El futuro digital es de todos

MinTIC

Conjuntos

Rogerio Orlando Beltrán Castro

Universidad Industrial de Santander

Conjuntos

Un conjunto es una colección desordenada de valores no repetidos.

Los conjuntos de Python son análogos a los conjuntos matemáticos. El tipo de datos que representa a los conjuntos se llama set.

El tipo set es mutable: una vez que se ha creado un conjunto, puede ser modificado.

Cómo crear conjuntos

Las dos maneras principales de crear un conjunto son:

usar un conjunto literal, entre llaves:

```
colores = {'azul', 'rojo', 'blanco', 'blanco'}
print(colores)
```

Note que el conjunto no incluye elementos repetidos, y que los elementos no quedan en el mismo orden en que fueron agregados.

usar la función set aplicada sobre un iterable:

```
conjunto=set('abracadabra')
print(conjunto)
conjunto=set(range(50, 2000, 400))
print(conjunto)
conjunto=set([(1, 2, 3), (4, 5), (6, 7, 8, 9)])
print(conjunto)
```

El conjunto vacío debe ser creado usando set(), ya que {} representa el diccionario vacío.

Cómo crear conjuntos

Los elementos de un conjunto deben ser inmutables. Por ejemplo, no es posible crear un conjunto de listas, pero sí un conjunto de tuplas:

s = {[2, 4], [6, 1]} ➔Marcar error

print(s)

s = {(2, 4), (6, 1)}

print(s)

Conjuntos

Como un conjunto no es ordenado, no tiene sentido intentar obtener un elemento usando un índice:

```
s = {'a', 'b', 'c'}
```

print(s[0]) ➔Marcar Error

Sin embargo, sí es posible iterar sobre un conjunto usando un ciclo for:

```
s = {'a', 'b', 'c'}
for i in s:
    print(i)
```

Operaciones sobre conjuntos

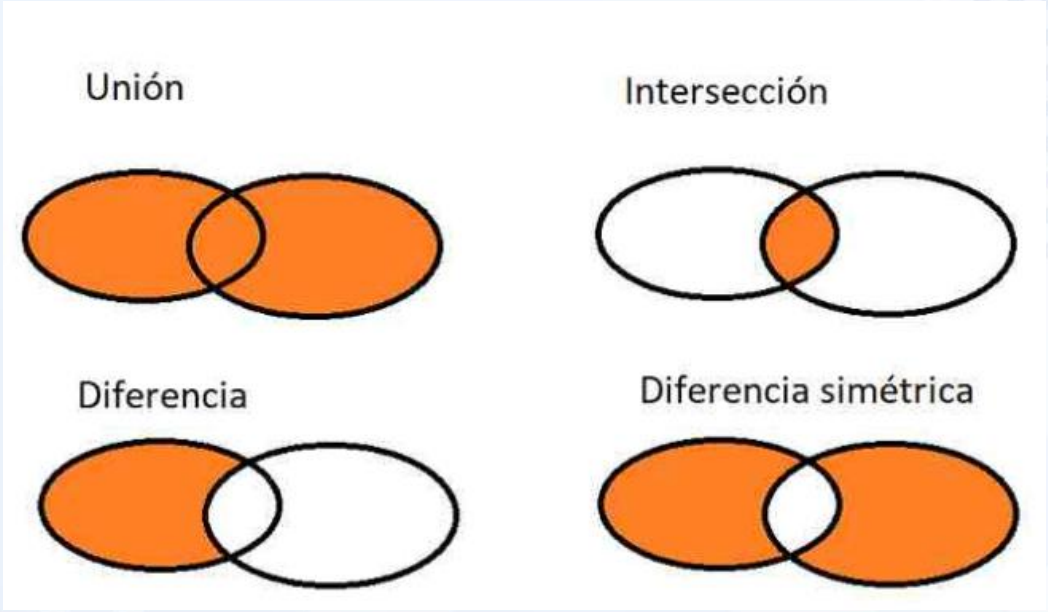
len(s) entrega el número de elementos del conjunto s:

```
print(len({'azul', 'verde', 'rojo'}))
print(len(set('abracadabra')))
print(len(set()))
```

Operaciones sobre conjuntos

Como hemos dicho, comenzaremos viendo una serie de sencillos ejemplos de operaciones entre conjuntos (los cuales, en Python representaremos entre llaves “{}“) Dichas operaciones las podremos efectuar de dos maneras: O bien utilizando la función correspondiente, o bien mediante el uso de un símbolo u operador (ej: &, ^,|..). Así, para cada una de las operaciones usaremos ambas métodos.

