Tableaux, suite

I. Création de tableaux

Pour l'instant, nous n'avons fait que parcourir un tableau. Nous allons maintenant voir comment créer et modifier les valeurs contenues dans un tableau.

Pour modifier une valeur présente dans un tableau, tout d'abord, c'est très simple, il suffit de faire une affectation. Par exemple :

```
>>> t = [1, 2, 3]
>>> t[1] = 10
>>> t
[1, 10, 3]
```

Pour la création d'un tableau, on a plusieurs possibilités.

Quelques créations de tableaux

Création « statique » lorsque l'on connaît exactement la taille et le contenu du tableau lors de sa création. C'est ce que nous avons fait dans l'exemple précédent.

Création en fonction de la taille Pour créer un tableau à la bonne taille, mais que l'on remplira plus tard, on peut utiliser la forme `[valeur] * taille` qui crée un tableau de la taille indiquée, et dont toutes les cases sont initialisées à `valeur`.

```
>>> def tableau_de_zeros(n):
...     return [0] * n
...
>>> tableau_de_zeros(10)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Création à l'aide d'une formule

```
>>> [2 * n + 1 for n in range(10)]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Remplissage au fur et à mesure Si l'on veut construire un tableau en le faisant `grandir` au fur et à mesure (si l'on ne connaît pas sa taille à l'avance, par exemple), on peut ajouter des éléments à droite du tableau à l'aide de la *méthode*[a] `append`.

```
>>> t = []
>>> t.append(1) # on ne renvoie rien, on modifie t
>>> t
[1]
>>> t.append(2)
>>> t
[1, 2]
>>> for i in range(3, 10):
...     t.append(i)
...
>>> t
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[a] C'est un peu comme une fonction, mais la syntaxe est différente

Exercice 1 Écrire une fonction `alternance` qui, étant donné un entier positif n, retourne une liste de taille n contenant une alternance de 1 et de -1.

```
>>> alternance(5)
[1, -1, 1, -1, 1]
```

Vous pourrez essayer d'en écrire plusieurs versions.

Exercice 2 Écrire une fonction `sommes_des_préfixes` telle qu'appelée avec comme argument un tableau t de longueur n, elle retourne le tableau u de même longueur tel que :

$$u_k = \sum_{i=0}^{k} t_i$$

On veut, par exemple :

```
>>> sommes_des_préfixes([4, -2, 1, 5, -3, -1])
[4, 2, 3, 8, 5, 4]
```

Exercice 3 À partir d'un entier strictement positive $a \in \mathbf{N}^\star$, on construit la suite (u_n) par :

$$u_0 = a, \qquad \forall\, n \in \mathbf{N},\ u_{n+1} = \begin{cases} \frac{1}{2} u_n & \text{si } u_n \text{ est pair ;} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

La conjecture de Syracuse postule que quelque soit la valeur initiale a, il y a nécessairement un terme de la suite égal à 1 (et les termes suivants sont alors 4, 2, 1, 4, 2,

1, 4, 2, 1, etc.)[1].

Il faut écrire une fonction `syracuse` qui, étant donné une valeur initiale a, renvoie les premiers termes de la suite correspondante, de la valeur initiale jusqu'au premier 1 rencontré. On rappelle que l'on peut tester si un nombre `n` est pair en vérifiant que son reste de la division euclidienne par 2, noté `n % 2` en Python, est nul.

On veut par exemple :

```
>>> syracuse(3)
[3, 10, 5, 16, 8, 4, 2, 1]
```

II. Chaînes de caractères

Une chaîne de caractères est le type de donnée utilisée pour représenter les données textuelles.

