1 Introduction

Dans le chapitre précédent, nous avons étudié les listes qui permettent de stocker, à l'instar de tableaux, différentes données afin de pouvoir les utiliser ultérieurement, voire les modifier ou les supprimer.

Dans certains cas, il peut être intéressant d'avoir une structure de données qui puisse stocker plusieurs éléments simultanément comme dans un tableau, mais qui ne puisse pas être modifié par la suite afin, par exemple, d'éviter certaines erreurs de manipulation.

En `Python`, ce type de structure est appelé **tuple** (ou **n-uplets** en référence à une structure mathématique analogue), et est de type `tuple`. Par construction, il est très proche du type `list` et est non modifiable.

Le but de ce chapitre est de comprendre et manipuler ce nouveau type de donnée.

2 Le type `tuple`

2.1 Définition des tuples

Définition 2.1 Un *tuple* est une collection ordonnée (ou séquence) d'éléments encadrée par des parenthèses, les éléments étant séparés par des virgules. Ce sont des données **non modifiables**.

Exemple 2.2 `(1,2, -5, 10)` est un tuple composé de quatre éléments.

Remarque 2.3 Comme pour les listes, les tuples peuvent être composées d'éléments de types différents, et même de listes ou de tuples.

Définition 2.4 (Longueur d'un tuple) On appelle *longueur d'un tuple*, le nombre d'éléments qui le compose.
Cette longueur peut-être obtenue à l'aide de la fonction `len`.

Exemple 2.5 L'instruction `len((1,2, -5, 10))` renvoie `4`.

Comme pour les listes, on peut définir un tuple par extension, par compréhension ou par transtypage.

Définition 2.6 Il existe plusieurs manières de définir une liste :

- **Tuple vide :** il est défini par l'instruction `()` ou par l'instruction `tuple()`.

- **Tuple par extension :** il est défini en indiquant directement entre parenthèses les éléments du tuple.
  Exemple : le tuple `(1, 2, -5, 10)` est un tuple défini par extension.
  Exemple : le tuple `(1,)` est un tuple défini par extension (ne pas oublier la virgule).

- **Tuple par compréhension :** il est défini en indiquant entre parenthèses la manière dont le tuple est construit. On utilise pour cela la syntaxe suivante :

```
tuple(fonction(i) for i in range(valeur_initiale, valeur_finale + 1) if condition(i))
```

  où `fonction(i)` est une fonction de paramètre `i` et où `condition(i)` est une condition que l'on impose sur `i`. Cette condition est optionnelle.
  Exemple : les deux tuples suivants sont des tuples définies par compréhension :
  - `tuple(i for i in range(5))` correspond au tuple `(0, 1, 2, 3, 4)`
  - `tuple(i**2 for i in range(5) if i%2 == 1)` correspond au tuple `(1, 9)`

- **Tuple par transtypage :** il est défini en appliquant la fonction `tuple` sur certains types de données (par exemple des chaînes de caractères).
  Exemple : l'instruction `tuple('Python')` donne le tuple `('P', 'y', 't', 'h', 'o', 'n')`

2.2 Indexation

⋆ Les éléments d'un tuple étant ordonnés, ils sont repérés à l'aide d'un indice de la façon suivante :

- le premier élément du tuple a pour indice `0` ;
- le deuxième élément du tuple a pour indice `1` ;
- le troisième élément du tuple a pour indice `2` ;
- etc.

Ainsi, **l'indice d'un élément dans un tuple est égal à la `position - 1` de l'élément dans ce tuple.**.

⋆ L'élément d'indice `j` d'un tuple `T` est défini par l'instruction `T[j]`.
Exemple : on considère le tuple `T = (1, 2, -5, 10)`. Alors :

- `T[0]` correspond à `1` (élément d'indice `0`);
- `T[1]` correspond à `2` (élément d'indice `1`);
- `T[2]` correspond à `-5` (élément d'indice `2`);
- `T[3]` correspond à `10` (élément d'indice `3`).

Remarque : si on veut accéder à un indice qui n'existe pas, on déclenche une erreur de type

`IndexError: tuple index out of range`,

c'est-à-dire que l'on a voulu utiliser un indice en dehors de la plage autorisée (de `0` jusqu'à `len - 1`).

