

Table of Contents

- 1 ¿Qué son los sets y como se definen?
- 2 Propiedades de los sets
- 3 Indexación de los sets
- 4 Métodos de los sets
 - 4.1 .add()
 - 4.2 .update()
 - 4.3 .copy()
 - 4.4 .pop()
 - 4.5 .remove()
 - 4.6 .discard()
 - 4.7 .clear()
- 5 Operaciones de conjuntos con sets
 - 5.1 .union()
 - 5.2 .intersection()
 - 5.3 .difference()
 - 5.4 .symmetric_difference()
 - 5.5 .isdisjoint()
 - 5.6 .issubset()
 - 5.7 .issuperset()
- 6 Ejercicios:

¿Qué son los sets y como se definen?

En Python, un conjunto (set) es una colección desordenada y no repetida de elementos únicos y mutables y vienen definidos por {} . Los conjuntos se utilizan principalmente para realizar operaciones matemáticas, como unión, intersección, diferencia y diferencia simétrica entre conjuntos.

Las principales características de los sets son:

- Colección no ordenada: los elementos dentro de un conjunto no tienen un orden específico. No se pueden acceder a los elementos de un conjunto mediante índices (como hacíamos en listas o tuplas), ya que no existe un orden definido.
- Elementos únicos: cada elemento dentro de un conjunto es único, es decir, no se pueden tener elementos duplicados en un conjunto.
- Mutable: el conjunto en sí mismo es mutable, lo que significa que se pueden agregar o eliminar elementos del conjunto.
- Métodos para realizar operaciones matemáticas: los conjuntos en Python vienen con

una serie de métodos integrados para realizar operaciones matemáticas, como unión, intersección, diferencia y diferencia simétrica entre conjuntos.

- No admite elementos mutables: los conjuntos no pueden contener elementos mutables como listas, conjuntos o diccionarios, ya que estos elementos pueden cambiar y afectar la integridad del conjunto.
- Iterables: los conjuntos son iterables, lo que significa que se pueden recorrer los elementos de un conjunto utilizando un bucle for.

En resumen, los conjuntos o sets en Python son colecciones desordenadas y no repetidas de elementos únicos e inmutables que se utilizan principalmente para realizar operaciones matemáticas eficientes. Los conjuntos son mutables y están implementados como una tabla hash. Los elementos dentro de un conjunto deben ser inmutables y no se pueden tener elementos duplicados en un conjunto.

Ya hemos visto que se definen con {} , pero veamos que formas tenemos para crear sets en Python:

- Utilizando llaves {}
- Utilizando la función set()

Veamos estos métodos con más detalle:

Utilizando llaves

```
In [1]: # definimos un set con una serie de strings
alergenos1 = {'Soja', 'Trigo', 'Cacahuetes', 'Leche', 'Huevos', 'Frutos secos', 'Crustáceos'}

# si nos fijamos, en el set hemos puesto dos veces el string "Huevos". Una de La
print("El set que acabamos de crear es:", alergenos1)
```

El set que acabamos de crear es: {'Pescado', 'Frutos secos', 'Cacahuetes', 'Trigo', 'Leche', 'Huevos', 'Soja', 'Crustáceos'}

Utilizando la función set()

```
In [2]: # Podemos crear un set que venga de una tupla, como hemos aprendido hasta ahora
tupla_alergenos = ('Leche', 'Huevos', 'Cacahuetes', 'Frutos secos', 'Pescado', 'Crustáceos')

# convertimos a set usando el método 'set()'
alergenos2 = set(tupla_alergenos)

print("El set que acabamos de crear es:", alergenos2)

# también podemos crear un set de una lista
lista_alergenos = ['Leche', 'Huevos', 'Cacahuetes', 'Frutos secos', 'Pescado', 'Crustáceos']

# convertimos a set usando el método 'set()'
alergenos3 = set(lista_alergenos)

print("El set que acabamos de crear es:", alergenos3)
```

```
El set que acabamos de crear es: {'Frutos secos', 'Cacahuetes', 'Trigo', 'Leche',
```

```
'Crustáceos', 'Huevos', 'Soja', 'Pescado'}
```

```
El set que acabamos de crear es: {'Frutos secos', 'Cacahuetes', 'Trigo', 'Leche',  
'Crustáceos', 'Huevos', 'Soja', 'Pescado'}
```

Propiedades de los sets

Al igual que en las listas y en las tuplas y diccionarios vamos a tener una serie de métodos que nos ayuden a conocer las principales propiedades de las tuplas. En este caso va a ser más sencillo, ya que de todos los que hemos ido aprendiendo, los únicos que podremos usar en sets son:

- `len()` : al igual que hasta ahora, este método se utiliza para obtener la cantidad de elementos que hay en un set.
- `in` y `not in` : se usan para verificar si un elemento está presente o no en una estructura de datos, como una lista, una tupla, un diccionario o un conjunto.

