



Professores:

Aula 1

Dante Corbucci Filho

Leandro A. F. Fernandes

Conteúdo:

- Introdução
- Ambiente de Desenvolvimento
- Correção de Exercícios

Objetivos

- Capacitar o aluno no uso de uma linguagem de programação procedural (neste caso Python) para:
 - Implementar
 - Executar
 - Testaras diferentes soluções concebidas para resolver um problema

Objetivos

- Capacitar o aluno no uso de uma linguagem de programação procedural (neste caso Python) para:
 - Implementar
 - Executar
 - Testaras diferentes soluções concebidas para resolver um problema
- Ao final da disciplina o aluno deverá estar apto a:
 - Implementar
 - Testar
 - Analisar
 - Documentarprogramas de computador em uma linguagem imperativa

Ementa

- (1) Introdução
- (2) O Ambiente de Desenvolvimento de Programas
- (3) Variáveis, Tipos e Comandos Básicos
- (4) Estruturas de Controle: Sequência, Seleção e Repetição
- (5) Subprogramação: Funções, Passagem de Parâmetros e Recursividade
- (6) Representação de Dados na forma de:
 - (6.1) Listas
 - (6.2) Vetores e Matrizes
 - (6.3) Strings e Tuplas
 - (6.4) Arquivos
 - (6.5) Conjuntos
 - (6.6) Dicionários
- (7) Noções de Complexidade de Algoritmo

Livros Texto

- Summerfield, M.
Programação em Python 3
Uma Introdução Completa à Linguagem Python
Editora Alta Books
(2013)
- Barry, P. & Griffiths, D.
Use a Cabeça! Programação.
Editora Alta Books
(2010)
- Barry, P.
Use a Cabeça! Python.
Editora Alta Books
(2012)

Algoritmo e Estrutura de Dados [Guim 94]

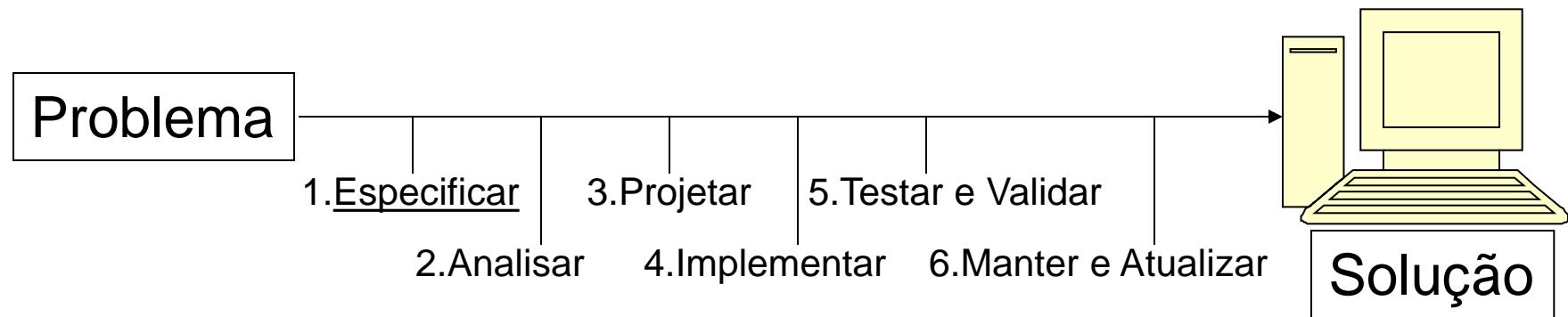
- Estrutura de dados é um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados eficientemente.
- Algoritmo é a descrição de um padrão de comportamento, especificado em termos de um conjunto bem definido e finito de ações primitivas que podem ser executadas.

Programa = Algoritmo + Estruturas de Dados

Programação Estruturada [Guim 94]

- Metodologia de projeto de programas que visa:
 1. Facilitar o desenvolvimento dos programas
 2. Facilitar a leitura (entendimento) dos programas
 3. Permitir a validação *a priori* dos programas
 4. Facilitar a manutenção e modificação dos programas

