CONQUER BLOCKS

PYTHON

SETS

REPASO

Estructuras de datos:

1. Listas → in - built
2. Arrays → importar modulo
3. Tuplas → in - built

QUE VEREMOS HOY

Estructuras de datos in - built:

1. Listas

2. Tuplas

3. Sets

4. Diccionarios

SETS

Definición: Colecciones *no ordenadas* de elementos *únicos* e *inmutables*.

➡ **Los elementos no llevan un indice asociado**

➡ **No podemos reasignar valores a los elementos del set**

➡ **Podemos añadir y borrar elementos**

➡ **Los elementos son únicos, no hay duplicados**

SINTAXIS BASICA DE UN CONJUNTO/SET

Set:

```
# sintaxis de un set
mi_set= {'fruta', 45, True}
print(mi_set)
```

✓ 0.0s

```
{'fruta', 45, True}
```

Tupla:

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
```

✓ 0.0s

```
<class 'tuple'>
```

Lista:

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
```

✓ 0.0s

```
<class 'list'>
```

```
# Crear set vacío
mi_set = set()
print(type(mi_set))
```

✓ 0.0s

```
<class 'set'>
```

CUIDADO →

```
# Crear set vacío
mi_set = {}
print(type(mi_set))
```

✓ 0.0s

```
<class 'dict'>
```

AUSENCIA DE ORDENAMIENTO

```
# sintaxis de un set
set_frutas = {'manzana', 'naranja', 'plátano'}
print(set_frutas)
```

✓ 0.0s

```
{'naranja', 'plátano', 'manzana'}
```

El orden del contenido no se preserva.

AUSENCIA DE ORDENAMIENTO

```
# sintaxis de un set
set_frutas = {'manzana', 'naranja', 'plátano'}
print(set_frutas)
```

✓ 0.0s

```
{'naranja', 'plátano', 'manzana'}
```

El orden del contenido no se preserva.

Los elementos no levan asignados un índice:

```
# no existen los indices
set_frutas = {'manzana', 'naranja', 'plátano'}
print(set_frutas[0])
```

⊗ 0.2s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[43], line 3
      1 # no existen los indices
      2 set_frutas = {'manzana', 'naranja', 'plátano'}
----> 3 print(set_frutas[0])

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

INMUTABILIDAD

```
# no existen los indices
set_frutas = {'manzana', 'naranja', 'plátano'}
set_frutas[0]="pera"
```

⊗ 0.0s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 3
      1 # no existen los indices
      2 set_frutas = {'manzana', 'naranja', 'plátano'}
----> 3 set_frutas[0]="pera"

TypeError: 'set' object does not support item assignment
```

No podemos reasignar valores

UNICIDAD

```
# los elementos son unicos
set_frutas = {'manzana','manzana', 'naranja', 'plátano'}
print(set_frutas)
```

✓ 0.0s

```
{'naranja', 'plátano', 'manzana'}
```

En un set todos los elementos son ***únicos***.
No hay repeticiones.

COMPROBAR PERTENENCIA

Sets:

```
# comprobar pertenencia
frutas = {'manzana', 'naranja', 'plátano'}
print('manzana' in frutas) # True
print('fresa' in frutas) # False
```

✓ 0.0s

```
True
False
```

Listas:

```
# comprobar pertenencia en listas
frutas = ['manzana', 'naranja', 'plátano']
print('manzana' in frutas) # True
print('fresa' in frutas) # False
```

✓ 0.0s

```
True
False
```

*Las pruebas de pertenencia son mucho más eficientes en **sets** que en **listas***

COMPROBAR PERTENENCIA

*Las pruebas de pertenencia son mucho **más eficientes** en **sets** que en **listas…*** ***¿Por qué?***

Los elementos en una lista tienen asociado un índice → Para comprobar la pertenencia se recorren todos los elementos de la lista hasta encontrar o no el coincidente

Los elementos en un set no tiene un indice si no un *hash*

(Un set es una hash table o tabla de hash)

El hash es único para cada elemento, de manera que ese elemento siempre va a estar guardado en el mismo lugar dentro de ese set (en le mismo “bucket”)

→ Python comprueba el bucket correspondiente a ese set y ve si esta lleno o no.

PROPIEDADES: LISTA VS ARRAY VS TUPLA VS SETS

Característica	Lista	Array	Tupla	Conjunto (Set)
Definición	Una colección de elementos ordenados y modificables	Un conjunto de elementos homogéneos ordenados y modificables	Una colección de elementos ordenados e inmutables	Una colección de elementos únicos e inmutables
Sintaxis	mi_lista = [1, 2, 3]	mi_array = np.array([1, 2, 3])	mi_tupla = (1, 2, 3)	mi_set = {1, 2, 3}
Índices	Sí	Sí	Sí	No
Modificable	Sí	Sí	No	Sí
Homogeneidad	No	Sí	No	No
Tamaño fijo	No	Sí	Sí	No
Únicos	No	No	No	Sí
Iterables	Sí	Sí	Sí	Sí

AÑADIR Y BORRAR ELEMENTOS

