



UFOP

Universidade Federal de Ouro Preto - UFOP
Departamento de Computação - DECOM
Programação de Computadores I – BCC701
www.decom.ufop.br/bcc701



Aula Teórica 09

Material Didático Proposto

Conteúdos da Aula

- **Vetores ou Listas no Python**
- **Algumas Funções Aplicadas a Vetores**
- **Exercícios**

No Python não existe vetor formalmente
Vetores para nós são Listas no Python
Lista = Vetor
Vetor = Lista

Problema

Suponha que você deseje **armazenar** a nota da primeira prova de programação, de cada um dos 40 alunos de BCC701.

Seu programa teria o seguinte código para leitura de dados:

```
Nota_01 = float(input("DIGITE A NOTA 1: "))  
Nota_02 = float(input("DIGITE A NOTA 2: "))  
Nota_03 = float(input("DIGITE A NOTA 3: "))  
Nota_04 = float(input("DIGITE A NOTA 4: "))  
Nota_05 = float(input("DIGITE A NOTA 5: "))  
Nota_06 = float(input("DIGITE A NOTA 6: "))  
Nota_07 = float(input("DIGITE A NOTA 7: "))  
• • •  
Nota_40 = float(input("DIGITE A NOTA 40: "))
```

Problema

Para a impressão de dados:

```
print(f"NOTA DO ALUNO 1: {Nota_01})  
print(f"NOTA DO ALUNO 2: {Nota_02})  
print(f"NOTA DO ALUNO 3: {Nota_03})  
print(f"NOTA DO ALUNO 4: {Nota_04})  
print(f"NOTA DO ALUNO 5: {Nota_05})  
print(f"NOTA DO ALUNO 6: {Nota_06})  
print(f"NOTA DO ALUNO 7: {Nota_07})
```

• • •

```
print(f"NOTA DO ALUNO 40: {Nota_40})
```

Com esta codificação:

- **É necessário um nome diferente de variável para cada nota de aluno.**
- **Se o programa manipulasse 1.000 alunos, seriam necessárias 1.000 variáveis com nomes diferentes e seria “impossível” manipular todas estas variáveis.**

Solução:

- Armazenar todos os dados referentes às notas dos alunos em uma estrutura com um único nome e várias posições
- Assim, todos os dados teriam um único nome de variável (o nome do conjunto) na memória.
- Identificamos cada elemento do conjunto, associando ao nome da variável um índice (por exemplo de 1 a 40).
- Você já faz este tipo de associação em Geometria. Se você tiver $\vec{v} = (v_1, v_2, v_3)$ e $\vec{w} = (w_1, w_2, w_3)$, podemos fazer $\vec{s} = \vec{v} + \vec{w}$

Em programação, este conjunto de dados recebe o nome de VETOR, e no nosso exemplo teríamos um vetor com o nome NOTAS ilustrado por:

notas dos alunos (conteúdo na posição)

NOTAS	9.8	7.6	2.5	4.9	...	9.6
	0	1	2	3	...	39

índices do vetor (posição)

Assim na posição 0 de NOTAS temos o valor 9.8, na posição 1, o valor 7.6 e assim sucessivamente.

Estrutura de Dados Vetor

- Um vetor tem um nome, como uma variável comum, para armazenar os dados na memória.
- Embora no Python possamos armazenar dados de diferentes tipos, teremos sempre o mesmo tipo de dado em um vetor.
- Os *índices* são inteiros iniciando em 0 e são as posições do vetor.
- Os elementos (conteúdos) de um vetor são unicamente identificados pelos seus respectivos índices (posições) .

No vetor NOTAS do exemplo temos:

NOTAS[0] contém o valor 9.8

NOTAS[2] contém o valor 7.6

NOTAS[3] contém o valor 2.5 e assim sucessivamente

Estrutura de Dados Vetor

- Um vetor é uma estrutura onde os dados estão armazenados em posições consecutivas e ordenadas.
- A ordem é determinada pelos índices inteiros: 0, 1, 2, 3, ... e na sequência em que foram “imputados”
- Um vetor representa um conjunto ordenado, pelo índice, de valores homogêneos (do mesmo tipo).
- Por esta razão um vetor é denominado uma Estrutura de Dados Homogênea.

