Exercices - Mise au point et gestion des bugs

Exercice 1 Voici le code d'une fonction `mystere` et un exemple d'appel de cette fonction avec comme paramètre le tableau `[-3, -7, 8, 6, -9, 4]`.

```
def mystere(tableau):
    indice_min = 0
    v_min = tableau[0]
    for i in range(1,len(tableau)):
        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)
```

1. Expliquer en une phrase ce que fait cette fonction ?
2. Quelle valeur sera affichée après l'exécution de la fonction ?
3. Combien d'itération à l'intérieur de la boucle `for` seront effectuées ?
4. Completer le tableau suivant décrivant la valeur des variables et des tests conditionnels à chaque itération.

Itération	`v_min`	`indice_min`	`i`	`tableau[i]`	`tableau[i] < v_min`
Initialisation					
Itération 1	...	...	...	...	...

Exercice 2 Voici le code d'une fonction `mystere` et un exemple d'appel de cette fonction avec comme paramètre le tableau `[-3, -2, 1, 0, 2, 5]`.

```
def mystere(tab):
    i = 0
    while (i < (len(tab) - 1)) and (tab[i] < tab[i+1]):
        i = i+1
    return i == (len(tab)-1)

r = mystere([-3, -2, 1, 0, 2, 5])
print(r)
```

1. Expliquer en une phrase ce que fait cette fonction ?
2. Quelle valeur sera affichée après l'exécution de la fonction ?
3. Combien d'itération à l'intérieur de la boucle `while` seront effectuées ?
4. Completer le tableau suivant décrivant la valeur des variables et des tests conditionnels à chaque itération.

Itération	`i`	`tab[i]`	`tab[i+1]`	`i < (len(tab)-1)`	`tab[i] < tab[i+1]`
Initialisation					
Itération 1	...	...	...	...	...

Exercice 3 Voici le code d'une fonction `palindrome`. Cette fonction retourne `True` si le paramètre `mot` est un palindrome, `False` sinon.

```
def palindrome(mot):
    i = 0
    while (i< len(mot)//2) and (mot[i] == mot[len(mot)-1-i]):
        i = i+1
    return i == len(mot)//2
```

1. La fonction `palindrome` est appelée avec le mot "kayak".

```
r = palindrome("kayak")
print(r)
```

   (a) Quelle valeur sera affichée après l'exécution de la fonction ?

   (b) Completer le tableau suivant décrivant la valeur des variables et des tests conditionnels à chaque itération.

Itération	`i`	`mot[i]`	`mot[len(mot)-1-i]`	`i<len(mot)//2`	`mot[i]==mot[len(mot)-1-i]`
Initialisation					
Itération 1	...	...	...	...	...

2. La fonction `palindrome` est appelée avec le mot "palindrome".

```
r = palindrome("palindrome")
print(r)
```

   (a) Quelle valeur sera affichée après l'exécution de la fonction ?

   (b) Completer le tableau suivant décrivant la valeur des variables et des tests conditionnels à chaque itération.

Itération	`i`	`mot[i]`	`mot[len(mot)-1-i]`	`i<len(mot)//2`	`mot[i]==mot[len(mot)-1-i]`
Initialisation					
Itération 1	...	...	...	...	...

Exercice 4

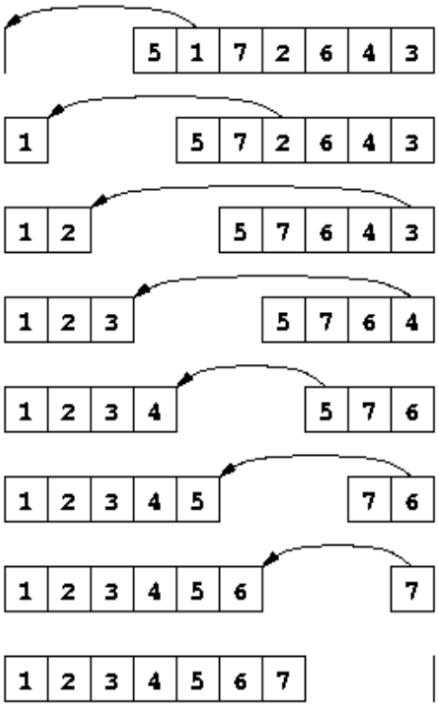
1. Écrire une fonction *max_et_indice* qui prend en paramètre un tableau non vide tab (type Python list) de nombres entiers et qui renvoie la valeur du plus grand élément de ce tableau ainsi que l'indice de sa première apparition dans ce tableau.
   L'utilisation de la fonction native max n'est pas autorisée.
2. Quel sera la valeur retournée par cet appel de fonction :
   max_et_indice([1, 5, 6, 9, 1, 2, 3, 7, 9, 8])
3. Ecrire un tableau d'exécution pas à pas de l'appel de fonction donné en exemple.

