
1º Passo: Elaboração do Conteúdo da Aula

Semana 21 - Aula 1

Tópico Principal da Aula: Vetores e Matrizes

Subtítulo/Tema Específico: Introdução aos Vetores: Declaração e Acesso a Elementos

Código da aula: [SIS]ANO1C1B3S21A1

Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando declaração e elementos de acesso a vetores.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazes.

Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: 07 e 08 - Primeiras ideias e Ponto de partida

- **Definição:** **Na programação, um vetor é uma estrutura de dados que armazena uma coleção de elementos, de forma sequencial e ordenada. Em Python, a estrutura de dados mais comum que funciona como um vetor é a lista (list).** Imagine uma estante de livros: a estante é o vetor, e cada livro é um elemento. Para pegar um livro específico, você precisa saber sua posição (o seu "índice").
- **Aprofundamento/Complemento:** As listas em Python são extremamente flexíveis. Elas são **dinâmicas**, o que significa que podem crescer ou encolher de tamanho conforme necessário, e são **mutáveis**, permitindo que seus elementos sejam alterados após a criação. Embora seja possível armazenar diferentes tipos de dados (números, textos, etc.) na mesma lista, a boa prática de programação recomenda manter elementos de um único tipo para garantir a clareza e a consistência do código.
- **Exemplo Prático:** Gerenciar notas de alunos. Em vez de criar uma variável para cada nota, podemos armazenar todas em uma única lista.
- Python

```
# Criando um vetor (lista) com as notas de 5 alunos
notas_turma = [8.5, 9.0, 7.5, 10.0, 6.5]
print("A lista de notas é:", notas_turma)
```

Referência do Slide: 11 e 12 - Construindo o conceito: Declaração e Acesso

- **Definição:** Para declarar um vetor (lista) em Python, utilizam-se colchetes `[]`, com os elementos separados por vírgulas. acesso a um elemento é feito pelo seu
- **índice** (posição), que sempre começa em **zero**. Portanto, para acessar o primeiro elemento, usamos o índice
- `0`; para o segundo, o índice `1`, e assim por diante.
- **Aprofundamento/Complemento:** Tentar acessar um índice que não existe na lista (por exemplo, o índice 5 em uma lista de 5 elementos, cujos índices vão de 0 a 4) resultará em um erro chamado `IndexError`. É fundamental estar atento ao tamanho da lista para evitar esse tipo de erro.
- **Exemplo Prático:**
- Python

```
# Vetor de frutas
frutas = ["Maçã", "Banana", "Laranja", "Morango"]

# Acessando o primeiro elemento (índice 0)
primeira_fruta = frutas[0]
print("A primeira fruta é:", primeira_fruta) # Saída: Maçã

# Acessando o terceiro elemento (índice 2)
terceira_fruta = frutas[2]
print("A terceira fruta é:", terceira_fruta) # Saída: Laranja
```

Referência do Slide: 13 - Construindo o conceito: Acesso negativo

- **Definição:** Python oferece uma forma muito útil de acessar elementos a partir do final da lista, utilizando índices negativos. O índice
- `-1` refere-se ao último elemento, `-2` ao penúltimo, e assim sucessivamente.
- **Aprofundamento/Complemento:** O acesso negativo é especialmente útil quando você precisa do último ou dos últimos elementos de uma lista, mas não sabe o seu tamanho exato. Isso torna o código mais limpo e legível, pois evita a necessidade de calcular o índice com base no comprimento da lista (por exemplo, `lista[len(lista)-1]`).
- **Exemplo Prático:**
- Python

```
notas = [85, 92, 78, 90, 88]

# Acessando a última nota
ultima_nota = notas[-1]
print("A última nota da lista é:", ultima_nota) # Saída: 88

# Acessando a penúltima nota
penultima_nota = notas[-2]
print("A penúltima nota da lista é:", penultima_nota) # Saída: 90
```

Tópico Principal da Aula: Vetores e Matrizes

Subtítulo/Tema Específico: Manipulação de Vetores: Tamanho, Alteração e Iteração

Código da aula: [SIS]ANO1C1B3S21A2

Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando declaração e elementos de acesso a vetores.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazes.

Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: 09 - Construindo o conceito: Tamanho do vetor

- **Definição:** Para descobrir quantos elementos existem em um vetor (lista), utiliza-se a função `len()`. Ela retorna um número inteiro que representa o tamanho total do vetor.
- **Aprofundamento/Complemento:** A função `len()` é uma das mais importantes ao se trabalhar com listas. Ela é essencial para controlar laços de repetição (loops) que precisam percorrer a lista inteira e para realizar verificações, evitando o erro `IndexError` ao garantir que você não tente acessar uma posição que está fora dos limites do vetor.
- **Exemplo Prático:**
- Python

```
participantes = ["Ana", "Bia", "Carlos", "Daniel"]
numero_de_participantes = len(participantes)
print(f"Existem {numero_de_participantes} participantes na lista.") # Saída: Existem 4 participantes na lista.
```