```
>>> texte = "Bonjour tout le monde"
```

De façon très schématique, on peut voir une chaîne de caractères comme un tableau de caractères et l'utiliser comme tel : elle a une longueur (déterminée à l'aide de `len`) et on peut accéder à ses éléments en fonction de leur position. Python utilise d'ailleurs la même syntaxe pour les deux :

```
>>> len(texte)
21
>>> texte[13]
'l'
```

Ainsi, les fonctions définies dans le TP précédent sur les parcours de tableaux peut s'appliquer aux chaînes de caractères. Notons de plus que l'on peut comparer des lettres, et même les ajouter... (mais l'effet n'est pas le même que pour les nombres).

Exercice 4

1. Dans les fonctions faites à la séance précédente, quels sont celles qui peuvent être utilisées avec des chaînes de caractères ?

2. Vérifiez vos suppositions.

Exercice 5

[1] C'est une conjecture, on ne sait pas le démontrer, mais ce n'est pas ce qui nous intéresse aujourd'hui.

1. Écrire une fonction `est_présent_à`, qui, étant donné deux chaînes de caractères, le `texte` et le `motif` et une position p, renvoie un booléen indiquant si le motif est présent dans le texte à la position p, autrement dit si :

$$\texttt{texte}[p] = \texttt{motif}[0], \texttt{texte}[p+1] = \texttt{motif}[1] \ldots$$
$$\ldots \text{ et } \texttt{texte}[p+\ell_m-1] = \texttt{motif}[\ell_m-1]$$

où ℓ_m désigne la longueur du motif. On supposera que p est tel que tous les accés aux tableaux sont bien définis.

```
>>> est_présent_à("abracadabra", "acada", 1)
False
>>> est_présent_à("abracadabra", "acada", 3)
True
```

2. Écrire une fonction `contient_motif` qui, étant donné un texte et un motif, renvoie `true` si le motif est présent dans le texte à une certain position p, et `false` sinon.

```
>>> contient_motif("abracadabra", "acada")
True
>>> contient_motif("abracadabra", "poulet")
False
```

III. Plusieurs boucles

Rien n'empêche d'utiliser plusieurs boucles, d'ailleurs dans l'exercice précédent de recherche de motifs, l'exécution repose sur deux boucles imbriquées (la première pour essayer les positions, la seconde pour rechercher le motif à une position donnée).

Exercice 6 – Suite de Catalan Écrire une fonction `catalan` qui renvoie le tableau contenant des $n+1$ premiers termes de la suite de Catalan $(C_n)_{n\in\mathbf{N}}$ (donc de C_0 à C_n inclus) définie par :

$$C_0 = 1 \qquad \forall\, n \in \mathbf{N}^\star,\ C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}.$$

Exercice 7

1. Écrire une fonction `distance_maximale` qui, étant donné un tableau de nombres t et une valeur v, renvoie la plus grande valeur absolue de la différence entre v et les éléments de t.

2. En déduire une fonction `diamètre` qui, étant donné un tableau non vide t de nombres, renvoie son diamètre, i.e. la distance maximale entre deux de ses éléments.

3. **(Facultatif)** Réécrire cette fonction sans faire appel à `distance_maximale` mais en utilisant deux boucles imbriquées. Vous essayerez de rendre votre fonction la plus efficace possible.

```
>>> distance_maximale([4, 7, 2, -4, 6], -1)
8
>>> diamètre([4, 7, 2, -4, 6])
11
```

IV. Exercices supplémentaires

1. Génération de nombres premiers

Exercice 8 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[2], 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]
```

2. Tri à bulle

Nous allons étudier un premier algorithme de tri, le tri à bulles, de tableaux et quelques variantes.

[2] La taille est telle que `t[`n`]` est bien défini.

2.1. Première version

Question 9 Écrire une fonction `echange(t, i)` qui compare `t[i]` et `t[i + 1]` et qui échange ces valeurs si `t[i]` > `t[i + 1]`. Elle ne retourne rien, ayant modifié le tableau.

```
>>> t = [4, 2, 1, 3, 6, 5]
>>> echange(t, 2)
>>> t # pas d'échange, 1 <= 3
[4, 2, 1, 3, 6, 5]
>>> echange(t, 1)
>>> t # échange, 2 > 1
[4, 1, 2, 3, 6, 5]
```

Question 10 Écrire une fonction `remontee(t)` qui effectue successivement tous les échanges sur `t` à toutes les positions allant de 0 à `len(t)` − 2 (inclus).

