Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

11
Tipo set

Índice de contenidos

Introducción

Este tema es la continuación de los anteriores.

Después de haber estudiado los tipos secuencia y el tipo diccionario, en esta ocasión vamos a descubrir los detalles del tipo conjunto.

Tipo set

El tipo `set` es la clase utilizada por el lenguaje para representar los conjuntos. **Un conjunto es una colección desordenada de elementos únicos**, es decir, que no se repiten.

Las principales operaciones que se pueden llevar a cabo con este tipo de datos son comprobar si un elemento pertenece a una colección y la mayoría de operaciones matemáticas sobre conjuntos conocidas (*unión*, *intersección*, *diferencia*, ...).

Cómo crear un conjunto

Para crear un conjunto, basta con encerrar una serie de elementos separados por comas entre llaves `{}`, o bien usar el constructor de la clase `set()` y pasarle como argumento un objeto *iterable* (como una lista, una tupla, una cadena ...). **Los elementos repetidos son eliminados al crear el conjunto.**

```
# Crea un conjunto con una serie
# de elementos entre llaves
>>> c = {1, 3, 2, 9, 3, 1}
>>> c
{1, 2, 3, 9}

# Crea un conjunto a partir de un string
# Los caracteres repetidos se eliminan
>>> a = set('Hola Pythonista')
>>> a
{'a', 'H', 'h', 'y', 'n', 's', 'P', 't',
' ', 'i', 'l', 'o'}

# Crea un conjunto a partir de una lista
>>> unicos = set([3, 5, 6, 1, 5])
>>> unicos
{1, 3, 5, 6}
```

Para crear un conjunto vacío, simplemente llama al constructor `set()` sin parámetros. `{}` **NO** crea un conjunto vacío, sino un diccionario vacío. Usa `set()` si quieres crear un conjunto sin elementos.

NOTA: A un conjunto se pueden añadir elementos de diferente tipo, pero todos deben ser de tipo *hashable*. 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 (por ejemplo: *int*, *float*, *str* o *tuple*).

set vs frozenset

En realidad, en Python existen dos clases para representar conjuntos: `set` y `frozenset`. La principal diferencia es que `set` es mutable, mientras que `frozenset` es inmutable (y por tanto, *hashable*) y su contenido no puede ser modificado una vez que ha sido inicializado.

Para crear un conjunto de tipo `frozenset`, se usa el constructor de la clase `frozenset()`:

```
>>> f = frozenset([3, 5, 6, 1, 5])
>>> f
frozenset({1, 3, 5, 6})
```

El único modo en Python de tener un conjunto de conjuntos es utilizando objetos de tipo frozenset como elementos del propio conjunto.

Cómo acceder a los elementos de un conjunto

Dado que los conjuntos son colecciones desordenadas, en ellos no se guarda la posición en la que son insertados los elementos como ocurre en los tipos `list` o `tuple`. Además, la clases que definen los conjuntos no implementan el método `__getitem__()`, que permite el acceso a un elemento de una colección. Por este motivo no se puede obtener los elementos de un conjunto a través de un índice.

Sin embargo, sí se puede acceder y/o recorrer todos los elementos de un conjunto usando un bucle `for`:

```
>>> mi_conjunto = {1, 3, 2, 9, 3, 1}
>>> for e in mi_conjunto:
...     print(e)
...
1
2
3
9
```

Añadir elementos a un conjunto

Para añadir un elemento a un conjunto se utiliza el método `add()`. También existe el método `update()`, que puede tomar como argumento una *lista*, una *tupla*, un *string*, un *conjunto* o cualquier objeto de tipo *iterable*.

```
>>> mi_conjunto = {1, 3, 2, 9, 3, 1}
>>> mi_conjunto
{1, 2, 3, 9}

# Añade el elemento 7 al conjunto
>>> mi_conjunto.add(7)
>>> mi_conjunto
{1, 2, 3, 7, 9}

# Los elementos repetidos no se añaden
# al conjunto
>>> mi_conjunto.update([5, 3, 4, 6])
>>> mi_conjunto
{1, 2, 3, 4, 5, 6, 7, 9}
```

NOTA: `add()` y `update()` no añaden elementos que ya existen al conjunto.

