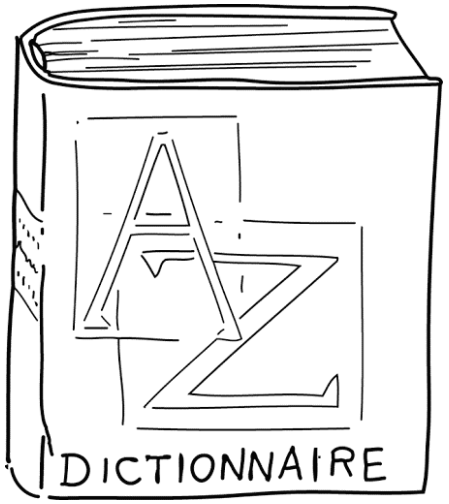
Un ou deux Objets à connaitre

Les tuples, les dictionnaires et les fonctions



Les tuples

Un tuple n’est rien d’autre qu’une liste, à la seule différence que le tuple est une liste qu’on ne peut pas modifier. Pour créer un tuple, cela se fait de la même manière que pour une liste, sauf que les crochets sont remplacés par des parenthèses.

Syntaxe :

```
mon_tuple=(ma_premiere_valeur,ma_deuxieme_valeur,
ma_troisieme_valeur,...)
```

Créons un tuple contenant cinq valeurs :

```
mon_tuple=(1,"deux",3,4,5)
print(type(mon_tuple))
print(mon_tuple)
```

```
<class 'tuple'>
(1, 'deux', 3, 4, 5)
```

Pour accéder à un élément d'un tuple, on utilise l'indexage simple, comme pour les listes. On ne peut pas ajouter, supprimer ou modifier des éléments comme pour les listes :

```
mon_tuple[0]
```

```
1
```

```
mon_tuple[2] = 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-5a4b944ad511> in <module>
----> 1 mon_tuple[2] = 2

TypeError: 'tuple' object does not support item assignment
```

Mais quelle est l'utilité des tuples par rapport aux listes ? Les deux plus importantes sont :

- l'assignation multiple
- la protection en écriture.

Imaginons, par exemple, que l'on veuille assigner des valeurs à plusieurs variables, on utilisera alors les tuples.

```
(a,b) = (1,2)
print(a, b)
```

```
1 2
```

```
a, b = 3, 4
print(a, b)
```

```
3 4
```

```
a=7
```

```
print(a, b)
```

```
7 4
```

Les dictionnaires

Un dictionnaire est une structure qui :

- **n'est pas ordonnée** (contrairement aux listes)
- **peut être modifiée**
- **est indexée**, tout comme les listes.

Il est aussi possible de mélanger les types de valeurs dans un dictionnaire (caractères, entiers...).

Le principe des dictionnaires est de lier une clé à une valeur et de pouvoir accéder aux valeurs grâce à ces clés.

```
notes_eleves = {
  "Anthony": 15,
  "Marie": 12,
  "Julien": "absent",
  "Mohammad": 9,
  "Brahiman": 17
}
print(notes_eleves)
```

```
{'Anthony': 15, 'Marie': 12, 'Julien': 'absent', 'Mohammad': 9, 'Brahiman': 17}
```

Plus puissant que les listes, le dictionnaire permet d'accéder aux valeurs en donnant la clé, plutôt que l'index.

```
print(notes_eleves["Anthony"])
```

```
15
```

Pour ajouter un nouveau couple clé-valeur dans un dictionnaire existant, la syntaxe est la suivante.

```
notes_eleves["Julien"] = 14
print(notes_eleves["Julien"])
```

```
14
```

Pour ajouter un nouveau couple clé-valeur dans un dictionnaire existant, la syntaxe est la suivante.

```
notes_eleves["Rafik"] = "C'est le formateur"
print(notes_eleves)
```

```
{'Anthony': 15, 'Marie': 12, 'Julien': 14, 'Mohammad': 9, 'Brahiman': 17, 'Rafik': "C'est le formateur"}
```

Pour les dictionnaires, il existe une méthode, .keys(), qui retourne la liste de toutes les clés d'un dictionnaire.

```
print(notes_eleves.keys())
```

```
dict_keys(['Anthony', 'Marie', 'Julien', 'Mohammad', 'Brahiman', 'Rafik'])
```

Parcourir un dictionnaire, c'est-à-dire itérer sur chaque élément de celui-ci, est possible soit par clés-valeurs, soit juste par les clés, soit juste par les valeurs. Pour cela, on utilisera une boucle for.

```
for i in notes_eleves.values():
    print(i)
```

```
15
12
14
9
17
C'est le formateur
```

La méthode .items() retourne une liste et chacun des éléments de la liste est un tuple qui lui-même contient un couple clés-valeur.

```
print(notes_eleves.items())
```

```
dict_items([('Anthony', 15), ('Marie', 12), ('Julien', 14), ('Mohammad', 9), ('Brahiman', 17), ('Rafik', "C'est le formateur")])
```

On va pouvoir donner deux noms de variables à la boucle for :

- La première variable prendra la valeur de la première valeur du tuple, la clé.
- La deuxième variable prendra comme valeur la valeur associée à la clé.

```
for i,j in notes_eleves.items():
    print(i,j)
```

```
Anthony 15
Marie 12
Julien 14
Mohammad 9
Brahiman 17
Rafik C'est le formateur
```

Intérêt des fonctions

- Les fonctions permettent de créer du code réutilisable.
- De plus, elles permettent d'associer un nom à une partie du code, nom qui idéalement doit décrire au mieux le traitement effectué.
- On peut toujours essayer d'écrire un programme sans utiliser de fonctions. Mais si le programme devient long, la gestion du code va devenir fatigante. Il vous poussera à effectuer un copier-coller afin de vous éviter l'écriture d'une fonction supplémentaire.