En la siguiente imagen se muestra una representación visual de las principales operaciones que pueden efectuarse entre conjuntos y cuya realización en Python, pasaremos a ver a continuación:



Operaciones sobre conjuntos

x in s permite saber si el elemento x está en el conjunto s:

```
print(3 in {2, 3, 4})
print(5 in {2, 3, 4})
```

x not in s permite saber si x no está en s:

```
print(10 not in {2, 3, 4})
```

Operaciones sobre conjuntos

s.add(x) agrega el elemento x al conjunto s:

```
s = {6, 1, 5, 4, 3}
print(s)
s.add(-37)
print(s)
s.add(4)
print(s)
```

Operaciones sobre conjuntos

s.remove(x) elimina el elemento x del conjunto s:

```
s = {6, 1, 5, 4, 3}
print(s)
s.remove(1)
print(s)
```

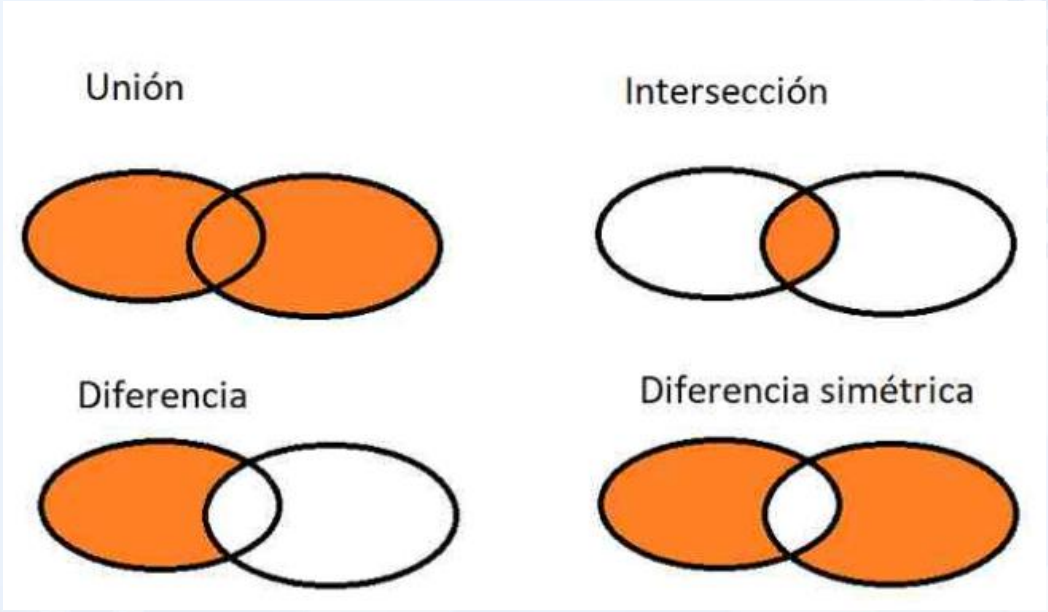
Si el elemento x no está en el conjunto, ocurre un error de llave:

```
s.remove(10)➔Marca Error
print(s)
```

Operaciones sobre conjuntos

Como hemos dicho, comenzaremos viendo una serie de sencillos ejemplos de operaciones entre conjuntos (los cuales, en Python representaremos entre llaves “{}“) Dichas operaciones las podremos efectuar de dos maneras: O bien utilizando la función correspondiente, o bien mediante el uso de un símbolo u operador (ej: &, ^,|..). Así, para cada una de las operaciones usaremos ambas métodos.

