Chapitre 14 - Tuple et portée des variables

Les listes, les dictionnaires permettent de stocker des données. On voit ici, encore une autre structure de stockage : les tuples. Mais rassurez-vous, elle ressemble beaucoup aux listes.

On s'intéresse également à ce que l'on appelle la portée des variables.

1- LES TUPLES :

Définition :

Un **tuple** est une liste qui ne peut pas être modifiée

a. CREER UN TUPLE :

Exemples ci-dessous exécutés dans la console :

On crée un tuple avec des parenthèses et des valeurs séparées par une virgule : `>>> a = (8 , 'février' , 2024)`	`>>> a`
Les parenthèses ne sont pas obligatoires, mais elles facilitent la lisibilité du code : `>>> b = 8 , 'février' , 2024`	`>>> b`

b. LIRE UNE VALEUR DANS UN TUPLE :

Le tuple est une sorte de liste, on peut donc utiliser la même syntaxe pour lire les données d'un tuple: `>>> a[0]`	
`for elt in b :` `  print(elt)`	
`for i in range(len(a)) :` `  print(a[i])`	
`>>> 'mars' in a`	
`>>> 2024 in a`	
`>>> a[1]="mars"`	

c. A QUOI SERT UN TUPLE ?

⇨ Le tuple permet une affectation multiple :

On peut affecter des valeurs à plusieurs variables en même temps :
`jour,mois,annee = 8,'février',2024`
	`>>> jour`
	`>>> mois`
	`>>> annee`

On peut inverser l'affectation entre 2 variables :
`x = 1`
`y = 2`
`x , y = y , x`
	`>>> x`
	`>>> y`

⇨ Le tuple permet de renvoyer plusieurs valeurs dans un return de fonction. Par exemple avec la fonction ci-contre :

```
def maFonction() :
    jour = 8
    mois = "février"
    annee = 2024
    return jour,mois,annee
```

On peut exécuter la fonction de cette manière :
`date = maFonction()`
	`>>> date`
	`>>> date[0]`
	`>>> date[1]`
	`>>> date[2]`

On peut également l'exécuter de cette autre manière :
`j,m,a = maFonction()`
Ou de celle-ci :
`(j,m,a) = maFonction()`
	`>>> j`
	`>>> m`
	`>>> a`

2- PORTEE DES VARIABLES :

Un code python est un fichier texte qui contient des lignes pythons. Elles sont exécutées de haut en bas.

On définit généralement les fonctions en début de code. En fin de code, on retrouve le programme principal qui exécutera les fonctions définies. Une fonction peut également être appelée dans une autre fonction.

Une fonction est une entité que l'on peut qualifier **d'hermétique**. Les variables utilisées à l'intérieur ne peuvent pas, dans la plupart des cas, interférer avec celles utilisées à l'extérieur. Cette propriété est un gage de sécurité. Elle permet de construire un code composé de multiples fonctions qui pourront être créées indépendamment. Les variables utilisées dans les différentes fonctions ne seront jamais en conflit.

Les choses ne sont pourtant pas aussi simples. Certains cas de figure ne respectent pas ce principe de fonctionnement. On voit cela ci-dessous :

Règles de fonctionnement pour une variable créée dans une fonction :

- Si une variable V est créée dans une fonction, elle ne pourra être utilisée qu'à l'intérieur de la fonction. A l'extérieur, cette variable V n'a pas d'existence.

```
def maFonction() :
    V = 2024
```

  Si on veut récupérer la valeur de V à l'extérieur, il faut renvoyer la valeur de V par un ***return***.

- Si on écrit `global V`, la variable V aura le statut de variable **globale**. La fonction ne sera plus du tout hermétique pour V.

```
def maFonction() :
    global V
    V = 2024
```

Règles de fonctionnement pour une variable créée dans le programme principal :

Si une variable V est définie ou plus généralement a une existence dans le **programme principal**, alors :

- V est accessible en LECTURE dans toutes les fonctions.
- V peut être modifiée A L'INTERIEUR d'une fonction :
  - Si V est retournée ensuite dans le programme principal,
  - Si V est une **liste** ou un **dictionnaire**, il n'est alors pas nécessaire de retourner V,
  - Si V a le statut de **variable globale**, il n'est alors pas nécessaire de retourner V.