Veámoslo en más detalle:

```
In [3]: print("La longitud del set 'alergenos1' es:", len(alergenos1))  
print("La longitud del set 'alergenos1' es:", len(alergenos2))  
print("La longitud del set 'alergenos1' es:", len(alergenos3))  
  
# esta la palabra "Huevos" dentro de nuestro set?  
print("¿Esta la palabra 'Huevos' en el set 'alergenos1?", 'Huevos' in alergenos1)  
  
# esta la palabra "Apio" en nuestro set?  
print("¿Esta la palabra 'Apio' en el set 'alergenos1?", 'Apio' in alergenos1)
```

La longitud del set 'alergenos1' es: 8
La longitud del set 'alergenos1' es: 8
La longitud del set 'alergenos1' es: 8
¿Esta la palabra 'Huevos' en el set 'alergenos1? True
¿Esta la palabra 'Apio' en el set 'alergenos1? False

Indexación de los sets

Hay que tener en cuenta que los conjuntos son estructuras de datos no ordenadas, por lo que no se puede acceder a los elementos por su posición y por lo tanto no podremos realizar la indexación como aprendimos con las listas y las tuplas. Aún así, veamos un ejemplo para ver el error que nos sale

```
In [4]: alergenos1[3]
```

```
-----  
TypeError
```

```
Traceback (most recent call last)
```

```
Cell In[4], line 1
```

```
----> 1 alergenos1[3]
```

```
TypeError: 'set' object is not subscriptable
```

¿Qué error nos devuelve?

TypeError: 'set' object is not subscriptable , es decir, no es indexable, no podemos acceder a sus elementos a partir de los índices.

Métodos de los sets

Los métodos de set son funciones que se pueden utilizar para manipular conjuntos, agregar o eliminar elementos y realizar operaciones entre conjuntos. Aquí hay algunos métodos comunes de set y una breve explicación de lo que hacen:

- `add()` : Agrega un elemento al conjunto. Si el elemento ya está en el conjunto, no se hace nada.
- `update()` : es un método de conjunto (set) que se utiliza para agregar múltiples elementos a un conjunto.
- `copy()` : se utiliza para crear una copia superficial de un conjunto existente
- `pop()` : Elimina y devuelve un elemento aleatorio del conjunto. Si el conjunto está vacío, se produce un error.
- `remove()` : Elimina un elemento del conjunto. Si el elemento no está en el conjunto, se produce un error.
- `discard()` : Elimina un elemento del conjunto. Si el elemento no está en el conjunto, no se produce ningún error.
- `clear()` : Elimina todos los elementos del conjunto.

.add()

Es un método de sets que se utiliza para agregar un elemento al conjunto.

```
In [ ]: # imaginemos que ahora queremos añadir un alérgeno a nuestro set usando el método
print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)
print("La cantidad de alergenos que tenemos en nuestro set son:", len(alergenos1))

# utilizando el método 'add()' añadimos 'Mostaza'
alergenos1.add('Mostaza')

print("El set 'alergenos1' DESPUÉS DE AÑADIR LA MOSTAZA contiene los siguientes")
print("La cantidad de alergenos que tenemos en nuestro set DESPUÉS DE AÑADIR LA

# Tal vez nos interese añadir más de un elemento a nuestro set, siguiendo la misma
alergenos1.add(['Brocoli', "Sulfitos"])
```

