# Tableaux

*Oui : cette fiche est trop longue pour une séance : il faudra choisir ...*

## 1 Intro

Un moyen fréquent d'enregistrer de nombreuses données est le tableau. À la base, lorsqu'on veut enregistrer $n$ données en mémoire, on réserve $n$ emplacements consécutifs. Dès lors, pour tout $i \in [\![0, n[\![$, une formule du type « emplacement de la première case + $i\times$ (taille d'une case) » permet d'accéder à la case $i$.

Cependant, Python complique la chose, afin de permettre de rajouter des éléments à la fin d'un tableau. On obtient alors des « tableaux redimensionnables » que Python appelle des « `list` ».

Ainsi, le type `list` permet une grande souplesse, au prix d'un ralentissement des programmes [1].

On peut remarquer que du point de vue de l'enseignement de l'informatique Python est assez peu pédagogique sur ce point : les élèves auront du mal à comprendre ce qu'est un vrai tableau et une vraie liste...

Au niveau du vocabulaire c'est assez peu pratique : il est plus juste de parler de tableaux, mais comme Python les appelle des `list`, beaucoup ont l'habitude d'utiliser le mot « liste ».

Dans les programmes officiels, pour ajouter à la confusion le programme de maths utilise le mot « liste », et le programme de NSI utilise « tableau ».

*Remarque :* Informatiquement, on dirait que le type `list` de Python est un mélange du type « tableau » et du type « pile ».

## 2 Exercices simples

1. Somme des éléments d'un tableau.
2. Maximum des éléments d'un tableau.
3. Appartenance d'un élément à un tableau. Plusieurs manières de faire... A débattre !
4. Garder les éléments positifs d'un tableau. Utilisation du `append`.
5. En lien avec la notion de suite :
   (a) Tester si les éléments d'un tableau forment une suite croissante.
   (b) Étant donnés une suite $u$ définie par son premier terme et une relation de récurrence, ainsi qu'un entier naturel $n$, écrire une fonction qui crée un tableau contenant les $n$ premiers termes de $u$.
   Pour commencer, prendre par exemple la suite $u$ telle que $u_0 = 2$ et $\forall i \in \mathbb{N}$, $u_{i+1} = \frac{u_i}{2} + \frac{1}{u_i}$.
   *Remarques :*
   - Python n'a pas de commande simple pour initialiser directement un tableau de $n$ cases... Par contre, il existe une syntaxe plus générale permettant de décrire un tableau en compréhension comme en maths. Je montre juste un exemple : `t = [ i**2 for i in ramnge(0, 10) ]`.
   - Pour les élèves, créer le tableau des valeurs successives de $u$ est souvent plus simple que de calculer uniquement le $n$-ième terme de $u$ !
   - Si le résultat d'un appel à cette fonction est rangé dans le tableau `t`, alors on trace facilement la suite par les commandes suivantes :

```
import matplotlib.pyplot as plt
plt.scatter( range(len(t)), t) # création du nuage de points
plt.show() # affichage du graphe

```

6. Plus difficile : supprimer les éléments négatifs d'un tableau, directement dans le tableau de départ (on dit « en place »). Pour supprimer le $i$-ème élément d'un tableau `t`, la commande est `t.pop(i)`. Je le montre essentiellement pour dire de ne pas le faire ! À mon avis, la commande `pop(i)` devrait être soigneusement cachée au élèves, comme toute commande qui prend quelques caractères à être écrite, mais qui nécessite un temps proportionel à la taille du tableau à être exécutée. Le but n'est pas que les élèves apprennent un maximum de commandes Python mais qu'ils comprennent comment on peut tout programmer à l'aide d'un petit nombre de commandes de base.

1. On dit qu'un programme Python est environ qutre fois plus lent qu'un programme C.

# 3 Spécificité du type tableau

Le type `list` a une grosse spécificité par rapport aux type de base des entiers, flottants, chaînes de caractères, booléens, ou complexes : il est modifiable (« mutable » en anglais). Cela signifie que certaines commandes permettent de modifier le contenu d'un tableau directement dans la mémoire. En l'occurrence, ce sont principalement les commandes `t[i]=...`, `t.append(...)` et `t.pop()` [2].

Par exemple, si `x` est une variable entière, lorsque je tape `x=x+1` je crée en fait une nouvelle variable, qui s'appelle encore `x`, et qui contient un de plus que le contenu du précédent `x`.

Lorsque `t` est un tableau et que je tape `t[0]=3`, je modifie directement le contenu de la première case de `t`.

Faire le QCM plickers pour voir les conséquences concrètes de cette différence.

Lorsqu'on manipule un tableau, on arrive naturellement à écrire des programmes qui ne renvoient rien (pas de `return`) mais qui *modifient* le tableau passé en argument. Un tel programme s'appelle une « procédure ».

Ce type de programme est impossible pour les types de base des nombres, booléens, ou chaînes de caractères, car ces types ne sont pas modifiables.

# 4 Application : crible Ératosthène

Pour la culture, je résume ici les trois premières lignes de la page Wikipédia sur Érathostène : c'était le directeur de la bibliothèque d'Alexandrie et précepteur du Pharaon Ptolémée IV. Connu pour son calcul du rayon de la Terre. La légende raconte qu'à la fin de sa vie, devenu aveugle, il se laissa mourir de faim car il ne pouvait plus voir les étoiles.

L'exercice du crible d'Ératosthène est un classique, et pas pour rien : il y a beaucoup de remarques et de commentaires intéressants à faire sur sa programmation.

Je propose les étapes suivantes. À chaque fois, vaut-il mieux utiliser une fonction ou une procédure ?

1. Initialisation : écrire un programme qui prend en entrée un entier $n$ et qui renvoie la tableau `[0,...,n-1]`.
2. Écrire un programme `barre` qui prend en entrée un tableau `t` et un entier `a` et qui « barre », c'est-à-dire met 0, dans les cases d'indice multiple de $a$ mais différent de $a$.
3. Écrire un programme `sansZéro` qui prend un tableau `t` et renvoie le tableau des éléments non nuls de `t`.
4. Pourquoi ne pas écrire un programme `enlèveZéros` qui se chargerait de supprimer les 0 du tableau ?
5. Écrire un programme final, qui prendra en entrée un entier $n$ et renvoie le tableau de tous les nombres premiers de $[\![0, n[\![$.

En général, on n'obtient pas du premier coup une version optimisée de `barre` ni de la fonction finale... Demandez-moi pour savoir si [3] on peut faire plus efficace !

2. Comme dit plus haut, je suis partisan de cacher aux élèves les fonctions plus évoluées.
3. *Seulement* si en vrai ;)

# 5 Pour ceux qui s'ennuient

Un prédicat est une fonction qui renvoie un booléen, c'est-à-dire `True` ou `False`. Par exemple :

```
def estPositif(x):
    if x>=0:
        return True
    else:
        return False
```

est un prédicat qui indique si un nombre est positif.
*Remarque :* On peut l'écrire de manière plus concise ainsi :

```
def estPositif(x):
    return x>=0
```

Programmer une fonction `pourTout` prenant en entrée un tableau et un prédicat et indiquant si tous les éléments du tableau vérifient le prédicat. Par exemple `pourTout( [0,4,5,2,3], estPositif)` renverra `True`, alors que `pourTout( [0,4,5,-2,3], estPositif)` renverra `False`.

Programmer ensuite sur le même principe des fonctions `existe` et `existeUnique`.

Plus astucieux, pour travailler la logique (les « et » et les « ou ») : reprogrammer ces fonctions sans utiliser de `if`.