Exercice 5 Dans cet exercice, on considère un tableau T d'entiers que l'on veut trier par ordre croissant.
Les algorithmes permettant d'effectuer cette tâche s'appellent des algorithmes de tri et ont une place fondamentale en algorithmique et en informatique car il est souvent nécessaire des trier des élément.
Nous allons étudier l'algorithme de tri par sélection.
Spécification d'un algorithme de tri

1. Entrée/Sortie : tableau T (de taille n, constitué d'entiers, les indices variant de 0 à n-1)
2. Rôle : trier T par ordre croissant
3. Précondition : T non vide
4. Postcondition : T est trié c'est-à-dire que chaque élément de T est inférieur ou égal à l'élément suivant : pour tout entier i compris entre 0 et n-2, $T[i] \leq T[i+1]$.

Voici une description de l'algorithme de tri par sélection :
On peut résumer le principe de fonctionnement de l'algorithme de tri par sélection avec le schéma ci-contre :
On peut traduire l'algorithme de tri par sélection de la façon suivante :

- Rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- Rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.



Tri par sélection

```
pour i de 0 à n-2 faire
    ind_min ← i
    pour j de i+1 à n-1 faire
        si T[j] < T[ind_min] alors
            ind_min ← j
        fin si
    fin pour
    echange(T, i, ind_min)   # échange T[i] et T[ind_min]
fin pour
```

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

Compléter le tableau suivant pour suivre l'état des variables et du tableau au cours de l'algorithme.

i	ind_min	j	T[j]<T[ind_min]	ind_min ← j	Etat du tableau T
0	0	1	True	ind_min = 1	[8, 3, 11, 7, 2]
0	1	2	False	X	[8, 3, 11, 7, 2]
0	1	3	False	X	[8, 3, 11, 7, 2]
0	1	4	True	ind_min = 4	Echange i et ind_min → **[2, 3, 11, 7, 8]**
...	...	...	...	...	...

Exercice 6 Voici le code d'une fonction `mystere` et un exemple d'appel de cette fonction.

```
def mystere(n):
    r = 1
    e = 0
    while r <= n:
        e = e+1
        r = r*2
    return e-1

r = mystere(1000)
print(r)
```

1. Completer le tableau suivant décrivant la valeur des variables et des tests conditionnels à chaque itération.

Itération	n	r	e	r<=n
Initialisation				
Itération 1	...	...	...	...

2. Quel est le résultat affiché lors de l'appel de cette fonction ?
3. Expliquer en une phrase ce que fait cette fonction ?

Exercice 7 Deux asserts ont été ajoutés dans le code de la fonction `palindrome`.
Ces deux asserts ne sont pas corrects, réecrivez la fonction en corrigeant les asserts.

```
def palindrome(mot):
    assert len(mot) < 1
    assert isinstance(mot,int)

    i = 0
    while (i<= len(mot)//2) and (mot[i] == mot[len(mot)-1-i]):
        i = i+1
    return i == len(mot)//2 + 1
```

Exercice 8 Voici le code d'une fonction qui convertit un nombre entier positif a en binaire sur n bits.
Compléter la fonction en ajoutant 5 asserts.

```
def dec_vers_bin(a,n):
        '''
        Convertit le nombre entier positif a en binaire sur n bits
        '''

    binaire=""
    for i in range(n):
        binaire = str(a%2) + binaire
        a = a//2
    return binaire
```

Exercice 9 La fonction `mystere` de l'exercice 1 a été complétée avec 5 tests unitaires. Deux d'entre eux sont incorrects. Indiquer lesquels et corriger les.

```
import doctest
def mystere(tableau):
    ''' Fonction mystere
    Test Unitaire 1
    >>> mystere([0])
    0

    Test Unitaire 2
    >>> mystere([0, 1])
    0

    Test Unitaire 3
    >>> mystere([2, -3, -4, 1])
    1

    Test Unitaire 4
    <<< mystere([2, 4, 0, 2, 8])
    2

    Test Unitaire 5
    >>> mystere([2, 4, 0, 2, 8])
    2
    '''
    indice_min = 0
    v_min = tableau[0]
    for i in range(1,len(tableau)):
        if tableau[i] < v_min:
            v_min = tableau[i]
            indice_min = i
    return indice_min

doctest.testmod()
```

Exercice 10 Reprendre le code de la fonction `mystere` de l'exercice 2 et ajouter 3 tests unitaires à l'aide du module `doctest`.