En la siguiente imagen se muestra una representación visual de las principales operaciones que pueden efectuarse entre conjuntos y cuya realización en Python, pasaremos a ver a continuación:



Operaciones sobre conjuntos

UNIÓN: La primera operación que vamos a ver es la unión de conjuntos, que como su nombre indica consiste en la obtención de un conjunto que sea el resultado de la unión de los otros (conteniendo los elementos distintos de 2 o más conjuntos). Para ello, en Python podemos usar la función “union()” o bien podemos usar el símbolo “|” entre los conjuntos. Veámoslo con un sencillo ejemplo entre dos conjuntos “{1, 2, 3, 4, 5}” y “{3, 4, 5, 6}” empleando los dos procedimientos:

```
a={1, 2, 3, 4, 5}
b={3, 4, 5, 6}
c=a.union(b)
print(c)
```

Operaciones sobre conjuntos

INTERSECCIÓN: En el caso de la intersección, lo que nos proponemos es obtener (dados dos o más conjuntos) únicamente, los elementos presentes en todos los conjuntos (es decir, los elementos comunes a los conjuntos implicados). Para ello usaremos la función "**intersection()**" o el símbolo "**&**":

```
a={1, 2, 3, 4, 5}
b={3, 4, 5, 6}
c=a.intersection(b)
print(c)
```

Operaciones sobre conjuntos

& y | son, respectivamente, los operadores de intersección y unión:

```
a = {1, 2, 3, 4}
print(a)
b = {2, 4, 6, 8}
print(b)
c= a & b
print(c)
d= a | b
print(d)
```

Operaciones sobre conjuntos

DIFERENCIA: En este caso, de lo que se trata es de restar a un conjunto dado, los elementos de otro. Para ello usaremos la función "**difference()**" o bien, el operador "–":

```
a={1, 2, 3, 4, 5}
b={3, 4, 5, 6}
c=a.difference(b)
print(c)
```

Operaciones sobre conjuntos

DIFERENCIA SIMÉTRICA: Consistente en los elementos de los conjuntos implicados, a excepción de los que se encuentran en la intersección (elementos comunes). En este caso, usaremos o bien la función "**symmetric_difference()**", o bien el símbolo "^":

```
a={1, 2, 3, 4, 5}
b={3, 4, 5, 6}
c=a.symmetric_difference(b)
print(c)
```

Operaciones sobre conjuntos

s - t entrega la diferencia entre s y t; es decir, los elementos de s que no están en t:

```
a = {1, 2, 3, 4}
print(a)
b = {2, 4, 6, 8}
print(b)
c= a - b
print(c)
```

s ^ t entrega la diferencia simétrica entre s y t; es decir, los elementos que están en s o en t, pero no en ambos:

```
d=a ^ b
print(d)
```

Operaciones sobre conjuntos

El operador < aplicado sobre conjuntos significa «es subconjunto de»:

```
a={1,2}
b={1,2,3}
print({1, 2} < {1, 2, 3})
print({1, 4} < {1, 2, 3})
```

s <= t también indica si s es subconjunto de t. La distinción ocurre cuando los conjuntos son iguales:

```
print({1, 2, 3} < {1, 2, 3})
print({1, 2, 3} <= {1, 2, 3})
```

OPERACIONES CON UN ÚNICO CONJUNTO:

De la misma manera que podemos realizar operaciones entre conjuntos distintos, también podemos realizarlas con los elementos de un conjunto determinado:

COMPROBAR EXISTENCIA DE UN ELEMENTO: La primera operación que podemos realizar dentro de un conjunto es la de comprobar si cierto elemento está o no incluido en el:

```
a={1, 2, 3, 4, 5}
print(2 in a)
print(7 in a)
```

OPERACIONES CON UN ÚNICO CONJUNTO:

AÑADIR ELEMENTOS: A su vez, también podemos usar la función “add()” para añadir nuevos elementos a un conjunto dado:

```
a={1, 2, 3, 4, 5}
print(‘Lista Original:’,a)
a.add(6)
print(‘Lista Modificada adiciono 6:’,a)
```