- Na Entrada dos dados de um vetor, os elementos são separados por vírgula

```
>>> massa = [20.6, 45, 13.8] (Inicialização por atribuição)
```

```
>>> massa
```

```
[20.6, 45, 13.8]
```

```
>>> peso = [23, 44, 78] (atribuição)
```

```
>>> peso
```

```
[23, 44, 78]
```

```
>>> massa[0]
```

```
20.6
```

```
>>> peso[2]
```

```
78
```

```
>>> vet = [] (inicialização lista vazia)
```

```
>>> vet.append(10) (acrescenta valor 10 ao final da lista)
```

```
>>> vet
```

```
[10]
```

```
>>> vet.append(20) (acrescenta valor 20 ao final da lista)
```

```
>>> vet
```

```
[10, 20]
```

```
>>> massa = [20.60, 45, 13.8] (atribuição)
```

```
>>> massa[0] = 0 (atribuição)
```

```
>>> massa
```

```
[0.0, 45, 13.8]
```

```
>>> massa[2] = massa[2] + massa[3] (operação)
```

```
>>> massa (vetor modificado)
```

```
[0.0, 58.8, 13.8]
```

```
V = [110, 220, 330, 440, 550] # entrada por atribuição

n = len(V) # tamanho de V, ou seja, V tem 5 elementos
print("IMPRESSÃO DOS ELEMENTOS DO VETOR")
for i in range(n):
    print(f"V[{i}] = {V[i]}")

print("Agora a função print(V)")
print(V)
```

Vetor - Definição

Criando vetores:

```
1. # Vetor de valores numéricos
2. num_vector = [ 2, 4.3, 6, 8.2, 10.8 ]
3.
4. # Vetor de booleanos
5. bool_vector = [ True, False, True, True ]
6.
7. # Vetor de strings
8. words_vector = [ 'alpha', 'bravo', 'celta' ]
9.
10. # Vetor vazio
11. empty_vector = [ ]
```

No Python não existe vetor formalmente
Vetores para nós são Listas no Python
Lista = Vetor
Vetor = Lista

Vetor: Operações

Podemos realizar várias operações com vetores. Listamos a seguir as principais operações exploradas em nossa disciplina:

1. Acessar e modificar elementos
2. Inserir um elemento no final do vetor
3. Remover um elemento do final do vetor
4. Descobrir a dimensão (quantidade de elementos)

Para conhecer todas as funções associadas a um vetor/lista basta dar um `dir(list)` no prompt do Python

```
>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__',
 '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
 '__setattr__', '__setitem__', '__sizeof__', '__str__',
 '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```


Vetor: Operações

Acessar e modificar elementos

```
1. V = [ 1, 2, 3, 4, 5 ]
2. print(f'V no endereço 2: {V[2]}')
3. # O programa vai imprimir:
4. # V no endereço 2: 3
5. V[0] = V[0] + 1
6. # O vetor V passa conter: [ 2, 2, 3, 4, 5 ]
```

Inserir elementos em um vetor (lista)

```
1. V = []          # cria um vetor vazio
2. # Inserindo o número 1 no vetor
3. V.append(1)
```

Remover elementos do vetor

```
1. V = [ 10, 20, 30, 40, 50 ] # cria um vetor com dados

2. x = V.pop(1)    # deleta o elemento na posição 1
                   # e retorna o valor em x
3. # O vetor V passa conter: [ 10, 30, 40, 50 ]

4. print(x) # imprime o valor 20
```

Descobrir a dimensão (quantidade de elementos)

```
1. V = [ 1, 2, 3, 4, 5 ]
2. tam = len(V)
3. # tam = 5
4. for i in range(len(V)): #poderia ser range(tam)
5.     print(f"Elemento na posição {i} = {V[i]}")
6. print("Fim do programa")
```

Vetor: leitura de um vetor posição a posição

