

## Semana 26 - Aula 1

Tópico Principal da Aula: Vetores (Listas) em Python: Criação, Inicialização e Operações

Subtítulo/Tema Específico: Introdução a Vetores e Operações Práticas

Código da aula: [SIS]ANO1C1B4S26A1ANO1C1B4S26A1.pdf]

### Objetivos da Aula:

- Compreender, de modo prático, os conceitos relacionados com **vetores** no contexto de desenvolvimento de software.
- Visualizar na prática a aplicação dos conceitos de operações com vetores utilizando a linguagem **Python**.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.

### Recursos Adicionais (Sugestão, pode ser adaptado):

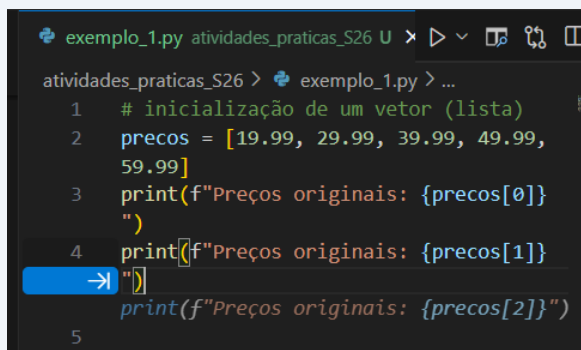
- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet;
- Ambiente de desenvolvimento Python (IDE ou Jupyter Notebook).

### Exposição do Conteúdo:

Referência do Slide: Introdução ao Conceito de Vetores e Listas em Python

- **Definição:** Em programação, um **vetor** (ou *array* unidimensional) é uma estrutura de dados fundamental que armazena uma **sequência ordenada** de elementos, geralmente do mesmo tipo, acessados por meio de um **índice**. Na linguagem Python, o tipo de dado nativo mais similar e amplamente utilizado como vetor é a **lista** (*list*).
- **Aprofundamento/Complemento (se necessário):** Enquanto em outras linguagens (como C ou Java) vetores são estruturas de tamanho fixo e homogêneas, as listas em Python são **dinâmicas** (podem crescer ou diminuir) e **heterogêneas** (podem conter tipos de dados diferentes, embora não seja uma boa prática para vetores numéricos). O uso da biblioteca **NumPy** é recomendado para trabalhar com vetores numéricos de forma otimizada.
- **Exemplo Prático:** Inicialização e acesso.

- exemplo 1



```
exemplo_1.py atividades_praticas_S26 U x ▶ v [?] [?] [?]  
atividades_praticas_S26 > exemplo_1.py > ...  
1 # inicialização de um vetor (lista)  
2 precos = [19.99, 29.99, 39.99, 49.99,  
3          59.99]  
3 print(f"Preços originais: {precos[0]}")  
4 print(f"Preços originais: {precos[1]}")  
5 print(f"Preços originais: {precos[2]}")
```

- **Vídeos Sugeridos:**

- [Vetor em Python: Aprenda a utilizar essa poderosa estrutura de dados](#)

- [Introdução aos vetores em Python e Manipulação de Listas](#)

#### Referência do Slide: Operações Comuns: Atribuição, Soma e Iteração

- **Definição:** As operações básicas incluem a **atribuição** (modificar um elemento em uma posição específica), **iteração** (percorrer todos os elementos) e operações **matemáticas** (aplicar cálculos aos elementos). Para operações matemáticas avançadas em todos os elementos simultaneamente (vetorização), utiliza-se o NumPy.
- **Aprofundamento/Complemento (se necessário):** A **iteração** é crucial para processar os dados em um vetor, usando um *loop for*. A soma de vetores em Python nativo ([1, 2] + [3, 4]) resulta na concatenação ([1, 2, 3, 4]), diferentemente da soma elemento a elemento da Álgebra Linear. Para a soma matemática de elementos, o *loop* ou a *list comprehension* é necessária.
- **Exemplo Prático:** Iteração para Soma e Criação de Novo Vetor.
- exemplo 2

```
atividades_praticas_S26 > exemplo 2.py > ...
1  # iteração soma
Click to add a breakpoint 2.0, 9.0, 6.5, 8.0]
3  soma_notas = 0
4  for nota in notas:
5      soma_notas += nota
6  media = soma_notas / len(notas)
7  print(f"Média das notas: {media:.2f}")
8
9  # multiplicar todos os elementos por 2(sem numpy)
10 dobro_notas = [nota * 2 for nota in notas]
11 print(f"Notas dobradas: {dobro_notas}")
12
```

---

## Semana 26 - Aula 2

Tópico Principal da Aula: Matrizes em Python: Representação e Manipulação Básica

Subtítulo/Tema Específico: Representação de Matrizes como Lista de Listas

Código da aula: [SIS]ANO1C1B4S26A2ANO1C1B4S26A2.pdf]

#### Objetivos da Aula:

- Compreender, de modo prático, os conceitos relacionados com **vetores** no contexto de desenvolvimento de software.
- Visualizar na prática a aplicação dos conceitos de operações com vetores utilizando a linguagem **Python**.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.

### Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet;
- Ambiente de desenvolvimento Python (IDE ou Jupyter Notebook).

### Exposição do Conteúdo:

#### Referência do Slide: Matrizes: Conceito e Representação

- **Definição:** Uma **matriz** é uma estrutura de dados bidimensional, que armazena elementos organizados em **linhas** e **colunas**. Em Python, o modo mais simples de representar uma matriz sem bibliotecas externas é utilizando **listas aninhadas** (uma *lista de listas*).
- **Aprofundamento/Complemento (se necessário):** Cada lista interna representa uma linha da matriz. Uma matriz de dimensão  $m \times n$  terá  $m$  listas externas (linhas) e cada lista interna terá  $n$  elementos (colunas). O acesso a um elemento específico é feito usando dois índices: `matriz[linha][coluna]`. O uso de NumPy com o objeto `ndarray` é o padrão para manipulação eficiente de matrizes em Ciência de Dados.
- **Exemplo Prático:** Inicialização e Acesso a Elemento Específico.
- exemplo 3

```
atividades_praticas_S26 > exemplo 3.py > ...
1  #matriz 3x3
2  matriz = [
3      [1, 2, 3],
4      [4, 5, 6],
5      [7, 8, 9]
6  ]
7  print(f"Elemento na posição [1][2]:
    {matriz[1][2]}")  # Acessa o elemento
6
```

- - **Vídeos Sugeridos:**
    - [Python: Manipulação de Listas e Matrizes \(Visão Geral\)](#)
    - [Matrizes Python: Como criar e imprimir matrizes usando NumPy \(Conceitos de Matrizes\)](#)

#### Referência do Slide: Manipulação: Seleção, Soma e Média por Linha

- **Definição:** A manipulação de matrizes envolve percorrer (iterar) os elementos. Para realizar operações como **soma** ou **média** de uma linha, é necessário um *loop* externo (para as linhas) e um *loop* interno (para as colunas/elementos da linha).
- **Aprofundamento/Complemento (se necessário):** Para somar todos os elementos de uma matriz, são necessários *loops* aninhados. Para a soma de uma linha

específica, basta acessar a linha como um vetor (`matriz[i]`) e usar funções como `sum()`.

- **Exemplo Prático:** Cálculo da Média por Linha.
- exemplo 4

```
atividades_praticas_S26 > exemplo_4.py > ...
1  matriz_notas = [
2      ...[8.5, 7.0, 9.0],...# Notas do aluno 1
3      ...[6.5, 8.0, 7.5],...# Notas do aluno 2
4      ...[9.0, 9.5, 8.0]...# Notas do aluno 3
5  ]
6
7  # Calcular a média de cada aluno
8  for i, linha in enumerate(matriz_notas):
9      ...media = sum(linha) / len(linha)
10     ...print(f"Média do aluno {i + 1}: {media:.2f}")
11
```

---

## Semana 26 - Aula 3

Tópico Principal da Aula: Slicing (Fatiamento) de Matrizes

Subtítulo/Tema Específico: Conceito e Sintaxe de Slicing

Código da aula: [SIS]ANO1C1B4S26A3ANO1C1B4S26A3.pdf]

### Objetivos da Aula:

- Compreender, de modo prático, os conceitos relacionados com **vetores** no contexto de desenvolvimento de software.
- Visualizar na prática a aplicação dos conceitos de operações com vetores utilizando a linguagem **Python**.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.

### Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet;
- Ambiente de desenvolvimento Python com **NumPy** instalado (padrão em nível técnico).

### Exposição do Conteúdo:

**Referência do Slide:** O que é Slicing e Sintaxe em 1D

- **Definição: Slicing** (fatiamento) é um método eficiente em Python para extrair uma **subsequência** de elementos de uma estrutura de dados (vetor/lista ou matriz). A **sintaxe básica em vetores (1D)** é `lista[start:stop:step]`.
- **Aprofundamento/Complemento (se necessário):**
  - `start`: Índice de início (inclusivo). Omissão implica início (índice 0).
  - `stop`: Índice de fim (exclusivo). Omissão implica fim do vetor.
  - `step`: incremento do índice. Omissão implica 1.
  - O uso de índices negativos permite contar a partir do final do vetor.

