



Engenharia de Software  
Computacional thinking with Python  
Aula 02

Prof. Dr. Francisco Elânio

# Agenda da Aula

- Estruturas de dados do tipo tupla (tuple)
- Estruturas de dados do tipo lista (list)
- Comparação tuplas vs listas
- Exercícios em sala

# Estrutura de Dados do Tipo Tupla

Uma tupla é semelhante a uma lista, mas é imutável, ou seja, seus elementos não podem ser alterados após a criação.

- Os elementos podem ser de diferentes tipos de dados.
- Os elementos são acessados por índices.
- Em Python, as tuplas são definidas entre parênteses ().

Exemplo em Python: (1, 'dois', 3.0, (4, 5)).

# Estrutura de Dados do Tipo Lista

Uma lista é uma estrutura de dados flexível em que os elementos podem ser adicionados, removidos e modificados facilmente.

- Os elementos não precisam ser do mesmo tipo de dados.
- Os elementos são acessados por índices semelhantes aos vetores.
- Em Python, as listas são definidas entre colchetes [].

Exemplo em Python: `[1, 'dois', 3.0, [4, 5]]`.

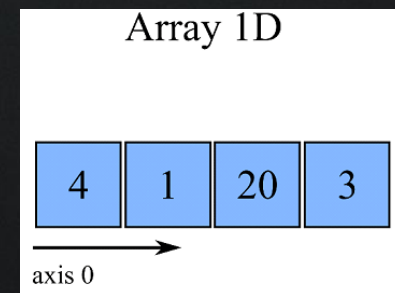


# Estrutura de Dados Unidimensional (Vetor)

Um vetor é uma estrutura de dados que armazena elementos em uma única dimensão.

- Todos os elementos são do mesmo tipo de dados.
- Cada elemento é acessado por meio de um índice inteiro.
- Os índices geralmente começam em 0 e vão até o tamanho do vetor menos um.

Exemplo em Python: `[1, 2, 3, 4, 5]`.

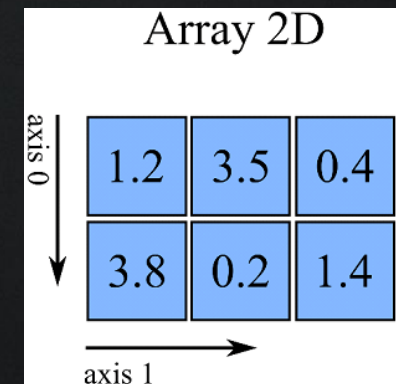


# Estrutura de Dados do Tipo Bidimensional (matriz)

Uma matriz é uma estrutura de dados bidimensional que armazena elementos em linhas e colunas.

- É uma coleção de vetores ou listas.
- Os elementos são acessados por meio de dois índices: um para a linha e outro para a coluna.
- Em Python, as matrizes são frequentemente representadas como uma lista de listas.

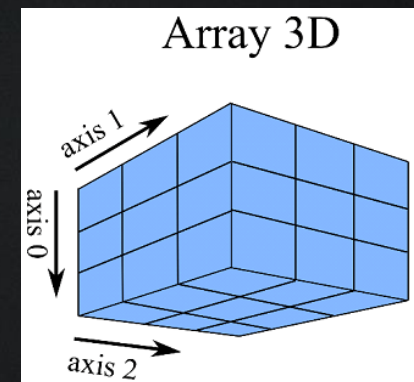
Exemplo em Python: `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.



# Estrutura de Dados do Tipo Tridimensional (matriz)

Uma estrutura de dados tridimensional é uma extensão da estrutura bidimensional, onde os elementos são organizados em três dimensões. Cada elemento é acessado através de três índices: um para a camada, um para a linha e outro para a coluna.

```
lista_tridimensional = [  
    [ # Camada 1  
      [1, 2, 3], [4, 5, 6], [7, 8, 9]],  
    [ # Camada 2  
      [10, 11, 12], [13, 14, 15], [16, 17, 18]]  
]
```



# Listas vs Tuplas

Listas	Tuplas
Lista é mutável	Tupla é imutável
Iteração da lista é mais lenta e consome mais tempo	Iteração da tupla é mais rápida
Lista consumo mais memória	Tuplas consumo menos memória
Operações de lista são mais propensas a erros.	As operações de tuplas são seguras.
Lista fornece muitos métodos integrados	Tuplas têm menos métodos embutidos
Lista é útil para operações de inserção e exclusão.	Tupla é útil para operações somente leitura, como acessar elementos.



