CHAPITRE 14

Types construits - les tuples

Les types entier, réel, booléen, caractère sont des types simples. Les p-uplets, les tableaux et les dictionnaires sont des types construits à partir de types simples. À l'identique de l'objet « voiture », qui est composé de différents objets : « volant », « roue », « siège » ...

Déf. 1

Un p-uplet est une suite ordonnée d'éléments. Chacun de ces éléments peut être de n'importe quel type.

En Python, un p-uplet est de type `tuple`.

I. Création d'un tuple

Un tuple est une collection d'éléments séparés par une virgule.

Exemple 1

```
>>> t = (1, 2, 3, 'bonjour', 8.5)
```

t est un tuple composé de 5 éléments :

- 3 éléments sont des entiers,
- 1 élément est une chaîne de caractères,
- 1 élément est un réel.

Activité 1.
Créer un tuple nommé *lundi* et contenant 2 éléments : le nom du jour et le nombre d'heures de cours de cette journée.

Un tuple peut aussi être composé de tuples :

```
>>>    t = ((1, 2, 'moi'), (3,4, 'toi'), (5,6, 'nous'))
```

Activité 2.
Créer un tuple emboîté nommé *semaine* et indiquant, pour chaque jour de la semaine, le nombre d'heures de cours.

II. Propriétés d'un tuple

Propriété 1 (non-mutabilité d'un tuple)

Un tuple est non-mutable (on dit aussi immuable), c'est à dire que l'on ne peut ni modifier l'un de ses éléments, ni supprimer l'un de ses éléments, ni ajouter un nouvel élément.

Exemple 2 (Erreur générée par la tentative d'affectation à un tuple)

```
>>> t = (1, 2, 'moi')
>>>    t[0]
1
>>> t[0] = 2
    Traceback (most recent call last):
    File "<ipython-input-3-0c6697a06bab>", line 1, in <module>
    t[0] = 2
    TypeError: 'tuple' object does not support item assignment
```

III. Indexation des éléments d'un tuple

Si un tuple est non-vide, ses n éléments sont indexés par les entiers 0, 1, 2,. . . , n - 1. On accède à chacun de ses éléments en utilisant des crochets et l'indice de l'élément : `tuple[indice]`.

Attention

Les indices commencent à 0.

Exemple 3 (Indexation d'un tuple simple)

```
>>> t = (1, 2, 3, 'bonjour', 8.5)
>>> t[0]
1
>>> t[3]
'bonjour'
```

Exemple 4 (Indexation d'un tuple emboîté)

```
>>> t = ((1, 2, 'moi'), (3, 4, 'toi'), (5, 6, 'nous'), (7, 8, 'vous'))
>>> t[0][0]
1
>>> t[0][2]
'moi'
>>> t[2][2]
'nous'
```

Activité 3.
Pour le tuple *semaine*, indiquer le(s) code(s) Python renvoyant la valeur 0.

Pour connaître la longueur d'un tuple, on pourra utiliser l'instruction Python `len()`

Exemple 5 (Longueur d'un tuple simple)

```
>>> t = (1, 2, 3, 'bonjour', 8.5)
>>> len(t)
5
```

Exemple 6 (Longueur d'un tuple emboîté)

```
>>> t = ((1, 2, 'moi'), (3, 4, 'toi'), (5, 6, 'nous'), (7, 8, 'vous'))
>>> len(t)
4
>>> len(t[1])
3
```

Activité 4.

1. Quelle est la longueur du tuple *lundi* ?
2. Quelle est la longueur du tuple *semaine* ?

À retenir

On peut affecter à une variable un élément d'un tuple. On pourra utiliser l'instruction Python `variable = tuple[indice]`.

Exemple 7 (Affectation d'un tuple)

```
>>> t = ((1, 2, 'moi'), (3, 4, 'toi'), (5, 6, 'nous'), (7, 8, 'vous'))
>>> sujet = t[0][2]
>>> sujet
'moi'
```

Activité 5.

1. Combien d'éléments du tuple *semaine* renvoient un nombre d'heures de cours inférieur à 5 ?
2. Affecter à une variable nommée *tranquille* les jours de la semaine où le nombre d'heures de cours est inférieur à 5.
3. De quel type sera cette nouvelle variable ?

IV. Opérations sur les tuples

Les opérateurs + et * s'utilisent comme avec les chaînes de caractères. Ce sont des opérateurs de concaténation. Le résultat est un tuple (un nouveau...).

Exemple 8 (Concaténation de tuples)

```
>>> t1 = ((1,2,'moi'),(3,4,'toi'))
>>> t2 = (0, 10, 20)
>>> t1 + t2
((1, 2, 'moi'), (3, 4, 'toi'), 0, 10, 20)
>>> 2*t2
(0, 10, 20, 0, 10, 20)
```

L'appartenance à un tuple se teste, comme pour les chaînes de caractères, avec l'opérateur `in`.