Referência do Slide: 10 - Construindo o conceito: Alterando elementos

- **Definição:** Como os vetores (listas) em Python são mutáveis, seus elementos podem ser alterados. Para modificar um elemento, basta acessar sua posição pelo índice e atribuir um novo valor utilizando o operador
- **Aprofundamento/Complemento:** A alteração por índice é uma operação direta e eficiente. Lembre-se que ao fazer `vetor[indice] = novo_valor`, o valor antigo que estava naquela posição é permanentemente substituído pelo novo valor.
- **Exemplo Prático:** Corrigir uma nota errada em uma lista de notas.
- Python

```
notas = [85, 92, 78, 90, 88]
print("Notas originais:", notas)

# A nota do terceiro aluno (índice 2) era 78, mas a correta é 82.
notas[2] = 82
print("Notas corrigidas:", notas) # Saída: [85, 92, 82, 90, 88]
```

Referência do Slide: 11 - Construindo o conceito: Percorrendo vetores

- **Definição:** Para executar uma ação em cada elemento de um vetor, um por um, utiliza-se um laço de repetição, como o `for`. Isso permite "varrer" a lista do início ao fim, tendo acesso a cada elemento dentro do laço.
- **Aprofundamento/Complemento:** Existem duas formas comuns de percorrer uma lista com `for`. A primeira (e mais "pythônica") é o `for-each`, que dá acesso direto ao elemento: `for item in lista:`. A segunda é percorrendo os índices: `for i in range(len(lista)):`, que é útil quando, além do valor, você também precisa da posição (índice) do elemento.
- **Exemplo Prático:** Imprimir cada nome de uma lista de convidados.
- Python

```
convidados = ["Alice", "Bruno", "Carla"]

print("Lista de Convidados:")
# Usando o laço 'for' para percorrer a lista
for nome in convidados:
    print(f"- {nome}")
```

Semana 21 - Aula 3

Tópico Principal da Aula: Vetores e Matrizes

Subtítulo/Tema Específico: Operações Avançadas em Vetores: Fatiamento e Adição

Código da aula: [SIS]ANO1C1B3S21A3

Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando declaração e elementos de acesso a vetores. [24]
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento. [25]
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazes. [26]

Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: 09 - Construindo o conceito: Fatiamento de vetores

- **Definição:** O fatiamento (slicing) é uma técnica poderosa para extrair uma sub-lista (uma "fatia") de um vetor. A sintaxe é `vetor[inicio:fim]`. O elemento no índice `inicio` é incluído, mas o elemento no índice `fim` é **excluído**.
- **Aprofundamento/Complemento:** A sintaxe completa do fatiamento é `vetor[inicio:fim:passo]`. Se o `inicio` for omitido, a fatia começa do índice 0. Se o `fim` for omitido, a fatia vai até o final da lista. O `passo` (step) permite pular elementos. Por exemplo, `[::2]` pega todos os elementos, pulando de 2 em 2.
- **Exemplo Prático:**
- Python

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

# Pega os elementos do índice 2 até o 4 (o 5 não entra)
fatia1 = numeros[2:5]
print("Fatia de 2 a 5:", fatia1) # Saída: [2, 3, 4]

# Pega os 3 primeiros elementos (do início até o índice 2)
fatia2 = numeros[:3]
print("Três primeiros:", fatia2) # Saída: [0, 1, 2]

# Pega os elementos do índice 6 até o final
fatia3 = numeros[6:]
print("Do índice 6 ao fim:", fatia3) # Saída: [6, 7, 8, 9]
```

Referência do Slide: 10 - Construindo o conceito: Adição de elementos

- **Definição:** Para adicionar um novo elemento **ao final** de um vetor (lista) existente, utiliza-se o método `.append()`. 30
- **Aprofundamento/Complemento:** O método `.append()` é a forma mais comum e eficiente de adicionar um único item ao final de uma lista. Ele modifica a lista original, aumentando seu tamanho em uma unidade. Para juntar duas listas, pode-se usar o operador `+` ou o método `.extend()`.
- **Exemplo Prático:** Adicionar uma nova tarefa a uma lista de afazeres.
- Python

```
tarefas = ["Lavar a louça", "Estudar Python"]
print("Tarefas iniciais:", tarefas)

# Adicionando uma nova tarefa no final da lista
tarefas.append("Ir à academia")
print("Tarefas atualizadas:", tarefas) # Saída: ['Lavar a louça', 'Estudar Python', 'Ir à academia']
```

Referência do Slide: 11 - Construindo o conceito: Inserção de elementos

- **Definição:** Se for necessário adicionar um elemento em uma **posição específica** do vetor, e não no final, usa-se o método `.insert()`. Sua sintaxe é `vetor.insert(indice, elemento)`.
- **Aprofundamento/Complemento:** Ao usar `.insert()`, todos os elementos a partir do índice especificado são "empurrados" uma posição para a direita para dar espaço ao novo elemento. Por essa razão, a inserção no meio ou no início de uma lista pode ser uma operação mais lenta do que o `.append()` em listas muito grandes.
- **Exemplo Prático:** Inserir um novo aluno em uma fila de chamada, na posição correta.
- Python

