Aula 5

Professores:

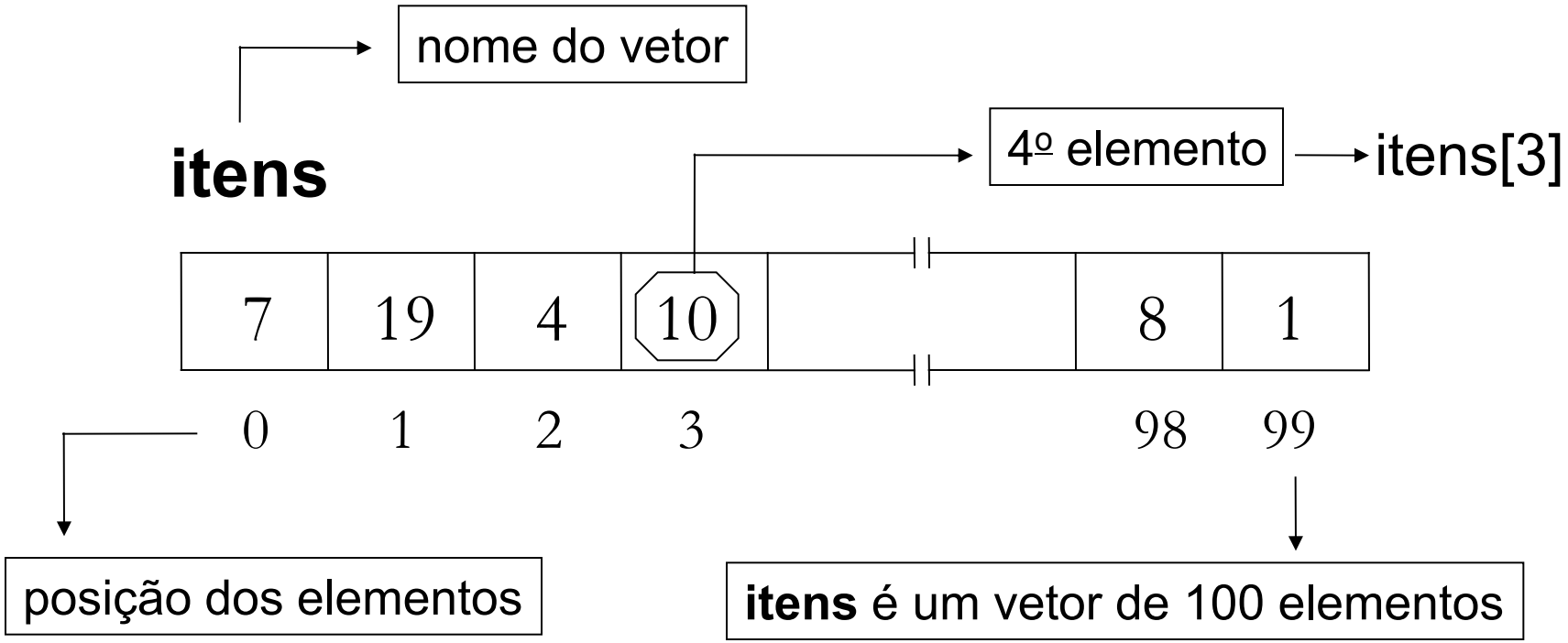
Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Dados:
  - Vetor
  - Matriz
  - String (Cadeia de Caracteres)
  - Tupla

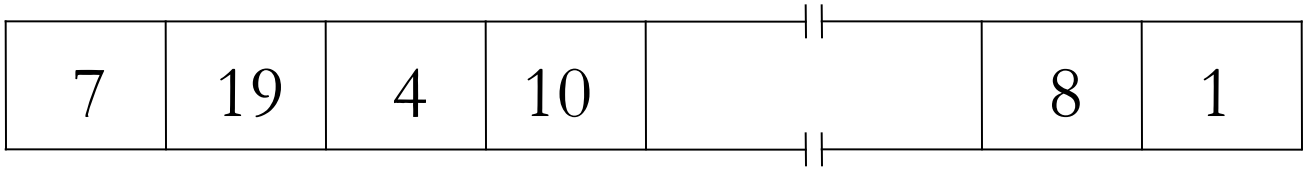
Vetor

Um vetor é um **agregado** de elementos (valores) de um mesmo tipo.



Vetor

itens



itens[0] representa o primeiro elemento e possui o valor 7.

itens[1] representa o segundo elemento e possui o valor 19.

⋮

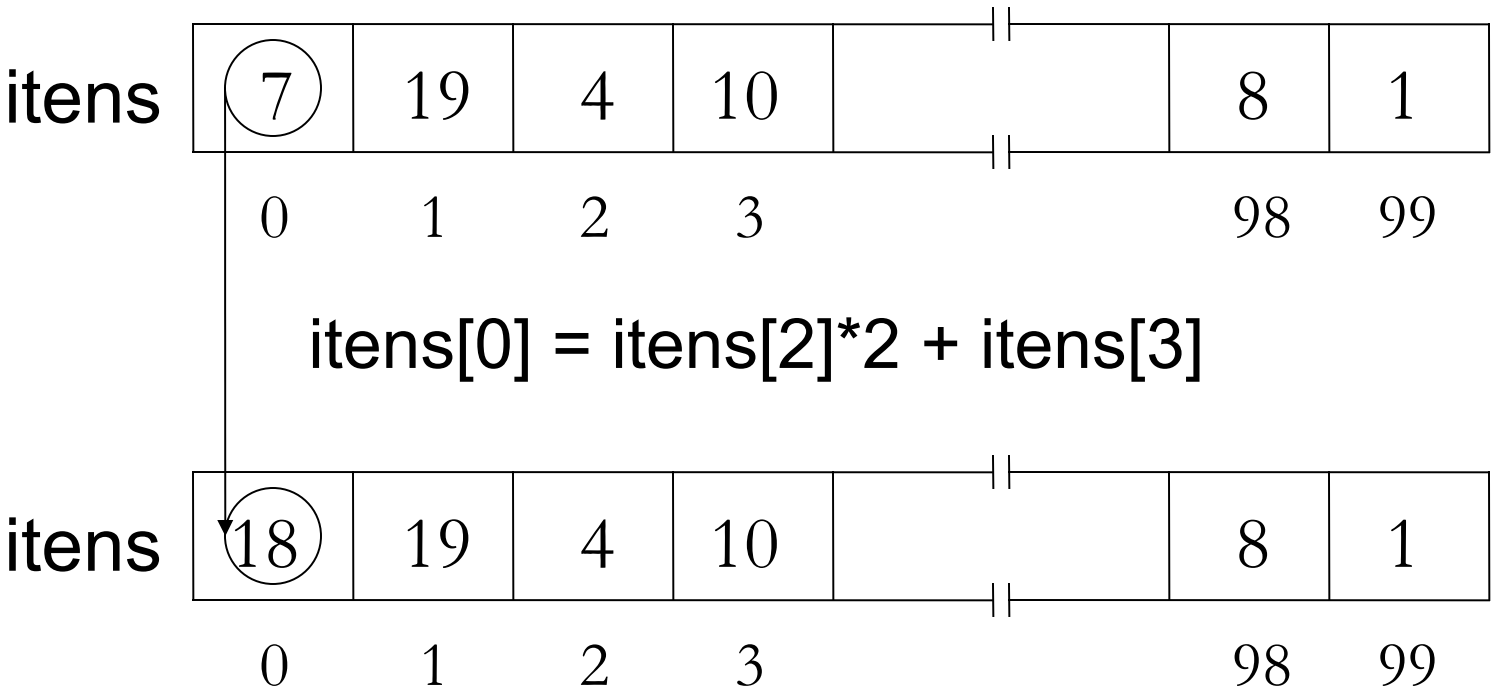
itens[99] representa o centésimo elemento e possui o valor 1.

itens[i-1] representa o i-ésimo elemento.

