Tuple

September 22, 2019

1 Les tuples

Les variables de type simple sont les entiers, les nombres flottants, les booléens et les chaînes de caractères.

D'autres types sont nécessaires selon les informations à traiter. On les appellent en Python les types **construits** (et pas compliqués contrairement à ce que l'on pourrait imaginer).

Nous allons cette année en étudier 3 : les N-uplets (ou tuples), les listes (type list) et les dictionnaires en Python (qui sont un cas particulier du type set). Ces 3 types ne se retrouvent pas forcément dans les autres langages. Par exemple, en Javascript, on retrouve les listes, appelées tableaux (array).

On va s'interesser dans un premier aux tuples.

1.1 C'est quoi ce nom ?

Quand on veut la position d'un point dans un plan, on a besoin de sa position horizontale x et de sa position verticale x : on obtient un couple de valeurs ordonnées $(x\ ,\ y)$ que l'on appelle aussi 2-uplet.

Un triplet sera alors un ensemble de trois valeurs $(x\ ,\ y\ ,\ z)$ qui pourrait correspondre à la position d'un point dans un espace à trois dimension : on a ainsi un 3-uplet.

Un quadruplet ou 4-uplet pourrait par exemple correspondre à *('nom' , 'prenom' , 'classe' , 'adresse mail')*. On se rend compte dans ce dernier exemple de l'importance de l'ordre :

('Dupont' , 'Archibald', '1NSI' , 'moulinsard@couteaux.fr').

De manière générale, **un N-uplets est une série de N valeurs ordonnées**.

1.2 Création d'un tuple.

Pour créer un N-uplter, Il suffit d'écrire N valeurs séparées par une virgule :

```
In [1]: t=1,3,'Dupont',5
        type(t)

Out[1]: tuple

In [2]: t

Out[2]: (1, 3, 'Dupont', 5)
```

1.3 Utilisation.

On récupère les valeurs de la même manière qu'avec les caractères dans une chaîne.

```
In [3]: a=('Dupont' , 'Archibald', '1NSI' , 'moulinsard@couteaux.fr')
        a[1]

Out[3]: 'Archibald'

In [4]: a[1:]

Out[4]: ('Archibald', '1NSI', 'moulinsard@couteaux.fr')

In [5]: a[-1]

Out[5]: 'moulinsard@couteaux.fr'
```

Avec **len(...)**, on récupère le nombre d'élèments du tuple :

```
In [6]: len(a)

Out[6]: 4
```

Comme pour les chaînes de caractères, les élèments ne sont pas modifiables :

```
In [7]: a[0]='Dupond'


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-7-6b8c05f76831> in <module>()
    ----> 1 a[0]='Dupond'


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

1.4 Opérations sur les tuples.

On retrouve les mêmes opérations vues sur les chaînes de caractères.

Concaténation : on peut ajouter des tuples.

```
In [10]: a=(0,'un',2,('trois',3))
         b=(4,'cinq')
         c=a+b
         c

Out[10]: (0, 'un', 2, ('trois', 3), 4, 'cinq')
```

On peut multiplier :

```
In [11]: d=4*b
         d

Out[11]: (4, 'cinq', 4, 'cinq', 4, 'cinq', 4, 'cinq')
```

On peut tester l'appartenance d'un élèment à un tuple avec l'instruction **in** :

```
In [12]: 'un' in a

Out[12]: True

In [13]: 'deux' in a

Out[13]: False
```

1.5 Tuples et fonctions.

Une fonction en général renvoie une valeur. Cette valeur peut être un nombre, une chaîne de caractères ... ou plusieurs. Dans ce dernier cas, la fonction renvoie un tuple.

```
In [14]: a=('Dupont' , 'Archibald', '1NSI' , 'moulinsard@couteaux.fr')
         def presentation(t):
             return t[1],t[0]
         c=presentation(a)
         type(c)

Out[14]: tuple

In [15]: d="Je m'appelle "+c[0]+" "+c[1]
         d

Out[15]: "Je m'appelle Archibald Dupont"
```

On aurait pu écrire aussi :

```
In [16]: d="Je m'appelle "+presentation(a)[0]+" "+presentation(a)[1]+" encore !"
         d

Out[16]: "Je m'appelle Archibald Dupont encore !"
```