Algoritmos e Estrutura de Dados I

Aula 12

Estruturas de Dados – Listas **(Vetores e Matrizes)**

Prof. Dr. Dilermando Piva Jr

1º Semestre - CDN

Estrutura de dados
LISTAS
Vetores e Matrizes

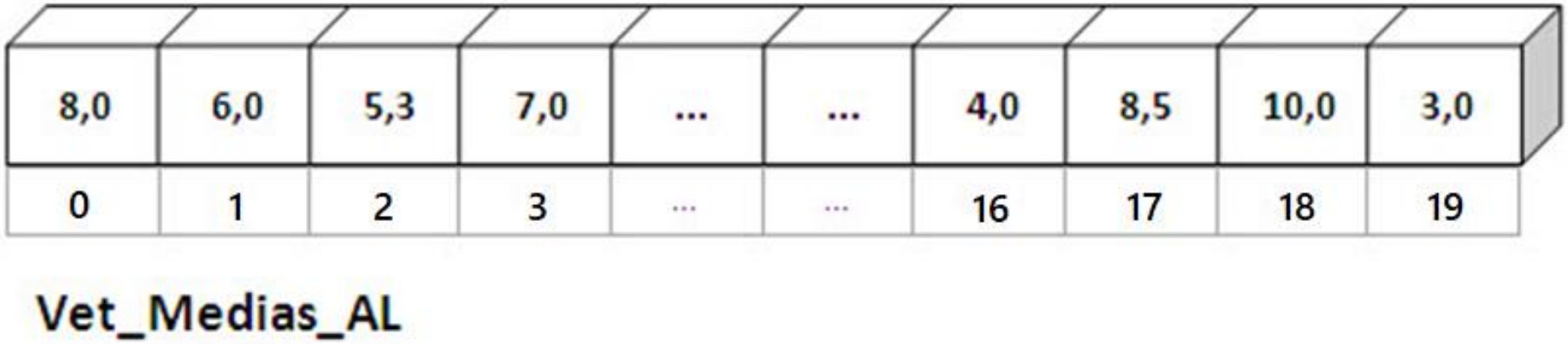
Para começar...

- Vamos imaginar um programa para armazenar as médias finais dos 20 alunos da disciplina de Algoritmos e, em seguida mostrar todas essas médias.
- Uma variável simples, ocupando determinada posição de memória, só consegue armazenar um valor, de um mesmo tipo de dado, por vez.
- Portanto, usando variáveis simples, cada nota digitada substituirá a anterior, dentro dessa variável.
- Para solucionar esse, e outros problemas relativos ao uso de variáveis temos o VETOR, também denominado variável composta homogênea unidimensional.

Vetores ou Listas

- O VETOR é uma variável composta homogênea Unidimensional.

Composta porque é constituído de n elementos ou variáveis;
Homogênea porque armazena dados de um único tipo; e
Unidimensional porque é linear ou seja possui somente uma dimensão.



Vetores ou Listas

- O VETOR é uma variável composta homogênea Unidimensional.

Composta porque é constituído de n elementos ou variáveis;
Homogênea porque armazena dados de um único tipo; e
Unidimensional porque é linear ou seja possui somente uma dimensão.

Listas...

- São coleções (caracteres, strings, números...) ordenados, separados por vírgula e que estão dentro de colchetes (limitados por colchetes).

- *Exemplo de Listas Homogêneas:*

[1,2,3,4,...]