O índice ou seletor do elemento indica sua ordem (posição)

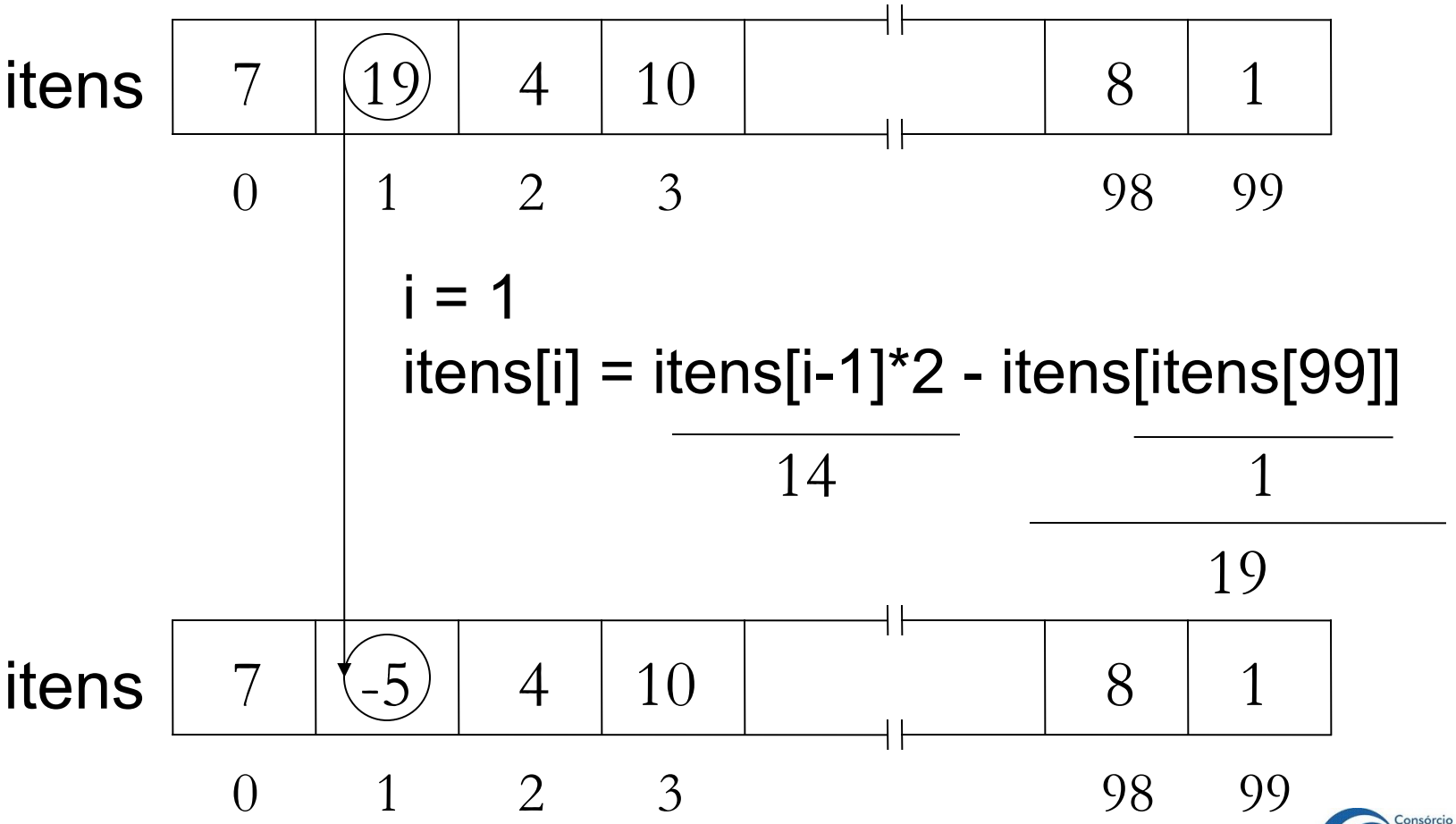
Vetor

Os elementos de um vetor podem ser caracterizados como variáveis comuns, podendo aparecer em expressões e atribuições.



Vetor

O índice de um elemento do vetor pode ser uma variável ou uma expressão.



Vetor

- Características:
  - Trata-se de uma estrutura homogênea, isto é, formada por elementos de um mesmo tipo, chamado tipo base;

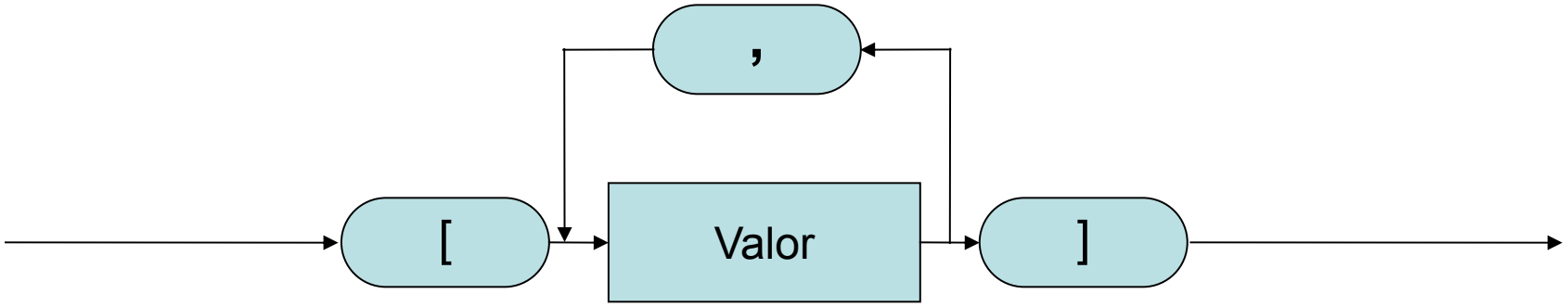
Vetor

- Características:
  - Trata-se de uma estrutura homogênea, isto é, formada por elementos de um mesmo tipo, chamado tipo base;
  - Todos os elementos da estrutura são igualmente acessíveis, ou seja, o tipo de procedimento para acessar qualquer elemento é igual;

Vetor

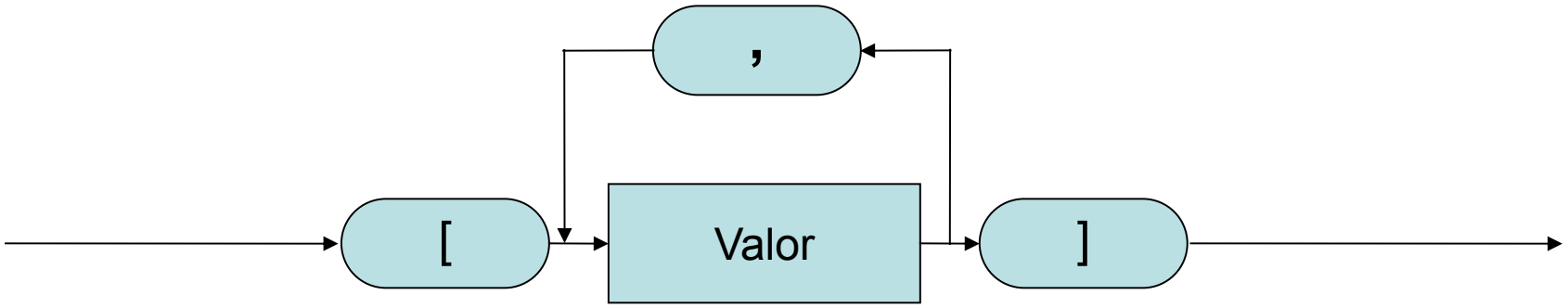
- Características:
  - Trata-se de uma estrutura homogênea, isto é, formada por elementos de um mesmo tipo, chamado tipo base;
  - Todos os elementos da estrutura são igualmente acessíveis, ou seja, o tipo de procedimento para acessar qualquer elemento é igual;
  - Cada elemento da estrutura tem um nome próprio, composto pelo nome do vetor e pelo índice.

Diagrama Sintático da Declaração de Vetor



Valor pode ser de qualquer tipo simples ou estruturado.

Diagrama Sintático da Declaração de Vetor



Valor pode ser de qualquer tipo simples ou estruturado.