Remarque 2.7 Les tuples n'étant pas modifiables, des instructions du type `T = (1, 2, -5, 10) ; T[1] = 3` renverront des erreurs du type

`'tuple' object does not support item assignment`

2.3 Opérateurs

Les deux opérateurs + et ⋆ sont compatibles avec les tuples :

⋆ L'opérateur + est un **opérateur de concaténation**, c'est-à-dire qu'il permet de rassembler deux tuples en un seul.
Exemple : l'instruction `(1, 2) + (3,)` renvoie le tuple `(1, 2, 3)`.

⋆ L'opérateur ⋆ est un **opérateur de duplication**, c'est-à-dire qu'il renvoie un tuple dont les éléments sont ceux d'un tuple qui ont été dupliqués plusieurs fois.
Exemple : l'instruction `(0,)*3` renvoie le tuple `(0, 0, 0)` constituée de l'élément `0` dupliqué trois fois.

2.4 Tests

⋆ On peut tester si deux tuples sont égaux ou différents grâce aux opérateurs de comparaison == et `!=`
Exemples :

- L'instruction `(1, 2, 3) == (1, 2, 3)` renvoie `True` (les deux tuples sont égaux).
- L'instruction `(1, 2, 3) != (1, 2, 3)` renvoie donc `False`.
- L'instruction `(1, 2, 3) == (1, 3, 2)` renvoie `False` (les deux tuples ne sont pas égaux).
- L'instruction `(1, 2, 3) != (1, 3, 2)` renvoie donc `True`.

⋆ On peut également tester si un élément appartient à un tuple ou non grâce aux opérateurs `in` et `not in`
Exemples :

- L'instruction `5 in (1, 2, 4, 5, 8)` renvoie `True` (`5` est un élément du tuple).
- L'instruction `5 not in (1, 2, 4, 5, 8)` renvoie donc `False`.
- L'instruction `6 in (1, 2, 4, 5, 8)` renvoie `False` (`6` n'est pas un élément du tuple).
- L'instruction `6 not in (1, 2, 4, 5, 8)` renvoie donc `True`.

3 Parcours de tuples

Un tuple est une structure qui peut être parcourue afin d'obtenir des informations sur chacun de ses éléments.

Comme pour les listes, il existe deux types de parcours : le **parcours par indice** et le **parcours par élément**.

Définition 3.1 On donne ci-dessous la syntaxe Python pour chacun des deux parcours pour un tuple T :

Parcours par indice :

```
for i in range(len(T)):
    bloc d'instructions
```

Parcours par élément :

```
for elem in T:
    bloc d'instructions
```

Remarque 3.2 Un tuple n'étant pas modifiable, ces deux parcours ne permettent d'accéder aux éléments qu'en lecture.

Exemple 3.3 Le tableau ci-dessous donne le code Python permettant d'afficher tous les éléments du tuple T définie à la première ligne suivant ces deux types de parcours :

Parcours par indice :

```
T = (1, 5, 3, 8, 9, 10, 7, 8)
for i in range(len(T)):
    print(T[i])
```

Parcours par élément :

```
T = (1, 5, 3, 8, 9, 10, 7, 8)
for elem in T:
    print(elem)
```

L'exécution de ces deux codes donne le même affichage :

```
1
5
3
8
9
10
7
8
```

4 Fonctions et méthodes agissant sur les tuples

Il existe un grand nombre de fonctions et méthodes agissant sur les tuples. On ne les présente pas toutes ici. La liste complète peut être obtenue avec l'instruction `help(tuple)`.

4.1 Fonctions

On donne ci-dessous quatre fonctions importantes dans le cadre des tuples numériques, c'est-à-dire à valeurs entières ou flottantes.

- L'instruction `sum(T)` renvoie la somme des éléments du tuple numérique `T`.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `sum(T)` renvoie `9`.

- L'instruction `min(T)` renvoie le plus petit élément du tuple numérique `T`.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `min(T)` renvoie `-1`.

- L'instruction `max(T)` renvoie le plus grand élément du tuple numérique `T`.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `max(T)` renvoie `5`.

- L'instruction `sorted(T)` renvoie une **liste** dont les éléments sont ceux du tuple numérique `T` rangés par ordre croissant.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `sorted(T)` renvoie la liste `[-1, 1, 1, 3, 5]`.