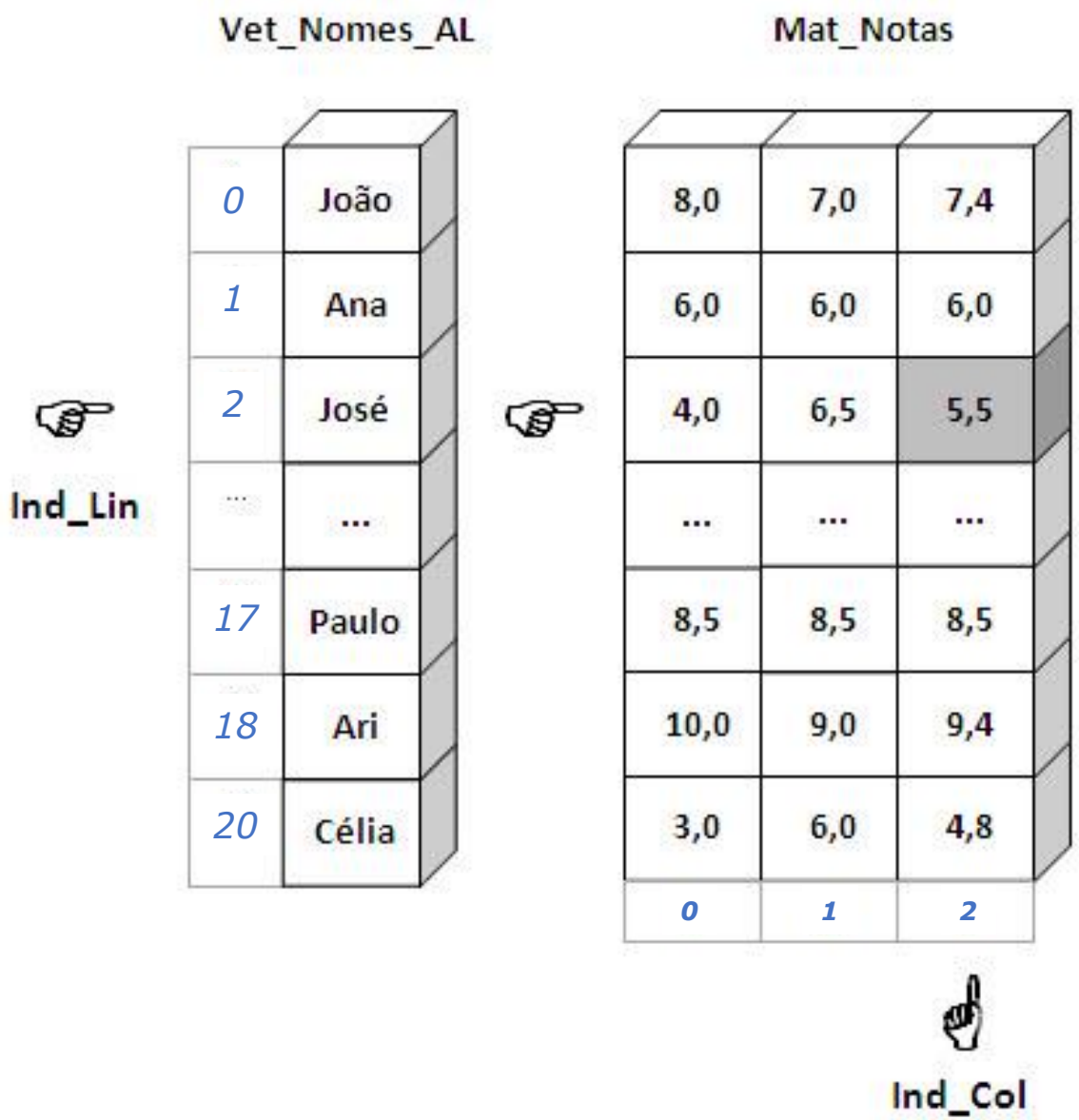
[‘a’, ‘b’, ‘c’, ...]

[‘texto1’, ‘texto2’, ...]

- *Exemplo de Listas Heterogêneas:*

[1, ‘a’, ‘texto1’, [1, 2, 3], 345, ...]

Listas de Listas (matrizes)



Listas...

- Atribuição e tipo
- Operações básicas
- A função list()
- Indexação e Fatiamento de Listas
- Modificação de itens de uma lista

Listas: constantes e índices

- Uma constante do tipo lista é escrita **entre colchetes** com os elementos **separados por vírgula**:

  [] # lista vazia

  [1,2] # lista com 2 elementos

- Os elementos de uma lista podem ser de qualquer tipo, *inclusive listas*. Ex.:

  lista = [1, 'a', 2+3j, ['ab', 'CD']]

- Os elementos de uma lista podem ser acessados por índices como strings
  - **O primeiro elemento tem índice 0**
  - O último elemento tem **índice -1**

Listas: constantes e índices

```
>>> lista = [1, 'a', 2+3j, ['ab', 'CD']]
>>> lista [0]
1
>>> lista [2]
(2+3j)
>>> lista [3]
['ab', 'CD']
>>> lista [-1]
['ab', 'CD']
>>> lista [0] = 2
>>> lista
[2, 'a', (2+3j), ['ab', 'CD']]
```

Listas...

