Variáveis Indexadas:

- Vetor
- Lista

Professor Edson de Oliveira

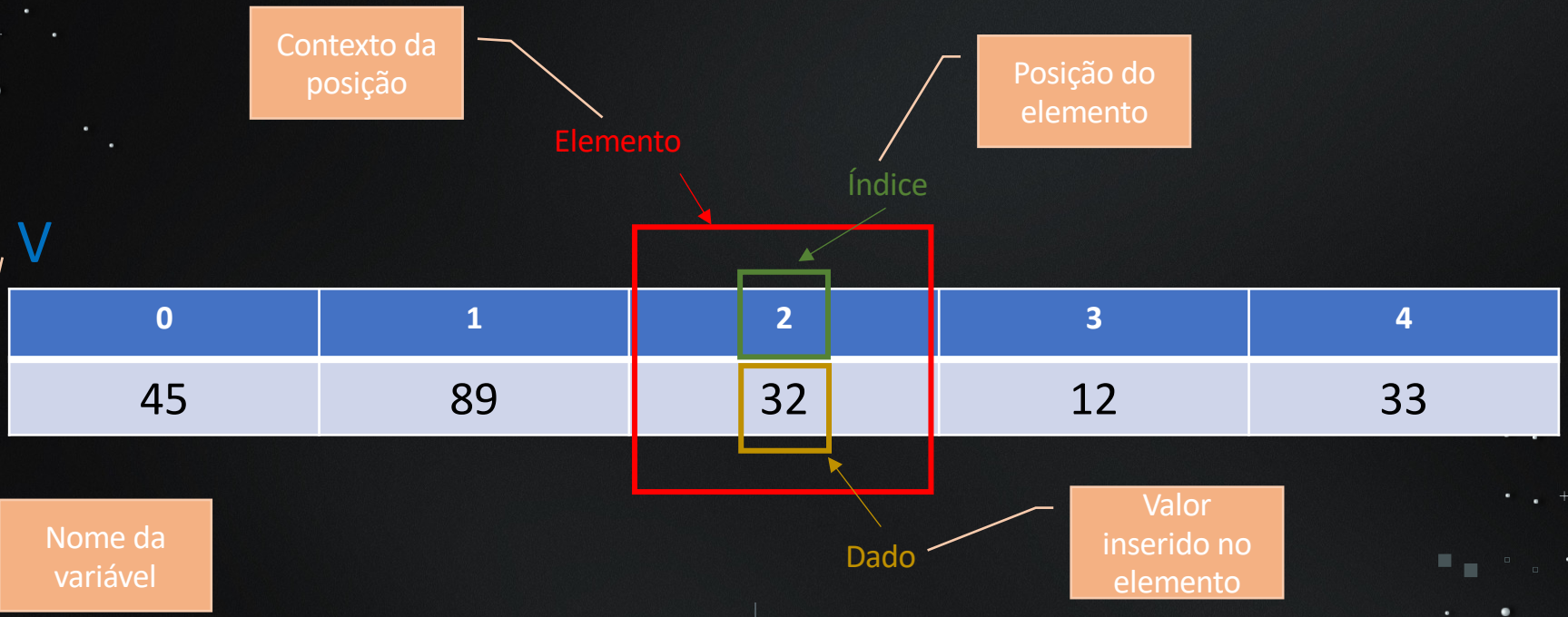
VARIÁVEIS INDEXADAS

Variáveis indexadas

- É uma classe de variável que permite armazenar **diversas informações** em apenas **um** identificador (variável).

- Elas são referenciadas pelo **índice** que começa do **zero** e vai até o limite da estrutura.

