TP – Tableaux et listes

Objectifs :

- ⇨ Découvrir les listes et les tableaux et commencer à les utiliser
- ⇨ Apprendre à manipuler des chaînes de caractère

Lorsque l'on souhaite manipuler une grande quantité de données, l'utilisation des simples variables n'est plus suffisante. On va prendre l'exemple de la pyramide des âges pour l'illustrer.

I – Le problème de la pyramide des âges

On veut faire un programme qui demande l'âge d'une personne et indique combien de français ont le même âge que lui. Les chiffres étant disponibles sur le site de l'insee, il est facile de les récupérer pour les incorporer dans un programme.
Pour mémoriser les valeurs, on pourrait créer une variable par âge :

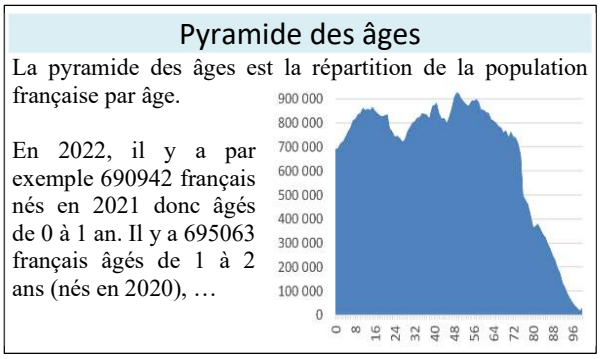
```
age0 = 690942
age1 = 695063
age2 = 716123
```

…

Ce qui nous donne 101 variables de `age0` à `age100`. Cela fait déjà beaucoup de variables, mais le problème est surtout pour les utiliser. Pour afficher le nombre de français ayant le même âge, il faudrait faire un programme du type :

```
age = int(input("Quel est votre age ? ")
print("Nombre de français ayant votre age (même année de naissance) :", end = "")
if age < 1 : print(age0)
elif age < 2 : print(age1)
elif age < 3 : print(age2)
```

…



Ce qui donne un code long et fastidieux. Si on complexifie un peu la requête (nombre de français entre deux âges quelconques par exemple), cela peut devenir vite inextricable.

Il faudrait que l'on puisse stocker tous les nombres dans une structure ordonnée permettant d'accéder au nombre de français directement par l'âge. C'est ce que permettent les tableaux.

II – Les tableaux

En python les tableaux n'existent pas directement, mais le langage propose des listes (type python : « `list` ») qui sont des objets plus complexes (on verra les différences dans le cours) mais qui peuvent entre autre être utilisées comme des tableaux.

1) Création d'un tableau

En python, une liste est délimitée par des crochets (`[ … ]`) et ses éléments sont séparés par des virgules.
Pour créer un tableau en python, on utilise la syntaxe suivante :

```
tab = [valeur] * nbElements
```

Ceci crée un tableau nommé `tab` et contenant `nbElements` valant tous `valeur`.
Exemple :
`maListe = [3.14] * 7` crée le tableau `[3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14]`

On peut créer des tableaux d'entiers, de pseudo-réels (nombres à virgules), de textes, de booléens, … en réalité de n'importe quel type d'objet.

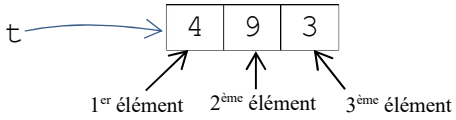
Il est possible de créer un tableau vide : `tabVide = []` ou bien de créer un tableau contenant déjà des données que l'on précise : `tabRempli = [valeur1, valeur2, valeur3, …]`

Exemples :

```
liste1 = []                 # Ce tableau est vide
tableau2 = [1,2,3,5,7,11]   # Ce tableau contient les 6 premiers nombres premiers
```

2) Utilisation d'un tableau

Un tableau est toujours **ordonné** et l'ordre dans lequel sont les données du tableau est conservé. Ainsi le deuxième élément du tableau `t = [4, 9, 3]` est `9`.
Tout se passe comme si un tableau était constitué de cases les unes à la suite des autres :



