Exercices pratiques

Exercice 1

Écrire une fonction `moyenne` qui prend en paramètre un tableau d'entiers non vide et qui renvoie un nombre flottant donnant la moyenne de ces entiers.

Attention : il est interdit d'utiliser la fonction `sum` ou la fonction `mean` (module `statistics`) de Python.

Exemples

```
>>> moyenne([1])
1.0
>>> moyenne([1, 2, 3, 4, 5, 6, 7])
4.0
>>. moyenne([1, 2])
1.5
```

Proposition de correction

```
def moyenne(tab):
    s = 0
    for v in tab:
        s = s + v
    return s / len(tab)
```

Exercice 2

Écrire la fonction `maximum_tableau`, prenant en paramètre un tableau non vide de nombres `tab` (de type `list`) et renvoyant le plus grand élément de ce tableau.

Exemples :

```
>>> maximum_tableau([98, 12, 104, 23, 131, 9])
131
>>> maximum_tableau([-27, 24, -3, 15])
24
```

Proposition de correction

```
def maximum_tableau(tab):
    vmax = tab[0]
    for v in tab:
        if v > vmax:
            vmax = v
    return vmax
```

Exercice 3

Programmer la fonction `recherche`, prenant en paramètres un tableau non vide `tab` (type `list`) d'entiers et un entier n, et qui renvoie l'indice de la **dernière** occurrence de l'élément cherché. Si l'élément n'est pas présent, la fonction renvoie None.

Exemples

```
>>> recherche([5, 3],1) # renvoie None
>>> recherche([2,4],2)
0
>>> recherche([2,3,5,2,4],2)
3
```

Proposition de correction

```
def recherche(tab, n):
    ind = None
    for i in range(len(tab)):
        if tab[i] == n:
            ind = i
    return ind
```

Exercice 4

On considère dans cet exercice une représentation binaire d'un entier non signé en tant que tableau de booléens.

Si

```
tab = [True, False, True, False, False, True, True]
```

est un tel tableau, alors l'entier qu'il représente est $2^6 + 2^4 + 2^1 + 2^0 = 83$. Cette représentation consistant à placer en premier le booléen indiquant la puissance la plus élevée de 2 est dite *big-endian* ou grand-boutiste.

Écrire une fonction `gb_vers_entier` qui prend en paramètre un tel tableau et renvoie l'entier qu'il représente.

Exemple :

```
>>> gb_vers_entier([])
0
>>> gb_vers_entier([True])
1
>>> gb_vers_entier([True, False, True,
        False, False, True, True])
83
>>> gb_vers_entier([True, False, False, False,
        False, False, True, False])
130
```

Proposition de correction

```
def gb_vers_entier(tab):
    s = 0
    n = len(tab)
    for i in range(n):
        v = tab[n - 1 - i]
        if v :
            s = s + 2**i
    return s
```

Exercice 5

Dans cet exercice, on cherche à calculer la moyenne pondérée d'un élève dans une matière donnée. Chaque note est associée à un coefficient qui la pondère.

Par exemple, si ses notes sont : 14 avec coefficient 3, 12 avec coefficient 1 et 16 avec coefficient 2, sa moyenne pondérée sera donnée par

$$\frac{14 \times 3 + 12 \times 1 + 16 \times 2}{3 + 1 + 2} = 14,333...$$

Écrire une fonction `moyenne` :

- qui prend en paramètre une liste `notes` non vide de tuples à deux éléments entiers de la forme `(note, coefficient)` (`int` ou `float`) positifs ou nuls ;
- et qui renvoie la moyenne pondérée des notes de la liste sous forme de flottant si la somme des coefficients est non nulle, `None` sinon.

Exemple :

```
>>> moyenne([(8, 2), (12, 0), (13.5, 1), (5, 0.5)])
9.142857142857142
>>> moyenne([(3, 0), (5, 0)])
None
```

Proposition de correction

```
def moyenne (tab):
    s_coef = 0
    s_note = 0
    for t in tab:
        s_coef = s_coef + t[1]
        s_note = s_note + t[0]*t[1]
    if s_coef == 0:
        return None
    else :
        return s_note / s_coef
```

Exercice 6

Écrire une fonction `min_et_max` qui prend en paramètre un tableau de nombres `tab` non vide, et qui renvoie la plus petite et la plus grande valeur du tableau sous la forme d'un dictionnaire à deux clés `min` et `max`.

