Révisions sur les tableaux

I. Quelques rappels

Si t est un tableau, alors sa longueur ℓ s'obtient à l'aide de la commande `len(t)`. Dans ce cas, les éléments du tableaux sont accessibles à l'aide d'indices allant de 0 inclus à $\ell - 1$ inclus (et ℓ exclu).

```
>>> t = [213, 45, 78, 319]
>>> len(t)
4
>>> t[2]
78
>>> t[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> t[3] = 12
>>> t
[213, 45, 78, 12]
>>> for i in range(len(t)):
...     print(t[i])
...
213
45
78
12
```

On peut ajouter un argument supplémentaire à `range` pour indiquer la valeur de départ.

```
>>> for i in range(5, 10):
...     print(i)
...
5
6
7
8
9
```

Pour information, on peut ajouter un troisième argument indiquant le pas de l'incrément, qui peut être négatif.

```
>>> for i in range(3, 0, -1):
...     print(i)
...
3
2
1
```

II. Parcours

Une part important de fonctions sur les tableaux repose sur l'idée de parcours séquentiel du tableau : on visite tous ses éléments dans l'ordre, de gauche à droite. On distinguera les cas où le parcours est *complet* (on parcourt le tableau jusqu'au bout) ou bien *partiel* (où l'on s'arrête avant la fin).

1. Parcours complet

Un premier type de parcours d'un tableau est de le parcourir en entier, et de visiter chacun de ses éléments dans l'ordre. Cela s'effectue avec une boucle **`for`**. La structure typique d'un tel parcours est le suivant en utilisant les indices :

```
def canevas_de_parcours_complet(tab):
    ... # initialisation
    for i in range(len(tab)):
        ... # on traite tab[i]
    ... # traitement final
```

En utilisant directement les valeurs, on peut écrire :

```
    def canevas_de_parcours_complet(tab):
        ... # initialisation
        for elt in tab:
            ... # on traite elt
        ... # traitement final
```

Exercice 1 Écrire une fonction `somme` qui calcule et retourne la somme des éléments du tableau d'entiers passé en argument.

```
>>> somme([1, 2, 4, 8, 16])
31
```

Exercice 2

1. Étant un tableau de chaînes de caractères, écrire une fonction `concat`. On rappelle que la concaténation de chaînes s'effectue à l'aide de l'opérateur « `+` ».

```
>>> concat(["il faut", "vous fier", "à vos intuitions"])
'il faut vous fier à vos intuitions'
```

2. Écrire une fonction `yoda` qui retourne la concaténation des éléments du tableau dans le sens droite-gauche, toujours séparés par un espace.

```
>>> yoda(["il faut", "vous fier", "à vos intuitions"])
'à vos intuitions vous fier il faut'
```

Vous pourrez essayer d'en écrire deux versions, une avec un parcours du tableau de droite à gauche, l'autre avec un parcours de gauche à droite.

Normalement, les fonctions précédentes ont la même structure : on utilise une variable dont la valeur est le résultat en cours de construction, que l'on appelle **accumulateur**. Les questions auxquelles il faut répondre sont alors :

- ⋆ Comment modifier l'accumulateur pour tenir compte de la valeur suivant dans le parcours du tableau ?
- ⋆ Comment initialiser l'accumulateur ?

Les exercices suivants se résolvent de façon similaire, en répondant à ces questions.

Exercice 3 Écrire une fonction `occurences` qui, étant donné un tableau t et un élément e, renvoie le nombre d'occurences de e dans t.

```
>>> occurences([5, 2, 4, 8, 7, 4, 1, 2, 5, 4], 4)
3
>>> occurences([5, 2, 4, 8, 7, 4, 1, 2, 5, 4], 6)
0
```

Exercice 4 Écrire une fonction `maximum` qui renvoie la plus grande valeur contenue dans le tableau supposé non vide passé en argument. **On n'utilisera pas la fonction `max`. À la place, on fera des comparaisons "à la main".**

Exercice 5 Écrire une fonction `position_maximum` qui renvoie la position du plus grand élément du tableau supposé non vide passé en argument. À nouveau, on n'utilisera pas la fonction `max`. Si le maximum apparaît plusieurs fois, on retournera le plus petit indice possible.

```
>>> position_maximum([5, 2, 4, 8, 7, 4, 1, 2, 5, 4])
3
>>> position_maximum([5, 7, 2, 6, 4, 7, 3, 1, 6])
1
```

