**Ciência de Dados com Python**

# Princípios de Python e Versionamento de Código com Git e GitHub

## Ambiente de Desenvolvimento e Primeiros Passos com Phyton

**Objetivos do Python:**

* Uma linguagem fácil e intuitiva;
* Código aberto, para que todos possam contribuir;
* Código tão inteligível quanto inglês;
* Adequada para tarefas diárias, e produtiva.

**Características da Linguagem:**

* Tipagem dinâmica e forte;
* Multiplataforma e multiparadigma;
* Comunidade gigante e ativa;
* Curva de aprendizado baixa.

**No terminal:**

* “python” + ‘Enter’: terminal interativo (para sair é ‘exit()’)
* Python -i nomeDoArquivo: executa o arquivo passado

## Conhecendo a Linguagem de Programação Python

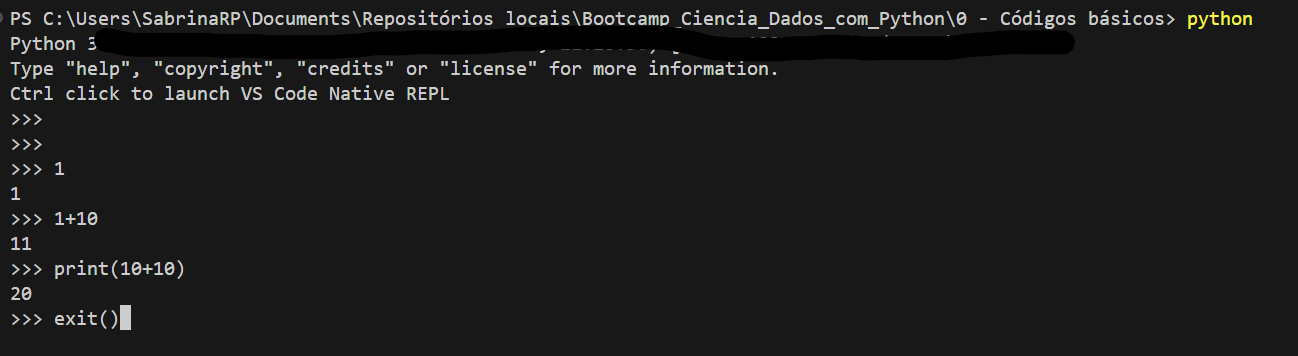
**Tipo de Dados:**

* Texto – str;
* Numéricos - int, float, complex;
* Sequência – list, tuple, range;
* Mapa – dict (chave: valor);
* Coleção – set, fronzenset;
* Booleano – bool;
* Binário – bytes, bytearray, memoryview;

**Modo interativo do terminal**

Programar/Teste rápido e ir vendo o resultado no terminal

Digitar “python” ou “python -i nome\_do\_programa.py”(esse útimo executa o programa passado e fica no modo interativo) no terminal. Para sair digitar “exit()”



* **Dir:** Sem argumento, retorna a lista de nomes no escopo local atual. Com um argumento, retorna uma lista de atributos válidos para o objeto. Exemplo
  + dir();
  + dir(100): lista de métodos dentro da classe int
* **Help:** Invoca o sistema de ajuda integrado. É possível fazer buscas em modo interativo ou informar por parâmetro qual o nome do módulo, função, classe, método ou variável. Exemplo:
  + help(): você consegue buscar o método e ele retorna todos os métodos e utilidades daquilo que foi buscado;
  + help(100): aqui vai trazer as informações sobre os métodos da classe int

**Variáveis e Constantes**

**Variáveis:** seu valor muda ao longo do programa. Definição: variável = valor

**Constantes:** Python não tem uma palavra reservada para isso. Constantes não muda o valor, uma vez atribuído um valor, ele sempre será. Exemplo: um programa sobre dados pessoais, o tipo sanguíneo será um valor constante. Convenççao: utiliza-se o nome da variável, por convenção, deve ser escrito todo em maiúsculo, VARIAVEL = valor.

**Boas práticas:**

* Padrão de nomes deve ser o snake case: substituir espaços pelo *underline*;
* Escolher nomes sugestivos;
* Nome de constantes todo em maiúsculo.

**Conversões de tipos:**

- Bootcamp\_Ciencia\_Dados\_com\_Python\0 - Códigos básicos\exe2\_conversao\_valores.py

//Quando a conversão não é possível, um erro é lançado:

**Funções de Entrada(input) e Saída(output):**

- Bootcamp\_Ciencia\_Dados\_com\_Python\0 - Códigos básicos\exe3\_entrada\_saida\_dados.py

## Versionamento de Código com Git e GitHub

-

# Introdução à Ciência de Dados e Python

## Tipos de Operadores com Python

**Operadores aritméticos:**

* + (adição)
* - (subtração)
* \* (multiplicação)
* / (divisão)
* // (dicisão onde mostra somente a parte inteira do resultado)
* % (módulo)
* \*\* (exponenciação)

Precedência de operações segue as regras da matemática, logo:

* Parêntesis
* Expoentes
* Multiplicação e divisão(da esquerda para a direita)
* Soma e subtração(da esquerda para a direita)

**Operadores de comparação:** são operadores utilizados para compara valores.

* == (igual)
* != (diferente)
* > (maior)
* >= (maior ou igual)
* < (menor)
* <= (menor ou igual)

**Operador de atribuição:** iniciar o valor de variável ou sobrescrever

* valor = 0 (valor recebe 0)
* valor += 200 (valor recebe valor + 200)

**Operadores lógicos:** usados com o de comparação cria-se sentenças

print("True and True: ", True and True)#True

print("False and False: ", False and False)#False

print("True and False: ", True and False)#False

print("True or False:", True or False)#True

print("True or True:", True or True)#True

print("False or False:", False or False)#False

**Operadores de identidade:** saber se as variáveis estão no mesmo espaço de memória

* Is (é)
* Is not (negação)

**Operadores de associação:** são operadores utilizados para verificar se um objeto está presente em uma sequência.

//Esses operadores são case sensitive, ou seja se está escrito com maiúscula ou minúscula faz diferença

* in (está)
* not in (não está)

## Estruturas Condicionais e de Repetição em Python

if / if else / elif: a estrutura condicional permite o desvio de fluxo de controle, quando determinadas expressões lógicas são atendidas.

* **If:** testa a condição lógica e da um retorno.
* **if-else:** torna possível a execução de dois desvios, ou seja, saber o que fazer em duas situações diferentes(contrárias).
* **if-elif-else:** torna possível o teste de vários desvios, ou seja, testa várias condições diferentes.
* **Comando for:** usado para percorrer um objeto iterável. Faz sentido usar ‘for’ quando sabemos o número exato de vezes que o bloco de código deve ser executado, ou quando queremos percorrer um objeto iterável.
* **Função Range:** é uma função built-in(própria) do Pyhton, ela é usada para produzir uma sequência de números inteiros a partir de um início(inclusivo, ou seja, está incluso na contagem) para um fim(exclusivo, ou seja, não aparece na contagem. Se o start e stop for(0,10) ele vai contar de 0 a 9).

