

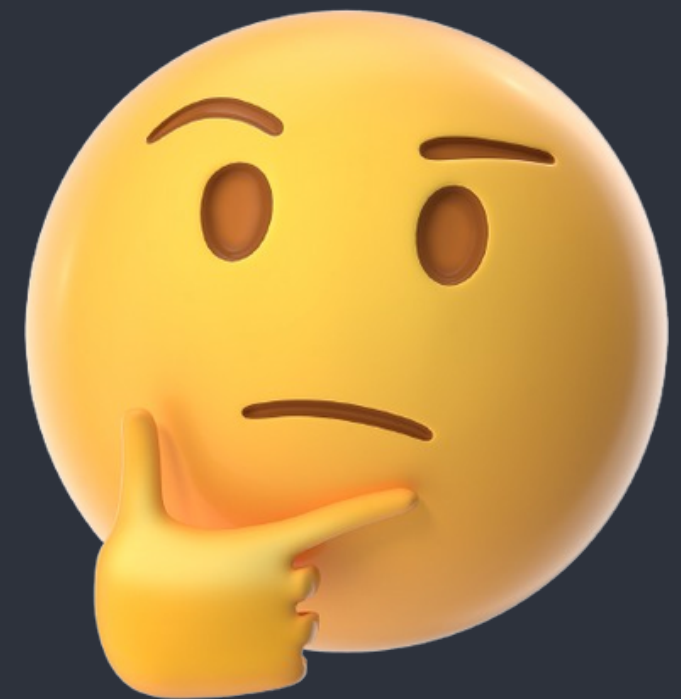
</

Listas

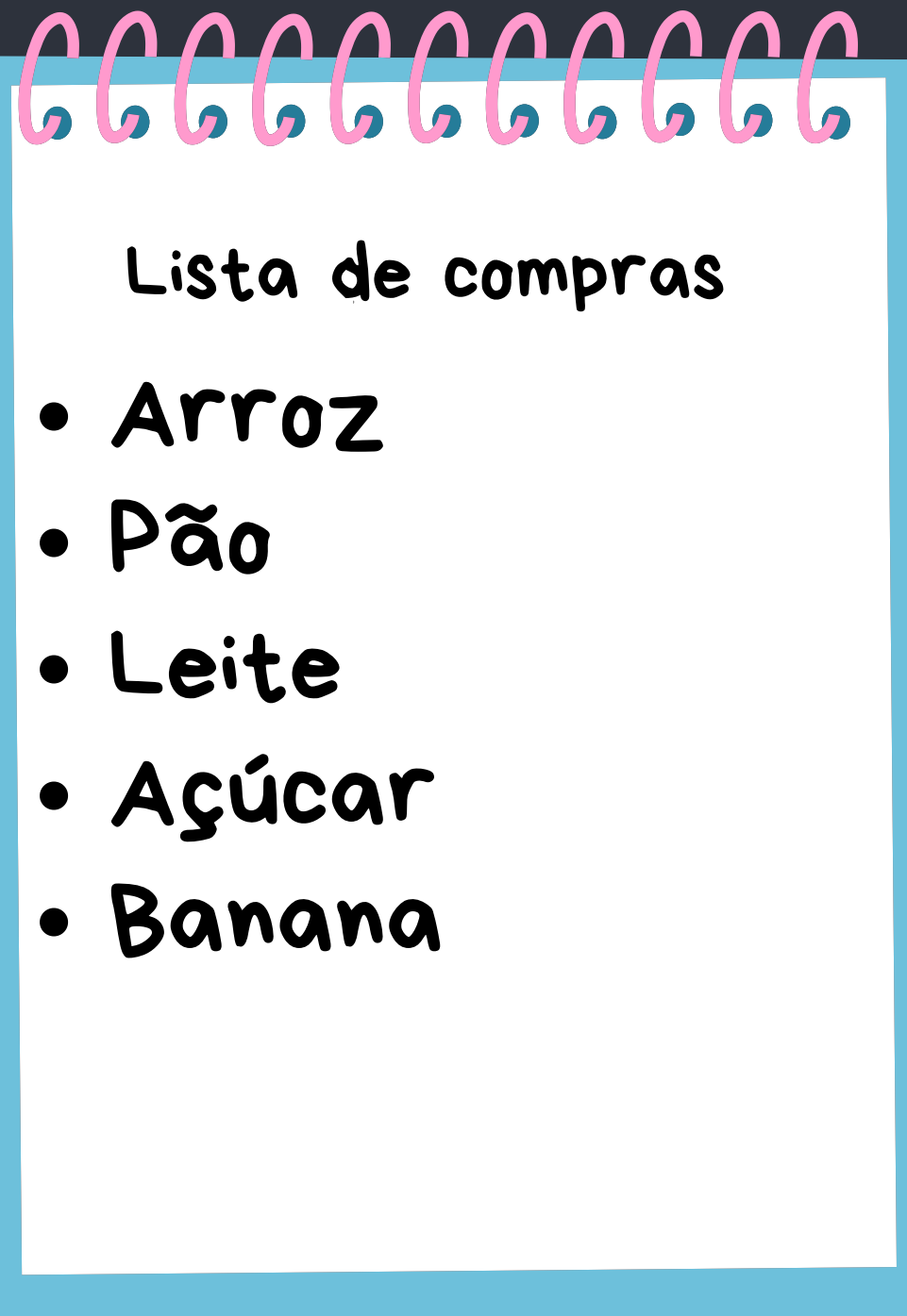
/>

# </ Listas

Imagine que você está indo  
ao supermercado e precisa  
anotar o que comprar



# </ Lista de Compras



# </ Listas em Python

Para criar uma lista em python utilizaremos os colchetes ([ ])

```
1 lista = []
```

# </ Listas em Python

Podemos criar a lista com dados pré definidos:

```
1 lista = ['item1', 2, 3.14]
```

Listas em python  
são heterogêneas,  
ou seja, aceita  
mais de um tipo  
de dado



# </ Lista de Compras

Exercício:

Crie a lista de compras em python e apresente o resultado

Lista de compras

- Arroz
- Pão
- Leite
- Açúcar
- Banana

## </ Lista de Compras

```
1  lista_de_compras = ['arroz', 'pão',  
2  |         |         |         |         |         'leite', 'açúcar',  
3  |         |         |         |         |         'banana']  
4  
5  print("lista de compras: ", lista_de_compras)  
lista de compras: ['arroz', 'pão', 'leite', 'açúcar', 'banana']
```