Exemplo:

```
# Declarando e atribuindo vetores a variáveis
cantores = ['Chico Buarque', 'Gal Costa, 'Maria Bethania', 'Gil', 'Caetano']
pessoas = cantores                  # ambas variáveis compartilham o vetor
print(pessoas)                      # escreve ['Chico Buarque', ...]
```

Vetor

- Acesso a elementos no vetor para uso:
  - Suponha que se queira somar todos os elementos de um vetor.

```
# Solução 1
soma = 0
for indice in range(0,100):
    soma = soma + itens[indice]
```

```
# Solução 2
soma = 0
for item in itens:
    soma = soma + item
```

Vetor

- Acesso a elementos no vetor para uso:
  - Suponha que se queira somar todos os elementos de um vetor.

```
# Solução 1
soma = 0
for indice in range(0,100):
    soma = soma + itens[indice]
```

```
# Solução 2
soma = 0
for item in itens:
    soma = soma + item
```

  - A função **len(***nome do vetor***)** que retorna o tamanho de um vetor.

```
# Solução 3
soma = 0
for indice in range(len(itens)):
    soma = soma + itens[indice]
```

Vetor

- Acesso a elementos no vetor para uso:
  - O acesso deve acontecer dentro dos limites do índice.

```
# Erros
nomes = [‘Gal’, ‘Bethania’, ‘Chico’, ‘Caetano’, ‘Gil’]
i = 200
artista = nomes[i]   # Exceção - IndexError: index out of range
```

Vetor

- A leitura de um vetor pode ser realizada:
  - Elemento a elemento, ou
  - Todas as informações podem ser lidas de uma vez, como uma linha de caracteres, sobre a qual se aplica a operação **split** para separá-las.

Vetor

- A leitura de um vetor pode ser realizada:
  - Elemento a elemento, ou
  - Todas as informações podem ser lidas de uma vez, como uma linha de caracteres, sobre a qual se aplica a operação **split** para separá-las.
- A escrita de um vetor pode ser feitas
  - Toda de uma vez, ou
  - Pode ser realizada elemento a elemento.

Exemplos de Leitura

Exemplo 1: Lendo um vetor, todos os valores de uma vez, informados como texto e separados por espaços em branco

```
valores = input(“Digite os valores na mesma linha: ”).split()
```

Exemplos de Leitura

Exemplo 1: Lendo um vetor, todos os valores de uma vez, informados como texto e separados por espaços em branco

```
valores = input("Digite os valores na mesma linha: ").split()
```

Exemplo 2: Lendo um vetor com 10 valores, um de cada vez

```
valores = [None]*10
for i in range(len(valores)):
    valores[i] = input("Digite um valor: ")
```

Exemplos de Escrita

Exemplo 1: Escrevendo um vetor com 10 valores inteiros, todos de uma vez

```
numeros = [4, 13, 8, 4, 5, 2, 2, 1, 55, 27]
print(numeros)                  # todos de uma vez
```

Saída:

```
[4, 13, 8, 4, 5, 2, 2, 1, 55, 27]
```

Exemplos de Escrita

Exemplo 1: Escrevendo um vetor com 10 valores inteiros, todos de uma vez

```
numeros = [4, 13, 8, 4, 5, 2, 2, 1, 55, 27]
print(numeros)                  # todos de uma vez
```

Saída:

```
[4, 13, 8, 4, 5, 2, 2, 1, 55, 27]
```

Exemplo 2: Escrevendo um vetor com 10 valores inteiros, um de cada vez

```
numeros = [4, 13, 8, 4, 5, 2, 2, 1, 55, 27]
for x in numeros:
    print(x, end="  ")    # um de cada uma vez
print()                   # pula de linha
```

Saída:

```
4 13 8 4 5 2 2 1 55 27
```

Exemplo com Vetor

Especificação do Problema:
Faça um programa para ler, do teclado, dez números reais e escrevendo-os, na saída padrão (vídeo), em ordem crescente.

Metodologia para a Solução:
1. Caso não exista, crie um projeto do qual seu programa fará parte;
2. Dentro do projeto, crie um nome para o programa: exemplo.py;
3. Identifique a "necessidade" de constantes: TAM;
4. Declare as variáveis necessárias: numeros;

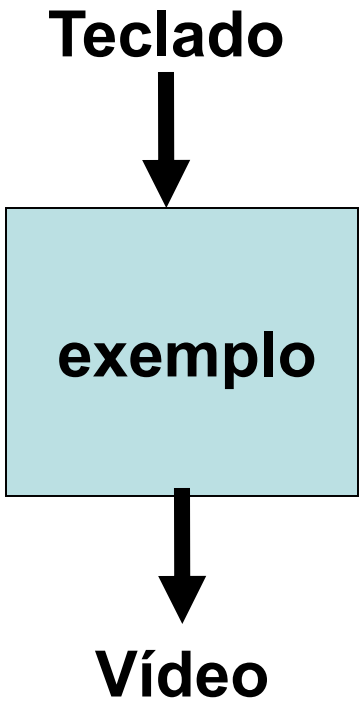
```
# Programa Principal - exemplo
TAM = 10
numeros = [0.0]*TAM          # Cria o vetor com dez valores zero
```

Exemplo com Vetor (continuação)

Metodologia para a Solução:

5. Identifique “macro” operações a serem realizadas: ler, ordenar e escrever;

```
# Subprogramas

# Programa Principal - exemplo
TAM = 10
numeros = [0.0]*TAM
ler(numeros)
escrever(numeros)
ordenar(numeros)
escrever(numeros)
```



Exemplo com Vetor (continuação)

Metodologia para a Solução:

6. Faça os cabeçalhos e blocos vazios (“stubs”) das operações;
7. Salve o programa novamente e execute-o;

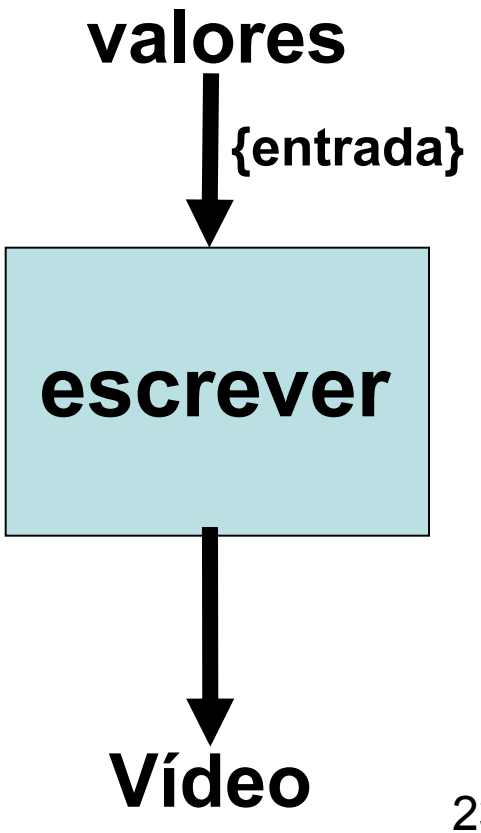
```
# Subprogramas
def escrever(valores):
    return None

def ler(valores):
    return None

def ordenar(valores):
    return None

# Programa Principal
TAM = 10
numeros = [0.0]*TAM
ler(numeros)
escrever(numeros)
ordenar(numeros)
escrever(numeros)
```

Exemplo com Vetor (continuação)

Metodologia para a Solução:

8. Escreva o corpo da primeira operação (escolhida por facilidade);
9. Salve e execute o programa;

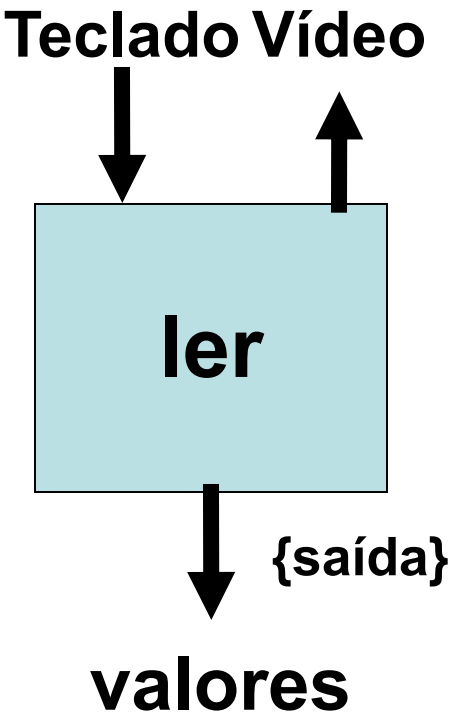
```
# Subprogramas
def escrever(valores):
    for item in valores:
        print(item, end=" ")
    print()
    return None
def ler(valores):
    return None
def ordenar(valores):
    return None
# Programa Principal
TAM = 10
numeros = [0.0]*TAM
ler(numeros)
escrever(numeros)
ordenar(numeros)
escrever(numeros)
```



