Séance du 20.02. - Les tableaux

Solution du jeu du 1 fatal d'Edouard

```
def tour():
    tour = 1
    relancer = 1
    score = 0
    while tour <= 9:
            if(relancer == 1):
                de = randint(1,6)
                if(de == 1):
                    print("Le dé affiche : ", de)
                    score = 0
                    tour = 10
                else :
                    print("Le dé affiche : ",de)
                    score = score + de
                    tour = tour + 1
                    relancer = int(input("Voulez vous relancer ? (1-OUI | 0-NON)"))
            elif relancer == 0:
                tour = 10
    print("Votre score est de : ", score)
    return score
```

```
########## DEBUT DU PROGRAMME ############################
from random import *
scoreJ1 = 0
scoreJ2 = 0
continuer = True

while continuer == True:
    print("\n***** Tour du Joueur 1 *****")
    scoreTourJ1 = tour()
    scoreJ1 = scoreJ1 + scoreTourJ1
    print("\nScore du J1 = ",scoreJ1)
    if(scoreJ1 >= 100):
        continuer = False
        gagnant = "Joueur 1"
    if continuer:
        print("****************************")
        print("\n***** Tour du Joueur 2 *****")
        scoreTourJ2 = tour()
        scoreJ2 = scoreJ2 + scoreTourJ2
        print("\nScore du J2 = ",scoreJ2)
        if(scoreJ2 >= 100):
            continuer = False
            gagnant = "Joueur 2"
    print("****************************")
    print("\nScore du Joueur 1 : ",scoreJ1, "| Score du Joueur 2 : ",scoreJ2,"\n")

print("Le gagnant est le ",gagnant)
```

6 Tableaux

6.1 Que sont les tableaux ?

À savoir

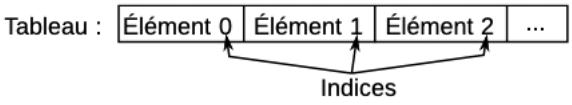
- Un tableau permet de stocker plusieurs valeurs dans une seule variable.
- Ces valeurs sont appelées éléments.
- Chaque élément est associé à un entier appelé indice ou rang qui repère sa position

6 Tableaux

6.1 Que sont les tableaux ?

À savoir

- Un tableau permet de stocker plusieurs valeurs dans une seule variable.
- Ces valeurs sont appelées éléments.
- Chaque élément est associé à un entier appelé indice ou rang qui repère sa position dans le tableau.



Remarque.
Il existe plusieurs structures de données permettant de regrouper plusieurs éléments dans une seule variable. Parmi elles :

- les tableaux, qui consistent à réserver un espace mémoire de taille fixe, et à stocker les éléments les uns après les autres dans cette espace.
- les listes chaînées, pour lesquelles on peut stocker les éléments n'importe où dans la mémoire. Cette structure de données a l'avantage d'avoir une taille extensible, mais pour s'y retrouver, chaque élément doit être accompagné de l'adresse mémoire de l'élément suivant, et accéder à un élément nécessite donc de suivre les adresses successives, ce qui est coûteux.

À savoir

En Python, les tableaux sont représentés par le type `list`.

Remarque.
L'appellation `list` est malheureuse, car les éléments de type `list` sont bien des tableaux et non des listes chaînées. Ce sont toutefois des tableaux dynamiques, c'est-à-dire un genre de tableaux améliorés, et qui présentent quelques avantages des listes.
Nous utiliserons par la suite quelques opérations qui sont interdites aux tableaux dans les autres langages.

6.2 Création, accès aux éléments, modifications

- En Python comme en pseudo-code, on représente un tableau en mettant entre crochets les éléments séparés par des virgules.
- Le tableau qui ne contient aucun élément s'appelle le tableau vide et se note simplement `[]`.

Exemple.
En pseudo-code :

```
tab ← [5, 1, 42]
Afficher : tab
tab_vide ← []
```

En Python :

```
tab = [5, 1, 42]
print(tab)
tab_vide = []
```

- Pour accéder à l'élément d'indice `i` du tableau `tab`, on utilise, en pseudo-code comme en Python, l'expression :

```
tab[i]
```

- On peut modifier les éléments d'un tableau (les tableaux sont un type dit "mutable") comme on modifie une variable (avec une affectation).
- Attention, les indices commencent à 0 en Python.

Exemple.
En pseudo-code :

```
tab ← [5, 1, 42, 8, 34]
Afficher : tab[2]
tab[3] ← 100
Afficher : tab
```

En Python :

```
tab = [5, 1, 42, 8, 34]
print(tab[2])
tab[3] = 100
print(tab)
```

Remarques.

