




Trabalhando com Listas em Python

Criando Listas em Python

As **listas** são como caixas organizadas onde podemos guardar vários itens! 

Elas armazenam **qualquer tipo de dado** e permitem mudanças depois da criação.

1 Como criar uma lista?

Podemos criar listas de **três formas principais**:

✓ Usando colchetes  (Forma mais comum)

```
frutas = ["Maçã", "Banana", "Morango"]  
print(frutas)  
# Saída: ['Maçã', 'Banana', 'Morango']
```

✓ Usando o construtor  `list()`

```
numeros = list((1, 2, 3, 4, 5))  
print(numeros)  
# Saída: [1, 2, 3, 4, 5]
```

✓ Usando  `range()` para gerar uma sequência

```
numeros = list(range(1, 6))  
print(numeros)  
# Saída: [1, 2, 3, 4, 5]
```

2 Listas são mutáveis!


Isso significa que **podemos modificar os elementos** depois que a lista for criada!

```
frutas[1] = "Laranja" # Mudamos "Banana" para "Laranja"
print(frutas)
# Saída: ['Maçã', 'Laranja', 'Morango']
```

Resumo Rápido:

Método	Como funciona?	Exemplo
<code>[]</code>	Criação direta	<code>["A", "B", "C"]</code>
<code>list()</code>	Converte tuplas/outros iteráveis	<code>list((1, 2, 3))</code>
<code>range()</code>	Cria sequência numérica	<code>list(range(1, 5))</code>

O que são Listas Aninhadas?

As **listas aninhadas** são listas **dentro de outras listas**!  

Isso permite armazenar dados **em formato de tabela**, como **planilhas ou matrizes**.

Como Criar uma Lista Aninhada?

Podemos criar uma lista **normal**, mas colocando **outras listas dentro dela**.

```
tabela = [
    [1, 2, 3], # Linha 0
    [4, 5, 6], # Linha 1
    [7, 8, 9] # Linha 2
]

print(tabela)
# Saída: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

 Cada **lista interna** representa uma **linha** da tabela.

2 Como Acessar Elementos em Listas Aninhadas?

Usamos **dois índices**:

- O **primeiro índice** representa a **linha**.
- O **segundo índice** representa a **coluna**.

```
print(tabela[0][1]) # Acessa a linha 0, coluna 1  
# Saída: 2
```

```
print(tabela[2][2]) # Acessa a linha 2, coluna 2  
# Saída: 9
```

3 Como Percorrer uma Lista Aninhada? 🔄

Podemos usar **dois loops** `for` para acessar cada item:

```
for linha in tabela:  
    for elemento in linha:  
        print(elemento, end=" ") # Exibe os números na mesma linha  
    print() # Quebra de linha após cada linha da tabela
```

📌 **Saída:**

```
1 2 3  
4 5 6  
7 8 9
```

📌 Resumo Rápido:

Operação	Código Exemplo	Saída
Criar lista aninhada	<code>[[1,2,3], [4,5,6]]</code>	Uma lista dentro de outra
Acessar elemento específico	<code>tabela[1][2]</code>	6
Percorrer lista aninhada	<code>for linha in tabela:</code>	Itera sobre as linhas

🔪 O que é Fatiamento? (Slicing) 🥪

Fatiamento é a **técnica de pegar pedaços de uma sequência** (strings, listas, tuplas, etc.). É como cortar uma fatia de pizza 🍕 ou pegar só um pedaço de bolo 🍰.

📌 Como funciona?

Podemos usar **três parâmetros** dentro de colchetes `[:]` para definir um pedaço de uma sequência:

```
sequencia[início:fim:passo]
```

- ◆ **Início (`start`)** → Onde começa o corte.
- ◆ **Fim (`stop`)** → Onde termina (o último elemento **não é incluído!**).
- ◆ **Passo (`step`)** → De quantos em quantos elementos avançar.

1 Pegando um pedaço da string

```
palavra = "Python"
print(palavra[0:3]) # Pega os caracteres do índice 0 ao 2
# Saída: Pyt
```

2 Pegando do início até um ponto

```
print(palavra[:4]) # Pega do começo até o índice 3
# Saída: Pyth
```

3 Pegando do meio até o final

```
print(palavra[2:]) # Pega do índice 2 até o final
# Saída: thon
```

4 Pulando caracteres