El set 'alergenos1' contiene los siguientes elementos: {'Pescado', 'Cacahuetes', 'Leche', 'Huevos', 'Frutos secos', 'Trigo', 'Soja', 'Crustáceos'}
La cantidad de alergenos que tenemos en nuestro set son: 8
El set 'alergenos1' DESPUÉS DE AÑADIR LA MOSTAZA contiene los siguientes elementos: {'Pescado', 'Cacahuetes', 'Leche', 'Huevos', 'Frutos secos', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos'}
La cantidad de alergenos que tenemos en nuestro set DESPUÉS DE AÑADIR LA MOSTAZA son: 9

```
-----
TypeError Traceback (most recent call last)
/tmp/ipykernel_721/1311920205.py in <module>
 11
 12 # Tal vez nos interese añadir más de un elemento a nuestro set, siguiendo
la misma lógica podríamos pensar que solo tendríamos que pasarselos al método 'ad
d()'. Probemos:
--> 13 alergenos1.add(['Brocoli', "Sulfitos"])

TypeError: unhashable type: 'list'
```

Vaya... 😳 nos devuelve un error. En este caso nos devuelve `TypeError: unhashable type: 'list'`. Basicamente, lo que nos esta diciendo es que con el método `.add()` no podemos usar listas.

Está bien, ya hemos comprobado que no podemos usar el método `.add()` cuando queremos insertar más de un elemento en nuestros sets. Usemos, entonces, el método `.update()` ↗

.update()

Es un método de conjunto (set) que se utiliza para agregar múltiples elementos a un conjunto, se lo deberemos pasar como lista.

```
In [ ]: print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)

# nos creamos una lista con los elementos que queremos introducir en nuestro set
actualizacion = ['Brocoli', 'Sulfitos']

# le pasamos la lista creada al método 'update()'
alergenos1.update(actualizacion)

print("El set 'alergenos1' DESPUÉS DE AÑADIR LOS ELEMENTOS contiene los siguiente
```

El set 'alergenos1' contiene los siguientes elementos: {'Pescado', 'Cacahuetes', 'Leche', 'Huevos', 'Frutos secos', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos'}
El set 'alergenos1' DESPUÉS DE AÑADIR LOS ELEMENTOS contiene los siguientes elementos: {'Pescado', 'Sulfitos', 'Cacahuetes', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos'}

```
In [ ]: print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)
# también le podemos pasar un set a este método, definimos un set con los nuevos
actualizacion2 = {'Altramuces', "Moluscos"}

# usamos el método 'update()'
alergenos1.update(actualizacion2)

print("El set 'alergenos1' DESPUÉS DE AÑADIR LOS ELEMENTOS contiene los siguiente
```

El set 'alergenos1' contiene los siguientes elementos: {'Pescado', 'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos'}

El set 'alergenos1' DESPUÉS DE AÑADIR LOS ELEMENTOS contiene los siguientes elementos: {'Pescado', 'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

.copy()

Al igual que en todos los tipos de datos aprendidos hasta ahora, en Python es un método de conjunto (set) que se utiliza para crear una copia superficial (shallow copy) de un conjunto existente. La copia superficial significa que se crea un nuevo conjunto con los mismos elementos que el conjunto original, pero los elementos en sí mismos no se copian, solo se hace una referencia a ellos.

```
In [ ]: print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)
# nos hacemos una copia de set alergenos
alergenos_copia = alergenos1.copy()
print("El set 'alergenos1' contiene los siguientes elementos:", alergenos_copia)
```

El set 'alergenos1' contiene los siguientes elementos: {'Pescado', 'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' contiene los siguientes elementos: {'Pescado', 'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

.pop()

El método `pop()` en Python es un método de conjunto (set) que se utiliza para eliminar y devolver un elemento aleatorio del conjunto. En este caso nos elimina un elemento al azar ya que los set, recordemos, que no tienen orden!

Igual que en los otros tipos de datos, el `pop` nos va a devolver el elemento que elimina de nuestro set, sobre escribiendo el set original.

```
In [ ]: # eliminamos un elemento de set. Como no tienen orden, lo eliminará de forma aleatoria
print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)
print("El set 'alergenos1' contiene ", len(alergenos1), "elementos")
print("-----\n")

# almacenamos en una variable el elemento que nos elimina el método '.pop()'
elemento = alergenos1.pop()
print("El elemento eliminado de nuestro set 'alergenos1' es:", elemento)
print("El set 'alergenos1' DESPUÉS DE HABER ELIMINADO UN ELEMENTO AL AZAR es:")
print("El set 'alergenos1' DESPUÉS DE HABER ELIMINADO UN ELEMENTO AL AZAR contiene")
```

El set 'alergenos1' contiene los siguientes elementos: {'Pescado', 'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' contiene 13 elementos

