El futuro digital es de todos

MinTIC

Diccionarios

Rogerio Orlando Beltrán Castro

Universidad Industrial de Santander

MisiónTIC2022

Diccionarios

La clase dict de Python es un tipo mapa que asocia claves a valores. A diferencia de los tipos secuenciales (list, tuple, range o str), que son indexados por un índice numérico, los diccionarios son indexados por claves. Estas claves siempre deben ser de un tipo inmutable, concretamente un tipo hashable.

🎯 NOTA: Un objeto es hashable si tiene un valor de hash que no cambia durante todo su ciclo de vida. En principio, los objetos que son instancias de clases definidas por el usuario son hashables. También lo son la mayoría de tipos inmutables definidos por Python (int, float o str).

Diccionarios

Piensa siempre en un diccionario como un contenedor de pares clave: valor, en el que la clave puede ser de cualquier tipo hashable y es única en el diccionario que la contiene. Generalmente, se suelen usar como claves los tipos int y str aunque, como te he dicho, cualquier tipo hashable puede ser una clave.

Las principales operaciones que se suelen realizar con diccionarios son almacenar un valor asociado a una clave y recuperar un valor a partir de una clave. Esta es la esencia de los diccionarios y es aquí donde son realmente importantes. En un diccionario, el acceso a un elemento a partir de una clave es una operación realmente rápida, eficaz y que consume pocos recursos si lo comparamos con cómo lo haríamos con otros tipos de datos.

Diccionarios

Otras características a resaltar de los diccionarios:

- Es un tipo mutable, es decir, su contenido se puede modificar después de haber sido creado.
- Es un tipo ordenado. Preserva el orden en que se insertan los pares clave: valor.

Cómo crear un diccionario

En Python hay varias formas de crear un diccionario. Las veremos todas a continuación.

La más simple es encerrar una secuencia de pares clave: valor separados por comas entre llaves {}

```
d = {1: 'hola', 89: 'Pythonista', 'a': 'b', 'c': 27}
```

En el diccionario anterior, los enteros 1 y 89 y las cadenas 'a' y 'c' son las claves. Como ves, se pueden mezclar claves y valores de distinto tipo sin problema.

Cómo crear un diccionario

Para crear un diccionario vacío, simplemente asigna a una variable el valor {}.

También se puede usar el constructor de la clase dict() de varias maneras:

- Sin parámetros. Esto creará un diccionario vacío.
- Con pares clave: valor encerrados entre llaves.
- Con argumentos con nombre. El nombre del argumento será la clave en el diccionario. En este caso, las claves solo pueden ser identificadores válidos y mantienen el orden en el que se indican. No se podría, por ejemplo, tener números enteros como claves.
- Pasando un iterable. En este caso, cada elemento del iterable debe ser también un iterable con solo dos elementos. El primero se toma como clave del diccionario y el segundo como valor. Si la clave aparece varias veces, el valor que prevalece es el último.

Cómo crear un diccionario

Veamos un ejemplo con todo lo anterior. Vamos a crear el mismo diccionario de todos los modos que te he explicado:

```
# 1. Pares clave: valor encerrados entre llaves
print('# 1. Pares clave: valor encerrados entre llaves')
d = {'uno': 1, 'dos': 2, 'tres': 3}
print(d)
{'uno': 1, 'dos': 2, 'tres': 3}
# 2. Argumentos con nombre
print('# 2. Argumentos con nombre')
d2 = dict(uno=1, dos=2, tres=3)
print(d2)
# 3. Pares clave: valor encerrados entre llaves
print('# 3. Pares clave: valor encerrados entre llaves')
d3 = dict({'uno': 1, 'dos': 2, 'tres': 3})
print(d3)
{'uno': 1, 'dos': 2, 'tres': 3}
# 4. Iterable que contiene iterables con dos elementos
print('# 4. Iterable que contiene iterables con dos elementos')
d4 = dict([('uno', 1), ('dos', 2), ('tres', 3)])
print(d4)
# 5. Diccionario vacío
print('# 5. Diccionario vacío')
d5 = {}
print(d5)
{}
# 6. Diccionario vacío usando el constructor
print('# 6. Diccionario vacío usando el constructor')
d6 = dict()
print(d6)
```