- Atribuição e tipo
- Operações básicas
- A função list()
- Indexação e Fatiamento de Listas
- Modificação de itens de uma lista

Listas: Concatenação e Repetição

- O operador + pode ser usado para concatenação e o operador * para repetição

```
>>> lista = [0]*4
>>> lista
[0, 0, 0, 0]
>>> lista = lista + [1]*3
>>> lista
[0, 0, 0, 0, 1, 1, 1]
```

Deletando elementos

- O operador *del* pode ser usado para remover elementos de uma lista
- Ex.:

```
>>> lista
[1, 2, 3, ['ab', 'CD']]
>>> del lista [2]
>>> lista
[1, 2, ['ab', 'CD']]
>>> del lista [2][1]
>>> lista
[1, 2, ['ab']]
```

Operador “in”

- Permite saber se um elemento pertence a uma lista
- Serve também para strings
- Ex.:

```
>>> lista = [1, 'a', 'bc']
>>> 1 in lista
True
>>> 2 in lista
False
>>> 'b' in lista
False
>>> 'b' in lista[2]
True
>>> 'bc' in 'abcd'
True
```

Inicializando listas

- Não é possível atribuir a uma posição inexistente de uma lista

```
>>> vetor = []
>>> vetor [0] = 1
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in -toplevel-
    vetor [0] = 1
IndexError: list assignment index out of range
```

- Se uma lista vai ser usada como um array, isto é, vai conter um número predeterminado de elementos, é conveniente iniciá-la

```
>>> vetor = [0]*10
>>> vetor [0] = 3
>>> vetor
[3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Usando *None*

- No uso de estruturas de dados, às vezes é importante preencher uma posição com um valor “não válido”
- A melhor opção para esse uso é empregar o valor especial None
  - Não faz parte de tipo nenhum
  - É melhor que usar 0, [] ou uma string vazia
- Útil para criar uma lista “vazia” mas com um número conhecido de posições. Ex.:

```
>>> lista = [None]*5
>>> lista
[None, None, None, None, None]
```

A função *range*

- Gera uma progressão aritmética de inteiros, que pode ser colocada em uma lista
- Forma geral: range (*início*, *parada*, *incremento*)
  - *início* (opcional) é o primeiro valor a ser gerado (default: 0)
  - *parada* é o limite da progressão: a progressão termina no último valor antes de parada
  - *incremento* (opcional) é o passo da progressão (default:1)
- Ex.:

```
>>> list(range(3))
[0, 1, 2]
>>> list(range(2,5,2))
[2, 4]
>>> list(range(5,2,-2))
[5, 3]
```

Len, min e max

- len (*lista*) retorna o número de elementos de *lista*
- min *(lista*) e max *(lista*) retornam o menor/maior elemento de *lista*
- Ex.:

```
>>> lista = [1, 2, 9, 3, 4]
>>> min (lista)
1
>>> len (lista)
5
>>> max (lista)
9
>>> max (['a', 'b', 'c'])
'c'
```

min e *max*

- Na verdade, min e max podem ser usados também com vários argumentos ao invés de uma lista
- Ex.:

