Diccionarios

Otro tipo de dato útil incluído en Python es el **diccionario**. En Python, un diccionario es una colección **no ordenada** de pares **clave - valor** donde la clave y el valor son objetos Python.

- Las claves son únicas (no existen dos elementos del diccionario con la misma clave).
- El acceso a los elementos de un diccionario se realiza a través de la clave.
- En otros lenguajes se les conoce como *tablas hash*.

Crear un diccionario

De la misma forma que con listas, es posible definir un diccionario directamente con los miembros que va a contener, o bien inicializar el diccionario vacío y luego agregar los valores.

- Los diccionarios se crean utilizando llaves **{ }**.
- Los elementos del diccionario son de la forma **clave : valor** y cada uno de los elementos se separan por comas.

```
dic = { }     # diccionario vacío
bool(dic)
```

```
False
```

La función **bool** es una función predefinida en Python Built-in Functions. Cuando se invoca con un diccionario como argumento, devuelve `True`, si el diccionario tiene elementos y `False` en caso contrario.

```
dic =  {1:'Lunes', 2:'Martes', 3:'Miercoles' }  # diccionario con tres elementos
dic
```

```
{1: 'Lunes', 2: 'Martes', 3: 'Miercoles'}
```

En el siguiente ejemplo mostramos cómo se crea un diccionario con dos elementos; las claves son las cadenas 'Hola' y 'Comer'. Los valores asociados a dichas claves las listas ['Hi', 'Hello'] y ['eat'] respectivamente.

```
d  = {'Hola'  : ['Hi', 'Hello'],
      'Comer' : ['eat'] }
len(d)                              # el tamaño del diccionario es 2
```

```
2
```

- La **clave** de un diccionario puede ser cualquier variable de tipo inmutable:
  - cadenas,
  - enteros,
  - tuplas (con valores inmutables en sus miembros), etc.
- Los **valores** de un diccionario pueden ser de cualquier tipo: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

```
d1 = {'Lunes' : 1, 'Martes' : 2 , 'Finde' : (6,7) }
d1
```

```
{'Finde': (6, 7), 'Lunes': 1, 'Martes': 2}
```

Otra forma de crear un diccionario es declararlo vacío y luego insertar los valores, se declara el diccionario vacío, y luego se asignan valores directamente a las claves.

```
d2 = { }
d2['Juan'] = 609922565
d2['Ana']  = 691252580
d2['Luis'] = 642589569
d2
```

```
{'Ana': 691252580, 'Juan': 609922565, 'Luis': 642589569}
```

Acceso a los elementos de un diccionario

Para acceder al valor asociado a una determinada clave, utilizamos la notación de corchetes **[ ]** usada en listas, pero utilizando la clave elegida en lugar del índice.

```
d2 ['Luis']
```

```
642589569
```

Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave. Si se asigna un valor a una clave ya existente, se reemplaza el valor anterior.

```
d2 ['Luis'] = 99991
d2 ['Ana'] = ['938941523', '609585962']
d2
```

```
{'Ana': ['938941523', '609585962'], 'Juan': 609922565, 'Luis': 99991}
```

Si se intenta acceder a una clave que no está en el diccionario, el acceso falla.

```
d2 ['Carlota']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-67adeed0b7c7> in <module>()
----> 1 d2 ['Carlota']

KeyError: 'Carlota'
```

Para evitar estos errores, podemos usar la función **in**, que comprueba si un elemento está en el diccionario o utilizar el método **get**, que devuelve el valor **None** si la clave no está en el diccionario.

```
'Carlota' in d2
```

```
False
```

```
'Luis' in d2
```

```
True
```

```
m = d2.get('Carlota')
type(m)
```

```
NoneType
```

Para eliminar elementos de un diccionario se utiliza el método **pop**:

```
d2
```

```
{'Ana': ['938941523', '609585962'], 'Juan': 609922565, 'Luis': 99991}
```

```
e = d2.pop('Ana')
e
```

```
['938941523', '609585962']
```

```
d2
```

```
{'Juan': 609922565, 'Luis': 99991}
```

```
d2.pop('Carlota')
d2
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-15-f090c8fc80b9> in <module>()
----> 1 d2.pop('Carlota')
      2 d2

KeyError: 'Carlota'
```

Métodos: keys, values, update

El método **keys** devuelve una lista con las claves del diccionario. El método **values** devuelve una lista con los valores del diccionario.

```
d2
```

```
{'Juan': 609922565, 'Luis': 99991}
```

```
d2.keys()
```

```
dict_keys(['Juan', 'Luis'])
```

```
d2.values()
```

```
dict_values([609922565, 99991])
```

Para fusionar diccionarios utilizamos el método **update**.

```
d1
```

```
{'Finde': (6, 7), 'Lunes': 1, 'Martes': 2}
```

```
d2
```

```
{'Juan': 609922565, 'Luis': 99991}
```

```
d1.update(d2)
d1
```

```
{'Finde': (6, 7), 'Juan': 609922565, 'Luis': 99991, 'Lunes': 1, 'Martes': 2}
```

La función *zip*

La función **zip** permite crear *un objeto iterable* a partir de los elementos de otras secuencias.

```
# Python 3.6
s1 = [1, 2, 3 , 4]
s2 = ['primavera', 'verano', 'otoño', 'invierno' ]
z = zip(s1, s2)
z
```

```
<zip at 0x15f8db35088>
```

```
list(z)    # creamos una lista a partir del objeto iterable 'z'
```

```
[(1, 'primavera'), (2, 'verano'), (3, 'otoño'), (4, 'invierno')]
```

También podemos hacer **unzip** para separar una lista de tuplas en varias secuencias.

```
# Python 3.6
s1 = [1, 2, 3 , 4]
s2 = ['primavera', 'verano', 'otoño', 'invierno' ]
z = zip(s1, s2)
c1, c2 = zip(*z)
c1, c2
```

```
((1, 2, 3, 4), ('primavera', 'verano', 'otoño', 'invierno'))
```

A partir de secuencias también es posible crear diccionarios.

```
map = dict(zip(s1,c2))
map
```

```
{1: 'primavera', 2: 'verano', 3: 'otoño', 4: 'invierno'}
```

Referencias

- Tutorial de Python. Por Guido Van Rossum