Se usarmos(i,j) será produzido:

I, i+1, i+2, i+3, ..., j-1.

Ela recebe 3 argumentos: stop(obrigatório), start(opcional) e step(opcional).

* **Comando while:** usado para repetir um bloco de código várias vezes. Faz sentido usar o while quando não sabemos o número exato de vezes que o bloco de código deve ser executado. O fim da execução é determinado por uma condição.

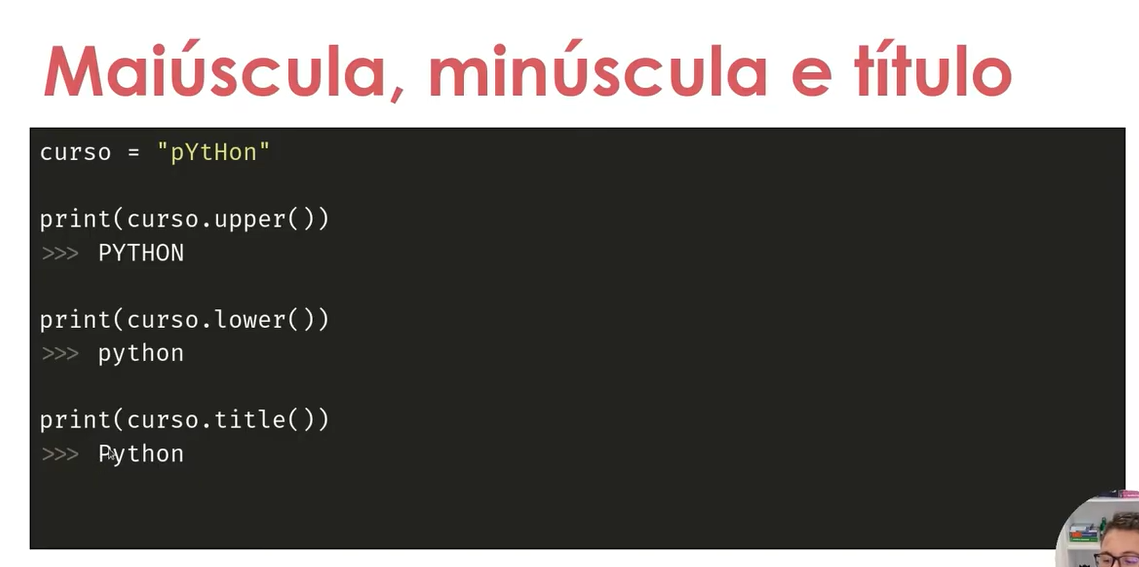
**//break:** palavra reservada que interrompe o loop a qualquer momento, mesmo que ele não tenha terminado ou a condição de parada não tenha sido satisfeita. Não é muito recomendada...

* **Comando continue:** comando utilizado para pular um ciclo de iteração dos comandos for e while.

## Manipulando Strings com Python

**Conhecendo métodos úteis:**

* Maiúsculas, minúsculas e título:

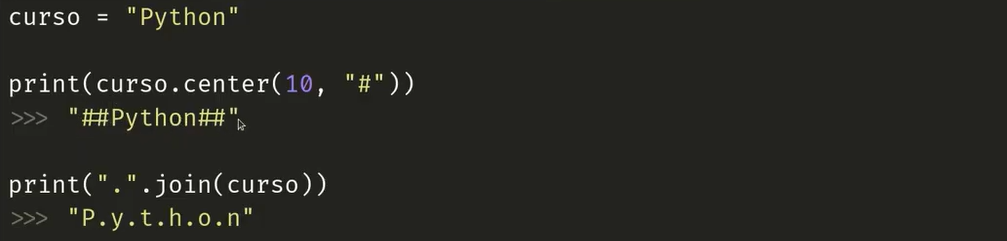
****

* Eliminando espaços em branco:

****

* Junções e centralização -> ex: função center precisa de 2 argumentos, o primeiro é a quantidade de caracteres que a variável vai ter, o segundo é opicional e reprensenta o que você quer colocar nos espaços, se nada for informado, vao ficar o espaço em branco.

Na função join, ele junta os itens com algo que for passado.

****

**Interpolação de variáveis:**

Em Python existe 3 formas de interpolar variáveis em strings:

* Usando o sinal de ‘%’ -> igual na linguagem C;
* Utilizando o método ‘format’;
* Utilizando ‘f strings’

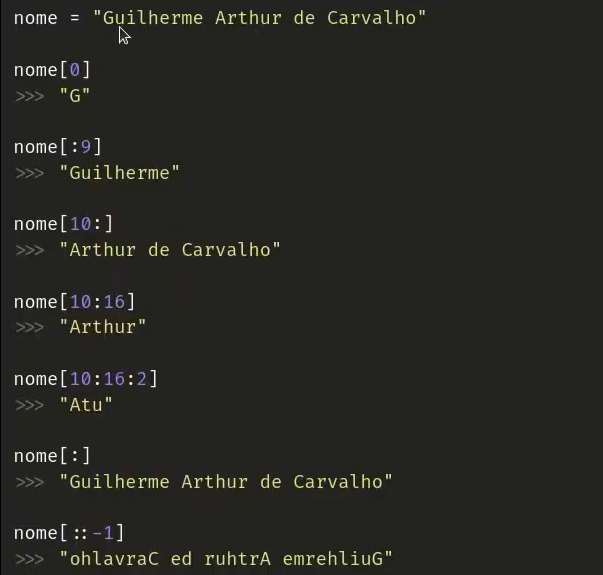
// a primeira forma não é recomendada...

- Bootcamp\_Ciencia\_Dados\_com\_Python\0 – Códigos básicos\exe6.1\_manipulacao\_strings\_interpolacao.py

**Fatiamento de string:**

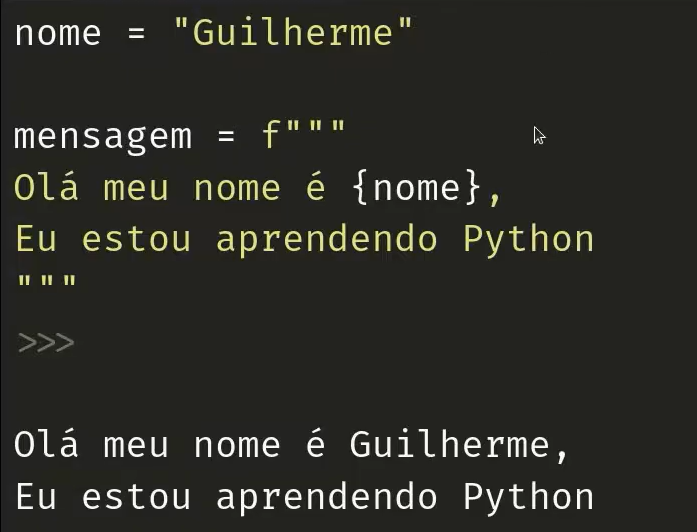
Fatiamento de strings é uma técnica utilizada para retornar substrings (partes da string original), informando inicio(start), fim(stop) e passo(step): [start:stop:step].

