Algoritmos Computacionais

Conteúdo: Listas

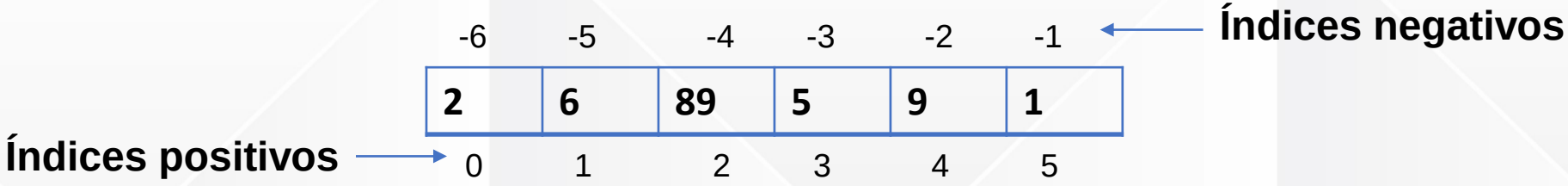
Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Listas

Listas

- As listas são estruturas de dados que permitem armazenar **múltiplos valores** em uma única **variável**. Cada valor na lista é chamado de elemento e pode ser de **qualquer tipo** de dado.
- Exemplo:

```
minha_lista = [2, 6, 89, 5, 9, 1]
```



Listas: criação e acesso

- Exemplo criação de listas

```
L = []  # Lista vazia
L = [1, 2, 3]  # Lista com elementos
L = ['a', 'b', 'c']  # Lista de strings
L = [1, 'olá', 3.14, True]  # Lista com diferentes tipos de dados
```

- Acessando elementos da lista:

```
L = [1, 2, 3, 4, 5]
primeiro_elemento = L[0]  # Acessa o primeiro elemento (índice 0)
segundo_elemento = L[1]  # Acessa o segundo elemento (índice 1)
ultimo_elemento = L[-1]  # Acessa o último elemento (índice -1)
```

Listas: modificação

- A diferença da tupla, a lista é um tipo de dado mutável, portanto podemos modificar seus elementos
- Exemplo:

```
L = [1, 2, 3, 4, 5]
L[0] = 10  # Modifica o primeiro elemento para 10
L[-1] = 50  # Modifica o último elemento para 50
```

Listas: concatenação

- A diferença da tupla, a lista é um tipo de dado mutável, portanto podemos modificar seus elementos
- Exemplo:

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_concatenada = lista1 + lista2
print(lista_concatenada)  # Output: [1, 2, 3, 4, 5, 6]
```

Comando in

- O comando in é usado para verificar se um elemento está presente em uma lista. Ele retorna um valor booleano True se o elemento estiver na lista e False caso contrário.
- Exemplo:

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

if 3 in L:
    print("3 está na lista")
else:
    print("3 não está na lista")
```

Comando del

- O comando del é usado para **remover** um **elemento** ou uma **fatia** (slice) de uma lista. Ele **altera** a **lista** original, removendo o elemento especificado.
- Remover um elemento específico:

```
L = [1, 2, 3, 4, 5]
del L[2]  # Remove o elemento de índice 2 (valor 3)
print(L)  # Saída: [1, 2, 4, 5]
```

- Remover uma fatia da lista:

```
L = [1, 2, 3, 4, 5]
del L[1:4]  # Remove os elementos de índice 1 a 3 (valores 2, 3, 4)
print(L)  # Saída: [1, 5]
```

Fatiamento

- O fatiamento de listas permite obter uma parte específica da lista com base em um intervalo de índices. O operador de slicing [:] é utilizado para realizar essa operação.
- Sintaxe básica :

```
lista[início:fim:passo]
```

- Onde:
  - **início**: o índice a partir do qual se deseja começar o fatiamento. Esse elemento será incluído na fatia resultante. Se não for fornecido, o fatiamento começará a partir do índice 0 (início da lista).
  - **fim**: o índice até o qual se deseja fazer o fatiamento. Esse elemento não será incluído na fatia resultante. Se não for fornecido, o fatiamento irá até o final da lista.
  - **passo** (**opcional**): o valor usado para determinar o passo entre os elementos selecionados na fatia. O valor padrão é 1, o que significa que todos os elementos entre início e fim serão selecionados. Se um valor de passo diferente de 1 for fornecido, o fatiamento selecionará elementos alternados ou pulados com base no valor do passo.

Fatiamento: exemplos

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Fatiamento básico
fatia = L[2:6]  # Seleciona os elementos de índice 2 a 5 (inclusive)
print(fatia)  # Output: [3, 4, 5, 6]

# Fatiamento do início até um índice específico
fatia_inicio = L[:4]  # Seleciona os elementos do início até o índice 3 (exclusive)
print(fatia_inicio)  # Output: [1, 2, 3, 4]

# Fatiamento a partir de um índice específico até o final
fatia_fim = L[6:]  # Seleciona os elementos do índice 6 até o final da lista
print(fatia_fim)  # Output: [7, 8, 9, 10]

# Fatiamento com passo
fatia_passo = L[1:8:2]  # Seleciona os elementos de índice 1 a 7 (exclusive), com passo 2
print(fatia_passo)  # Output: [2, 4, 6, 8]

# Fatiamento para obter uma cópia da lista original
copia_lista = L[:]  # Seleciona todos os elementos da lista
print(copia_lista)  # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Inserção de elementos em listas

