Exercice 1 :

Quelle est la liste *L* retournée à la fin de ce programme ?

```
L=[]
for i in range(5):
    L.append(i)
L=L[:-1]
L=L+L[::-1]#L=L+L[len(L)-1::-1]   possible aussi
```

Réponse : [0, 1, 2, 3, 3, 2, 1, 0]

Exercice 2 :

Quelle est la liste *L* retournée à la fin de ce programme ?

```
L=[]
for i in range(5,-1,-2):
    L.append(i)
```

Réponse : [5, 3, 1]

Exercice 3 :

Ecrire une fonction *longueur* qui prend en paramètre une chaîne de caractère ou une liste et renvoie la longueur de la chaîne ou de la liste. Il est interdit d'utiliser la fonction *len*.

Réponse :

```
def longueur(sequence):
    compteur=0
    for i in sequence:
        compteur=compteur+1
    return compteur
```

Exercice 4 :

Ecrire une fonction *parité* qui prend en argument une liste d'entiers et renvoie un couple de deux listes : la première contient les nombres pairs et la seconde les nombres impairs.

```
def parité(L):
    Lp,Li=[],[]
    for i in L:
        if i%2==0:
            Lp.append(i)
        else :
            Li.append(i)
    return Lp,Li
```

Autre possibilité, par compréhension :

```
def parite2(L):
    Lp=[i for i in L if i%2==0]
    Li=[i for i in L if i%2!=0]
    return Lp,Li
```

Exercice 5 :

Ecrire une fonction *produit* qui prend en paramètres un entier naturel *n* et une liste de nombre *nombres* et renvoie une nouvelle liste obtenue en multipliant chaque élément de la liste *nombre* par *n*.

```
def produit(L,n):
    L_new=[]
    for i in range(len(L)):
        L_new.append(n*L[i])
    return L_new
def produit2(L,n):
    return [n*i for i in L]
```

Exercice 6 :

Ecrire une fonction *distance* qui prend en argument une liste de nombres appelée *nombres* et qui renvoie la somme de la valeur absolue des écarts de chaque élément de *nombres* avec la moyenne de *nombres* . En python, la fonction *abs* renvoie la valeur absolue du nombre donné en paramètre. La fonction *mean* n'est pas autorisée.

```
def distance(liste):
    somme=0
    for i in liste:
        somme=somme+i
    moyenne=somme/(len(liste))
    somme_ecart=0
    for i in liste:
        somme_ecart=somme_ecart+abs(i-moyenne)
    return somme_ecart#complexité linéaire
```

Exercice 7 :

On considère une liste *L* d'entiers.

1) Ecrire une fonction qui prend *L* en argument et un entier *elt* et qui renvoie *True* si *elt* est dans *L* et *False* dans le cas contraire (en utilisant une boucle *while*)
2) Ecrire une fonction qui prend *L* en argument et un entier *elt* et qui renvoie *True* si *elt* est dans *L* et *False* dans le cas contraire (en utilisant une boucle *for*)

```
"""avec une boucle while"""
def occurrence(L,elt):
    i=0
    while i<len(L):
        if L[i]==elt:
            return True
        else :
            i=i+1
    return False
"""avec une boucle for"""
def occurence2(L,elt):
    for i in L:
        if i==elt:
            return True
    return False
```

Ou plus simplement :

```
def occurence3(L,elt):
    return elt in L
```

Exercice 8 :

On considère une liste *L* d'entiers.

1) Ecrire une fonction qui prend *L* en argument et un entier *elt* et qui renvoie l'indice de l'élément recherché si *elt* est dans *L* et *False* dans le cas contraire (en utilisant une boucle *while*).
2) Ecrire une fonction qui prend *L* en argument et un entier *elt* et qui renvoie l'indice de l'élément recherché si *elt* est dans *L* et *False* dans le cas contraire (en utilisant une boucle *for*).

```
def ocurence3(L,elt):
    i=0
    while i<len(L):
        if L[i]==elt:
            return i
        else :
            i=i+1
    return False
"""avec une boucle for"""
def ocurence4(L,elt):
    for i in range(len(L)):
        if L[i]==elt:
            return i
    return False
```

