

ESTRUTURAS DE DADOS EM PYTHON



LISTAS EM PYTHON

Uma lista em Python é uma coleção ordenada e mutável de elementos. Diferente de algumas outras linguagens, uma lista em Python pode conter elementos de tipos diferentes, como números, strings, e até outras listas.

- Ordenada: A ordem dos elementos na lista é mantida.
- Mutável: Você pode modificar uma lista após sua criação (adicionar, remover ou alterar elementos).
- Dinâmica: O tamanho da lista pode mudar dinamicamente conforme elementos são adicionados ou removidos.

LISTAS EM PYTHON

Listas são criadas utilizando colchetes [], com os elementos separados por vírgulas.

```
# Lista vazia
minha_lista = []

# Lista de números inteiros
numeros = [1, 2, 3, 4, 5]

# Lista de strings
cidades = ["São Paulo", "Rio de Janeiro", "Belo Horizonte"]

# Lista mista (diferentes tipos de dados)
mista = [1, "Python", 3.14, True]
```

LISTAS EM PYTHON


Você pode acessar elementos individuais de uma lista utilizando o índice do elemento. Em Python, os índices começam em 0.

```
python Copy code  
  
cidades = ["São Paulo", "Rio de Janeiro", "Belo Horizonte"]  
  
# Acessando o primeiro elemento  
print(cidades[0]) # Saída: São Paulo  
  
# Acessando o segundo elemento  
print(cidades[1]) # Saída: Rio de Janeiro  
  
# Acessando o último elemento  
print(cidades[-1]) # Saída: Belo Horizonte
```

LISTAS EM PYTHON

Você pode modificar elementos de uma lista acessando-os pelo índice e atribuindo novos valores.

python

 Copy code

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

# Modificando o terceiro elemento
numeros[2] = 10
print(numeros) # Saída: [1, 2, 10, 4, 5]
```

MÉTODOS E OPERAÇÕES COMUNS

- `append(elemento)`: Adiciona um elemento ao final da lista.
- `insert(indice, elemento)`: Insere um elemento em uma posição específica.

```
python Copy code  
  
lista = [1, 2, 3]  
lista.append(4)  
print(lista) # Saída: [1, 2, 3, 4]  
  
lista.insert(1, 5)  
print(lista) # Saída: [1, 5, 2, 3, 4]
```

MÉTODOS E OPERAÇÕES COMUNS


- `remove(elemento)`: Remove a primeira ocorrência de um elemento.
- `pop(indice)`: Remove e retorna o elemento na posição especificada. Se o índice não for especificado, remove o último elemento.
- `clear()`: Remove todos os elementos da lista.

```
python Copy code  
  
lista = [1, 2, 3, 4, 2]  
lista.remove(2)  
print(lista) # Saída: [1, 3, 4, 2]  
  
lista.pop(1)  
print(lista) # Saída: [1, 4, 2]  
  
lista.clear()  
print(lista) # Saída: []
```

MÉTODOS E OPERAÇÕES COMUNS

- `len(lista)`: Retorna o número de elementos na lista.
- `sort()`: Ordena a lista em ordem crescente.
- `reverse()`: Inverte a ordem dos elementos na lista.

python

 Copy code

```
lista = [3, 1, 4, 2]
print(len(lista)) # Saída: 4


lista.sort()
print(lista) # Saída: [1, 2, 3, 4]

lista.reverse()
print(lista) # Saída: [4, 3, 2, 1]
```


MÉTODOS E OPERAÇÕES COMUNS

- `copy()`: Retorna uma cópia superficial da lista.

python

 Copy code

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


# Criando uma cópia da lista
copia = lista.copy()
print(copia) # Saída: [1, 2, 3, 4]
```

MÉTODOS E OPERAÇÕES COMUNS

- Quando você usa o método `copy()` em uma lista em Python, ele cria uma cópia superficial (shallow copy) da lista original. Isso significa que ele cria um novo objeto de lista que contém referências (ou "apontamentos") aos mesmos elementos da lista original.
- Uma cópia profunda, por outro lado, cria um novo objeto para a lista e também cria novos objetos para todos os elementos contidos dentro dela, recursivamente. Isso significa que mudanças em qualquer um dos elementos da cópia profunda não afetarão a lista original e vice-versa. Para criar uma cópia profunda em Python, você pode usar o módulo `copy` com o método `deepcopy()`.

