Chapitre 6

Les tableaux informatiques

Pour manipuler une valeur, on la stocke dans une variable. Si on doit manipuler 100 valeurs, trouver 100 noms de variables va être très factidieux : on a besoin de manipuler une collection de valeur : cette structure de données s'appelle un **tableau** (ou *array* en anglais)

1 Tableaux python

Il existe deux types de tableaux :
- des tableaux non modifiables, de type **tuple**, définis avec **()**
- des tableaux modifiables (*mutables*), de type **list**, définis avec **[]**

Un tableau de type *tuple* est défini une fois pour toutes au moment de l'affectation : on ne peut plus le modifier. Si on veut le refaire, il faut le supprimer, puis recommencer.
Vous avez déjà rencontré des tuples avec les fonctions en python, car une fonction python qui envoie au moins 2 éléments, renvoie en réalité un tuple composé des éléments à renvoyer (cf cours sur les fonctions).

On travaille principalement avec des tableaux modifiables de type *list*.

```
T = [] # tableau vide
T = [0]*3 # [0,0,0] valeurs séparées par des virgules
[1,2] + [3,4,5] # [1,2,3,4,5]
[3,4,5] + [1,2] # [3,4,5,1,2]
T = [1, 'a', 'chaine', 10.02, True] # on met ce qu'on veut dans un tableau python
```

2 Index d'un élément dans un tableau

Pour pouvoir une valeur stockée dans un tableau, il faut pouvoir accéder à cette valeur : c'est le rôle de l'index.
`T = [1,2,3]`
`T[0]` correspond à 1 (on utilise ici aussi la notation des crochets, mais dans un contexte différent)
`T[1]` correspond à 2
`T[2]` correspond à 3
`T[3]` renvoie une `Index Error`

Nombre d'éléments contenus dans un tableau : `len(T)`
Toutes les règles vues sur les index dans les chaînes de caractère sont valables sur les tableaux : notamment `T[::-1]` renvoie le tableau à l'envers !

On peut modifier un élément dans un tableau de type *list* : si on écrit `T[0] = 4`, alors le tableau `T` est devenu le tableau `[4,2,3]`.

3 Ajout ou retrait d'un élément dans un tableau

On a un tableau `T` et on voudrait lui ajouter une nouvelle valeur (le contenu de la variable `x`) : on a deux manières de faire :

- `T = T + [x]` ou plus simplement `T += [x]`
  cette manière de faire recopie à chaque fois le tableau, pas très efficace sur de grands tableaux, prend de plus en plus de temps dès que le tableau grandit
- `T.append(x)`
  peu importe la taille du tableau `T`, l'ajout se fait en temps constant

Pour effacer / enlever la dernière valeur du tableau `T` : `[1,2,3]` → `[1,2]`

- `del(T[len(T)-1])`
- `x = T.pop()` (x contient la valeur qui est supprimée, ici 3)

4 Parcours d'un tableau

Si on veut utiliser les valeurs stockées dans un tableau, il faut le **parcourir**.
`T = [1, 'A', 10.02, True]`
Il y a deux manières de parcourir un tableau en python :

```
for x in T :
    print(x)
```

On manipule ici directement les valeurs, mais on n'a pas accès à leur index. On ne peut pas modifier les éléments du tableau.

```
for i in range(len(T)) :
    print(T[i])
```

On manipule les valeurs via leur index, on peut modifier le tableau si besoin !

A l'index `i` correspond la valeur `x = T[i]`.
Pour mettre en oeuvre ces deux manières de faire, on va faire deux exercices.

Exercice 15 (Moyenne)

Ecrire une fonction `moyenne(T)` qui renvoit la moyenne des notes contenues dans `T`

```
1 def moyenne(T) :
2     somme = 0
3     for x in T :
4         somme += x
5     return somme / len(T)
```

Exercice 16

Ecrire une fonction `remplace` qui prend en argument un tableau `T` d'entiers et qui modifie ce tableau pour que les nombres pairs soient remplacés par `'x'`.

```
1 def remplace(T) :
2     for i in range(len(T)) :
3         if T[i]%2 == 0 :
4             T[i] = 'x'
```

Ici, on ne renvoie rien (donc python renvoie quand même `None`), on modifie le tableau `T` en place.