Eliminar un elemento

La clase `set` ofrece cuatro métodos para eliminar elementos de un conjunto. Son: `discard()`, `remove()`, `pop()` y `clear()`. A continuación, te explico qué hace cada uno de ellos.

- `discard(elemento)` y `remove(elemento)` eliminan `elemento` del conjunto. La única diferencia es que si `elemento` no existe, `discard()` no hace nada mientras que `remove()` lanza la excepción `KeyError`.

- `pop()` es un tanto peculiar. Este método devuelve un elemento aleatorio del conjunto y lo elimina del mismo. Si el conjunto está vacío, lanza la excepción `KeyError`.

- `clear()` elimina todos los elementos contenidos en el conjunto.

```
>>> mi_conjunto = {1, 3, 2, 9, 3, 1, 6,
4, 5}
>>> mi_conjunto
{1, 2, 3, 4, 5, 6, 9}

# Elimina el elemento 1 con remove()
>>> mi_conjunto.remove(1)
>>> mi_conjunto
{2, 3, 4, 5, 6, 9}

# Elimina el elemento 4 con discard()
>>> mi_conjunto.discard(4)
>>> mi_conjunto
{2, 3, 5, 6, 9}

# Trata de eliminar el elemento 7 (no
# existe) con remove()
# Lanza la excepción KeyError
>>> mi_conjunto.remove(7)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 7

# Trata de eliminar el elemento 7 (no
# existe) con discard()
# No hace nada
>>> mi_conjunto.discard(7)
>>> mi_conjunto
{2, 3, 5, 6, 9}
```

```
# Obtiene y elimina un elemento
# aleatorio con pop()
>>> mi_conjunto.pop()
2
>>> mi_conjunto
{3, 5, 6, 9}

# Elimina todos los elementos del
# conjunto
>>> mi_conjunto.clear()
>>> mi_conjunto
set()
```

Número de elementos de un conjunto

Como con cualquier otra colección, puedes usar la función `len()` para obtener el número de elementos contenidos en un conjunto:

```
>>> mi_conjunto = set([1, 2, 5, 3, 1,
5])
>>> len(mi_conjunto)
4
```

Cómo saber si un elemento está en un conjunto

Con los conjuntos también se puede usar el operador de pertenencia `in` para comprobar si un elemento está contenido, o no, en un conjunto:

```
>>> mi_conjunto = set([1, 2, 5, 3])
>>> print(1 in mi_conjunto)
True
>>> print(6 in mi_conjunto)
False
>>> print(2 not in mi_conjunto)
False
```

Operaciones del álgebra de conjuntos

Tal y como te adelanté al comienzo de esta sección, uno de los principales usos del tipo `set` es utilizarlo en operaciones del álgebra de conjuntos: *unión*, *intersección*, *diferencia*, *diferencia simétrica*, ...

A continuación, veremos cómo llevar a cabo estas operaciones en Python.

Unión de conjuntos en Python

La unión de dos conjuntos `A` y `B` es el conjunto `A ∪ B` que contiene todos los elementos de `A` y de `B`.

En Python se utiliza el operador `|` para realizar la unión de dos o más conjuntos.

```
>>> a = {1, 2, 3, 4}
>>> b = {2, 4, 6, 8}
>>> a | b
{1, 2, 3, 4, 6, 8}
```

Intersección de conjuntos en Python

La intersección de dos conjuntos `A` y `B` es el conjunto `A ∩ B` que contiene todos los elementos comunes de `A` y `B`.

En Python se utiliza el operador `&` para realizar la intersección de dos o más conjuntos.

```
>>> a = {1, 2, 3, 4}
>>> b = {2, 4, 6, 8}
>>> a & b
{2, 4}
```

Diferencia de conjuntos en Python

La diferencia entre dos conjuntos `A` y `B` es el conjunto `A \ B` que contiene todos los elementos de `A` que no pertenecen a `B`.