Exercice 6 (un peu plus difficile, vous pouvez y revenir plus tard) Écrire une fonction `position_2_maximums` qui, étant donné un tableau d'entiers de taille au moins 2, renvoie un couple d'entiers distincts (p,q) indiquant les positions des deux plus grandes valeurs. Si la plus grande valeur apparaît plusieurs fois, on renverra les positions des première et seconde occurences. Sinon, q est la position de la première occurence de la seconde valeur par ordre décroissant.

```
>>> position_2_maximums([3, 5, 1, 2, 5, 4, 5])
(1, 4) # plusieurs valeurs maximales
>>> position_2_maximums([3, 2, 5, 2, 6, 4, 5])
(4, 2) # une seule valeur maximale
```

Vous essaierez, dans la mesure du possible, de programmer cette fonction avec une **unique** boucle **`for`**.

2. Parcours partiel

Il n'est parfois pas nécessaire de parcourir un tableau en entier pour avoir le résultat désiré. Dans ce cas, on peut avoir une condition d'arrêt anticipé qui indique si l'on peut interrompre le parcours. Pour en tenir compte, il est naturel d'utiliser une boucle **`while`**[1]

Pour cela, quelques précautions sont nécessaires.

- ⋆ Tout d'abord, il faut gêrer un indice de parcours « à la main » (ce qui apparaît dans la programme suivant avec les lignes `i = 0` et `i = i + 1`).
- ⋆ Il faut de plus s'assurer que si l'on veut, par exemple, utiliser la valeur `tab[i]`, il faut s'assurer qu'elle est bien définie. Pour cela, on fera débuter le test du **`while`** par une condition du type

```
i < len(tab) and ...
```

Ainsi, une fois ce test passé, on est sûr que `tab[i]` est bien défini.

```
def canevas_de_parcours_partiel(tab):
    ... # initialisation
    i = 0
    while i < len(tab) and ...: # condition d'arrêt anticipé
        ... # on fait quelque chose avec tab[i]
        i = i + 1
    ... # traitement final
```

On portera une attention particulière à la condition qui a déclenché la sortie de la boucle **`while`**. En effet, la condition d'arrêt anticipé P(`i`) peut dépendre de la

[1] Une autre approche est d'utiliser une boucle **`for`** avec un **`break`** ou un **`return`**.

valeur `tab[i]`, or cette dernière n'est pas nécessairement définie. On a donc *a priori* deux cas :

- ⋆ soit `i = len(tab)` et `tab[i]` n'est pas définie ;
- ⋆ soit `i < len(tab)`, `tab[i]` est définie et P(`i`) est fausse.

Exercice 7 Écrire une fonction `est_présent` qui, étant donné un tableau t et une valeur e, renvoie **`True`** si e apparaît dans t, et **`False`** sinon.

```
>>> est_présent([4, 1, 2, 5], 3)
False
>>> est_présent([4, 1, 2, 5], 1)
True
```

Exercice 8 Écrire une fonction `est_croissante` qui indique si le tableau passé en argument est croissant (au sens large).

```
>>> est_croissant([2, 4, 4, 5, 7])
True
>>> est_croissant([5, 7, 3, 4, 8])
False
```

Exercice 9 Écrire une fonction `est_modal` qui indique si le tableau passé en argument est *modal*, c'est-à-dire que le tableau peut être décomposé en deux phases : une phase croissante puis une phase décroissante (chaque phase pouvant être de longueur nulle). On a, bien sûr, le droit de faire plusieurs boucles **`while`** à la suite.

```
>>> est_modal([2, 3, 3, 4, 5, 5, 3, 3, 1])
True
>>> est_modal([1, 2, 3, 2, 3, 4])
False
```

3. Quelques exercices supplémentaires

Exercice 10 Écrire une fonction `écart_max` qui, tant donné un tableau d'entiers de longueur au moins 2, renvoie l'écart maximal (autrement dit la plus grand valeur absolue de la différence) entre deux éléments du tableau.

```
>>> écart_max([5, 7, 3, 4, 8])
5
```

