ne peuvent apparaitre qu'une et une seule fois et que la liste n'est pas triée.

a. Définissez une première version de la fonction en utilisant un parcours systématique (boucle for)
b. Définissez une deuxième version de votre fonction en utilisant une boucle while qui s'arrête lorsque l'élément e est trouvé ou quand on arrive en fin de liste dans le cas où l'élément n'est pas présent.

2) fonction **nb_occurrences(lst,e)** qui admet en paramètres une liste lst d'entiers et un entier e, et retourne le nombre d'occurrences de l'entier e dans la liste lst. On suppose que la liste lst peut comporter des répétitions.

NB : pour tester votre fonction positionTri, vous pourrez utiliser la fonction sorted(L) qui renvoie une nouvelle liste correspondant à la liste L triée ou la méthode sort de la classe List qui effectue un tri sur la liste L elle-même (L.sort()).

Exercice 3 - Séparation

lst étant supposée non triée, on désire classer les nombres de la liste lst dans une deuxième liste lsep de même dimension en plaçant ensemble les nombres négatifs (sans les classer entre eux), ensemble les nombres nuls et ensemble les nombres positifs (sans les classer entre eux). La liste lsep sera remplie au fur et à mesure du parcours de lst, en plaçant les nombres négatifs à gauche en partant de la première case et les nombres positifs à droite en partant de la dernière case. Les cases comportant des zéros seront situées entre les nombres négatifs et les nombres positifs.

Définissez une fonction **separer()** qui prend en paramètre une liste d'entiers lst et retourne la liste lsep construite selon le principe décrit précédemment.

Exercice 4 - FizzBuzz

Un exercice simple et récurrent dans de nombreux tests de recrutements.
Ecrire une fonction qui itère sur les nombres de 1 à n, et qui pour chaque nombre i, écrit :

- Fizz si le nombre est multiple de 3 ;
- Buzz si le nombre est multiple de 4 ;
- FizzBuzz si le nombre est à la fois multiple de 3 et de 4;
- i si aucune des conditions n'est vérifiée.

Exercice 5 - Rendu de monnaie

Ecrire une fonction, qui prend en paramètre une somme d'argent à rendre, et une collection (par exemple une liste) indiquant le nombre de pièces et billets à disposition, et retourne le nombre de pièces/billets de chaque valeur à rendre.

PARTIE 2 - EXERCICES APPROFONDIS

Exercice 6 - Agencement d'objets dans deux vitrines

On dispose de deux vitrines contenant chacune nbEmplacements emplacements (un entier), et on a lObjets objets (une liste d'entiers), à afficher dans celles-ci. Pour des raisons esthétiques, chaque vitrine ne peut pas contenir deux objets (entiers) identiques. On cherche à savoir s'il existe une configuration (un couple ou une liste de deux listes d'entiers) qui permet d'afficher

tous les objets en vitrine et qui respecte cette contrainte. Ecrivez un programme qui permet de répondre à cette question en renvoyant le couple (ou la liste) des deux listes d'entiers à placer dans chacune des deux vitrines.
Exemple : pour les entrées nbEmplacement = 4 et lObjets = [1,2,2,3,4,5,5], il existe au moins une solution qui est ([1,2,3,5], [2,4,5]).

Exercice 7 - Tests, recherche d'erreurs

Question 1 : On considère une fonction **present(L,e)** qui admet en paramètres une liste d'entiers L et un nombre entier e et retourne un booléen True si l'élément e est présent dans la liste L et False sinon.
Définissez une fonction **test_present(*present:callable*)** qui admet en **paramètre une fonction *present*** et effectue les tests successifs suivants:

- Appel de la fonction present sur une liste vide et affichage "SUCCES : test liste vide" si la fonction renvoie False et "ECHEC: test liste vide" si la fonction renvoie True.
- Appels successifs de la fonction present sur une liste d'entiers L comportant environ 10 valeurs dans 4 cas:
  - un entier situé en debut de liste (position 0): "test debut"
  - un entier situé en fin de liste: "test fin"
  - un entier situé en milieu de liste: "test milieu"
  - un entier non présent dans la liste: "test absence"

  Pour chacun de ces cas, un message "SUCCES: test ..." ou "ECHEC: test ..." sera affiché.

