

Algoritmos e Lógica de Programação

80 horas // 4 h/semana

Vetores

Aula 9 – parte 1

Prof. Piva

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

- 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.

- Sendo o VETOR uma variável composta de n elementos, então devemos, no momento de sua definição, estabelecer o número máximo de elementos que ele irá conter.

Vetores

- Por exemplo, para armazenar as médias finais dos 20 alunos, usando variáveis simples, teríamos a seguinte definição em Algoritmo.

Var

Media_AL1, Media_AL2,...,Media_AL20 : Real

- Usando VETOR, teríamos a seguinte definição em Algoritmo

Var

Vet_Medias_AL : VETOR [1..20] de Real

Vetores

- Representação simbólica da alocação do vetor Vet_Medias_AL, na memória do computador:

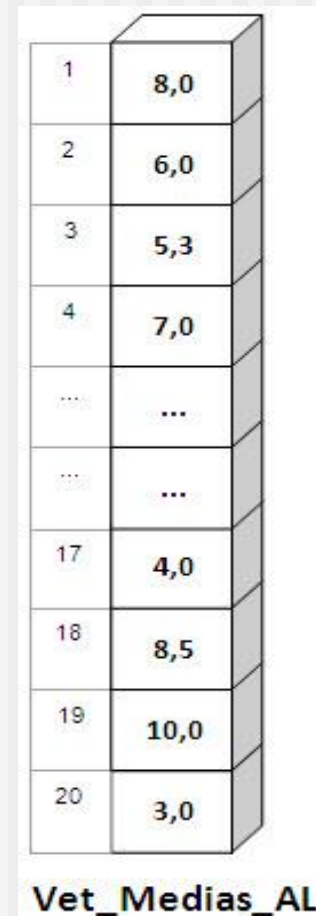
8,0	6,0	5,3	7,0	4,0	8,5	10,0	3,0
1	2	3	4	17	18	19	20

Vet_Medias_AL

- Essa representação simbólica demonstra o conceito da linearidade do Vetor.

Vetores

- A representação simbólica do Vetor poderia ser feita na forma vertical.



1	8,0
2	6,0
3	5,3
4	7,0
...	...
...	...
17	4,0
18	8,5
19	10,0
20	3,0

Vet_Medias_AL

Vetores

- Para o computador acessar um Vetor é preciso que ele conheça o Nome do Vetor, e um valor contido em uma Variável Índice que irá apontar para o elemento do vetor cujo conteúdo será acessado.
- O Índice de um vetor deve conter um valor numérico inteiro sem sinal, podendo ser:
 - a) uma variável simples;
 - b) uma constante numérica, ou mesmo;
 - c) uma expressão aritmética simples, desde que esta retorne um valor numérico inteiro sem sinal.

Vetores

- Definição do Índice do vetor Vet_Medias_AL:

Var

IndVet: Inteiro

- Supondo que Ind_Vet contenha um valor igual a 19. Com Ind_Vet apontando para o vetor Vet_Medias_AL, acessamos a média 10,0.

8,0	6,0	5,3	7,0	4,0	8,5	10,0	3,0
1	2	3	4	17	18	19	20

Vet_Medias_AL



Ind_Vet

Vetores

■ Combinando Vetores

Supondo que além de armazenar as médias finais dos alunos seja necessário armazenar , também, seus Nomes. Como dados Médias e Nomes são de tipos diferentes (real e cadeia de caracteres), então a solução é definir um vetor para conter os nomes – Vet_Nomes_AL, e em seguida combina-lo com o Vet_Medias_AL.

Definição dos vetores em VisuAlg:

Var

Vet_Medias_AL : VETOR [1..20] de Real

Vet_Nomes_AL : VETOR [1..20] de Caractere

Vetores

- O mesmo Índice que aponta para um elemento do vetor Vet_Medias_AL, apontará para o elemento correspondente no vetor Vet_Nomes_AL:

João	Maria	José	Paula	Vera	Pedro	Ana	André
1	2	3	4	17	18	19	20

Vet_Nomes_AL

Ind_Vet

8,0	6,0	5,3	7,0	4,0	8,5	10,0	3,0
1	2	3	4	17	18	19	20

Vet_Medias_AL

Ind_Vet

Algoritmos e Lógica de Programação

80 horas // 4 h/semana

Matrizes

Aula 9 – parte 2

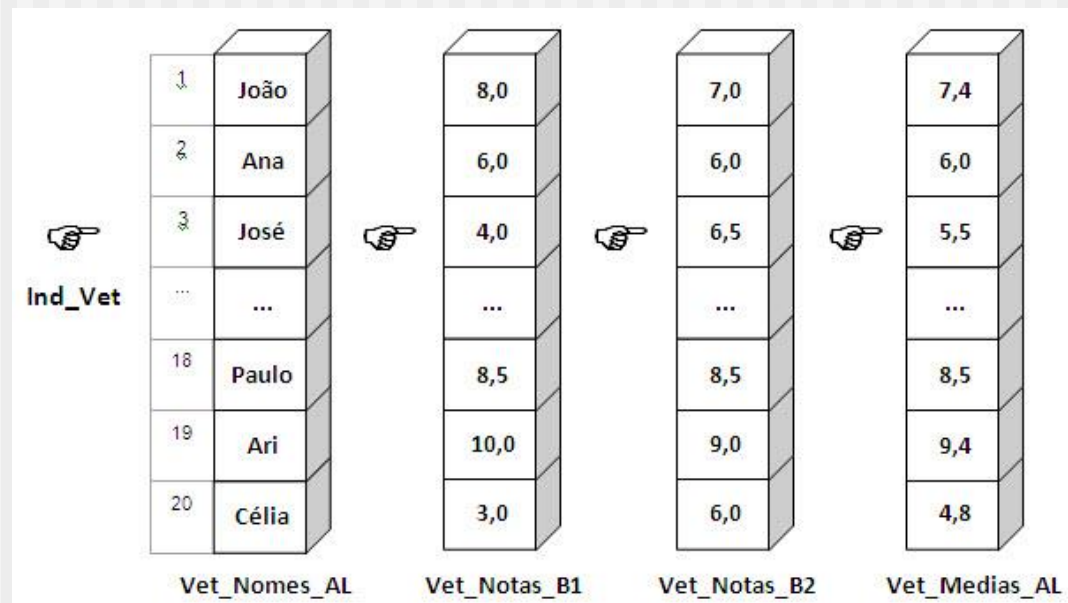
Prof. Piva

Para começar...

- Vamos considerar um algoritmo que foi elaborado para para armazenar os Nomes e as Médias Finais dos 20 alunos de uma determinada disciplina (em dois vetores distintos), em seguida , mostrar esses dados.
- Podemos alterar esse programa (algoritmo), para não receber a média final mas, calcular essa média a partir de duas notas bimestrais informadas pela secretaria acadêmica.
- Aplicando os conceitos sobre Vetor, precisaremos definir **3 vetores** para armazenarem respectivamente: nota bimestral 1, nota bimestral 2 e média final.

Para começar...

- Feita essa alteração, ao mostrar os dados de cada aluno, combinamos todos os vetores e acessamos seu conteúdo, conforme já aprendemos.
- Representação simbólica da solução usando vetores:



Matrizes

- A MATRIZ é uma variável composta homogênea Multidimensional.

Composta porque é constituído de n elementos ou variáveis; Homogênea porque armazena dados de um único tipo; e Multidimensional porque pode armazenar os dados e disponibilizá-los considerando multi perspectivas, ou seja, várias dimensões.

- Sendo a MATRIZ uma variável composta de n elementos, então devemos, no momento de sua definição, estabelecer o número máximo de elementos que ela irá conter, declarando suas dimensões.

Matrizes

- Exemplo de uma MATRIZ multidimensional são as agendas eletrônicas, cujos elementos de um único tipo de dado, que é textual, armazenam o descritivo dos compromissos agendados.
- Supondo que para ter acesso a um determinado compromisso nessa agenda, precisamos saber : o ano, mês, semana, dia, dia da semana e horário. Então essas são as dimensões da matriz, ou seja, a perspectiva dos armazenados.
- Essa matriz possui seis dimensões, e para acessá-las precisamos de cinco índices ou apontadores da matriz.