Exercice 9 :

On considère une liste *L* d'entiers. Chercher l'indice de la dernière occurrence de l'élément *elt* dans L.

```
def occurence(L,elt):
    i=len(L)-1
    while i>=0:
        if L[i]==elt:
            return i
        else:
            i=i-1
    return False

L=[0,1,2,3,4,5,6,7,8,9,10]
print(occurence(L,-1))


def occurence2(L,elt):
    for i in range(len(L)-1,-1,-1):
        if L[i] == elt:
            return i
    return False

L=[0,1,2,3,4,5,6,7,8,9,10]
print(occurence2(L,-1))
```

Exercice 10 :

On souhaite trouver le maximum d'une liste *L*. Si la liste est non vide, on suppose alors que le maximum est le 1e élément de la liste, puis on parcourt *L* et chaque fois que l'on rencontre un élément plus grand que le maximum provisoire, on dit que c'est ne nouveau maximum provisoire.

Proposer une fonction prenant une liste en argument et permettant de décrire la situation ci-dessus. Le maximum et sa position seront renvoyés.

```
def maximum(L):
    indice=0
    maxi=L[0]
    for i in range(1,len(L)):
        if L[i]>maxi:
            maxi=L[i]
            indice=i
    return indice
```

Exercice 11 :

Ecrire une fonction *indice_maximum* qui prend en paramètre une liste de nombres et renvoie le maximum de ces nombres avec son indice dans la liste. Si plusieurs éléments sont égaux au maximum, la fonction renvoie l'indice le plus grand parmi les indices de ces éléments.

```
def indice_maximum(L):
    indice=0
    maxi=L[0]
    for i in range(1,len(L)):
        if L[i]>=maxi:
            maxi=L[i]
            indice=i
    return indice,maxi
```

Exercice 12 :

Ecrire une fonction *indice_maximum*2 qui prend en paramètre une liste de nombres non vide et renvoie le maximum de ces nombres avec la liste des indices de tous les éléments égaux au maximum.

```
def indice_maximum2(L):
    maxi=L[0]
    for i in range(1,len(L)):
        if L[i]>maxi:
            maxi=L[i]
    liste=[]
    for i in range(len(L)):
        if L[i]==maxi:
            liste.append(i)
    return liste
```

Exercice 13 :

On cherche maintenant à trouver les deux plus grands éléments d'une liste L de longueur $n > 2$ en appliquant la méthode suivante : les deux premiers éléments de L constituent le couple recherché, puis on parcourt la liste pour remplacer éventuellement l'un de ces deux éléments.

Proposer un programme permettant de retourner l'index de ces deux maximas ainsi que leur valeur respective.

```
def maximum2(L):
    max1,max2=L[0],L[1]
    index1,index2=0,1
    if max1>max2:
        max1,max2=max2,max1
        index1,index2=index2,index1
    for i in range(2,len(L)):
        if L[i]>max2:
            max1=max2
            index1=index2
            max2=L[i]
            index2=i
        elif L[i]>max1:
            max1=L[i]
            index1=i
    return index1,max1,index2,max2

def maximum3(L):
    max1,max2=L[0],L[1]
    index1,index2=0,1
    if max1>max2:
        max1,max2=max2,max1
        index1,index2=index2,index1
    for i in range(2,len(L)):
        if L[i]<max2:
            if L[i]>max1:
                max1=L[i]
                index1=i
        else :
            max2,max1=L[i],max2
            index2,index1=i,index2
    return index1,max1,index2,max2
```

Exercice 14 :

On donne la fonction mystère suivante :

```
def mystere(liste_1,liste_2):
    liste=[]
    i,j=0,0
    while i<len(liste_1) and j<len(liste_2):
        if liste_1[i]<liste_2[j]:
            liste.append(liste_1[i])
            i=i+1
        else :
            liste.append(liste_2[j])
            j=j+1
    return liste
```

On appelle cette fonction avec l'instruction *mystere*([2,4,6,8], [1,3,5,7,9])

Quel est le résultat envoyé ?

Cette fonction permet d'obtenir une liste triée à partir des éléments de deux listes elles-mêmes déjà triées