Vetor



- Os dados nele inseridos são homogêneos (mesmo tipo)
- O seu tamanho é pré-definido

Vetor – Aplicação Prática

V

0	1	2	3	4
45	89	32	12	33

Código-Fonte:

```python
# Insere os elemento no vetor diretamente
#    0   1   2   3   4   <= ÍNDICES
v = [45, 89, 32, 12, 33]
print(f"Exibe a posição 2 do vetor: v[2] = {v[2]}")
print(f"Exibe o vetor inteiro: V = {v}")
# Mudando a primeira posição para 10
v[0] = 10
print(f"V = {v}")
```

Saída:

```
Exibe a posição 2 do vetor: v[2] = 32
Exibe o vetor inteiro: V = [45, 89, 32, 12, 33]
V = [10, 89, 32, 12, 33]
```

Vetor – Lista de exercícios

Nível 1

v

0	1	2	3	4
45	-89	32	-12	33

EXERCÍCIOS:

Considere o vetor de nome ‘v’ acima e faça os seguintes subalgoritmos:

1. Fazer uma Função que retorne o primeiro elemento do vetor.

```
x = primeiro_elemento(v)
>> x valerá 45
```

2. Fazer um procedimento que exiba somente os números negativos contidos no vetor.

```
exibe_vegativos(v)
>> -89 -12
```

3. Fazer uma função que retorne a soma dos elementos do vetor

```
x = soma_elementos (v)
>> x valerá 9
```

4. Fazer uma função que retorne a media dos elementos do vetos

```
x = media_elementos(v)
>> x valerá 1.8
```

5. Fazer um procedimento que exiba na tela os numeros ímpares contidos no vetor

```
elementos_impares(v)
>> 45 -89 33
```

Vetor – Lista de exercícios

Nível 1

v

0	1	2	3	4
45	-89	32	-12	33

EXERCÍCIOS:

Considere o vetor de nome ‘v’ acima e faça os seguintes subalgoritmos:

6. fazer um procedimento que exiba na tela o primeiro e o ultimo elemento do vetor.

```
exibe_extremos(v)
>> 45 33
```

7. Fazer um procedimento que exiba os elementos cujos índices sejam pares.

```
exibe_indice_impar(v)
>> -89 -12
```

8. Fazer uma função que retorne True caso um valor passado por parâmetro exista no vetor, senão False.

```
x = busca(v, 32)
>> s valerá True porque existe o elemento 32 no vetor
```

9. Fazer um procedimento que ordene os elementos do vetor.

```
ordena(v)
>> -89 -12 32 33 45
```

Vetor – Lista de exercícios

Nível 2

Considere o valor dos vetores abaixo para fazer os próximos exercícios:

```
v1 = [41, 72, 39, 4, 35]
v2 = [0, 0, 0, 0, 0]
```

10. Fazer um procedimento chamado 'copia_vetor(v1, v2)' que copie os elementos do vetor v1 em v2.

```
copia(v1, v2)
>> 45 33
```

11. Fazer um procedimento chamado 'inverte_vetor(v1, v2)' que copie os elementos invertidos do vetor v1 em v2, ou seja, o primeiro elemento de v1 será o último de v2.

```
inverte_vetor(v1, v2)
>> [41, 72, 39, 4, 35]
>> [35, 4, 39, 72, 41]
```

12. Fazer um procedimento chamado 'ordena_vetor_crescente(v)' que ordene de forma crescente o vetor passado por parâmetro.

```
ordena_vetor_crescente(v1)
>> [4, 35, 39, 41, 72]
```

13. Fazer um procedimento chamado 'ordena_vetor_decrescente(v)' que ordene de forma decrescente o vetor passado por parâmetro.

```
ordena_vetor_decrescente(v1)
>> [72, 41, 39, 35, 4]
```

Vetor – Lista de exercícios

Nível 3

Considere o valor dos vetores abaixo para fazer os próximos exercícios:

```
v1 = [41, 72, 39, 4, 35]
v2 = [0, 0, 0, 0, 0]
```

14. Fazer um procedimento chamado 'ordena_vetor(v, forma)' que baseado na forma ('c' para crescente ou 'd' para decrescente) ordene na ordem solicitada.

```
ordena_vetor(v1, 'c')
>> [4, 35, 39, 41, 72]
```

