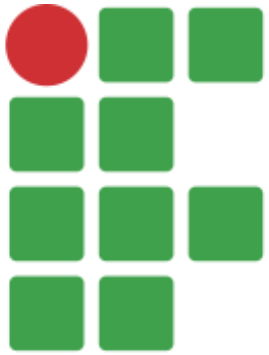


INSTITUTO FEDERAL
Paraíba

Campus Cajazeiras

ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

PROFESSORES LEANDRO, MICHEL
CAJAZEIRAS / IFPB

Set(s)

OBJETIVOS

- Compreender os tipos de dados em Set (conjuntos);
- Desenvolver algoritmos utilizando Set (conjuntos);

ROTEIRO

- Programação básica em Python (Coleções)
  - Tuplas
  - Listas
  - Dicionários
  - Conjuntos (set)

Dados estruturados

- Listas:
  - **As listas estão ordenadas** - as listas lembram a ordem dos itens inseridos.
  - **Acessado por índice** - os itens em uma lista podem ser acessados por meio de um índice.
  - **As listas podem conter qualquer tipo de objeto** - podem ser números , strings , tuplas e até outras listas.
  - **As listas são alteráveis (mutáveis)** - você pode alterar uma lista no local, adicionar novos itens e excluir ou atualizar itens existentes.
- Tupla:
  - **As tuplas estão ordenadas -** as tuplas mantêm uma ordem posicional da esquerda para a direita entre os itens que contêm.
  - **Acessado por índice -** os itens em uma tupla podem ser acessados por meio de um índice.
  - **As tuplas podem conter qualquer tipo de objeto -** podem ser números, strings, listas e até mesmo outras tuplas.
  - **As tuplas são imutáveis** - você não pode adicionar, excluir ou alterar itens após a tupla ser definida.

A **imutabilidade** da tupla é aplicável apenas ao nível superior da própria tupla, não ao seu conteúdo.

Dados estruturados

- Listas:
  - **As listas es**
  - **Acessado**
  - **As listas p**
  - **As listas sã**
    existentes.
- Tupla:
  - **As tuplas e**
    contêm.
  - **Acessado**
  - **As tuplas p**
  - **As tuplas s**

tas.

luir ou atualizar itens

re os itens que

utras tuplas.

```
1   #
2
3   # exemplo tupla
4
5   t1 = ([1,2,3], {'a':1,'b':2,'c':3})
6   print(t1)
7   t1[0][2] = 4
8   print(t1)
9   t1[0].append(5)
10  print(t1)
11  # imprime ([1, 2, 3], {'a': 1, 'b': 2, 'c': 3})
12  # imprime ([1, 2, 4], {'a': 1, 'b': 2, 'c': 3})
13  # imprime ([1, 2, 4, 5], {'a': 1, 'b': 2, 'c': 3})
14
```

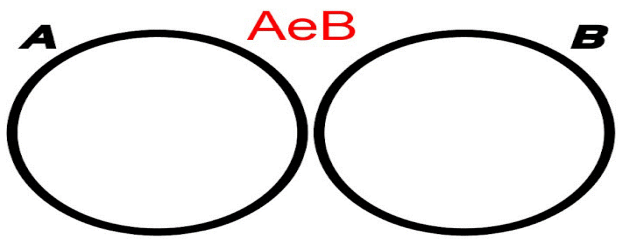
A **imutabilidade** da tupla é aplicável apenas ao nível superior da própria tupla, não ao seu conteúdo.

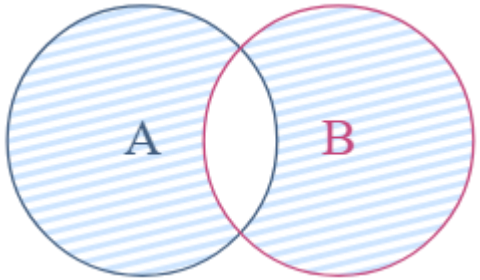
Dados estruturados

- Dicionário:
  - Os dicionários são a implementação do Python de uma estrutura de dados, geralmente conhecida como matrizes associativas (**lista de associações compostas**).
  - Podemos pensar em um dicionário como um mapeamento entre um conjunto de índices (conhecidos como **chaves**) e um conjunto de valores.
  - Cada chave mapeia para um valor. A associação de uma chave e um valor é chamada de **chave: objeto** ou às vezes um **item** .
  - **As chaves devem ser exclusivas** - Uma chave pode aparecer em um dicionário apenas uma vez.
  - Mesmo se especificar uma chave mais de uma vez durante a criação de um dicionário, o último valor dessa chave se torna o valor associado.
  - **A chave deve ser do tipo imutável** - podemos usar qualquer objeto de tipo imutável como chaves de dicionário - como números, strings, booleanos ou tuplas.

