Fundamentos da Linguagem Python I

Rodrigo Richard Gomes

O que veremos nessa unidade?

- Tipos de Dados
- Identificadores
- Variáveis
- Type casting
- Comentários
- Estruturas de Dados
  - Listas, tuplas, dicionários e conjuntos

Tipos de dados

Tipos de dados

- Marca
- Modelo
- Ano de fabricação
- Quantidade de portas
- Quantidade de lugares
- Comprimento (mm)
- Largura (mm)
- Cor
- Potência do motor
- Torque
- Preço
- Tipo de combustível
- Consumo de combustível
- Tipo da carroceria
- Velocidade máxima
- Possui ABS?
- Possui câmera de ré?
- Possui airbag?
- Quantidade de airbags

Tipos de dados

- Ano de fabricação
- Quantidade de portas
- Quantidade de lugares
- Comprimento (mm)
- Largura (mm)
- Quantidade de airbags

TIPO INTEIRO

Tipos de dados

- Potência do motor
- Torque
- Preço
- Consumo de combustível
- Velocidade máxima

TIPO REAL/PONTO FLUTUANTE

Tipos de dados

- Marca
- Modelo
- Cor
- Tipo de combustível
- Tipo da carroceria

TIPO CARACTERE/STRING

Tipos de dados

- Possui ABS?
- Possui câmera de ré?
- Possui airbag?

TIPO LÓGICO/BOOLEANO

Tipos de dados

- Em programação, os tipos de dados são elementos fundamentais que compõem a estrutura de um programa
- Eles determinam o tipo de informação que uma variável pode armazenar e como essa informação pode ser manipulada

Tipos de dados

- Em Python, existem dois grupos de tipos de dados:
  - Os tipos de dados primitivos (simples)
  - Os tipos de dados compostos (estruturas de dados)

Tipos de dados

- Os tipos de dados primitivos (ou tipos simples) em Python são aqueles que armazenam um único valor simples
- Eles incluem inteiros, números de ponto flutuante, booleanos e strings

Tipos de dados

- Os inteiros são números inteiros sem parte decimal
- Os números de ponto flutuante são números com casas decimais
- Os booleanos são valores lógicos que representam verdadeiro ou falso
- As strings são sequências de caracteres