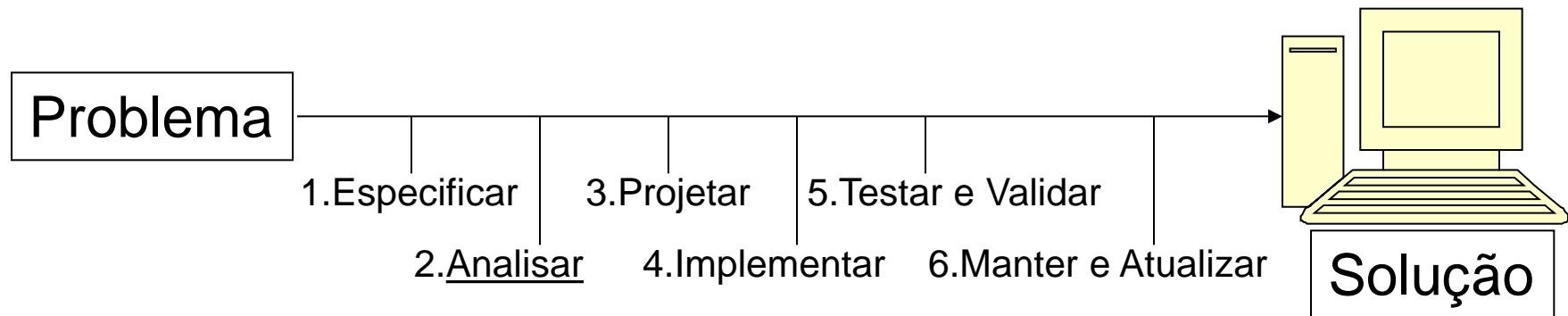
Ciclo de Vida do Software [Koff 94]



1. Especificar os Requisitos do Problema

- Preparar uma especificação completa e não ambígua.

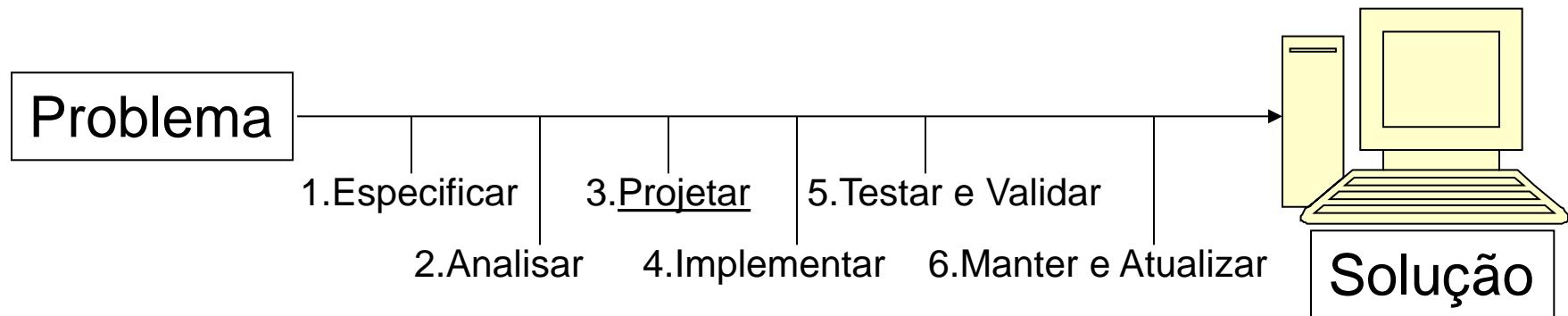
Ciclo de Vida do Software [Koff 94]



2. Analisar o Problema

- Entender o problema,
- Avaliar soluções alternativas,
- Escolher a solução mais adequada.

Ciclo de Vida do Software [Koff 94]

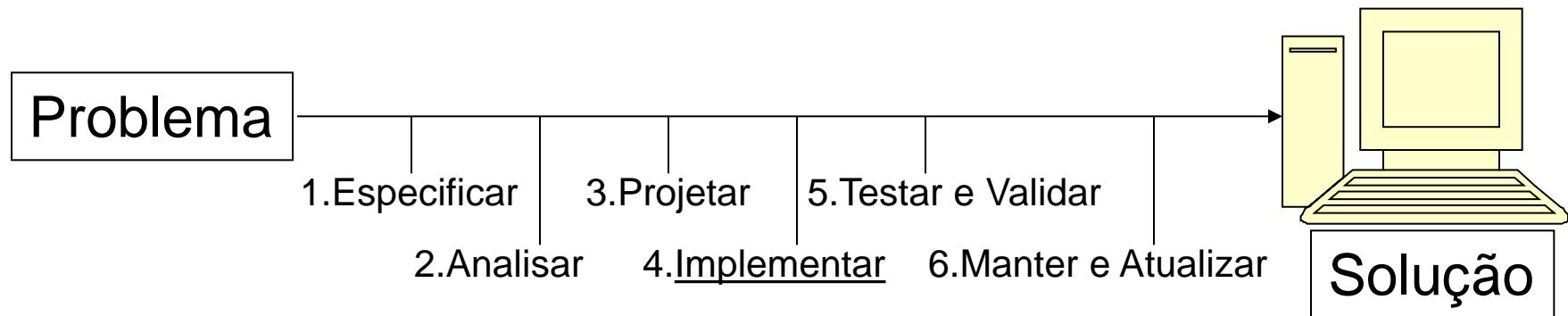


3. Projetar o Programa para Solucionar o Problema

- Fazer projeto de cima-para-baixo (*top-down*) do sistema,
- Para cada módulo, identificar as principais estruturas de dados e subprogramas associados,
- Desenvolver algoritmos e estruturas de dados dos subprogramas.

