# Tuplas&Diccionarios

October 15, 2017

## 1 Tuplas

Las tuplas son un nuevo tipo de dato en Python. Las tuplas son listas inmutables, es decir, no las podemos modificar una vez que estas han sido creadas. Por lo tanto, a una tupla no podemos añadir, eliminar o mover elementos. Si que permiten extraer una porción, el resultado de esto es una nueva tupla. También permiten comprobar si un determinado elemento se encuentra dentro de una tupla.

Las tuplas respecto a las listas tienen una serie de ventajas: la primera ventaja respecto a las listas es que es más rápida en tiempo de ejecución, ocupan menos espacio en memoria que una lista, nos van a permitir formatear cadenas y se pueden usar como clave en un diccionario.

A la hora de crear una tupla las creamos con la sintáxis entre paréntesis, aunque estos son opcionales de usar. A la hora de recorrer una tupla seguimos las misma sintáxis que en las listas.

```
In [1]: #Nos creamos nuestra primera tupla
        mitupla = ('Madrid', 'Getafe', 'Alcorcón')
        print(mitupla)
```

```
('Madrid', 'Getafe', 'Alcorcón')
```

```
In [2]: #Nos creamos tupla sin hacer uso de los paréntesis
        mitupla2 = 'Madrid', 'Getafe', 'Alcorcón'
        print(mitupla2)
```

```
('Madrid', 'Getafe', 'Alcorcón')
```

```
In [3]: #Accedemos al primer elemento de la tupla
        print(mitupla[0])
```

```
Madrid
```

Tenemos dos métodos que permiten convertir tuplas en listas y listas en tuplas. Para convertir una tupla en una lista contamos con el método list().

```
In [7]: #Convertimos una tupla en lista
        lista_CM = list(mitupla)
        print(lista_CM)
```

```
['Madrid', 'Getafe', 'Alcorcón']
```

Para realizar el proceso inverso, es decir, convertir una lista en tupla contamos con el método tuple()

```
In [8]: #Convertimos una lista en tupla
        tupla_CM = tuple(lista_CM)
        print(tupla_CM)
```

```
('Madrid', 'Getafe', 'Alcorcón')
```

Si queremos comprobar si un elemento se encuentra presente en una tupla, disponemos del método in. Este método nos devuelve True en caso de que el elemento se encuentre en el interior de nuestra lista y False en caso constrario.

```
In [9]: #Vemos si el barrio de Vayecas está en nuestra lista
        print('Vayecas' in tupla_CM)
```

```
False
```

Las tuplas también disponen del método count() que nos permiter averiguar el número de veces que aparace un elemento en una tupla.

```
In [10]: #Vemos el número de veces que aparece la ciudad de Madrid en nuestra tupla
         print(tupla_CM.count('Madrid'))
```

```
1
```

El método len() nos permite averiguar la longitud de una tupla.

```
In [12]: print(len(tupla_CM))
```

```
3
```

También podemos crear tuplas unitarias, es decir, tuplas de un solo elemento, pero a la hora de declarar este tipo de tuplas denemos de indicar el elemento seguido de una coma.

```
In [16]: #Nos creamos una tupla unitaria haciendo uso de los paréntesis
         tupla_unitaria = ('Madrid',)
         print(len(tupla_unitaria))
         #Nos creamos una tupla unitaria sin hacer uso de los paréntesis
         tupla_unitaria = 'Madrid',
         print(len(tupla_unitaria))
```

```
1
1
```

Python dispone de lo que se conoce como desempaquetado de tupla, es decir, podemos asignarle los valores de una tupla a varias variables de una tacada.

```
In [18]: #Desempaquetamos nuestras tuplas
         tupla_capitales = ('Roma', 'París', 'Amsterdam')
         Italia, Francia, Holanda = tupla_capitales
         print(Italia)
         print(Francia)
         print(Holanda)
```

```
Roma
París
Amsterdam
```

## 2 Diccionarios

Los diccionarios son estructuras de datos que nos permiten almacenar valores de diferente tipo, incluso listas y otros diccionarios. La principal característica de los diccionarios es que los datos se almacenan asociados a una clave de tal forma que se crea una asociación de tipo clave:valor. El orden en el que se almacene un valor en python da exactamente igual ya que a cada valor se le asigna una clave.