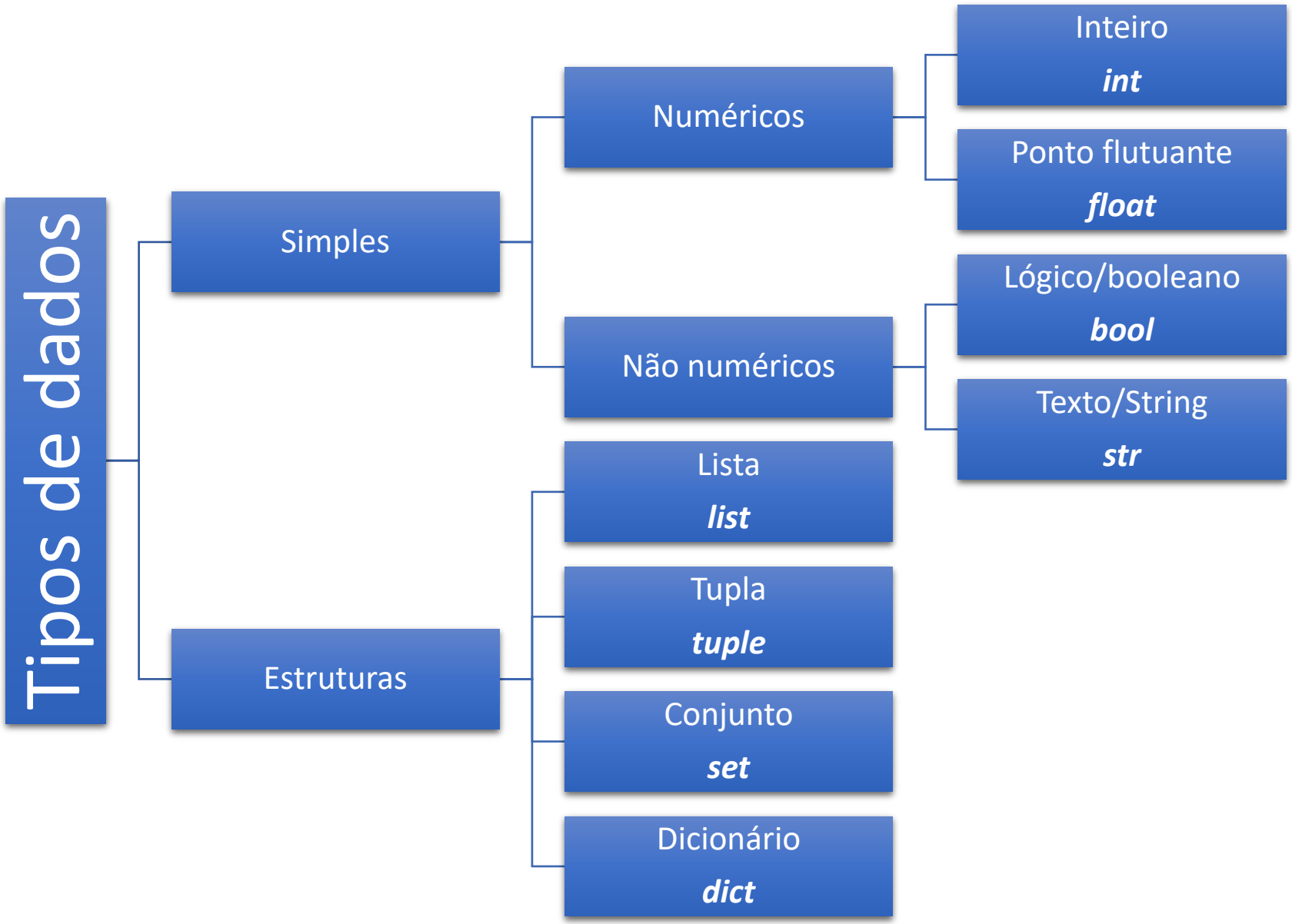
Tipos de dados

- Já os tipos de dados compostos em Python (ou estruturas de dados) são aqueles que podem armazenar vários valores em uma única variável
- Eles incluem listas, tuplas, dicionários e conjuntos

Tipos de dados

- As listas são sequências ordenadas de valores que podem ser modificadas
- As tuplas são sequências ordenadas de valores imutáveis
- Os dicionários são coleções de pares chave-valor
- Os conjuntos são coleções não ordenadas de elementos únicos

Tipos de dados



Tipos de dados

- É importante conhecer os tipos de dados em Python para poder trabalhar com eles de forma eficiente e correta
- Saber a diferença entre tipos de dados primitivos e compostos permite escolher o tipo de dado mais apropriado para cada situação, melhorando a performance do programa e evitando erros

Tipos de dados

- Além disso, o conhecimento dos tipos de dados permite manipular as informações de forma adequada, garantindo a integridade e precisão dos dados manipulados pelo programa

Tipos de dados

- Nome
- Idade
- Peso
- Altura
- Sexo
- CPF
- RG
- Salário
- Cor dos olhos
- Estado civil
- Você é casado?
- Número do sapato
- Você tem CNH?
- Data de nascimento
- Número de telefone
- Tipo sanguíneo

Tipos de dados

- Nome *string*
- Idade *inteiro*
- Peso *real*
- Altura *real*
- Sexo *string*
- CPF *string*
- RG *string*
- Salário *real*
- Cor dos olhos *string*
- Estado civil *string*
- Você é casado? *booleano*
- Número do sapato *inteiro*
- Você tem CNH? *booleano*
- Data de nascimento *string*
- Número de telefone *string*
- Tipo sanguíneo *string*

Identificadores

Identificadores

Em Python, um **Identificador** é um nome atribuído a uma variável, função, classe ou outros objetos do programa

Identificadores

Regras para formação de identificadores

- O nome do identificador pode conter letras, números e o caractere "_"
- O nome do identificador deve começar com uma letra ou com o caractere "_"
- O nome do identificador não pode começar com um número

