

Estruturas de Dados



Conteudista: Prof. Me. Hugo Batista Fernandes

Revisão Textual: Prof.^a Dra. Luciene Oliveira da Costa Granadeiro

Objetivo da Unidade:

- Estudar formas para armazenar diversos valores (N valores) em uma mesma variável.

☰ Contextualização

☰ Material Teórico

☰ Material Complementar

☰ Referências

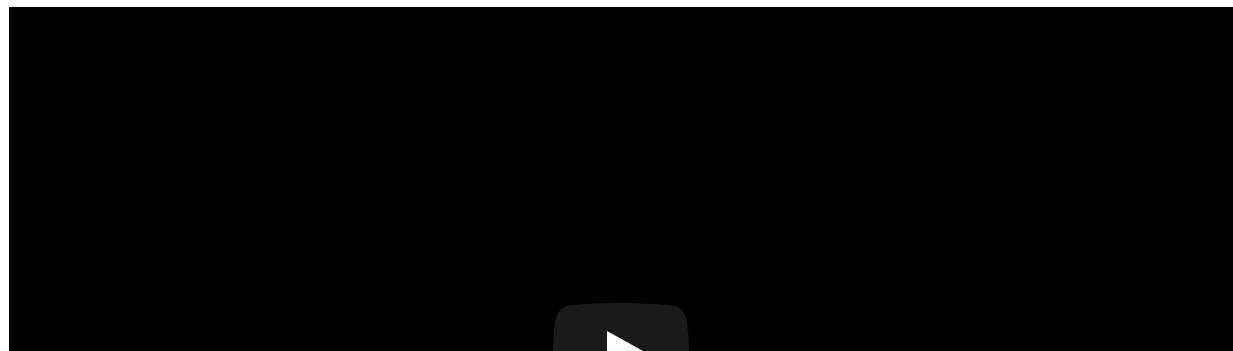
Contextualização

Anteriormente, aprendemos diversos conceitos de programação de computadores: variáveis, uso de operadores, estruturas de decisão e estruturas de repetição. Sempre quando precisamos, utilizamos variáveis, porém, podemos perceber que essas variáveis armazenam apenas um valor por vez. Para cada variável, podemos armazenar somente um valor.

Nesta Unidade, iremos estudar formas para armazenar diversos valores (N valores) em uma mesma variável. Esse conceito permitirá um salto de qualidade e eficiência em nossos códigos de programação. No estudo de algoritmos, chamamos esse conceito de vetores. Na linguagem *Python*, temos: *Lists*, *Dictionarys* e *Tuples*.

Vídeo

[Vetores – Curso de Algoritmos](#)





Material Teórico

Listas

Uma lista é uma estrutura de dados em *Python* que é uma sequência ordenada de elementos e mutável (pode ter seus valores atualizados). Cada elemento corresponde a um valor e é chamado de item e são definidos entre colchetes [].

Utilizamos listas quando precisamos trabalhar com muitos valores relacionados, pois elas permitem que se mantenham diversos dados juntos, crie um código mais intuitivo e execute métodos e operações em vários valores de uma só vez.