Metodologia para a Solução:

10. Escreva o corpo de uma próxima operação (escolhida por facilidade);
11. Salve e execute o programa;

```
# Subprogramas
def escrever(valores):
    for item in valores:
        print(item, end=" ")
    print()
    return None
def ler(valores):
    for ind in range(len(valores)):
        valores[ind] = float(input("vs["+str(ind+1)+"] = "))
    return None
def ordenar(valores):
    return None
# Programa Principal
TAM = 10
numeros = [0.0]*TAM
ler(numeros)
escrever(numeros)
ordenar(numeros)
escrever(numeros)
```

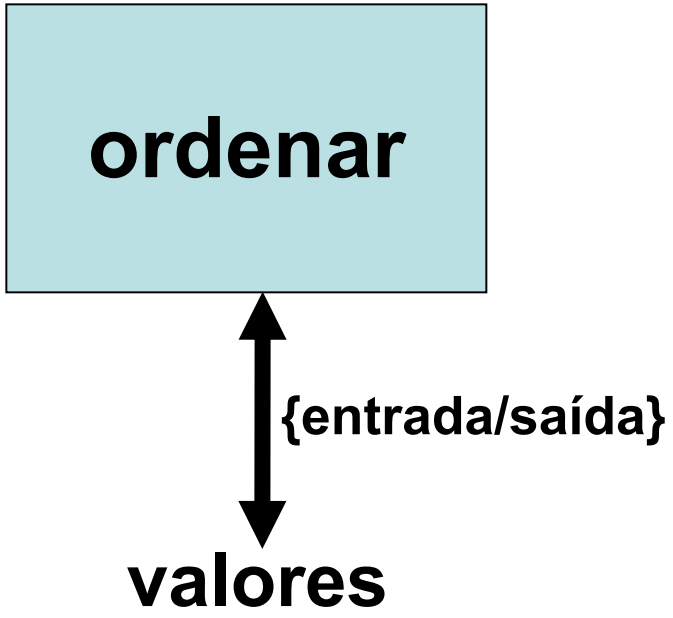


Exemplo com Vetor (continuação)

Metodologia para a Solução:

12. Escreva o corpo da próxima operação (escolhida por facilidade);

```
def ordenar(valores):
    # Função Interna: ondeMenor

    def ondeMenor(vals, inicio):
        posMenor = inicio
        for p in range(inicio+1, len(vals)):
            if vals[p]<vals[posMenor]:
                posMenor = p
        return  posMenor

    for ind in range(len(valores)-1):
        posicao = ondeMenor(valores, ind)
        temp = valores[ind]
        valores[ind] = valores[posicao]
        valores[posicao] = temp
    return None
```



```
# Subprogramas
def escrever(valores):
    for item in valores:
        print(item, end=" ")
    print()
    return None
def ler(valores):
    for ind in range(len(valores)):
        valores[ind] = float(input("vs["+str(ind+1)+"] = "))
    return None
def ordenar(valores):
  def ondeMenor(vals, inicio):
    posMenor = inicio
    for p in range(inicio+1, len(vals)):
      if vals[p]<vals[posMenor]:
        posMenor = p
    return  posMenor
  for ind in range(len(valores)-1):
    posicao = ondeMenor(valores, ind)
    temp = valores[ind]
    valores[ind] = valores[posicao]
    valores[posicao] = temp
  return None
# Programa Principal
TAM = 10
numeros = [0.0]*TAM
ler(numeros)
escrever(numeros)
ordenar(numeros)
escrever(numeros)
```

Formas Alternativas de Leitura de um Vetor

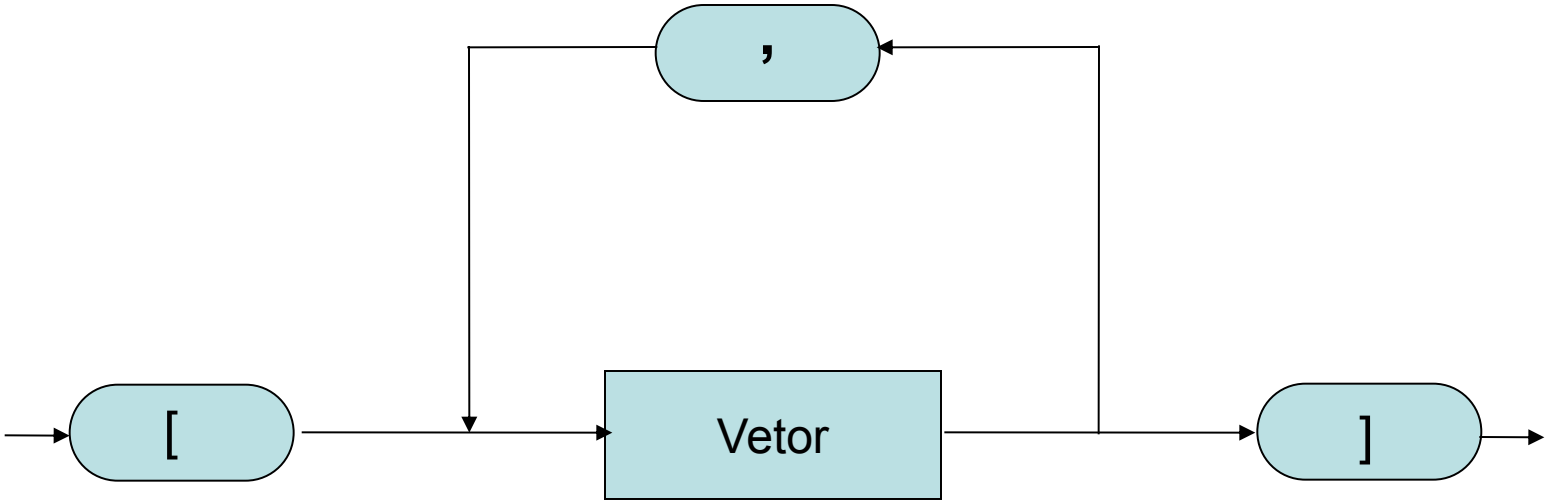
```
# Entrada de Valores de um Vetor: Elemento a Elemento.
def ler(valores):
    for ind in range(len(valores)):
        valores[ind] = float(input("vs["+str(ind+1)+"] = "))
    return None
```

Formas Alternativas de Leitura de um Vetor

```
# Entrada de Valores de um Vetor: Elemento a Elemento.
def ler(valores):
    for ind in range(len(valores)):
        valores[ind] = float(input("vs["+str(ind+1)+"] = "))
    return None
```

```
# Entrada de Valores de um Vetor: Todos de uma Vez.
def ler(valores):
    linha = input("Digite os Valores do Vetor, separados por espaços:\n")
    partes = linha.split()                    # particiona a linha em um vetor de strings.
    for ind in range(len(valores)):
        valores[ind] = float(partes[ind])     # converte uma string numérica em número.
    return None
```

Diagrama Sintático da Declaração de Matriz 2D



Matriz 2D é um Vetor de Vetores

A declaração a seguir corresponde a uma matriz (de duas dimensões) cujo tipo base é **boolean**.

```
celulas = [[True, False, False, True, True],
           [True, True, False, True, False]]
```