Identificadores

Regras para formação de identificadores

- O nome do identificador é sensível a maiúsculas e minúsculas, ou seja, "***Nota***", "***NOTA***" e "***nota***" são identificadores diferentes
- O nome do identificador não pode ser uma palavra-chave reservada da linguagem, como "***if***", "***for***", "***while***", entre outras

Identificadores

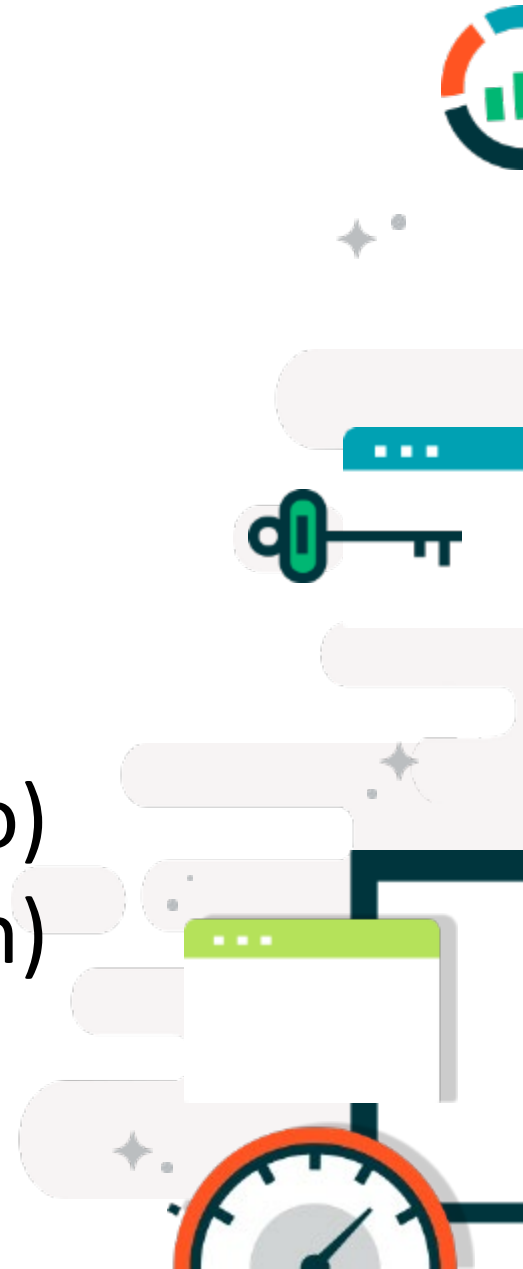
Identificadores válidos

- i
- _J
- FRUTA
- salario
- A12
- i23ER21

Identificadores

Identificadores inválidos

- 25 (começa com um número)
- nome-fruta (contém um caractere inválido)
- 5Jose (começa com um número)
- A$1 (contém um caractere inválido)
- Nome do aluno (contém um caractere inválido)
- Break (palavra reservada da linguagem Python)



Identificadores

- A boa prática de programação nos diz que um identificador deve ser não apenas válido, mas também significativo/adequado

- Um identificador **deve descrever precisamente** o dado que a variável armazena

Identificadores

Identificadores adequados

- salario
- idade
- total_compra
- nome_aluno
- dtnasc
- preco_produto
- complemento_endereco_correspondencia

Identificadores

Identificadores inadequados

- a
- jose5
- xyz
- i23ER21
- mfpxy
- fruta

Identificadores

Convenção para identificadores

No PEP8, Guido van Rossum sugere que nomes de variáveis e funções devem ter apenas letras minúsculas e utilizar o underscore no caso de identificadores contendo mais de uma palavra

(ex: nome_func, perc_aumento)

Variáveis

Variáveis

- **Variável** é um espaço na memória do computador, utilizado para que um programa possa armazenar dados

- Variáveis são usadas para guardar valores que serão usados mais tarde pelo programa