- En Python, un tableau est non-typé : il peut contenir des variables de types différents.
- En Python, on peut aussi accéder aux éléments à partir de la fin du tableau en utilisant des indices négatifs.
- `tab[a:b]` est une tranche du tableau contenant les éléments d'indices allant de a (inclus) à b (exclu).

6.3 Longueur et parcours d'un tableau

- La longueur d'un tableau est le nombre d'éléments de celui-ci. En Python, elle est donnée par la fonction `len`.
- Pour parcourir l'un après l'autre les éléments d'un tableau, on utilise une boucle. Il y a deux façon de faire. Par exemple pour afficher les éléments :

Méthode 1 :
En pseudo-code :

```
Pour i Allant De 0 À
    longueur(tab) (exclu):
    Afficher : tab[i]
Fin Pour
```

En Python :

```
for i in range(0, len(tab)):

    print(tab[i])
```

Méthode 2 :
En pseudo-code :

```
Pour elt Dans tab:
    Afficher : elt
Fin Pour
```

En Python :

```
for elt in tab:
    print(elt)
```

Exemple.
Ajout de 1 à tous les éléments d'un tableau d'entiers :

En pseudo-code :

```
Algorithme : ajout de 1
Variables : tab(tableau d'
   entiers), i (entier)
DEBUT
tab ← [30, 18, 4, 2, 9]
Pour i Allant De 0 À
   longueur(tab) (exclu):
    tab[i] ← tab[i] + 1
Fin Pour
Afficher : tab
FIN
```

En Python :

```
# Algorithme : ajout de 1



tab =  [30, 18, 4, 2, 9]
for i in range(0, len(tab)):

    tab[i] = tab[i] + 1

print(tab)
```

6.4 Opérations sur les tableaux en Python

6.4.1 Concaténation

Si l'on dispose de deux tableaux, on peut vouloir les coller l'un à l'autre pour n'avoir plus qu'un tableau. Cette opération s'appelle la concaténation.
Pour faire une concaténation, on utilise l'opérateur +.

Exemple.
En pseudo-code :

```
tab ← [1, 2, 3] + [4, 5]
Afficher : tab
tab2 ← tab + [6, 7]
Afficher : tab2
```

En Python :

```
tab = [1, 2, 3] + [4, 5]
print(tab)
tab2 = tab + [6, 7]
print(tab2)
```

Remarques.

- L'opérateur + ne modifie pas directement un tableau. Il crée un nouveau tableau qu'il faut affecter à une variable.
- On peut également concaténer plusieurs fois un tableau à lui-même avec le symbole : ∗.

```
tab = [1, 2] * 3
print(tab)
```

6.4.2 Ajout d'un élément à la fin d'un tableau

Pour ajouter l'élément `elt` à la fin du tableau `tab`, on peut :

- utiliser la concaténation :

En pseudo-code :

```
tab ← tab + [elt]
```

En Python :

```
tab = tab + [elt]
```

- utiliser la méthode `append` :

En pseudo-code :

```
Ajouter elt à la fin de tab
```

En Python :

```
tab.append(elt)
```

Exemple.

En pseudo-code :

```
tab ← [1, 2, 3]
tab ← tab + [4]
Ajouter 5 à la fin de tab
Afficher : tab
```

En Python :

```
tab = [1, 2, 3]
tab = tab + [4]
tab.append(5)
print(tab)
```

6.4.3 Copie d'un tableau

Un tableau en Python est stocké sous la forme d'une adresse mémoire. Lorsqu'on veut copier un tableau, c'est l'adresse du tableau qui est copiée. On obtient alors deux variables qui pointent vers le même tableau, et toute modification de l'une modifiera aussi l'autre.

Exemple.
Dans l'exemple ci-dessous, a est modifié en même temps que b :

```
a = [1,2,3,4,5]
b = a
b[1] = 10
print(a)
```

Pour copier un tableau `tab1` dans un tableau `tab2` stocké dans un emplacement mémoire différent, il faut utiliser l'instruction :

```
tab2 = tab1[:]
```

Exemple.

```
a = [1,2,3,4,5]
b = a[:]
b[1] = 10
print(a)
```

6.4.4 Création d'un tableau de taille donnée

Il arrive souvent que l'on ne connaisse pas dès le départ tous les éléments d'un tableau. Pour construire un tableau petit à petit, il y a deux façons de procéder :

- **La méthode 1 :** consiste à créer un tableau de la taille voulue dont tous les éléments sont identiques (par exemple 0), puis à modifier les éléments un par un. (Cela nécessite de connaître la taille à l'avance).