```
fila = ["João", "Maria", "Pedro"]
print("Fila original:", fila)

# Inserindo 'Ana' na primeira posição (índice 0)
fila.insert(0, "Ana")
print("Fila com Ana no início:", fila) # Saída: ['Ana', 'João', 'Maria', 'Pedro']

# Inserindo 'Carlos' na terceira posição (índice 2)
fila.insert(2, "Carlos")
print("Fila com Carlos no meio:", fila) # Saída: ['Ana', 'João', 'Carlos', 'Maria', 'Pedro']
```

Semana 21 - Aula 4

Tópico Principal da Aula: Vetores e Matrizes

Subtítulo/Tema Específico: Tópicos Especiais: Remoção, List Comprehension e Matrizes

Código da aula: [SIS]ANO1C1B3S21A4

Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando declaração e elementos de acesso a vetores.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazes.

Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: 09 - Construindo o conceito: Remoção de elementos

- **Definição:** Existem duas formas principais de remover elementos. Para remover um elemento **pelo seu valor**, usa-se o método `.remove(valor)`. Para remover um elemento
- **pela sua posição (índice)**, usa-se o comando `del vetor[indice]`.
- **Aprofundamento/Complemento:** Se houver valores duplicados na lista, `.remove()` irá apagar apenas a primeira ocorrência do valor. Outro método útil é o `.pop(indice)`, que remove o elemento de um índice e o retorna, permitindo que você o use em outra variável. Se `.pop()` for chamado sem um índice, ele remove e retorna o último elemento da lista.
- **Exemplo Prático:**
- Python

```
compras = ["Arroz", "Feijão", "Macarrão", "Batata"]
print("Lista de compras:", compras)

# Removendo o item 'Macarrão' pelo valor
compras.remove("Macarrão")
print("Após remover 'Macarrão':", compras) # Saída: ['Arroz', 'Feijão', 'Batata']

# Removendo o primeiro item ('Arroz') pelo índice 0
del compras[0]
print("Após remover o item no índice 0:", compras) # Saída: ['Feijão', 'Batata']
```

Referência do Slide: 10 - Construindo o conceito: List comprehensions

- **Definição:** "List Comprehension" é uma sintaxe especial do Python para criar uma nova lista de forma concisa e elegante, baseando-se em uma lista (ou outra sequência) existente. Ela permite aplicar uma expressão e/ou uma condição a cada item da sequência original.
- **Aprofundamento/Complemento:** A sintaxe básica é `[expressao for item in iteravel]`. É possível adicionar uma condição de filtro: `[expressao for item in iteravel if condicao]`. List comprehensions são frequentemente mais rápidas e sempre mais legíveis do que usar um laço `for` com `.append()` para criar uma lista.
- **Exemplo Prático:** Criar uma lista com o quadrado dos números de 0 a 9.
- Python

```
# Forma tradicional com loop
quadrados_loop = []
for x in range(10):
    quadrados_loop.append(x**2)
print("Com loop:", quadrados_loop)

# Forma concisa com List Comprehension
quadrados_lc = [x**2 for x in range(10)]
print("Com List Comprehension:", quadrados_lc) # Saída: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Referência do Slide: 11 - Construindo o conceito: Matrizes (listas de listas)

- **Definição:** Uma matriz (ou tabela) pode ser representada em Python como uma **lista de listas**. Cada lista interna representa uma linha da matriz. Para acessar um elemento, são necessários dois índices:
- matriz[linha][coluna].
- **Aprofundamento/Complemento:** Assim como nos vetores, a contagem de linhas e colunas também começa em zero. Portanto, matriz[0][0] é o elemento na primeira linha e primeira coluna. Essa estrutura é a base para manipulação de dados tabulares, jogos (como tabuleiro de xadrez ou campo minado) e muitas aplicações em computação gráfica e ciência de dados.
- **Exemplo Prático:** Representar um tabuleiro de jogo da velha 3x3.
- Python

```
# ' ' representa um espaço vazio
jogo_da_velha = [
    ['X', 'O', 'X'],  # Linha 0
    ['O', 'X', 'O'],  # Linha 1
    ['O', ' ', 'X']   # Linha 2
]

# Acessando o elemento na segunda linha (índice 1), terceira coluna (índice 2)
elemento = jogo_da_velha[1][2]
print("Elemento na linha 1, coluna 2 é:", elemento) # Saída: O

# Alterando o espaço vazio para 'O'
jogo_da_velha[2][1] = 'O'
# Imprimindo a linha 2 para ver a alteração
print("Linha 2 atualizada:", jogo_da_velha[2]) # Saída: ['O', 'O', 'X']
```