Matriz 2D é um Vetor de Vetores

A declaração a seguir corresponde a uma matriz (de duas dimensões) cujo tipo base é **boolean**.

```
celulas = [[True, False, False, True, True],
           [True, True, False, True, False]]
```

celulas

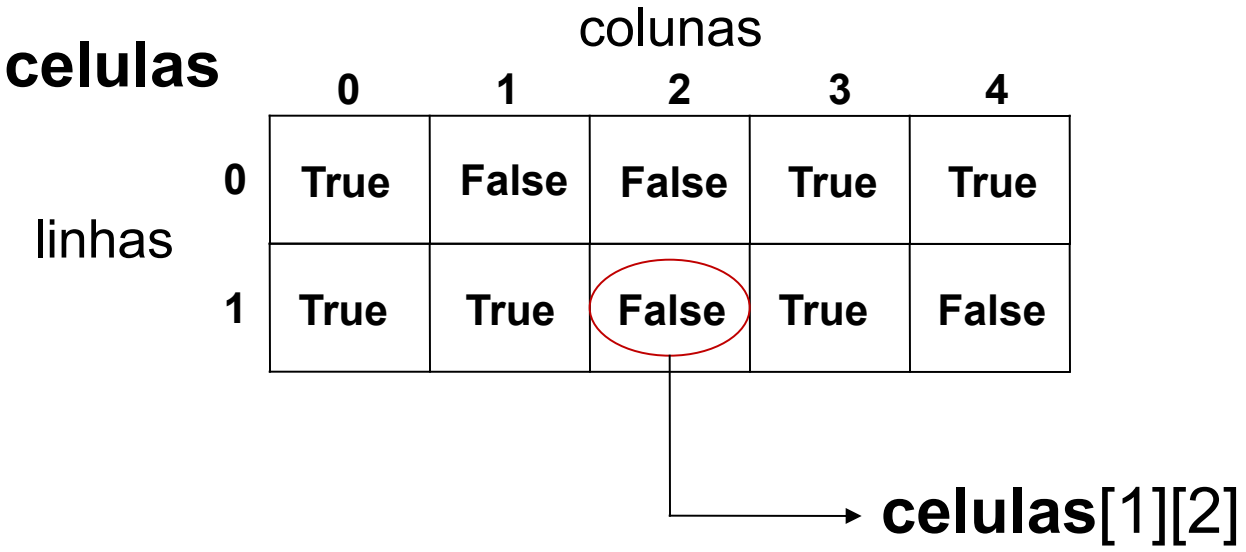
linhas \ colunas	0	1	2	3	4
0	True	False	False	True	True
1	True	True	False	True	False

O primeiro índice representa a linha e o segundo a coluna.

Matriz 2D é um Vetor de Vetores

A declaração a seguir corresponde a uma matriz (de duas dimensões) cujo tipo base é **boolean**.

```
celulas = [[True, False, False, True, True],
           [True, True, False, True, False]]
```



O primeiro índice representa a linha e o segundo a coluna.

Exemplos

Uma matriz de 13 linhas e 3 colunas para a loteria esportiva

```
aposta = [ [" ", "X", " "],
           ["X", " ", " "],
           [" ", " ", "X"],
           ["X", " ", " "],
           [" ", " ", "X"],
           ["X", " ", " "],
           [" ", "X", " "],
           [" ", "X", " "],
           [" ", " ", "X"],
           ["X", " ", " "],
           [" ", "X", " "],
           [" ", " ", "X"],
           [" ", " ", "X"] ]
```

aposta → representa uma matriz de 13 linhas por 3 colunas;
aposta[i] → representa a i-ésima linha da matriz, ou seja, um vetor de 3 caracteres;
aposta[i][j] → representa o caractere que está na linha i e coluna j.

Exemplos

Estado do Jogo da Velha

```
# O valor 0 representa que a célula está vazia, ou seja, ainda não usada.
tabuleiro = [[0,0,0],
             [0,0,0],
             [0,0,0]]
print(tabuleiro)
tabuleiro[1][1] = 1      # o jogador 1 marcou a célula central
print(tabuleiro)
tabuleiro[0][2] = 2      # o jogador 2 marcou a célula superior direita
print(tabuleiro)
tabuleiro[0][0] = 1      # o jogador 1 marcou a célula superior esquerda
print(tabuleiro)
tabuleiro[2][2] = 2      # o jogador 2 marcou a célula inferior direita
print(tabuleiro)
```

Exemplos

Tabela com os resultados de uma turma de cinco alunos com três provas cada.

```
# Subprograma
def listaAprovadosReprovados(estudantes, notas, minimo):
    return None
# Programa Principal
alunos = [“Maria”, “Lucas”, “Ana”, “Juca”, “Carlos”]
resultados = [    [7.2, 4.5, 6.1],
      [3.3, 8.5, 4.5],
      [7.8, 6.7, 8.3],
      [4.0, 6.0, 9.2],
      [2.3, 3.4, 4.0]    ]
print(alunos)
print(resultados)
listaAprovadosReprovados(alunos, resultados, 6.0)
```

```
# Subprograma
def listaAprovadosReprovados(estudantes, notas, minimo):
    for pos in range(len(estudantes)):
        media = (notas[pos][0]+notas[pos][1]+notas[pos][2])/3
        if media>=minimo:
            print(estudantes[pos], "Aprovado com nota:", media)
    print("------------------------------------------------------------")
    for pos in range(len(estudantes)):
        media = (notas[pos][0]+notas[pos][1]+notas[pos][2])/3
        if media<minimo:
            print(estudantes[pos], "Reprovado com nota:", media)
    return None

# Programa Principal
alunos = ["Maria", "Lucas", "Ana", "Juca", "Carlos"]
resultados = [ [7.2, 4.5, 6.1],
      [3.3, 8.5, 4.5],
      [7.8, 6.7, 8.3],
      [4.0, 6.0, 9.2],
      [2.3, 3.4, 4.0] ]
print(alunos)
print(resultados)
listaAprovadosReprovados(alunos, resultados, 6.0)
```

Exemplos

- Tabela com os resultados de uma turma de cinco alunos com três provas cada.
  - Outra representação: vetor de “vetores heterogêneos”

```
# Subprograma
def listaAprovadosReprovados(infos, minimo):
    return None
# Programa Principal
resultados = [ [“Maria”,   [7.2, 4.5, 6.1]],
      [“Lucas”,  [3.3, 8.5, 4.5]],
      [“Ana”,     [7.8, 6.7, 8.3]],
      [“Juca”,    [4.0, 6.0, 9.2]],
      [“Carlos”,  [2.3, 3.4, 4.0]] ]
print(resultados)
listaAprovadosReprovados(resultados, 6.0)
```

```
# Subprograma
def listaAprovadosReprovados(infos, minimo):
    for pos in range(len(infos)):
        media = (infos[pos][1][0]+infos[pos][1][1]+infos[pos][1][2])/3
        if media>=minimo:
            print(infos[pos][0], "Aprovado com nota:", media)
    print("-------------------------------------------------------------")
    for pos in range(len(infos)):
        media = (infos[pos][1][0]+infos[pos][1][1]+infos[pos][1][2])/3
        if media<minimo:
            print(infos[pos][0], "Reprovado com nota:", media)
    return None

# Programa Principal
resultados = [    ["Maria",   [7.2, 4.5, 6.1]],
    ["Lucas",  [3.3, 8.5, 4.5]],
    ["Ana",    [7.8, 6.7, 8.3]],
    ["Juca",   [4.0, 6.0, 9.2]],
    ["Carlos",  [2.3, 3.4, 4.0]]   ]
print(resultados)
listaAprovadosReprovados(resultados, 6.0)
```

Entrada Dinâmica de Vetores e Matrizes

Em Python, vetores e matrizes são implementados por listas, assunto que trataremos nas próximas aulas. No entanto, aqui utilizaremos a operação que anexa um valor ao final de uma lista: **append()**.