```
1. N = int(input('Quantidade de valores: '))

2. vet = [ ]

3. for ind in range(N):

5.     valor = float(input('Número {ind}: '))

6.     vet.append(valor)

8. for ind in range(N): # poderia ser range(len(vet))

10.    print(f'Vetor[{ind}] = {vet[ind]:6.2f}')

11.print("Fim do programa")
```

Comando eval () – avalia e transforma o conteúdo

Leitura de um vetor “em lote” e somando os seus elementos

```
1. vet = eval(input('Digite o vetor: ')) # avalia e transforma
    # digitar no formato [3.6, -5.8, 5, 10] pode ser inteiro ou
    # float
2. n = len(vet)
3. soma = 0
4. for i in range(n):
5.     soma += vet[i]
6. print(f'A soma dos {n} elementos do vetor é: {soma}')
7. print("Fim do programa")
```

Comando eval ()

Ler “em lote” 2 vetores: i) nome do produto e ii) o respectivo preço unitário.
Verificar se os vetores têm o mesmo tamanho.

Imprimir o nome do produto seguido do preço unitário

```
1. nome_prod = eval(input('Nome dos produtos: ')) # literal
2. preco_prod = eval(input('Preço unitários: ') # float
3. if len(nome_prod) == len(preco_prod): # vetores válidos
4.     print('Beleza, vetores com mesmo tamanho')
5.     for p in range(len(nome_prod)):
6.         print(f'{nome_prod[p]} -> R${preco_prod[i]:.2f}')
7. else:
8.     print('Caraca véi, vetores com tam. diferentes')
9. print("Fim do programa")
```

Exemplo 1.

Calcular a média de um conjunto de valores pré-definidos pelo usuário e depois imprimir os valores que estiverem acima da média

```
1. N = int(input('Quantidade de valores: '))
2. vet = [ ]
3. media = 0
4. for ind in range(N):
5.     valor = int(input('Informe um valor inteiro: '))
6.     media = media + valor
7.     vet.append(valor)
8. media = media / N
9. for ind in range(N): # poderia ser range(len(vet))
10.    if vet[ind] > media:
11.        print(f'Valor acima da média: {vet[ind]}')
12.print("Fim do programa")
```

Exercícios

Exercícios propostos

1. Faça um programa que preencha um vetor de n elementos através de entradas do usuário **posição a posição**. Após a leitura do vetor, calcule a média dos valores e mostre o resultado.
2. Faça um programa que preencha dois vetores com 5 números digitados pelo usuário. Posteriormente, construa um terceiro vetor dado pela soma de cada elemento dos vetores de entrada. Imprima o conteúdo do vetor calculado.
 - Exemplo: $v1 = [1, 3, 2]$, $v2 = [5, 2, 6]$, resultado = $[6, 5, 8]$
3. Faça um programa que preencha dois vetores de 10 elementos através de entradas do usuário. Após a definição dos vetores, construa um terceiro vetor (20 elementos) onde os elementos de *índice ímpar* recebem os valores do *primeiro vetor* e os elementos de *índice par* recebem os valores do *segundo vetor*. Imprima o conteúdo do vetor calculado.
 - Exemplo: $v1 = [1, 3, 2]$, $v2 = [5, -1, 6]$, resultado = $[1, 5, 3, -1, 2, 6]$

Exercícios propostos

4. Escreva um programa que preencha um vetor com entradas do usuário. Considere que o usuário definirá apenas valores numéricos positivos, e que, ao desejar encerrar a definição dos elementos ele digite um valor negativo. Após a entrada de todos os elementos do vetor, calcule e imprima o seu somatório.
5. Escreva um programa que leia um vetor, e preencha 2 vetores, o primeiro recebe os valores pares e o segundo os valores ímpares do vetor lido.

Exemplo: Ler $V = [2, 5, 7, 85, 34, 25]$ e gerar

$V1 = [2, 34]$ e

$V2 = [5, 7, 85, 25]$