Matrizes

- Voltando ao exemplo das notas dos alunos, ao invés de definir 3 vetores para armazenar essas notas, podemos definir somente **uma Matriz**.
- Nessa Matriz, cada linha irá armazenar as notas de um aluno, e cada coluna dessa linha irá armazenar um tipo de nota.
- A coluna 3 da matriz receberá o resultado do cálculo da Média Final, cuja fórmula é:
$$\text{Média Final} = \text{Nota do Bimestre 1} \times 0,4 + \text{Nota do Bimestre 2} \times 0,6$$
- Essa matriz será bidimensional ou matriz linha x coluna.

Matrizes

- Definição da Matriz de notas em VisuAlg:

Var

Mat_Notas : VETOR [1..20,1..3] de Real

Note que sintaticamente, em relação ao vetor , a definição da matriz recebe uma ou mais dimensões.

No exemplo temos uma matriz com **20** linhas e **3** colunas cada linha.

- Para acessar o conteúdo dessa Matriz, precisamos definir dois índices, uma que irá apontar para a linha e outro para a coluna da matriz, onde na interseção linha x coluna, temos o dado do aluno.

Matrizes

- Definição da Matriz de notas em VisuAlg:

Var

Mat_Notas : VETOR [1..20,1..3] de Real

Note que sintaticamente, em relação a vetor , a definição da matriz contempla uma ou mais dimensões.

- Para acessar o conteúdo dessa Matriz, precisamos definir dois índices, uma que irá apontar para a linha e outro para a coluna da matriz.

Var

Mat_Notas : VETOR [1..20,1..3] de Real

Ind_Lin, Ind_Col : Inteiro

Na definição tanto do vetor, quanto da matriz, sintaticamente declaramos a palavra VETOR.

- Na interseção linha x coluna, temos o dado do aluno.

Matrizes

- Para o computador acessar uma Matriz é preciso que ele conheça o Nome da Matriz, e os valores contidos nas Variáveis Índices que apontarão para o elemento da Matriz, cujo conteúdo será acessado.
- O Índice de uma Matriz deve conter um valor numérico inteiro sem sinal, podendo ser:
 - a) **uma variável simples;**
 - b) **uma constante numérica, ou mesmo;**
 - c) **uma expressão aritmética simples, desde que esta retorne um valor numérico inteiro sem sinal.**

Matrizes

■ Combinando Vetor e Matriz

Supondo que além de armazenar as notas e médias finais dos alunos seja necessário armazenar , também, seus Nomes.

Como dados Notas e Nomes são de tipos diferentes (real e cadeia de caracteres), então a solução é definir um vetor para conter os nomes – Vet_Nomes_AL, e em seguida associá-lo com a matriz Mat_Notas.

Definição das variáveis em VisuAlg:

Var

Mat_Notas : VETOR [1..20,1..3] de Real

Vet_Nomes_AL : VETOR [1..20] de Caractere

Ind_Lin, Ind_Col : Inteiro

Matrizes

- Supondo que as colunas 1 e 2 da Mat_Notas armazenam, respectivamente, as Notas do bimestre 1 e 2, e a coluna 3 armazena a Média Final.
- Então, para acessar a Média Final do aluno José, temos a seguinte representação simbólica da combinação de Vetor e Matriz.

Vet_Nomes_AL		Mat_Notas		
1	João	8,0	7,0	7,4
2	Ana	6,0	6,0	6,0
3	José	4,0	6,5	5,5
...
18	Paulo	8,5	8,5	8,5
19	Ari	10,0	9,0	9,4
20	Célia	3,0	6,0	4,8
		1	2	3

Ind_Lin

Ind_Col

Ind_Lin = 3 e Ind_Col = 3

Exemplo...

- Considerando que o vetor:
- **Vet_Nomes_AL** e a matriz **Mat_Notas**
- já estão alocados na memória do computador, e já possuem valores
- Portanto, para acessar e mostrar o conteúdo deles, vamos construir trechos de programas em VisuAlg.

Exemplo - Algoritmo

Algoritmo "Media_Final"

Var

Mat_Notas : Vetor [1..20,1..3] de Real

Vet_Nomes_AL : Vetor [1..20] de Caractere

Ind_Lin, Ind_Col : Inteiro

Inicio

Limpatela

Para Ind_Lin := 1 ate 20 faca

Escreval ("Nome do Aluno(a) - ", Vet_Nomes_AL[Ind_Lin])

Escreval ("Nota do Bimestre 1: ", Mat_Notas[Ind_Lin,1])

Escreval ("Nota do Bimestre 2: ", Mat_Notas[Ind_Lin,2])

Escreval ("Média Final: ", Mat_Notas[Ind_Lin,3])

FimPara

.....