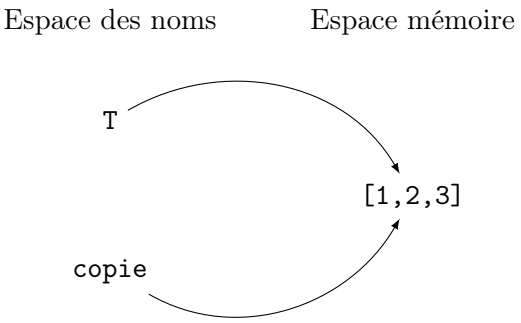
5 Tableaux par compréhension

`T1 = [i for i in range(10) if i%2==0]` : tous les entiers pairs compris entre 0 inclu et 10 exclu
`T2 = [2*x**3-4 for x in T1]` où `T1` est le tableau précédent
`T3 = [randint(1,12) for x in range(4)]` : tire aléatoirement 4 fois un entier dans l'intervalle [1,12].

6 Copie d'un tableau

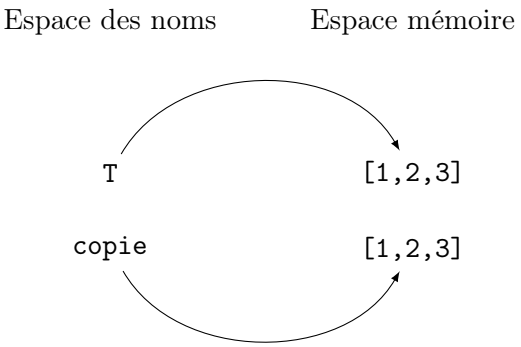
Si on ne veut pas modifier un tableau, mais travailler sur une copie, on ne peut pas faire simplement `copie = T`.

```
>>> T = [1,2,3]
>>> copie = T
>>> copie
[1, 2, 3]
>>> copie[0]=4
>>> copie
[4, 2, 3]
>>> T
[4, 2, 3]
```



On peut faire une **copie légère** du tableau `T` avec `copie = T[:]` comme par exemple :

```
>>> T = [1,2,3]
>>> copie = T[:]
>>> copie[0]=4
>>> copie
[4, 2, 3]
>>> T
[1, 2, 3]
```



Maintenant, chaque variable pointe vers une zone spécifique et différente de la mémoire, modifier l'une ne change rien pour l'autre.

Un autre moyen de contourner le problème est de réaliser une *copie profonde* :

```
from copy import deepcopy
copie = deepcopy(T) # copie est une version indépendante de T
```

7 Recherche d'un maximum dans un tableau d'entier

On ne considère dans cette partie que des tableaux d'entiers (naturels ou relatifs).

Exercice 17 (Maximum d'un tableau)

Ecrire une fonction `maximum` qui prend en argument un tableau `T` d'entiers et qui renvoie la valeur maximale de `T`.

```
1 def maximum(T) :
2     maxi = T[0]
3     for x in T :
4         if x > maxi :
```

```
            maxi = x
    return maxi
```

Exercice 18 (Index du maximum d'un tableau)

Ecrire une fonction `maximum` qui prend en argument un tableau `T` d'entiers et qui renvoie la valeur maximale de `T`.

```
def maximum(T) :
    maxi, pos = T[0], 0
    for i in range(len(T)) :
        if T[i] > maxi :
            maxi = x
            pos = i
    return pos
```

```
def maximum(T) : # proposition de Nassime en 2022 sans mini
    pos = 0 # indice du minimum
    for i in range(len(T)) :
        if T[i] > T[pos] :
            pos = i
    return pos
```

Exercice 19 (Index du maximum d'un sous-tableau)

Ecrire une fonction `maximum` qui prend en argument un tableau `T` d'entiers et un entier `j`. La fonction renvoie la valeur maximale du sous-tableau `T[j..n]`.

```
def maximum(T) :
    maxi = T[j] # et non T[0]
    for i in range(j+1, len(T)) :
        if T[i] > maxi :
            maxi = x
    return maxi
```

Exercice 20 (Recherche du second maximum d'un tableau)

Ecrire une fonction `deux_maxima` qui prend en argument un tableau `T` d'entiers. La fonction renvoie les deux valeurs maximales du tableau, la plus grande en premier.

```
def maximum(T) :
    "max1 est la plus grande valeur"
    max1, max2 = T[0], T[1]
    if max2 > max 1 :
        max1, max2 = max2, max1
    for x in T :
        if x > max1 :
            max2 = max1
            max1 = x
        elif x > max2 :
            max2 = x
    return max1, max2
```