Les tableaux seront représentés sous forme de liste Python.

L'utilisation des fonctions natives `min`, `max` et `sorted`, ainsi que la méthode `sort` n'est pas autorisée.

Exemples :

```
>>> min_et_max([0, 1, 4, 2, -2, 9, 3, 1, 7, 1])
{'min': -2, 'max': 9}
>>> min_et_max([0, 1, 2, 3])
{'min': 0, 'max': 3}
>>> min_et_max([3])
{'min': 3, 'max': 3}
>>> min_et_max([1, 3, 2, 1, 3])
{'min': 1, 'max': 3}
>>> min_et_max([-1, -1, -1, -1, -1])
{'min': -1, 'max': -1}
```

Proposition de correction

```
def min_et_max(tab):
    vmin = tab[0]
    vmax = tab[0]
    for v in tab:
        if v > vmax:
            vmax = v
        if v < vmin:
            vmin = v
    return {"min":vmin, "max":vmax}
```

Exercice 7

On souhaite programmer une fonction indiquant le point le plus proche d'un point de départ dans un tableau de points. Les points sont tous à coordonnées entières et sont donnés sous la forme d'un tuple de deux entiers. Le tableau des points à traiter est donc un tableau de tuples.

On rappelle que la distance d entre deux points du plan de coordonnées $(x; y)$ et $(x'; y')$ vérifie la formule :

$$d^2 = (x - x')^2 + (y - y')^2$$

Compléter le code des fonctions `distance_carre` et `point_le_plus_proche` fournies ci-dessous pour qu'elles répondent à leurs spécifications.

```
def distance_carre(point1, point2):
    """ Calcule et renvoie la distance au carre entre
    deux points."""
    return (...)**2 + (...)**2

def point_le_plus_proche(depart, tab):
    """ Renvoie les coordonnées du premier point du tableau tab se
    trouvant à la plus courte distance du point depart."""
    min_point = tab[0]
    min_dist = ...
    for i in range(1, len(tab)):
        if distance_carre(tab[i], depart) < ...:
            min_point = ...
            min_dist = ...
    return min_point
```

Exemples :

```
>>> distance_carre((1, 0), (5, 3))
25
>>> distance_carre((1, 0), (0, 1))
2
>>> point_le_plus_proche((0, 0), [(7, 9), (2, 5), (5, 2)])
(2, 5)
>>> point_le_plus_proche((5, 2), [(7, 9), (2, 5), (5, 2)])
(5, 2)
```

Proposition de correction

```
def distance_carre(point1, point2):
    """ Calcule et renvoie la distance au carre entre
    deux points."""
    return (point1[0] - point2[0])**2 + (point1[1] - point2[1])**2

def point_le_plus_proche(depart, tab):
    """ Renvoie les coordonnÃ©es du premier point du tableau tab se
    trouvant Ã  la plus courte distance du point depart."""
    min_point = tab[0]
    min_dist = distance_carre(tab[0], depart)
    for i in range(1, len(tab)):
        if distance_carre(tab[i], depart) < min_dist:
            min_point = tab[i]
            min_dist = distance_carre(tab[i], depart)
    return min_point
```

Exercice 8

On considère l'algorithme de tri de tableau suivant : à chaque étape, on parcourt le sous-tableau des éléments non rangés et on place le plus petit élément en première position de ce sous-tableau.

Exemple avec le tableau : `t = [41, 55, 21, 18, 12, 6, 25]`

- Étape 1 : on parcourt tous les éléments du tableau, on permute le plus petit élément avec le premier.

  Le tableau devient `t = [6, 55, 21, 18, 12, 41, 25]`

- Étape 2 : on parcourt tous les éléments **sauf le premier**, on permute le plus petit élément trouvé avec le second.

  Le tableau devient : `t = [6, 12, 21, 18, 55, 41, 25]`

Et ainsi de suite.