On pourra utiliser l'instruction Python `element in tuple`.

Exemple 9 (Appartenance à un tuple)

```
>>> t1 = (0, 10, 20)
>>> 20 in t1
True
>>> 4 in t1
False
```

V. Itération sur un tuple

Un tuple est un **itérable**. On peut donc itérer sur les éléments d'un tuple. On peut réaliser soit un parcours par index, soit un parcours par élément.

V. 1. Parcours par index

Pour parcourir un tuple, on procède en deux étapes :

- ▷ on définit une nouvelle variable définie par la longueur de la chaîne ;
- ▷ on utilise une boucle bornée (boucle for) pour l'index variant entre 0 et la longueur de la chaîne (exclue).

Exemple 10 (Affichage du contenu d'un tuple)

```
        t = (1, 2, 3, 'bonjour', 8.5)

        for i in range (0, len(t), 1):
        print(t[i])
```

V. 2. Parcours par élément

Pour parcourir un tuple, on peut aussi parcourir le tuple élément par élément.

Exemple 11 (Affichage du contenu d'un tuple)

```
t = (1, 2, 3, 'bonjour', 8.5)

for element in t:
    print(element)
```

Je retiens

- ▷ Un tuple est une collection d'éléments placés entre parenthèses, et séparés par une virgule.
- ▷ Un tuple est non mutable (on dit aussi immuable).
- ▷ L'instruction `len()` permet de connaître le nombre d'éléments (la longueur) d'un tuple.
- ▷ Les n éléments d'un tuple sont indexés par les entiers $0, 1, 2, \ldots, n-1$. On accède à l'élément d'incide i par l'instruction `tuple[i]`.
- ▷ Un tuple est itérable, c'est à dire que l'on peut itérer sur chacun de ses éléments. On peut faire un parcours par index ou un parcours par élément.
- ▷ Les opérateurs + et * s'utilisent comme avec les chaînes de caractères. Ce sont des opérateurs de concaténation. Le résultat est un tuple.
- ▷ L'appartenance d'un élément se teste avec le mot-clé `in`.

VI. Exercices

Exercice 1 (QCM).

1. Soit la définition suivante :

```
1 t = ('foo', 'bar', 'baz')
```

Laquelle des propositions suivantes permet de remplacer l'élément 'bar' par 'qux' ?

- ☐ t[1] = 'qux'
- ☐ t(1) = 'qux'
- ☐ t[1 :1] = 'qux'
- ☐ Ce n'est pas possible de faire ce remplacement !

2. Laquelle des propositions suivantes permet de créer un p-uplet nommé p contenant la chaîne 'foo' ?

- ☐ p = puplet('foo')
- ☐ p = ('foo',)
- ☐ p = ('foo')
- ☐ On ne peut pas créer un tuple comportant 1 seul élément !

3. Qu'affiche le programme suivant ?

```
1 tuple=(1,5,7,9,10,15)
2 print("tuple[5] = ",tuple[5])
```

- ☐ tuple[5] = 5
- ☐ tuple[5] = '5'
- ☐ tuple[5] = 15
- ☐ tuple[5] = 10

Exercice 2.

1. Quelle est la longueur du tuple (1, 2, (2, 4), 5) ?
2. Si t = ('a','b','c'), que renvoie l'instruction t[1] ?
3. Que renvoie l'expression (1,"lundi") + (3,"manteau",5.0) ?
4. Que renvoie l'expression 3*(1, "poele") ?
5. Si t = (3, 5, 1), quelle valeur vaut le booléen t[1] == 3 ?
6. Quelle est la valeur retournée par l'instruction (0, 2) + (0, 3) == (0, 5) ?

Exercice 3.

1. Écrire une fonction *mem*() prenant *tup* et *donnee* en argument, et qui renvoie True si x est un élément du tuple.
   Par exemple, si t prend la valeur `('peche', ('pomme', 'poire'), 12.5, -5)`, alors `mem(t, 'peche')` renvoie `True`.
2. Modifier cette fonction pour qu'elle renvoie aussi l'index.
   Par exemple, si t prend la valeur `('peche', ('pomme', 'poire'), 12.5, -5)`, alors`mem(t, 'peche')` renvoie `(True, 0)`.

Exercice 4.

Écrire un fonction *compte*() prenant *tup* et *element* en argument, et qui renvoie le nombre de fois où cet élément est présent dans le tuple.
Par exemple, si t prend la valeur `(3, "manteau", 5.0, "pêche", 3)`, alors`compte(t, 3)` renvoie `2`.

Exercice 5.

Écrire un fonction *division_euclidienne*() prenant en argument deux nombres $n1$ et $n2$ et qui renvoie un tuple constitué du quotient et du reste de la division euclidienne de n1 par n2.
Par exemple, `division_euclidienne(5, 2)` renvoie `(2, 1)`.