Dans ce TP, nous continuerons à implémenter les boucles `TANT QUE` et découvrirons les fonctions.

I Notions de tableaux

Lors du TP 1, nous avions vu 3 types de variables : .....

Les tableaux sont un quatrième type de variable (déjà rencontré). En effet, une chaîne de caractères est en réalité un tableau de caractères que l'on ne peut modifier.

⚠ sous `python`, les tableaux ont des indices débutant à 0. Ainsi le dernier indice est égal à la taille du tableau moins 1.

e.g. Tab="Il a 16 euro."

indice	0	1	2	3	4	5	6	7	8	9	10	11	12
valeur	I	l		a		1	6		e	u	r	o	.

`Tab[0]` renvoie 'I' ; `Tab[11]` renvoie 'o'

Que renvoie `Tab[2]`, `Tab[9]` ?

Sous `python`, tableaux et listes sont quasi-identiques. Par la suite, tableau et liste sont interchangeables.

- ◇ on définit un tableau à l'aide de crochets, un tableau composé de dix 0 avec `Tab=10*[0]`,
- ◇ on copie un tableau avec `newTab = Tab.copy()`,
- ◇ on obtient la taille d'un tableau avec la fonction `len()`,
- ◇ on parcourt un tableau à l'aide d'une boucle `POUR` ou `TANT QUE`.

expl : implémenter sous `python` bloc par bloc ces lignes-ci.

```
Tab = [10, -12, "plein", 2, "data"]
print(len(Tab), Tab[1], Tab[2])
# avec une boucle POUR
for element in Tab:
    print(element)
# avec une boucle TANT QUE
i=0
while i<len(Tab):
    print(Tab[i])
    i+=1
```

Exo 1 : Définir la chaîne de caractères correspondant à la phrase de Norbert Zongo (journaliste burkinabè assassiné en 1 998) : "Le pire n'est pas la méchanceté des gens mauvais mais le silence des gens bien".

a) afficher 1 à 1 tous les caractères composants cette chaîne, en `python` de trois manières différentes.

b) afficher un caractère sur trois, en `python` de deux manières différentes.

II Notions de fonctions

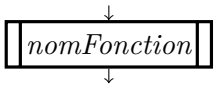
Dans un programme long ou complexe, afin de **clarifier la structure** du programme et donc les étapes de résolution, nous pouvons être amenés à découper le problème en plusieurs sous-problèmes. Chacun des sous-problèmes pourra être alors gérer par une procédure (/un programme) qui lui sera propre. Si le programme est long, il est possible qu'un jeu d'opérations doive être effectué de nombreuses fois. On pourra alors vouloir **mutualiser le code** qui traite ce jeu d'opérations afin de ne pas devoir recopier les mêmes lignes.

Cela permet d'avoir un code plus ....., plus ..... et plus .....

sous `python`

```
def nomFonction(variables):
    bloc d instructions
    return valeur
```

dans un organigramme



expl : si on souhaite calculer l'image de 3 par la fonction f définie par $f(x) = 3x^2 + 1$. On pourra définir la fonction f ainsi :

```
def f(x):
    return 3*x**2+1
```

Exo 2 : Proposer une fonction `zerof` qui étant donné 3 réels a, b et c définissant la fonction polynomiale du 2^n degré $x \mapsto ax^2 + bx + c$ renvoie un booléen vrai s'il existe des racines, faux si elles n'existent pas.

expl : on peut parfois mettre des tableaux comme variables d'une fonction. Par exemple si on souhaite calculer les coordonnées du milieu d'un segment [AB] (où $A(0;1)$ et $B(3;5)$), on peut écrire :

```
def coordonneesMilieu(A,B):
    xC=(A[0]+B[0])/2
    yC=(A[1]+B[1])/2
    return [xC,yC]

coordonneesMilieu([0,1],[3,5])
```

Exo 3 : On considère le code python suivant

```
def mystere(m, T):
    i=0
    while i<len(T):
        if m==T[i]:
            return True
        i=i+1
    return False
```

Dans un tableau, faites fonctionner ce programme pour
- ◇ m="c" et T="abc"
- ◇ m="c" et T="xyz" .

a) Que vous semble faire ce programme ?

b) Implémenter ce programme et confirmer votre intuition.

c) Comparer avec la commande `m in T`.

Travail à faire pour la prochaine séance

Exo 4 : Proposer une fonction `tableMulti` qui prend en argument N et qui affiche la table de multiplication de N de 1 à 10 (*cf.* TP4).

Exo 5 : On reprend le jeu du "juste prix" (deviner un nombre choisi aléatoirement, *cf.* TP4). Proposer une fonction `jeuDevine` qui prend en argument un entier N et qui renvoie le nombre d'essai nécessaire à l'utilisateur pour deviner le nombre.

Exo 6 : Proposer une fonction

a) `maximum` qui prend en argument deux réels a et b et renvoie le plus grand.

b) `maxTab` qui prend en argument un tableau de réels et renvoie le plus grand élément.

Exo 7 : Pour les exemples $Tab = [0, 4, 7, 4, -2, 5]$. Proposer une fonction

a) `opposeTab(T)` qui prend un tableau de nombres en paramètre et renvoie un tableau composé des opposés.

b) `absolueTab(T)` qui prend un tableau de nombres en paramètre et renvoie un tableau composé des valeurs absolues.

par exemple, `opposeTab(Tab)` renverra $[0, -4, -7, -4, 2, -5]$ et `absolueTab(Tab)` renverra $[0, 4, 7, 4, 2, 5]$

Exo 8 : Proposer une fonction

a) `differenceTabs` qui prend en argument deux tableaux (`T1` et `T2`) de même taille et renvoie le tableau des différences cases à cases (*e.g.* `T1[0]-T2[0]`).

b) `diffCasesTab` qui prend en argument un tableau de taille n et renvoie un tableau de taille n-1 de la différence entre deux cases contiguës (*e.g.* `T[1]-T[0]`).

Exo 9 : On souhaite faire un puissance quatre simplifié entre deux joueurs 1 et -1. La grille G sera un tableau de taille dix initialiser à 0 au départ. On définit les fonctions suivantes :

- ◇ `demandePosition(j,G)` qui affiche la grille G puis demande une position au joueur j tant que celui-ci en fournie une déjà occupée. Renvoie la position libre x choisie par le joueur j.
- ◇ `insertion(j,x,G)` qui renvoie la nouvelle grille G avec le jeton 'j' nouvellement inséré à la colonne x.
- ◇ `gagne(j,G)` qui renvoie vrai si 4 cases contiguës de G possède la valeur j, faux sinon.

expl avec $G =$ | −1 | −1 | −1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

a) `insertion(1,3,G)` renvoie | −1 | −1 | −1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

b) avec la nouvelle grille G, `gagne(-1,G)` renvoie faux mais `gagne(1,G)` renvoie vrai.

1. sachant que l'on a les fonctions `demandePosition(j,G)`, `insertion(j,x,G)` et `gagne(j,G)`, proposer un organigramme du jeu entier.
2. proposer une fonction `demandePosition(j,G)`
3. proposer une fonction `insertion(j,x,G)`
4. proposer une fonction `gagne(j,G)`
5. implémenter le jeu entier