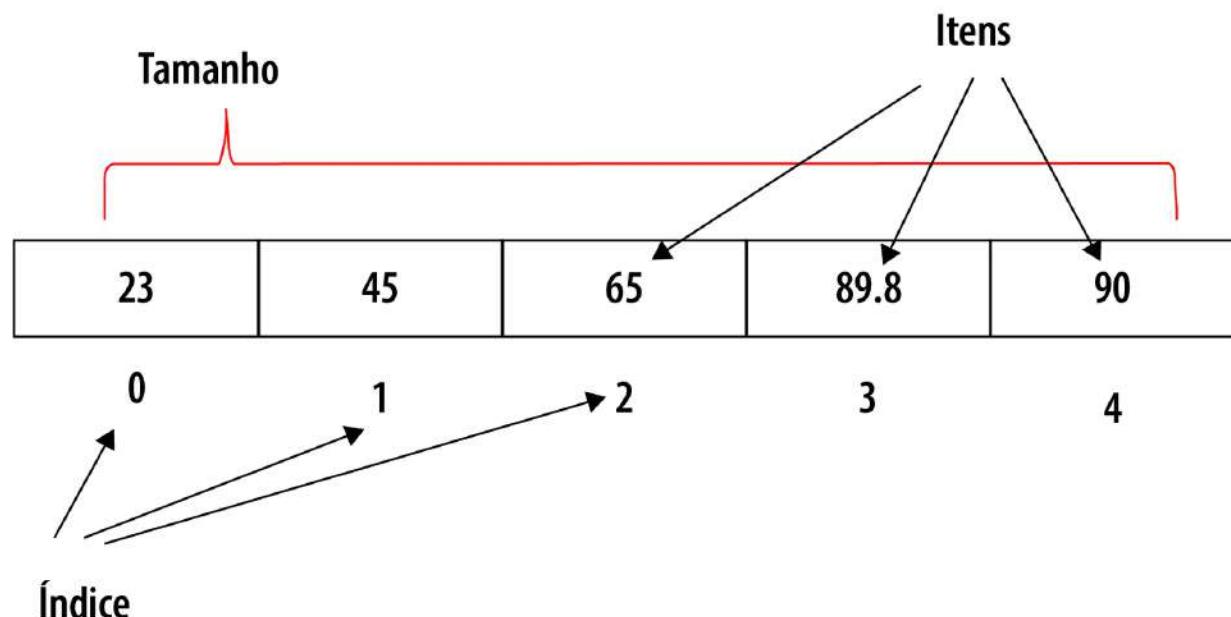


Figura 1

Vejamos alguns conceitos básicos de listas:

- **Itens (ou elemento):** são os valores armazenados dentro de cada posição da lista;
- **Tamanho:** é o tamanho da lista (quantos itens possui a lista);
- **Índice:** é a posição do elemento onde um valor está armazenado.

Declarando Listas e Atribuindo Valores

A forma de declaração de uma lista é muito parecida com a declaração de uma variável comum, contudo, os valores são indicados entre colchetes [] separando cada valor por vírgula. Vejamos um exemplo:

```
minhaLista = [1, 2, 3, 4, 5]
```

O código acima declara uma lista com o nome de “minhaLista” e atribui os valores 1, 2, 3, 4 e 5.

As regras para nomear listas em *Python* seguem as mesmas regras de nomeação de variáveis comuns:

- As variáveis podem ter letras (A-Z e a-z), dígitos (0-9) e sublinhados (_).
- Não podem começar com números;
- Não podem conter pontuação, caracteres especiais (!, @, \$, #, etc) ou espaços;
- Não podem ser utilizadas palavras reservadas da linguagem.

Para atribuição de valores em listas, no *Python*, diferente de outras linguagens de programação, podemos armazenar tipos de dados diferentes em uma mesma variável.

```
minhaLista2 = [1, "Silva", 4.5]
```

No código acima, atribuímos valores de tipos diferentes, porém para a mesma variável.

Podemos armazenar valores de todos os tipos primitivos de dados do *Python*: *string*, *int*, *float* e *boolean*, além de objetivos e outros tipos de dados.

Outra forma de criar uma lista e atribuir valores é criar uma lista vazia e por meio da *append*, armazenar valores. A função *append* adiciona um elemento sempre no final da lista.

Vejamos um exemplo:

```
1  ListaNomes = []
2  ListaNomes.append("João")
3  ListaNomes.append("Maria")
4  ListaNomes.append("Anna")
5  ListaNomes.append("Clara")
6
```

Figura 2

Fonte: Acervo do Conteudista

Explicando o Código

- Linha 1: declaramos a lista (vazia) com o nome de “ListaNomes”. Utilizamos colchetes para indicar que se trata de uma lista;
- Linha 2: atribuímos o valor “João” para a lista “ListaNomes”;
- Linhas 3 a 5: atribuímos valores à lista por meio da função *append*.

Se desejarmos adicionar mais elementos à lista, bastaria seguir adicionando instruções *append*.

Vejamos um exemplo onde criamos um programa capaz de armazenar N nomes em uma lista digitada pelo usuário. Damos ao usuário a possibilidade de seguir ou não cadastrando nomes nessa lista. Vamos ao código.

```
1  ListaNomes = []
2
3  while True:
4      nome = input("Digite um nome ")
5      ListaNomes.append(nome)
6
7      continuar = input("Deseja continuar? Digite Sim ou Não ")
8      if(continuar=="Não" or continuar=="NÃO"):
9          break
10
11 print(ListaNomes)
12
```

Figura 3

Fonte: Acervo do Conteudista

Explicando o Código

- **Linha 1:** declaramos a lista (vazia) com o nome de “ListaNomes”. Utilizamos colchetes para indicar que se trata de uma lista;
- **Linha 2:** assinatura da instrução *while*. Como condição, declaramos a palavra reservada “*True*”, dessa forma, forçamos o *while* continuar executando de forma infinita ou até que alguma ação faça o *while* finalizar;
- **Linha 4:** solicitamos ao usuário que digite um nome e atribuímos o valor digitado à variável “*nome*”;
- **Linha 5:** utilizamos a função *append* para armazenar o valor contido na variável “*nome*” na lista (“ListaNomes”);
- **Linha 7:** a cada repetição, é solicitado ao usuário do programa que digite “Sim” ou “Não”. O valor digitado é atribuído à variável “*continuar*”;
- **Linha 8:** caso o valor contido na variável “*continuar*” seja igual a “Não” ou “NÃO”, o *while* será encerrado (instrução *break* na linha 9);
- **Linha 9:** por meio da função *print*, são exibidos os valores contidos na lista “ListaNomes”.

