# Travaux pratiques - Mise au point et gestion des bugs - correction

**Exercice** 1 Reprendre le code de la fonction `mystere` de l'exercice 1 du TD, coder la fonction et ajouter des `print` dans le code affin d'aficher le tableau d'exécution pas à pas de la fonction.
Tester votre fonction avec le même exemple que dans l'exercice 1 et vérifier que l'affichage correspond au tableau du TD.

```
def mystere(tableau):
    indice_min = 0
    v_min = tableau[0]
    for i in range(1,len(tableau)):
        print(v_min, " | ",indice_min, " | ",i," | ",tableau[i]," | ",tableau[i] <
            v_min)
        if tableau[i] < v_min:
            v_min = tableau[i]
            indice_min = i
    return indice_min


r = mystere([-3, -7, 8, 6, -9, 4])
print(r)
```

**Exercice** 2 Reprendre le code de la fonction `max_et_indice` de l'exercice 4 du TD, coder la fonction et ajouter 4 tests unitaires (à l'aide du module `doctest`) dans le code.

```
import doctest
def max_et_indice(tab):

    '''
    >>> max_et_indice([1, 2, 3, 4, 5])
    (5, 4)

    >>> max_et_indice([-1, -6, 7, 8, -1, -7])
    (8, 3)

    >>> max_et_indice([-8])
    (-8, 0)

    >>> max_et_indice([-8,-7,-1, -7])
    (-1, 2)
    '''
    imax = 0
    vmax = tab[0]
    for i in range(len(tab)):
        if tab[i] > vmax:
            vmax = tab[i]
            imax = i
    return (vmax, imax)

doctest.testmod()
```

**Exercice** 3 Réaliser une fonction prenant en paramètre un tableau de nombre non vide et qui retourne la moyenne des valeurs du tableau.
Tester la fonction avec la tableau [5,9,2,3,6,10,13,45,7].
La fonction devra contenir un assert.

```
def moyenne(tab):
    assert tab != [],"  le tableau ne doit pas etre vide"
    s=0
    for n in tab:
        s+=n
    return s/len(tab)


print(moyenne([5,9,2,3,6,10,13,45,7]))
#OUTPUT
11.1111111111111111
```

**Exercice** 4 On rappelle que :

- le nombre $a^n$ est le nombre $a \times a \times \ldots \times a \times a$ où le facteur $a$ apparaît $n$ fois,
- en langage Python, l'instruction `t[-1]` permet d'accéder au dernier élément du tableau `t`.

Dans cet exercice, l'opérateur `**` et la fonction `pow` ne sont pas autorisés.

1. Programmer en langage Python une fonction `liste_puissances` qui prend en argument un nombre entier $a$, un entier strictement positif $n$ et qui renvoie la liste de ses puissances $[a^1, a^2, \ldots, a^n]$. La fonction devra contenir au moins 4 assert.
2. Programmer également une fonction `liste_puissances_borne` qui prend en argument un nombre entier $a$ supérieur ou égal à 2 et un entier `borne`, et qui renvoie la liste de ses puissances, à l'exclusion de $a^0$, strictement inférieures à borne. La fonction devra contenir au moins deux asserts.

Exemples :

```
def liste_puissances(a,n):
    t = [a]
    for i in range(1, n):
        t.append(a*t[i-1])
    return t

def liste_puissances_borne(a, borne):
    if a >= borne:
        return []
    t = [a]
    i = 1
    while a*t[i-1] < borne:
        t.append(a*t[i-1])
        i = i + 1
    return t

print(liste_puissances(3, 5))
#OUTPUT
[3, 9, 27, 81, 243]
print(liste_puissances(-2, 4))
#OUTPUT
[-2, 4, -8, 16]
print(liste_puissances_borne(2, 16))
#OUTPUT
[2, 4, 8]
print(liste_puissances_borne(2, 17))
#OUTPUT
[2, 4, 8, 16]
print(liste_puissances_borne(5, 5))
#OUTPUT
[]
```

**Exercice** 5 Programmer une fonction `converion_duree` qui prend en paramètre une durée exprimée en secondes et qui retourne cette durée exprimée en heures, minutes secondes
au format : hh :mm :ss.
Exemples :

```
def conversion_duree(duree):
    heure = duree // 3600
    minute = (duree % 3600) // 60
    seconde = (duree % 3600) % 60
    if heure < 10:
        str_h = "0"+str(heure)
    else:
        str_h = str(heure)

    if minute < 10:
        str_m = "0"+str(minute)
    else:
        str_m = str(minute)

    if seconde < 10:
        str_s = "0"+str(seconde)
    else:
        str_s = str(seconde)

    return str_h+":" + str_m + ":" + str_s

print(conversion_duree(3250))
#OUTPUT
00:54:10
print(conversion_duree(3361))
#OUTPUT
01:01:01
print(conversion_duree(0))
#OUTPUT
00:00:00
```