En Python se utiliza el operador `-` para realizar la diferencia de dos o más conjuntos.

```
>>> a = {1, 2, 3, 4}
>>> b = {2, 4, 6, 8}
>>> a - b
{1, 3}
```

Diferencia simétrica de conjuntos en Python

La diferencia simétrica entre dos conjuntos `A` y `B` es el conjunto que contiene los elementos de `A` y `B` que no son comunes.

En Python se utiliza el operador `^` para realizar la diferencia simétrica de dos o más conjuntos.

```
>>> a = {1, 2, 3, 4}
>>> b = {2, 4, 6, 8}
a ^ b
{1, 3, 6, 8}
```

Inclusión de conjuntos en Python

Dado un conjunto `A`, subcolección del conjunto `B` o igual a este, sus elementos son un subconjunto de `B`. Es decir, `A` es un subconjunto de `B` y `B` es un superconjunto de `A`.

En Python se utiliza el operador `<=` para comprobar si un conjunto `A` es subconjunto de `B` y el operador `>=` para comprobar si un conjunto `A` es superconjunto de `B`.

```
>>> a = {1, 2}
>>> b = {1, 2, 3, 4}
>>> a <= b
True
>>> a >= b
False
>>> b >= a
True
>>> a = {1, 2}
>>> b = {1, 2}
>>> a < b  # Ojo al operador < sin el =
False
>>> a <= b
True
```

Conjuntos disjuntos en Python

Dos conjuntos `A` y `B` son disjuntos si no tienen elementos en común, es decir, la intersección de `A` y `B` es el conjunto vacío.

En Python se utiliza el método `isdisjoint()` de la clase `set` para comprobar si un conjunto es disjunto de otro.

```
>>> a = {1, 2}
>>> b = {1, 2, 3, 4}
>>> a.isdisjoint(b)
False

>>> a = {1, 2}
>>> b = {3, 4}
>>> a.isdisjoint(b)
True
```

Igualdad de conjuntos en Python

En Python dos conjuntos son iguales si y solo si todos los elementos de un conjunto están contenidos en el otro. Esto quiere decir que cada uno es un subconjunto del otro.

```
>>> a = {1, 2}
>>> b = {1, 2}
>>> id(a)
4475070656
>>> id(b)
4475072096
>>> a == b
True
```

Métodos para las operaciones del álgebra de conjuntos

Además de los operadores que hemos visto para llevar a cabo las operaciones del álgebra de conjuntos (unión `|`, intersección `&`, etc.), la clase `set` implementa una serie de métodos que también permiten realizar dichas operaciones. En la tabla siguiente te muestro cuáles son:

Método	Descripción
`difference(iterable)`	Devuelve la diferencia del conjunto con el `iterable` como un conjunto nuevo.
`difference_update(iterable)`	Actualiza el conjunto tras realizar la diferencia con el `iterable`.
`intersection(iterable)`	Devuelve la intersección del conjunto con el `iterable` como un conjunto nuevo.

`intersection_update(iterable)`	Actualiza el conjunto tras realizar la intersección con el `iterable`.
`issubset(iterable)`	Devuelve `True` si el conjunto es subconjunto del `iterable`.
`issuperset(iterable)`	Devuelve `True` si el conjunto es superconjunto del `iterable`.
`symmetric_difference(iterable)`	Devuelve la diferencia simétrica del conjunto con el `iterable` como un conjunto nuevo.
`symmetric_difference_update(iterable)`	Actualiza el conjunto tras realizar la diferencia simétrica con el `iterable`.
`union(iterable)`	Devuelve la unión del conjunto con el `iterable` como un conjunto nuevo.
`update(iterable)`	Actualiza el conjunto tras realizar la unión con el `iterable`.

IMPORTANTE: Los operadores |, &, ... toman siempre como operandos objetos de tipo set. Sin embargo, sus respectivas versiones como métodos union(), intersection(), ... toman como argumento cualquier tipo iterable (lista, tupla, conjunto, etc.).