```
>>> min (1,2,3,4)
1
>>> max (3,4,5)
5
>>> max ([],[1],['a'])
['a’]  → versões anteriores a 3.5
Versões posteriores:
TypeError: '>' not supported between instances of 'str' and 'int'
```

Listas...

- Funções builtins usados em listas

Função	Descrição
`len()`	Retorna o número de elementos de uma lista
`sum()`	Retorna a soma dos números de uma lista
`any()`	Retorna True se qualquer um dos valores booleanos de uma lista for Verdadeiro
`all()`	Retorna True se todos os valores booleanos de uma lista forem Verdadeiros.
`sorted()`	Retorna uma cópia modificada (ordenada) de uma lista e mantem a lista original intacta.

Listas...

- Atribuição e tipo
- Operações básicas
- A função list()
- Indexação e Fatiamento de Listas
- Modificação de itens de uma lista

A função *list*

- Pode ser usada para converter uma string numa lista
- É útil pois uma lista pode ser modificada, mas uma string, não
- Para fazer a transformação inversa, pode-se usar o *método* join (veremos métodos mais tarde)
- Ex.:
  - >>> lista = list('alo')
  - >>> lista
  - ['a', 'l', 'o']
  - >>> lista[1]='xx'
  - >>> lista
  - ['a', 'xx', 'o']
  - >>> ''.join(lista)
  - 'axxo'

Listas...

- Atribuição e tipo
- Operações básicas
- A função list()
- Indexação e Fatiamento de Listas
- Modificação de itens de uma lista

Listas: fatias (slices)

- A notação de fatias também pode ser usada, inclusive para atribuição:

```
>>> lista = [1, 'a', 2+3j, ['ab', 'CD']]
>>> lista [1:]
['a', (2+3j), ['ab', 'CD']]
>>> lista [:1]
[1]
>>> lista [1:2]
['a']
>>> lista [0:-1]
[1, 'a', (2+3j)]
```

Listas: atribuição a fatias

- A atribuição a uma fatia requer que o valor atribuído seja uma sequência (uma lista ou uma string, por exemplo)
- A atribuição substitui os elementos da fatia pelos da sequência

```
>>> lista = [1, 'y', ['ab', 'CD']]
>>> lista [1:1] = ['z']
>>> lista
[1, 'z', 'y', ['ab', 'CD']]
>>> lista [1:3] = [['x']]
>>> lista
[1, ['x'], ['ab', 'CD']]
>>> lista [1:-1]= [2,3,4]
>>> lista
[1, 2, 3, 4, ['ab', 'CD']]
>>> lista [:2] = 'xyz'
>>> lista
['x', 'y', 'z', 3, 4, ['ab', 'CD']]
```

Incrementos em Fatias

- É possível usar um terceiro número na notação de fatias designando o incremento
  - Default é 1 , ou seja, toma os elementos de um em um do menor para o maior índice
  - Pode-se usar qualquer número inteiro diferente de 0
    - a[0:10:2] retorna uma lista com os 10 primeiros elementos de a tomados de 2 em 2 (5 elementos, no máximo)
    - a[5:0:-1] retorna uma lista com os 5 primeiros elementos de a tomados da esquerda para a direita

Incrementos em Fatias

- Exemplo

```
>>> a = ['a', 2, 3, 'd', 'x']
>>> a [:3:2]
['a', 3]
>>> a [::-1]
['x', 'd', 3, 2, 'a']
```

Incrementos em Fatias

- Se um incremento de fatia é diferente de 1, uma atribuição à fatia deve ter o mesmo número de elementos:

```
>>> l = [1,2,3,4,5]
>>> l [0::2] = ['x','y','z']
>>> l
['x', 2, 'y', 4, 'z']
>>> l [0::2] = [6,7]

Traceback (most recent call last):
  File "<pyshell#17>", line 1, in -toplevel-
    l [0::2] = [6,7]
ValueError: attempt to assign sequence of size 2 to extended slice of size 3
```

Listas...

- Atribuição e tipo
- Operações básicas
- A função list()
- Indexação e Fatiamento de Listas
- Modificação de itens de uma lista

Exemplo - Algoritmo

```
mat_notas = []
notas = []
vet_nomes_al = []
ind_lin, ind_col = 0, 0
for ind_lin in range(0, 5):
    nome = input("Nome do Aluno: ")
    vet_nomes_al.append(nome)
    for ind_col in range(0, 3):
        if ind_col != 2:
            nota = float(input(f'Digite a nota Bim {ind_col+1}: '))
            notas.append(nota)
        else:
            nota = float(input('Digite a Média Final : '))
            notas.append(nota)
    mat_notas.append(notas)
```

Exemplo – Para mostrar...