10

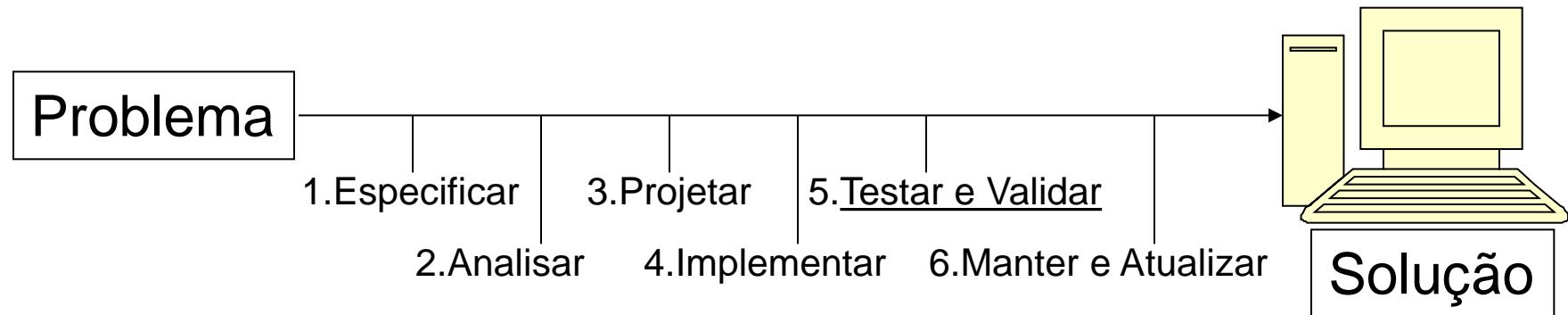
Ciclo de Vida do Software [Koff 94]



4. Implementar o Projeto

- Codificar a solução,
- Corrigir erros de codificação.

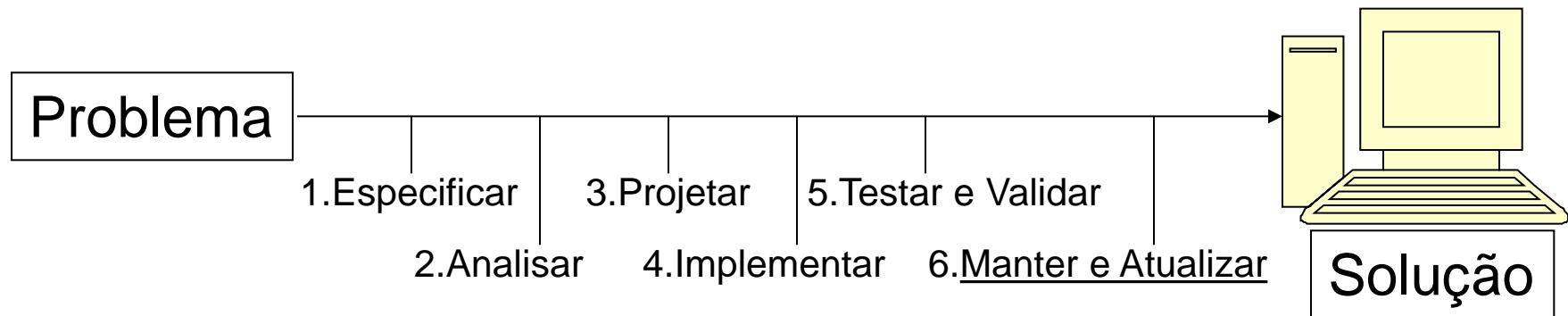
Ciclo de Vida do Software [Koff 94]



5. Testar e Validar o Programa

- Testar o código e validá-lo, se correto.

Ciclo de Vida do Software [Koff 94]



6. Manter e Atualizar o Programa

- Executar o sistema,
- Avaliar seu desempenho,
- Remover novos erros identificados, assim que detectados,
- Realizar modificações de forma a manter o sistema atualizado,
- Validar as modificações.

