

LISTAS

NO PYTHON

Seu Guia **definitivo** para dominar essa estrutura de dados tão importante e ser um **Pythonista** de verdade!



CONHEÇA A PYTHON ACADEMY

Olá, Pythonista!

Nós existimos para levar o conhecimento de Python a todos e todas. Acreditamos que o Python tem um potencial não apenas como uma linguagem de programação, mas como uma poderosa **ferramenta para mudar seu futuro!**

Através da simplicidade dessa linguagem, é possível fazer coisas **incríveis!** Desenvolvimento Web, Machine Learning, Deep Learning, Blockchain, Análise de Dados, Automatização de tarefas repetitivas... Para tudo isso e mais, **criamos a Python Academy lá em 2018!**

No nosso **Blog** você vai encontrar artigos completos que vão te explicar sobre diversas ferramentas do Python!

Ao baixar nossos **Ebooks** você pode desfrutar do nosso conteúdo onde quer que esteja: no trabalho, no ônibus, no metrô, no avião!

A Python Academy existe para te auxiliar nessa **Jornada**, portanto: **CONTE CONOSCO!**

Vinícius de A. Ramos

Fundador da Python Academy

**Antes de começar, quero
te fazer um convite...**

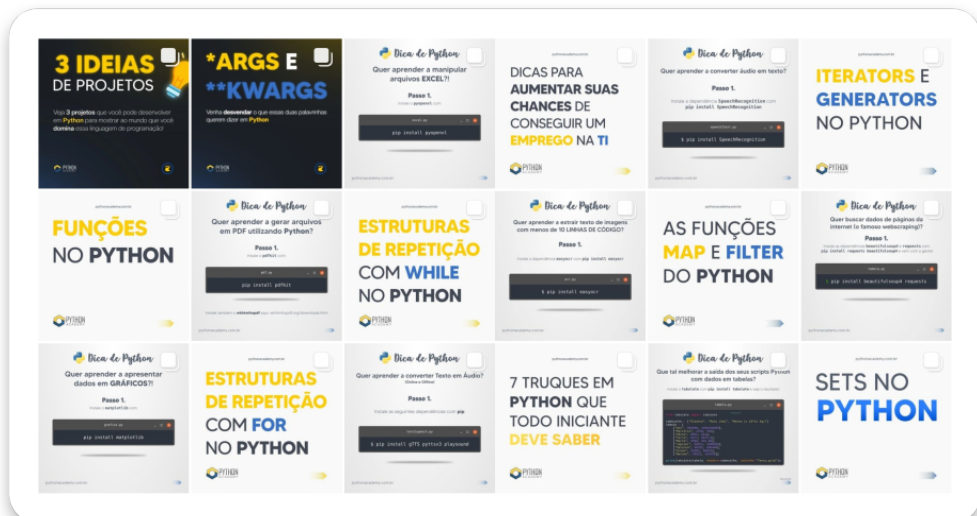


O que você acha de aprender Python com conteúdos diários, posts completos e Quizzes que vão te desafiar a aprender cada vez mais?

Se sua resposta for SIM, gostaríamos de te convidar a seguir a Python Academy no Instagram!

>>> Clique aqui para conhecer nossa página! <<<

Olha o que você está perdendo:



Siga a **Python Academy** no Instagram para conteúdos diários sobre **Python**!