El elemento eliminado de nuestro set 'alergenos1' es: Pescado

El set 'alergenos1' DESPUÉS DE HABER ELIMINADO UN ELEMENTO AL AZAR es: {'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' DESPUÉS DE HABER ELIMINADO UN ELEMENTO AL AZAR contiene 12 elementos

.remove()

Es una función de los sets en Python que se utiliza para eliminar un elemento específico de un conjunto. Es importante tener en cuenta que el método `remove()` solo elimina un elemento a la vez.

```
In [ ]: print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)
```

```
# imaginemos que queremos eliminar 'Huevos' y "Soja" de nuestro set, como hemos
alergenos1.remove(["Huevos", "Soja"])
```

El set 'alergenos1' contiene los siguientes elementos: {'Sulfitos', 'Cacahuete', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

```
-----  
TypeError Traceback (most recent call last)  
/tmp/ipykernel_721/3204689325.py in <module>  
 2  
 3 # imaginemos que queremos eliminar 'Huevos' y "Soja" de nuestro set, como  
hemos dicho tendremos que hacerlo por separado, pero para confirmarlo, veamos que  
error nos devuelve cuando intentamos eliminar dos elementos a la vez  
----> 4 alergenos1.remove(["Huevos", "Soja"])
```

`TypeError: unhashable type: 'list'`

Este error lo que nos esta diciendo es que este método no acepta listas. Por lo tanto, tendremos que hacerlo de forma individual. Veámoslo:

```
In [ ]: # en este caso usaremos el set alergenos2  
print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)  
print("El set 'alergenos1' ANTES DE ELIMINAR UN ELEMENTO contiene ", len(alergenos1))  
print("-----\n")  
  
# aplicamos la función '.remove()' incluyendo el elemento que queremos eliminar  
alergenos1.remove("Huevos")  
print("El set 'alergenos1' DESPUÉS DE ELIMINAR 'Huevos' contiene los siguientes elementos")  
print("El set 'alergenos1' DESPUÉS DE HABER ELIMINADO 'Huevos' contiene ", len(alergenos1))  
print("-----\n")  
  
# seguimos eliminando más elementos  
alergenos1.remove("Soja")  
print("El set 'alergenos1' DESPUÉS DE ELIMINAR 'Soja' contiene los siguientes elementos")  
print("El set 'alergenos1' DESPUÉS DE HABER ELIMINADO 'Soja' contiene ", len(alergenos1))
```

El set 'alergenos1' contiene los siguientes elementos: {'Sulfitos', 'Cacahuetes', 'Leche', 'Huevos', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' ANTES DE ELIMINAR CUALQUIER ELEMENTO contiene 12 elementos

El set 'alergenos1' DESPUÉS DE ELIMINAR 'Huevos' contiene los siguientes elementos: {'Sulfitos', 'Cacahuetes', 'Leche', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Soja', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' DESPUÉS DE HABER ELIMINADO 'Huevos' contiene 11 elementos

El set 'alergenos1' DESPUÉS DE ELIMINAR 'Soja' contiene los siguientes elementos: {'Sulfitos', 'Cacahuetes', 'Leche', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' DESPUÉS DE HABER ELIMINADO 'Soja' contiene 10 elementos

In []: # ¿Qué pasaría si intentamos eliminar un elemento que no existe en nuestro set?
alergenos1.remove("Patata")

KeyError Traceback (most recent call last)
/tmp/ipykernel_721/2124778752.py in <module>
 1 # ¿Qué pasaría si intentamos eliminar un elemento que no existe en nuestro set?
----> 2 alergenos1.remove("Patata")

KeyError: 'Patata'

Nos va a dar un error, ya que ese elemento no existe en nuestro set.

.discard()

Es una función de los sets en Python que se utiliza para eliminar un elemento específico de un conjunto, **pero sin generar un error si el elemento no existe en el conjunto**. Es importante tener en cuenta que el método discard() solo elimina un elemento a la vez.