# Estruturas de dados do tipo tupla

# Estruturas de dados com Python

## Tupla

É uma sequência imutável de comprimento fixo de objetos Python. A maneira mais fácil de criar um é com uma sequência de valores separados por vírgula:

Entrada - tupla = 4, 5, 6

Saída - (4, 5, 6)

### Índice

Entrada – tupla[0]

Saída - 4

Entrada – tupla[1]

Saída - 5

Entrada – tupla[2]

Saída - 6

Sobre a Imersão

FRONT-END

A

# Estruturas de dados com Python

## Tupla de Tupla

Quando você está definindo tuplas em expressões mais complicadas, muitas vezes é necessário colocar os valores entre parênteses, como neste exemplo de criação de uma tupla de tuplas:

### Índice de uma tupla dentro de uma outra tupla

```
tupla_tupla= (4, 5, 6), (7, 8)
```

```
(4, 5, 6), (7, 8))
```

tupla\_tupla[0]

(4,5,6)

tupla\_tupla[1]

(7,8)

# Estruturas de dados com Python

Valor específico dentro de uma dupla

Tupla 0

`tupla_tupla[0][0]`

4

`tupla_tupla[0][1]`

5

`tupla_tupla[0][2]`

6

Tupla 1

`tupla_tupla= (4, 5, 6), (7, 8)`

4, 5, 6), (7, 8))

`tupla_tupla[1][0]`

7

`tupla_tupla[1][1]`

8



# Estruturas de dados com Python

Converter qualquer sequência em uma dupla

Sequência em dupla

```
tupla([4, 0, 2])  
4,0,2
```

Caracteres não numéricos

```
tupla = tuple('string')  
( 's', 't', 'r', 'i', 'n', 'g' )
```

Índice de Caracteres não numéricos

```
tupla = tuple('string')  
tupla[0] – 's'  
tupla[1] – 't'  
tupla[2] – 'i'  
.  
.  
.  
tupla[5] – 'g'
```

( ) → Parênteses · [ ] → Colchetes · { } → Chaves

# Estruturas de dados com Python

## Exemplo da criação de uma tupla e armazenamento em cada slot

Embora os objetos armazenados em uma tupla possam ser mutáveis, uma vez que a tupla é criada não é possível modificar qual objeto é armazenado em cada slot:

```
tupla = tuple(['foo', [1, 2], True])  
('foo', [1, 2], True)  
tupla[2] = False
```

**TypeError: 'tuple' object does not support item assignment**

# Estruturas de dados com Python

## Modificando no local

Se um objeto dentro de uma tupla for mutável, como uma lista, você poderá modificá-lo no local:

```
tupla[1].append(2)  
('foo', [1, 2, 2], True)
```

```
tupla[1].append(3)  
('foo', [1, 2, 2, 3], True)
```

```
tupla[1].append(5)  
('foo', [1, 2, 2, 3, 5], True)
```

# Estruturas de dados com Python

## Modificando no local

Como modificar os dados da primeira tupla?

```
tupla = tuple(['foo', [1, 2], True])
```

```
lista = list(tupla)
```

→ Transformar em uma lista

```
lista.insert(0, 'hoje')
```

→ Usar o comando insert e inserir a posição da tupla e o nome

```
nova_tupla = tuple(lista)
```

→ Transformar em uma dupla novamente

```
('hoje', 'foo', [1, 2], True)
```



# Estruturas de dados com Python

## Modificando no local

Como modificar os dados da primeira tupla?

```
tupla = tuple(['foo', [1, 2], True])
```

```
lista = list(tupla) → Transformar em uma lista
```

```
posicao_insercao = lista.index('foo') + 1 → Usar index e inserir o ponto de  
lista.insert(posicao_insercao, 'hoje') partida + a posição desejada
```

```
nova_tupla = tuple(lista) → Transformar em uma dupla  
nova_tupla novamente
```

('hoje', 'foo', [1, 2], True)