15. Fazer um procedimento chamado 'separa_pares_impares(v)' que coloque nas posições mais a esquerda os valores pares e mais a diretia os impares.

```
separa_pares_impares(v1)
>> [72, 4, 41, 39, 35]
```

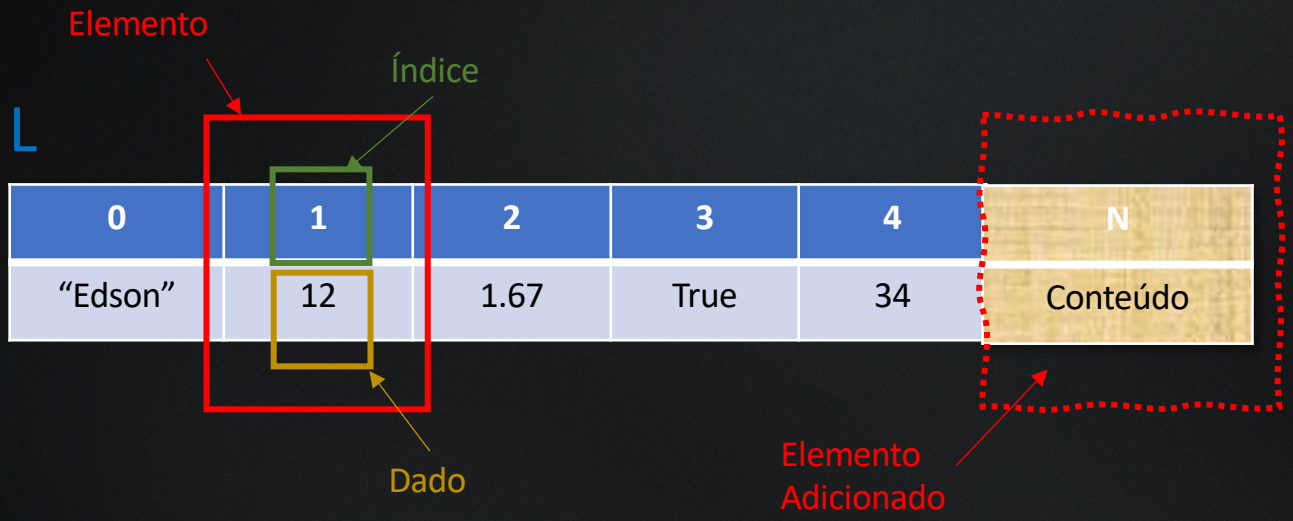
16. Fazer uma função chamada 'conta_acima_media(v) que retorne quantos elementos do vetor estão acima da média.

```
x = conta_acima_media(v1)
>> x valerá 3
```

17. Fazer uma função chamada 'maior_elemento(v)' que retorne o elemento de maior valor do vetor.

```
x = maior_elemento(v1)
>> x valerá 72
```

Lista



- Os dados nela inseridos podem ser heterogêneos (tipos diferentes)
- O seu tamanho é dinâmico (Pode mudar no decorrer do programa)

Lista – Aplicação Prática

0	1	2	3	4
Edson	12	1.67	True	34

Código-Fonte:

```python
# Insere os elemento na lista diretamente
#       0      1   2     3    4   <= ÍNDICES
l = ["Edson", 12, 1.67, True, 34]
print(f"Exibe a posição 1 da lista: L[1] = {l[1]}")
print(f"Exibe a lista inteira: L = {l}")
# Mudando a primeira posição para 10
l[4] = "Edson"
print(f"L = {l}")
```

Saída:

```
Exibe a posição 1 da lista: l[1] = 12
Exibe a lista inteira: L = ['Edson', 12, 1.67, True, 34]
V = ['Edson', 12, 1.67, True, 'Edson']
```

Métodos para Listas

list() ou []

Cria uma lista vazia.

```
lista1 = list()
print(lista1)
>> []
lista2 = []
print(lista2)
>> []
```

Criou a lista1 e lista2 como vazias.