In []: # en este caso usaremos el set alergenos2
print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)
print("El set 'alergenos1' ANTES DE ELIMINAR UN ELEMENTO contiene ", len(alergenos1))
print("-----\n")

aplicamos la función '.remove()' incluyendo el elemento que queremos eliminar
alergenos1.discard("Sulfitos")
print("El set 'alergenos1' DESPUÉS DE ELIMINAR 'Sulfitos' contiene los siguientes elementos:")
print("El set 'alergenos1' DESPUÉS DE HABER ELIMINADO 'Huevos' contiene ", len(alergenos1))
print("-----\n")

El set 'alergenos1' contiene los siguientes elementos: {'Sulfitos', 'Cacahuetes', 'Leche', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' ANTES DE ELIMINAR UN ELEMENTO contiene 10 elementos

El set 'alergenos1' DESPUÉS DE ELIMINAR 'Sulfitos' contiene los siguientes elementos: {'Cacahuetes', 'Leche', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos1' DESPUÉS DE HABER ELIMINADO 'Huevos' contiene 9 elementos

In []: # A diferencia del .remove(), en el caso del .discard() si le pasamos un elemento
alergenos1.discard("Patata")

.clear()

Es una función de los sets en Python que se utiliza para eliminar todos los elementos de un conjunto.

In []: # en este caso usaremos el set alergenos2
print("El set 'alergenos1' contiene los siguientes elementos:", alergenos1)

aplicamos la función '.clear()', que nos dejará el set vacío
alergenos2.clear()
print("El set 'alergenos2' DESPUÉS DE HABER APLICADO EL MÉTODO clear() contiene")

El set 'alergenos1' contiene los siguientes elementos: {'Sulfitos', 'Cacahuetes', 'Leche', 'Frutos secos', 'Brocoli', 'Altramuces', 'Trigo', 'Mostaza', 'Crustáceos', 'Moluscos'}

El set 'alergenos2' DESPUÉS DE HABER APLICADO EL MÉTODO clear() contiene los siguientes elementos: set()

Operaciones de conjuntos con sets

Los sets en Python admiten una variedad de operaciones para manipular y comparar conjuntos o sets. A continuación, presentamos las operaciones más comunes que se pueden realizar con los sets en Python:

- **Unión** : La unión de dos conjuntos se puede realizar utilizando el operador | o el método union(). Este operador y método retornan un nuevo conjunto que contiene todos los elementos únicos de ambos conjuntos.
- **Intersección** : La intersección de dos conjuntos se puede realizar utilizando el operador & o el método intersection(). Este operador y método retornan un nuevo conjunto que contiene solo los elementos que están en ambos conjuntos.
- **Diferencia** : La diferencia de dos conjuntos se puede realizar utilizando el operador - o el método difference(). Este operador y método retornan un nuevo conjunto que contiene los elementos que están en el primer conjunto pero no en el segundo conjunto.

- **Diferencia simétrica** : La diferencia simétrica de dos conjuntos se puede realizar utilizando el operador `^` o el método `symmetric_difference()`. Este operador y método retornan un nuevo conjunto que contiene los elementos que están en uno de los conjuntos pero no en ambos.
- **Elementos en común** : Este método se utiliza para verificar si dos conjuntos no tienen elementos en común. Para utilizar este método, se debe llamar al método `isdisjoint()` en uno de los conjuntos y pasar como argumento el otro conjunto. El método retorna un valor booleano True si los conjuntos son disjuntos (no tienen elementos en común) y False en caso contrario.
- **Subset** : Este método se utiliza para verificar si un conjunto es un subconjunto de otro conjunto. Para utilizar este método, se debe llamar al método `issubset()` en el conjunto que se desea verificar y pasar como argumento el conjunto del que se desea verificar si es subconjunto. El método retorna un valor booleano True si el conjunto es un subconjunto y False en caso contrario.
- **Superset** : Este método es utilizado para determinar si un conjunto (set) contiene a otro conjunto (set), es decir, si el primer conjunto es un superconjunto del segundo conjunto. Para utilizar el método `issuperset()`, se debe llamar a este método en un conjunto y pasar como argumento el otro conjunto que se desea verificar si es un subconjunto. El método devuelve un valor booleano True si el primer conjunto contiene a todos los elementos del segundo conjunto y devuelve False en caso contrario.