# Estruturas de dados com Python

## Concatenando Tuplas

Embora os objetos armazenados em uma tupla possam ser mutáveis, uma vez que a tupla é criada não é possível modificar qual objeto é armazenado em cada slot:

`(4, None, 'foo') + (6, 0) + ('bar', 1)`  $\longrightarrow$  `(4, None, 'foo', 6, 0, 'bar', 1)`

`('foo', 'bar') * 4`  $\longrightarrow$  `('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar')`

`(4, 8, 12) + (3, 6, 9)`  $\longrightarrow$  `(4, 8, 12, 3, 6, 9)`

# Estruturas de dados com Python

## Multiplicando cada elemento da tupla

Embora os objetos armazenados em uma tupla possam ser mutáveis, uma vez que a tupla é criada não é possível modificar qual objeto é armazenado em cada slot:

```
tupla = (4, 8, 12)
```



```
tupla = (4, 8, 12)
```

```
for numero in tupla:  
    novo_numero = numero * 2  
    print(novo_numero)
```

```
tupla = (4, 8, 12)
```

```
nova_tupla = []
```

```
for numero in tupla:  
    novo_numero = numero * 2  
    nova_tupla.append(novo_numero)
```

```
nova_tupla = tuple(nova_tupla)
```

# Estruturas de dados com Python

## Descompactando tuplas

Se você tentar atribuir a uma expressão de variáveis semelhante a uma tupla, o Python tentará descompactar o valor no lado direito do sinal de igual:

tupla = (4, 5, 6) → (4, 5, 6)

a, b, c = tupla → a = 4  
b = 5  
c = 6

tupla = 4, 5, (6, 7) → (4, 5, (6, 7))

a, b, (c, d) = tupla → d = 7



# Estruturas de dados com Python

## Trocando nomes de variáveis

`a, b = 1, 2`

`b, a = a, b`



`a = 1`  
`b = 2`

`a = 2`  
`b = 1`

`a = 10`  
`b = 5`

`temp = a`  
`a = b`  
`b = temp`

`print("Depois da troca:")`  
`print("a =", a)`  
`print("b =", b)`

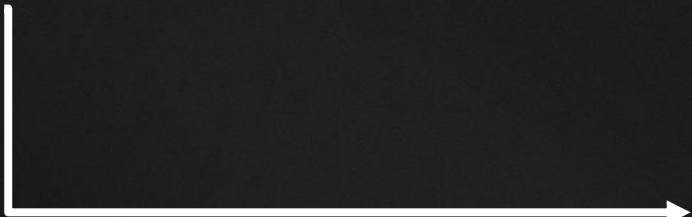
# Estruturas de dados com Python

## Descompactação de variáveis

Um uso comum de descompactação de variáveis é iterar sobre sequências de tuplas ou listas

```
seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

```
for a, b, c in seq:  
    print('a={0}, b={1}, c={2}'.format(a, b, c))
```



a=1, b=2, c=3

a=4, b=5, c=6

a=7, b=8, c=9

# Exercícios

1. Crie uma tupla chamada frutas com os seguintes elementos: 'maçã', 'banana', 'laranja', 'uva'.
2. Acesse o segundo elemento da tupla frutas.
3. Substitua o terceiro elemento da tupla frutas por 'manga' e armazene a tupla modificada em uma nova variável chamada frutas\_modificadas.
4. Concatene a tupla frutas\_modificadas com a tupla ('abacaxi', 'limão') e armazene o resultado em uma nova tupla chamada frutas\_concatenadas.
5. Verifique se o elemento 'uva' está presente na tupla frutas\_concatenadas.

# Exercícios

6. Descubra o índice do elemento 'banana' na tupla frutas\_concatenadas.
7. Conte quantas vezes o elemento 'uva' aparece na tupla frutas\_concatenadas.
8. Crie uma tupla chamada numeros com os números de 1 a 5.
9. Multiplique cada elemento da tupla numeros por 2 e armazene o resultado em uma nova tupla chamada numeros\_dobrados.
10. Crie uma tupla vazia chamada vazia e verifique seu comprimento.



# Resolução

1. Crie uma tupla chamada frutas com os seguintes elementos: 'maçã', 'banana', 'laranja', 'uva'.

**Resposta:** `frutas = ('maçã', 'banana', 'laranja', 'uva')`

2. Acesse o segundo elemento da tupla frutas.

**Resposta:** `segunda_fruta = frutas[1]`

3. Substitua o terceiro elemento da tupla frutas por 'manga' e armazene a tupla modificada em uma nova variável chamada frutas\_modificadas.

**Resposta:** `frutas_modificadas = frutas[:2] + ('manga',) + frutas[3:]`

# Resolução

4. Concatene a tupla `frutas_modificadas` com a tupla `('abacaxi', 'limão')` e armazene o resultado em uma nova tupla chamada `frutas_concatenadas`.

Resposta: `frutas_concatenadas = frutas_modificadas + ('abacaxi', 'limão')`

5. Verifique se o elemento `'uva'` está presente na tupla `frutas_concatenadas` usando estrutura de condição `if` e `else`.

Resposta:

