PYTHON – Tuples

Introduction

Les tuples sont des listes immuables

Les tuples sont des séquences, assez semblables aux listes, sauf **qu'on ne peut modifier un tuple après qu'il ait été créé**.

Cela signifie qu'on définit le contenu d'un tuple (les objets qu'il doit contenir) lors de sa création, mais qu'on ne peut en ajouter ou en retirer par la suite.

Création

Un tuple se définit comme une liste, sauf qu'on utilise comme délimiteur des **parenthèses** au lieu des crochets.

```
tuple_vide = ()
tuple_non_vide = (1,) # est équivalent à ci-dessous
tuple_non_vide = 1,
tuple_avec_plusieurs_valeurs = (1, 2, 5)
```

Rappel : À la différence des listes, les tuples, une fois créés, ne peuvent être modifiés : on ne peut plus y ajouter d'objet ou en retirer.

Une petite subtilité ici : si on veut créer un tuple contenant un unique élément, on doit quand même mettre une virgule après celui-ci. Sinon, Python va automatiquement supprimer les parenthèses et on se retrouvera avec une variable lambda et non une variable contenant un tuple.

Accéder aux éléments d'un tuple

Vous pouvez accéder aux éléments d'un tuple en vous référant au numéro d'index, entre crochets

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

L'indexation négative signifie qu'à partir de la fin, -1 correspond au dernier élément, -2 correspond à l'avant-dernier élément, etc.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Plage d'index

Vous pouvez spécifier une plage d'index en précisant où commencer et où terminer la plage.

Lorsque vous spécifiez une plage, la valeur de retour sera un nouveau tuple avec les éléments spécifiés.

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

NB : La recherche commencera à l'index 2 (inclus) et se terminera à l'index 5 (non inclus).

Spécifiez des index négatifs si vous voulez lancer la recherche à partir de la fin de la ligne :

Cet exemple renvoie les éléments de l'index -4 (inclus) à l'index -1 (exclu)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

Modifier les éléments d'un tuple

Une fois qu'un tuple est créé, vous ne pouvez pas changer ses valeurs. Les tuples sont immuables.

Mais il y a une solution de contournement. Vous pouvez convertir le tuple en une liste, changer la liste, et convertir la liste en un tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

```
('apple', 'kiwi', 'cherry')
```

Boucler dans un tuple

Vous pouvez boucler sur les éléments d'un tuple en utilisant la boucle *for*.

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

Vérifier si un élément existe dans un tuple

Pour déterminer si un élément spécifié est présent dans un tuple, utilisez le mot-clé *in* :

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
      print("Oui, 'apple' est dans le tuple")
```

```
Oui, 'apple' est dans le tuple
```

Longueur d'un tuple

Pour déterminer le nombre d'éléments d'un tuple, utilisez la méthode *len()* :

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Joindre 2 tuples

Pour joindre deux tuples ou plus, vous pouvez utiliser l'opérateur + :

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Supprimer un tuple

Les tuples ne sont pas modifiables, vous ne pouvez donc pas les supprimer, mais vous pouvez les supprimer complètement :

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple)
```

```
Traceback (most recent call last):
  File "C:\Users\ronan\Desktop\toto.py", line 3, in <module>
    print(thistuple) #this will raise an error because the tuple no longer exists
NameError: name 'thistuple' is not defined
```

cela va générer une erreur car le tuple n'existe plus

A quoi servent les tuples ?

Il est assez rare que l'on travaille directement sur des tuples.

Mais il peut être utile de travailler sur des données sans pouvoir les modifier.

Voici quelques cas où nous avons utilisé des tuples sans le savoir.

1- Affectation multiple

Parfois, les tuples ne sont pas entourés de parenthèses, même s'il s'agit quand même de tuples.

Ainsi, on peut utiliser la notation suivante :

```
>>> a, b = 3, 4
>>> a
3
>>> b
4
>>>
```

En fait, cela revient à :

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

Autre exemple :

```
>>> adresse = 'monty@python.org'
>>> nom_utilisateur, domaine = adresse.split('@')
 >>> nom_utilisateur
'monty'
>>> domaine
'python.org'
```

La valeur de retour de split est une liste de deux éléments ; le premier élément est assigné à nom_utilisateur et le second à domaine.

2- Permutation de variables

On a également utilisé les tuples pour permuter deux variables

```
>>> a, b = b, a
>>>
```

3- Une fonction renvoyant plusieurs valeurs

Nous ne l'avons pas vu jusqu'ici mais une fonction peut renvoyer deux valeurs ou même plus :

```
def decomposer(entier, divise_par):
    """Cette fonction retourne la partie entière et le reste de entier / divise_par"""
    p_e = entier // divise_par
    reste = entier % divise_par
    return p_e, reste
```

Et on peut ensuite capturer la partie entière et le reste dans un tuple, au retour de la fonction :

```
retour = decomposer(20, 3)
print(retour)
```

```
(6,2)
```

Ou utiliser l'affectation de tuple pour stocker les éléments séparément :

```
retour1,retour2 = decomposer(20, 3)
print(retour1)
print(retour2)
```

```
6
2
```