4.2 Méthodes

Un tuple n'étant pas modifiable, seules les méthodes `count` et `index` sont autorisées.

- L'instruction `T.count(elem)` renvoie le nombre d'occurences de l'élément `elem` dans le tuple `T`.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `T.count(1)` renvoie `2`.

- L'instruction `T.index(elem)` renvoie l'index de la première occurence de l'élément `elem` dans le tuple `T` si `elem` appartient à `T`, et renvoie une erreur sinon.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `T.index(5)` renvoie `1`.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `T.index(1)` renvoie `0`.
  Exemple : si `T = (1, 5, -1, 3, 1)`, alors `T.index(6)` renvoie l'erreur

```
ValueError: tuple.index(x): x not in tuple
```

5 Lien avec les fonctions

Lorsque dans une fonction on renvoie n valeurs ($n \geqslant 2$) en utilisant une instruction de la forme `return val1, val2,..,valn`, la fonction renvoie en fait un tuple. Les données renvoyées peuvent être ensuite récupérées soit à l'aide d'une affectation parallèle, soit dans une variable de type tuple.

Exemple 5.1 La fonction `somme_produit(a,b)` ci-dessous. renvoie deux résultats : la somme et le produit. On peut les récupérer sous la forme d'un tuple ou bien dans deux variables distinctes `somme` et `produit`.

```
def somme_produit(a,b):
    return a + b, a * b

res = somme_produit(5,6)
somme, produit = somme_produit(5,6)
```

- La variable `res` contient le tuple `(11,30)`
- La variable `somme` contient l'entier `11`
- La variable `produit` contient l'entier `30`

6 Exercices

Exercice 6.1 (Manipulations basiques) Compléter le Notebook *NSI Première Partie 1 Chapitre 7 Tuples 1/2*.

Exercice 6.2 (Un peu de réflexion) Compléter le Notebook *NSI Première Partie 1 Chapitre 7 Tuples 2/2*.

Exercice 6.3 (QCM)

1. On considère le tuple `T = (1, 5, -1, 2)`. Parmi les instructions suivantes, lesquelles sont vraies ?
   (a) `len(T)` renvoie `3`
   (b) `T[len(T)]` renvoie `2`
   (c) `T[3]` renvoie `2`
   (d) `T[1]` renvoie `1`

2. On considère le tuple `T = (1, 3, 0, -1)`. Qu'obtient-on à l'issue de l'instruction `T[3]=2` ?
   (a) rien
   (b) une erreur
   (c) le tuple `(1, 3, 0, 2)`
   (d) le tuple `(1, 2, 0, -1)`

3. On considère le tuple `T = (2, 3, -5, 0)`. Parmi les instructions suivantes, lesquelles sont vraies ?
   (a) `max(T)` renvoie le nombre d'éléments de `T`
   (b) `min(T)` renvoie `0`
   (c) `T.append(3)` modifie `T` en le tuple `(2, 3, -5, 0, 3)`
   (d) `sum(T)` renvoie `0`

4. On considère la fonction suivante :

```
def perimetre_aire(longueur, largeur):
    perimetre = 2*(longueur + largeur)
    aire = longueur * largeur
    return perimetre, aire
```

Parmi les affirmations suivantes, lesquelles sont vraies à l'issue de l'instruction `rectangle = perimetre_aire(3, 2)` ?

(a) la variable `rectangle` contient la liste `[10, 6]`
(b) la variable `rectangle` contient le tuple `(10, 6)`
(c) la variable `rectangle` contient le tuple `(6, 10)`
(d) la variable `rectangle[1]` contient la valeur de l'aire du rectangle de longueur 3 et de largeur 2

5. On considère les instructions suivantes :

```
def cherche_indice(elem, T):
    """
    elem est une donnée de type int, float, list, tuple ou str
    T est un tuple
    """
    if elem in T:
        return T.index(elem)
    return -1
```

Parmi les affirmations suivantes, lesquelles sont vraies ?

(a) l'instruction `cherche_indice(3, (1, 5, 4, 4))` renvoie une erreur
(b) l'instruction `cherche_indice(3, (1, 5, 4, 4))` renvoie `-1`
(c) l'instruction `cherche_indice(4, (1, 5, 4, 4))` renvoie `-1`
(d) l'instruction `cherche_indice(4, (1, 5, 4, 4))` renvoie `2`