Acessando Elementos em uma Lista

Para acessar um elemento específico em uma lista, é preciso indicar o índice onde o elemento está armazenado. Chamamos de índice o valor inteiro que representa a posição do elemento que desejamos acessar.

Os índices em uma lista (ou vetor) sempre iniciam em zero (0). Isso significa que uma lista de 2 elementos, temos o índice 0 e índice 1. Nesse cenário, se tentarmos acessar o índice “2”, ao executar o programa, retornará um erro de execução, pois estamos tentando acessar um

terceiro elemento (que não existe) na lista. Por exemplo, para obter qualquer item de uma lista, basta indicar o índice entre colchetes. Vejamos:

```
1 minhaLista = [1,2]
2 print(minhaLista[0])
3
```

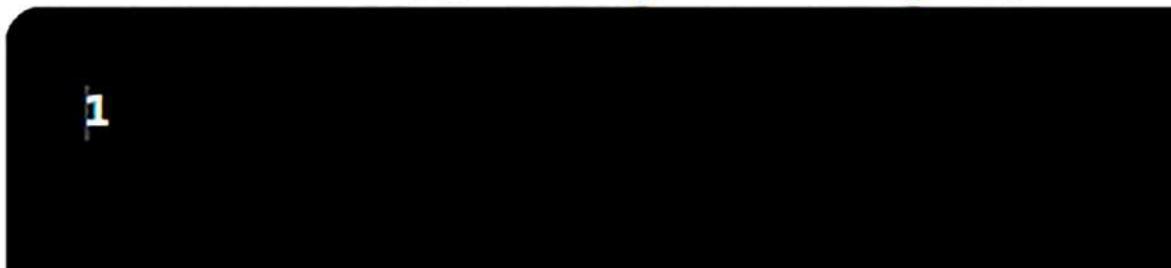
Figura 4

Fonte: Acervo do Conteudista

Saída:

Result

CPU Time: 0.01 sec(s), Memory: 7548 kilobyte(s)



```
1
```

Figura 5

Fonte: Acervo do Conteudista

Explicando o código, na linha 1, temos a declaração da lista com a atribuição de valores. Na linha 2, dentro da função *print*, descrevemos o nome da lista e, entre colchetes, indicamos o índice do elemento que desejamos visualizar.

Outra forma para acessar e visualizar os elementos de uma lista é por meio de uma estrutura de repetição *for*. Vejamos um exemplo:

```
1 minhaLista3 = ["João", "Maria", "Anna", "Silva"]
2
3 for elementos in minhaLista3:
4     print(elementos)
5
```

Figura 6

Fonte: Acervo do Conteudista

Saída:

Result

CPU Time: 0.00 sec(s), Memory: 7716 kilobyte(s)

```
João
Maria
Anna
Silva
```

Figura 7

Fonte: Acervo do Conteudista

Explicando o código, na linha 1, temos a declaração da lista com a atribuição de valores. Na linha 3, declaramos o *for*, criamos a variável de iteração chamada “elementos” e em seguida, a

cláusula `in` e a descrição da lista que desejamos visualizar. Dessa forma, a cada repetição do `for`, o valor contido no elemento da lista será atribuído para a variável de iteração.

É importante ressaltar que a quantidade de repetições que o `for` irá executar é igual à quantidade de elementos existentes na lista.

Atualizando Elementos em uma Lista

Podemos atualizar um elemento específico em uma lista, indicando entre colchetes o índice onde o item está armazenado. Vamos a um exemplo:

```
1 minhaLista2 = [1, "Silva", 4.5 ]  
2 minhaLista2[0] = 2  
3 minhaLista2[1] = "João da Silva"  
4
```

Figura 8

Fonte: Acervo do Conteudista

No código acima, na linha 1 declaramos uma lista e atribuímos 3 valores. Na linha 2, atualizamos o valor contido no item armazenado no índice zero (0), o valor original que era “1”, passa agora a ser “2”. Na linha 3, atualizamos o valor contido no item armazenado no índice um (1), o valor original que era “Silva”, passa agora a ser “João da Silva”.

Removendo Elementos de uma Lista

No contexto de listas, no `Python`, podemos remover elementos específicos ou elementos no final de uma lista. Para remover um elemento no final da lista, utilizamos a função `pop`. Vejamos um exemplo.

```
1 minhaLista3 = ["João", "Maria", "Anna", "Silva"]  
2 minhaLista3.pop()  
3
```

Figura 9

Fonte: Reprodução

Na linha 2, descrevemos o nome da lista e, em seguida, logo após o sinal de ponto final, descrevemos a função *pop()*. Essa função remove o elemento contido no **final da lista**. Nesse exemplo, a lista removerá o valor “Silva”.