Question 2 : Testez le fonctionnement de votre fonction de test sur les quatre versions de la fonction present(L,e) proposées ci-dessous. Ces fonctions sont toutes fausses!

Question 3 : En vous aidant des résultats de tests, étudiez à présent le code de chacune des versions, identifiez les erreurs commises et proposez des corrections.

```
#VERSION 1
def present1 (L, e) :
        for i in range (0, len(L), 1) :
                if (L[i] == e) :
                        return(True)
                 else :
                        return (False)
return (False)
```

```
#VERSION 2
def present2 (L, e) :
        b=True
        for i in range (0, len(L), 1) :
                if (L[i] == e) :
                        b=True
                 else :
                        b=False
        return (b)
```

```
#VERSION 3
def present3 (L, e) :
```

```
        b=True
        for i in range (0, len(L), 1) :
                return (L[i] == e)
```

```
#VERSION 4
def present4 (L, e) :
        b=False
        i=0
        while (i<len(L) and b) :
                if (L[i] == e) :
                        b=True
        return (b)
```

Question 4 : On considère à présent une fonction **pos** admettant en paramètre une liste d'entiers L et un entier e, et retournant la liste des positions d'apparition de e dans L:

Ex= Lres=pos([3,4,5,7,2,7], 7) = [3,5]

1) Définissez une fonction **test_pos(fonctionPos)** de test similaire à la fonction test_present de la question précédente
2) Utilisez votre fonction test_pos pour tester les quatre versions de la fonction pos proposées ci-dessous.
3) Pour chacune des versions, identifiez les erreurs commises et proposez des corrections.

```
#VERSION 1
def pos1(L, e) :
        Lres = list(L)
        for i in range (0, len(L), 1) :
                if (L[i] == e) :
                        Lres += [i]
        return Lres
```

```
# VERSION 2
def pos2(L, e) :
        Lres = list(L)
        for i in range (0, len(L), 1) :
                if (L[i] == e) :
                        Lres[i] = i
        return Lres
```

```
# VERSION 3
def pos3(L, e) :
        nb= L.count(e)
        Lres = [0]*nb
        for i in range (0, len(L), 1) :
                if (L[i] == e) :
                        Lres.append(i)
        return Lres
```

```
# VERSION 4
def pos4(L, e) :
        nb= L.count(e)
        Lres = [0]*nb
        j=0
```

```
for i in range (0, len(L), 1) :
        if (L[i] == e) :
                Lres[j]= i
return Lres
```

Exercice 8 - Gestion de score de tennis

On cherche à développer une petite application permettant d'incrémenter en direct les scores des tennismen.

1. Répertorier dans un document les règles du tennis concernant l'évolution du score. On prendra en compte le *tie break*.
2. Présenter plusieurs cas d'utilisation : comment gagne-t-on un jeu? un set? un match?
3. Proposer un algorithme en langage naturel qui effectue les vérifications et les modifications lorsqu'un joueur marque un point. On précisera les éventuelles structures de données utilisées.
4. Implémenter le programme. Chaque point marqué fera l'objet d'un *log* textuel.

Exercice 9 - Evaluation d'un polynôme

Rappel : un polynôme est une expression de la forme $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$.

1. On se place dans le cas d'un polynôme de degré 2 : $ax^2 + bx + c$ Combien y a-t-il d'additions ? de multiplications ?
2. Même question mais dans le cas d'un polynôme de degré n.
3. Ecrire une fonction naïve qui évalue la valeur d'un polynôme.
4. Montrer que $a_2 x^2 + a_1 x + a_0 = \Big(\big((a_2 x + a_1)x + a_0\big)\Big)$. Cette forme est appelée *forme de Horner*. Combien d'additions et de multiplications sont réalisées ?

   Forme de Horner générale : $a_0 + x\big(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n)\cdots)\big)$
5. Ecrire une fonction qui évalue le polynôme en s'appuyant sur la forme de Horner : on part de a_n, puis à chaque itération, on multiplie par x et on ajoute a_{n-1}.