  Exemple. Remplissons par exemple un tableau avec dix nombres aléatoires.

```
tab = [0]*10  # On crée un tableau contenant 10 zéros
for i in range(0,10) :
    tab[i] = randint(1,100) # On modifie les éléments
print(tab)
```

- **La méthode 2 :** consiste à créer un taleau vide auquel on ajoute des éléments petit à petit.

  Exemple. Remplissons par exemple un tableau avec dix nombres aléatoires.

```
tab = []  # On part du tableau vide
for i in range(0,10) :
    tab = tab + [randint(1,100)] # Ajout d'un élément
print(tab)
```

6.4.5 Autres opérations

Si on dispose d'un tableau appelé `tab`, on peut utiliser différentes fonctions :

- Faire la somme des éléments du tableau : `somme = sum(tab)`
- Trouver le maximum du tableau : `maximum = max(tab)`
- Trouver le minimum du tableau : `minimum = min(tab)`

Il existe aussi différentes méthodes de tableaux :

- Trier un tableau : `tab.sort()`
- Trouver l'indice d'un élément : `tab.index(2)`
- Inverser un tableau : `tab.reverse()`
- Enlever un élément : `tab.remove(2)`
- La liste complète des méthodes de tableaux s'obtient en tapant dans l'interpréteur : `help(list)`

Remarque.
On s'interdira d'utiliser les fonctions et méthodes toutes faites pour tous les exercices de ce document.

6.5 Exercices

Exercice 39 (Pour se tester).

1) Qu'est un tableau ?
2) Comment est représenté un tableau en Python
3) Comment accéder à l'élément d'indice `i` d'un tableau `tab` ?
4) Comment parcourir un tableau ?
5) Comment concaténer deux tableaux ?
6) Comment ajouter un élément à la fin d'un tableau ?
7) Comment construire un tableau de taille `n` contenant par exemple des entiers aléatoires ?

Exercice 40.
Traduire en Python l'algorithme donné ci-dessous en pseudo-code et le tester.

```
Algorithme : remplacement des 3 par des 8
Variables :
DEBUT
tab ← [3, 1, 2, 3, 8, 9, 3]
Pour i Allant De 0 À longueur(tab) (exclu) :
    Si tab[i] = 3:
        tab[i] ← 8
    Fin Si
Fin Pour
FIN
```

Exercice 41.
On veut écrire en Python une procédure `afficheL` prenant en paramètre un tableau et affichant l'un après l'autre tous les éléments de ce tableau. Reproduire puis corriger la fonction ci-dessous en testant autant que nécessaire :

```
function affiche_tab(tab):
    tab = []
    for i in range(1, 10):
        print [i]
```

Exercice 42 (Les algo de référence).

1) Écrire en pseudo-code et en Python une fonction `somme` qui prend en paramètre un tableau et retourne la somme des éléments du tableau.
   Par exemple `somme([2, 4, 10])` devra retourner `16`.
2) Écrire en pseudo-code et en Python une fonction `compte` qui prend en paramètre une valeur `val` et un tableau `tab` et qui retourne le nombre d'apparitions de `val` dans `tab`.
   Par exemple `compte(8, [8, 4, 1, 8, 10, 8])` devra retourner `3`.
3) Écrire en pseudo-code et en Python une fonction `maximum` qui prend en paramètre un tableau et retourne la plus grande valeur de ce tableau.
   Par exemple `maximum([1, 42, 8, 5])` devra retourner `42`.
4) Écrire en pseudo-code et en Python un algorithme qui :
   - demande de saisir un entier `n`,
   - crée un tableau de taille `n` composée d'entiers aléatoires entre 1 et 10,
   - teste les fonctions précédentes sur ce tableau (pour le paramètre `val`, on choisira 8).

On indice les cases d'un échiquier avec deux indices i et j variant tous deux de 1 à 8. La case (i, j) est sur la ligne i et la colonne j. Par convention, la case $(1, 1)$ est noire.

Exercice 16 Couleurs

Écrire un programme demandant à l'utilisateur de saisir les deux coordonnées i et j d'une case, et lui disant s'il s'agit d'une case blanche ou noire.

Exercice 17 Cavaliers

Écrire un programme demandant à l'utilisateur de saisir les coordonnées (i, j) d'une première case et les coordonnées (i', j') d'une deuxième case. Dites-lui ensuite s'il est possible de déplacer un cavalier de (i, j) à (i', j').