Diccionarios

Diccionarios

Las **listas** son estructuras de datos extremadamente flexibles, que nos permiten acceder directamente a cualquier elemento mediante su posición. Sin embargo, cuando necesitamos acceder a un dato y no sabemos dónde se encuentra, primero debemos averiguar su posición para luego acceder a él. El método de búsqueda funciona inspeccionando una por una las posiciones de la lista hasta detectar el objeto buscado.

Serán muchas las situaciones en las que necesitemos acceder a información basándonos en su contenido en lugar de su posición. Para facilitar este tipo de procesos, Python provee una estructura de datos ideal a tales efectos: los **diccionarios**.

Un diccionario (*dict*) es una estructura que permite mapear una **clave** con un **valor**. De este modo, cuando necesitemos acceder a un valor en particular, no indicaremos su posición, sino su clave.

Un diccionario ubica en memoria los elementos de acuerdo al resultado de aplicar una función *hash* a su clave. Por eso se lo denomina **estructura hasheable**. En otros lenguajes de programación, a los diccionarios también se los llama arreglos asociativos, mapas o tablas de hash.

La ventaja de utilizar un diccionario por sobre una lista, es que el tiempo de búsqueda tiende a permanecer constante sin importar el tamaño del mismo. Por el contrario, realizar búsquedas en una lista tiende a ser menos eficiente a medida que el tamaño de la lista aumenta.

En Python, los diccionarios se escriben como una colección de pares (`clave:valor`), separados por coma y entre llaves. Por ejemplo:

```
edades = {
    'juan': 30,
    'nicolas': 22,
    'ana': 20
}
```

Para acceder a un valor, lo hacemos mediante la clave. Por ejemplo, `edades['juan']` nos devolverá 30.

La clave del diccionario puede ser cualquier tipo de dato **inmutable**: números, strings, booleanos, y tuplas, y se admite heterogeneidad (no necesariamente tienen que ser todas las claves del mismo tipo). Los valores pueden ser cualquier tipo de dato, mutable o inmutable.

Métodos de la clase diccionario

Método	Descripción
clear()	Elimina todos los elementos
pop(clave)	Elimina un par clave/valor, buscando por su clave
popitem()	Elimina el último item insertado (Python 3.7 en adelante)
fromkeys(secuencia, valor)	Crea un diccionario con los items de la secuencia como claves, y con el valor especificado (o valor None si no se especifica)
items()	Devuelve una lista de tuplas con la forma (clave, valor)
keys()	Devuelve una lista con las claves
values()	Devuelve una lista con los valores
update(dict)	Actualiza un diccionario con las claves de otro diccionario. Agrega las que no existen, y para las que existen actualiza su valor.

Uno de los usos frecuentes de los diccionarios es cuando queremos guardar varios datos sobre diferentes entidades. Por ejemplo, información de alumnos de una materia (nombre, notas, condición, etc), o información de productos (código, descripción, precio, etc).

Ejemplo:

```
productos = {
    '10001': ('Aceite', 10),
    '10012': ('Harina', 6),
    '10310': ('Fideos', 8),
}
```

Ordenamiento

Normalmente, cuando iteramos un diccionario usando un bucle, las claves aparecen en el orden que fueron guardadas. Pero si queremos ver la información ordenada por la clave (por ejemplo si las claves son nombres, en

orden alfabético, o si son números, de menor a mayor), se puede armar una lista de claves, ordenarla, y usar esa lista ordenada para acceder a los datos del diccionario. Esta lista ordenada también se conoce como **índice**.

Por ejemplo, este programa pide datos de personas, los almacena en un diccionario cuya clave es el DNI de la persona, luego ordena las claves (según el número de DNI) y usa la lista ordenada de claves para iterar por el diccionario.

```
dicci = {}
print('Datos de Clientes, * para terminar')
dni = input('Dni: ')
while dni != '*':
    nom = input('Nombre: ')
    ape = input('Apellido: ')
    edad = int(input('Edad: '))
    while edad not in range(18, 130):
        edad = int(input('Edad: '))
        dicci[dni] = [nom, ape, edad]
        dni = input('Dni: ')
dicciOrden = sorted(dicci)
for pers in dicciOrden:
print(pers, dicci[pers][0], dicci[pers][1], dicci[pers][2])
```

💡 Cuando se pasa como parámetro de una función un diccionario (o una lista), no se crea una copia del objeto original, sino que se pasa una **referencia** al objeto. Esto significa que cualquier cambio que la función haga se verá reflejado en nuestro objeto original, y también significa que la función no necesita devolver nada. Si no queremos compartir el objeto original, necesitamos hacer una copia (antes de llamar a la función, o dentro de la función), y la función deberá devolver esa estructura copiada.