FimAlgoritmo

VAMOS PARA A PRÁTICA ?!!!



Python: Listas

Estruturas de dados

- Maneira de organizar dados de maneira a facilitar seu acesso
- Algumas formas são clássicas:
 - Listas
 - Arrays (vetores e matrizes)
 - Tuplas (registros)
 - Dicionários (chave – valor)
 - Árvores
- Linguagens frequentemente possuem primitivas para construção dessas E.D.
 - Estruturas de dados *embutidas*
- Outras E.D. mais complexas podem ser construídas combinando as E.D. clássicas

Estrutura de dados abstrata

- É uma especificação matemática que define uma coleção de dados e uma série de operações sobre ela
- É *abstrata* porque não especifica como as operações são feitas mas somente os dados de entrada e o resultado
- Numa linguagem de programação, essa coleção de operações é chamada de *interface* ou API (*Application Programming Interface*)
- Usuários da e.d.a devem se preocupar com a *interface* e não com a implementação, que pode mudar com o tempo
- A *implementação* de uma e.d.a. requer cuidados quanto à correção e a eficiência da mesma

Listas

- São arranjos sequenciais de informações mais simples
- Caracterizam-se por permitir o acesso eficiente aos seus elementos em ordem sequencial
- A definição clássica de uma lista como estrutura de dados abstrata compreende:
 - Operação de construção de uma lista vazia
 - Operação que testa se uma dada lista é vazia
 - Operação para obter o primeiro elemento de uma lista
 - Uma operação para adicionar um novo elemento no início de uma lista
 - Operação para retirar o elemento inicial de uma lista

Listas em Python

- A estrutura conhecida como *lista* (*list*, em inglês) em Python é mais geral do que e.d.a. *lista* clássica
- Na verdade, pode ser vista como uma implementação tanto de *listas* como de *arrays*
 - Além de acesso sequencial, suportam também acesso direto através de índices
- Listas são variedades de sequências assim como strings e portanto têm APIs semelhantes
 - Podem ser indexadas e fatiadas
 - Podem ser concatenadas (+) e repetidas

Listas em Python

- Entretanto, há diferenças importantes entre listas e strings
 - Seqüência genérica X de seqüência de caracteres
 - Elementos de listas podem ser alterados individualmente mas os de strings, **não**
- Listas constituem o tipo de agregação de dados mais versátil e comum da linguagem Python
 - Podem ser usadas para implementar estruturas de dados mais complexas como matrizes e árvores, por exemplo

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: 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']]
```

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

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]
```

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']
```

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'`

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]
```

Comando *for*

- Permite iterar sobre os elementos de uma lista, ou uma sequência de inteiros
- Forma geral: `for var in lista : comandos`
 - Os *comandos* são repetidos para cada valor de *lista*
 - Durante a repetição, *var* possui o valor corrente da *lista*
- Uma grande utilidade da função `range` é construir a lista de iteração
- Ex.:

```
>>>for i in range(1,7): print (i, end=" ")  
1 2 3 4 5 6
```

Comparando listas

- Listas são comparadas lexicograficamente
 - Se duas listas são iguais até os k -ésimos elementos, o resultado da comparação depende da comparação entre os $(k+1)$ -ésimos elementos
 - Se alguma das listas tem somente k elementos, então esta é a menor
 - Duas listas são iguais se e somente se têm o mesmo comprimento e todos os elementos de mesma posição são iguais
- Uma lista é maior que um número mas menor que uma string
 - Não me pergunte por quê!

Comparando listas

```
>>> [1,2] < [2, 3]
```

```
True
```

```
>>> [1,2] < [1, 2, 3]
```

```
True
```

```
>>> [1,2] != [1,2]
```

```
False
```

```
>>> min([[1],[2,3],[3,4],[]])
```

```
[]
```

```
>>> max([[1],[2,3],[3,4],[]])
```

```
[3, 4]
```

```
>>> min(0,[],"")
```

```
0
```

```
>>> max(0,[],"")
```

```
"
```