# </ Lista de Compras

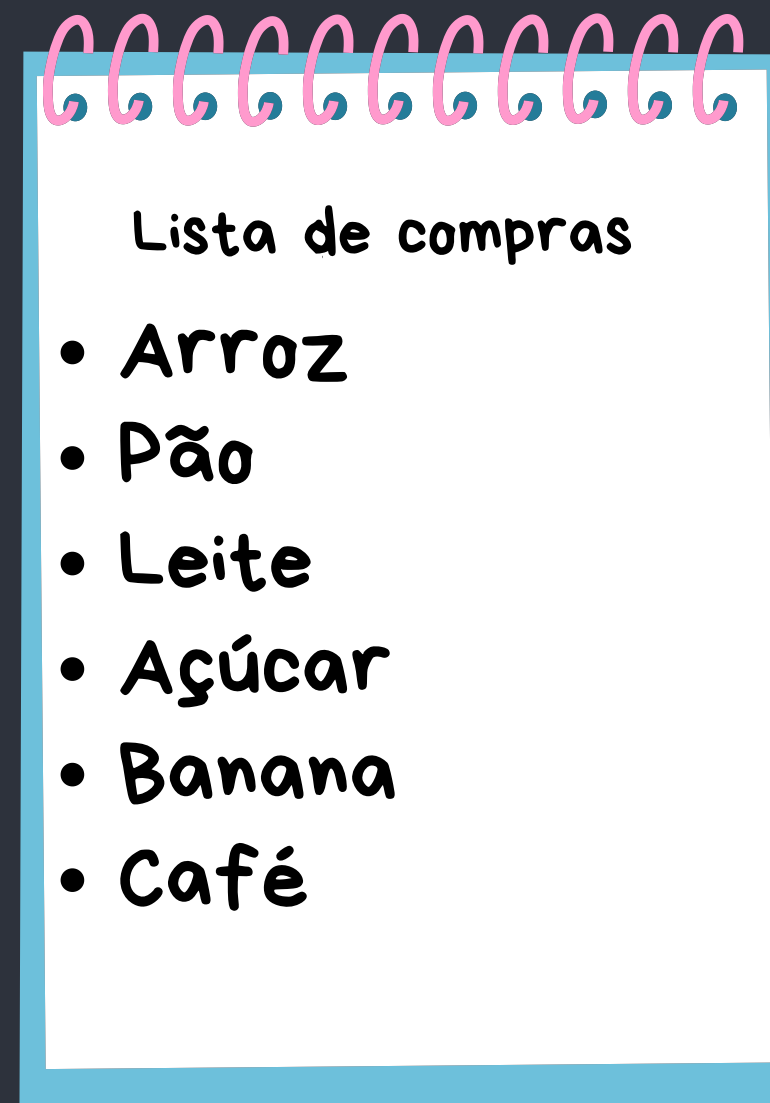
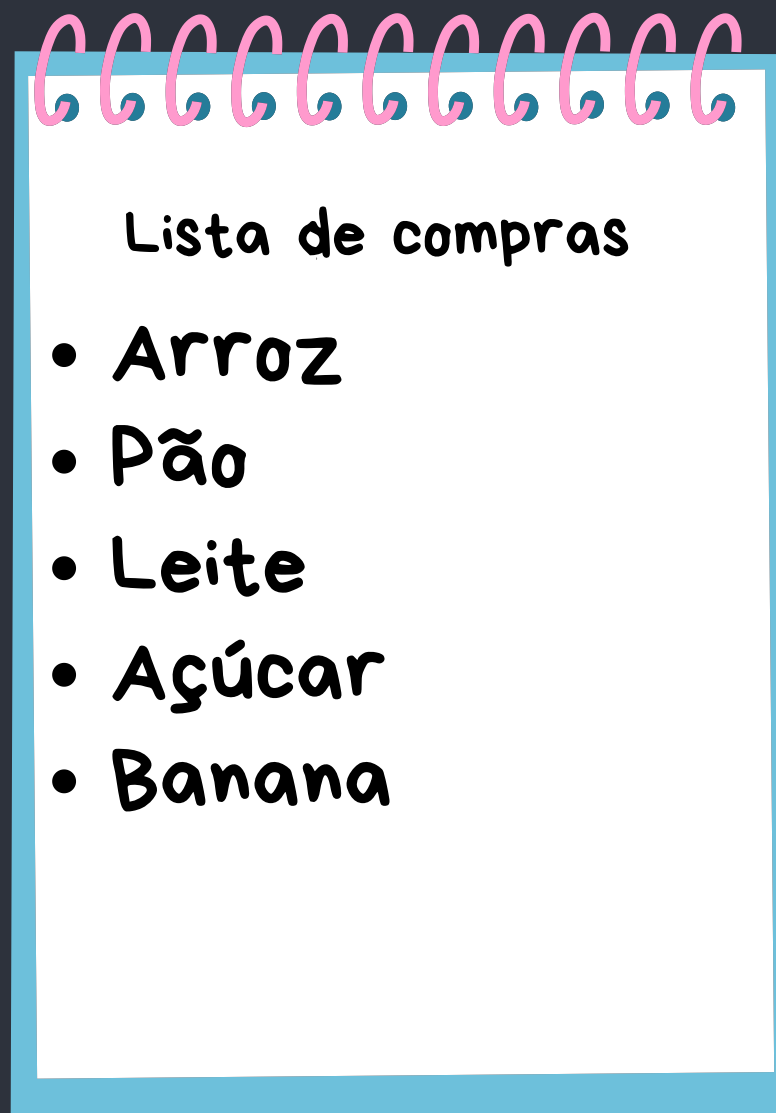
E se eu quiser adicionar café a minha lista de compras, como faço?