Os textos em amarelo dentro do quadro são os resultados dos comandos na tela.

Métodos para Listas

append()

Adiciona um dado no final da lista.

```
lista = list()
lista.append(22)
lista.append("Lógica")
print(lista)
>> [22, 'Lógica']
```

Adicionou os elementos 22 e “lógica”.

Métodos para Listas

```
insert(posição, elemento)
```

Insere um dado numa posição da lista.

```
lista = [22, "Lógica"]
>> [22, 'Lógica']
lista.insert(1, 57.7)
>> [22, 57.7, 'Lógica']
```

Inseriu o elemento 57.7 na posição cujo índice é 1.

Métodos para Listas

pop(posição)

Remove o elemento com a posição informada. Se não for informada a posição, remove o ultimo elemento..

```
lista = [22, 57.7, "Lógica"]
lista.pop(0)
>> [57.7, 'Lógica']
lista.pop()
>> [57.7]
```

Remove o elemento com o índice 0 (zero) depois remove o último elemento.

Métodos para Listas

remove(elemento)

Remove o elemento pelo conteúdo.

```
lista = [22, 57.7, "Lógica"]
lista.remove(57.7)
>> [22, “Lógica”]
```

Remove o elemento com o conteúdo 57.7 da lista.

Métodos para Listas

index(elemento)

Retorna o índice do elemento passado por parâmetro.

```
lista = [22, 57.7, "Lógica"]
indice = lista.index("Lógica")
print(f"Índice = {indice}")
>> Índice = 2
```

Retorna o índice 2 do elemento “Lógica” passado por parâmetro.
Caso o elemento não exista, haverá um erro de compilação.

Métodos para Listas

count(elemento)

Conta quantos elementos específicos existem na lista.

```
lista = [22, 22, 57.7, "Lógica"]
qtd = lista.count(22)
print(f"Quantidade = {qtd}")
>> Quantidade = 2
```

Retorna 2 porque existem 2 elementos 22 na lista.

Métodos para Listas

len(lista)

Conta quantos elementos existem na lista.

```
lista = [22, 22, 57.7, "Lógica"]
qtd_elementos = len(lista)
print(f"Quantidade = {qtd_elementos}")
>> Quantidade = 4
```

Retorna 2 porque existem 2 elementos 22 na lista.

Métodos para Listas

```
sum(lista)
```

Soma os elementos da lista caso todos sejam numéricos.

```
lista = [19, 4, 25, 33, -5]
print(sum(lista))
>> 76
```

Soma os elementos da lista

Métodos para Listas

+

Concatena (junta) listas.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista3 = lista1 + lista2
print(f"Lista1 = {lista1}")
>> Lista1 = [1, 2, 3]
print(f"Lista2 = {lista2}")
>> Lista2 = [4, 5, 6]
print(f"Lista3 = {lista3}")
>> Lista3 = [1, 2, 3, 4, 5, 6]
```

Concatenou a list1 com a lista2 e atribuiu à lista 3. As listas 1 e 2 não foram modificadas

Métodos para Listas

extend(lista)

Adiciona ao final da lista outra lista.

```
lista1 = [1, 2, 3]
print(f"Lista1 = {lista1}")
>> Lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
print(f"Lista2 = {lista2}")
>> Lista2 = [4, 5, 6]
lista2.extend(lista1)
>> Lista2 = [4, 5, 6, 1, 2, 3]
print(f"Lista2 = {lista2}")
```

Criou a lista1 e lista2 e adicionou no final da lista2 a lista1.

Métodos para Listas

copy()

Copia uma lista em outra.