```
if 'uva' in frutas_concatenadas:
```

```
    print("A uva está presente na tupla.")
```

```
else:
```

```
    print("A uva não está presente na tupla.")
```

# Resolução

6. Descubra o índice do elemento 'banana' na tupla frutas\_concatenadas.

Resposta: `indice_banana = frutas_concatenadas.index('banana')`

7. Conte quantas vezes o elemento 'uva' aparece na tupla frutas\_concatenadas.

Resposta: `contagem_uva = frutas_concatenadas.count('uva')`

8. Crie uma tupla chamada numeros com os números de 1 a 5.

Resposta: `numeros = (1, 2, 3, 4, 5)`

# Resolução

9. Multiplique cada elemento da tupla `numeros` por 2 e armazene o resultado em uma nova tupla chamada `numeros_dobrados`.

Resposta: `numeros_dobrados = tuple(num * 2 for num in numeros)`

10. Crie uma tupla vazia chamada `vazia` e verifique seu comprimento.



# Estruturas de dados do tipo lista

# Estruturas de dados com Python

## Métodos de Tupla

Como o tamanho e o conteúdo de uma tupla não podem ser modificados, ela é muito leve em métodos de instância. Uma particularmente útil (também disponível em listas) é a contagem, que conta o número de ocorrências de um valor.

`a = (1, 2, 2, 2, 3, 4, 2)`       $\longrightarrow$       `(1, 2, 2, 2, 3, 4, 2)`

`a.count(2)`       $\longrightarrow$       `4`

`a.count(4)`       $\longrightarrow$       `1`

# Estruturas de dados com Python

## Processamento de dados

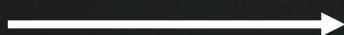
Listas e tuplas são semanticamente semelhantes (embora as tuplas não possam ser modificadas) e podem ser usadas alternadamente em muitas funções. A função de lista é frequentemente usada no processamento de dados como uma maneira de materializar uma expressão iteradora ou geradora.

`gen = range(10)`



`range(0, 10)`

`list(gen)`



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

# Estruturas de dados com Python

## Adicionando e removendo elementos

Os elementos podem ser acrescentados ao final da lista com o método `append`:

`b_list = ['agua', 'fogo', 'terra']` → `['agua', 'fogo', 'terra']`

`b_list.append('ar')` → `['agua', 'fogo', 'terra', 'ar']`

`b_list.insert(1, 'eter')` → `['agua', 'eter', 'fogo', 'terra', 'ar']`

`b_list.remove('fogo')` → `['agua', 'eter', 'terra', 'ar']`



# Estruturas de dados com Python

## Concatenando e Combinando Listas

Os elementos podem ser acrescentados ao final da lista com o método `append`:

`[4, None, 'foo'] + [7, 8, (2, 3)]`  $\longrightarrow$  `[4, None, 'foo', 7, 8, (2, 3)]`

`x = [4, None, 'foo']`  $\longrightarrow$  `[4, None, 'foo']`

`x.extend([7, 8, (2, 3)])`  $\longrightarrow$  `[4, None, 'foo', 7, 8, (2, 3)]`

# Estruturas de dados com Python

## Concatenando e Combinando Listas

Obs: a **concatenação de lista por adição é uma operação comparativamente cara**, pois uma nova lista deve ser criada e os objetos copiados. Usar `extend` para acrescentar elementos a uma lista existente, especialmente se você estiver construindo uma lista grande.