Variáveis do tipo *list*

- Uma variável do tipo lista na verdade *contém uma referência* para um valor do tipo lista
 - Atribuir uma variável a outra, cria uma nova referência mas não uma nova lista
 - Para se criar um novo valor, pode-se usar uma expressão que retorne o valor desejado
 - Para saber se duas variáveis se referem ao mesmo valor pode-se usar o operador **is**

Variáveis do tipo *list*

```
>>> a = b = [1,2,3]
```

```
>>> c = a
```

```
>>> d = c[:]
```

```
>>> a is b
```

```
True
```

```
>>> c is b
```

```
True
```

```
>>> d is c
```

```
False
```

```
>>> a [1]=5
```

```
>>> b
```

```
[1, 5, 3]
```

```
>>> d
```

```
[1, 2, 3]
```

A Classe *list*

- Uma lista é na verdade um *objeto* de uma *classe* chamada *list*
 - Não vimos ainda programação OO, mas alguns pontos devem ser enfatizados
- Listas possuem *métodos* que podem ser aplicados a elas
 - Um método é semelhante a uma função, mas são invocados de forma diferente: `objeto.método(args)`
 - Ex.: `lista.reverse()` inverte a ordem dos elementos da lista
 - Para saber todos os métodos de listas, escreva `help(list)`

A Classe *list*

■ Principais métodos...

<code>list.append(obj)</code>	Adiciona um objeto à lista
<code>list.insert(index, obj)</code>	Insere um objeto à lista em determinada posição
<code>list.count(obj)</code>	Retorna a quantidade de vezes que um determinado objeto ocorre na lista
<code>list.index(obj)</code>	Retorna o primeiro índice de ocorrência de um determinado objeto
<code>list.remove(obj)</code>	Remove um objeto da lista
<code>list.reverse()</code>	Reverte o ordenamento da lista
<code>list.sort()</code>	Ordena a lista

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*

■ `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*

■ `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*

■ `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']
```

Matrizes

- Listas podem ser usadas para guardar matrizes
- Por exemplo, podemos criar uma matriz-identidade de 3x3 com o código:

```
m = []  
for i in range(3):  
    m.append([0]*3)  
    m[i][i]=1
```

- Obs.: Não é boa ideia iniciar uma matriz assim:

```
m = [[0]*3]*3  
for i in range(3): m[i][i]=1  
print (m)
```

- Resultado: `[[1, 1, 1], [1, 1, 1], [1, 1, 1]]`
- Por quê? (Na realidade foram criadas referências)

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 algoritmo que carregue um vetor de 10 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 de dez 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 VisuALG existe o comando **RAND** que retorna um valor aleatório (randômico entre 0 e 1)

RAND (exemplo)

```
algoritmo "NUMERO ALEATÓRIO"
var
  x:inteiro
  y:real
inicio
  y<-rand
  x<-int(rand*10)
  escreval("x=", x)
  escreval("y=", y)
fimalgoritmo
```

```
Início da execução
x= 7
y= 0.203424050239846

Fim da execução.
```

Note que em cada atribuição,
rand gera um valor diferente!

EXERCÍCIO 5

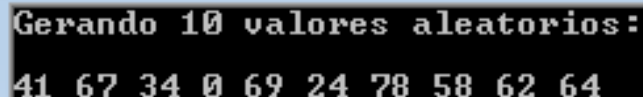
Faça um Algoritmo que simule 6000 jogadas de um dado de 6 faces. Para simular o resultado utilize a função **rand**

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

Gerando números aleatórios em C

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int main(void)
{
    int i;
    printf("Gerando 10 valores aleatorios:\n\n");
    for (i = 0; i < 10; i++)
    {
        /* gerando valores aleatórios entre zero e 100 */
        printf("%d ", rand() % 100);
    }
    getch();
    return 0;
}
```

A screenshot of a terminal window with a black background and white text. It displays the output of the C program: "Gerando 10 valores aleatorios:" followed by a new line and then ten random numbers: "41 67 34 0 69 24 78 58 62 64".

Gerando 10 valores aleatorios:
41 67 34 0 69 24 78 58 62 64

EXERCÍCIO 6

Faça um algoritmo que simule a jogada de dois dados de 6 faces. O programa deve usar `rand` para rolar o primeiro dado e deve usar `rand` 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 $\frac{1}{6}$ 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 1

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 2

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 3

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 4

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 5

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

EXERCÍCIO 6

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 7

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 8

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.