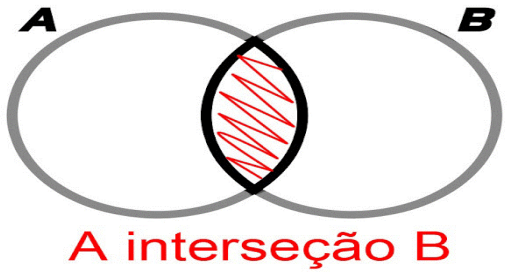
Set (conjunto)

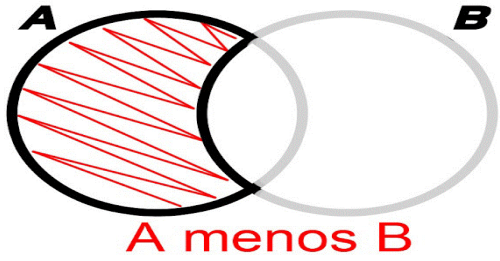
- O Set (conjunto) Python é uma coleção não ordenada de itens exclusivos.
- Eles são comumente usados para calcular operações matemáticas, como **união**, **interseção**, **diferenca** e **diferenca simétrica**.





Diferença simétrica
DICA: *"Une e arranca a interseção."*
Elementos que estão em ambos, mas não nos dois





A U B (união)
A ∩ B (interseção)
A – B (diferença)
A△B=(AUB)−(A∩B) (diferença simétrica)

https://blogdoenem.com.br/conjuntos-matematica-enem/

Set (conjunto)

- As propriedades importantes dos conjuntos Python são as seguintes:

  - **Conjuntos não ordenados** - os itens armazenados em um conjunto não são mantidos em uma ordem específica.
  - **Os itens definidos são únicos** - itens duplicados não são permitidos. (chave do dic?)
  - **Os conjuntos não são indexados** - você não pode acessar os itens dos conjuntos referindo-se a um índice.
  - **Os conjuntos são mutáveis** - eles podem ser alterados no local, podem aumentar e diminuir sob demanda.

Set

Exemplos

Sintaxe

conjunto = {'a', 'b', ..., 'z'}

Você pode criar um conjunto colocando uma sequência de itens separados por vírgulas entre chaves {}.

main.py

```
1  # criando um conjunto vazio
2  s1 = set()
3  print(s1)
4  print(type(s1))
5
```

```
set()
<class 'set'>
>
```

main.py

```
1  # criando com elementos
2  s2 = {1,2,3}
3  print(s2)
4
```

```
{1, 2, 3}
>
```

Set

Exemplos

O construtor **set()**

Podemos criar um conjunto usando um construtor de tipo chamado **set()**.

```
main.py
1  # Conjunto de itens em um iterável
2  s3 = set('abc')
3  print(s3)
4  # imprime {'a', 'b', 'c'}
5
```

```
{'c', 'b', 'a'}
>
```

```
main.py
1  # Conjunto de inteiros sucessivos
2  s4 = set(range(0, 4))
3  print(s4)
4  # imprime {0, 1, 2, 3}
5
6  # Converter lista em conjunto
7  s5 = set([1, 2, 3])
8  print(s5)
9  # imprime {1, 2, 3}
10
```

```
{0, 1, 2, 3}
{1, 2, 3}
>
```

Set (conjunto)

- Um conjunto em si é **mutável** (alterável), mas não pode conter objetos **mutáveis**. Portanto, objetos **imutáveis** como números, strings, tuplas podem ser um item definido, mas listas e dicionários **são mutáveis**,

main.py

```
1  # Conjunto de imutáveis
2  s6 = {1, 'abc', ('a', 'b'), True}
3  print(s6)
4  # imprime {1, 'abc', ('a', 'b')}
5
6  # Conjunto de mutáveis (não pode)
7  s7 = {[1, 2], {'a':1, 'b':2}}
8  # imprime: TypeError: unhashable type: 'list'
```

```
{1, 'abc', ('a', 'b')}
Traceback (most recent call last):
  File "main.py", line 7, in <module>
    s7 = {[1, 2], {'a':1, 'b':2}}
TypeError: unhashable type: 'list'
>
```

Adicionar itens a um conjunto

- Podemos adicionar um único item a um conjunto usando o método **add().**
- Podemos adicionar vários itens a um conjunto usando o método **update()**.