[@pythonacademybr](https://www.instagram.com/pythonacademybr)

Sumário

Apresentação	1
Sumário	3
Introdução	5
CRIANDO LISTAS	6
ACESSANDO OS DADOS DAS LISTAS	7
INDEXAÇÃO NEGATIVA	8
LISTAS DE LISTAS	8
FATIANDO LISTAS (SLICING)	9
PERCORRENDO LISTAS	10
Manipulação de listas	13
MÉTODOS	13
MÉTODO index	13
MÉTODO count	14
MÉTODO append	14
MÉTODO insert	15
MÉTODO extend	15
MÉTODO remove	16
MÉTODO pop	18
MÉTODO clear	18
MÉTODO copy	18

MÉTODO reverse	19
MÉTODO sort	20
KEYWORDS PARA MANIPULAÇÃO DE LISTAS	21
KEYWORD len()	21
KEYWORD min()	21
KEYWORD max()	22
KEYWORD sorted()	22
KEYWORD reversed()	22
List Comprehensions	24
INTRODUÇÃO	24
LIST COMPREHENSIONS COM 'IF'	25
LIST COMPREHENSIONS COM VÁRIOS 'IF'	26
LIST COMPREHENSIONS COM 'IF' E 'ELSE'	27
MÚLTIPLAS LIST COMPREHENSIONS	28

CAPÍTULO 1

INTRODUÇÃO

Uma Lista em Python, é uma coleção ordenada de valores, separados por vírgula e dentro de colchetes [].

Elas são utilizadas para armazenar diversos itens em uma única variável. Entender este conteúdo é de extrema importância para dominar a linguagem por completo!

Abaixo temos um código exemplo de uma lista em Python:

```
# Exemplo de Lista:  
lista = ['Python', 'Academy']  
  
print(lista)
```

Saída do código acima:

```
['Python', 'Academy']
```

Podemos observar o tipo de uma lista com a função **type()**:

```
lista = print(type(lista))
```

E o resultado será:

```
<class 'list'>
```

CRIANDO LISTAS

Existem várias maneiras de se criar uma lista. A maneira mais simples é envolver os elementos da lista com colchetes, por exemplo:

```
# Lista com apenas um elemento  
lista = ["PythonAcademy"]
```

Também podemos criar uma **lista vazia**:

```
lista = []
```

Para criar uma lista com diversos itens, podemos fazer:

```
lista = ['Python' , 'Academy', 2021]
```

Também podemos utilizar a função `list` do próprio Python (*built-in function*):

```
lista = list(["Python Academy"])
```

Outra forma é criar listas resultantes de uma operação de *List Comprehensions*! Assunto que será tratado com mais detalhes no Capítulo 3!

```
[item for item in iteravel]
```

Podemos ainda criar listas através da função `range()`, dessa forma:

```
list(range(10))
```

O que resultará em:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

ACESSANDO OS DADOS DAS LISTAS

Todos os itens de uma lista são indexados, ou seja, para cada item da lista um **índice** é atribuído da seguinte forma: `lista[índice]`.

Exemplo com itens:

```
frutas = ['Maçã', 'Banana', 'Jaca', 'Melão', 'Abacaxi']
```

E assim ficaria a sequência de índices: 0 - Maça, 1 - Banana, 2 - Jaca, 3 - Melão e 4 - Abacaxi. Lembre-se: em Python, **os índices são iniciados em 0**.

Como podemos acessar o primeiro item da lista que é o índice 0?

Veja abaixo:

```
print(frutas[0])
```

A saída como previsível foi a string com a palavra Maça por ocupar o índice 0:

```
Maça
```

Agora vamos falar sobre **Indexação Negativa!**

INDEXAÇÃO NEGATIVA

E se o item desejado for o **último**? Neste momento entramos no conceito de indexação negativa, que significa começar do fim. O índice **-1** irá se referir ao último item e o **-5** ao primeiro.

Dessa forma, para buscar pelo último item da lista teremos:

```
print(frutas[-1])
```

E o resultado será:

```
Abacaxi
```

LISTAS DE LISTAS

Suponha que exista uma lista dentro de uma lista, assim:

```
lista = ['item1', ['python', 'Academy'], 'item3']
```

Como podemos acessar o primeiro índice do item que é uma lista?