Question 11 Écrire une fonction `tri_a_bulles(t)` qui effectue `len(t)` remontées.

Il s'agit bien d'un algorithme de tri, puisque après avoir appelé `tri_a_bulle` sur un tableau, celui-ci a été trié. Ainsi, vous devez avoir quelque chose comme cela :

```
>>> t
[4, 2, 1, 3, 6, 5]
>>> tri_a_bulle(t)
>>> t
[1, 2, 3, 4, 5, 6]
```

Le nom « tri à bulles » vient du fait que lors d'une remontée, les plus grand éléments ont tendance à remonter dans le tableau (à la *surface*), telle une bulle d'air dans une coupe de liquide pétillant.

2.2. Analyse de l'algorithme

Question 12 – Correction Nous allons tout d'abord prouver que le tableau est nécessairement trié à l'issue de l'exécution de l'algorithme.

1. Montrer qu'à l'issue de la première remontée, le plus grand élément est à sa place dans le tableau.

2. Montrer qu'à l'issue de la seconde remontée, les deux plus grands éléments sont à leur place et, plus généralement, qu'à l'issue de k remontées, les k plus grands éléments sont bien placés.

3. En déduire qu'à l'issue de l'exécution de l'algorithme, le tableau est bien trié.

Nous allons maintenant essayer d'estimer le temps mis pour exécuter un tri à bulle en fonction de la taille du tableau, en comptant le nombre d'appels à la fonction `echange`.

Question 13 – Complexité Justifier que pour un tableau de taille n, l'algorithme de tri à bulles va effectuer $n^2 - n$ appels à la fonction `echange`.

On a ce que l'on appelle un algorithme quadratique : la durée d'exécution de l'algorithme varie (à peu près) comme le carré de la taille de l'entrée (ici la taille du tableau).

2.3. Quelques améliorations

Question 14 Que se passe-t-il si l'on effectue un tri à bulles sur un tableau déjà trié ?

Nous allons essayer de rendre l'algorithme plus efficace pour, par exemple, gérer correctement le cas des tableaux déjà triés (on parle alors de tri *adaptatif*).

Question 15 Modifier la fonction `echange` en une fonction `echange2` pour que, en plus de la modification du tableau, elle retourne un booléen indique si un échange a bien été effectué ou non.

En reprenant l'exemple précédent, on doit avoir maintenant :

```
>>> t = [4, 2, 1, 3, 6, 5]
>>> echange2(t, 2)
False
>>> t
[4, 2, 1, 3, 6, 5]
>>> echange2(t, 1)
True
>>> t
[4, 1, 2, 3, 6, 5]
```

Question 16 Modifier la fonction `remontee` en une fonction `remontee2` qui utilise la fonction `echange2` et qui renvoie `True` indiquant si, durant la remontée, un échange au moins a été effectué (autrement dit si un appel à `echange2` au moins a retourné `True`) et `False` sinon.

Question 17 Modifier la fonction `tri_a_bulles2` qui utilise la fonction `remontee2` et qui effectue des remontées tant que des échanges ont effectivement eu lieu.

Question 18 Justifier qu'avec `tri_a_bulles2`, si l'on essaye de trier un tableau de

taille n qui est déjà ordonné par ordre croissante, on effectue de l'ordre de n appels à `echange` et non de l'ordre de n^2.

Question 19 À l'issue d'un premier appel à `remontee`, on a vu que le dernier élément du tableau est à la bonne place. Ainsi, pour les remontées suivantes, il est inutile d'aller jusqu'en « haut ». Essayer de modifier `tri_a_bulles2` et les autres fonctions correspondantes pour tenir compte de cela.