Pour accéder aux éléments d'un tableau, il existe plusieurs méthodes :

a. On veut accéder à tous les éléments les uns après les autres

Dans ce cas, on utilise naturellement une boucle for. En effet, une liste en python est un itérateur. On pourra donc avoir une structure du type :

```
tableau = [1, 5, 3, -1, 2]
somme = 0
for element in tableau :
    print(element)
    somme = somme + element
print('total :', somme)
```

Le programme affichera :

```
1
5
3
-1
2
total : 10
```

b. On veut accéder à un élément en particulier

On peut accéder à un élément de tableau en indiquant son **indice**. On a vu précédemment que les données du tableau étaient stockées en ordre les unes derrière les autres. Elles ont ainsi toutes un **numéro d'ordre <u>en partant de 0</u> pour le premier élément** du tableau jusqu'à n-1 pour un tableau de longueur n.

Exemple :

```
tableau = [8, 5, 13, -1, 72]
```

`tableau[0]` vaut `8`
`tableau[1]` vaut `5`
`tableau[2]` vaut `13`
`tableau[3]` vaut `-1`
`tableau[4]` vaut `72`

c. On veut connaitre la taille du tableau

Pour cela il existe une fonction built-in très simple et qui fonctionne sur la plupart des objets de python : la fonction `len()`. Elle renvoie la taille du tableau qui lui est passé en argument.

Exemples :

```
t = [1, 2, 3, 4, 5]
taille = len(t)               # taille vaut 5
tabVide = []
longueur = len(tabVide)       # longueur vaut 0
```

III – Applications

1) Retour sur la pyramide des âges

Maintenant qu'on sait définir et utiliser un tableau, utilisons ces notions pour résoudre notre problème de pyramide des âges.

Q1 : Ouvrir le programme « Question1.py » qui contient la définition du tableau `pyramide` contenant le nombre de français âgés de 0 à 1 an, puis de 1 à 2 ans, 2 à 3 ans, …

```
Français âgés de 0 à 1 ans : 690942
Français âgés de 1 à 2 ans : 695063
Français âgés de 2 à 3 ans : 716123
Français âgés de 3 à 4 ans : 725576
Français âgés de 4 à 5 ans : 743039
```

1) Ecrire une procédure `affiche_pyramide(p)` qui prend en argument un tableau de pyramide des âges `p` et affiche les valeurs qu'il contient sous la forme indiquée ci-contre.
2) Ecrire une procédure `meme_age(p)` qui demande son âge à l'utilisateur et affiche le nombre de français qui ont le même âge que lui dans la pyramide des âges `p` fournie en paramètre.
3) Ecrire une fonction `tranche_age(p, age_min, age_max)` qui renvoie le nombre total de personnes ayant un âge compris entre `age_min` et `age_max` dans la pyramide des âges `p` fournie en paramètre. S'en servir pour écrire un programme qui demande deux âges et affiche le nombre de français qui ont un âge compris entre ces deux valeurs.

Vérification : Il y a 2 102 128 français entre 0 et 2 ans et 12 648 343 français entre 15 et 30 ans.

2) Travail sur les tableaux

Q2 :

1) Ecrire une fonction `tableau_aleatoire` qui prend un argument obligatoire `n` représentant la taille du tableau (le nombre d'entier qu'il doit contenir) et deux arguments optionnels `min` et `max` représentant les deux bornes entre lesquels doivent être les nombres du tableau. Cette fonction doit renvoyer un tableau contenant `n` entiers aléatoires compris entre `min` et `max` (prendre comme valeurs par défaut 0 et 10). Testez votre fonction avec différentes valeurs de paramètres.
2) Ecrire une fonction `nombre_occurences(tab, n)` qui renvoi le nombre d'occurrence de l'entier n dans le tableau tab. Par exemple `nombre_occurences([3,2,0,6,4,3,7],3)` renvoie `2`. Ecrire un bout de programme principal en utilisant la fonction précédente (`tableau_aleatoire`) pour tester que votre fonction donne le résultat attendu.
3) Ecrire une fonction `minimum(tab)` qui renvoie l'entier le plus petit dans le tableau `tab` et une fonction `pos_min(tab)` qui renvoie l'indice de l'entier le plus petit dans le tableau `tab`. Tester vos fonctions en utilisant `tableau_aleatoire`.