A resposta é simples, basta selecionar a posição em que se localiza a lista para ter acesso a ela, assim:

```
sublista = lista[1]  
print(sublista[0])
```

Ou ainda:

```
print(lista[1][0])
```

Ambos obtêm o mesmo resultado:

```
'python'
```

FATIANDO LISTAS (*SLICING*)

O fatiamento de listas, do inglês **slicing**, é a extração de um conjunto de elementos contidos numa lista. Ele é feito da seguinte forma:

```
lista[inicio:fim:passo]
```

Explicando cada elemento:

- **início:** refere-se ao índice de início do fatiamento.

- **fim:** refere-se ao índice final do fatiamento. A lista final não vai conter esse elemento.
- **passo:** é um parâmetro opcional e é utilizado para se pular elementos da lista original

Vamos entender melhor em seguida!

Se quisermos criar uma fatia de uma lista do índice 2 ao 4, podemos fazer da seguinte forma:

```
lista = [10, 20, 30, 40, 50, 60]  
print(lista[2:5])
```

O slicing conta a partir do índice 2 até o índice 5 (mas não o utiliza), pegando os índices 2, 3, 4.

Sua saída será:

```
[30, 40, 50]
```

PERCORRENDO LISTAS

A forma mais comum de percorrer os elementos em uma lista é com um loop **for elemento in lista**, assim:

```
lista = [10, 20, 30, 40, 50, 60]
for num in lista:
    print(num)
```

A saída será:

```
10
20
30
40
50
60
```

Com a função **enumerate()** podemos percorrer também o índice referente a cada valor da lista:

```
lista = [10, 20, 30, 40, 50, 60]

for indice, valor in enumerate(lista):
    print(f'índice={indice}, valor={valor}')
```

Sua saída será:

```
índice=0, valor=10
índice=1, valor=20
índice=2, valor=30
índice=3, valor=40
índice=4, valor=50
```

```
índice=5, valor=60
```

Que tal poupar algumas linhas e obter o mesmo resultado com *List Comprehension*?

```
[print(num) for num in lista]

# Com enumerate:
[print(f'índice={índice}, valor={valor}') for índice,
valor in enumerate(lista)]
```

A saída será a mesma! Mas não se preocupe, trataremos *List Comprehension* com mais detalhes no **Capítulo 3**!

Como vimos neste capítulo, dominar o uso de listas ajuda bastante o desenvolvedor de Python. No próximo capítulo veremos as melhores formas de manipulação de listas em Python.

CAPÍTULO 2

MANIPULAÇÃO DE LISTAS

Neste capítulo iremos tratar dos métodos para manipulação de listas e suas **Keywords** (palavras reservadas).

MÉTODOS

O Python tem vários métodos disponíveis em listas que nos permite manipulá-las. Veremos agora quais são os métodos e como utilizá-los

MÉTODO **index**

Esse método é utilizado para encontrar a posição de um valor especificado. Exemplo:

```
lista = ['Carro', 'Casa', 'Hotel', 'Casa']  
pos = lista.index('Casa')  
print(f'O item desejado está na posição: {pos}')
```

Saída:

```
O item desejado está na posição: 1
```

MÉTODO **count**

O método **count(elemento)** retorna o número de vezes que o valor especificado aparece na lista. Exemplo:

```
lista = ['Carro', 'Casa', 'Hotel', 'Casa']  
pos = lista.count('Casa')  
print(f'O item desejado aparece: {pos}')
```

Saída:

```
O item desejado aparece: 2
```

MÉTODO **append**

Para adicionar um elemento ao final da lista, use o método **append(elemento)**:

```
lista = ['Python', 'Academy']  
lista.append('adicionado')  
print(lista)
```


Saída:

```
['Python', 'Academy', 'adicionado']
```

MÉTODO **insert**

Para adicionar um item em um índice especificado, use o método `insert(índice, elemento)`:

```
lista = ['Python', 'Academy']  
lista.insert(0, 'Blog')  
print(lista)
```