//A contagem da string começa a partir de 0

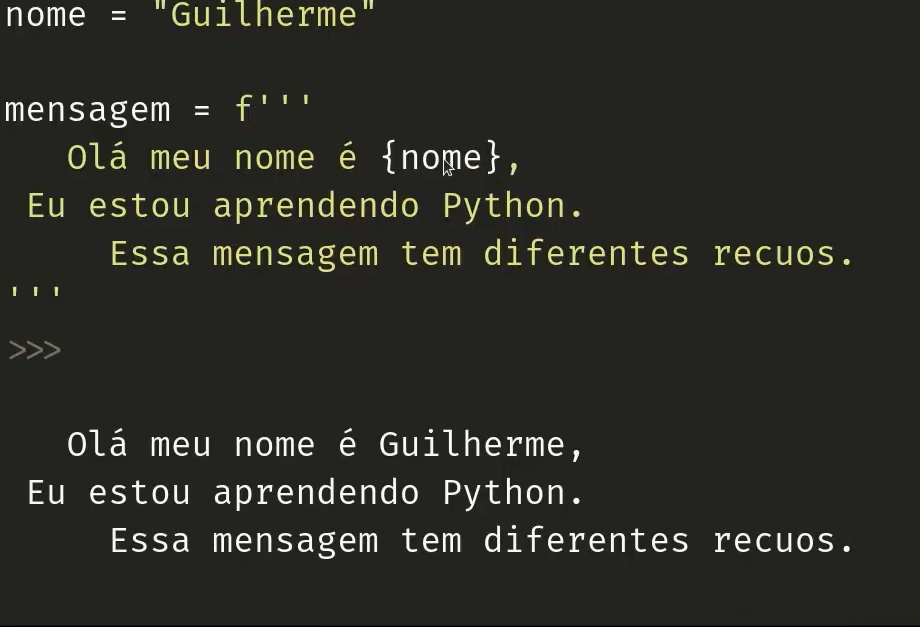


**Strings múltiplas linhas:**

Strings de múltiplas linhas (ou strings triplas) são definidas informando 3 aspas simples ou duplas durante a atribuição. Elas podem ocupar várias linhas do código, e todos os espaços em brancos são incluídos na string final.



//Perceba nesse novo exemplo como os espaços são preservados



<https://docs.python.org/pt-br/3/library/string.html>

<https://docs.python.org/pt-br/3/library/stdtypes.html#textseq>

## Trabalhando com Listas em Python

Listas em Pyhton podem armazenar de maneira sequencial qualquer tipo de objeto. Podemos criar listas utilizando o construto **list**, a função range ou colocando valores separados por vírgula dentro de colchetes. Listas são objetos mutáveis, portanto podemos alterar seus valores após a criação.

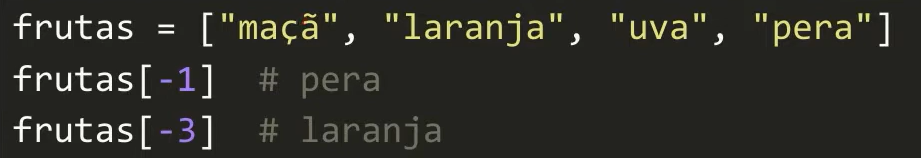


A lista é uma sequência, portanto podemos acessar seus dados utilizando índices. Esse acesso é chamado de **Acesso direto**. Contamos o índice de determinada sequência a partir de zero.



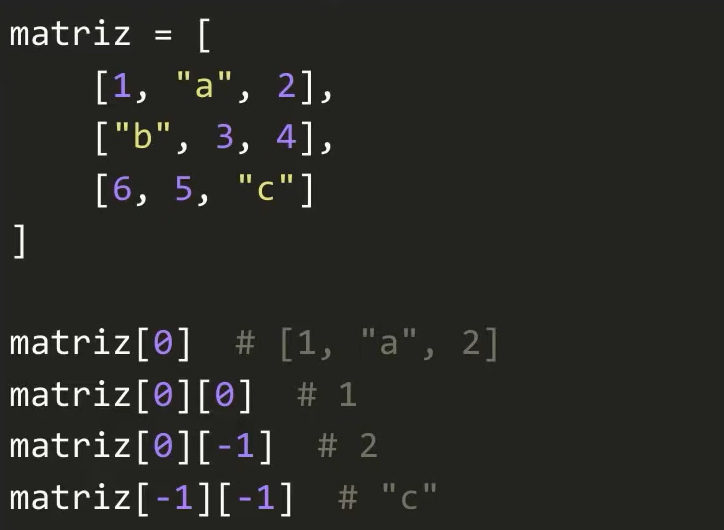
Qualquer tipo de sequência em Python é possível ser acessado utilizando índices negativos também, de trás para a frente.

//Quando se trata da contagem negativa o índice começa em **-1**

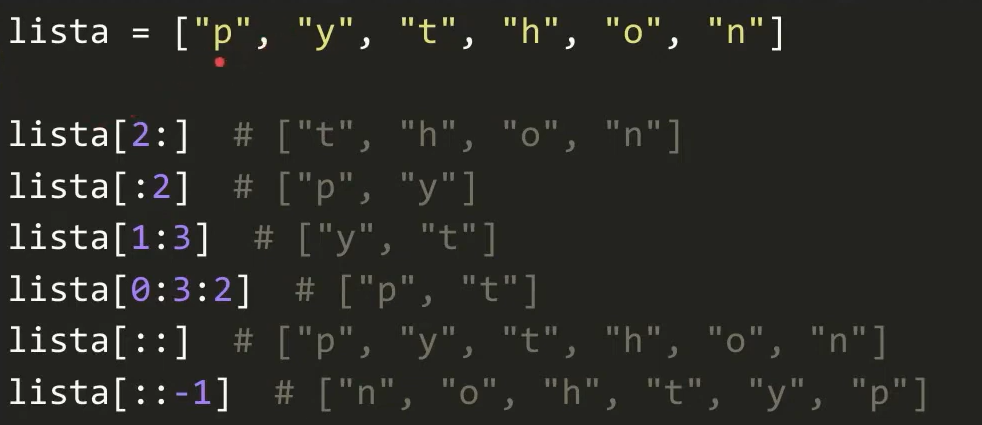


Listas podem armazenar todos os tipos de objetos Python, portanto podemos ter listas que armazenam outras listas(chamado de **Listas aninhadas**). Com isso podemos criar estruturas bidimensionais(tabelas), e acessar informando os índices de linha e coluna.