# </ Lista de Compras

Adicionamos um item ao final da lista já existente



## </ O método append()

Em python utilizamos o método append para adicionar um novo item a uma lista já existente:

```
1  lista = ["item1", 2, 3.14]
2
3  print("lista: ", lista)
4
5  lista.append("novo item")
6
7  print("lista atualizada: ", lista)
lista: ['item1', 2, 3.14]
lista atualizada: ['item1', 2, 3.14, 'novo item']
```

## </ O método append()

Outra forma de se adicionar um item a uma lista:

```
1  lista = ['item1', 2, 3.14]
2
3  print("lista: ", lista)
4
5  lista = lista + ['novo item']
6
7  print("lista: ", lista)
lista: ['item1', 2, 3.14]
lista atualizada: ['item1', 2, 3.14, 'novo item']
```

O que estamos fazendo aqui é a concatenação de listas, ou seja, unindo duas listas



# </ Lista de Compras

## Exercício:

Crie a lista de compras em python, e depois adicione o item “café” a ela e apresente o resultado

O método `append()` recebe apenas um parâmetro que pode ser de qualquer tipo (string, int, float, etc)



### Lista de compras

- Arroz
- Pão
- Leite
- Açúcar
- Banana

# </ Lista de Compras

```
1  lista_de_compras = ['arroz', 'pão',  
2  | | | | | 'leite', 'açúcar',  
3  | | | | | 'banana']  
4  
5  print("lista de compras: ", lista_de_compras)  
6  
7  lista_de_compras.append('café')  
8  
9  print("lista de compras atualizada: ", lista_de_compras)
```

```
lista de compras: ['arroz', 'pão', 'leite', 'açúcar', 'banana']  
lista de compras atualizada: ['arroz', 'pão', 'leite', 'açúcar', 'banana', 'café']
```



## </ Atividade

Faça um código que receba os produtos e salve eles em uma lista de compras.

- A lista pode receber qualquer quantidade de produtos
- O código deve parar quando o usuário digitar “sair”
- Apresente a lista de compras ao final do código

entrada:

```
Arroz  
Pão  
Leite  
Macarrão  
sair
```

Saída:

```
Lista de compras: ['Arroz', 'Pão', 'Leite', 'Macarrão']
```

# </ Índices

Os Índices são os endereços de cada item na lista, como se fosse o “número da casa” deles

```
1 casas = ['Gabriel', 'Letícia', 'Vamberto']
```



# </ Índices

Para descobrirmos quem mora na casa de número dois vamos utilizar o seguinte código:

```
1  casas = ['Gabriel', 'Letícia', 'Vamberto']
2
3  print("Morador da casa 2:", casas[2])
Morador da casa 2: Vamberto
```

Sintaxe: lista[índice]

Importante!  
os índices das listas sempre  
iniciam em 0, então se  
queremos o terceiro item da  
lista iremos utilizar lista[2]  
já que teremos as posições  
0, 1 e 2





## </ Índices

E se tentarmos acessar uma posição que não exista?

```
1  casas = ['Gabriel', 'Letícia', 'Vamberto']
2
3  print("Morador da casa 3:", casas[3])
Traceback (most recent call last):
  File "c:\Users\gabri\Documents\pop\aula 1\2-condicional.py", line 3, in <module>
    print("Morador da casa 3:", casas[3])
                                ~~~~~^
IndexError: list index out of range
```

A casa de número 3 não existe, logo não é possível acessar

## </ Índices

E se eu quiser acessar o último valor da lista? ou o penúltimo?  
Podemos utilizar índices negativos:

```
1  casas = ['Gabriel', 'Letícia', 'Vamberto']
2
3  print("Morador da última casa:", casas[-1])
Morador da última casa: Vamberto
```

## </ Índices

E se eu quiser acessar o último valor da lista? ou o penúltimo?  
Podemos utilizar índices negativos:

Índices positivos:    0                      1                      2  
                             |                      |                      |

```
1  casas = ['Gabriel', 'Letícia', 'Vamberto']
```

Índices Negativos:    -3                      -2                      -1  
                             |                      |                      |

## </ Índices

E se eu quiser acessar o último valor da lista? ou o penúltimo?