Saída:

```
['Blog', 'Python', 'Academy']
```

MÉTODO **extend**

O método `extend` (iterável) adiciona os elementos de uma lista especificada (ou qualquer outro iterável) ao final da lista:

```
sacola = ['Laranja', 'Banana']  
legumes = ['Xuxu', 'Batata']  
sacola.extend(legumes)
```

```
print(sacola)
```

Saída:

```
['Laranja', 'Banana', 'Xuxu', 'Batata']
```

É possível concatenar as listas para obter o mesmo resultado em uma nova variável. Exemplo:

```
sacola = ['Laranja', 'Banana']  
legumes = ['Xuxu', 'Batata']  
  
juntos = sacola + legumes  
  
print(juntos)
```

Saída:

```
['Laranja', 'Banana', 'Xuxu', 'Batata']
```

E também podemos percorrer uma das listas, adicionando elementos à outra com o método `append()`, assim:

```
sacola = ['Laranja', 'Banana']  
legumes = ['Xuxu', 'Batata']  
  
for legume in legumes:  
    sacola.append(legume)  
  
print(sacola)
```

Saída:

```
['Laranja', 'Banana', 'Xuxu', 'Batata']
```

MÉTODO **remove**

Para remover um item com valor especificado, use o método `remove(elemento)`:

```
lista = ['Blog', 'Python', 'Academy']  
lista.remove('Blog')  
print(lista)
```

Saída:

```
['Python', 'Academy']
```

Outra forma de remover elementos de uma lista é utilizando a função `del` do próprio Python, assim:

```
lista = [10, 20, 30, 40, 50]
del lista[2]
print(lista)
```

Saída:

```
[10, 20, 40, 50]
```

Com a `del` também é possível excluir uma lista completamente:

```
lista = [10, 20, 40, 50]
del lista
print(lista)
```

Aqui o Python irá retornar um erro, já que a variável `lista` não está mais definida no momento da chamada do `print`:

```
Traceback (most recent call last):
  File "listas_testes.py", line 3, in <module>
    print(lista)
NameError: name 'lista' is not defined
```

MÉTODO `pop`

Para remover um item do índice especificado e ainda retorná-lo, use o método `pop(índice)`, dessa forma:

```
lista = ['Banana', 'limão', 'Carro', 'Laranja']  
  
item = lista.pop(2)  
  
print('Item:', item)  
print('Lista: ', lista)
```

Saída:

```
Item: Carro  
Lista: ['Banana', 'limão', 'Laranja']
```

MÉTODO `clear`

Esse método é utilizado para remover todos os elementos de uma lista, dessa forma:

```
lista = [10, 20, 40, 50]  
  
lista.clear()  
  
print(lista)
```

Saída:

```
[]
```

MÉTODO `copy`

Esse método retorna uma cópia da lista especificada.

```
lista = ['Python', 'Academy']  
  
lista_copiada = lista.copy()  
  
print(lista_copiada)  
print(lista)
```

Saída:

```
['Python', 'Academy']  
['Python', 'Academy']
```

MÉTODO `reverse`

O método `reverse` é utilizado para reverter a ordem dos elementos de uma lista:

```
lista = [1, 2, 3, 4, 5]  
  
lista.reverse()  
  
print(lista)
```

Saída:

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

MÉTODO `sort`

Esse método é utilizado para ordenar a lista. Também é possível criar uma função para definir seus próprios critérios de ordenação com `sort(key=funcao)`. Exemplo:

```
lista = [1, 4, 5, 2, 4]
lista.sort()
print(lista)
```

Saída:

```
[1, 2, 4, 4, 5]
```

Adicionando o parâmetro `reverse=True`, é possível ordenar a lista em ordem decrescente. Para deixar do modo padrão basta colocar `reverse=False`:

```
lista = [1, 4, 5, 2, 4]
lista.sort(reverse=True)
print(lista)
```

Saída:

```
[5, 4, 4, 2, 1]
```

KEYWORDS PARA MANIPULAÇÃO DE LISTAS

Nesta parte veremos algumas Keyword da própria linguagem que utilizam listas como parâmetro, executando alguma ação.

