Colecciones Python: Sets

Contenidos

Introducción

al indice

Otra colección *bulit-in* de Python, compuesta por un conjunto de *valores*. Muy parecidos a las listas. Tienen las siguientes características:

- **Mutables**: podemos modificarlos una vez se hayan creado
- **No ordenado**
- **Elementos unicos**: se compone de un conjunto de valores únicas

¿Cuándo usar sets? Cuando estemos buscando unicidad en nuestros datos y no nos importe el orden.

Te dejo por aquí la documentación para realizar consultas sobre los sets

Si sintaxis es:

```
mi_set = {elemento1, elemento2, elemento3}
```

Tiene una sintaxis muy parecida a la de las listas, pero en este caso no se utilizan los `{}`.

In [4]:
```
colores = {"azul", "rojo", "verde","naranja", "azul", "verde"}
print(len(colores))
print(colores)
print(type(colores))
```

```
4
{'rojo', 'naranja', 'verde', 'azul'}
<class 'set'>
```

Los elementos del set son unicos, por lo que si en la declaración, o posteriormente añadiendo elementos hubiese algún duplicado, el set lo ignoraría. Es más, cuando aplicas el `len`, muestra la cantidad de valores únicos que tiene.

In [6]:
```
# Añadir valores:
colores.add("magenta")
print(colores)

colores.update(["celeste", "burdeos","naranja"])
print(colores)

colores.remove("naranja")
print(colores)

for color in colores:
    print(color)
```

```
{'magenta', 'azul', 'rojo', 'naranja', 'verde'}
{'magenta', 'celeste', 'azul', 'rojo', 'naranja', 'verde', 'burdeos'}
{'magenta', 'celeste', 'azul', 'rojo', 'verde', 'burdeos'}
magenta
celeste
azul
rojo
verde
burdeos
```

Para comprobar si dos sets tienen los mismos elementos

In [7]:
```
set1 = {1,2,4,4,2,1,2,1}
set2 = {2.1,4}

set1 == set2
```

Out[7]:
```
False
```

In [9]:
```
lista_con = [100,10,23,23,15,32,45,10,100,23]
print(lista_con)
print(list(set(lista_con)))
lista_con = list(set(lista_con))
print(lista_con)
```

```
[100, 10, 23, 23, 15, 32, 45, 10, 100, 23]
[32, 100, 10, 45, 15, 23]
[32, 100, 10, 45, 15, 23]
```

In [ ]:
```

```