```
all = []  
for pedaco in lista_da_lista:  
    all.extend(pedaco)
```

# Estruturas de dados com Python

## Concatenando e Combinando Listas

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

```
all = []
```

```
for pedaco in lista_da_lista:  
    all.extend(pedaco)
```

```
all = []
```

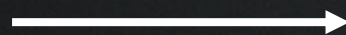
```
for pedaco in lista_da_lista:  
    all = all + pedaco
```

**Mais rápido do que concatenar usando +**

# Estruturas de dados com Python

## Concatenando e Combinando Listas

`a = [7, 2, 5, 1, 3]`



`[7, 2, 5, 1, 3]`

`a.sort()`



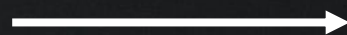
`[1, 2, 3, 5, 7]`

`b = ['saw', 'small', 'He', 'foxes', 'six']`



`['saw', 'small', 'He', 'foxes', 'six']`

`b.sort(key=len)`



`['He', 'saw', 'six', 'small', 'foxes']`

Podemos classificar uma coleção de cadeias de caracteres por seus comprimentos



# Estruturas de dados com Python

## Slice (“Fatia”)

Você pode selecionar seções da maioria dos tipos de sequência usando a notação de fatia, que em sua forma básica consiste em start:stop passado para o operador de indexação []:

<code>seq = [7, 2, 3, 7, 5, 6, 0, 1]</code>	→	<code>[7, 2, 3, 7, 5, 6, 0, 1]</code>
<code>seq[1:5]</code>	→	<code>[2, 3, 7, 5]</code>
<code>seq[3:4]</code>	→	<code>[7]</code>
<code>seq[:5]</code>	→	<code>[7, 2, 3, 7, 5]</code>
<code>seq[3:]</code>	→	<code>[7, 5, 6, 0, 1]</code>
<code>seq[-4:]</code>	→	<code>[5, 6, 0, 1]</code>

*Os índices negativos cortam a sequência em relação ao final*

# Estruturas de dados com Python

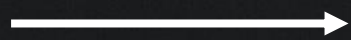
## Slice (“Fatia”)

Você pode selecionar seções da maioria dos tipos de sequência usando a notação de fatia, que em sua forma básica consiste em start:stop passado para o operador de indexação []:

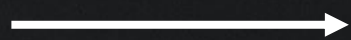
```
seq = [7, 2, 3, 7, 5, 6, 0, 1]
```

```
seq[::2]
```

```
seq[::3]
```



```
[7, 2, 3, 7, 5, 6, 0, 1]
```



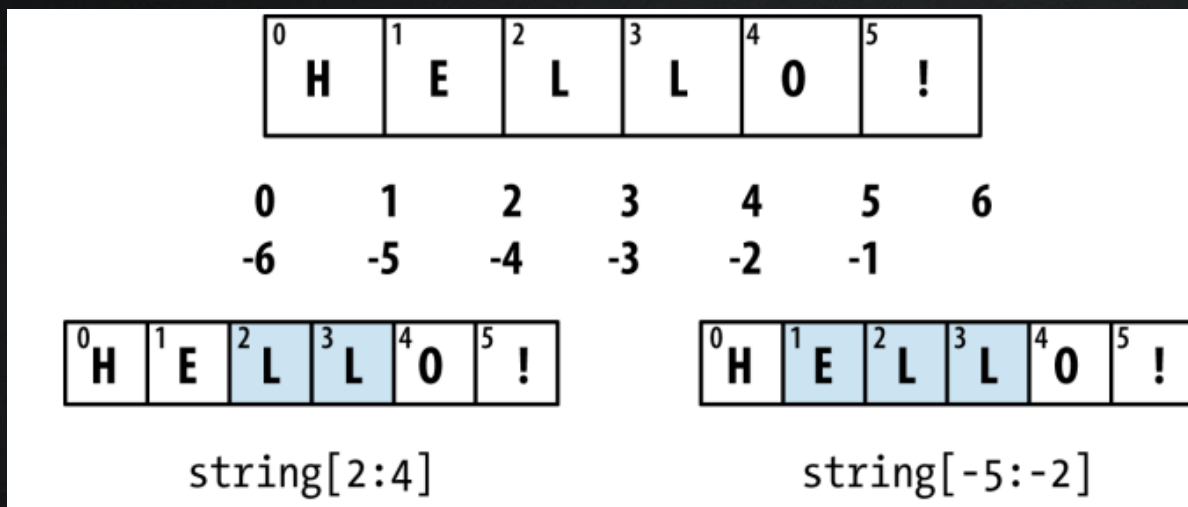
```
[7, 3, 5, 0]
```