MÉTODOS E OPERAÇÕES COMUNS

python

 Copy code

```
import copy

lista_original = [[1, 2, 3], [4, 5, 6]]
copia_profunda = copy.deepcopy(lista_original)

# Modificando um elemento na cópia profunda
copia_profunda[0][0] = 99

print("Original:", lista_original) # Saída: [[1, 2, 3], [4, 5, 6]]
print("Cópia Profunda:", copia_profunda) # Saída: [[99, 2, 3], [4, 5, 6]]
```

MÉTODOS E OPERAÇÕES COMUNS

- `count(elemento)`: Retorna o número de ocorrências de um elemento na lista.
- `index(elemento)`: Retorna o índice da primeira ocorrência do elemento especificado.

```
python Copy code  
  
lista = [1, 2, 2, 3, 3, 3, 4]  
  
# Contando o número de ocorrências de um elemento  
qt = lista.count(3)  
print(qt) # Saída: 3  
  
# Encontrando o índice da primeira ocorrência de um elemento  
pos = lista.index(3)  
print(pos) # Saída: 3
```

MÉTODOS E OPERAÇÕES COMUNS

- `sort()`: Ordena a lista em ordem crescente.
- `reverse()`: Inverte a ordem dos elementos na lista.

```
python Copy code  
  
lista = [3, 1, 4, 2]  
  
# Ordenando a lista em ordem crescente  
lista.sort()  
print(lista) # Saída: [1, 2, 3, 4]  
  
# Invertendo a ordem dos elementos na lista  
lista.reverse()  
print(lista) # Saída: [4, 3, 2, 1]
```

MATRIZES

Uma matriz é uma estrutura de dados bidimensional que pode ser vista como uma lista de listas, onde cada sublista representa uma linha. As matrizes são amplamente usadas em matemática, ciência de dados, e programação para representar tabelas, gráficos, e outros conjuntos de dados multidimensionais. `reverse()`: Inverte a ordem dos elementos na lista.

```
python Copy code  
  
# Criando uma matriz 3x3  
matriz = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```

MATRIZES

Você pode acessar os elementos de uma matriz usando índices duplos. O primeiro índice refere-se à linha, e o segundo ao elemento dentro dessa linha (ou coluna).

```
python Copy code  
  
# Acessando o elemento na primeira linha, segunda coluna  
elemento = matriz[0][1]  
print(elemento) # Saída: 2  
  
# Acessando o elemento na terceira linha, terceira coluna  
elemento = matriz[2][2]  
print(elemento) # Saída: 9
```

MATRIZES

Assim como listas, os elementos de uma matriz podem ser modificados diretamente acessando o índice desejado.

```
python Copy code  
  
# Modificando o elemento na segunda linha, segunda coluna  
matriz[1][1] = 10  
print(matriz)  
# Saída:  
# [  
#     [1, 2, 3],  
#     [4, 10, 6],  
#     [7, 8, 9]  
# ]
```


MATRIZES

Você pode usar loops aninhados para percorrer os elementos de uma matriz. Isso é útil quando você precisa realizar operações em cada elemento da matriz.

```
python Copy code  
  
# Modificando o elemento na segunda linha, segunda coluna  
matriz[1][1] = 10  
print(matriz)  
# Saída:  
# [  
#     [1, 2, 3],  
#     [4, 10, 6],  
#     [7, 8, 9]  
# ]
```

DICIONÁRIOS

Dicionários são criados utilizando chaves {} e pares de chave-valor separados por dois pontos :.

```
python                                                                    Copy code

# Dicionário vazio
dicionario_vazio = {}

# Dicionário com dados
dicionario = {
    "nome": "Alice",
    "idade": 25,
    "cidade": "São Paulo"
}
```

DICIONÁRIOS

Você pode acessar os valores armazenados em um dicionário utilizando suas chaves.

```
python Copy code  
  
# Acessando o valor da chave "nome"  
nome = dicionario["nome"]  
print(nome) # Saída: Alice  
  
# Acessando o valor da chave "idade"  
idade = dicionario["idade"]  
print(idade) # Saída: 25
```

DICIONÁRIOS

Para adicionar um novo par de chave-valor ou modificar um valor existente, basta atribuir um valor a uma chave.

```
python Copy code

# Adicionando um novo par de chave-valor
dicionario["profissão"] = "Engenheira"
print(dicionario)
# Saída: {'nome': 'Alice', 'idade': 25, 'cidade': 'São Paulo', 'profissão': 'Engenheira'}

# Modificando o valor de uma chave existente
dicionario["idade"] = 26
print(dicionario)
# Saída: {'nome': 'Alice', 'idade': 26, 'cidade': 'São Paulo', 'profissão': 'Engenheira'}
```

DICIONÁRIOS

- `del`: Remove um par de chave-valor específico.
- `pop(chave)`: Remove um item e retorna o valor associado. `popitem()`: Remove e retorna o último par de chave-valor inserido.
- `clear()`: Remove todos os itens do dicionário.

```
python Copy code

# Removendo um item usando del
del dicionario["cidade"]
print(dicionario)
# Saída: {'nome': 'Alice', 'idade': 26, 'profissão': 'Engenheira'}

# Removendo um item usando pop
profissao = dicionario.pop("profissão")
print(profissao) # Saída: Engenheira
print(dicionario) # Saída: {'nome': 'Alice', 'idade': 26}

# Removendo o último item inserido usando popitem
ultimo_item = dicionario.popitem()
print(ultimo_item) # Saída: ('idade', 26)
print(dicionario) # Saída: {'nome': 'Alice'}

# Removendo todos os itens usando clear
dicionario.clear()
print(dicionario) # Saída: {}
```



THANK YOU

Wagner Coutinho

<https://www.linkedin.com/in/wagner-coutinho-mf/>

wagner.filho@rarolabs.com.br