```
print(palavra[::2]) # Pula de 2 em 2
# Saída: Pto
```

5 String ao contrário


```
print(palavra[::-1]) # Inverte a string  
# Saída: nohtyP
```

Resumo Rápido

Sintaxe	O que faz?	Exemplo de saída
<code>texto[a:b]</code>	Pega do índice <code>a</code> até <code>b-1</code>	"Pyt"
<code>texto[:b]</code>	Do começo até <code>b-1</code>	"Pyth"
<code>texto[a:]</code>	De <code>a</code> até o final	"thon"
<code>texto[::n]</code>	Pula <code>n</code> caracteres	"Pto"
<code>texto[::-1]</code>	Inverte a string	"nohtyP"

O que é a função `enumerate()` ?

Às vezes, ao percorrer uma **lista** com um `for`, precisamos saber o **número do índice** de cada item.

A função `enumerate()` nos ajuda a fazer isso de forma fácil! 

1 Como usar `enumerate()` no `for` ?

Sem `enumerate()`, precisaríamos usar um contador manual:

```
frutas = ["Maçã", "Banana", "Morango"]  
  
indice = 0  
for fruta in frutas:  
    print(f"{indice}: {fruta}")  
    indice += 1
```

Saída:

```
0: Maçã  
1: Banana  
2: Morango
```

2 Agora usando `enumerate()` (Forma Correta) ✓

O `enumerate()` já nos dá o **índice** e o **valor**, sem precisar de um contador!

```
frutas = ["Maçã", "Banana", "Morango"]
```

```
for indice, fruta in enumerate(frutas):  
    print(f"{indice}: {fruta}")
```

📌 Saída:

```
0: Maçã  
1: Banana  
2: Morango
```

Muito mais simples, né? 🚀🐍

3 Começando com um índice diferente

Podemos dizer ao `enumerate()` **por qual número começar**:

```
for indice, fruta in enumerate(frutas, start=1):  
    print(f"{indice}: {fruta}")
```

📌 Saída:

```
1: Maçã  
2: Banana  
3: Morango
```

Agora a contagem começa em `1` ao invés de `0`! 🎯

📌 Resumo Rápido

Método	O que faz?	Exemplo de saída
<code>enumerate(lista)</code>	Retorna índice e valor	<code>0: Maçã</code>
<code>enumerate(lista, start=n)</code>	Começa a contagem do número <code>n</code>	<code>1: Maçã</code>



O que é Compreensão de Listas?

A **compreensão de listas** (*list comprehension*) é uma maneira rápida e elegante de criar listas **em uma única linha de código**. 🎯

Ela é útil para:

- ✅ Criar uma **nova lista** baseada em outra.
- ✅ **Filtrar** elementos.
- ✅ **Modificar** elementos antes de adicioná-los à nova lista.

1 Criando uma lista com **for** (Forma Tradicional)

Sem compreensão de listas, faríamos assim:

```
numeros = [1, 2, 3, 4, 5]
quadrados = []

for num in numeros:
    quadrados.append(num ** 2)

print(quadrados)
# Saída: [1, 4, 9, 16, 25]
```

2 Agora com Compreensão de Lista ✅

Compreensão de listas permite reduzir tudo para **uma linha**!

```
quadrados = [num ** 2 for num in numeros]
print(quadrados)
# Saída: [1, 4, 9, 16, 25]
```



Mais curto, mais rápido e mais legível! 🚀🐍

3 Filtrando Elementos na Compreensão de Lista

Podemos adicionar um **if** para filtrar valores.

```
pares = [num for num in numeros if num % 2 == 0]
print(pares)
```

```
# Saída: [2, 4]
```

◆ Pegamos apenas os números pares! 🎯

4 Modificando Elementos na Compreensão de Lista

Podemos **transformar os valores** antes de adicioná-los à nova lista:

```
nomes = ["Alice", "bob", "carlos"]
nomes_maiusculos = [nome.upper() for nome in nomes]
print(nomes_maiusculos)
# Saída: ['ALICE', 'BOB', 'CARLOS']
```

◆ Transformamos todos os nomes para maiúsculas! 📖

📌 Resumo Rápido

Operação	Código Exemplo	Saída
Criar nova lista baseada em outra	<code>[x * 2 for x in lista]</code>	<code>[2, 4, 6]</code>
Filtrar valores	<code>[x for x in lista if x > 10]</code>	<code>[12, 15]</code>
Modificar elementos	<code>[x.upper() for x in lista]</code>	<code>['A', 'B', 'C']</code>

📌 Métodos da Classe `list` em Python

A **classe** `list` em Python tem vários métodos úteis para manipular listas. Vamos ver os mais importantes com exemplos práticos!

1 Adicionar Elementos à Lista

✓ `append()` → Adiciona um item ao final da lista.