```
lista1 = [1, 2, 3]
lista2 = lista1.copy()
print(f"Lista1 = {lista1}")
>> Lista1 = [1, 2, 3]
print(f"Lista2 = {lista2}")
>> Lista2 = [1, 2, 3]
```

Criou a lista1 fez uma cópia na lista 2.

Métodos para Listas

sort(reverse)

Ordena os elementos da lista. O parâmetro reverse=True permite que seja ordenada em ordem decrescente.

```
lista = [19, 4, 25, 33, -5]
lista.sort()
print(lista)
>> [-5, 4, 19, 25, 33]
lista.sort(reverse=True)
print(lista)
>> [33, 25, 19, 4, -5]
```

Ordenou a lista em ordem crescente, depois em ordem decrescente

Métodos para Listas

reverse()

Inverte a ordem dos elementos dentro da lista.

```
lista = [19, 4, 25, 33, -5]
lista.reverse()
print(lista)
>> [-5, 33, 25, 4, 19]
```

Inverteu os elementos da lista, colocando o último elemento como primeiro e primeiro como último e assim sucessivamente.

Métodos para Listas

clear()

Apaga todos os elementos da lista.

```
lista = [19, 4, 25, 33, -5]
lista.clear()
print(lista)
>> []
```

Removeu todos os elementos da lista, mas ela continua existindo.

Métodos para Listas

del(lista)

Exclui (desaloca) a lista da memória

```
lista = [19, 4, 25, 33, -5]
del(lista)
>> a lista não existe mais
```

Excluiu a lista da memória com todos os seus elementos.

Listas – Lista de exercícios

1. Fazer um procedimento chamado ‘preenche_lista(l)’ que preencha uma lista passada por parâmetro.

```
preenche_lista(lista)
```

2. Fazer um procedimento chamado ‘exibe_lista(l)’ que exiba os elementos da lista passada por parâmetro..

```
exibe_lista(lista)
```

3. Sabemos que a função len() do Python retorna a quantidade de elementos de uma lista. Faça uma função chamada ‘conta_elementos(l)’ que tenha o mesmo efeito.

```
x = conta_elementos(lista)
```

4. Sabemos que a função index() do Python retorna o índice do elemento passado por parâmetro. Faça uma função parecida chamada ‘retorna_indice(elemento)’ com a melhoria de retornar -1 caso o elemento não seja encontrado.

```
x = retorna_indice(elemento)
```

5. Sabemos que a função count() do Python retorna a quantidade de um elemento especifico. Faça uma função chamada ‘busca(l,elemento)’ que tenha o mesmo efeito.

```
x = busca(lista, elemento)
```

6. Fazer uma função chamada ‘conta_inteiro(l)’ que conte quantos elementos inteiros existem em uma lista.

```
x = conta_inteiro(lista)
```

Listas – Lista de exercícios

6. Fazer uma função chamada ‘conta_string(l)’ que conte quantos elementos strings existem em uma lista.
7. Fazer uma função chamada ‘conta_boolean(l)’ que conte quantos elementos lógicos existem em uma lista.
8. Fazer uma função chamada ‘conta_float(l)’ que conte quantos elementos float existem em uma lista.
9. Fazer um procedimento chamado ‘copia_int(lista1, lista2)’ que verifique na lista1 se um dado é inteiro e copie para a lista2 este dado convertido em inteiro.

```
Lista1 = [‘eds’, ‘56’, ‘fiap’, ‘ester’, ‘True’, ‘45.78’, ’12’, ’78’]
6. >> 3 strings
7. >> 1 boolean
8. >> 1 float

If Not conta_inteiro(lista) and not in (‘True’, ‘False’) and dsafkasdjklfsdjlkfsajdlkf

“45,6”
[‘4’,’5’,’6’,’.’]
Split(), join()



9.
copia_int(lista1, lista2)
print(lista1) # [‘eds’, ‘56’, ‘fiap’, ‘ester’, ‘True’, ‘45.78’, ’12’, ’78’]
print(lista2) # [56, 12, 78]
```