```
for ind_lin in range(0, 5):
    print(vet_nomes_al[ind_lin], " ", end='')
    for ind_col in range(0, 3):
        print(mat_notas[ind_lin][ind_col], " ", end='')
    print('')
```

Listas...

- Métodos de uma lista.

Função	Sintaxe	Descrição
`append()`	`List.append(item)`	Insere um elemento no final da lista. Modifica a original.
`count()`	`List.count(item)`	Retorna a quantidade de vezes que um item ocorre em uma lista
`insert()`	`List.insert(index,item)`	Insere um item em um dado local (index), deslocando os elementos da lista para direita.
`extend()`	`List.extend(List2)`	Adiciona os itens da Lista2 no final da lista List.
`index()`	`List.index(item)`	Busca um item em uma lista e retorna a sua posição. Se tiver mais de uma ocorrência do item, retorna a primeira. Se não encontrar o item, retorna Erro para o método.
`remove()`	`List.remove(item)`	Remove a primeira ocorrência do item em uma lista. Se o item não estiver presente na lista, retorna um erro para o método.
`sort()`	`List.sort()`	Ordena os itens de uma lista (na própria lista). Modifica a original.
`reverse()`	`List.reverse()`	Reverte a ordem dos itens de uma lista (na própria lista). Modifica a original.
`pop()`	`List.pop([index])`	Remove um determinado item de uma lista (retornando o valor do item da posição index). Se o valor de index for omitido, retorna o valor mais a direita.

"List" deve ser substituído pelo nome atual da lista (nome da variável)

Alguns métodos da classe *list*

- append(*elemento*)
  - Acrescenta o elemento no fim da lista
  - Observe que a operação *altera* a lista, e não simplesmente retorna uma lista modificada
  - Ex.:

```
>>> lista = [1,2]
>>> lista.append(3)
>>> lista
[1, 2, 3]
>>> lista.append([4,5])
>>> lista
[1, 2, 3, [4, 5]]
```

Alguns métodos da classe *list*

- count(*elemento*)
  - Retorna quantas vezes o elemento aparece na lista
  - Ex.:

```
>>> [1,2,3,1,2,3,4].count(1)
2
```

- extend(*lista2*)
  - Acrescenta os elementos de *lista2* ao final da lista
  - OBS.: *Altera* a lista ao invés de *retornar* a lista alterada
  - Ex.:

```
>>> lista=[1,2]
>>> lista.extend([3,4])
>>> lista
[1, 2, 3, 4]
```

Alguns métodos da classe *list*

- count(*elemento*)
  - Retorna quantas vezes o elemento aparece na lista
  - Ex.:

```
>>> [1,2,3,1,2,3,4].count(1)
2
```

- extend(*lista2*)
  - Acrescenta os elementos de *lista2* ao final da lista
  - OBS.: *Altera* a lista ao invés de *retornar* a lista alterada
  - Ex.:

```
>>> lista=[1,2]
>>> lista.extend([3,4])
>>> lista
[1, 2, 3, 4]
```

Alguns métodos da classe *list*

- insert(*indice, elemento*)
  - insere *elemento* na lista na posição indicada por *índice*
  - Ex.:

```
>>> lista = [0,1,2,3]
>>> lista.insert(1,'dois')
>>> lista
[0, 'dois', 1, 2, 3]
```

  - Como o extend, *altera* a lista ao invés de *retornar* a lista
    - O valor retornado é None!
  - Atribuições a fatias servem para a mesma finalidade mas são menos legíveis

```
>>> lista = [0,1,2,3]
>>> lista [1:1] = ['dois']
>>> lista
[0, 'dois', 1, 2, 3]
```

Alguns métodos da classe *list*

- index(*elemento*)
  - Retorna o índice da primeira ocorrência de *elemento* na lista
  - Um erro ocorre se *elemento* não consta da lista
  - Ex.:

```
>>> lista = [9,8,33,12]
>>> lista.index(33)
2
>>> lista.index(7)

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in -toplevel-
    lista.index(7)
ValueError: list.index(x): x not in list
```

Alguns métodos da classe *list*

- pop(*índice*)
  - Remove da lista o elemento na posição *índice* e o retorna
  - Se índice não for mencionado, é assumido o último
  - Ex.:

```
>>> lista = [1,2,3,4]
>>> lista.pop()
4
>>> lista
[1, 2, 3]
>>> lista.pop(1)
2
>>> lista
[1, 3]
```

Alguns métodos da classe *list*

- remove(*elemento*)
  - Remove da lista o primeiro elemento igual a *elemento*
  - Se não existe tal elemento, um erro é gerado
  - Ex.:

```
>>> lista = ['oi', 'alo', 'ola']
>>> lista.remove('alo')
>>> lista
['oi', 'ola']
>>> lista.remove('oba')

Traceback (most recent call last):
  File "<pyshell#24>", line 1, in -toplevel-
    lista.remove('oba')
ValueError: list.remove(x): x not in list
```

Alguns métodos da classe *list*

- reverse()
  - Inverte a ordem dos elementos da lista
  - Ex.:

```
>>> lista=[1,2,3]
>>> lista.reverse()
>>> lista
[3, 2, 1]
```

Alguns métodos da classe *list*

- sort(*cmp=None*, *key=None*, ***reverse=False***)
  - Ordena a lista
  - Os argumentos são opcionais. Por default, a lista é ordenada crescentemente
  - Ex.:

```
>>> lista = [9,8,7,1,4,2]
>>> lista.sort()
>>> lista
[1, 2, 4, 7, 8, 9]
```

Alguns métodos da classe *list*

- sort(*cmp=None*, *key=None*, *reverse=False*)
  - É possível obter a ordem inversa, passando *True* para o argumento *reverse*
  - Ex.:

```
>>> lista = [9,8,7,1,4,2]
>>> lista.sort(reverse=True)
>>> lista
[9, 8, 7, 4, 2, 1]
```

  - OBS.: A notação acima permite passar um argumento sem especificar os anteriores, mas poderíamos ter escrito:

```
>>> lista = [9,8,7,1,4,2]
>>> lista.sort(None,None,True)
>>> lista
[9, 8, 7, 4, 2, 1]
```

Alguns métodos da classe *list*

- sort(***cmp=None***, *key=None*, *reverse=False*)
  - O argumento *cmp* especifica uma função de comparação
    - É uma função que o sort chama para definir se um elemento é anterior ou posterior a outro
    - A função a ser passada tem a forma *comp*(*elem1*,*elem2*) e deve retornar um inteiro negativo caso *elem1* seja anterior a *elem2*, positivo caso *elem2* seja anterior a *elem1* e zero se tanto faz
  - Ex.:

```
>>> def compara(elem1,elem2):
        return elem1%10 - elem2%10
>>> compara(100,22)
-2
>>> lista=[100,22,303,104]
>>> lista.sort(compara)
>>> lista
[100, 22, 303, 104]
```

Alguns métodos da classe *list*

- sort(*cmp=None*, ***key=None***, *reverse=False*)
  - O argumento *key* especifica uma função aplicada a cada elemento
    - Se for passada uma função *f*, em vez de ordenar os elementos baseado em seus valores *v*, ordena baseado em *f(v)*
  - Ex.:

```
>>> lista = ['abc','de','fghi']
>>> lista.sort(key=len)
>>> lista
['de', 'abc', 'fghi']
```

Listas...

Fazer um programa em python que:

- Receba uma quantidade ilimitada (não se sabe quantas) de nome de cidades e guarde essas cidades em uma lista.
- Para encerrar a digitação das cidades, deve-se digitar: sair
- Em seguida, deve-se ordenar essa lista de cidades em ordem crescente.
- E mostrar a relação de cidades ordenadas, 1 por linha.

VAMOS PARA A PRÁTICA ?!!!

EXERCÍCIOS

Algoritmos ... vetores

EXERCÍCIO 1

Faça um algoritmo que carregue um vetor de 10 elementos numéricos inteiros.
Após a finalização da entrada, o algoritmo deve escrever o mesmo vetor, na ordem inversa de entrada.

EXERCÍCIO 2

Faça um algorimto que carregue um vetor de 5 elementos numéricos inteiros.
Após a finalização da entrada, o algoritmo deve escrever o maior valor e sua posição.

EXERCÍCIO 3

Faça algoritmo que carregue dois vetores (listas) de cinco elementos numéricos cada um e mostre um vetor resultante na intercalação desses dois vetores

EXERCÍCIO 4

Faça um algoritmo que leia 20 palavras de no máximo 10 caracteres, e após a leitura, realize um processo qualquer que inverta os caracteres de cada uma das palavras.

Valor Aleatório (randômico)

- Nas principais linguagens de programação existem comandos específicos para gerar números aleatórios.
- Em Python existe o pacote **random** e lá existem várias funções. Uma delas é a randint que retorna um valor inteiro entre x e y:
- randint(x,y)

RAND (exemplo)

