IV.2- Les tuples

En Python, les tuples sont similaires aux listes, mais ils présentent une différence essentielle : les tuples sont **immuables**, ce qui signifie que leurs éléments ne peuvent pas être modifiés une fois qu'ils sont créés. Les tuples sont utiles lorsque vous souhaitez garantir que les données restent constantes tout au long du programme.

Ci-dessous un guide complet sur les tuples en Python, incluant des explications, des exemples et des résultats.

2.1. Création de tuples

Les tuples sont créés en plaçant des éléments entre parenthèses **()** séparés par des virgules.

 # Création de différents types de tuples

```
empty_tuple = ()        # Tuple vide

single_item_tuple = (42,)          # Tuple à un seul élément (remarquez la virgule)

numbers_tuple = (1, 2, 3, 4, 5)

mixed_tuple = (1, "apple", 3.14)

print(empty_tuple)

print(single_item_tuple)

print(numbers_tuple)

print(mixed_tuple)
```

Sortie :

```
()
(42,)
(1, 2, 3, 4, 5)
(1, 'apple', 3.14)
```

Explication :

- Un tuple vide est créé en utilisant des parenthèses vides ().
- Pour créer un tuple à un seul élément, une virgule finale est nécessaire (ex : (42,)).
- Les tuples peuvent contenir des éléments de différents types (entiers, chaînes de caractères, nombres à virgule flottante, etc.).

2.2. Accès aux éléments d'un tuple

Vous pouvez accéder aux éléments d'un tuple en utilisant l'indexation. Python utilise une indexation basée sur zéro, ce qui signifie que le premier élément a l'indice 0.

Exemple :

Code Python

```
fruits = ("apple", "banana", "cherry")

print(fruits[0])   # Premier élément
```

```
print(fruits[1])   # Deuxième élément
print(fruits[-1])  # Dernier élément (indexation négative)
```

Sortie (résulat de l'exécution):

```
apple
banana
cherry
```

Explication :

- L'indice 0 permet d'accéder au premier élément (« apple »).
- L'indice -1 permet d'accéder au dernier élément (« cherry ») en utilisant l'indexation négative.

2.3. Découpage des tuples (Tuple Slicing)

Comme pour les listes, Le **découpage des tuples en Python** (appelé aussi *slicing*) permet d'extraire une partie d'un tuple en utilisant des indices.

La syntaxe est comme suit :

```
tuple[start:stop:step]
```

tels que :

- **start** : indice de début (inclus)
- **stop** : indice de fin (exclu)
- **step** : pas (optionnel)

Exemple 1

```
numbers = (10, 20, 30, 40, 50)
print(numbers[1:4])
```

Résultat :

```
(20, 30, 40)
```

✓ On commence à l'indice 1 (20)
✓ On s'arrête avant l'indice 4

Cas courants

✓ Prendre les premiers éléments

```
print(numbers[:3])
```

Résultat :

```
(10, 20, 30)
```

✓ Prendre les derniers éléments

print(numbers[2:])

Résultat :

(30, 40, 50)

✓ Copier tout le tuple

print(numbers[:])

✓ Utiliser un pas (step)

print(numbers[::2])

Résultat :

(10, 30, 50)

Prend un élément sur deux

✓ Inverser un tuple

print(numbers[::-1])

Résultat :

(50, 40, 30, 20, 10)

✓ Indices négatifs

Les indices négatifs commencent depuis la fin :

print(numbers[-3:-1])

Résultat :

(30, 40)

Exemple 2 :

numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

print(numbers[2:5]) # Éléments de l'indice 2 à 4

print(numbers[:3]) # Éléments du début jusqu'à l'indice 2

print(numbers[5:]) # Éléments de l'indice 5 jusqu'à la fin

print(numbers[::2]) # Un élément sur deux

Sortie :

(2, 3, 4)

(0, 1, 2)

(5, 6, 7, 8, 9)

(0, 2, 4, 6, 8)

Explication :

- numbers[2:5] : extrait les éléments de l'indice 2 à 4 (l'indice 5 est exclu).
- numbers[::2] : extrait un élément sur deux.

2.4. Modification des tuples

Les tuples sont immuables, ce qui signifie que leurs éléments ne peuvent pas être modifiés après leur création.

Cependant, vous pouvez créer un nouveau tuple à partir de l'ancien.

Exemple :

```
fruits = ("apple", "banana", "cherry")

# La tentative de modification d'un tuple par exemple la commande fruits[0] = "pear"  provoquera une erreur.

# fruits[0] = "pear"  # Décommenter cette ligne lèvera une erreur TypeError

# Création d'un nouveau tuple par concaténation

new_fruits = ("pear",) + fruits[1:]

print(new_fruits)
```

Sortie :

```
('pear', 'banana', 'cherry')
```

Explication :

- Puisque les tuples sont immuables, vous ne pouvez pas modifier directement leur contenu. Cependant, vous pouvez créer un nouveau tuple en concaténant des parties du tuple original.

2.5. Concaténation et répétition des tuples

Vous pouvez concaténer plusieurs tuples en utilisant l'opérateur + et répéter des tuples en utilisant l'opérateur *.

Exemple :

```
tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

# Concaténation

concatenated_tuple = tuple1 + tuple2

print("Concaténé :", concatenated_tuple)

# Répétition

repeated_tuple = tuple1 * 2

print("Répété :", repeated_tuple)
```

Sortie :

```
Concaténé : (1, 2, 3, 4, 5, 6)
```

Répété : (1, 2, 3, 1, 2, 3)

Explication :

- L'opérateur + concatène tuple1 et tuple2.
- L'opérateur * répète les éléments de tuple1 deux fois.