Entrada Dinâmica de Vetores e Matrizes

Em Python, vetores e matrizes são implementados por listas, assunto que trataremos nas próximas aulas. No entanto, aqui utilizaremos a operação que anexa um valor ao final de uma lista: **append()**.

```
# Subprograma
def listaAprovadosReprovados(infos, minimo):
    ...
    return None
def leAlunosComNotas(qtdAlunos, qtdNotas):
    resposta = []      # inicializa a resposta como uma lista vazia
    for indAluno in range(qtdAlunos):
        nome = input("Diga o nome do aluno "+str(indAluno+1)+" : ")
        linha = [nome,[]]   # cada linha tem um nome e uma lista vazia de notas
        for indNota in range(qtdNotas):
            nota = float(input("Diga a nota "+str(indNota+1)+" = "))
            linha[1].append(nota) # anexa ao final da lista de notas a nota lida
        resposta.append(linha)    # anexa ao final da lista a linha com nome e notas
    return resposta
# Programa Principal
resultados = leAlunosComNotas(5, 3)
listaAprovadosReprovados(resultados, 6.0)
```

String, ou Cadeia de Caracteres

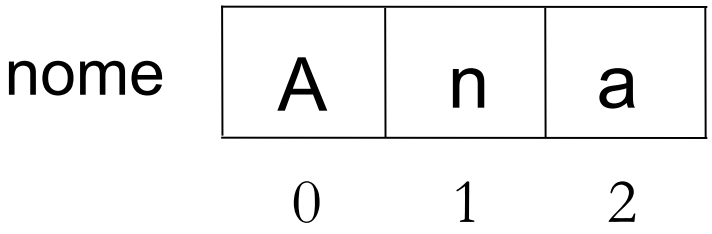
- Um objeto do tipo **str** representa uma cadeia de caracteres, de tamanho e valor imutáveis.

String, ou Cadeia de Caracteres

- Um objeto do tipo **str** representa uma cadeia de caracteres, de tamanho e valor imutáveis.

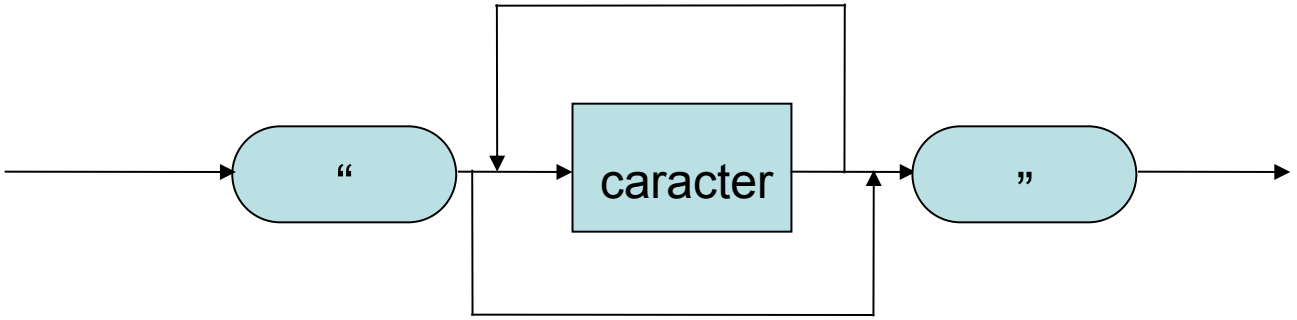
- Declaração e atribuição:

nome = “Ana”



  - O exemplo acima indica que a variável **nome** representa uma cadeia de 3 caracteres e valor definido.

Diagrama Sintático da Declaração de String



- Podem ser iniciados e finalizados por aspas duplas ou simples.
- Seu comprimento pode ser consultado pela função len(String).

```
# programa comprimento de uma String
x = “ABCD”
print(len(x))        # escreve 4.
```

String, ou Cadeia de Caracteres

- Comparação de Strings
  - Operadores relacionais: ==, !=, <=, >=, <, >

```
# programa comparando lexicograficamente Strings
x = “ABCD”
y = “ABCZ”
z = “ABCDEFG”

# As seguintes comparações retornam True
print(x < y)
print(x != “DCBA”)
print(x < z)
print(y > z)
```

String, ou Cadeia de Caracteres

- Indexação de cada caractere
  - O tipo String pode ser tratado também como um vetor.

String, ou Cadeia de Caracteres

- Indexação de cada caractere
  - O tipo String pode ser tratado também como um vetor.

- Considere a declaração e atribuição:

nome = “Ana”

Temos então que:

nome[0] = “A”
nome[1] = “n”
nome[2] = “a”

String, ou Cadeia de Caracteres

- O operador +, quando aplicado a dois operandos Strings x e y retorna uma String que é a concatenação das Strings x e y.

String, ou Cadeia de Caracteres

- O operador +, quando aplicado a dois operandos Strings x e y retorna uma String que é a concatenação das Strings x e y.

- Considere as declarações e atribuições:

nome = “Ana”
sobrenome = “Carvalho”
nomeCompleto = nome+sobrenome

Temos então que:

nomeCompleto = “AnaCarvalho”

String, ou Cadeia de Caracteres

- Retornando uma “nova” sub-String de uma String, via fatiamento:

nomeString[*posição inicial* : *posição final* + 1]

String, ou Cadeia de Caracteres

- Retornando uma “nova” sub-String de uma String, via fatiamento:

nomeString[*posição inicial* : *posição final* + 1]

- Considere as declarações o trecho de programa:

```
nome = “Ana Carvalho”
a = nome[4:12]
```

Temos então que:

a = “Carvalho”

String, ou Cadeia de Caracteres

- Método **find(***subStringProcurada***)**

  Retorna a posição do índice da primeira ocorrência da subStringProcurada na String sendo consultada. Caso não encontre, retorna menos um (-1).

  Considere as declarações e atribuições abaixo:

  nome = “Ana Carvalho”
  i = nome.find(“Carvalho”)

  Temos então que:

  i = 4

String, ou Cadeia de Caracteres

- Método **find(***subStringProcurada***)**

  Retorna a posição do índice da primeira ocorrência da subStringProcurada na String sendo consultada. Caso não encontre, retorna menos um (-1).

  Considere as declarações e atribuições abaixo:

  nome = “Ana Carvalho”
  i = nome.find(“Carvalho”)

  Temos então que:

  i = 4

- Método **find(***subStringProcurada*, *posInício***)** e método **find(***subStringProcurada*, *posInício*, *posFim***)**

  Retornam a posição do índice da primeira ocorrência da subStringProcurada na String sendo consultada, considerado a fatia de caracteres: a partir de posInício e entre posInício e posFim, respectivamente. Caso não encontrem, retornam -1.

Outros Métodos Importantes

1. replace(*subStringProcurada*, *subStringNova***)**

Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.

Outros Métodos Importantes

1. **replace(***subStringProcurada*, *subStringNova***)**
   Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.
2. **count(***subStringProcurada***)**
   Retorna a quantidade de ocorrências da *subStringProcurada* na String.

Outros Métodos Importantes

1. **replace(***subStringProcurada*, *subStringNova***)**
   Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.
2. **count(***subStringProcurada***)**
   Retorna a quantidade de ocorrências da *subStringProcurada* na String.
3. **upper()**
   Retorna uma cópia da String, convertendo os eventuais caracteres alfabéticos minúsculos para caracteres maiúsculos correspondentes.

Outros Métodos Importantes

1. **replace(***subStringProcurada*, *subStringNova***)**
   Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.
2. **count(***subStringProcurada***)**
   Retorna a quantidade de ocorrências da *subStringProcurada* na String.
3. **upper()**
   Retorna uma cópia da String, convertendo os eventuais caracteres alfabéticos minúsculos para caracteres maiúsculos correspondentes.
4. **lower()**
   Idem ao anterior, convertendo os maiúsculos para minúsculos.

Outros Métodos Importantes

1. **replace(***subStringProcurada*, *subStringNova***)**
   Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.
2. **count(***subStringProcurada***)**
   Retorna a quantidade de ocorrências da *subStringProcurada* na String.
3. **upper()**
   Retorna uma cópia da String, convertendo os eventuais caracteres alfabéticos minúsculos para caracteres maiúsculos correspondentes.
4. **lower()**
   Idem ao anterior, convertendo os maiúsculos para minúsculos.
5. **strip()**
   Retorna uma cópia da String, removendo todos os eventuais caracteres brancos do início e do final.

Outros Métodos Importantes

1. **replace(***subStringProcurada*, *subStringNova***)**
   Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.
2. **count(***subStringProcurada***)**
   Retorna a quantidade de ocorrências da *subStringProcurada* na String.
3. **upper()**
   Retorna uma cópia da String, convertendo os eventuais caracteres alfabéticos minúsculos para caracteres maiúsculos correspondentes.
4. **lower()**
   Idem ao anterior, convertendo os maiúsculos para minúsculos.
5. **strip()**
   Retorna uma cópia da String, removendo todos os eventuais caracteres brancos do início e do final.
6. **split()**
   Retorna uma lista de todas as palavras da String.