- Existem várias formas de inserir elementos em listas em Python. As principais são:
- **O método append():** é usado para adicionar um elemento no **final** da lista. Ele **modifica** a lista original adicionando o elemento fornecido como argumento. Exemplo:

```
L = [1, 2, 3]
L.append(4)
print(L)  # Saida: [1, 2, 3, 4]
```

- **O método insert():** é usado para inserir um elemento em uma **posição específica** da lista. Ele **modifica** a lista original, **deslocando** os elementos existentes para a **direita**.
  - A sintaxe é **lista.insert(índice, elemento).**
  - Exemplo:

```
L = [1, 2, 3]
L.insert(1, 10)  # Insere o elemento 10 no índice 1
print(L)  # Saida: [1, 10, 2, 3]
```

Inserção de elementos em listas

- **Usando o operador de fatiamento [:] com atribuição:**
- Podemos usar operador de slicing [:] para adicionar elementos em uma posição específica da lista. Essa técnica cria uma **cópia** da **lista** e permite a modificação da cópia. Exemplo:

```
L = [1, 2, 3]
L = L[:1] + [10] + L[1:]
print(L)   # Saida: [1, 10, 2, 3]
```

- **O método extend():** é usado para adicionar **múltiplos elementos** no final da lista. Ele **modifica** a lista original, adicionando os elementos fornecidos como argumento. Os elementos a serem adicionados devem ser **passados** como uma **sequência**, como outra lista. Exemplo:

```
L = [1, 2, 3]
L = L[:1] + [10] + L[1:]
print(L)   # Saida : [1, 10, 2, 3]
```

Remoção de elementos em listas

- Existem várias formas de remover elementos de listas em Python. As principais formas são:

- **O método remove():** é usado para remover a **primeira ocorrência** de um elemento específico da lista. Ele modifica a lista original, removendo o elemento fornecido como argumento. Exemplo:

```
L = [1, 2, 3, 2, 4]
L.remove(2)
print(L)  # Saida: [1, 3, 2, 4]]
```

Remoção de elementos em listas

- **O comando del**:
- O comando del é usado para remover **um elemento** ou uma **fatia** (slice) de uma lista. Ele **altera** a lista original, removendo o elemento especificado. Exemplos:

  - Remover um elemento específico:

```
L = [1, 2, 3, 4, 5]
del L[2]  # Remove o elemento de índice 2 (valor 3)
print(L)  # Saida: [1, 2, 4, 5]
```

  - Remover uma fatia da lista:

```
L = [1, 2, 3, 4, 5]
del L[1:4]  # Remove os elementos de índice 1 a 3 (valores 2, 3, 4)
print(L)  # Saida: [1, 5]
```

Remoção de elementos em listas

- **O método pop():** é usado para remover e **retornar** o **último** elemento da lista ou um elemento **específico** com base no **índice** fornecido. Ele **modifica** a lista original. Se nenhum índice for fornecido, o método pop() remove e retorna o último elemento da lista. Exemplos:

  - Remover o último elemento:

```
L = [1, 2, 3, 4, 5]
elemento = L.pop()
print(elemento)  # Saida: 5
print(L)  # Saida: [1, 2, 3, 4]
```

  - Remover um elemento específico:

```
L = [1, 2, 3, 4, 5]
elemento = L.pop(2)  # Remove e retorna o elemento de índice 2 (valor 3)
print(elemento)  # Saida: 3
print(L)  # Saida: [1, 2, 4, 5]
```

Outras funções

- Além das operações de inserção e remoção, as listas em Python possuem várias outras funções úteis. A seguir são mostradas algumas delas:

- **O método len():** retorna o tamanho da lista, ou seja, o número de elementos presentes nela. Exemplo:

```
L = [1, 2, 3, 4, 5]
tamanho = len(L)
print(tamanho)   # Saida: 5
```

- **A função sorted():** retorna uma **nova lista** contendo os elementos da lista original, mas **ordenados** em ordem **crescente**. A lista **original não** é **modificada**. Exemplo:

```
L = [3, 1, 4, 2, 5]
lista_ordenada = sorted(L)
print(lista_ordenada)   # Saida: [1, 2, 3, 4, 5]
```

Outras funções

- **O método index():** retorna o **índice** da **primeira ocorrência** de um elemento específico na lista. Caso o elemento não esteja presente, um **erro** é lançado. Exemplo:

```
L = [1, 2, 3, 4, 5]
indice = L.index(3)
print(indice)   # Saida : 2
```

- **O método count():** retorna o número de **ocorrências** de um elemento específico na lista. Exemplo:

```
L = [1, 2, 3, 2, 4]
ocorrencias = L.count(2)
print(ocorrencias)   # Saida: 2
```

  - O método count() conta apenas as **ocorrências exatas** do elemento na lista. Se o elemento não estiver presente na lista, retornará 0. Além disso, o método count() **não permite** contar o número de ocorrências de um elemento em uma **faixa** específica da lista, ele considera todas as ocorrências na lista completa.

Outras funções

- **O método reverse():** inverte a ordem dos elementos na lista. A lista original é modificada. Exemplo:

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

- Existem muitas outras funções, como **min**(), **max**(), **sum**(), entre outras, que podem ajudar na realização de diferentes tarefas com listas.
  - Consulte a documentação oficial do Python para obter uma lista completa das funções e métodos disponíveis para listas.