

CONJUNTOS. SET. <CLASS 'SET'>.

¿ALGUNAS CARACTERÍSTICAS?

- Los conjuntos / set : (**set**) [Sets](#) / [conjunto mutable](#) / : <class 'set'>..
- + Es un **dato compuesto** de 0 a N valores, contenido en cualquier tipo de objeto ([literal](#), [variables](#), [expresiones](#), etc.).
- Nota:** Para crear un set vacío, es necesario utilizar la función **set ()**, pues si se utiliza el literal (llaves y nada en medio), **nos crea un diccionario**, no un conjunto.
- + Los datos que contienen:
 - Forman secuencias **Inmutables** (no pueden ser “modificados” una vez creados), pero si añadir o eliminar elementos.
 - **No** están **ordenados**, los elementos aparecen en un orden aleatorio.
 - **No** están **indexados**, los elementos no pueden ser accedidos por su posición/índice.
 - Pueden ser de **distintos** tipos ([numero](#), [cadena](#), [tupla](#), [lista](#), etc.).
 - Los valores **son únicos**, no pueden estar repetidos-duplicados.
- Nota:** aunque se admita “repetir” al crear el literal, cuando lo visualizamos solo sale una ocurrencia, incluso si un mismo valor se encuentra en ítems que son variables con el mismo valor.
- + Se permiten también la **anidación** de conjuntos.

1.1. CREACIÓN.

Se pueden crear datos tipo set <class 'set'>, utilizando:

- Formato **literal**, entre llaves { }, y cada valor separado por comas.

```
[ var_set = ] { valor1 [ , valorN ] . . . }
```

- Función constructora: [set\(\)](#).

set (x) ----- La función constructora [set\(\)](#). Devuelve un tipo de dato conjunto.

+**Nota:** hay que utilizar los dobles corchetes

```
[ var_set = ] set ( ( valor1 [ , valorN ] . . . ) )
```

- **Comprehension Set.**

Se puede crear a partir de cualquier objeto iterable (cadena, tupla, lista...)

1.2. OPERACIONES:

- **De “Secuencia”:**
 - + Como dato **compuesto**, se puede conocer el número de elementos o longitud de la tupla, con la función incorporada **len ()**.
 - + Al **NO estar indexados**(subíndices), **NO** se puede hacer referencia por su posición, es decir utilizar corchetes, NI a una parte (rebanada).
 - + Pero **pueden recorrerse** los elementos del conjunto usando un bucle **for in**, o preguntar si un valor específico está presente en un conjunto, usando la palabra clave **in**.
 - + Al ser **inmutables**, no puede modificarse su contenido, el nombre de los objetos y el valor de los literales, pero si el contenido de un elemento que sea una variable.
- **Operaciones del algebra de conjuntos.**
 - + Permiten las operaciones típicas del algebra de conjuntos: existencia (**in** , **not in**) , unión (**|**), intersección (**&**), diferencia simétrica (**^**) , etc..
- **Otras**, según sus métodos.

1.3. MÉTODOS.

Los conjuntos / set disponen de **17** métodos:

☑ Ver propiedades/métodos de los set.

● Añadir elementos al set:

Se pueden añadir elementos a un set con **add** o **update**.

add (x) ----- El método [add\(\)](#) . Agrega un elemento al conjunto. Siempre al final del conjunto. Si añadimos un valor que ya existe, no hace nada.

update () ----- El método [update\(\)](#). Actualiza el set con la unión de este set y otros. Como [add\(\)](#), al final, es como si se llamara a [add\(\)](#), tantas veces como elementos tenga el set que unimos.

● Eliminar elementos al set:

Se pueden eliminar elementos de un set con: **clear** , **discard**, **pop** , **remove**.

pop () ----- El método [pop\(\)](#) . Elimina, el primer elemento del conjunto. Devuelve el elemento eliminado (por si hay que recuperar).

clear () ----- El método [clear\(\)](#) . Elimina **todos** los elementos del conjunto. Devuelve **None**.

discard () ----- El método [discard\(\)](#). Eliminar el elemento especificado. Si no existe el elemento no sucede nada. Devuelve **None**.

remove () ----- El método [remove\(\)](#). Elimina el elemento especificado. Si no existe el elemento genera una excepción `/error`. Devuelve **None**.

● Operaciones de algebra de conjuntos:

union (x) ----- El método [union\(\)](#). Devuelve un set que contiene la unión de sets.

intersection (x) ----- El método [intersection\(\)](#) . Devuelve un conjunto, que es la intersección de otros dos conjuntos.

difference () . ----- El método [difference\(\)](#). Devuelve un conjunto que contiene la diferencia entre dos o más conjuntos.

symmetric_difference () ----- El método [symmetric_difference\(\)](#). Devuelve un conjunto con las diferencias simétricas de dos conjuntos.