```
# Añadir elementos
frutas = {'manzana', 'naranja', 'plátano'}
frutas.add('fresa')
print(frutas)
```

✓ 0.0s

```
{'naranja', 'plátano', 'fresa', 'manzana'}
```

Podemos añadir elementos usando el método *add()*

AÑADIR Y BORRAR ELEMENTOS

```
# usar remove para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.remove('naranja')
print(frutas)
```

✓ 0.0s

```
{'plátano', 'manzana'}
```

```
# usar discard para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.discard('naranja')
print(frutas)
```

✓ 0.0s

```
{'plátano', 'manzana'}
```

Podemos usar los métodos remove() o discard() para borrar elementos de un set.

AÑADIR Y BORRAR ELEMENTOS

```
# usar remove para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.remove('fresa')
print(frutas)
```

⊗ 0.0s

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[52], line 3
      1 # usar remove para eliminar un elemento
      2 frutas = {'manzana', 'naranja', 'plátano'}
----> 3 frutas.remove('fresa')
      4 print(frutas)

KeyError: 'fresa'
```

La diferencia entre remove() y discard():

Si intentamos borrar un elemento que no existe en el set remove() nos devuelve un **error**

AÑADIR Y BORRAR ELEMENTOS

```
# usar remove para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.remove('fresa')
print(frutas)
```

⊗ 0.0s

```
KeyError                                  Traceback (most recent call last)
Cell In[52], line 3
      1 # usar remove para eliminar un elemento
      2 frutas = {'manzana', 'naranja', 'plátano'}
```

```
# usar discard para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.discard('fresa')
print(frutas)
```

✓ 0.0s

```
{'naranja', 'plátano', 'manzana'}
```

La diferencia entre remove() y discard():

Si intentamos borrar un elemento que no existe en el set remove() nos devuelve un **error**

Mientras que discard() simplemente no hace **nada**

PASAR DE LISTAS A SETS Y VICEVERSA

```python
# pasamos de lista a set
mi_lista = ['manzana', 'naranja', 'plátano']
print(type(mi_lista))
mi_set = set(mi_lista)
print(type(mi_set))
print(mi_set)
```

✓ 0.0s

```
<class 'list'>
<class 'set'>
{'naranja', 'plátano', 'manzana'}
```

```python
# pasamos de set a lista
mi_set = {'manzana', 'naranja', 'plátano'}
print(type(mi_set))
mi_lista = list(mi_set)
print(type(mi_lista))
print(mi_lista)
```

✓ 0.0s

```
<class 'set'>
<class 'list'>
['naranja', 'plátano', 'manzana']
```

PASAR DE LISTAS A SETS Y VICEVERSA

```python
# pasamos de lista a set
mi_lista = ['manzana', 'naranja', 'plátano']
print(type(mi_lista))
mi_set = set(mi_lista)
print(type(mi_set))
print(mi_set)
```

✓ 0.0s

```
<class 'list'>
<class 'set'>
{'naranja', 'plátano', 'manzana'}
```

```python
# pasamos de set a lista
mi_set = {'manzana', 'naranja', 'plátano'}
print(type(mi_set))
mi_lista = list(mi_set)
print(type(mi_lista))
print(mi_lista)
```

✓ 0.0s

```
<class 'set'>
<class 'list'>
['naranja', 'plátano', 'manzana']
```

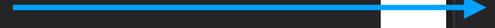
TRABAJANDO CON SETS – EJEMPLO

CREAR UNA LISTA NUEVA ELIMINANDO DUPLICADOS:

```
# Eliminar duplicados en una lista
lista_alumnos = ["Pedro", "Lucas", "Juan", "Lucas"]
set_alumnos = set(lista_alumnos)
lista_alumnos_unico = list(set_alumnos)
print(lista_alumnos_unico)
```

