	Université de Corse - Pasquale PAOLI
	Diplôme : **Licence SPI 3ème année** — 2023-2024
	**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, Marie-Laure NIVET, Evelyne VITTORI

PARTIE 1 - EXERCICES ESSENTIELS

EXERCICE 1 - Calculs, Comptage, Maximum, fonctions entières, structures itératives

1) Définissez trois versions d'une fonction **somme(L)** qui admet en paramètre une liste d'entiers L et retourne la somme des valeurs de la liste :
   - 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 L)
   - une version basée sur une boucle while

Quelle est la version qui vous semble la plus adaptée ?

2) Tester votre fonction somme en utilisant la fonction test_exercice1 suivante :

```
def test_exercice1 ():
        print("TEST SOMME")
        #test liste vide
        print("Test liste vide : ", somme([]))
        #test somme 11111
        lst2test1=[1,10,100, 1000,10000]
        print("Test somme 1111 : ", somme(lst2test1))
```

Pour chacune des fonctions des questions suivantes, vous complèterez la fonction de test par des tests pertinents permettant de prouver la validité de vos fonctions.

3) fonction **moyenne(L)** qui admet en paramètre une liste d'entiers L et renvoie la moyenne des valeurs de la liste. La fonction doit renvoyer 0 si la liste est vide.
4) fonction **nb_sup (L,e)** qui admet en paramètre une liste d'entiers L et un entier e et retourne le nombre de valeurs strictement supérieures à e
   Définissez deux 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 L)
5) fonction **moy_sup (L,e)** qui admet en paramètre une liste d'entiers L et un entier e, et retourne la moyenne des valeurs de la liste strictement supérieures à e.
6) fonction **val_max(L)** qui prend en paramètre une liste d'entiers et retourne la valeur maximale de cette liste.
7) fonction **ind_max(L)** 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*

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.

3) fonction **est_triee(lst)** qui admet en paramètres une liste lst d'entiers et retourne un booléen vrai si la liste est triée par ordre croissant, faux sinon.

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 dès qu'un élément d'indice i est identifié comme supérieur à l'élément d'indice i+1 ou quand on arrive en fin de liste dans le cas où la liste est effectivement triée.

A votre avis quelle est la meilleure version ?

4) fonction **position_tri**(lst,e) similaire à la fonction position de la question 1 mais qui suppose que la liste lst est triée et utilise l'algorithme de recherche dichotomique décrit en annexe.

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()).

5) fonction **a_repetitions(lst)** qui admet en paramètres une liste lst d'entiers et retourne un **booléen** True si la liste lst comporte des répétitions de valeurs et False sinon.

Le principe de l'algorithme consiste à utiliser une liste T.

```
# initialisation de la liste T à la liste vide
#Pour chaque élément de la liste L (boucle while):
        si l'élément n'est pas présent dans la liste T (NB: on peut utiliser l'opérateur in (if x in T)):
                on ajoute l'élément dans la liste T
        sinon
                on peut conclure qu'il y a une répétition
```

Utilisez une boucle while qui s'arrête lorsqu'une répétition est trouvée ou quand on arrive en fin de liste dans le cas où l'élément n'est pas présent.

EXERCICE 3 - Séparation

L étant supposée non triée, on désire classer les nombres de la liste L 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 L, 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 L et retourne la liste LSEP construite selon le principe décrit précédemment.

PARTIE 2 - EXERCICES APPROFONDIS

EXERCICE 4 - Application : fonctions - histogramme

Le programme à définir doit permettre d'étudier les propriétés d'une fonction (injectivité, surjectivité, bijectivité) représentée par une liste d'entiers.
On s'intéresse aux fonctions de la forme f : I → J où I=[0 ;MAXTAILLE] et J=[0 ; MAXVALEUR] ou MAXTAILLE et MAXVALEUR sont des entiers positifs.
ex: L= f(0),f(1),f(2,f(3), ... f(MAXTAILLE)]
Le programme final devra permettre l'affichage des propriétés de cette fonction.

Question 1 : Etude des propriétés de fonctions

Définissez les quatre fonctions détaillées dans les paragraphes suivants ainsi qu'une procédure de test associée.

1) fonction **histo(F)** (comptage de fréquences)

Cette fonction admet en paramètre la liste d'entiers F définissant une fonction et renvoie une liste d'entiers H représentant l'histogramme de F.
La liste H a pour but de comptabiliser le nombre d'apparitions de chaque valeur de l'ensemble J dans la liste F. La liste H comporte donc autant de cases que de valeurs possibles dans F. H est donc indicé de 0 à MAXVALEUR.
L'algorithme commence par initialiser à 0 les cases de la liste H puis parcourt la liste F en incrémentant les cases concernées de la liste H.
Exemple:
F=[6,5,6,8,4,2,1,5]
H est une liste indicée de 0 à 8 (la valeur maximum de F)
H=[0,1,1,0,1,2,2,0,1]

2) fonction **est_injective(F)**

Cette fonction renvoie la valeur True si la fonction représentée par la liste F est une injection. Dans le cas contraire, elle renvoie False.
On dit qu'une fonction f : I →J est injective si tout élément de f(I) est l'image d'un seul élément de I. En d'autres termes, f est injective si chaque valeur de J possède au plus un antécédent dans I.
Pour savoir si f est injective, il suffit de vérifier que tous les éléments de la liste H (obtenue par appel de la fonction histo) ont une valeur inférieure ou égale à 1.