Outros Métodos Importantes

1. **replace(***subStringProcurada*, *subStringNova***)**
   Retorna uma cópia da String sendo consultada, substituindo todas as ocorrências da *subStringProcurada* pela *subStringNova*.
2. **count(***subStringProcurada***)**
   Retorna a quantidade de ocorrências da *subStringProcurada* na String.
3. **upper()**
   Retorna uma cópia da String, convertendo os eventuais caracteres alfabéticos minúsculos para caracteres maiúsculos correspondentes.
4. **lower()**
   Idem ao anterior, convertendo os maiúsculos para minúsculos.
5. **strip()**
   Retorna uma cópia da String, removendo todos os eventuais caracteres brancos do início e do final.
6. **split()**
   Retorna uma lista de todas as palavras da String.
7. **split(***subStringSeparadora***)**
   Retorna uma lista de todas as palavras da String, sendo o delimitador procurado entre palavras aquele especificado em *subStringSeparadora*.

String, ou Cadeia de Caracteres

- Leitura e Escrita
  - Variáveis do tipo String podem ser lidas e escritas através dos comandos **input**, **readline** (a ser visto), **print** e **write** (a ser visto).

Exemplo

Especificação do Problema:

Faça um programa para ler, do teclado, uma String contendo zero ou mais operandos numéricos e zero ou mais operadores de soma “+”, constituindo uma expressão numérica válida. Seu programa deve avaliar e escrever o valor resultante da expressão lida.

Exemplo

Especificação do Problema:

Faça um programa para ler, do teclado, uma String contendo zero ou mais operandos numéricos e zero ou mais operadores de soma “+”, constituindo uma expressão numérica válida. Seu programa deve avaliar e escrever o valor resultante da expressão lida.

Alguns Casos de Testes para o Problema:

Entrada: “”
Saída: {} = 0.0

Entrada: “ 13.17 ”
Saída: { 13.17 } = 13.17

Entrada: “2+3.11+4+5+2.3”
Saída: {2+3.11+4+5+2.3} = 16.41

Exemplo

```
# Subprograma
def avalia(expressao):
    valor = 0
    if expressao!="":
        partes = expressao.split("+")
        for p in partes:
            valor = valor + float(p)
    return valor

# Programa Principal
lida = input("Entre com uma expressão numérica válida: ")
print("{"+lida+"} =", avalia(lida.strip()))
```

Tupla

- Uma tupla é uma sequência ordenada de zero ou mais referências a objetos.

Tupla

- Uma tupla é uma sequência ordenada de zero ou mais referências a objetos.

- Suportam o mesmo fatiamento, o mesmo acesso por iteradores e o mesmo desempacotamento que Vetores e Strings.

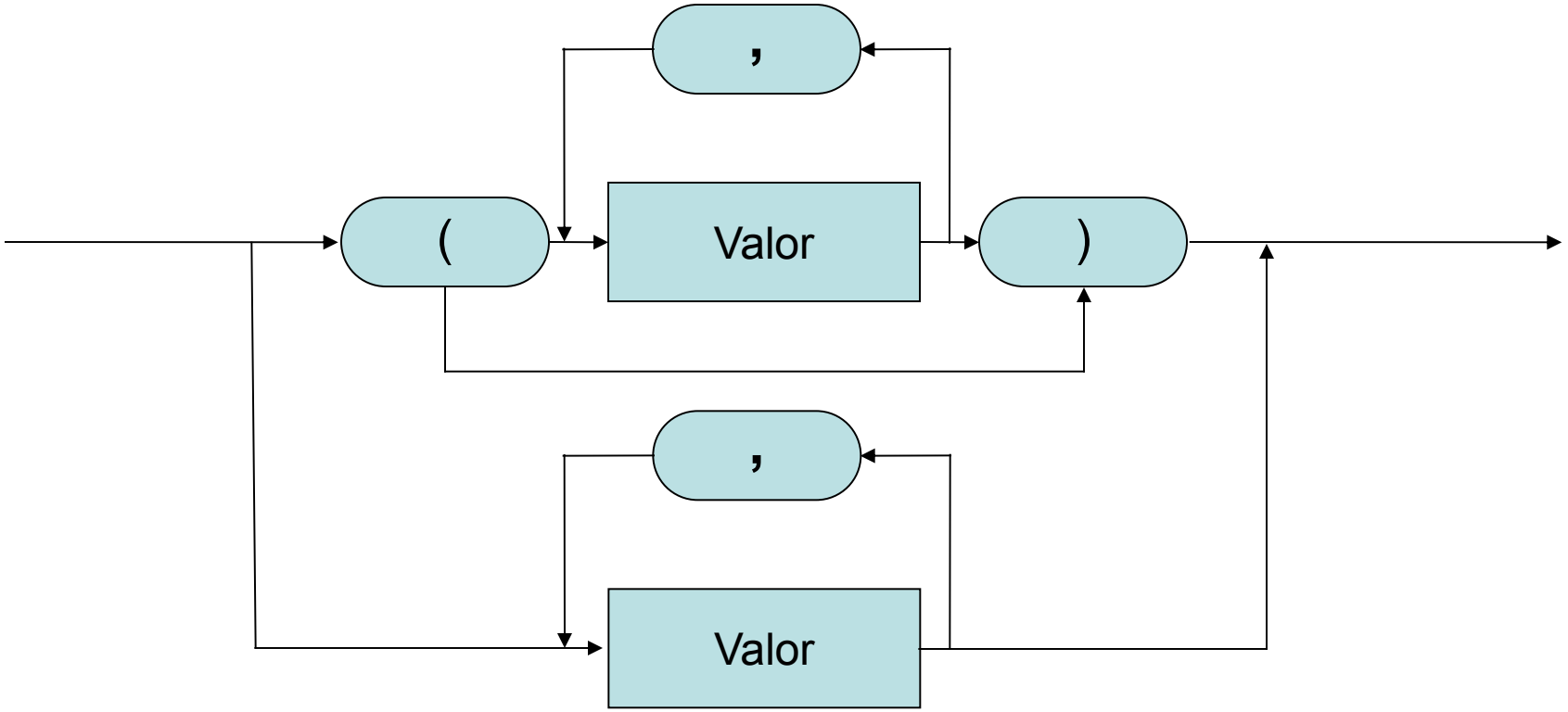
Tupla

- Uma tupla é uma sequência ordenada de zero ou mais referências a objetos.
- Suportam o mesmo fatiamento, o mesmo acesso por iteradores e o mesmo desempacotamento que Vetores e Strings.
- Assim como Strings, tuplas são imutáveis.

Tupla

- Uma tupla é uma sequência ordenada de zero ou mais referências a objetos.
- Suportam o mesmo fatiamento, o mesmo acesso por iteradores e o mesmo desempacotamento que Vetores e Strings.
- Assim como Strings, tuplas são imutáveis.
- A tupla pode ser vazia.

Diagrama Sintático da Declaração de Tupla



Valor pode ser de qualquer tipo simples ou estruturado.

Exemplos de Declaração e Empacotamento

A função **tuple()** retorna uma tupla vazia.

```
vazio = tuple()          # ou vazio = ( )
print(vazio)
```

Exemplos de Declaração e Empacotamento

A função **tuple()** retorna uma tupla vazia.

```
vazio = tuple()          # ou vazio = ( )
print(vazio)
```

Tuplas não vazias podem ser declaradas com ou sem parênteses.

```
valores = (“abacaxi”, 500, 4.99)
print(valores)

valores = “abacaxi”, 500, 4.99
print(valores)
```

Exemplos de Declaração e Empacotamento

A função **tuple()** retorna uma tupla vazia.

```
vazio = tuple()          # ou vazio = ( )
print(vazio)
```

Tuplas não vazias podem ser declaradas com ou sem parênteses.

