Tableaux, algorithmes de base

L'une des manières les plus importantes d'utiliser un tableau ou, plus généralement, une structure *itérable* est d'en effectuer un **parcours**, qui consiste à visiter tous ses éléments une fois au plus. Nous allons étudier deux types de parcours :

- ⋆ les parcours **complets** où chaque élément est visité exactement une fois,
- ⋆ les parcours **partiels** où l'on peut s'arrêter avant d'avoir visité tous les éléments.

I. Parcours complet

L'idée usuelle est de construire une quantité qui dépend des différentes valeurs présentes dans un tableau, en visitant chaque élément une unique fois.

Exemple On peut calculer de cette façon la somme d'un tableau, le plus grand élément d'un tableau, etc.

Méthodologie

On va construire progressivement la quantité voulue au fur et à mesure du parcours. En cours de parcours, après avoir lu les k premières valeurs, la quantité correspond à ces k premières valeurs. À la fin, sa valeur est bien celle obtenue à partir de toutes les valeurs du tableau.

Le parcours s'effectue avec une **boucle `for`**. Il y a 2 problèmes principaux à résoudre :

- ⋆ **Incorporation d'une nouvelle valeur** Si on a lu les k premières valeurs du tableau `t` (d'indices allant de 0 à $k-1$ incluse) et que l'on veut incorporer la $k+1$-ème valeur (de position k), comment fait-on ?
- ⋆ **Initialisation** Comment initialise-t-on la quantité à calculer ? Parfois, l'initialisation se fait avant d'avoir lu la moindre valeur du tableau, parfois elle se fait en utilisant la première valeur[ab].

[a] Il faut alors que le tableau soit non vide.
[b] Voire plus.

Remarque En particulier, on ne modifie ni le contenu du tableau, ni sa structure : pas de `append`, de `pop`, de `del`, etc.

La structure d'un parcours est la suivante :

⋆ Version par valeurs :

```
... # initialisation
for elt in tab:
    # on traite elt
    ...
... # finalisation
```

⋆ Version par indice :

```
... # initialisation
for i in range(len(tab)):
    # on traite tab[i]
    ...
... # finalisation
```

Exercice 1 – Somme d'un tableau On veut calculer la somme des valeurs présentes dans un tableau.

1. **Nouvelle valeur** À partir de la somme des k premières valeurs du tableau et de la $k+1$-ème valeur, comment calcule-t-on la somme des $k+1$ premières valeurs ?
2. **Initialisation** Comment initialise-t-on notre somme partielle ? En particulier, peut-on faire l'initialisation sans avoir lu de valeur ? La somme est-elle définie pour un tableau vide ?
3. En déduire le code d'une fonction `somme` qui retourne la somme des valeurs présentes dans le tableau passé en argument.

Exercice 2 – Maximum d'un tableau On veut calculer le maximum des valeurs présentes dans un tableau non vide.

1. **Nouvelle valeur** À partir du maximum des k premières valeurs du tableau et de la $k+1$-ème valeur, comment calcule-t-on le maximum des $k+1$ premières valeurs ?
2. **Initialisation** Comment initialise-t-on notre maximum partielle ? En particulier, peut-on faire l'initialisation sans avoir lu de valeur ? Le maximum est-il défini pour un tableau vide ?
3. En déduire le code d'une fonction `somme` qui retourne la somme des valeurs présentes dans le tableau passé en argument.

Exercice 3 En utilisant la méthodologie précédente, écrire une fonction qui retourne la position du plus grand élément d'un tableau non vide. Si le maximum

apparaît plusieurs fois, on retournera le plus petit indice valide.

Exercice 4 En utilisant la méthodologie précédente, écrire une fonction qui calcule la moyenne des valeurs présentes dans un tableau non vide.

Exercice 5 Étant donné un tableau `t` et une valeur `v`, écrire une fonction qui calcule le nombre d'occurences de `v` dans `t`, autrement dit le nombre de fois que `v` est présent dans `t`.