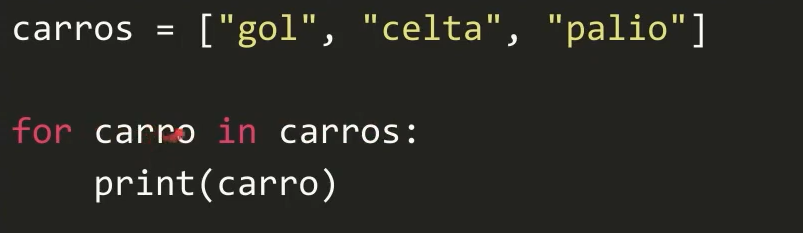
//Exemplo de matriz 3X3



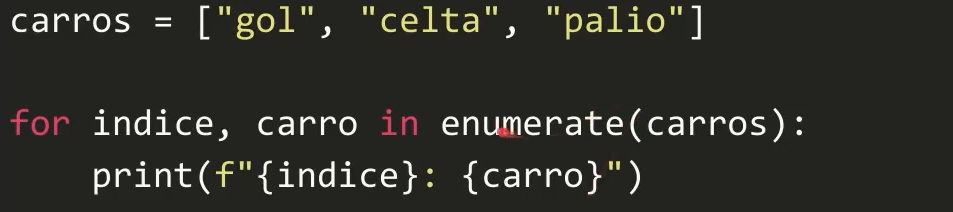
Em listas, além de acessar elementos diretamente, podemos extrair um conjunto de valores de uma sequência(**Fatiamento**). Para isso basta passar índice inicial e/ou final para acessar o conjunto. Podemos ainda informar quantas posições o cursor deve ‘pular’ no acesso.



A forma mais comum para percorrer os dados de uma lista é utilizando o comando **for.**

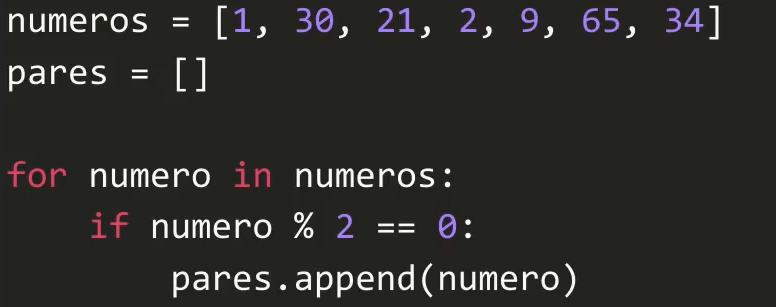
****

Às vezes é necessário saber qual o índice do objeto dentro do laço **for**. Para isso podemos usar a função **enumerate**.

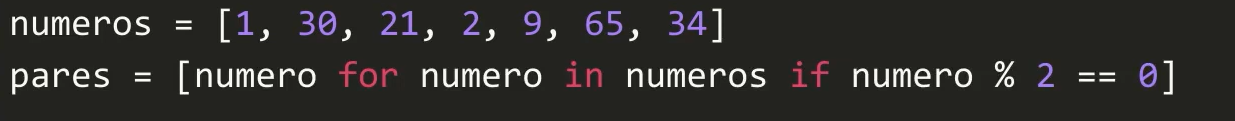


A **Compreensão de lista** oferece uma sintaxe mais curta quando você deseja: criar uma nova lista com base nos valores de uma lista existente(filtro) ou gerar uma nova lista aplicando alguma modificação nos elementos de uma lista existente.

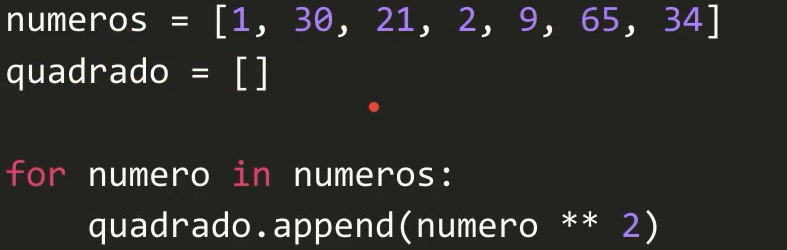
Exemplo 1 - modo ‘normal’



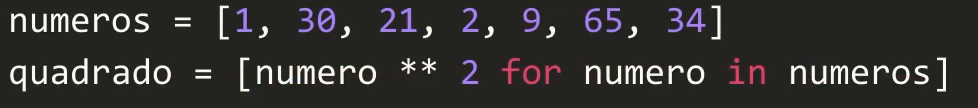
Exemplo 1 - usando o **Comprehensions**



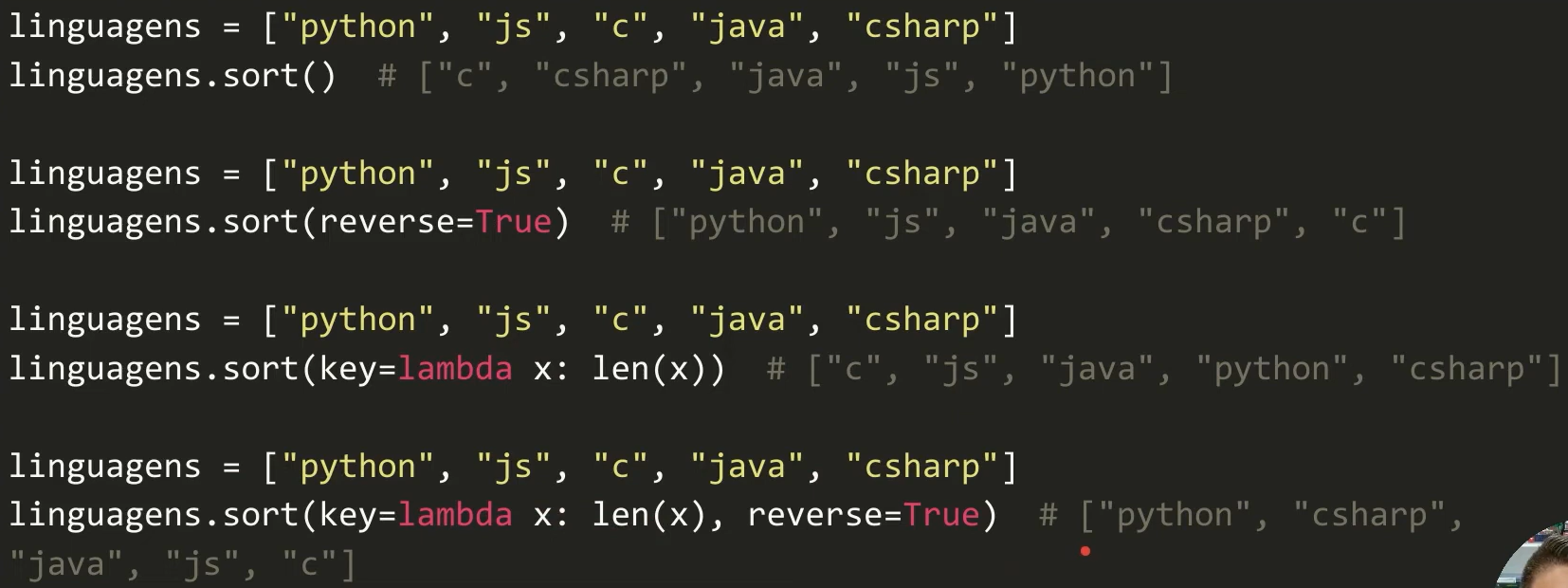
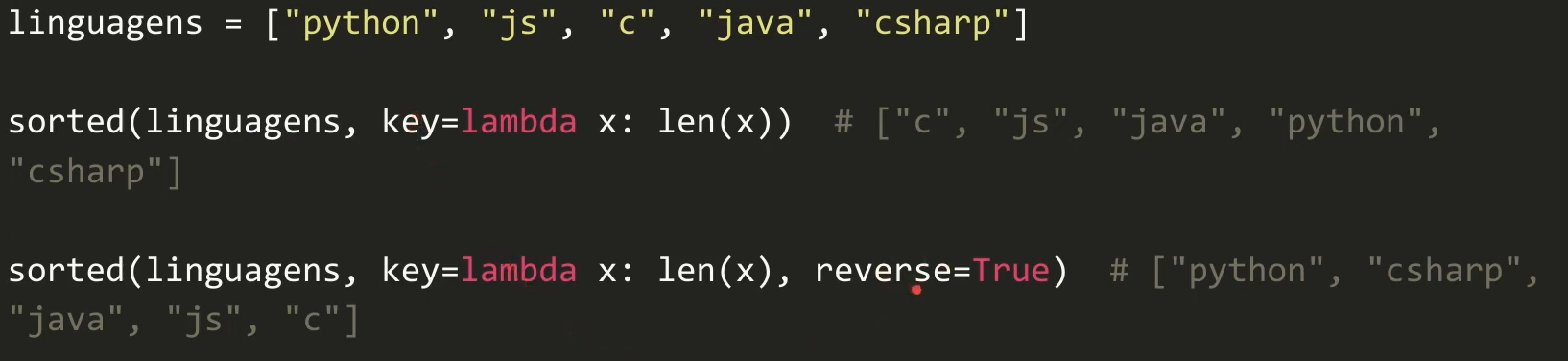
Exemplo 2 – modo ‘normal’