● Sin organizar:

copy () ----- El método [copy\(\)](#). Devuelve una copia del conjunto.

difference_update () ----- El método [difference_update\(\)](#). Elimina los elementos de este conjunto que también se incluyen en otro conjunto especificado.

intersection_update () ----- El método [intersection_update\(\)](#). Quita los elementos de este conjunto que no están presentes en otros conjuntos especificados.

isdisjoint () ----- El método [isdisjoint\(\)](#). Devuelve si dos conjuntos tienen una intersección o no.

issubset() -----El método [issubset\(\)](#). Devuelve si otro conjunto contiene este conjunto o no.

issuperset() -----El método [issuperset\(\)](#). Devuelve si este conjunto contiene otro conjunto o no.

symmetric_difference_update() --El método [symmetric_difference_update\(\)](#). inserta las diferencias simétricas de este conjunto y otro

FROZENSET. <CLASS 'FROZENSET'>.

¿ALGUNAS CARACTERÍSTICAS?

- Los conjuntos / set: (**frozenset**) / : <class 'set'>..

Los objetos de tipo frozenset representan **conjuntos** que **no pueden ser modificados** una vez que son definidos.

frozenset(x) -----La función [frozenset\(\)](#) /.Devuelve un tipo de objeto <class 'frozenset'> .

```
[var_frozebset=] frozenset (iterable)
```

- Los conjuntos / set : (**frozenset**) disponen de 8 métodos:

☒ Listar métodos de los **frozenset**..

DICCIONARIOS. <CLASS 'DICT'>.

¿ALGUNAS CARACTERÍSTICAS?

- Los diccionarios : (**dict**) [Dictionaries](#) / [diccionario](#) / : <class 'dict'>.

+ Es un **dato compuesto** de 0 a N valores, contenido en cualquier tipo de objeto (*literal, variables, expresiones, etc.*).

- A diferencia del resto de las secuencias, los elementos de un diccionario forman un par: *clave:valor*, separado por comas.
- La clave sirve tanto para declarar y acceder al valor (se utiliza la clave en lugar del índice).
- Las claves deben ser **únicas**.

+ Los datos que contienen:

- Forman secuencias **mutables**, pueden ser “modificados” una vez creados.
- **No están ordenados**, los elementos aparecen en un orden aleatorio.
- **No están indexados** por posición/índice, pero SI por su **clave**.
- Pueden ser de **distintos** tipos (*numero, cadena, tupla, lista, etc.*).
- Los valores **son únicos**, no pueden estar repetidos-duplicados.

+ Se permiten también la **anidación** de diccionarios.

1.4. CREACIÓN.

Se pueden crear datos tipo set <class 'dict'>, utilizando:

- Formato **literal**, entre llaves { }, y cada par clave:valor (separado por :), separado del siguiente por comas.

```
[var_dict =] { [ 'clave_1' : valor_1 ] [ , 'clave_N' : valor_N ] ... }
```

- Función constructora: [dict\(\)](#).

dict (x) ----- La función constructora [dict\(\)](#). Devuelve un tipo de dato diccionario.
 + **Nota:** Ahora SIN los dobles corchetes.
 + Se utiliza **=** en lugar de los dos puntos **:**.
 + Las claves solo pueden ser objetos inmutables (números, cadenas, tuplas, set).
Nota: Si es variable cadena, hay que utilizar una cadena, al indexar. EXCEPTO si son de valor numérico entero
COMPROBAR OTROS.

```
[var_dict =] dict (clave = valor [ , clave = valor ] . . . )
```

• Comprehension dict.

1.5. OPERACIONES.

• De “Secuencia”:

- + Como dato **compuesto**, se puede conocer el número de elementos o longitud de la tupla, con la función incorporada **len()**.
- + Se pueden referencia, no por posición, sino por la **clave**.
 - Se accede a los elementos de un diccionario refiriéndose a su nombre de **clave**, entre **corchetes**:
- + Pero **pueden recorrerse** los elementos del conjunto usando un bucle **for in**, o preguntar si un valor específico está presente en un conjunto, usando la palabra clave **in**.
- + Al ser **Mutable**s, puede modificarse su contenido.

• Otras, según sus métodos.

1.6. MÉTODOS.

Los diccionarios disponen de **11** métodos:

☑ Listar métodos de los diccionarios.

• Eliminar elementos (pares) de un diccionario:

Se pueden eliminar elementos de un diccionario con: **clear**, **discard**, **pop**, **popitem**.

pop() ----- El método [pop\(\)](#). Elimina el elemento con la clave especificada.

popitem() ----- El método [popitem\(\)](#). Elimina el último par clave-valor insertado

clear() ----- El método [clear\(\)](#). Elimina todos los elementos del diccionario.

• Sin organizar:

update() ----- El método [update\(\)](#). Actualiza el diccionario con los pares clave-valor especificados

copy() ----- El método [copy\(\)](#). Devuelve una copia del diccionario.

fromkeys() ----- El método [fromkeys\(\)](#). Devuelve un diccionario con las claves y valores especificados

get() ----- El método [get\(\)](#). Devuelve el valor de la clave especificada.

-
- items ()** -----El método [items\(\)](#). Devuelve una lista que contiene una tupla para cada par de valores clave
- keys ()** -----El método [keys\(\)](#). Devuelve una lista que contiene las claves del diccionario
- setdefault ()** -----El método [setdefault\(\)](#). Devuelve el valor de la clave especificada. Si la clave no existe: inserte la clave, con el valor especificado
- values ()** -----El método [values\(\)](#). Devuelve una lista de todos los valores en el diccionario.