Exemples:
F1=[6,5,6,7,4,2,1,5]

H=[0,1,1,0,1,2,2,1]
La fonction définie par la liste F1 n'est pas injective car H[5]=2

F2=[3,0,6,7,4,2,1,5]
H=[1,1,1,1,1,1,1,1]
La fonction définie par la liste F2 est injective acar aucun élément de H n'est supérieur à 1.

3) fonction **est_surjective(F)**

Cette fonction renvoie la valeur True si la fonction représentée par la liste F est une surjection. Dans le cas contraire, elle renvoie False.
On dit qu'une fonction f: I $\rightarrow$J est surjective si tout élément de J admet au moins un antécédent dans I. Pour savoir si f est surjective, il suffit ainsi de vérifier que tous les éléments de la liste H (obtenue par appel de la fonction histo) ont une valeur supérieure ou égale à 1.

Exemples:
F1=[6,5,6,7,4,2,1,5]
H=[0,1,1,0,1,2,2,1]
La fonction définie par la liste F1 n'est pas surjective car H[0]=0

F2=[3,0,6,7,4,2,1,5]
H=[1,1,1,1,1,1,1,1]
La fonction définie par la liste F2 est injective car aucun élément de H n'est inférieur à 1.

4) fonction **est_bijective(F)**

Cette fonction renvoie la valeur True si la fonction représentée par la liste F est une bijection. Dans le cas contraire, elle renvoie False.
On dit qu'une fonction f est bijective si elle est à la fois injective et surjective.
Exemples:
F1=[6,5,6,7,4,2,1,5]
H=[0,1,1,0,1,2,2,1]
La fonction définie par la liste F1 n'est pas bijective car elle n'est pas injective.

F2=[3,0,6,7,4,2,1,5]
H=[1,1,1,1,1,1,1,1]
La fonction définie par la liste F2 est bijective car elle est injective et surjective.

Question 2 : fonction affiche_histo Affichage de l'histogramme

Complétez votre bibliothèque de fonctions de la question 1 par la définition de la fonction afficheHisto(F) qui affiche une représentation graphique de l'histogramme associé à la liste d'entiers F.

Le format d'affichage est illustré par l'exemple d'exécution de la figure 1.

Remarques et indications:

- Le symbole # est utilisé pour représenter les occurrences de chaque valeur.
- Aprés avoir générer la liste H (histo), déterminez la valeur maximale MAXOCC de la liste H en appelant une des fonctions max définies dans l'exercice 1. Cette valeur MAXOCC vous donnera le nombre de lignes à afficher dans le graphique. Dans l'exemple de la figure 2, MAXOCC=3.
- En python, pour éviter le passage à la ligne lors de l'utilisation de la fonction print, utilisez le paramètre end en lui affectant la valeur " (ce paramètre a par défaut la valeur "retour chariot")
  - print(" Texte à afficher" , end =' ')

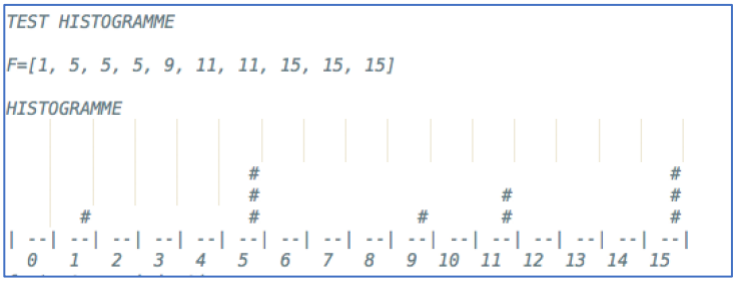

Figure 1 : Exemple d'exécution Test_Histogramme

Question 3 : Affichage de l'histogramme avec MatplotLib

La fonction hist(F) de librairie MatplotLib.pyplot permet trés simplement d'afficher graphiquement un histogramme (cf:résultat illustré en figure 2).
Elle admet en paramètre une liste.
Redéfinissez la fonction de la question précédente en utilisant la fonction hist.
NB: Pensez à importer la librairie en début de programme : import matplotlib.pyplot as plt

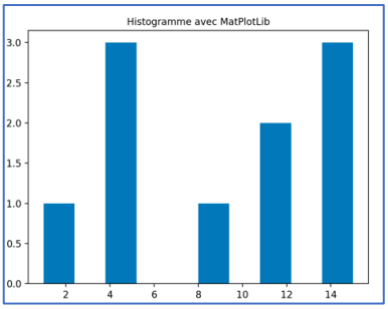

Figure 2 : Exemple d'exécution Test_Histogramme(version MatplotLib)

EXERCICE 5 - 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 6 - 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
```