Exercice 6 Étant donné un tableau `t` et une valeur `v`, écrire une fonction `partage` qui renvoie un couple composé de deux tableaux, le premier contenant les éléments de `t` qui sont inférieurs ou égaux à `v`, le second les éléments qui lui sont strictement supérieurs.

```
>>> partage([2, 4, 7, 11, 1, 16, 20, 7, 1, 3], 8)
[2, 4, 7, 1, 7, 1, 3], [11, 16, 20]
```

Exercice 7

1. Étant donné un tableau `t` non vide, écrire une fonction `records` renvoyant la liste des positions des *records*, un record étant une valeur du tableau strictement supérieure aux précédentes.

   ```
   >>> records([2, 4, 7, 6, 10, 15, 12, 7, 15, 3])
   [0, 1, 2, 4, 5]
   ```

2. Modifier la fonction pour prendre en compte les tableaux vides.

Exercice 8 Écrire une fonction `filtre` qui, étant donné un tableau `t` et une fonction `p` qui renvoit un booléen, retourne la liste des éléments du tableau pour lesquels `p` renvoie **`True`**.

```
>>> def est_pair(n):
...     return n % 2 == 0
...
>>> filtre([2, 4, 7, 6, 10, 15, 12, 7, 15, 3], est_pair)
[2, 4, 6, 10, 12]
```

II. Parcours partiel

Parfois, il n'est pas nécessaire de parcourir le tableau en entier pour déterminer la valeur qui nous intéresse.

Exemple On désire savoir si une valeur `v` est présente dans un tableau `t`. Pour cela, on parcourt le tableau `t`.

- ⋆ Quand peut-on être sûr que `v` **est** présente dans `t` ?
- ⋆ Quand peut-on être sûr que `v` **n'est pas** présente dans `t` ?
- ⋆ Comment écrire une fonction qui prends deux arguments `t` et `v` et renvoie **`True`** si `v` est présente dans `t`, et **`False`** sinon.

Méthodologie

Pour un parcours partiel, on ne connaît *a priori* pas à l'avance le nombre de valeurs du tableau à visiter. Pour cela, deux stratégies principales sont possibles :

1. utiliser une boucle **`for`** par indice ou valeur avec sortie anticipée (à l'aide d'un **`break`** ou d'un **`return`**) ;
2. utiliser une boucle **`while`** par indice avec une condition adaptée. Cette version est nécessaire si l'on a besoin de la valeur de l'indice en sortie de boucle.

Dans tous les cas, une question primordiale est de **déterminer quand on s'arrête**.

Pour la version avec **`while`**, le test incluera une partie qui assure que le tableau n'a pas été parcouru en entier.

Exercice 9 Écrire une fonction `sont_tous_pairs` qui prends en entrée un tableau `t` d'entiers (éventuellement vide) et qui renvoie **`True`** si toutes les valeurs présentes dans `t` sont paires, et **`False`** sinon.
Comme indiqué précédemment, on s'assurera de bien comprendre quand renvoyer tel ou tel résultat, et si l'on a besoin de parcourir le tableau en entier.

Méthodologie

La fonction précédente est **caractéristique** des parcours partiels : on a un prédicat P (c'est-à-dire une fonction qui renvoie un booléen) et on teste si

Exercice 10 Écrire la fonction `est_present` qui, étant donné un tableau `t` et une valeur `v`, retourne **`True`** si `v` est présent dans `t`, et **`False`** sinon.

Exercice 11 Écrire une fonction `est_croissant` qui indique si le tableau passé en argument est croissant.

Exercice 12 Écrire une fonction `deux_apres_un`, étant donné un tableau `t`, renvoie `True` si `t` contient la valeur `1` ainsi que la valeur `2` à un indice supérieur.

```
>>> deux_apres_un([1, 1, 5, 9, 4, 2, 8, 6, 8, 7])
True
>>> deux_apres_un([6, 2, 4, 1, 2, 8, 5, 7])
True
>>> deux_apres_un([5, 2, 7, 1, 3])
False
```