Variáveis

Atributos de uma variável

- Endereço na memória
- Tipo
  - Inteiro (int), real (float), lógico (bool) ou string (str)
- Identificador
  - Apenas letras, números e *underscore*, iniciando com letra ou *underscore*

Variáveis

- Para criar uma variável em Python basta atribuir um valor conforme a sintaxe abaixo

```
variavel = expressão
```

- `expressão` pode ser um valor, uma outra variável, uma expressão aritmética ou o retorno de uma função
- O tipo da variável é definido dinamicamente conforme o tipo do valor atribuído

Variáveis

Exemplo de criação de variáveis

```
In [1]: qtde = 5  # cria uma variável inteira (int)

In [2]: type(qtde)
Out[2]: int

In [3]: preco_produto = 125.50  # cria uma variável real (float)

In [4]: type(preco_produto)
Out[4]: float

In [5]: nome = "Maria"  # cria uma variável string (str)

In [6]: type(nome)
Out[6]: str

In [7]: tem_casa = True  # cria uma variável lógica (bool)

In [8]: type(tem_casa)
Out[8]: bool

In [9]: total_compra = qtde * preco_produto # criando a variável total_compra

In [10]: total_compra
Out[10]: 627.5
```

Type Casting
(Conversão de tipos)

Comentários

Comentários

```
# Comentário de bloco
comando_1
comando_3
...
comando_n
```

```
comando_x  # comentário de linha
```

```
# Um comentário pode ser longo e
# ocupar várias linhas.
```

```
"""
Outra forma de se criar comentários de várias
linhas é através da utilização de 3 aspas duplas
na abertura e no fechamento do comentário """
```

Estruturas de Dados
Lista

Listas

- Lista é uma coleção de elementos
- Semelhante ao **vetor** de outras linguagens de programação (exceto pelo fato de seu tamanho dinâmico)

Listas

- Cada elemento possui uma posição dentro de uma lista. Essa posição é chamada índice
- O primeiro elemento fica armazenado no índice **0** e o último elemento no índice **n-1** (*n é a quantidade de elementos da lista*)

Listas

- Cada elemento possui uma posição dentro de uma lista. Essa posição é chamada índice
- O primeiro elemento fica armazenado no índice **0** e o último elemento no índice **n-1** (*n é a quantidade de elementos da lista*)

0	1	2	3	4	5	6	7	8
23	**56**	**2**	**-3**	**10**	**2**	**18**	**0**	**5**

Listas

- Cada elemento possui uma posição dentro de uma lista. Essa posição é chamada índice
- O primeiro elemento fica armazenado no índice **0** e o último elemento no índice **n-1** (*n é a quantidade de elementos da lista*)

0	1	2	3	4	5	6	7	8
23	**56**	**2**	**-3**	**10**	**2**	**18**	**0**	**5**
-9	-8	-7	-6	-5	-4	-3	-2	-1

Listas

1. Como criar uma lista
2. Como acessar elementos de uma lista
3. Como modificar elementos de uma lista
4. Principais métodos de um objeto lista
5. Funções aplicáveis a uma lista

Listas - métodos

Método	Descrição	Exemplo
append	Adiciona um elemento no final da lista	`lista.append(5)`
clear	Apaga todos os elementos de uma lista	`lista.clear()`
copy	Retorna uma cópia dos elementos da lista	`copia = lista.copy()`
count	Retorna a quantidade de ocorrências de um elemento na lista	`qt = lista.count(5)`
extend	Adiciona os elementos de outra lista passada por parâmetro	`lista.extend(outra_lista)`
index	Retorna o índice do elemento passado por parâmetro (primeira posição)	`pos5 = lista.index(5)`
insert	Adiciona um elemento em uma posição passada por parâmetro (adiciona no final caso a posição não exista)	`lista.insert(3, "João")`
pop	Remove o elemento na posição passada por parâmetro (provoca um erro caso a posição não exista)	`elemento = lista.pop(3)`
remove	Remove o elemento passado por parâmetro (provoca um erro caso o elemento não exista)	`lista.remove(5)`
reverse	Inverte a ordem dos elementos de uma lista	`lista.reverse()`
sort	Ordena os objetos de uma lista	`lista.sort()`
		`lista.sort(reverse=True)`