Exemplo 2 - usando o **Comprehensions**



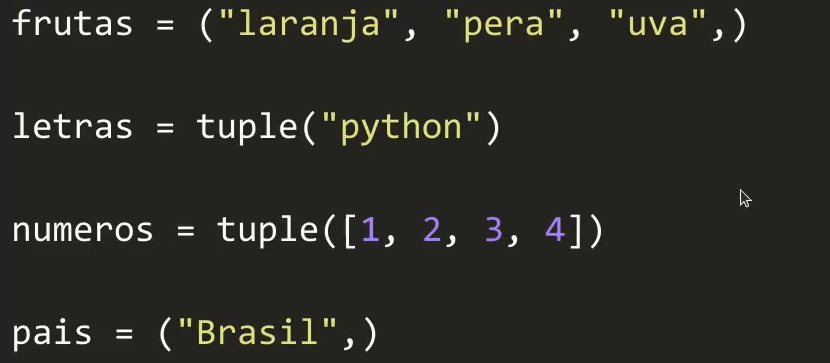
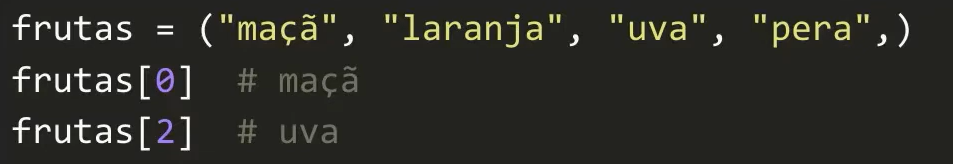
**Método da classe list:**

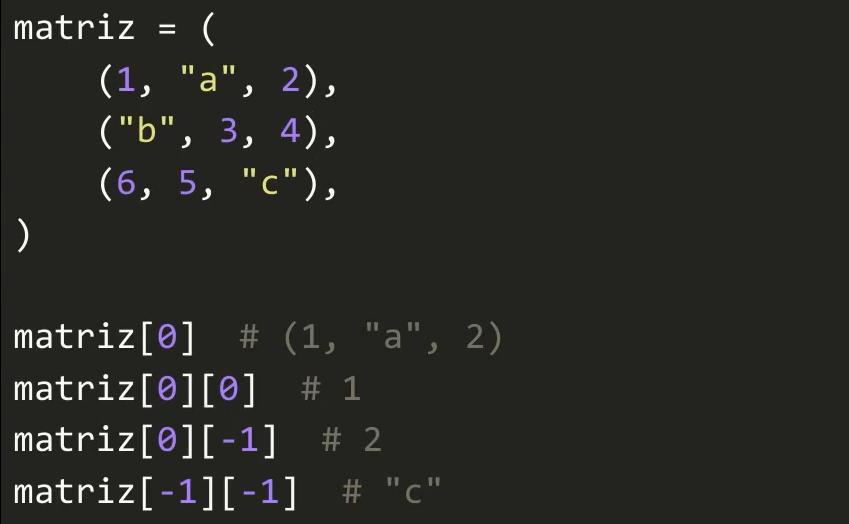
* **append:** adiciona elemento na última posição da lista -> lista.append(item);
* **clear:** limpa a lista -> lista.clear();
* **copy:** faz uma cópia da lista, é igual, porém é outra instância -> lista.copy();
* **count:** serve para contar quantas vezes determinado item aparece na lista -> lista.count(item);
* **extend:** serve para ‘mergiar’ duas listas. Por exemplo, existe uma lista e vc quer adicionar outros itens dentro(lista) dela -> lista.extend([item, item]);
* **index:** mostra qual o primeiro índice que aparece o item passado -> lista.index(item);
* **pop:** retira o último item da lista -> lista.pop(). Ou passando qual o índice do elemento a ser retirado -> lista.pop(1);
* **remove:** remove o elemento passado. Se houver mais de uma ocorrência do mesmo item, vai ser removido a primeira incidência encontrada do elemento -> lista.remove(item);
* **reverse:** serve para transpôr a lista, ou seja, espelhar ela -> lista.reverse():
* **sort:** serve para ordenar a lista. Por padrão, ordena em ordem alfabética -> lista.sort(). É possível passar argumento para o método: reverse e key.
  + **reverse:** pega o ordenamento padrão e inverte(espelha) -> lista.sort(reverse=True);
  + **key:** usado quando se quer ordenar de outra forma, sem ser alfabética; 
* **len:** serve para ver o tamanho da lista, ou seja, quantos elementos ela tem -> lista.len();
* **sorted:** função build in. Também serve para ordenção e também é possível utilizar a passagem de parâmetros. A diferença é que o **sorted** é uma função própria da linguagem e o **sort** é um método para manuseio de listas. 