```
from random import randint
a = randint(1, 60)
print(a)
```

Gera um número randômico entre 1 e 59 (60 é exclusivo)

EXERCÍCIO 5

Faça um Algoritmo que simule 6000 jogadas de um dado de 6 faces. Para simular o resultado utilize a função randint

Ao final, mostre a frequência de sorteio de cada uma das faces

Gerando números aleatrórios em Python

```python
from random import randint

# Gerando 10 jogadas
for i in range (0,10):
    lado = randint(1, 7)
    print(lado, end=" ")
```

Resultado

```
6 5 2 5 1 3 5 7 5 2
Process finished with exit code 0
```

EXERCÍCIO 6

Faça um algoritmo que simule a jogada de dois dados de 6 faces. O programa deve usar randint para rolar o primeiro dado e deve usar randint novamente para rolar o segundo dado. A soma das duas faces deve ser calculada. Assim: a soma variará de 2 a 12

O programa deve rolar 30.000 vezes e mostrar a frequência com que a soma (de 2 a 12) aparecem. Verifique se o valor 7 corresponde a 1/6 das jogadas!

EXERCÍCIO 7

Faça um algoritmo que armazenará os 10 primeiros números primos acima de 100. Ao final, o algoritmo deve mostrar os valores desse vetor.

EXERCÍCIO 8

Faça um algoritmo que lê 10 números inteiros e os armazena em um vetor A.
Depois de armazenado, o algoritmo fará a ordenação desses números (em ordem crescente de valores) e os colocará no vetor B
Ao final o algoritmo deve mostrar os dois vetores: A e B.

EXERCÍCIOS

Matrizes...

EXERCÍCIO 9

Faça um algoritmo que leia uma matriz 2x2 e imprima os seus elementos na ordem:

1,1 =

1,2 =

2,1 =

2,2 =

Obs: linha, coluna

EXERCÍCIO 10

Faça um algoritmo que leia uma matriz 2x2, calcule e mostre uma matriz resultante que será a matriz digitada, multiplicada pelo maior elemento da matriz.

EXERCÍCIO 11

Faça um algoritmo que leia os dados de uma matriz de 4 linhas e 4 colunas, composta de elementos reais, e calcule a soma dos elementos da diagonal principal da matriz.

EXERCÍCIO 12

Faça um algoritmo que leia os valores de uma matriz 3x3 de elementos reais e crie a matriz transposta da matriz fornecida.

Matriz transposta: Igual a simétrica. Em matemática, é o resultado da troca de linhas por colunas em uma determinada matriz.

$$A[i,j] = A[j,i]$$

EXERCÍCIO 13

Faça um algoritmo que receba uma matriz 10x10 de elementos inteiros e localize a posição (linha e coluna) do maior elemento da matriz.

EXERCÍCIO 14

Faça um algoritmo que leia uma matriz 10x20 com números inteiros e some cada uma das linhas, armazenando o resultado das somas em um vetor. A seguir, multiplique cada elemento da matriz pela soma da linha e mostre a matriz resultante.

EXERCÍCIO 15

Crie um algoritmo que receba uma matriz 8x8 com números inteiros e mostre uma mensagem dizendo se a matriz digitada é simétrica ou não. Uma matriz só pode ser considerada simétrica se $A[i,j] = A[j,i]$

EXERCÍCIO 16

Faça um algoritmo que receba uma matriz de 5x5 com números reais. Ao final o algoritmo deve calcular e mostrar a média dos elementos que estão nas linhas pares da matriz.