✓ 0.0s

```
['Pedro', 'Lucas', 'Juan']
```



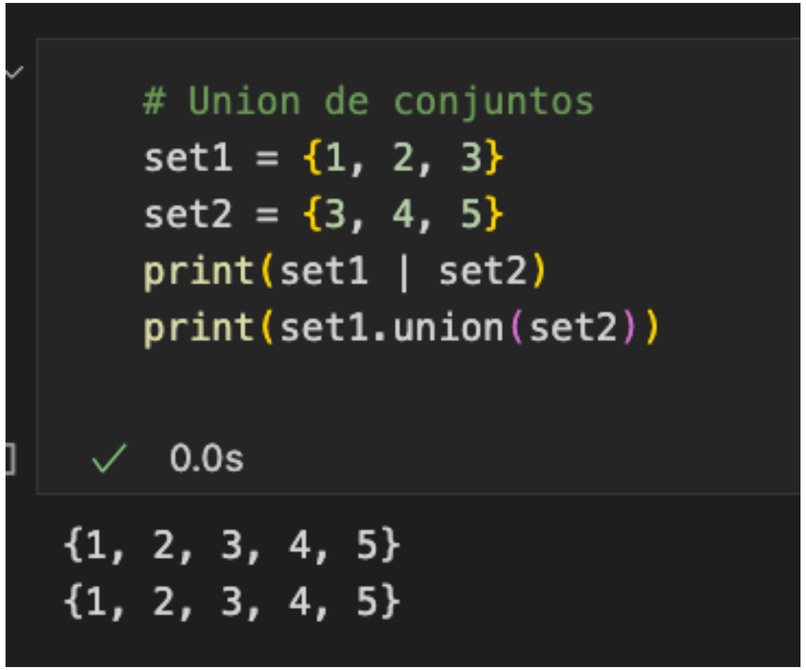
```
# Eliminar duplicados en una lista
lista_alumnos = ["Pedro", "Lucas", "Juan", "Lucas"]
set_alumnos = set(lista_alumnos)
print(set_alumnos)
```

✓ 0.0s

```
{'Pedro', 'Lucas', 'Juan'}
```

TRABAJANDO CON SETS

UNION DE CONJUNTOS:



TRABAJANDO CON SETS

INTERSECCION DE CONJUNTOS:

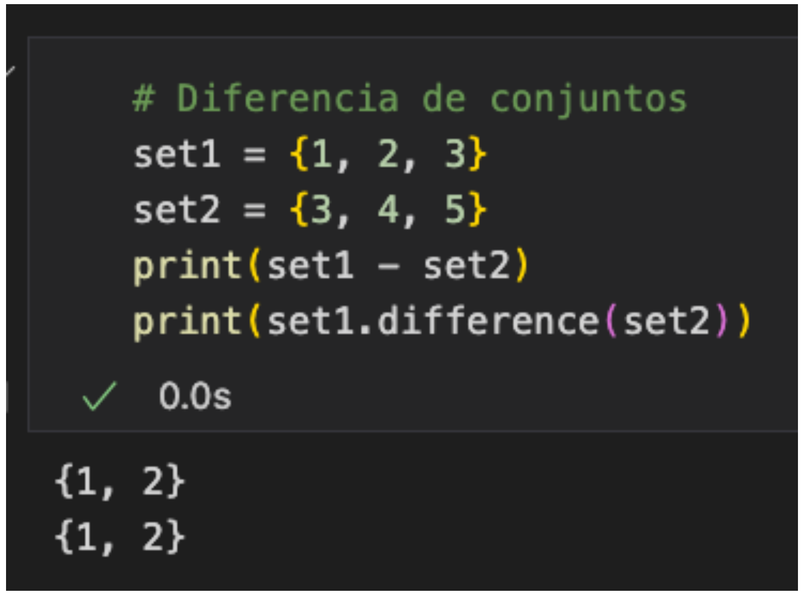
```
# Interseccion de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 & set2)
print(set1.intersection(set2))
```

✓ 0.0s

```
{3}
{3}
```

TRABAJANDO CON SETS

DIFERENCIA DE CONJUNTOS:



TRABAJANDO CON SETS

DIFERENCIA SIMETRICA DE CONJUNTOS:

```python
# Diferencia simétrica de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 ^ set2)
print(set1.symmetric_difference(set2))
```

✓ 0.0s

```
{1, 2, 4, 5}
{1, 2, 4, 5}
```

TRABAJANDO CON SETS

DIFERENCIA SIMETRICA DE CONJUNTOS:

```python
# Diferencia simétrica de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 ^ set2)
print(set1.symmetric_difference(set2))
```

✓ 0.0s

```
{1, 2, 4, 5}
{1, 2, 4, 5}
```

REPASO

1) Que es un set y su sintaxis
2) Propiedades de un set
3) Indices y hash
4) Añadir y borrar elementos de un set
5) Trabajar con sets y listas
6) Operaciones con sets (union, intersección y diferencias)

CONQUER BLOCKS