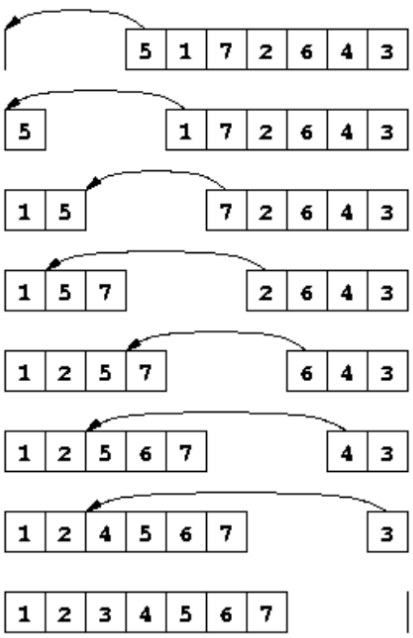
**Exercice** 6 Programmer une fonction `dec_to_bin` qui prend paramètre un nombre entier positif et qui retourne l'écriture binaire de nombre entier sous la forme d'une chaine de caractère.
Exemples :

```
def dec_to_bin(n):
    binaire = str(n %2)
    n = n //2
    while n  > 0:
        binaire = str(n % 2) + binaire
        n= n // 2

    return binaire

print(dec_to_bin(219))
#OUTPUT
11011011

print(dec_to_bin(0))
#OUTPUT
0

print(dec_to_bin(127))
#OUTPUT
1111111

print(dec_to_bin(26780))
#OUTPUT
110100010011100
```

**Exercice 7** Dans cet exercice, on considère un tableau T d'entiers que l'on veut trier par ordre croissant en utilisant l'algorithme de **tri par insertion**.

Voici une description de l'algorithme de tri par sélection :
On peut traduire l'algorithme de tri par insertion de la façon suivante :

- Prendre le deuxième élément du tableau et l'insérer à sa place parmi les éléments qui le précèdent
- Prendre le troisième élément du tableau et l'insérer à sa place parmi les éléments qui le précèdent
- Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié




**Tri par insertion**

```
pour i de 1 à n-1 faire
    x ← T[i]
    j ← i
    tant que j > 0 et x < T[j-1] faire
        T[j] ← T[j-1]
        j ← j - 1
    fin tant que
    T[j] ← x
fin pour
```

On applique cet algorithme au tableau T = [8, 3, 11, 7, 2].

1. Compléter le tableau suivant pour suivre l'état des variables.

| i | x | j | j>0 et x<T[j-1] | T[j] ← T[j-1] | j ← j-1 | T[j]← x | Etat du tableau T |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 1 | True | T[1]=8 | j=0 | X | [8, 8, 11, 7, 2] |
| 1 | 3 | 0 | False | X | X | T[0]=3 | [3, 8, 11, 7, 2] |
| 2 | 11 | 2 | False | X | X | T[2]=11 | [3, 8, 11, 7, 2] |
| 3 | 7 | 3 | True | T[3] = 11 | j=2 | X | [3, 8, 11, 11, 2] |
| 3 | 7 | 2 | True | T[2] = 8 | j=1 | X | [3, 8, 8, 11, 2] |
| 3 | 1 | 1 | False | X | X | T[1]=7 | [3, 7, 8, 11, 2] |
| 4 | 2 | 4 | True | T[4] = 11 | j=3 | X | [3, 7, 8, 11, 11] |
| 4 | 2 | 3 | True | T[3] = 8 | j=2 | X | [3, 7, 8, 8, 11] |
| 4 | 2 | 2 | True | T[2] = 7 | j=1 | X | [3, 7, 7, 8, 11] |
| 4 | 2 | 1 | True | T[1] = 3 | j=0 | X | [3, 3, 7, 8, 11] |
| 4 | 2 | 0 | False | X | X | T[0]=2 | [2, 3, 7, 8, 11] |

2. Programmer une fonction `tri_insertion`, prenant en paramètre un tableau `tab` de nombres non vide et qui retourne le tableau `tab` trié dans l'ordre croissant. Tester la fonction avec l'exemple de l'exerccie et créer 3 tests unitaires à l'aide du module `doctest`.

```
import doctest
# Programme Python pour l'implementation du tri par insertion
def tri_insertion(tab):
    """
    fonction de tri par insertion
    >>> tri_insertion([2, 6, 3])
    [2, 3, 6]
    >>> tri_insertion([2,1,1])
    [1, 1, 2]
    >>> tri_insertion([-1, -2, 3, -7, 8])
    [-7, -2, -1, 3, 8]
    """
   # Parcour de 1 a la taille du tab
    for i in range(1, len(tab)):
        k = tab[i]
        j = i
        while j > 0 and k < tab[j-1] :
                tab[j] = tab[j-1]
                j -= 1
        tab[j] = k
    return tab

# Programme principale pour tester le code ci-dessus

doctest.testmod()

tab = [8, 3, 11, 7, 2]
print(tri_insertion(tab))
```