Podemos utilizar índices negativos:

Índices positivos:    0                      1                      2  
                             |                      |                      |

```
1  casas = ['Gabriel', 'Letícia', 'Vamberto']
```

Índices Negativos:    -3                      -2                      -1  
                             |                      |                      |

## </ Índices

Com os índices eu posso trocar os valores de um item

```
1  casas = ['Gabriel', 'Letícia', 'Vamberto']
2
3  print("Moradores:", casas)
4
5  casas[1] = 'Natália'
6
7  print("Moradores:", casas)
Moradores: ['Gabriel', 'Letícia', 'Vamberto']
Moradores: ['Gabriel', 'Natália', 'Vamberto']
```

## </ Índices



## </ Atividade

Faça um código que altere o primeiro e o último valor (usando índices negativos) da lista abaixo:

```
Lista de compras: ['Arroz', 'Pão', 'Leite', 'Macarrão']
```

## </ Atividade

```
1  lista_de_compras = ['Arroz', 'Pão', 'Leite', 'Macarrão']
2
3  print(lista_de_compras)
4
5  lista_de_compras[0] = 'Feijão'
6
7  lista_de_compras[-1] = 'Banana'
8
9  print(lista_de_compras)
['Arroz', 'Pão', 'Leite', 'Macarrão']
['Feijão', 'Pão', 'Leite', 'Banana']
```





## </ len

A função len() é utilizada para descobrir quantos elementos existem na lista

```
1  lista_de_compras = ['Arroz', 'Pão', 'Leite', 'Macarrão']  
2  
3  print(len(lista_de_compras))
```

</ for

Listas e loops são frequentemente usados juntos para processar dados de maneira eficiente

Em python, podemos percorrer uma lista com um for de forma fácil e eficiente

</ for

Oque você acha que esse código faz?

```
1  lista_de_compras = ['Arroz', 'Pão', 'Leite', 'Macarrão']
2
3  for item in lista_de_compras:
4      print('Item atual:', item)
```

# </ for

```
1  lista_de_compras = ['Arroz', 'Pão', 'Leite', 'Macarrão']
2
3  for item in lista_de_compras:
4      print('Item atual:', item)
```

O código irá percorrer toda a lista, e a cada iteração a variável “item” vai assumir um valor de lista\_de\_compras (do primeiro ao último índice)

```
Item atual: Arroz
Item atual: Pão
Item atual: Leite
Item atual: Macarrão
```

## </ Atividade

Faça um código que, com base na lista abaixo, diga se o aluno foi aprovado ou reprovado (notas iguais ou maiores que 7 estão aprovados)

```
1 notas = [10, 4, 5, 7, 9, 4]
```

## </ Atividade

```
1  notas = [10, 4, 5, 7, 9, 4]
2
3  for nota in notas:
4      if nota >= 7:
5          print('Aprovado')
6      else:
7          print('Reprovado')
```

```
Aprovado
Reprovado
Reprovado
Aprovado
Aprovado
Reprovado
```



## </ Atividade

Faça um código que, com base na lista abaixo, calcule a média de notas da turma

```
1 notas = [10, 4, 5, 7, 9, 4]
```

## </ Atividade

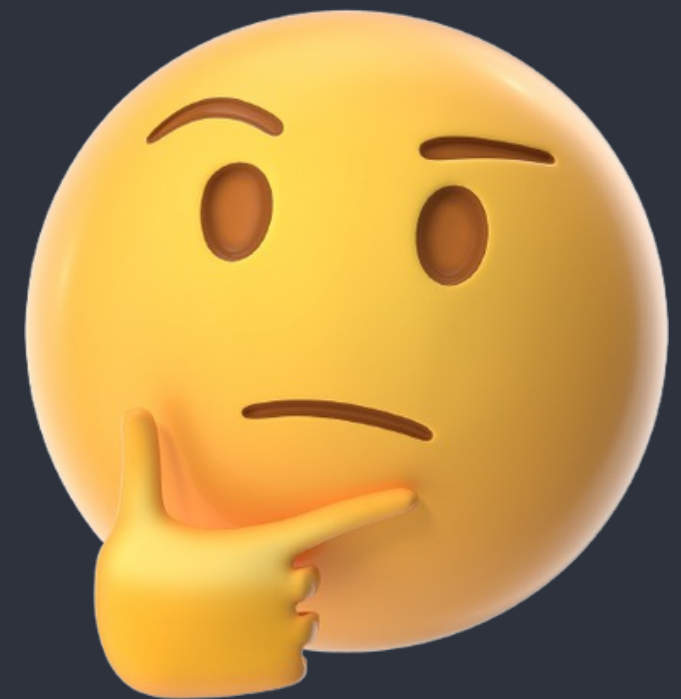
```
1  notas = [10, 4, 5, 7, 9, 4]
2  soma = 0
3
4  for nota in notas:
5      soma += nota
6
7  media = soma / len(notas)
8
9  print("Média:", media)
Média: 6.5
```





`</ for`

E se eu quiser alterar os valores da lista dentro do loop?