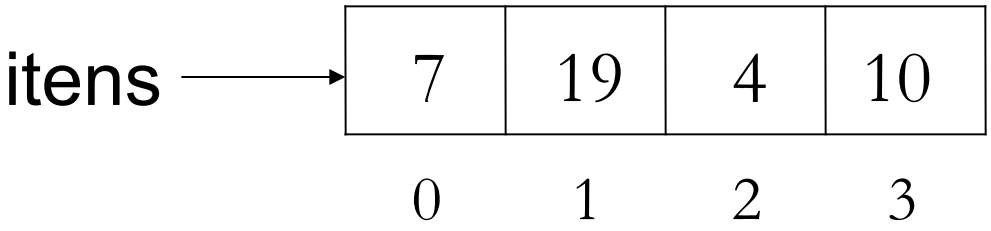
```
valores = (“abacaxi”, 500, 4.99)
print(valores)

valores = “abacaxi”, 500, 4.99
print(valores)
```

Vetores e Strings podem ser empacotados como tuplas através da função **tuple()**.

```
trio = tuple([1, 2, 3])    # ou trio = (1, 2, 3)
print(trio)

vogais = tuple(“aeiou”)    # ou vogais = (“a”, “e”, “i”, “o”, “u”)
print(vogais)
```

Exemplo de Acesso a Items

O índice de um elemento de uma tupla pode ser uma variável ou uma expressão.

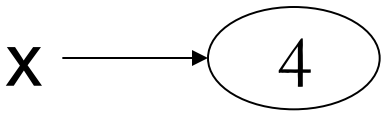


i = 1

x = itens[i-1]*2 - itens[3]

14 10

4



Exemplo de Desempacotamento e Iteração

O conteúdo de tuplas pode ser facilmente atribuído a variáveis isoladas por meio de desempacotamento.

```
nome, idade = (“Maria José”, 39)
print(nome)
print(idade)
```

ou

```
(nome, idade) = (“Maria José”, 39)
print(nome)
print(idade)
```

Exemplo de Desempacotamento e Iteração

O conteúdo de tuplas pode ser facilmente atribuído a variáveis isoladas por meio de desempacotamento.

```
nome, idade = (“Maria José”, 39)
print(nome)
print(idade)
```

ou

```
(nome, idade) = (“Maria José”, 39)
print(nome)
print(idade)
```

É possível iterar sobre os itens de uma tupla.

```
olhos = (“castanhos”, “verdes”, “azuis”, “pretos”)

for cor in olhos :
    print(cor)
```

Métodos Disponíveis sobre Tuplas

1. count(valor)

Retorna a quantidade de ocorrências de um determinado valor na tupla.

```
valores = (“abacaxi”, 500, 4.99, 500)
print(valores.count(500))                    # imprime 2
```

Métodos Disponíveis sobre Tuplas

1. count(valor)

Retorna a quantidade de ocorrências de um determinado valor na tupla.

```
valores = (“abacaxi”, 500, 4.99, 500)
print(valores.count(500))                    # imprime 2
```

2. index(valor)

Retorna o índice da primeira ocorrência do valor informado como argumento.

```
valores = (“abacaxi”, 500, 4.99, 500)
print(valores.index(500))                    # imprime 1
```

Operadores sobre Tuplas

1. Concatenação: a + b

Operador infixado que gera uma nova tupla a partir do conteúdo da primeira (a), seguido do conteúdo da segunda (b).

Operadores sobre Tuplas

1. Concatenação: a + b

Operador infixado que gera uma nova tupla a partir do conteúdo da primeira (a), seguido do conteúdo da segunda (b).

2. Replicação: a * *n*

Operador que gera uma nova tupla a partir do conteúdo da primeira (a), seguido pela repetição do mesmo conteúdo *n* – 1 vezes.

Operadores sobre Tuplas

1. Concatenação: a + b

Operador infixado que gera uma nova tupla a partir do conteúdo da primeira (a), seguido do conteúdo da segunda (b).

2. Replicação: a * *n*

Operador que gera uma nova tupla a partir do conteúdo da primeira (a), seguido pela repetição do mesmo conteúdo *n* – 1 vezes.

3. Fatiamento: a[*posição inicial* : *posição final* + *1*]

Operador que gera uma nova tupla a partir de um subconjunto de elementos contidos na tupla original (a).

Operadores sobre Tuplas

1. Concatenação: a + b

Operador infixado que gera uma nova tupla a partir do conteúdo da primeira (a), seguido do conteúdo da segunda (b).

2. Replicação: a * *n*

Operador que gera uma nova tupla a partir do conteúdo da primeira (a), seguido pela repetição do mesmo conteúdo *n* – 1 vezes.

3. Fatiamento: a[*posição inicial* : *posição final* + *1*]

Operador que gera uma nova tupla a partir de um subconjunto de elementos contidos na tupla original (a).

4. Operadores de atribuição incremental: a += b ou a *= *n*

Equivale aos operadores de concatenação e replicação, porém é atribuído à variável (a) a referência para a nova tupla gerada.

Operadores sobre Tuplas

1. Concatenação: a + b

Operador infixado que gera uma nova tupla a partir do conteúdo da primeira (a), seguido do conteúdo da segunda (b).

2. Replicação: a * *n*

Operador que gera uma nova tupla a partir do conteúdo da primeira (a), seguido pela repetição do mesmo conteúdo *n* – 1 vezes.

3. Fatiamento: a[*posição inicial* : *posição final* + *1*]

Operador que gera uma nova tupla a partir de um subconjunto de elementos contidos na tupla original (a).

4. Operadores de atribuição incremental: a += b ou a *= *n*

Equivale aos operadores de concatenação e replicação, porém é atribuído à variável (a) a referência para a nova tupla gerada.

5. Operadores de comparação: <, <=, ==, !=, > ou >=

Comparação item a item e, recursivamente, para itens aninhados.

Operadores sobre Tuplas

1. Concatenação: a + b

Operador infixado que gera uma nova tupla a partir do conteúdo da primeira (a), seguido do conteúdo da segunda (b).

2. Replicação: a * *n*

Operador que gera uma nova tupla a partir do conteúdo da primeira (a), seguido pela repetição do mesmo conteúdo *n* – 1 vezes.

3. Fatiamento: a[*posição inicial* : *posição final* + *1*]

Operador que gera uma nova tupla a partir de um subconjunto de elementos contidos na tupla original (a).

4. Operadores de atribuição incremental: a += b ou a *= *n*

Equivale aos operadores de concatenação e replicação, porém é atribuído à variável (a) a referência para a nova tupla gerada.

5. Operadores de comparação: <, <=, ==, !=, > ou >=

Comparação item a item e, recursivamente, para itens aninhados.

6. Teste de associação: in e not in

Verifica a pertinência de um valor em uma tupla.

Exemplo

Especificação do Problema:

Construa uma lista de compras representada por um vetor de tuplas, onde cada tupla contém o nome do produto, a quantidade e o preço unitário definidos pelo usuário. Mostre na saída padrão o conteúdo da lista e o total gasto na compra.

Formato da Entrada:

Cada produto comprado é especificado pelo usuário em três linhas. A primeira contém o nome do produto, a segunda contém a quantidade e a terceira o preço unitário. O produto com nome “Fim” representa o término da lista e não deve ser incluído na mesma.

Exemplo

Especificação do Problema:

Construa uma lista de compras representada por um vetor de tuplas, onde cada tupla contém o nome do produto, a quantidade e o preço unitário definidos pelo usuário. Mostre na saída padrão o conteúdo da lista e o total gasto na compra.

Formato da Entrada:

Cada produto comprado é especificado pelo usuário em três linhas. A primeira contém o nome do produto, a segunda contém a quantidade e a terceira o preço unitário. O produto com nome “Fim” representa o término da lista e não deve ser incluído na mesma.

Exemplo de Entrada:

```
Fruta do Conde
3
7.80
Leite
2
3.99
Fim
```

```
# Subprograma
def preencher():
    itens = []
    nome = input("Nome do Produto: ")
    while nome != "Fim":
        qtd = int(input("Quantidade: "))
        preco = float(input("Preco Unitario: "))
        itens.append((nome, qtd, preco))
        nome = input("Nome do Produto: ")
    return itens

# Subprograma
def processa(itens):
    total = 0.0
    for (nome, qtd, preco) in itens:
        total += qtd * preco
        print("Nome:", nome, "Quantidade:", qtd, "Preço:", preco)
    print("Total Gasto:", total)
    return None

# Programa Principal
compras = preencher()
processar(compras)
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:

Aula 5

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Dados:
  - Vetor
  - Matriz
  - String (Cadeia de Caracteres)
  - Tupla