Vérification : `minimum([13,9,6,4,3,7])` renvoie `3` et `pos_min([13,9,6,4,3,7])` renvoie `4`.

3) Les chaînes de caractère

La plupart des méthodes, opérateurs ou fonctions qui marchent sur les listes, fonctionnent également avec les chaînes de caractères qui sont alors traitées comme des tableaux.

Exemple :

```
c = "abc"*3      # c vaut 'abcabcabc'
print(c[4])      # affiche le caractère 'b' (5° caractère de la chaîne)
l = len(c)       # l vaut 9 (nombre de caractères dans la chaîne c)
```

Q3 :
On veut écrire une fonction `nombre_occurences(c, l)` qui renvoi le nombre d'occurrence de la lettre `l` dans la chaîne `c`. Par exemple `nombre_occurences("Physique-chimie",'i')` renvoie `3`. La fonction programmée à la question 2 peut-elle être réutilisée telle-quelle (si oui, expliquez pourquoi), doit-on faire des adaptations (si oui, précisez lesquelles) ou doit-on la ré-écrire complètement (dans ce cas écrire la fonction correspondante) ?

Q4 :
On veut écrire une fonction qui teste si une chaine est un palindrome, c'est-à-dire un mot ou une phrase qui s'écrit pareil dans les deux sens.
Par exemple « kayak », « ressasser », « radar », sont des palindromes.
Ouvrir le programme « Question4.py ». Utiliser la trame fournie pour écrire la fonction `estPalindrome` qui prend en argument une chaîne et renvoi un booléen (True si la chaine est un palindrome, False sinon).

4) Analyse et modification d'un programme

Analysons le programme suivant (qui se trouve dans le répertoire de l'activité dans « Question5.py ») :

```
def f1(a) :
    b = [0]*a
    for i in range(a) :
        b[i] = 2**i
    return b

def f2(a) :
    n = len(a)
    for i in range(n) :
        a[i] = a[n - 1 - i]
    return a

print(f2(f1(8)))
```

Q5 :
1) SANS EXECUTER LE PROGRAMME, commenter chaque ligne pour indiquer ce qu'elle fait. Rajouter un commentaire multiligne sous le nom de chaque fonction pour indiquer son rôle et la signification de ses arguments d'entrée et de sa valeur de retour.
2) Exécuter le programme et valider les réponses précédentes (ou apporter les corrections nécessaires).
3) La fonction f2 ne fonctionne pas comme elle le devrait puisque la deuxième partie du tableau est le miroir de la première. Modifier la fonction pour qu'elle réalise une vraie inversion de l'ordre des éléments du tableau.

PROLONGEMENT :

Q1 : Ecrire une fonction qui prend en argument un entier n et qui renvoi un tableau d'entiers de 0 à n-1.

Q2 : On considère la fonction f(x) = (1-x)(9-x) définie pour tout entier x compris entre 0 et 10 (Il s'agit en réalité d'une suite).

Construire un programme comportant les fonctions (ou procédures) suivantes :

- « `f` »qui prend en entrée un entier `x` et qui renvoie en sortie un entier qui vaut f(x)
- « `table` » qui ne prend aucune entrée et qui renvoie en sortie un tableau d'entiers contenant les 11 valeurs de f(x) lorsque x varie entre 0 et 10 (utiliser une formule et une boucle)
- « `afficher_table` » qui prend en entrée un tableau de valeurs entières, ne renvoie aucune sortie et qui affiche pour chaque case mémoire son contenu précédé de son numéro de 0 à (taille du tableau -1)

Le programme principal utilisera les trois fonctions précédentes pour afficher la table de valeurs de la fonction f.

Q3 : Rajouter la fonction « `trace_table` » qui trace la fonction à l'aide de la bibliothèque `turtle`.