Travaux pratique - Mise au point et gestion des bugs

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.

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.

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.

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 :

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

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 :

```
>>> converion_duree(3250)
00:54:10
>>> converion_duree(3661)
01:01:01
>>> converion_duree(0)
00:00:00
```

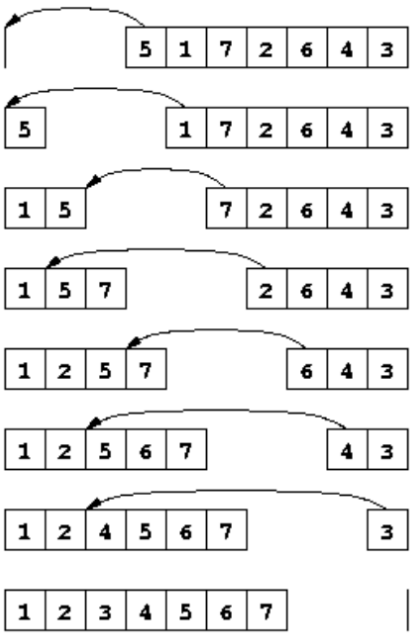
Exercice 6 Programmer une fonction `dec_to_bin` qui prend paramètre un nombre entier positif et qui retourne l'écriture binaire de nombre entier sous la forme d'une chaine de caractère.
Exemples :

```
>>> dec_to_bin(219)
11011011
>>> dec_to_bin(0)
0
>>> dec_to_bin(127)
1111111
>>> dec_to_bin(26780)
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]
...	...	...	...	...	...	...	...

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`.