```
[7, 7, 0]
```

# Estruturas de dados com Python

## Slice (“Fatia”)



*Fonte: Wes McKinney. Python for Data Analysis: data Wrangling with pandas, numpy, and ipython, O'Reilly.*

# Exercícios

1. Crie uma lista com os números de 1 a 10 e imprima-a.
2. Crie uma lista vazia e adicione os números pares de 1 a 10 a ela.
3. Dada uma lista de números, encontre o maior elemento.
4. Dada uma lista de palavras, crie uma nova lista com todas as palavras em maiúsculas.
5. Dada uma lista de números, calcule a soma de todos os elementos.
6. Dada uma lista de números, conte quantos números são maiores que 5.
7. Dada uma lista de strings, ordene-a em ordem alfabética.
8. Dada uma lista de números, remova os elementos repetidos.
9. Dada uma lista de strings, encontre a palavra mais longa.
10. Dada uma lista de listas, concatene todas as sublistas em uma única lista.



# Resolução

1. Crie uma lista com os números de 1 a 10 e imprima-a.

Resposta: `lista = list(range(1, 11))`

`print(lista)`

2. Crie uma lista vazia e adicione os números pares de 1 a 10 a ela.

Resposta:

`lista = []`

`for i in range(1, 11):`

`if i % 2 == 0:`

`lista.append(i)`

`print(lista)`

# Resolução

3. Dada uma lista de números, encontre o maior elemento.

Resposta: `numeros = [5, 12, 8, 3, 17, 10]`

```
maior = max(numeros)
```

```
print(maior)
```

4. Dada uma lista de palavras, crie uma nova lista com todas as palavras em maiúsculas.

```
resposta: palavras = ['python', 'é', 'incrível']
```

```
maiuculas = [palavra.upper() for palavra in palavras]
```

```
print(maiuculas)
```

# Resolução

5. Dada uma lista de números, calcule a soma de todos os elementos.

Resposta: `numeros = [2, 4, 6, 8, 10]`

```
soma = sum(numeros)
```

```
print(soma)
```

6. Dada uma lista de números, conte quantos números são maiores que 5.

Resposta:

```
numeros = [3, 7, 10, 2, 8, 5]
```

```
contagem = 0
```

```
for num in numeros:
```

```
    if num > 5:
```

```
        print(num)
```

```
        contagem += 1
```

```
print("Quantidade de números maiores que 5:", contagem)
```

# Resolução

7. Dada uma lista de strings, ordene-a em ordem alfabética.

```
palavras = ['banana', 'maçã', 'abacaxi', 'laranja']
```

```
palavras.sort()
```

```
print(palavras)
```

8. Dada uma lista de números, remova os elementos repetidos.

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

```
numeros_sem_repeticao = list(set(numeros))
```

```
print(numeros_sem_repeticao)
```



# Resolução

9. Dada uma lista de strings, encontre a palavra mais longa.

```
palavras = ['python', 'programação', 'linguagem', 'computador']
```

```
mais_longa = max(palavras, key=len)
```

```
print(mais_longa)
```

10. Dada uma lista de listas, concatene todas as sublistas em uma única lista.

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

```
lista_concatenada = []
```

```
for sublist in lista_de_listas:
```

```
    lista_concatenada.extend(sublist)
```

```
print(lista_concatenada)
```

*“O que sabemos é uma gota; o  
que ignoramos é um oceano.”  
(Issac Newton)*

# Referências

- **ASCENCIO, A. F. G, CAMPOS, E. A. V. Fundamentos da Programação de Computadores: algoritmos, Pascal, C/C++ e Java, 2ª Edição, São Paulo: Pearson 2007.**
- **FURGERI, Sérgio. Introdução à Programação em Python. São Paulo: Editora Senac, 2021.**
- **MENEZES, Nilo. Introdução à Programação em Python. São Paulo: Novatec, 2019**
- **SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madson. Algoritmos. São Paulo: Pearson, 2004.**