- **Exemplo Prático:** Slicing em Vetor 1D.
- exemplo 5

```
atividades_praticas_S26 > exemplo_5.py > ...
1  # \Slicing em vetor 1D
2  vetor = [10, 20, 30, 40, 50]
3  sub_vetor = vetor[1:4]
4  sub_vetor_2 = vetor[::3]
5  print(f"Sub_vetor: {sub_vetor}")
❖ 6  print(f"Sub_vetor_2: {sub_vetor_2}"]
```

- - **Vídeos Sugeridos:**
    - [NumPy Array Slicing in Python \(Foco em sintaxe e regras\)](#)
    - [Array slicing in NumPy \(Exemplo em vídeo\)](#)

## Referência do Slide: Slicing em Matrizes (2D)

- **Definição:** Em matrizes (2D), o *slicing* estende-se para dois eixos: linhas e colunas, utilizando a sintaxe `matriz[linhas_slice, colunas_slice]`. Para o uso dessa sintaxe com vírgula e dois eixos, é fundamental utilizar a biblioteca **NumPy**.
- **Aprofundamento/Complemento (se necessário):** O operador dois-pontos (`:`) sozinho é usado para selecionar **todos** os elementos em uma dimensão. Por exemplo, `matriz[1, :]` seleciona a segunda linha inteira, e `matriz[:, 2]` seleciona a terceira coluna inteira. Isso permite extrair submatrizes de forma concisa.
- **Exemplo Prático:** Extração de Submatriz e Coluna.
- exemplo 6

```
atividades_praticas_S26 > exemplo_6.py > ...
1  # requer import numpy as np como np.array
2  import numpy as np
3
4  matriz = np.array([
5      ... [1, 2, 3],
6      ... [4, 5, 6],
7      ... [7, 8, 9]
8  ])
9  sub_matriz = matriz[0:2, 1:3] # Linhas 0 a 1, colunas 1 a 2
10 print(f"Sub_matriz:\n{sub_matriz}")
11 coluna_meio = matriz[:, 1] # toda a coluna 1
❖ 12 print(f"Coluna do meio: {coluna_meio}"]
```

## Semana 26 - Aula 4

Tópico Principal da Aula: Compreensões (List Comprehensions) com Matrizes

Subtítulo/Tema Específico: List Comprehensions para Criação e Transformação de Matrizes

Código da aula: [SIS]ANO1C1B4S26A4ANO1C1B4S26A4.pdf]

### Objetivos da Aula:

- Compreender, de modo prático, os conceitos relacionados com **vetores** no contexto de desenvolvimento de software.
- Visualizar na prática a aplicação dos conceitos de operações com vetores utilizando a linguagem **Python**.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.

### Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet;
- Ambiente de desenvolvimento Python.

### Exposição do Conteúdo:

#### Referência do Slide: Conceito de List Comprehension

- **Definição:** A **List Comprehension** é uma forma concisa e eficiente (*Pythonic*) de criar listas a partir de *iterables* existentes (como outras listas, *ranges* ou tuplas). Ela substitui *loops* `for` e condicionais `if` usados para construir listas, tornando o código mais legível e rápido.
- **Aprofundamento/Complemento (se necessário):** A sintaxe é: `[expressão for item in iterável if condição (opcional)]`. É amplamente utilizada para mapeamento (transformar cada elemento) e filtragem (selecionar elementos com base em uma condição).
- **Exemplo Prático:** Filtragem e Mapeamento em Vetor (1D).
- exemplo 7

```
atividades_praticas_S26 > exemplo_7.py > ...
1  numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2
3  # Mapeamento (Dobrar apenas os números pares)
4  pares_dobro = [n * 2 for n in numeros if n % 2 == 0]
5  print(pares_dobro) # Saída: [4, 8, 12, 16]
6
7  # Versão com loop for tradicional:
8  # pares_dobro = []
9  # for n in numeros:
10 # ...if n % 2 == 0:
11 # .....pares_dobro.append(n * 2)
12
```

•

- **Vídeos Sugeridos:**
  - [Nested List Comprehensions in Python \(Criação de matrizes\)](#)
  - [List Comprehension in Python \(Visão geral e vantagens\)](#)

#### Referência do Slide: List Comprehension Aninhada para Matrizes

- **Definição:** Para trabalhar com matrizes (listas de listas), utiliza-se o conceito de **List Comprehension Aninhada**, onde o *iterable* é outra *list comprehension*. Isso é perfeito para criar matrizes, aplanar (*flatten*) ou calcular a transposta de forma compacta.
- **Aprofundamento/Complemento (se necessário):** A sintaxe de aninhamento para criar uma matriz é lida de dentro para fora, seguindo a ordem dos *loops for*. O *loop* mais à esquerda corresponde à dimensão externa (linhas), e o *loop* mais à direita, à dimensão interna (colunas).
- **Exemplo Prático:** Criação de Matriz 3x3 e Transposição.
- exemplo 8

```
atividades_praticas_S26 > exemplo_8.py > ...
1  # 1. criação de Matriz 3x3 com valores 0 a 2 em cada linha
2  matriz_A = [[j for j in range(3)] for i in range(3)]
3  print(matriz_A)  # Saída: [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
4  # 2. Aplanar (flattening) a matriz original [matriz = [[0, 1, 2], [0, 1,
5  2], [0, 1, 2]]]
6  matriz_original = [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
7  vetor_plano = [val for row in matriz_original for val in row]
8  print(vetor_plano)  # Saída: [0, 1, 2, 0, 1, 2, 0, 1, 2]
```