Pode-se utilizar a função *pop()* quantas vezes forem necessárias ao seu algoritmo.

```
1 minhaLista3 = ["João", "Maria", "Anna", "Silva"]  
2 minhaLista3.pop()  
3 minhaLista3.pop()  
4 minhaLista3.pop()  
5 print(minhaLista3)
```

Figura 10

Fonte: Reprodução

No código acima, a cada vez que a função *pop()* é executada, é removido o elemento contido no final da lista. Desse modo, ao final da execução, restará apenas o elemento contendo o valor “João”.

Outra maneira para remover um elemento de uma lista é por meio da função *remove()* tendo como parâmetro o valor desejado. A função *remove()* permite remover a primeira ocorrência de

um elemento pelo seu valor. Vejamos um exemplo.

```
1 minhaLista3 = ["João", "Maria", "Anna", "Silva"]
2 minhaLista3.remove("Maria")
3 print(minhaLista3)
```

Figura 11

Fonte: Reprodução

Na linha 2, descrevemos o nome da lista e em seguida, logo após o sinal de ponto final, descrevemos a função `remove()`. Essa instrução irá remover o primeiro elemento cujo valor seja igual à “Maria”. Caso o valor indicado na função `remove()` não exista na lista, ocorrerá um erro de execução.

Site

Jdoodle – Online Python 3 IDE

Pode-se sempre utilizar um bloco de exceções (*Try Except*) para tratar exceções em seu código. No cenário da função `remove`, veja um exemplo a seguir de um código contendo o bloco de exceção *Try Except*.

Clique no botão para conferir o conteúdo.

ACESSE

Leitura

Tratamento de Exceções

Clique no botão para conferir o conteúdo.

ACESSE

Funções de listas podem ser executadas diversas vezes ao longo do código. Vejamos um exemplo onde utilizamos a funções para adicionar e remover elementos.

```
1 minhaLista3 = ["João", "Maria", "Anna", "Silva"]
2 minhaLista3.pop()
3 minhaLista3.pop()
4 minhaLista3.append("Fernanda")
5 print(minhaLista3)
```

Figura 12

Fonte: Reprodução

Ao final da execução do programa, a lista irá conter 3 elementos com os seguintes valores: “João”, “Maria” e “Fernanda”.

Leitura

Como trabalhar com listas em Python

Clique no botão para conferir o conteúdo.

ACESSE

Tuplas

A tupla é uma estrutura de dados em *Python* que é uma sequência ordenada de elementos e imutável (os valores contidos em seus elementos não podem ser atualizados ou excluídos). Tuplas são similares às listas quanto às regras para nomeação e tipos de dados que podem ser armazenados. Cada elemento dentro de uma tupla é chamado de item, porém, seus valores são definidos entre parênteses (). Por sua característica imutável, tuplas podem ser utilizadas, por exemplo, em cenários onde é preciso garantir que dados sejam protegidos contra gravação ao longo da execução do programa.

Declarando Listas e Atribuindo Valores

A forma de declaração de uma tupla é muito parecida com a declaração de uma lista, contudo, os valores são indicados entre parênteses () separando cada valor por vírgula. Vejamos um exemplo:

```
minhaTupla = (1, 2, 3, 4, 5)
```

O código acima declara uma lista com o nome de “minhaTupla” e atribui os valores 1, 2, 3, 4 e 5.

Acessando Elementos em uma Lista

Para acessar um elemento específico em uma tupla, é preciso indicar o índice onde o elemento está armazenado. Vejamos um exemplo.

```
1 | minhaTupla = ("João", "Anna", "Silva")
2 | print(minhaTupla[1])
```

Figura 13

Fonte: Reprodução

Saída:

Result

CPU Time: 0.01 sec(s), Memory: 7708 kilobyte(s)

```
| Anna
```

Figura 14

Fonte: Reprodução

Explicando o código, na linha 1, temos a declaração da tupla com a atribuição de valores. Na linha 2, dentro da função *print*, descrevemos o nome da lista e, entre colchetes, indicamos o índice do elemento que desejamos visualizar.

Similar à lista, outra forma para acessar e visualizar os elementos de uma tupla é por meio de uma estrutura de repetição *for*. Vejamos um exemplo:

```
1 minhaTupla = ("João", "Anna", "Silva")
2
3 for elemento in minhaTupla:
4     print(elemento)
5
```

Figura 15

Fonte: Reprodução

Saída:

Result

CPU Time: 0.00 sec(s), Memory: 7612 kilobyte(s)

```
João
Anna
Silva
```

Figura 16

Fonte: Reprodução

Explicando o código: na linha 1, temos a declaração da lista com a atribuição de valores. Na linha 3, declaramos o *for*, criamos a variável de iteração chamada “elemento” e em seguida, a cláusula *in* e a descrição da lista que desejamos visualizar. Dessa forma, a cada repetição do *for*, o valor contido no elemento da lista será atribuído para a variável de iteração.

É importante ressaltar que a quantidade de repetições que o *for* irá executar é igual à quantidade de elementos existentes na lista.

Atualizando Elementos em uma Lista

Conforme vimos, tuplas não podem ser atualizadas, dessa forma, caso necessário atualizar elementos de um tupla, deve-se declarar novamente a tupla.