# </ for

usamos um loop **for** junto com a função **range(len())**

**range(len())** cria um intervalo que vai de 0 até o comprimento da lista - 1. Isso nos permite iterar sobre os índices da lista.

```
1  valores = [0, 1, 2, 3, 4, 5]
2
3  for i in range(len(valores)):
4      valores[i] = valores[i] * 10
5
6  print(valores)
[0, 10, 20, 30, 40, 50]
```

## </ Atividade

Faça um código que, dada a lista de salários abaixo, acrescente 10% a cada valor e apresente o resultado

```
salarios = [2000, 2500, 3000, 3500, 4000]
```

## </ Atividade

```
1  salarios = [2000, 2500, 3000, 3500, 4000]
2
3  porcentagem = 0.1 # ou 10/100 (10%)
4
5  for i in range(len(salarios)):
6      |      salarios[i] += salarios[i] * porcentagem
7
8  print("Salários com aumento:", salarios)
Salários com aumento: [2200.0, 2750.0, 3300.0, 3850.0, 4400.0]
```



# </ Métodos e funções

Agora vamos falar dos métodos e das funções das listas

- **Métodos:**

Geralmente alteram algo na lista alvo (adicionar, remover, alterar a ordem)

**Sintaxe:**

**lista.metodo()**

- **Funções:**

Nunca alteram a lista, apenas leem os dados e manipulam eles. O resultado deve ser salvo em alguma variável ou utilizados na hora se não serão perdidos.

**Sintaxe:**

**funcao(lista)**

## </ Métodos de inserção

Os métodos de inserção adicionam dados a uma lista

## </ Métodos de inserção

Método .append(item)

Método append()

```
1  lista = [1, 2]
2
3  lista.append(3)
4
5  print(lista)
6
7  # saída: [1, 2, 3]
```

Como já visto anteriormente, o método append adiciona um novo item ao final da lista

# </ Métodos de inserção

Método `.insert(indice, item)`

Método `insert()`

```
1  lista = [1, 2]
2
3  lista.insert(0, 3)
4
5  print(lista)
6
7  # saída: [3, 1, 2]
```

Semelhante ao método `append`, ele irá adicionar um item a lista, mas recebendo o índice em que será adicionado



# </ Métodos de inserção

Método .extend(item: lista)

Método insert()

```
1 lista = [1, 2]
2 lista2 = [3, 4]
3
4 lista.extend(lista2)
5
6 print(lista)
7
8 # saída: [1, 2, 3, 4]
```

O método extend irá juntar duas listas, a lista base com a passada como parâmetro semelhante a concatenação vista anteriormente

## </ Métodos de remoção

Os métodos de remoção são removem dados a uma lista

# </ Métodos de remoção

Método .pop(indice = -1)

## Método pop()

```
1 lista = [1, 2, 3, 4, 5]
2 lista.pop()
3
4 print(lista)
5 # [1, 2, 3, 4]
6 lista.pop(1)
7
8 print(lista)
9 #[1, 3, 4]
```

O método pop irá remover o último item da lista

É possível passar um parâmetro índice, removendo assim o item no índice indicado ao invés do último

Caso o índice não exista na lista irá causar um IndexError

# </ Métodos de remoção

Método .remove(item)

Método remove()

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

O método remove irá remover o item passado como parâmetro  
Caso o item não exista na lista irá causar um ValueError

## </ Métodos de remoção

Método .clear()

Método clear()

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

O método clear irá apagar todos os dados da lista

# </ Métodos de remoção

Palavra-chave del lista[indice]

palavra-chave del

```
1  lista = [1, 2, 3, 4, 5]
2  del lista[1]
3
4  print(lista)
5  # [1, 3, 4, 5]
```

Diferente dos anteriores, o del é uma palavra-chave do python para remoção de dados de listas

Sintaxe: del lista[indice]

Sua funcionalidade é semelhante a do .pop(), sendo obrigatório o índice

## </ Métodos de ordenação

Os métodos de alteram a ordem dos itens da lista

# </ Métodos de ordenação

Método .sort(reverse = False)

Método sort()

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

O método sort irá ordenar os itens da lista, em ordem numérica ou alfabética. Para poder utilizar esse método é necessário que todos os itens sejam do mesmo tipo, caso exista algum diferente irá causar um `TypeError`. Caso o parâmetro `reverse` for passado como `True` a lista será ordenada e depois retornada invertida.



