

# Conjuntos y Diccionarios

## Conjuntos

El concepto de **conjunto** en Python es similar al matemático: una colección de elementos sin orden ni repetición. Las operaciones básicas para trabajar con conjuntos permiten crear un conjunto, calcular la unión, intersección, diferencia y diferencia simétrica.

In [1]:  *# Podemos formar conjuntos directamente, por extensión, con la notación*


```
conj_a = {"a", "e", "i", "o", "u"}
conj_b = {"a", "b", "r", "a", "c", "a", "d", "a", "b", "r", "a"}
```

*# Y con ellos es posible usar las operaciones típicas de conjuntos:*

```
print("A = ", conj_a)
print("B = ", conj_b)
print("A unión B = ", conj_a | conj_b)
print("A intersección B = ", conj_a & conj_b)
print("A dif. simétrica B = ", conj_a ^ conj_b)
```

```
A = {'o', 'i', 'u', 'e', 'a'}
B = {'r', 'c', 'b', 'd', 'a'}
A unión B = {'o', 'r', 'i', 'c', 'u', 'b', 'd', 'e', 'a'}
A intersección B = {'a'}
A dif. simétrica B = {'r', 'o', 'c', 'u', 'd', 'e', 'i', 'b'}
```

Un conjunto puede crearse a partir de una lista, y también es posible añadir elementos a un conjunto o suprimirlos, así como comprobar la pertenencia de un elemento a un conjunto fácilmente:

In [2]:  `conj_A = set("aeiou")`

```
print(conj_A)
conj_A.add("A")
print(conj_A)
conj_A.remove("A")
print(conj_A)

if "a" in conj_A:
    print("a está en el conjunto")
else:
    print("a NO está en el conjunto")
```

```
{'o', 'i', 'u', 'e', 'a'}
{'o', 'A', 'i', 'u', 'e', 'a'}
{'o', 'i', 'u', 'e', 'a'}
a está en el conjunto
```

In [3]: `# Las operaciones típicas de conjuntos pueden realizarse con la notación`

```
conj_a = {"a", "e", "i", "o", "u"}
conj_b = {"a", "b", "r", "a", "c", "a", "d", "a", "b", "r", "a"}
print("A = ", conj_a)
print("B = ", conj_b)
print(conj_a.union(conj_b))
print(conj_a.intersection({"a", "r", "p", "e", "g", "i", "o"}))
print(conj_a.difference({"a", "g", "r", "i", "o"}))
print("A = ", conj_a)
print("B = ", conj_b)
conj_a.issubset({"t", "r", "o", "m", "p", "e", "t", "a"})
```

```
A = {'o', 'i', 'u', 'e', 'a'}
B = {'r', 'c', 'b', 'd', 'a'}
{'o', 'r', 'i', 'c', 'u', 'b', 'd', 'e', 'a'}
{'e', 'i', 'a', 'o'}
{'e', 'u'}
A = {'o', 'i', 'u', 'e', 'a'}
B = {'r', 'c', 'b', 'd', 'a'}
```

Out[3]: False

In [4]: `# Con conjuntos es posible y muy cómoda la notación por comprensión o in`

```
conj_a = {c for c in "aeiou"}
conj_b = {c for c in "abracadabra"}
print("A = ", conj_a)
print("B = ", conj_b)
```

```
A = {'o', 'i', 'u', 'e', 'a'}
B = {'r', 'c', 'b', 'd', 'a'}
```

In [5]: `# Divisores:`

```
def divisores(n):
    """pre: n > 0"""
    return {i for i in range(2, n+1) if n % i == 0}

print(divisores(24))
```

```
{2, 3, 4, 6, 8, 12, 24}
```

## Ejercicio

El máximo común divisor de dos números se puede calcular así: calculamos el conjunto de los divisores de cada uno de los números; la intersección da los divisores comunes, y de ellos, nos quedamos con el mayor (prueba la función max). No es el método más eficiente para calcular el m.c.d. de dos enteros, pero nos sirve para practicar un poco con conjuntos.