Removendo Elementos de uma Lista

Do mesmo modo que não se pode atualizar um elemento em um tupla, também não se pode excluir. Caso seja necessário, é permitida a exclusão completa da tupla por meio da instrução *del*.

```
1 minhaTupla = ("João", "Anna", "Silva")
2 del minhaTupla
3 print(minhaTupla)
4
```

Figura 17

Fonte: Reprodução

Saída:

Result

CPU Time: 0.01 sec(s), Memory: 8128 kilobyte(s)

```
| Traceback (most recent call last):
|   File "/home/jdoodle.py", line 3, in <module>
|     print(minhaTupla)
|   NameError: name 'minhaTupla' is not defined
```

Figura 18

Fonte: Reprodução

Explicando o código: na linha 1, temos a declaração da lista com a atribuição de valores. Na linha 2, descrevemos a cláusula *del* e, em seguida, descrevemos o nome da tupla que desejamos excluir.

Na linha 3, ao executar a função *print*, tendo como parâmetro a tupla criada na linha 1, o programa retornará um erro de execução, pois a tupla foi excluída na linha anterior (linha 2).

Leitura

Entenda o que é e quais as Diferenças entre Listas, Sets, Tuplas e Dicionários no Python

Clique no botão para conferir o conteúdo.

ACESSE

Dicionários

Um dicionário é uma estrutura de dados em *Python* que é uma sequência não ordenada de elementos e mutável. Cada item um dicionário possui um par de chave/valor. As regras para nomeação de dicionários seguem as mesmas utilizadas para tuplas e listas.

Declarando Dicionários e Atribuindo Valores

A forma de declaração de um dicionário é similar à declaração de uma lista ou tupla. Os valores são indicados entre o símbolo de chaves {} separando cada valor por vírgula. Como dito, todo item dentro de um dicionário é composto pelo conjunto de chave/valor. Embora os valores

possam ser de qualquer tipo de dados e possam se repetir, as chaves devem ser do tipo imutável (*string*, número ou tupla com elementos imutáveis) e devem ser exclusivas.

Vejamos a sintaxe de declaração de um dicionário:

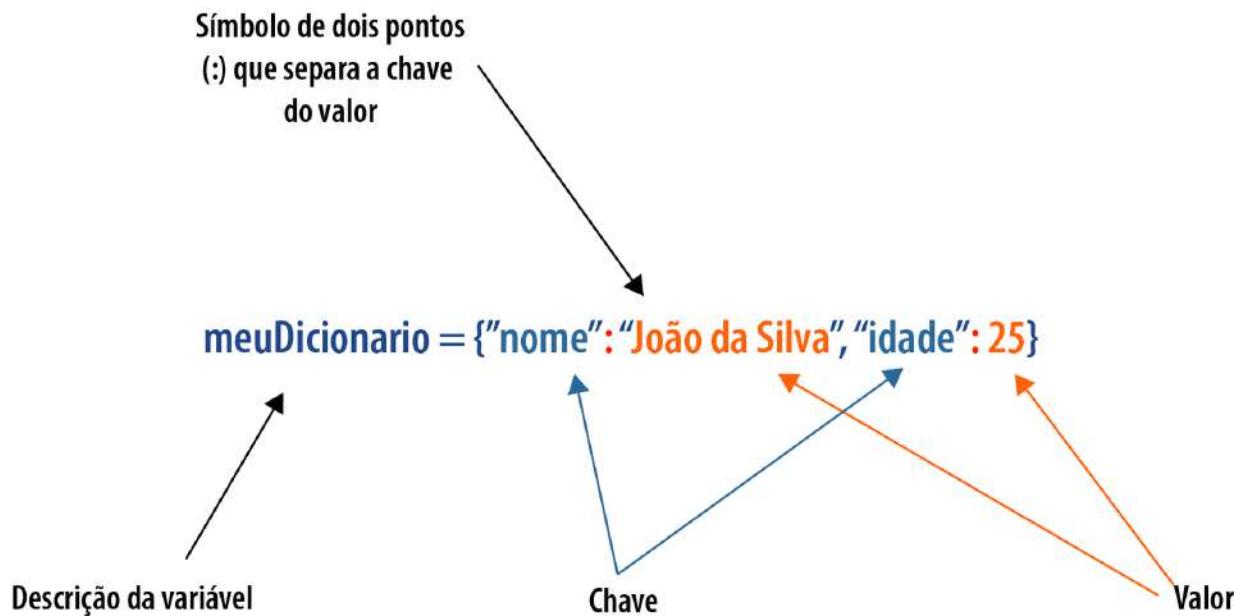


Figura 19

Em Python temos:

```
meuDicionario = {"nome": "João da Silva", "idade":25}
```

O código acima declara um dicionário com o nome de “meuDicionario” e atribui o valor “João da Silva” para a chave “nome” e o valor “25” para a chave “idade”. As chaves declaradas nesse dicionário são únicas (não se repetem) e são do tipo *string*.

