Set

Set (conjuntos)

▼ Sintaxe (**{ }** e **set( )**)

- Sets podem ser definidos utilizando `{}` ou a função `set()`. Exemplo:

```
s1 = {1, 2, 3}
s2 = set([3, 4, 5])
print(s1, s2)  # Exibe: {1, 2, 3} {3, 4, 5}
```

▼ Definição e características

> ❓ Os **sets** são coleções não ordenadas de elementos únicos, ou seja, um **set** armazena valores que **não podem se repetir** e não têm uma ordem fixa. Eles são úteis em operações que envolvem comparação, remoção de duplicados ou teste de pertencimento

- Remoção de valores duplicados:
  - Sets eliminam automaticamente valores duplicados de um iterável. Exemplo:

```
lista = [1, 1, 2, 2, 3, 3]
cnjunto = set(lista)  # Remove duplicados
print(cnjunto)  # Exibe: {1, 2, 3}
```

- Valores únicos:
  - Um set **não permite elementos duplicados**. Se você tentar adicionar um valor já existente, ele será ignorado

```
s = {1, 2, 3}
s.add(2)  # Tentativa de adicionar um valor já existente
print(s)  # Exibe: {1, 2, 3}
```

- Não aceitam valores mutáveis:
  - Sets só podem conter elementos **imutáveis** (strings, números, tuplas)
  - Objetos como listas ou outros sets não podem ser elementos de um set

```
s = {1, 2, (3, 4)}  # Tuplas são permitidas
print(s)  # Exibe: {1, 2, (3, 4)}

# Não é permitido:
s = {1, [2, 3]}  # Erro: TypeError
```

- Não garantem ordem
  - A ordem dos elementos em um set **não é garantida**. Isso significa que o set pode apresentar os elementos em qualquer ordem. Exemplo:

```
s = {3, 1, 2}
print(s)  # Exibe: {1, 2, 3} ou {3, 1, 2}, depende do Python
```

  - Exemplo com string:

```
s = set("Gustavo")
print(s)  # Exibe: {'a', 'u', 's', 'o', 'v', 't', 'G'}
```

- São iteráveis (for, in, not in)
  - Você pode usar loops como `for`, além de operadores como `in` e `not in` para verificar se um elemento está no set. Exemplo:

```
s = {1, 2, 3}
print(2 in s)  # Exibe: True
print(4 not in s)  # Exibe: True
```

- Não tem índice
  - Não é possível acessar elementos de um set por índice como em listas. Você precisa iterar sobre o set para acessar seus elementos. Exemplo:

```
s = {1, 2, 3}
for elemento in s:
    print(elemento)
```

▼ Remover elementos repetidos de uma lista usando set

- É possível converter uma lista em um set para remover elementos repetidos com facilidade. Assim, ao criar uma nova lista com os valores desse set, obtém-se uma lista sem elementos repetidos. Exemplo:

```
l1 = [1, 2, 3, 3, 2, 1]
s1 = set(l1)
l2 = list(s1)
print(l2)  # Exibe: [1, 2, 3]
```

- O problema de realizar essa ação é que os sets não garantem manter a ordem da lista original. Ademais, sets não aceitam valores mutáveis

▼ Adicionar elemento (**add( )**)

- Adiciona um elemento ao set. Se o elemento já existir, nada acontece. Exemplo:

```
s = {1, 2, 3}

s.add(4)
print(s)  # Exibe: {1, 2, 3, 4}
```

```
s.add(3)
print(s)  # Exibe: {1, 2, 3, 4}
```

▼ Remover um elemento (**remove( )** e **discard( )**)

- **remove( )** serve para remove um elemento específico do set. Gera erro (KeyError) se o elemento **não** existir. Exemplo:

```
s = {1, 2, 3}

s.remove(2)
print(s)  # Exibe: {1, 3}

s.remove(5)
print(s)  # Erro: KeyError
```

- **discard( )** remove um elemento do set, mas **não gera erro** se o elemento não existir. Exemplo:

```
s = {1, 2, 3}

s.discard(2)
print(s)  # Exibe: {1, 3}

s.discard(5)
print(s)  # Exibe: {1, 3}
```

▼ Atualizar set (**update( )**)

- Serve para adiciona elementos de um iterável (lista, outro set, etc.) ao set. Exemplo:

```
s = {1, 2, 3}

s.update([3, 4, 5])  # Adiciona os elementos da lista
print(s)  # Exibe: {1, 2, 3, 4, 5}
```

▼ Remover todos os elementos (**clear( )**)

- Remove todos os elementos do set, deixando-o vazio. Exemplo:

```
s = {1, 2, 3}

s.clear()
print(s)  # Exibe: set()
```

▼ Operadores úteis + métodos equivalentes (**|**, **&**, **-**, **^**)

- União (`|`)
  - Combina todos os elementos de dois sets, eliminando duplicados. Exemplo:

```
s1 = {1, 2, 3}
s = {3, 4, 5}
resultado = s1 | s2
print(resultado)  # Exibe: {1, 2, 3, 4, 5}
```

  - *Também é equivalente ao método*: `s1.union(s2)`
- Interseção (`&`)
  - Retorna apenas os elementos presentes em ambos os sets. Exemplo:

```
s1 = {1, 2, 3}
s = {3, 4, 5}
resultado = s1 & s2
print(resultado)  # Exibe: {3}
```

  - *Também é equivalente ao método*: `s1.intersection(s2)`
- Diferença (`-`)
  - Retorna os elementos que estão no primeiro set, mas não estão no segundo. Exemplo:

```
s1 = {1, 2, 3}
s = {3, 4, 5}

resultado = s1 - s2
print(resultado)  # Exibe: {1, 2}
```

```
resultado = s2 - s1
print(resultado)  # Exibe: {4, 5}
```

  - *Também é equivalente ao método*: `s1.difference(s2)`

- Diferença Simétrica (`^`)
  - Retorna os elementos que estão em um dos sets, mas não em ambos. Em outras palavras, representa elementos que estão fora da interseção. Exemplo:

```
s1 = {1, 2, 3}
s = {3, 4, 5}
resultado = s1 ^ s2
print(resultado)  # Exibe: {1, 2, 4, 5}
```

  - *Também é equivalente ao método*: `s1.symmetric_difference(s2)`