Intérêt des fonctions

Prenons l'exemple du code suivant :

```
TotalTTC = 0
Prix1HT = 18
TVA = 0.2
Prix1TTC = Prix1HT*(1+TVA)
TotalTTC = TotalTTC + Prix1TTC
Prix2HT = 21
Prix2TTC = Prix2HT*(1+TVA)
TotalTTC = TotalTTC + Prix2TTC
print(TotalTTC)
```

```
46.8
```

Le code est lisible. Les variables sont correctement nommées.

Cependant, pour savoir ce que fait ce bout de code, il faut un peu se concentrer pour vérifier les actions effectuées à chaque ligne.

Intérêt des fonctions

Nous allons améliorer la structure de notre code en créant une fonction pour cette tâche :

```
def PrixTTC(prixHT):
    TVA = 0.2
    return prixHT * (1 + TVA)

Prix1HT = 18
Prix2HT = 21

TotalTTC = 0
TotalTTC = TotalTTC + PrixTTC(Prix1HT)
TotalTTC = TotalTTC + PrixTTC(Prix2HT)

print(TotalTTC)
```

```
46.8
```

La déclaration ou définition d'une fonction se fait par l'utilisation du mot-clé def. La syntaxe d'une déclaration doit respecter l'ordre suivant :

- mot-clé **def**
- **nom** de la fonction : PrixTTC
- parenthèse ouvrante : (
- **noms des paramètres** séparés par des virgules s'il y en a plusieurs
- parenthèse fermante :)
- fin de la déclaration par l'utilisation d'un deux-points à la fin de la ligne
- Instructions
- Le mot-clé return est utilisé pour renvoyer le résultat calculé par la fonction.

Les paramètres nommés

Une des facilités du langage Python, est de nommer les arguments passés lors de l'appel d'une fonction. Cela peut sembler représenter un travail supplémentaire.

```python
def PrixTTC(prixHT, TVA):
    return prixHT * (1 + TVA)

TotalTTC = 0
TotalTTC = TotalTTC + PrixTTC(18, 0.2)
TotalTTC = TotalTTC + PrixTTC(0.4, 21)

print(TotalTTC)
```

```
30.4
```

```python
def PrixTTC(prixHT, TVA):
    return prixHT * (1 + TVA)

TotalTTC = 0
TotalTTC = TotalTTC + PrixTTC(prixHT=18, TVA=0.2)
TotalTTC = TotalTTC + PrixTTC(TVA=0.4, prixHT=21)

print(TotalTTC)
```

```
51.0
```

Les valeurs par défaut des paramètres

Inutile de transmettre des valeurs pour tous ces paramètres, car souvent ils ont des valeurs par défaut.

```
def PrixTTC(PrixHT,TVA=0.2):
    return PrixHT * (1 + TVA)

# syntaxe utilisant la valeur par défaut
print(PrixTTC(PrixHT=100))                  # 120
print(PrixTTC(100))                         # 120

#syntaxe avec une TVA de 10%
print(PrixTTC(PrixHT=100,TVA=0.1))          # 110
print(PrixTTC(100,0.1))                     # 110
print(PrixTTC(100,TVA=0.1))                 # 110
```

```
120.0
120.0
110.00000000000001
110.00000000000001
110.00000000000001
```

Cela permet la plupart du temps d'utiliser la TVA classique de 20 %. Mais si pour un produit vous avez une TVA différente, vous pouvez aussi passer la nouvelle valeur en paramètre.

Passer et retourner un argument de type liste

Dans cette exemple la variable a est **associée** à la liste désignée par la variable b. Ces deux variables sont liées au même objet liste. L'une et l'autre désignent le même objet.

```
def test(a):
    a[0] = 5
    a[1] = 8
    print('a : ',a)
    return a

b = [1 , 2]
c = test(b)
print('b : ',b)
print('c : ',c)
```

```
a :  [5, 8]
b :  [5, 8]
c :  [5, 8]
```

- Dans la fonction test(), lorsque nous modifions les éléments de la liste désignée par la variable a, la liste désignée par la variable b subit aussi ces modifications.
- Lors de l'instruction return a couplée à l'affectation c = test(b), l'opération effectuée est la suivante : c = a
- Ainsi, la variable c est liée à la même liste que la variable a, liste déjà associée à la variable b. Par conséquent, les variables a et c désignent toutes deux le même objet liste

Lors du passage et du retour d'arguments de **type numérique** (int, float) ce lien est rompus contrairement au list.