Acessando Elementos em uma Lista

Para acessar um elemento específico em um dicionário, basta indicar a chave do elemento entre colchetes. Vejamos um exemplo:

```
1 meuDicionario = {"nome": "João da Silva", "idade":25}  
2 print(meuDicionario["nome"])  
3
```

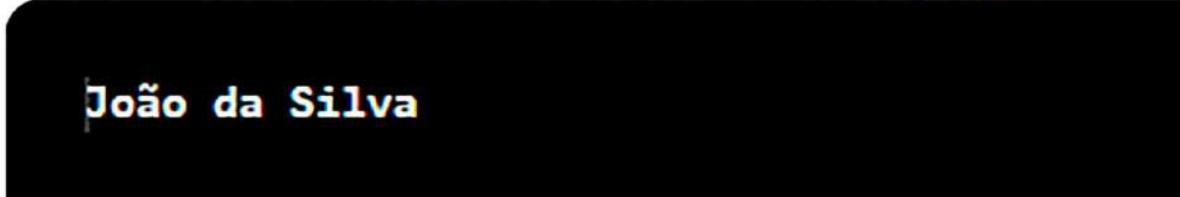
Figura 20

Fonte: Reprodução

Saída:

Result

CPU Time: 0.00 sec(s), Memory: 7800 kilobyte(s)



```
João da Silva
```

Figura 21

Fonte: Reprodução

Atualizando e Adicionando Elementos em um Dicionário

Podemos atualizar um elemento específico em um dicionário, basta indicar a chave do elemento entre colchetes. Vamos a um exemplo:

```
1 meuDicionario = {"nome": "João da Silva", "idade":25}
2 meuDicionario["idade"] = 38
3
4 print(meuDicionario["idade"])
5
```

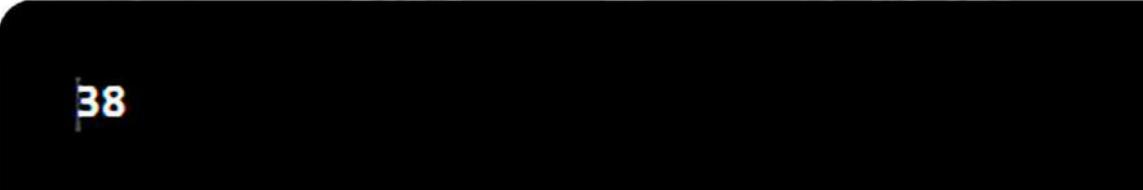
Figura 22

Fonte: Reprodução

Saída:

Result

CPU Time: 0.01 sec(s), Memory: 7664 kilobyte(s)



```
38
```

Figura 23

Fonte: Reprodução

No código acima, na linha 1, declaramos um dicionário e atribuímos seus valores. Na linha 2, atualizamos o valor referente à chave “idade” para “38”.

Para adicionar um elemento ao dicionário, deve-se declarar a nova chave do dicionário e atribuir seu valor. Vejamos um exemplo:

```
1 meuDicionario = {"nome": "João da Silva", "idade":25}
2
3 meuDicionario["email"] = "joao@gmail.com"
4
5 print(meuDicionario)
6
```

Figura 24

Fonte: Reprodução

No código acima, na linha 1, declaramos um dicionário e atribuímos seus valores. Na linha 2, adicionamos uma nova chave com o nome de “email” e atribuímos o valor “joao@gmail.com”. Na linha 3, por meio da função *print*, visualizamos todos os dados contidos no dicionário.

Removendo Elementos de um Dicionário

Em um dicionário, podemos remover elementos específicos, um elemento contido no final do dicionário ou apagar todos os elementos. Para remover um elemento no final da lista, utilizamos a função *popitem()*. Vejamos um exemplo:

```
1 meuDicionario = {"nome": "João da Silva", "idade":25, "email":'joão@msn.com'}
2 meuDicionario.popitem()
3
```

Figura 25

Fonte: Reprodução

Na linha 2, descrevemos o nome do dicionário e, em seguida, logo após o sinal de ponto final, descrevemos a função *popitem()*. Essa função remove o elemento contido no final do dicionário. Nesse exemplo, o dicionário removerá a chave “email” e seu valor.

Pode-se utilizar a função `popitem()` quantas vezes forem necessárias ao seu algoritmo.

Para remover um elemento específico, basta indicar o nome da chave como parâmetro na função `pop()`. Vejamos um exemplo:

```
1 meuDicionario = {"nome": "João da Silva", "idade":25, "email":'joão@msn.com'}
2 meuDicionario.pop("nome")
3
```

Figura 26

Fonte: Reprodução

Na linha 2, descrevemos o nome do dicionário e, em seguida, logo após o sinal de ponto final, descrevemos a função `pop()` e, como parâmetro, descrevemos a chave “nome”. Dessa forma, o elemento removerá a chave “nome” e seu valor.

