DECOUVERTE DES TABLEAUX SOUS PYTHON :

Introduction au tableau :

Imaginez que vous souhaitiez utiliser dans un programme un ensemble de nombres, par exemple le nombre de jours de chaque mois de l'année. Une première solution serait d'utiliser 12 variables :

```
nbJoursJanvier = 31
nbJoursFevrier = 29
nbJoursMars = 31
nbJoursAvril = 30
nbJoursMai = 31
nbJoursJuin = 30
nbJoursJuillet = 31
nbJoursAout = 31
nbJoursSeptembre = 30
nbJoursOctobre = 31
nbJoursNovembre = 30
nbJoursDecembre = 31
print("Nombre de jours en Janvier :")
print(nbJoursJanvier)
```

```
Nombre de jours en Janvier :
31
```

Plutôt long et répétitif, non ?

Comme vous le savez, une variable est comme une boîte qui permet de stocker de l'information. Au lieu d'utiliser une boîte pour chaque mois de l'année, une autre solution serait d'utiliser une seule grosse boîte appelée *nbJours*, et de mettre plusieurs petites boîtes dans la grande. Pour pouvoir identifier ces petites boîtes on leur donne alors un numéro. Ainsi on demandera à lire la valeur de "la petite boîte numéro 5 située dans la boîte 'nbJours'".

nbJours											
0	1	2	3	4	5	6	7	8	9	10	11
31	29	31	30	31	30	31	31	30	31	30	31

Pour le moment vous avez vu différents type de variables qui peuvent contenir différentes choses : des entiers, des booléens et des nombres à virgule. Il existe un autre type de variable qui permet des stocker ces "petites boîtes" identifiées par des numéros. Ces variables s'appellent des tableaux.

Les tableaux :

Pour créer un tableau, on utilise le code suivant :

```
nbJours = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

Ceci va créer un tableau *nbJours* contenant 12 valeurs entières.

Les positions de ces valeurs (les numéros des petites boîtes) vont de 0 (pour la première) à 11 (pour la dernière). **La numérotation démarre en effet à 0** en Python.

On peut ensuite accéder aux éléments du tableau. Par exemple, le code suivant affiche la valeur des éléments numéros 0 et 5 du tableau *nbJours* :

```
print(nbJours[0])
print(nbJours[5])
```

```
31
30
```

Si on regarde à nouveau le code tout en haut, celui qui utilisait 12 variables, et qu'on regarde le code suivant qui fait la même chose mais en utilisant des tableaux

```
nbJours = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
print("Nombre de jours en Janvier :")
print(nbJours[0])
```

```
Nombre de jours en Janvier :
31
```

alors on voit très bien l'un des intérêts de ces tableaux : le code est beaucoup plus court ! Imaginez si au lieu de mémoriser 12 éléments on avait voulu en mémoriser 100 ou 1000 !

Les tableaux ont bien sur d'autres avantages et vous les découvrirez au fur et à mesure des exercices.

ACCES EN DEHORS DU TABLEAU :

L'une des erreurs les plus courantes avec les tableaux est d'essayer de lire un élément qui n'existe pas, c'est-à-dire d'utiliser un indice trop grand :

```
hauteurs = [1, 2, 3]
print(hauteurs[0])
print(hauteurs[1])
print(hauteurs[2])
print(hauteurs[3])
```

```
Traceback (most recent call last):
  File "./run/exe", line 5, in
    print(hauteurs[3])
IndexError: list index out of range
```

L'erreur a donc lieu à la ligne 5 : les 3 premières valeurs se sont donc affichées comme il fallait mais comme il n'existe pas d'élément d'indice 3 (les indices vont de 0 à 2) une erreur est survenue. Si on essaie de traduire ce message cela veut donc dire "Erreur d'indice : l'indice du tableau est en dehors de l'intervalle"

TABLEAUX ET INDICES NEGATIFS :

Regardons sur un exemple :

```
hauteurs = [1, 2, 3]
print(hauteurs[0])
print(hauteurs[-1])
print(hauteurs[-2])
print(hauteurs[-3])
```