main.py

```
1  #
2  s1 = {'a', 'b', 'c'}
3  print(s1)
4  s1.add('d')
5  print(s1)
6  # imprime {'b', 'c', 'a'}
7  # imprime {'b', 'd', 'c', 'a'}
8  # a impressão é aleatória
9
```

```
{'b', 'c', 'a'}
{'b', 'd', 'c', 'a'}
```

main.py

```
1  #
2  s2 = {'a', 'b', 'c'}
3  print(s2)
4  s2.update(['d','e'])
5  print(s2)
6  # imprime {'c', 'a', 'b'}
7  #         {'b', 'd', 'c', 'a', 'e'}
8  # a impressão é aleatória
9
```

```
{'c', 'a', 'b'}
{'b', 'd', 'c', 'a', 'e'}
```

Remover itens de um conjunto

- Para remover um único item de um conjunto, use o método **remove()** ou **discard()**.

main.py

```
1  # método remove()
2  s3 = {'a', 'b', 'c'}
3  print(s3)
4  s3.remove('b')
5  print(s3)
6  # imprime {'c', 'a', 'b'}
7  #         {'c', 'a'}
8  # a impressão é aleatória
```

```
{'c', 'a', 'b'}
{'c', 'a'}
```

main.py

```
1  # método discard()
2  s4 = {'a', 'b', 'c'}
3  print(s4)
4  s4.discard('b')
5  print(s4)
6  # imprime {'c', 'b', 'a'}
7  #         {'c', 'a'}
8  # a impressão é aleatória
```

```
{'c', 'b', 'a'}
{'c', 'a'}
```

Remover itens de um conjunto

- **remover() vs descartar()**
- Ambos os métodos funcionam exatamente da mesma forma.
- A única diferença é que se o item especificado não estiver presente em um conjunto:
  - método **remove()** levanta KeyError
  - método **discard()** não faz nada

```
{'c', 'a', 'b'}
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    s4.remove('d')
KeyError: 'd'
>
```

main.py

```
1  # método discard()
2  s4 = {'a', 'b', 'c'}
3  print(s4)
4  s4.remove('d')
5  print(s4)
6  # imprime {'c', 'b', 'a'}
7  #         KeyError: 'd'
8  # a impressão é aleatória
9
```

Remover itens de um conjunto

- O método **pop()** remove o item aleatório de um conjunto.
- Use o método **clear()** para remover todos os itens do conjunto.

main.py

```python
# método pop()
s5 = {'a', 'b', 'c'}
print(s5)
s5.pop()
print(s5)
# imprime {'c', 'b', 'a'}
# imprime {'b', 'a'}
# a impressão é aleatória
```

```
{'c', 'b', 'a'}
{'b', 'a'}
```

main.py

```python
# método clear()
s6 = {'a', 'b', 'c'}
print(s6)
s6.clear()
print(s6)
# imprime {'a', 'b', 'c'}
# imprime set()
# a impressão é aleatória
```

```
{'a', 'b', 'c'}
set()
```

Encontrar o tamanho do conjunto

- Para descobrir quantos itens um conjunto possui, use o método **len()**.

main.py

```
1  # método len()
2  s7 = {'a', 'b', 'c'}
3  print(s7)
4  print(len(s7))
5  # imprime {'a', 'c', 'b'}
6  # imprime 3
7  # a impressão é aleatória
8
```

```
{'a', 'c', 'b'}
3
>
```

Iterar através de um conjunto

- Para iterar sobre os itens de um conjunto.

main.py

```
1  # iteração
2  s8 = {'a', 'b', 'c'}
3  print(s8)
4  for i in s8:
5      print(i)
6  # imprime {'a', 'c', 'b'}
7  # imprime a c b
8  # a impressão é aleatória
9
```

```
{'a', 'c', 'b'}
a
c
b
>
```

main.py

```
1  # iteração
2  s9 = {'a', 'b', 'c'}
3  print(s9)
4  if 'b' in s9:
5      print("sim")
6  # imprime {'a', 'c', 'b'}
7  # imprime sim
8  # a impressão é aleatória
9
```

```
{'a', 'c', 'b'}
sim
>
```

Operações de conjunto

- podemos realizar a **união** em dois ou mais conjuntos usando o método **union()** ou operador pipe " | ".