## Diccionarios

Un **diccionario** (también llamado **memoria asociativa**) es parecido a un array, pero en vez de dar información a partir de un índice, que es la posición, da información a partir de una clave cualquiera.

```
In [6]: ▶ # Mis teléfonos:

telefonos = {"despacho": 3957, "casa": 917121314}
print(telefonos)
telefonos["movil"] = 687001122
print(telefonos)
print(telefonos["movil"])
del telefonos["movil"]
print(telefonos)

{'despacho': 3957, 'casa': 917121314}
{'despacho': 3957, 'casa': 917121314, 'movil': 687001122}
687001122
{'despacho': 3957, 'casa': 917121314}
```

```
In [7]: ▶ # Consulta de un elemento inexistente:

print(telefonos)
print(telefonos["manuel"]) # Esto dará lugar a un error

{'despacho': 3957, 'casa': 917121314}

-----
----
KeyError                                Traceback (most recent call 1
ast)
<ipython-input-7-5e0b10da71ad> in <module>
      2
      3 print(telefonos)
----> 4 print(telefonos["manuel"]) # Esto dará lugar a un error

KeyError: 'manuel'
```

```
In [8]: ▶ # Se puede comprobar primero si está (la clave de) el elemento:

if "manuel" in telefonos.keys():
    print(telefonos["manuel"])
else:
    print("manuel" + " no tiene un teléfono registrado")

manuel no tiene un teléfono registrado
```

```
In [9]: ▶ # Las claves en el diccionario y los valores se pueden extraer por separ

print(telefonos.keys()) # Ojo: sin un orden prefijado
print(list(telefonos.keys())) # En forma de lista

print(telefonos.values()) # Ojo: sin un orden prefijado
print(list(telefonos.values())) # En forma de lista

dict_keys(['despacho', 'casa'])
['despacho', 'casa']
dict_values([3957, 917121314])
[3957, 917121314]
```

In [10]: **▶** *# Recorridos en diccionarios:*

```
for sitio in telefonos:
    print(sitio, telefonos[sitio])
print("-----")
for sitio, num_tel in telefonos.items():
    print(sitio, num_tel)
```

```
despacho 3957
casa 917121314
-----
despacho 3957
casa 917121314
```

In [11]: **▶** *# A cada clave se le puede asignar una estructura de datos más compleja:*

```
agenda = {"Fernando": {"tel.casa": 11111111,
                        "tel.movil": 22222222,
                        "direccion": "Pozuelo"},
          "Elena": {"tel.casa": 55555555,
                    "tel.movil": 66666666,
                    "direccion": "Colón"},
          }
for persona in agenda:
    print(persona + " vive en " + agenda[persona]["direccion"])
```

```
Fernando vive en Pozuelo
Elena vive en Colón
```

## Ejercicio sencillo:

Partimos de un archivo de texto, un fragmento literario tel vez.

- Se pide extraer la lista de las palabras que están en dicho archivo.
- Se pide ahora extraer el conjunto, esto es, con las palabras pero sin repetición.
- Se desea también contar el número de veces que aparece cada palabras en dicho archivo.
- Por último, se desea construir un diccionario cuyas claves sean frecuencias y cuyos valores conjunto de palabras:

frecs = {1: {palabras con frecuencia 1}, ...}

## Práctica:

Deseamos gestionar el inventario de un supermercado. Cada producto se localiza mediante su código, y además de eso se guarda de él lo siguiente:

- La ubicación en las estanterías del supermercado
- La cantidad máxima de unidades de ese producto que caben en dicha estantería correspondiente
- La cantidad mínima de unidades de ese producto que debe haber en dicha estantería
- Las existencias en un momento dado, en su estantería
- Una descripción del producto

Un diccionario puede archivar la situación en un instante dado, siendo la clave el código de cada producto. Las operaciones necesarias representarán las siguientes situaciones:

- Deseamos ver un producto del supermercado.
- Deseamos obtener un listado del inventario.
- Se reponen las existencias de un producto, dando su número y la cantidad de unidades que se reponen. Esta cantidad debe ser compatible con la cantidad máxima que admiten las estanterías.
- Un cliente compra una cierta cantidad de unidades de un producto. Como consecuencia, se modifica el inventario. Además, en caso de que las existencias queden por debajo del mínimo, se detectará la situación para avisar de que es necesario reponer las existencias.

Es fácil convertir este ejercicio en una pequeña colección de funciones y, luego, en una aplicación que:

- Lee la situación de un inventario, guardada en un archivo de texto.
- Lee un tíquet de compra, y va actualizando el inventario y avisando de las reposiciones que se van necesitando.

## Referencias

Algunos ejemplos de este script se han tomado de las siguientes URLs:

<https://docs.python.org/2/tutorial/datastructures.html#sets>

<https://docs.python.org/2/tutorial/datastructures.html#dictionaries>