## Conhecendo Tuplas em Python

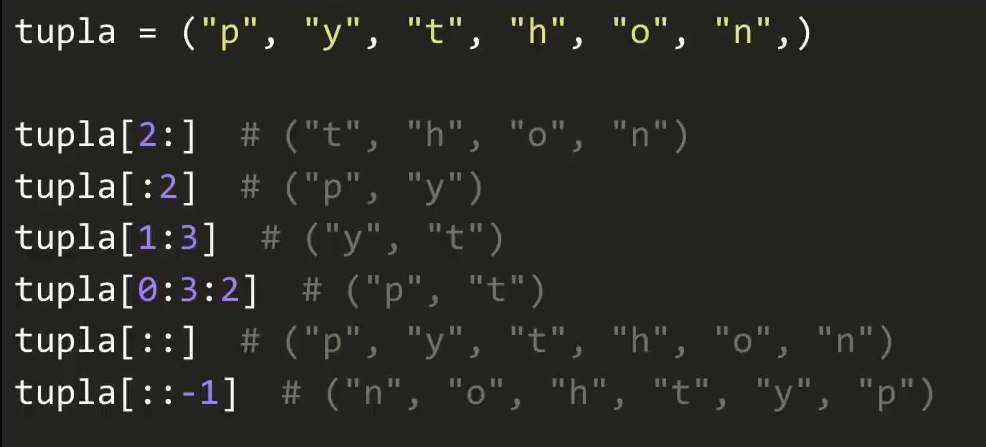
Tuplas são estruturas de dados muito parecidas com as listas, a principal diferença é que **tuplas são imutáveis** enquando listas são mutáveis. Podemos criar tuplas através da classe tuple, ou colocando valores separados por vírgula de parenteses.

//Quando se declara uma tupla usando parenteses, para não dar conflito para o interpretador, é uma boa prática pôr uma vírgula ao final.

Tuplas podem armazenar todos os tipos de dados, inclusive outras tuplas, sendo assim é possível criar estruturas bidimensionais (tabela) e acessar a informação passando linha e coluna. 

Fatiamento de Tuplas:



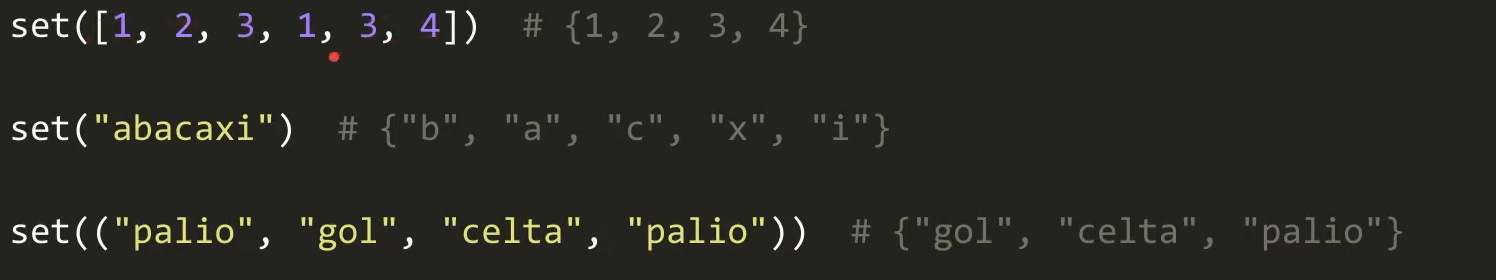
**Métodos que a tupla possue**(funciona igual nas listas). Os únicos métodos que não possue são aqueles que faz alterações:

* **count();**
* **index();**
* **len;**

## Explorando Conjuntos em Pyhton

Estrutura de dados **set**: um set é uma coleção que não possui objetos repetidos, usamos sets para representar conjuntos matemáticos ou eliminar itens duplicados de um iterável.

//Ao usar o set, a ordem dos elementos não é garantida. No exemplo abaixo são passadas algumas estruturas(lista, string, tupla) para eliminar duplicidade:

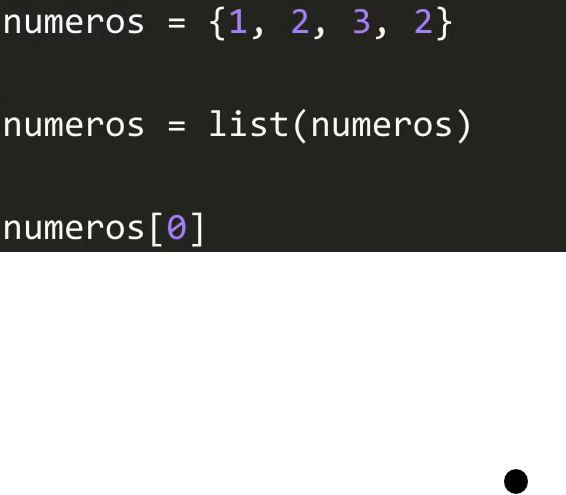


//Pode se criar um set, declarando direto utilizando {}:

nome = {valor1, valor1, valor, vlr}

print(nome) -> ao printar isso já vai sair sem os valores duplicados

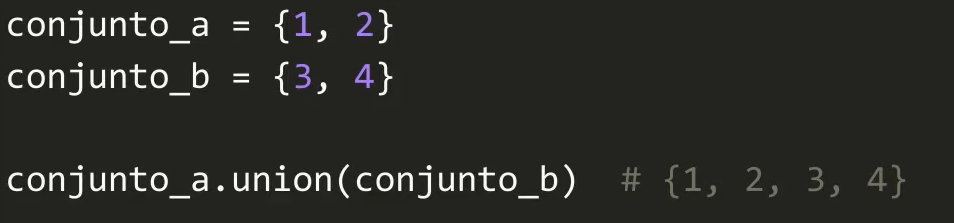
Conjunto em Python não suportam indexação e nem fatiamento, caso queira acessar os seus valores é necessa´rio converter o conjunto para lista.



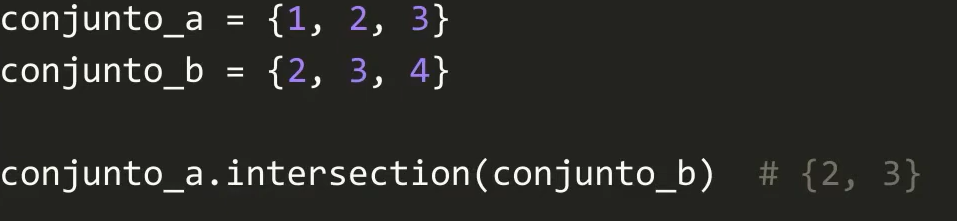
É possível iterar sobre um conjunto. A forma mais comum de percorrer os dados de um conjutno é utilizando o comando **for**.

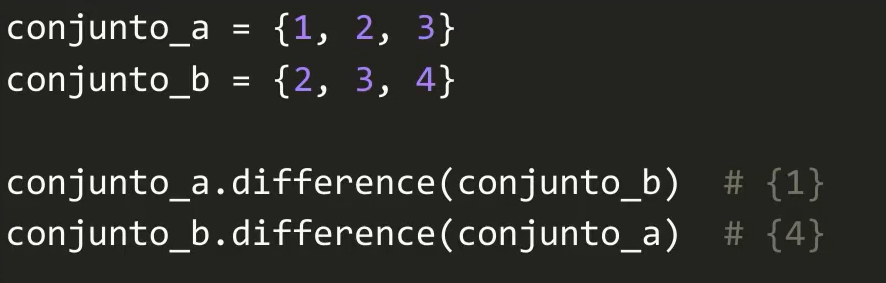
**Métodos da classe set:**

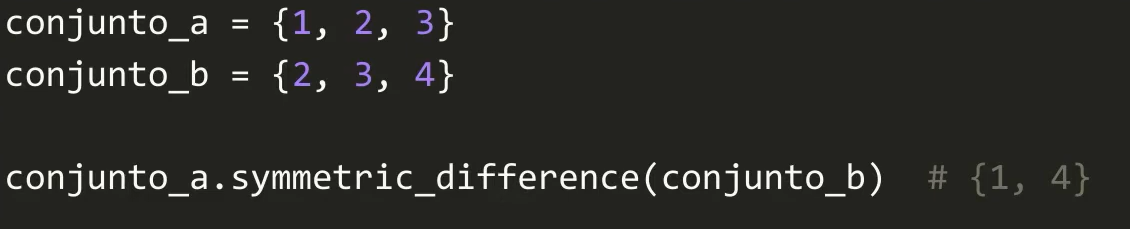
* **union:** juntar elementos de dois conjuntos