OPERACIONES CON UN ÚNICO CONJUNTO:

ELIMINAR ELEMENTOS: De modo semejante podemos proceder para eliminar elementos de un conjunto. Solo que en este caso, podremos usar las funciones “remove()” o “discard()“:

```
print('Elimniar')
a={1, 2, 3, 4, 5}
print('Lista Original:',a)
a.remove(3)
print('Lista Modificada eliminamos 3:',a)
a.discard(2)
print('Lista Modificada eliminamos 2:',a)
```

OPERACIONES CON UN ÚNICO CONJUNTO:

Vemos como en los dos casos realizamos un cambio en el conjunto, consistente en la eliminación del elemento introducido como argumento de la correspondiente función. En este caso puede surgir la pregunta acerca de la diferencia entre “remove()” y “discard()“. Tal cuestión quedará aclarada si intentamos eliminar del conjunto un elemento no incluido en el:

letras={'A','B','C'}

letras.remove('F’) ## ➔Marca Error

letras.discard('F')

OPERACIONES CON UN ÚNICO CONJUNTO:

Vemos como en los dos casos realizamos un cambio en el conjunto, consistente en la eliminación del elemento introducido como argumento de la correspondiente función. En este caso puede surgir la pregunta acerca de la diferencia entre “remove()” y “discard()“. Tal cuestión quedará aclarada si intentamos eliminar del conjunto un elemento no incluido en el:

letras={'A','B','C'}

letras.remove('F’) ## ➔Marca Error

letras.discard('F’)

emos como empleando “remove()” obtenemos un error derivado del hecho de no existir dentro del conjunto el elemento que queremos eliminar. Mientras que usando “discard()” no obtenemos ningún error (simplemente no se realiza acción alguna). En pocas palabras, que usando “discard()” eliminaremos un elemento en el caso de que este se encuentre ya incluido.

OPERACIONES CON UN ÚNICO CONJUNTO:

COPIAR CONJUNTO: También podemos hacer la copia del contenido de un conjunto, usando la función “copy()” que aplicaremos sobre el conjunto a copiar y que no necesitará de ningún argumento:

```
letras={'A','B','C'}
print('Lista Original:',letras)
nuevo=letras.copy()
print('Lista Nueva:',nuevo)
```

Métodos de la clase set en Python

Termino este tutorial listando los métodos principales de la clase set en Python:

Método	Descripción
`add(e)`	Añade un elemento al conjunto.
`clear()`	Elimina todos los elementos del conjunto.
`copy()`	Devuelve una copia superficial del conjunto.
`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`.
`discard(e)`	Elimina, si existe, el elemento del conjunto.
`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`.
`isdisjoint(iterable)`	Devuelve `True` si dos conjuntos son disjuntos.
`issubset(iterable)`	Devuelve `True` si el conjunto es subconjunto del `iterable`.

Métodos de la clase set en Python

Termino este tutorial listando los métodos principales de la clase set en Python:

`issuperset(iterable)`	Devuelve `True` si el conjunto es superconjunto del `iterable`.
`pop()`	Obtiene y elimina un elemento de forma aleatoria del conjunto.
`remove(e)`	Elimina el elemento del conjunto. Si no existe lanza un error.
`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`.

Ejercicios

Considere las siguientes asignaciones:

a = {5, 2, 3, 9, 4}

b = {3, 1}

c = {7, 5, 5, 1, 8, 6}

d = [6, 2, 4, 5, 5, 3, 1, 3, 7, 8]

e = {(2, 3), (3, 4), (4, 5)}

f = [{2, 3}, {3, 4}, {4, 5}]

Sin usar el computador, indique cuál es el resultado y el tipo de las siguientes expresiones. A continuación, verifique sus respuestas en el computador.

- len(c)
- len(set(d))
- a & (b | c)
- (a & b) | c
- c - a
- max(e)
- f[0] < a
- set(range(4)) & a
- (set(range(4)) & a) in f
- len(set('perro'))
- len({'perro'})