Por fim, para excluir todos os elementos do dicionário, basta descrever a cláusula `del` seguida do nome do dicionário.

```
del meuDicionario
```

Leitura

Dicionários no Python

Clique no botão para conferir o conteúdo.

ACESSE

Manipulação de *Strings*

Em *Python*, uma *string* é uma sequência de caracteres *Unicode*. *Strings* podem ser criadas colocando caracteres entre aspas simples ou aspas duplas. Mesmo aspas triplas podem ser usadas em *Python*, mas geralmente usadas para representar *strings* de várias linhas.

```
1 minhaString = 'Olá'  
2 print(minhaString)  
3  
4 minhaString = "Olá"  
5 print(minhaString)  
6  
7 minhaString = '''Olá'''  
8 print(minhaString)  
9  
10 # 3 aspas duplas para suportar textos com múltiplas linhas  
11 minhaString = """Olá, bem vindo  
12 | | ao mundo do Python"""  
13 print(minhaString)  
14
```

Figura 27

Fonte: Reprodução

Saída:

Result

CPU Time: 0.01 sec(s), Memory: 7892 kilobyte(s)

```
|Olá  
Olá  
Olá  
Olá, bem vindo  
      ao mundo do Python
```

Figura 28

Fonte: Reprodução

É possível acessar caracteres específicos em uma *string*. Para isso, basta indicar o número do índice desejado.

```
1 minhaString = 'São Paulo'  
2 print(minhaString[0])  
3
```

Figura 29

Fonte: Reprodução

Saída:

Result

CPU Time: 0.01 sec(s), Memory: 7688 kilobyte(s)



Figura 30

Fonte: Reprodução

É possível concatenar (juntar) duas ou mais *strings* utilizando o sinal de adição (+).

```
1 minhaString = 'São Paulo' + ' é um estado ' + 'do Brasil'  
2 print(minhaString)  
3
```

Figura 31

Fonte: Reprodução

Saída:

Result

CPU Time: 0.00 sec(s), Memory: 7864 kilobyte(s)

```
São Paulo é um estado do Brasil
```

Figura 32

Fonte: Reprodução

Também é possível identificar um caractere específico ou uma sequência de caracteres contida em uma *string*. Para isso, utilizamos a cláusula *in*.

```
1 minhaString = 'São Paulo' + ' é um estado ' + 'do Brasil'  
2  
3 if 'São Paulo' in minhaString:  
4     print("Sequência encontrada")  
5 else:  
6     print("Sequência não encontrada")  
7
```

Figura 33

Fonte: Reprodução

Saída:

Result

CPU Time: 0.00 sec(s), Memory: 7752 kilobyte(s)

Sequência encontrada

Figura 34

Fonte: Reprodução

É importante o aprofundamento nos estudos acerca de *strings*, pois essas possibilitam uma infinidade de funções e instruções para sua manipulação.

Em Síntese

Nesta Unidade, estudamos os conceitos introdutórios acerca das principais estruturas de dados e *strings* do Python. É importante que assista à videoaula desta unidade e que leia os livros e materiais complementares indicados nesta Unidade de estudo. É fundamental que, além dos estudos em Python, busque estudar ou retomar conceitos de desenvolvimento de algoritmos, em especial, o tema desta Unidade.