KEYWORD `len()`

Retorna a quantidade de itens em uma lista, utilizando o método `len(iterável)`:

```
lista = [10, 20, 30, 40, 50, 60]  
print('Quantidade de itens:', len(lista))
```

Saída:

```
Quantidade de itens: 6
```


KEYWORD `min()`

A função `min(iterável)` devolve o item com menor valor da lista ou iterável de entrada:

```
lista = [2, 4, 8, 1]
print('Menor valor da lista:', min(lista))
```

Saída:

```
Menor valor da lista: 1
```

KEYWORD `max()`

Retorne o maior valor da lista ou iterável especificado `max(iterável)`:

```
lista = [2, 4, 8, 1]
print('Maior valor da lista:', max(lista))
```

Saída:

```
Maior valor da lista: 8
```

KEYWORD `sorted()`

A função `sorted()` é utilizada para ordenar a lista de entrada:

```
lista = [2, 4, 8, 1]

lista_ord = sorted(lista)
print(lista_ord)
```

Saída:

```
[1, 2, 4, 8]
```

KEYWORD `reversed()`

Essa função reverte a ordem da lista de entrada.

```
lista = [1, 2, 3, 4, 7]

for item in reversed(lista):
    print(item)
```

Saída:

```
7
```

```
4  
3  
2  
1
```

Dominando todos os métodos e keywords apresentados, manipular Listas em Python será cada vez mais rápido, prático e eficiente para o desenvolvedor.

No próximo capítulo iremos aprender os conceitos e a manipulação de Listas utilizando o conceito de **List Comprehensions** ou **Compreensão de Listas**.

CAPÍTULO 3

LIST COMPREHENSIONS

Neste capítulo iremos aprender sobre uma ferramenta muito útil no dia a dia do Pythonista: *List Comprehensions*!

Com esse conceito, podemos otimizar a utilização de listas, sua criação e seu manuseio (e de quebra, diminuir algumas linhas de código).

Vamos ver as mais diversas formas de se utilizar essa ferramenta e praticar com exemplos!

INTRODUÇÃO

List Comprehension foi concebida na PEP 202 e é uma forma concisa de criar e manipular listas.

Sua sintaxe básica é a seguinte:

```
[expr for item in lista]
```

Ou seja, aplica a expressão **expr** em cada item da lista.

Exemplo 1: elevar os elementos da lista ao quadrado (à 2ª potência) utilizando **for**:

```
for item in range(10):  
    lista.append(item**2)
```

Podemos reescrevê-lo, utilizando List Comprehensions, da seguinte forma:

```
lista = [item**2 for item in range(10)]
```

Exemplo 2: elementos (palavras) da lista são transformados em maiúsculos:

```
for item in lista:  
    resultado.append(str(item).upper())
```

Podemos reescrevê-lo da seguinte forma, em list comprehensions:

```
resultado = [str(item).upper() for item in lista]
```

LIST COMPREHENSIONS COM “IF”

As estruturas em List comprehensions podem utilizar expressões condicionais para criar listas ou modificar listas existentes. Sua sintaxe básica é:

```
[expr for item in lista if cond]
```

Ou seja, aplica a expressão `expr` em cada item da lista caso a condição `cond` seja satisfeita.