Exercice 11 (un peu plus difficile, vous pouvez y revenir plus tard) On suppose maintenant que l'on a un tableau a double entrée t de taille $n \times n$ tel que $t[i][j]$ est égal à $|x_i - x_j|$ pour une suite de réels $\mathbf{x} = (x_k)_{k \in [\![0,n-1]\!]}$. On cherche à déterminer le diamètre de $\mathbf{x}$, autrement dit

$$\max_{i,j \in [\![0,n-1]\!]} |x_i - x_j|$$

Ainsi, pour $\mathbf{x} = (3, 1, -2, 6)$, on aura le tableau de tableaux

`[[0, 2, 5, 3], [2, 0, 3, 5], [5, 3, 0, 8], [3, 5, 8, 0]]`.

1. Écrire une fonction `diamètre` qui, étant donné un tableau `t` précédent, retourne le diamètre.
2. **encore plus difficile** Si votre algorithme est de complexité quadratique en n, essayez de le faire en complexité linéaire.

III. Constructions de tableaux

Jusqu'à présent, toutes les fonctions utilisaient un tableau passé en argument. Nous allons maintenant voir quelques exercices où l'on renvoie un tableau construit pour l'occasion.

Pour cela, nous disposons de deux méthodes principales :

1. si l'on connaît à l'avance la taille du tableau, on commence par le créer à la bonne taille avant de le remplir.

```
>>> [0] * 10 # création d'un tableau de 0 de taille 10
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> t = [0] * 10 # création puis...
>>> for i in range(10): # ...remplissage
...     t[i] = 2 ** i - 1
...
>>> t
[0, 1, 3, 7, 15, 31, 63, 127, 255, 511]
```

   Il est aussi possible de le déclarer et le remplir en même temps à l'aide d'une définition dite « par compréhension ».

```
>>> [3 * i for i in range(10)] # création et remplissage
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

2. si l'on ne connaît pas sa taille à l'avance, on peut ajouter les éléments un par un à l'aide d'un `append`.

```
>>> t = []
>>> n = 42
```

```
>>> while n > 0:
...     t.append(n % 2)
...     n = n // 2 # division entière
...
>>> t
[0, 1, 0, 1, 0, 1]
```

Exercice 12

1. Écrire une fonction `suite1` telle que `suite1(n)` renvoie les $n \geqslant 1$ premiers termes de la suite (u_n) définie par :

$$\forall\, n \in \mathbf{N},\ u_n = n^2 - n + 1.$$

2. Écrire une fonction `suite2` telle que `suite2(n)` renvoie les n premiers termes de la suite (u_n) définie par :

$$u_0 = 0 \qquad \text{et} \qquad \forall\, n \in \mathbf{N},\ u_{n+1} = \frac{u_n + 1}{u_n + 2}.$$

Exercice 13 – Génération des premiers nombres premiers Écrire une fonction `ératosthène` qui prends en entrée un entier n **strictement positif** et retourne la liste de tous les nombres premiers inférieurs ou égaux à n. Pour cela, on propose d'utiliser la méthode du *crible d'Érathostène* qui procède ainsi :

1. On crée un tableau `t` de $n+1$ booléens[a], tous initialisés à la valeur **`True`** ;
2. On affecte **`False`** à `t[0]` et `t[1]` puisqu'ils ne sont pas premiers ;
3. Pour tous les entiers k de 2 à n, si `t[`k`]` est égal à **`True`** alors k est un nombre premier. Dans ce cas, on l'ajoute à la liste des nombres premiers puis on mets à **`False`** tous ses multiples dans `t`.
4. À la fin, la liste des nombres premiers s'obtient en renvoyant les indices p tels que `t[`p`]` est égal à **`True`**.

```
>>> erathostène(25)
[2, 3, 5, 7, 11, 13, 17, 19, 23]
```

[a] La taille est choisie de façon à ce que `t[`n`]` soit bien défini.

Exercice 14 Écrire une fonction `dénivelés` qui, étant donné une liste de nombres représentant les altitudes successives pendant une randonnée, retourne la liste des hauteurs des dénivelés successifs de montées et de descentes.

```
>>> dénivelés([0, 2, 7, 13, 12, 7, 15, 22, 21, 25, 37])
[13, -6, 15, -1, 16]
```