Cómo acceder a los elementos de un diccionario en Python

Acceder a un elemento de un diccionario es una de las principales operaciones por las que existe este tipo de dato. El acceso a un valor se realiza mediante indexación de la clave. Para ello, simplemente encierra entre corchetes la clave del elemento d[clave]. En caso de que la clave no exista, se lanzará la excepción KeyError.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}

print(d['dos'])
```

Cómo acceder a los elementos de un diccionario en Python

La clase dict también ofrece el método get(clave[, valor por defecto]). Este método devuelve el valor correspondiente a la clave clave. En caso de que la clave no exista no lanza ningún error, sino que devuelve el segundo argumento valor por defecto. Si no se proporciona este argumento, se devuelve el valor None.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
print(d.get('uno'))

# Devuelve 4 como valor por defecto si no encuentra la clave
print(d.get('cuatro', 4))

# Devuelve None como valor por defecto si no encuentra la clave
a = d.get('cuatro')
print(a)
print(type(a))
```

for dict Python – Recorrer un diccionario

Hay varias formas de recorrer los elementos de un diccionario: recorrer solo las claves, solo los valores o recorrer a la vez las claves y los valores.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
for e in d:
    print(e)
# Recorrer las claves del diccionario
for k in d.keys():
    print(k)
# Recorrer los valores del diccionario
for v in d.values():
    print(v)
# Recorrer los pares clave valor
for i in d.items():
    print(i)
```

Bucle for en diccionarios

Un caso es especial de bucle for se da al recorrer los elementos de un diccionario. Dado que un diccionario está compuesto por pares clave/valor, hay distintas formas de iterar sobre ellas.

```
print('1 – Recorrer las claves del diccionario.')
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for k in valores:
    print(k)
print('2 – Iterar sobre los valores del diccionario')
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for v in valores.values():
    print(v)
print('3 – Iterar a la vez sobre la clave y el valor de cada uno de los elementos del diccionario.')
valores = {'A': 4, 'E': 3, 'I': 1, 'O': 0}
for k, v in valores.items():
    print('k=', k, ', v=', v)
```

Añadir elementos a un diccionario en Python

Como te decía, la clase dict es mutable, por lo que se pueden añadir, modificar y/o eliminar elementos después de haber creado un objeto de este tipo.

Para añadir un nuevo elemento a un diccionario existente, se usa el operador de asignación =. A la izquierda del operador aparece el objeto diccionario con la nueva clave entre corchetes [] y a la derecha el valor que se asocia a dicha clave.

```
d = {'uno': 1, 'dos': 2}

print(d)

# Añade un nuevo elemento al diccionario

d['tres'] = 3

print(d)
```

🎯 NOTA: Si la clave ya existe en el diccionario, se actualiza su valor.

Añadir elementos a un diccionario en Python

También existe el método setdefault(clave[, valor]). Este método devuelve el valor de la clave si ya existe y, en caso contrario, le asigna el valor que se pasa como segundo argumento. Si no se especifica este segundo argumento, por defecto es None.

```
d = {'uno': 1, 'dos': 2}
d.setdefault('uno', 1.0)
print(d)
d.setdefault('tres', 3)
print(d)
d.setdefault('cuatro')
print(d)
```

Modificar elementos de un diccionario

En el apartado anterior hemos visto que para actualizar el valor asociado a una clave, simplemente se asigna un nuevo valor a dicha clave del diccionario.

```
d = {'uno': 1, 'dos': 2}
print(d)
d['uno'] = 1.0
print(d)
```

Eliminar un elemento de un diccionario en Python

En Python existen diversos modos de eliminar un elemento de un diccionario. Son los siguientes:

- pop(clave [, valor por defecto]): Si la clave está en el diccionario, elimina el elemento y devuelve su valor; si no, devuelve el valor por defecto. Si no se proporciona el valor por defecto y la clave no está en el diccionario, se lanza la excepción KeyError.
- popitem(): Elimina el último par clave: valor del diccionario y lo devuelve. Si el diccionario está vacío se lanza la excepción KeyError. (NOTA: En versiones anteriores a Python 3.7, se elimina/devuelve un par aleatorio, no se garantiza que sea el último).
- del d[clave]: Elimina el par clave: valor. Si no existe la clave, se lanza la excepción KeyError.
- clear(): Borra todos los pares clave: valor del diccionario.

Eliminar un elemento de un diccionario en Python

En Python existen diversos modos de eliminar un elemento de un diccionario. Son los siguientes:

```
d = {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}

print('# Elimina un elemento con pop()')

d.pop('uno')

print(d)

print('# Trata de eliminar una clave con pop() que no existe')

d.pop(6)

print('# Elimina un elemento con popitem()')

d.popitem()

print(d)

print('# Elimina un elemento con del')

del d['tres']

print(d)

print('# Trata de eliminar una clave con del que no existe')

del d['seis']

print('# Borra todos los elementos del diccionario')

d.clear()

print(d)
```

Número de elementos (len) de un diccionario en Python

Al igual que sucede con otros tipos contenedores, se puede usar la función de Python len() para obtener el número de elementos de un diccionario.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}

print(len(d))
```