Vamos criar algumas listas utilizando condições. Por exemplo, podemos excluir os números ímpares de um conjunto de número da seguinte forma:

```
resultado = [numero for numero in range(20) if numero % 2 == 0]
```

O que resulta em:

```
resultado = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

A seguir vamos ver como fica com: ‘vários IFs’, ‘IF + ELSE’ e por último o caso de múltiplas List Comprehensions (aninhadas).

LIST COMPREHENSIONS COM VÁRIOS “IF”

Podemos verificar condições em duas listas diferentes dentro da mesma *List Comprehension*.

Por exemplo: gostaríamos de saber os Múltiplos Comuns de 5 e 6. Utilizando múltiplos if's e list comprehensions, podemos criar o seguinte código:

```
resultado = [numero for numero in range(100) if numero  
% 5 == 0 if numero % 6 == 0]
```

Ou seja, o número só será passado para a lista resultado caso sua divisão **por 5 E por 6** seja igual à zero.

O resultado do código acima será:

```
resultado = [0, 30, 60, 90]
```

LIST COMPREHENSIONS COM “IF” E “ELSE”

Outra forma de se utilizar expressões condicionais e list comprehension é usar o conjunto **if + else**. A sintaxe básica para essa construção é:

```
[resultado_if if expr else resultado_else for item in lista]
```

Em outras palavras: para cada item da lista, aplique o resultado **resultado_if** se a expressão **expr** for verdadeira, caso contrário, aplique **resultado_else**.

Por exemplo, se quisermos criar uma lista que contenha “1” quando determinado número for múltiplo de 5 e “0” caso contrário. Podemos codificá-la da seguinte forma:

```
resultado = ['1' if numero % 5 == 0 else '0' for numero in range(16)]
```

Dessa forma, teremos o seguinte resultado:

```
resultado = ['1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1']
```

MÚLTIPLAS LIST COMPREHENSIONS

É aqui que a brincadeira fica séria! Vamos supor que queiramos transpor uma matriz.

Se você não lembra o que é a Transposição de uma Matriz, vamos relembrar:

Transpor uma matriz, significa transformar as linhas em colunas e vice-versa. Ou seja, dada a seguinte matriz em Python:

```
matrix = [[1, 2, 3, 4],  
          [5, 6, 7, 8],  
          [9, 10, 11, 12]]
```

Queremos o seguinte resultado:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}^T$$

Em Python, podemos fazer isso da seguinte forma:

```
transposta = []  
matriz = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
  
for i in range(len(matriz[0])):  
    linha_transposta = []
```

```
for linha in matriz:  
    linha_transposta.append(linha[i])  
transposta.append(linha_transposta)
```

A matriz transposta conteria:

```
transposta = [[1, 4, 9], [2, 5, 10], [3, 6, 11], [4, 8,  
12]]
```

Podemos reescrever o código acima, de transposição de matrizes, da seguinte forma, utilizando list comprehension:

```
matriz = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
transposta = [[linha[i] for linha in matriz] for i in  
range(4)]
```

No código acima:

- No **primeiro loop**, i assume o valor de 0, portanto [linha[0] for linha in matriz] vai retornar o primeiro elemento de cada linha: [1, 4, 9]
- No **segundo loop**, i assume o valor de 1, portanto [linha[1] for linha in matriz] vai retornar o segundo elemento de cada linha: [2, 5, 10]

- No **terceiro loop**, i assume o valor de 2, portanto [linha[2] for linha in matriz] vai retornar o terceiro elemento de cada linha: [3, 6, 11]
- No **quarto loop**, i assume o valor de 3, portanto [linha[3] for linha in matriz] vai retornar o quarto elemento de cada linha: [4, 8, 12]

Obtendo, assim, o mesmo resultado!

Apesar de ser um pouco complexa, a ferramenta de **List Comprehensions** é muito poderosa para que o Pythonista desenvolva seus projetos de forma mais rápida e eficiente, mas para isso é preciso estudá-la e dominá-la.

Nos cursos da Python Academy você irá aprender e dominar LISTAS e **MUITO MAIS!!**

Você também pode aprender Python, de **forma gratuita**, com nossos E-books, artigos do Blog e posts/stories no Instagram, [@pythonacademybr](https://www.instagram.com/pythonacademybr).

Fique ligado e siga a Python Academy!!