2.6. Vérification de l'appartenance dans les tuples

Vous pouvez utiliser le mot-clé « **in** » pour vérifier si un élément spécifique existe dans un tuple.

Exemple :

```
fruits = ("apple", "banana", "cherry")


if "banana" in fruits:
   print("Banana est dans le tuple.")
else:
   print("Banana n'est pas dans le tuple.")
```

Sortie :

Banana est dans le tuple.

Explication :

- L'opérateur « **in** » vérifie si l'élément « banana » existe dans le tuple fruits.

2.7. Affectation multiple (Tuple Unpacking)

Le **déballage de tuple** consiste à attribuer simultanément les valeurs d'un tuple à plusieurs variables en une seule ligne.

Exemple :

```
person = ("John", 25, "Engineer")
# Déballage du tuple dans des variables
name, age, profession = person
print("Nom :", name)
print("Âge :", age)
print("Profession :", profession)
```

Sortie :

Nom : John

Âge : 25

Profession : Engineer

Explication :

- Le déballage de tuple affecte chaque élément du tuple person à une variable correspondante (name, age et profession).

2.8. Imbrication des tuples

Les tuples peuvent contenir d'autres tuples (tuples imbriqués), ce qui permet de créer des structures de données complexes.

Exemple :

```
nested_tuple = ((1, 2, 3), ("apple", "banana"), (True, False))


# Accès aux éléments dans des tuples imbriqués
print(nested_tuple[0])       # Premier tuple
print(nested_tuple[1][1])    # Deuxième élément du deuxième tuple
print(nested_tuple[2][0])    # Premier élément du troisième tuple
```

Sortie :

```
(1, 2, 3)
banana
True
```

Explication :

- Les tuples peuvent contenir d'autres tuples, et vous pouvez accéder à leurs éléments en utilisant plusieurs indices.

2.9. Méthodes des tuples

Bien que les tuples soient immuables, ils fournissent certaines méthodes intégrées utiles.

2 .9.1. count()

La méthode count() renvoie le nombre d'occurrences d'un élément spécifié dans le tuple.

Exemple :

```
numbers = (1, 2, 3, 1, 4, 1, 5)

count_of_ones = numbers.count(1)

print("Nombre de 1 :", count_of_ones)
```

Sortie :

```
Nombre de 1 : 3
```

Explication :

- La méthode count() renvoie le nombre de fois que la valeur 1 apparaît dans le tuple.

2.9.2. index()

La méthode index() renvoie l'indice de la première occurrence d'un élément spécifié dans le tuple.

Exemple :

```
fruits = ("apple", "banana", "cherry", "banana")


index_of_banana = fruits.index("banana")


print("Première occurrence de 'banana' :", index_of_banana)
```

Sortie :

Première occurrence de 'banana' : 1

Explication :

- La méthode index() renvoie l'indice de la première occurrence de la valeur « banana ».

2.10. Tuples vs Listes

Bien que les tuples et les listes soient similaires, il existe quelques différences importantes :

- **Immutabilité :** Les tuples sont immuables, ce qui signifie que vous ne pouvez pas les modifier après leur création. Les listes, en revanche, sont modifiables.
- **Performance :** Les tuples sont plus rapides que les listes lorsque vous avez seulement besoin de lire les données.
- **Utilisation :** Les tuples sont généralement utilisés lorsque les données ne doivent pas être modifiées.

Exemple :

```
# Tuple (immuable)

numbers_tuple = (1, 2, 3)

# numbers_tuple[0] = 100  # Décommenter cette ligne provoquera une erreur TypeError


# Liste (modifiable)

numbers_list = [1, 2, 3]

numbers_list[0] = 100  # La liste peut être modifiée


print("Liste modifiée :", numbers_list)
```

Sortie :

Liste modifiée : [100, 2, 3]

Explication :

- Le tuple numbers_tuple ne peut pas être modifié, tandis que la liste numbers_list peut être mise à jour.

2.11. Longueur d'un tuple

Vous pouvez trouver le nombre d'éléments dans un tuple en utilisant la fonction len().

Exemple :

```
fruits = ("apple", "banana", "cherry")

length_of_tuple = len(fruits)

print("Nombre d'éléments dans le tuple :", length_of_tuple)
```

Sortie :

```
Nombre d'éléments dans le tuple : 3
```

Explication :

- len() renvoie le nombre d'éléments dans le tuple.

2.12. Conversion entre listes et tuples

Vous pouvez facilement convertir des tuples en listes et inversement en utilisant les fonctions list() et tuple().

Exemple :

```
# Conversion d'un tuple en liste

fruits_tuple = ("apple", "banana", "cherry")

fruits_list = list(fruits_tuple)

print("Liste :", fruits_list)

# Conversion d'une liste en tuple

fruits_tuple_again = tuple(fruits_list)

print("Tuple :", fruits_tuple_again)
```

Sortie :

```
Liste : ['apple', 'banana', 'cherry']

Tuple : ('apple', 'banana', 'cherry')
```

Explication :

- La fonction list() convertit un tuple en liste, et la fonction tuple() convertit une liste en tuple.

Conclusion

Les tuples en Python sont des collections ordonnées et immuables d'éléments, utiles lorsque vous devez stocker des données qui ne doivent pas être modifiées.

Bien qu'ils présentent de nombreuses similitudes avec les listes, les tuples offrent un accès plus rapide et l'avantage de l'immutabilité, ce qui les rend idéaux pour des données en lecture seule.