`.union()`

Nos devuelve la unión entre los dos `sets` que le pasemos, es decir, los todos los elementos que están en los dos sets.

```
In [ ]: # Lo primero que hacemos es definir dos sets
set1 = {'a', 'b', 'c'}
set2 = {'b', 'c', 'd'}

print("El set1 contiene los siguientes elementos:", set1)
print("El set2 contiene los siguientes elementos:", set2)
print("-----\n")

# imaginemos que queremos unir todos los elementos que tienen los dos sets(), us
set_union = set1.union(set2)
print("Todos los elementos de los 'set1' y el 'set2' son:", set_union)
```

```
El set1 contiene los siguientes elementos: {'c', 'b', 'a'}
El set2 contiene los siguientes elementos: {'c', 'b', 'd'}
-----
```

```
Los elementos que tienen en común el 'set1' y el 'set2' son: {'c', 'b', 'd',
'a'}
```

`.intersection()`

Nos va a devolver los elementos comunes en los dos sets

```
In [ ]: print("El set1 contiene los siguientes elementos:", set1)
print("El set2 contiene los siguientes elementos:", set2)
print("-----\n")

# imaginemos que queremos sacar solo aquellos elementos que están en común en los dos sets
set_intersección = set1.intersection(set2)
print("Los elementos que tienen en común el 'set1' y el 'set2' son:", set_intersección)

El set1 contiene los siguientes elementos: {'c', 'b', 'a'}
El set2 contiene los siguientes elementos: {'c', 'b', 'd'}
-----
Los elementos que tienen en común el 'set1' y el 'set2' son: {'c', 'b'}
```

.difference()

Nos va devolver los elementos que son diferentes entre los dos sets que tenemos.

Aquí, el orden es importante. Por ejemplo:

No es lo mismo restar 5 - 2 que 2 - 5 .

Es decir, nos devuelve los elementos presentes en un set , pero no en el otro.

```
In [ ]: print("El set1 contiene los siguientes elementos:", set1)
print("El set2 contiene los siguientes elementos:", set2)
print("-----\n")

# ¿qué elementos tenemos en el set1 pero no en el set2?
set_diferencia1 = set1.difference(set2)
print("Los elementos que están en el set1 pero no en el set2 son:", set_diferencia1)

El set1 contiene los siguientes elementos: {'c', 'b', 'a'}
El set2 contiene los siguientes elementos: {'c', 'b', 'd'}
-----
Los elementos que están en el set1 pero no en el set2 son: {'a'}
```

```
In [ ]: print("El set1 contiene los siguientes elementos:", set1)
print("El set2 contiene los siguientes elementos:", set2)
print("-----\n")

# ¿qué elementos tenemos en el set2 pero no en el set1?
set_diferencia2 = set2.difference(set1)
print("Los elementos que están en el set2 pero no en el set1 son:", set_diferencia2)

El set1 contiene los siguientes elementos: {'c', 'b', 'a'}
El set2 contiene los siguientes elementos: {'c', 'b', 'd'}
-----
Los elementos que están en el set2 pero no en el set1 son: {'d'}
```

.symmetric_difference()

Es similar al .difference() pero en este caso, este método nos devolverá los

elementos de ambos conjuntos, que no están presentes en el otro. En este caso, nos va a devolver un set con los elementos que son únicos en cada uno de los sets que comparamos. Por lo tanto, en este caso, el orden ya no es importante.

```
In [ ]: print("El set1 contiene los siguientes elementos:", set1)
print("El set2 contiene los siguientes elementos:", set2)
print("-----\n")

# que elementos son unicos del set1 y set2?
set_diferencia_simetrica = set1.symmetric_difference(set2)
print("Los elementos que aparecen en un set pero no en el otro son:", set_difere
```

El set1 contiene los siguientes elementos: {'c', 'b', 'a'}
 El set2 contiene los siguientes elementos: {'c', 'b', 'd'}

 Los elementos que aparecen en un set pero no en el otro son: {'a', 'd'}

.isdisjoint()

En algunos casos, los sets que queremos comparar no tienen ningún elemento en común. Como hemos dicho, es utilizado para determinar si dos conjuntos (sets) son disjuntos, es decir, **si no tienen elementos en común**.