main.py

```python
# união
s1 = {'a', 'b', 'c'}
s2 = {'d', 'e', 'a'}
print(s1)
print(s2)
# operador pipe
print(s1 | s2)
# imprime {'c', 'b', 'a'}
# imprime {'a', 'd', 'e'}
# imprime {'a', 'd', 'e', 'c', 'b'}
# a impressão é aleatória
```

```
{'c', 'b', 'a'}
{'a', 'd', 'e'}
{'a', 'd', 'e', 'c', 'b'}
```

main.py

```python
# união
s1 = {'a', 'b', 'c'}
s2 = {'d', 'e', 'a'}
print(s1)
print(s2)
# método union
print(s1.union(s2))
# imprime {'a', 'c', 'b'}
# imprime {'a', 'e', 'd'}
# imprime {'c', 'd', 'a', 'b', 'e'}
# a impressão é aleatória
```

```
{'a', 'c', 'b'}
{'a', 'e', 'd'}
{'c', 'd', 'a', 'b', 'e'}
```

Operações de conjunto

- podemos realizar a **interseção** em dois ou mais conjuntos usando o método **intersection()** ou operador “ **&** “.

main.py

```
1  # interseção
2  s1 = {'a', 'b', 'c'}
3  s2 = {'d', 'e', 'a'}
4  print(s1)
5  print(s2)
6  # operador &
7  print(s1 & s2)
8  # imprime {'b', 'a', 'c'}
9  # imprime {'a', 'd', 'e'}
10 # imprime {'a'}
   ) é aleatória
```

```
{'b', 'a', 'c'}
{'a', 'd', 'e'}
{'a'}
```

main.py

```
1  # interseção
2  s1 = {'a', 'b', 'c'}
3  s2 = {'d', 'e', 'a'}
4  print(s1)
5  print(s2)
6  # método intersection()
7  print(s1.intersection(s2))
8  # imprime {'b', 'a', 'c'}
9  # imprime {'a', 'e', 'd'}
10 # imprime {'c', 'd', 'a', 'b', 'e'}
11 # a impressão é aleatória
12
```

```
{'b', 'a', 'c'}
{'a', 'e', 'd'}
{'a'}
```

Operações de conjunto

- podemos calcular a **diferença** entre dois ou mais conjuntos usando o método **difference**() ou operador “ **–** “.

main.py

```
1  # diferença
2  s1 = {'a', 'b', 'c'}
3  s2 = {'d', 'e', 'a'}
4  print(s1)
5  print(s2)
6  # operador -
7  print(s1 - s2)
8  # imprime {'a', 'b', 'c'}
9  # imprime {'d', 'e', 'a'}
10 # imprime {'b', 'c'}
11 # a impressão é aleatória
12
```

```
{'a', 'b', 'c'}
{'d', 'e', 'a'}
{'b', 'c'}
>
```

main.py

```
1  # diferença
2  s1 = {'a', 'b', 'c'}
3  s2 = {'d', 'e', 'a'}
4  print(s1)
5  print(s2)
6  # método difference()
7  print(s1.difference(s2))
8  # imprime {'c', 'b', 'a'}
9  # imprime {'d', 'e', 'a'}
10 # imprime {'c', 'b'}
11 # a impress
12
```

```
{'c', 'b', 'a'}
{'d', 'e', 'a'}
{'c', 'b'}
>
```

Operações de conjunto

- podemos calcular a **diferença simétrica** entre dois ou mais conjuntos usando o método **symmetric_difference()** ou operador " ^ ".

main.py

```
1  # diferença simétrica
2  s1 = {'a', 'b', 'c'}
3  s2 = {'d', 'e', 'a'}
4  print(s1)
5  print(s2)
6  # operador ^
7  print(s1 ^ s2)
8  # imprime {'b',
9  # imprime {'d',
10 # imprime {'b',
11 # a impressão é
12
```

```
{'b', 'c', 'a'}
{'d', 'a', 'e'}
{'b', 'd', 'c', 'e'}
```

main.py

```
1  # diferença simétrica
2  s1 = {'a', 'b', 'c'}
3  s2 = {'d', 'e', 'a'}
4  print(s1)
5  print(s2)
6  # método symmetric_difference()
7  print(s1.symmetric_difference(s2))
8  # imprime {'c', 'a', 'b'}
9  # imprime
10 # imprime
11 # a impress
12
```

```
{'c', 'a', 'b'}
{'e', 'd', 'a'}
{'e', 'd', 'b', 'c'}
```

Exercícios A

1 - ler um conjunto de 4 número e em seguida mostra a média.

2- ler um conjunto de 5 números e imprimir o menor e maior valor.

3 – dado uma lista de valores, verifique quanto vezes cada elemento se repete (use conjuntos).

```
l1 = ['a','b','b','a','c','c','c','d','d',’e']
```