Listas – algumas funções

Função	Descrição	Exemplo
len	Retorna a quantidade de elementos de uma lista	`print(len(lista))`
max	Retorna o maior elemento de uma lista	`maior = max(lista)`
min	Retorna o menor elemento de uma lista	`menor = min(lista)`
sum	Retorna o somatório dos elementos de uma lista	`soma = sum(lista)`

Estruturas de Dados
Tupla

Tuplas

- Assim como a **Lista**, uma **Tupla** em Python é uma coleção de elementos
- A principal diferença entre essas estruturas é a propriedade de imutabilidade das tuplas
- Isso significa que, uma vez definidos os seus elementos, a tupla não pode mais ser alterada

Tuplas

- Cada elemento possui uma posição dentro de uma tupla. Essa posição é chamada índice
- O primeiro elemento fica armazenado no índice **0** e o último elemento no índice **n-1** (*n é a quantidade de elementos da tupla*)

Tuplas

- Cada elemento possui uma posição dentro de uma tupla. Essa posição é chamada índice
- O primeiro elemento fica armazenado no índice **0** e o último elemento no índice **n-1** (*n é a quantidade de elementos da tupla*)

0	1	2	3	4	5	6	7	8
23	**56**	**2**	**-3**	**10**	**2**	**18**	**0**	**5**

Tuplas

- Cada elemento possui uma posição dentro de uma tupla. Essa posição é chamada índice
- O primeiro elemento fica armazenado no índice **0** e o último elemento no índice **n-1** (*n é a quantidade de elementos da tupla*)

0	1	2	3	4	5	6	7	8
23	**56**	**2**	**-3**	**10**	**2**	**18**	**0**	**5**
-9	-8	-7	-6	-5	-4	-3	-2	-1

Tuplas

1. Como criar uma tupla
2. Como acessar elementos de uma tupla
3. Imutabilidade de uma tupla
4. Principais métodos de um objeto tupla
5. Funções aplicáveis a uma tupla
6. Convertendo uma tupla em lista

Estruturas de Dados
Dicionário

Dicionários

- Listas e tuplas associam uma informação (valor/elemento) a um índice, de forma automática
- Em um dicionário, nós armazenamos uma informação (***valor***) que queremos localizar e recuperar posteriormente através de uma ***chave***
- Assim, o principal propósito de um dicionário é associar um valor (***value***) a uma chave (***key***)

Dicionários

- Um dicionário em Python é então uma coleção de pares de ***chave/valor***
- Cada chave está associada a um valor, e através dessa chave acessamos o valor associado a ela

Dicionários

- Nesse exemplo temos um dicionário onde a chave é a sigla de um Estado brasileiro, e o valor é o nome completo desse Estado

Key	Value
AC	Acre
AL	Alagoas
AM	Amazonas
AP	Amapá
BA	Bahia
...	...
RS	Rio Grande do Sul
SC	Santa Catarina
SE	Sergipe
SP	São Paulo
TO	Tocantins

Dicionários

1. Como criar um dicionário
2. Como acessar um valor em um dicionário
3. Como modificar valores
4. Principais métodos de um objeto dicionário

Dicionários - métodos

Método	Descrição	Exemplo
clear	Apaga todos os elementos de um dicionário	`dic.clear()`
copy	Retorna uma cópia dos elementos de um dicionário	`dic_copia = dic.copy()`
fromkeys	Retorna um dicionário a partir de uma sequência de chaves	`exemplo = dic.fromkeys({"k1", "k4", "k5"})`
get	Retorna o valor associado a uma chave	`dic.get("k1")`
		`dic.get("k1", 0)`
items	Retorna uma visão dos pares chave/valor de um dicionário	`dic.items()`
keys	Retorna uma visão das chaves de um dicionário	`dic.keys()`
pop	Remove e retorna o elemento associado à chave passada por parâmetro (provoca erro se a chave não existir)	`dic.pop(2)`
popitem	Remove e retorna o último elemento adicionado ao dicionário	`dic.popitem()`
setdefault	Retorna o valor associado a uma chave (se a chave existir). Caso não exista, insere a chave com o valor (opcional) no dicionário	`dic.setdefault(2, "dois")`
update	Adiciona ao dicionário os pares de chave/valor de outro dicionário passado por parâmetro	`dic.update(dic2)`
values	Retorna uma visão dos valores de um dicionário	`dic.values`