Comprobar si un elemento está en un diccionario en Python

Al operar con diccionarios, se puede usar el operador de pertenencia in para comprobar si una clave está contenida, o no, en un diccionario. Esto resulta útil, por ejemplo, para asegurarnos de que una clave existe antes de intentar eliminarla.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}

print(len(d))

print('uno' in d)

print(1 in d)

print(1 not in d)

# Intenta eliminar la clave 1 si existe

if 1 in d:

    del d[1]

print(d)
```

Comparar si dos diccionarios son iguales

En Python se puede utilizar el operador de igualdad == para comparar si dos diccionarios son iguales. Dos diccionarios son iguales si contienen el mismo conjunto de pares clave: valor, independientemente del orden que tengan.

Otro tipo de comparaciones entre diccionarios no están permitidas. Si se intenta, el intérprete lanzará la excepción TypeError.

```
d1 = {'uno': 1, 'dos': 2}
d2 = {'dos': 2, 'uno': 1}
d3 = {'uno': 1}
print(d1 == d2)
print(d1 == d3)
print(d1 > d2)
```

Diccionarios anidados en Python

Un diccionario puede contener un valor de cualquier tipo, entre ellos, otro diccionario. Este hecho se conoce como diccionarios anidados.

Para acceder al valor de una de las claves de un diccionario interno, se usa el operador de indexación anidada [clave1][clave2]...

Veámoslo con un ejemplo:

```
d = {'d1': {'k1': 1, 'k2': 2}, 'd2': {'k1': 3, 'k4': 4}}

print(d['d1']['k1'])

print(d['d2']['k1'])

print(d['d2']['k4'])

print(d['d3']['k4'])
```

Obtener una lista con las claves de un diccionario

En ocasiones, es necesario tener almacenado en una lista las claves de un diccionario. Para ello, simplemente pasa el diccionario como argumento del constructor list(). Esto devolverá las claves del diccionario en una lista.

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
print(list(d))
```

Objetos vista de un diccionario

La clase dict implementa tres métodos muy particulares, dado que devuelven un tipo de dato, iterable, conocido como objetos vista. Estos objetos ofrecen una vista de las claves y valores contenidos en el diccionario y si el diccionario se modifica, dichos objetos se actualizan al instante.

Los métodos son los siguientes:

- keys(): Devuelve una vista de las claves del diccionario.
- values(): Devuelve una vista de los valores del diccionario.
- items(): Devuelve una vista de pares (clave, valor) del diccionario.

Objetos vista de un diccionario

```
d = {'uno': 1, 'dos': 2, 'tres': 3}
# d.keys() es diferente a list(d), aunque ambos
# contengan las claves del diccionario
# d.keys() es de tipo dict_keys y list(d) es de tipo list
v = d.keys()
print(type(v))
print(v)
l = list(d)
print(type(l))
print(l)
v = d.values()
print(type(v))
print(v)
v = d.items()
print(type(v))
print(v)
```

Listado de métodos de la clase dict

Termino este tutorial mostrándote el listado de los principales métodos de la clase dict. Algunos de ellos ya los hemos visto en las secciones anteriores:

Método	Descripción
`clear()`	Elimina todos los elementos del diccionario.
`copy()`	Devuelve una copia poco profunda del diccionario.
`get(clave[, valor])`	Devuelve el valor de la `clave`. Si no existe, devuelve el valor `valor` si se indica y si no, `None`.
`items()`	Devuelve una vista de los pares *clave: valor* del diccionario.
`keys()`	Devuelve una vista de las claves del diccionario.
`pop(clave[, valor])`	Devuelve el valor del elemento cuya clave es `clave` y elimina el elemento del diccionario. Si la clave no se encuentra, devuelve `valor` si se proporciona. Si la clave no se encuentra y no se indica `valor`, lanza la excepción `KeyError`.
`popitem()`	Devuelve un par *(clave, valor)* aleatorio del diccionario. Si el diccionario está vacío, lanza la excepción `KeyError`.
`setdefault(clave[, valor])`	Si la `clave` está en el diccionario, devuelve su valor. Si no lo está, inserta la `clave` con el valor `valor` y lo devuelve (si no se especifica `valor`, por defecto es `None`).
`update(iterable)`	Actualiza el diccionario con los pares *clave: valor* del `iterable`.
`values()`	Devuelve una vista de los valores del diccionario.