```
1
3
2
1
```

On remarque donc que l'indice "-1" correspond au premier élément en partant de la fin, que "-2" correspond au second élément en partant de la fin et ainsi de suite. Attention cependant :

```
hauteurs = [1, 2, 3]
print(hauteurs[-4])
```

```
Traceback (most recent call last):
  File "./run/exe", line 2, in
    print(hauteurs[-4])
IndexError: list index out of range
```

On obtient une erreur pour l'indice "-4" car le quatrième en partant de la fin n'existe pas.

Les indices négatifs sont donc valables en Python mais nous vous conseillons de ne pas les utiliser car ils sont source d'erreurs.

Tableaux de taille variable

Jusqu'à présent, nous avons vu comment créer un tableau de taille fixe en indiquant les valeurs initiales de ses éléments. Mais comment faire si le tableau contient 1 000 éléments ? Ou si sa taille dépend des entrées ?

Voici comment créer un tableau de taille 1 000 contenant uniquement des zéros :

```
notes = [0] * 1000
```

On peut bien entendu utiliser une variable :

```
nbNotes = 1000
notes = [0] * nbNotes
```

Il est également possible d'utiliser une valeur initiale différente de 0

```
nbNotes = 1000
notes = [20] * nbNotes
```

ou d'utiliser une variable comme valeur initiale :

```
nbNotes = 1000
noteInitiale = 20
notes = [noteInitiale] * nbNotes
```

CALCULER LA TAILLE D'UN TABLEAU :

Dans les exercices de ce chapitre, vous manipuliez toujours des tableaux dont la taille était sauvegardée dans une variable. Le langage Python propose toutefois un moyen de connaître la taille d'un tableau, à partir du tableau lui-même.

Supposons qu'on ait déclaré un tableau de la manière suivante :

```
hauteur = [10, 48, -5, 99, -20, 4, 32, 16, 0, 100, 42]
```

alors, si on souhaite connaître sa taille, il suffit de faire :

```
tailleHauteur = len(hauteur)
```

TABLEAUX A PLUSIEURS DIMENSIONS :

On a vu comment créer un tableau à une dimension, en utilisant la syntaxe :

```
tableau = [1, 2, 3, 4, 5]
```

Imaginons qu'on souhaite stocker dans un tableau (à deux dimensions) la table d'addition suivante :

```
0       1       2       3       4
1       2       3       4       5
2       3       4       5       6
3       4       5       6       7
4       5       6       7       8
```

En Python, un tel tableau se déclare ainsi :

```
tableAddition = [[0, 1, 2, 3, 4], [1, 2, 3, 4, 5], [2, 3, 4, 5, 6], [3, 4, 5, 6, 7], [4, 5, 6,

print(tableAddition[3][4])
```

```
↳ 7
```

Ainsi, la syntaxe pour déclarer un tableau à 2 dimensions est :

```
grille = [ligne1, ligne2, ligne3, ..., ligneN]
```

et pour accéder à un élément du tableau, on utilise la syntaxe :

```
grille[indiceLigne][indiceColonne]
```

Trier un tableau

Il est facile de trier un tableau en Python car un algorithme de tri est déjà fourni. Voici un code complet, nous l'expliquerons en dessous :

```
# Défini le tableau
poids = [45, 80, 2]

# Tri le tableau
poids.sort()

# Affiche le tableau
for indice in range(3):
    print(poids[indice])
```

```
↳ 2
45
80
```

Il suffit donc d'appeller la fonction `sort()` sur le tableau, à l'aide du code

```
poids.sort()
```

afin de demander le tri du tableau.

Algorithmes de tri

Ici, nous avons simplement utilisé un tri qui existe déjà dans Python. Il est bien sur possible de programmer son propre tri (et il existe beaucoup de tris différents !) mais pour le moment le plus simple est d'utiliser le tri déjà fourni. Nous aurons l'occasion de vous présenter les différents algorithmes de tri plus tard.