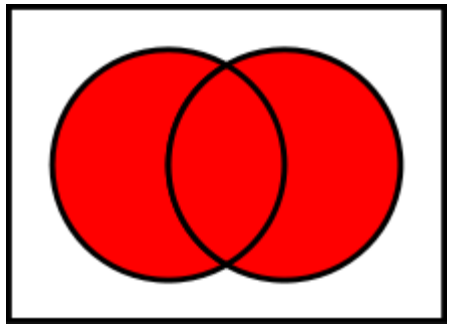
Estruturas de Dados
Conjunto

Conjuntos

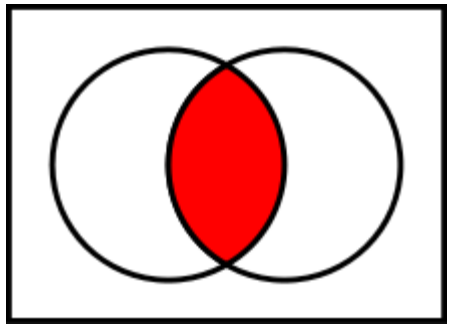
- Conjuntos são coleções de elementos **únicos**
- Principais características:
  - Os elementos não são armazenados em uma ordem específica
  - Conjuntos não contém elementos repetidos
- Conjuntos não suportam indexação como as listas e tuplas

Conjuntos

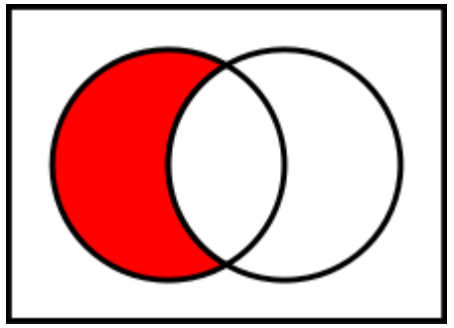
- Conjuntos fornecem métodos para as operações mais conhecidas de teoria dos conjuntos, como união, interseção e diferença, além de outras



União



Interseção



Diferença

Fonte das imagens: Wikipedia

Conjuntos

1. Como criar um conjunto
2. Como acessar um valor em um conjunto
3. Como modificar elementos
4. Principais métodos de um objeto conjunto
5. Funções aplicáveis a um conjunto

Conjuntos

Método	Descrição	Exemplo
add	Adiciona um elemento ao conjunto	`conj1.add(5)`
clear	Apaga todos os elementos de um conjunto	`conj1.clear()`
copy	Retorna uma cópia dos elementos de um conjunto	`conj_copia = conj1.copy()`
difference	Retorna o conjunto de elementos de conj1 que não pertencem a conj2	`conj1.difference(conj2)`
intersection	Retorna o conjunto de elementos presentes tanto em conj1 quanto em conj2	`conj1.intersection(conj2)`
isdisjoint	Retorna True se conj1 e conj2 forem disjuntos (ou seja, não possuem elementos em comum)	`conj1.isdisjoint(conj2)`
issubset	Retorna True se conj1 for um subconjunto de conj2 (conj1 está contido em conj2)	`conj1.issubset(conj2)`
issuperset	Retorna True se conj1 for um superconjunto de conj2 (conj1 contém conj2)	`conj1.issuperset(conj2)`
remove e discard	Removem um elemento do conjunto	`conj1.remove(5)` ou `conj1.discard(5)`
simmetric_difference	Retorna o conjunto de elementos que não são comuns aos dois conjuntos (o contrário da interseção)	`conj1.simmetric_difference(conj2)`
union	Retorna a união entre 2 conjuntos	`conj1.union(conj2)`