# </ Métodos de ordenação

Método .reverse()

Método reverse()

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

O método reverse irá retornar a lista invertida (primeiro item como último, segundo como penúltimo)

Diferente do reverse o sort(), esse não irá ordenar a lista, apenas inverter

Os itens não precisam ser do mesmo tipo para utilizar esse método

## </ Métodos de informação

Os métodos de informação retornam informações de uma lista

# </ Métodos de informação

Método .count(item)

Método count()

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

O método count irá contar quantas ocorrências do item passado como parâmetro existe na lista

# </ Métodos de informação

Método .index(item)

Método index()

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

O método index irá retornar o índice do **PRIMEIRO** item passado como parâmetro na lista

Caso o item não esteja na lista irá causar um ValueError

## </ Funções de listas

As funções de listas são funções que RECEBEM listas como parâmetros e retornam dados relacionados aquela lista, sem alterar a lista original

# </ Funções de listas

Função len(lista)

Função index()

```
1  lista = [1, 3, 2, 5, 4, 2, 2, 2]
2  print(len(lista))
3
4  # 8
```

Como visto anteriormente, a função len irá retornar a quantidade de itens na lista

# </ Funções de listas

Função min(lista)

Função min()

```
1  lista = [1, 3, 2, 5, 4, 2, 2, 2]
2  print(min(lista))
3
4  # 1
```

A função min irá retornar o menor item na lista

A lista precisa ter apenas um tipo de item nela, senão irá causar um TypeError

Caso os itens sejam do tipo string, ela irá retornar o valor com menor quantidade de letras e mais próximo do 'a'

# </ Funções de listas

Função max(lista)

Função max()

```
1  lista = [1, 3, 2, 5, 4, 2, 2, 2]
2  print(max(lista))
3
4  # 5
```

A função max irá retornar o maior item na lista

A lista precisa ter apenas um tipo de item nela, senão irá causar um TypeError

Caso os itens sejam do tipo string, ela irá retornar o valor com maior quantidade de letras e mais próximo do 'z'



# </ Funções de listas

Função sum(lista)

Função sum()

```
1  lista = [1, 3, 2, 5, 4, 2, 2, 2]
2  print(sum(lista))
3
4  # 21
```

A função sum irá retornar a soma de todos os itens da lista

A lista precisa ter apenas itens do tipo numérico, senão irá causar um TypeError

# </ Funções de listas

Função sorted(lista)

Função sorted()

```
1  lista = [1, 3, 2, 5, 4, 2, 2, 2]
2  print(sorted(lista))
3
4  # [1, 2, 2, 2, 2, 3, 4, 5]
```

A função sorted retorna uma nova lista ordenada com base nos elementos da lista original

Diferente do método sort(), ela não modifica a lista original, mas retorna uma nova lista ordenada.

# </ Funções de listas

Função any(lista)

Função any()

```
1  lista = [True, False, True, True]
2  print(any(lista))
3
4  # True
```

A função any() retorna True se qualquer elemento na lista for verdadeiro

**Atenção, qualquer item diferente de False, None e "" é considerado True**

# </ Funções de listas

Função all(lista)








Função all()

```
1  lista = [True, False, True, True]
2  print(all(lista))
3
4  # False
```

A função all() retorna True se **TODOS** os elementos da lista forem verdadeiro

**Atenção, qualquer item diferente de False, None e "" é considerado True**

# </ Revisão

Python List Methods CheatSheet		
Input	Method Name	Output
	<code>append()</code>	
	<code>count()</code>	2
	<code>copy()</code>	
	<code>index()</code>	2
	<code>insert(1, )</code>	
	<code>reverse()</code>	
	<code>pop()</code>	
	<code>clear()</code>	
	<code>pop(1)</code>	