```
frutas = ["Maçã", "Banana"]
frutas.append("Morango")
print(frutas)
# Saída: ['Maçã', 'Banana', 'Morango']
```

✓ `insert()` → Adiciona um item em uma posição específica.


```
frutas.insert(1, "Uva")
print(frutas)
# Saída: ['Maçã', 'Uva', 'Banana', 'Morango']
```

✓ `extend()` → Junta duas listas.

```
frutas.extend(["Pera", "Melancia"])
print(frutas)
# Saída: ['Maçã', 'Uva', 'Banana', 'Morango', 'Pera', 'Melancia']
```

2 Remover Elementos da Lista

✓ `remove()` → Remove o **primeiro** item encontrado.

```
frutas.remove("Uva")
print(frutas)
# Saída: ['Maçã', 'Banana', 'Morango', 'Pera', 'Melancia']
```

✓ `pop()` → Remove **e retorna** um item pelo índice (ou o último, se vazio).

```
item_removido = frutas.pop(2)
print(item_removido) # Saída: 'Morango'
print(frutas) # Saída: ['Maçã', 'Banana', 'Pera', 'Melancia']
```

✓ `clear()` → Remove todos os itens da lista.

```
frutas.clear()
print(frutas)
# Saída: []
```

3 Ordenar e Inverter Listas

✓ `sort()` → Ordena a lista **em ordem crescente**.

```
numeros = [5, 3, 8, 2]
numeros.sort()
```

```
print(numeros)
# Saída: [2, 3, 5, 8]
```

✓ `sort(reverse=True)` → Ordena a lista **em ordem decrescente**.

```
numeros.sort(reverse=True)
print(numeros)
# Saída: [8, 5, 3, 2]
```

✓ `reverse()` → Inverte a ordem da lista.

```
numeros.reverse()
print(numeros)
# Saída: [2, 3, 5, 8]
```

4 Buscar e Contar Elementos

✓ `index()` → Retorna o índice da **primeira ocorrência** do elemento.

```
nomes = ["Ana", "Carlos", "Bianca"]
posicao = nomes.index("Carlos")
print(posicao)
# Saída: 1
```

✓ `count()` → Conta quantas vezes um item aparece na lista.

```
numeros = [1, 2, 2, 3, 4, 2]
print(numeros.count(2))
# Saída: 3
```

5 Criar Cópias de Listas

✓ `copy()` → Cria uma cópia da lista (sem alterar a original).

```
numeros_copia = numeros.copy()
print(numeros_copia)
# Saída: [1, 2, 2, 3, 4, 2]
```

Resumo Rápido

Método	O que faz?	Exemplo
<code>append(x)</code>	Adiciona <code>x</code> no final	<code>lista.append(5)</code>
<code>insert(i, x)</code>	Adiciona <code>x</code> no índice <code>i</code>	<code>lista.insert(1, 9)</code>
<code>extend(lista2)</code>	Junta duas listas	<code>lista.extend([7,8])</code>
<code>remove(x)</code>	Remove o primeiro <code>x</code>	<code>lista.remove(3)</code>
<code>pop(i)</code>	Remove e retorna item do índice <code>i</code>	<code>lista.pop(2)</code>
<code>clear()</code>	Remove todos os elementos	<code>lista.clear()</code>
<code>sort()</code>	Ordena a lista em ordem crescente	<code>lista.sort()</code>
<code>sort(reverse=True)</code>	Ordena em ordem decrescente	<code>lista.sort(reverse=True)</code>
<code>reverse()</code>	Inverte a ordem da lista	<code>lista.reverse()</code>
<code>index(x)</code>	Retorna o índice do primeiro <code>x</code>	<code>lista.index(4)</code>
<code>count(x)</code>	Conta quantas vezes <code>x</code> aparece	<code>lista.count(2)</code>
<code>copy()</code>	Cria uma cópia da lista	<code>lista2 = lista.copy()</code>

By **MChiodi**