La sintáxis de un diccionario es hacer uso de las llaves y dentro de estas la estructura clava:valor, podemos añadir tantos elementos como queramos separados por coma.

```
In [25]: #Nos creamos un diccionario
         paises_dic = {'España' : 'madrid', 'Francia': 'París', 'Italia': 'Roma'}
         print(paises_dic)
```

```
{'Italia': 'Roma', 'Francia': 'París', 'España': 'madrid'}
```

Podemos pensar que teniendo las listas los diccionarios no son útiles, para ver que esto no es cierto veamos el siguiente ejemplo. Supongamos que tenemos dos listas una que almacena países y otra que almacena capitales.

```
In [20]: #Nos creamos dos listas
         list_paises = ['España', 'Francia', 'Italia']
         list_capitales = ['Madrid', 'París', 'Roma']
```

Ahora supongamos que queremos acceder a la capital de Francia, ahora para hacer esto debemos acceder al índice en el cual se encuentra Francia y con ese índice acceder a lista de capitales.

```
In [21]: #Accedemos a la capita de Francia
         print(list_capitales[list_paises.index('Francia')])
```

```
París
```

En el caso de un diccionario acceder a esta información sería mucho más sencillo, puesto que la información se almacena con el formato clave:valor, bastaría con acceder a la clave (Francia) y obtendríamos la capital de dicho país.

```
In [22]: #Accedemos al elemento que corresponde a la clave Francia
         print(paises_dic['Francia'])
```

```
París
```

Para agregar elementos a un diccionario basta con hacer uso de la sintáxis diccionario[nueva_clave] = valor.

```
In [26]: #Agregamos un nuevo elemento al diccionario
         paises_dic['Portugal'] = 'Lisboa'
         print(paises_dic)
```

```
{'Italia': 'Roma', 'Portugal': 'Lisboa', 'Francia': 'París', 'España': 'madrid'}
```

Podemos sobreescribir el valor de una determinada clave, sin más que hacer uso de la sintáxis diccionario[clave] = nuevo_valor.

```
In [27]: #Sobreescribimos el nombre de la capita de España
         paises_dic['España'] = 'Madrid'
         print(paises_dic)
```

```
{'Italia': 'Roma', 'Portugal': 'Lisboa', 'Francia': 'París', 'España': 'Madrid'}
```

Para eliminar elementos de un diccionario hacemos uso del método del().

```
In [28]: #Eliminamos la información respecto a Italia
         del(paises_dic['Italia'])
         print(paises_dic)
```

```
{'Portugal': 'Lisboa', 'Francia': 'París', 'España': 'Madrid'}
```

Podemos crear diccionarios que alternen distintos tipos de datos tanto para la clave como para el valor.

```
In [29]: #Nos creamos un diccionario donde ahora las claves son numéricas
         DreamTeam_USA = {9: 'Michael Jordan', 8: 'Scottie Pippen', 7: 'Larry Bird', 15: 'Magic
         print(DreamTeam_USA)
```

```
{8: 'Scottie Pippen', 9: 'Michael Jordan', 15: 'Magic Johnson', 11: 'Karl Malone', 7: 'Larry Bir
```

Un diccionario puede almacenar tuplas.

```
In [32]: #Almacenamos dentro de un diccionario una tupla
         MichaelJordan_dic = {'Nombre': 'Michael', 'Equipo': 'Chicago',
                              'anillos': (1991, 1992, 1993, 1996,1997,1988)}
         print(MichaelJordan_dic)
```

```
{'anillos': (1991, 1992, 1993, 1996, 1997, 1988), 'Equipo': 'Chicago', 'Nombre': 'Michael'}
```

También podemos guardar un diccionario dentro de otro diccionario.

```
In [42]: #Almacenamos dentro de un diccionario otro diccionario
         MichaelJordan_dic = {'Nombre': 'Michael', 'Equipo': 'Chicago',
                              'anillos':
                              {'Temporadas': (1991, 1992, 1993, 1996,1997,1988)}}
         print(MichaelJordan_dic)
         #Accedemos a temporadas
         print(MichaelJordan_dic['anillos']['Temporadas'])
```

```
{'anillos': {'Temporadas': (1991, 1992, 1993, 1996, 1997, 1988)}, 'Equipo': 'Chicago', 'Nombre':
(1991, 1992, 1993, 1996, 1997, 1988)
```

Los diccionarios disponen del método keys que nos permite obtener las claves de un diccionario.

```
In [36]: print(MichaelJordan_dic.keys())
```

```
dict_keys(['anillos', 'Equipo', 'Nombre'])
```

También disponen del métod values que nos permite obtener los valores de un diccionario.

```
In [38]: print(MichaelJordan_dic.values())
```

```
dict_values([{'Temporadas': (1991, 1992, 1993, 1996, 1997, 1988)}, 'Chicago', 'Michael'])
```

El método len nos permite obtener la longitud de un diccionario.

```
In [39]: print(len(MichaelJordan_dic))
```

```
3
```

El método in nos permite saber si una determinada clave se encuentra en nuestro diccionario.

```
In [40]: print('Nombre' in MichaelJordan_dic)
```

```
True
```