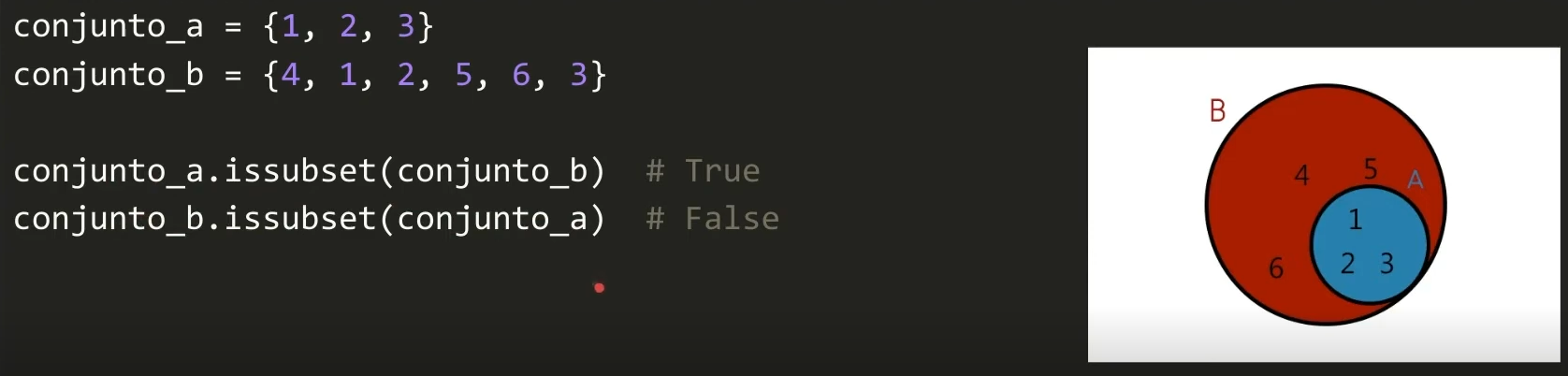


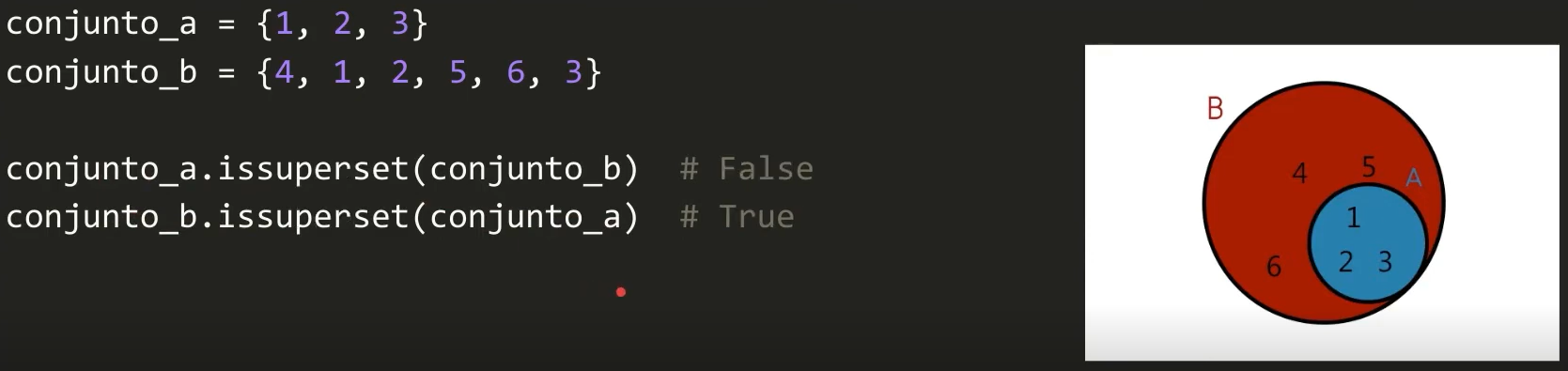
* **intersection:** interseção dos dois conjuntos, ou seja, juntar os elementos que existe nos dois conjuntos



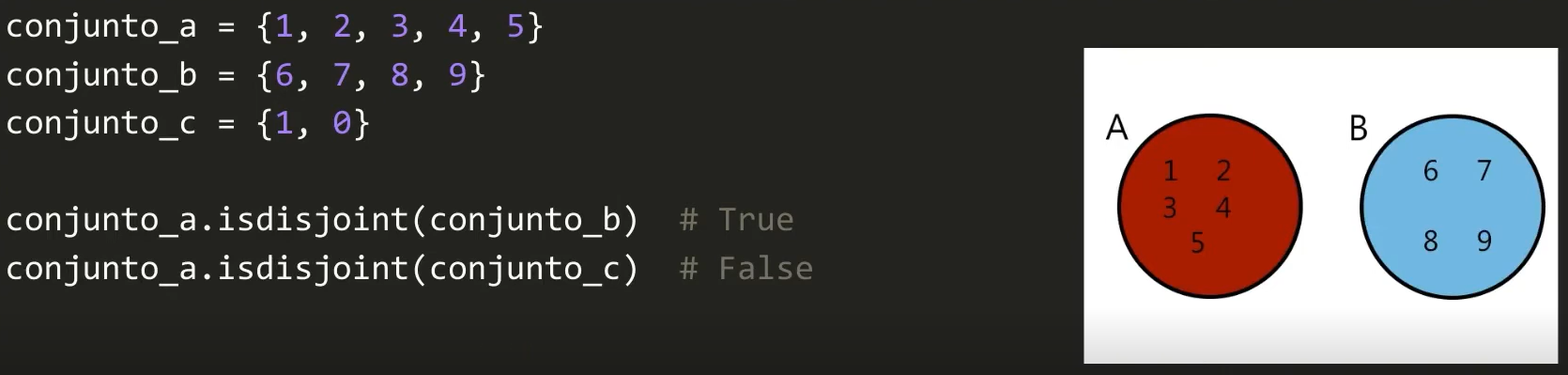
* **difference:** tudo que existe em um conjunto que não existe no outro
* **symmetric\_difference:** junta todos os elementos que não estão na intersecção, ou seja, só existe em um conjunto ou no outro