Para utilizar el método `disjoint()`, se debe llamar a este método en un conjunto y pasar como argumento el otro conjunto que se desea comparar. El método devuelve un valor booleano True si los dos conjuntos son disjuntos (no tienen elementos en común) y devuelve False en caso contrario.

```
In [ ]: print("El set1 contiene los siguientes elementos:", set1)
print("El set2 contiene los siguientes elementos:", set2)
print("-----\n")

set_disjoint1 = set1.isdisjoint(set2)
print("¿Tienen los sets los elementos completamente diferentes?", set_disjoint1)

# en este caso nos sale False porque Los dos sets comparten al menos algún eleme
```

El set1 contiene los siguientes elementos: {'c', 'b', 'a'}
 El set2 contiene los siguientes elementos: {'c', 'b', 'd'}

 ¿Tienen los sets los elementos completamente diferentes?: False

```
In [ ]: # definimos dos sets nuevos
set3 = {'b', 'c'}
set4 = {'a', 'd'}

print("El set3 contiene los siguientes elementos:", set3)
print("El set4 contiene los siguientes elementos:", set4)
print("-----\n")

# que elementos son unicos del set1 y set2?
set_disjoint2 = set3.isdisjoint(set4)
print("¿Tienen los sets los elementos completamente diferentes?", set_disjoint2)
```

```
# en este caso nos devuelve True ya que ninguno de los elementos en común.
```

```
El set3 contiene los siguientes elementos: {'c', 'b'}
El set4 contiene los siguientes elementos: {'d', 'a'}
```

```
-----
```

¿Tienen los sets los elementos completamente diferentes?: True

.issubset()

Si no son disjuntos, puede que todos los elementos de un set ya estén definidos en el otro: todos los mamíferos son animales, pero no todos los animales son mamíferos. Pues los mamíferos son un *subset* de los animales, y como sinónimo los animales son un *superset* de los mamíferos. Ambos pueden ser comprobados usando los símbolos < y >. Empezamos con el `subset`:

```
In [ ]: # definimos dos sets nuevos
set5 = {"a", "b", "d"}
set6 = {"a", "d"}

print("El set5 contiene los siguientes elementos:", set5)
print("El set6 contiene los siguientes elementos:", set6)
print("-----\n")

# que elementos son únicos del set1 y set2?
set_issubset1 = set5.issubset(set6)
print("¿Están todos los elementos del set5 en el set6? Es decir, ¿todos los animales son mamíferos?:", set_issubset1)

# en este caso nos devuelve False ya que no todos los elementos del set5 están contenidos en el set6
```

```
El set5 contiene los siguientes elementos: {'b', 'a', 'd'}
El set6 contiene los siguientes elementos: {'d', 'a'}
```

```
-----
```

¿Están todos los elementos del set5 en el set6? Es decir, ¿todos los animales son mamíferos?: False

```
In [ ]: # hagamos el ejemplo inverso
print("El set5 contiene los siguientes elementos:", set5)
print("El set6 contiene los siguientes elementos:", set6)
print("-----\n")

# que elementos son únicos del set1 y set2?
set_issubset2 = set6.issubset(set5)
print("¿Están todos los elementos del set6 en el set5? Es decir, ¿todos los mamíferos son animales?:", set_issubset2)

This comment is explaining the result of the previous line of code, which checks if all elements of set6 are contained in set5.
```

```
# en este caso nos devuelve True ya que todos los elementos del set6 están contenidos en el set5
```

```
El set5 contiene los siguientes elementos: {'b', 'a', 'd'}
```

```
El set6 contiene los siguientes elementos: {'d', 'a'}
```

```
-----
```

¿Están todos los elementos del set6 en el set5? Es decir, ¿todos los mamíferos son animales?: True

.issuperset()

Podríamos decir, que es la inversa del método anterior.

```
In [ ]: print("El set5 contiene los siguientes elementos:", set5)
print("El set6 contiene los siguientes elementos:", set6)
print("-----\n")

# que elementos son unicos del set1 y set2?
set_issuperset1 = set5.issuperset(set6)
print("¿Están todos los elementos del set6 en el set5?:", set_issuperset1)
```

```
El set5 contiene los siguientes elementos: {'b', 'a', 'd'}
El set6 contiene los siguientes elementos: {'d', 'a'}
-----
```

```
¿Están todos los elementos del set6 en el set5?: True
```

```
In [ ]: print("El set5 contiene los siguientes elementos:", set5)
print("El set6 contiene los siguientes elementos:", set6)
print("-----\n")

# que elementos son unicos del set1 y set2?
set_issuperset1 = set6.issuperset(set5)
print("¿Están todos los elementos del set5 en el set6?:", set_issuperset1)
```

```
El set5 contiene los siguientes elementos: {'b', 'a', 'd'}
El set6 contiene los siguientes elementos: {'d', 'a'}
-----
```

```
¿Están todos los elementos del set5 en el set6?: False
```

```
In [ ]:
```