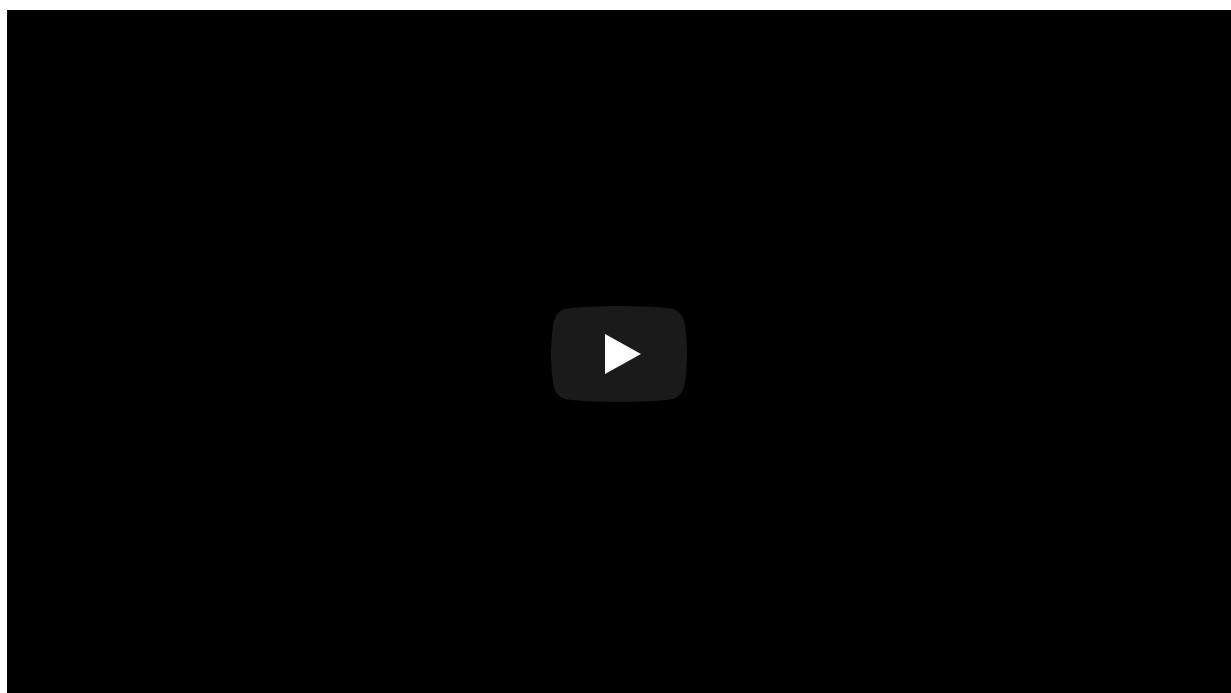


Material Complementar

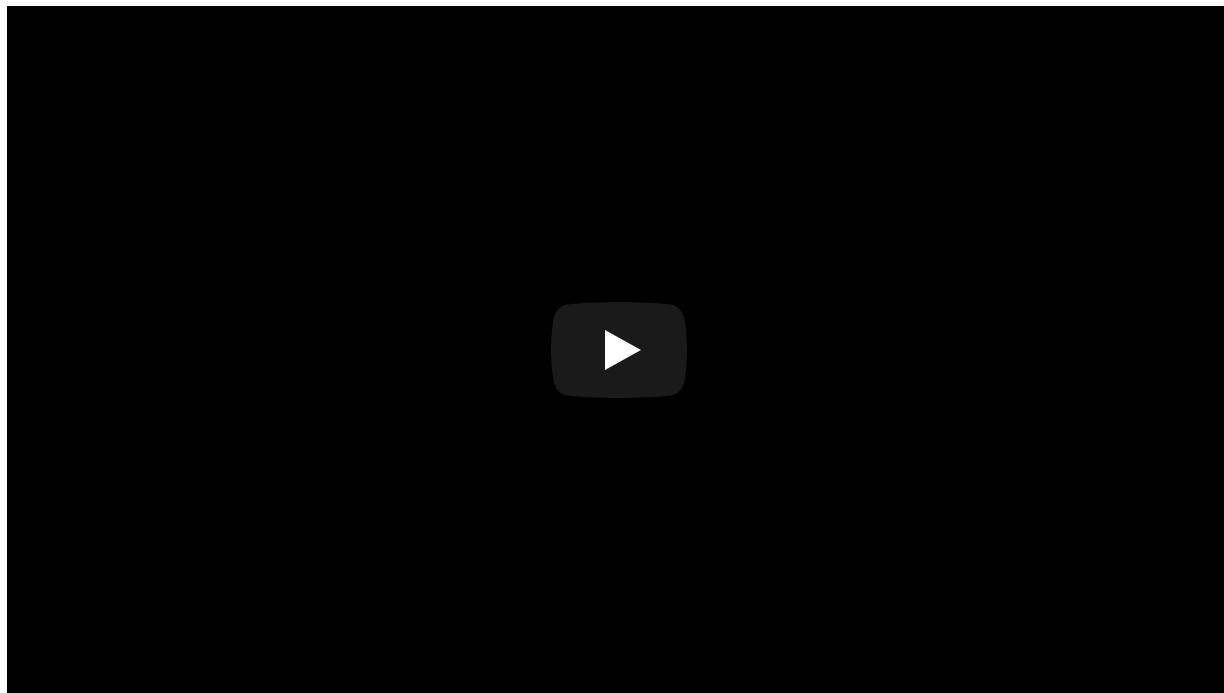
Indicações para saber mais sobre os assuntos abordados nesta Unidade:

Vídeos

Dicionários em Python



O que são as Listas em *Python*? Como Você pode Utilizá-las?



Leitura

Manipulação de *Strings* em *Python*

Clique no botão ao lado para conferir o conteúdo.

ACESSE

Tuplas em *Python*

Clique no botão ao lado para conferir o conteúdo.



 Referências

BANIN, S. L. *Python 3 – Conceitos e Aplicações – Uma abordagem didática*. São Paulo: Érica, 2018. (*e-book*)

PERKOVIC, L. *Introdução à Computação Usando Python – Um Foco no Desenvolvimento de Aplicações*. Rio de Janeiro: LTC, 2016. (*e-book*)

WAZLAWICK, R. *Introdução a Algoritmos e Programação com Python – Uma Abordagem Dirigida por Testes*. Rio de Janeiro: Elsevier, 2017. (*e-book*)