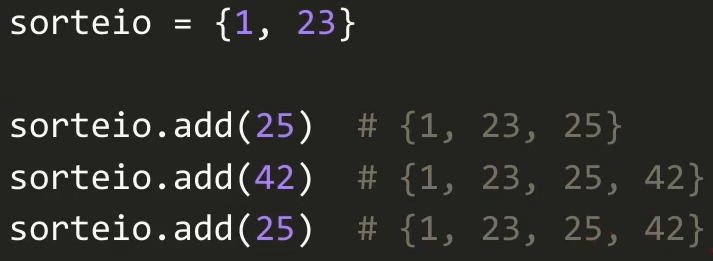
* **issubset:** retorna valor boolean, retorna se um conjunto está dentro do outro
* **issuperset:** é ao contrário do método acima



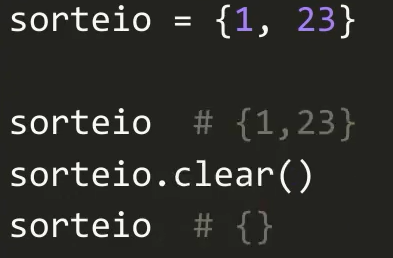
* **isdisjoint:** todos os elementos de um conjunto, não está presente no outro conjunto

****

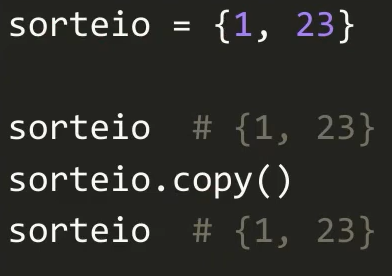
* **add:** adiciona um elemento como último elemento do conjunto caso esse elemento não exista no conjunto



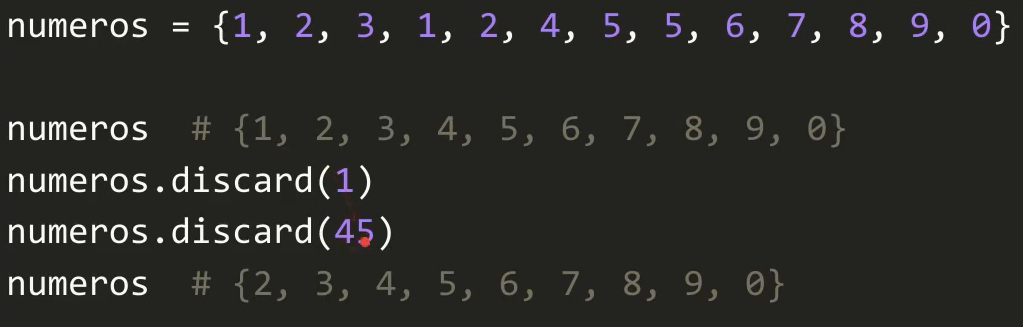
* **clear:** limpa todo o conjunto



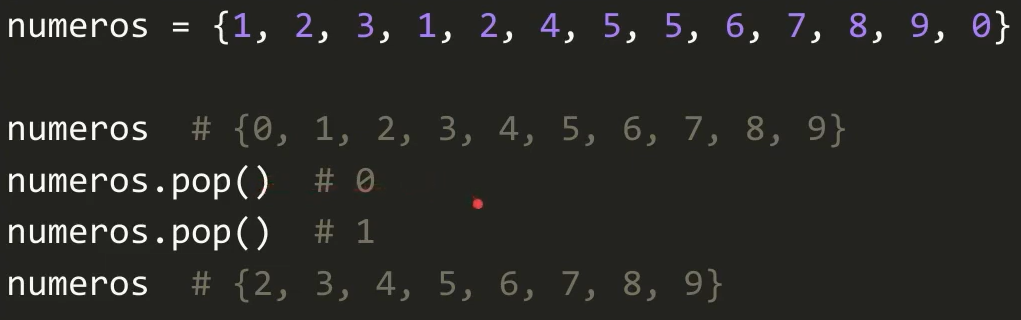
* **copy:** gera uma cópia do conjunto



* **discard:** elimina um valor do conjunto e se esse elemento não existir no conjunto ele não faz nada

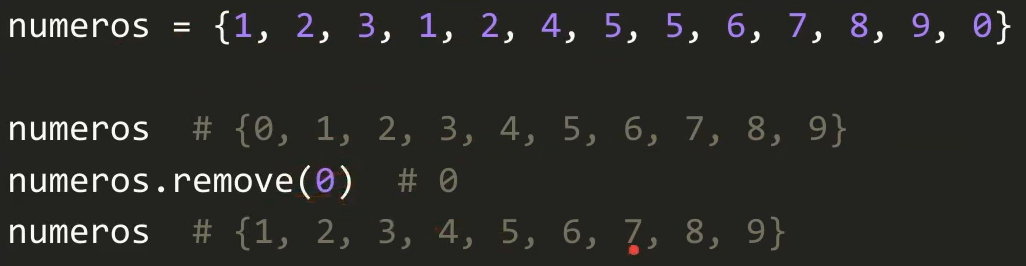


* **pop:** no caso do conjunto vai retirando o elemento da primeira posição

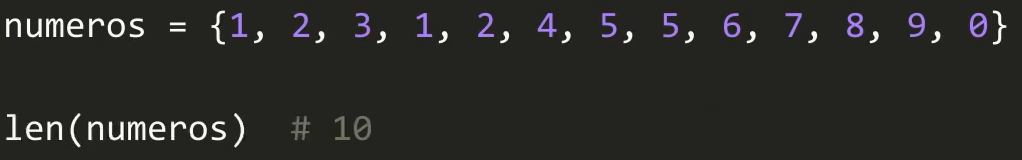


* **remove:** passa o elemento que quer retirar. A diferença do remove para o discard é que o remove se tentar remover um elemento que não existe no conjunto ele da erro.

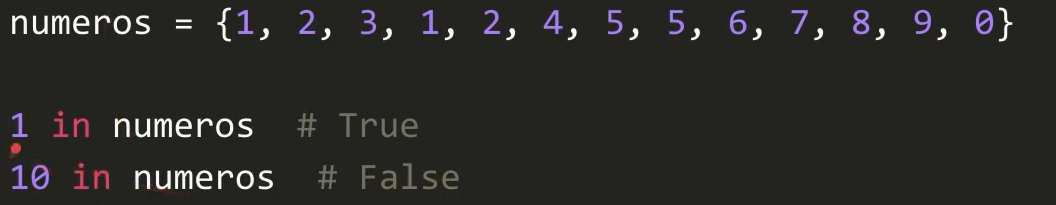
//nem sempre não dar erro é uma coisa boa, mas também ficar dando erro pode não servir para o propósito do programador. O uso de um ou do outro depende do contexto em que é inserido



* **len:** tamanho do conjunto

****

* **in:** verificar se um valor existe dentro do conjunto



## Aprendendo a Utilizar Dicionários em Python

-

## Dominando Funções em Python

-

## Desafios de Código: Aperfeiçoando Sua Lógica e Pensamento Computacional

-

## Resolvendo Cálculos com Python

-

# Introdução a Banco de Dados Relacionais e ETL

-

# Análise de Dados com Excel e Copilot

-

# Noções de Computação em Nuvem

-

# Fundamentos e Técnicas de Machine Learning

-