Le programme ci-dessous implémente cet algorithme.

```
def echange(tab, i, j):
    '''Echange les éléments d'indice i et j dans le tableau tab.'''
    temp = ...
    tab[i] = ...
    tab[j] = ...

def tri_selection(tab):
    '''Trie le tableau tab dans l'ordre croissant
    par la méthode du tri par sélection.'''
    N = len(tab)
    for k in range(...):
        imin = ...
        for i in range(..., N):
            if tab[i] < ...:
                imin = i
        echange(tab, ..., ...)
```

Compléter ce code de façon à obtenir :

```
>>> tab = [41, 55, 21, 18, 12, 6, 25]
>>> tri_selection(tab)
>>> tab
[6, 12, 18, 21, 25, 41, 55]
```

Proposition de correction

```
def echange(tab, i, j):
    '''Echange les Ã©lÃ©ments d'indice i et j dans le tableau tab.'''
    temp = tab[i]
    tab[i] = tab[j]
    tab[j] = temp

def tri_selection(tab):
    '''Trie le tableau tab dans l'ordre croissant
    par la mÃ©thode du tri par sÃ©lection.'''
    N = len(tab)
    for k in range(N-1):
        imin = k
        for i in range(k+1, N):
            if tab[i] < tab[imin]:
                imin = i
        echange(tab, k, imin)
```

Exercice 9

La fonction `tri_insertion` suivante prend en argument un tableau `tab` (type `list`) et trie ce tableau en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

On rappelle le principe du tri par insertion : on considère les éléments à trier un par un, le premier élément constituant, à lui tout seul, un tableau trié de longueur 1. On range ensuite le second élément pour constituer un tableau trié de longueur 2, puis on range le troisième élément pour avoir un tableau trié de longueur 3 et ainsi de suite…

A chaque étape, le premier élément du sous-tableau non trié est placé dans le sous-tableau des éléments déjà triés de sorte que ce sous-tableau demeure trié.

Le principe du tri par insertion est donc d'insérer à la n-ième itération, le n-ième élément à la bonne place.

```
def tri_insertion(tab):
    '''Trie le tableau tab par ordre croissant
    en appliquant l'algorithme de tri par insertion'''
    n = len(tab)
    for i in range(1, n):
        valeur_insertion = ...
        # la variable j sert à déterminer
        # où placer la valeur à ranger
        j = ...
        # tant qu'on n'a pas trouvé la place de l'élément à
        # insérer on décale les valeurs du tableau vers la droite
        while j > ... and valeur_insertion < tab[...]:
            tab[j] = tab[j-1]
            j = ...
        tab[j] = ...
```

Exemple :

```
>>> tab = [98, 12, 104, 23, 131, 9]
>>> tri_insertion(tab)
>>> tab
[9, 12, 23, 98, 104, 131]
```

Proposition de correction

```
def tri_insertion(tab):
    '''Trie le tableau tab par ordre croissant
    en appliquant l'algorithme de tri par insertion'''
    n = len(tab)
    for i in range(1, n):
        valeur_insertion = tab[i]
        # la variable j sert Ã  dÃ©terminer
        # oÃ¹ placer la valeur Ã  ranger
        j = i
        # tant qu'on n'a pas trouvÃ© la place de l'Ã©lÃ©ment Ã 
        # insÃ©rer on dÃ©cale les valeurs du tableau vers la droite
        while j > 0 and valeur_insertion < tab[j-1]:
            tab[j] = tab[j-1]
            j = j - 1
        tab[j] = valeur_insertion
```

Exercice 10

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.

Compléter l'algorithme de dichotomie donné ci-après.

```
def dichotomie(tab, x):
    """applique une recherche dichotomique pour déterminer
    si x est dans le tableau trié tab.
    La fonction renvoie True si tab contient x et False sinon"""

    debut = 0
    fin = ...
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
        if x > tab[m]:
            debut = ...
        else:
            fin = ...
    return False
```

Exemples :

```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 27)
False
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33], 1)
False
>>> dichotomie([], 28)
False
```

Proposition de correction

```
def dichotomie(tab, x):
    """applique une recherche dichotomique pour dÃ©terminer
    si x est dans le tableau triÃ© tab.
    La fonction renvoie True si tab contient x et False sinon"""

    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = (debut + fin) // 2
        if x == tab[m]:
            return True
        if x > tab[m]:
            debut = m + 1
        else:
            fin = m - 1
    return False
```