A Linguagem Python

- Python foi criado idealizado na década de 80 e sua implementação começou em 1989, por Guido van Rossum
 - 2000: Python 2
 - 2008: Python 3
- Características da linguagem:
 - Multiparadigma (estruturado, orientado a objetos, funcional e orientado a aspectos)
 - Multiplataforma (Windows, Linux, iOS, etc.)
 - Interpretada
 - Tipagem dinâmica
 - Gerência de memória automatizada (coletor de lixo)



Guido van Rossum,
o criador do Python

14

O Ambiente de Desenvolvimento

- Ambiente de desenvolvimento de software Python
 - Usaremos Python 3
- O ambiente de desenvolvimento Python inclui:
 - Um amplo conjunto de Interfaces de Programação de Aplicações (APIs, do inglês “*Application Programming Interfaces*”)
 - Ferramentas de compilação e depuração de código
- Python 3 não é um Ambiente Integrado de Desenvolvimento (IDE, do inglês “*Integrated Development Environment*”)
 - A linguagem Python por si só não oferece editores de código ou ambiente de programação
 - Usaremos PyCharm como IDE (editor + ambiente de programação)



Instalação do Python 3

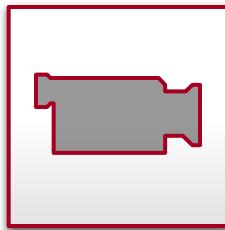
(<https://www.python.org>)

e

do PyCharm

(<https://www.jetbrains.com/pycharm>)

Clique no botão para assistir ao tutorial:



16

Correção de Exercícios

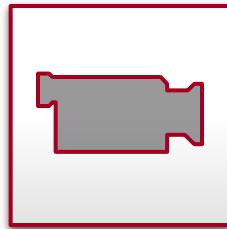
- Faça o Chinês, ou Teste de Mesa
 - Validação do programa “em papel”
 - Represente graficamente as variáveis utilizadas e acompanhe a atualização de seus valores em função da execução passo-a-passo do algoritmo implementado
 - Você desempenha a tarefa do compilador e do executor do programa!!!
- Juiz Remoto (“*Online Judge*”):
 - Sistema web que compila e executa um código fonte submetido a julgamento. Ele testa a saída gerada pelo programa submetido contra um banco de testes, com entradas e saídas previamente elaboradas.
 - Avalia se, para toda entrada conhecida pelo juiz, o programa submetido produz saída idêntica à esperada
 - Usaremos o URI Online Judge

Cadastro no URI Online Judge (<http://urionlinejudge.com.br>)

e

Submissão de Um Exemplo

Clique no botão para assistir ao tutorial:



Referências Autorais

- Estas aulas foram preparadas a partir de uma primeira versão, em Pascal, da Disciplina Fundamentos de Programação, do Curso de Tecnologia em Sistemas de Computação do Cederj.
- Autores da versão Pascal:
Prof. Alexandre Plastino de Carvalho e
Prof. Dante Corbucci Filho
- A versão Pascal foi produzida baseada no livro:
Título: Pascal e Técnicas de Programação
Autores: Eber Assis Schmitz e
Antônio Aníbal de Souza Teles
Editora: LTC



Professores:

Aula 1

Dante Corbucci Filho

Leandro A. F. Fernandes

Conteúdo:

- Introdução
- Ambiente de Desenvolvimento
- Correção de Exercícios



Aula 2

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Variáveis
- Tipos
- Comandos Básicos

Variáveis

- Área de memória que mantém um valor, que pode ser mudado.
- Identificador da variável: é o nome da variável, representada por uma sequência de caracteres, iniciada por uma letra minúscula.
 - Exemplos: nota, saldo, deposito, saque, casa13Buzios.

Variáveis

- Área de memória que mantém um valor, que pode ser mudado.
- Identificador da variável: é o nome da variável, representada por uma sequência de caracteres, iniciada por uma letra minúscula.
 - Exemplos: nota, saldo, deposito, saque, casa13Buzios.
- Nossa padronização: não deve ter acento e deve possuir apenas caracteres alfa-numéricos. Usar notação de camelCase
 - Exemplos: minhaNota, notaTurma, cartoesAmarelos, cartoesVermelhos.
 - Isto é, sempre que for uma composição de várias palavras, iniciar a próxima com letra maiúscula e as demais letras sempre deverão ser minúsculas.

Tipos Básicos (Embutidos)

- Tipos básicos são imutáveis

Tipos Básicos (Embutidos)

- Tipos básicos são imutáveis
- Tipos Integrais
 - Inteiro: (**int**)
 - Pode ter centenas de dígitos, limitado apenas pela memória do computador;
 - O padrão é decimal, mas pode-se usar outras bases como binária (iniciada com **0b**), octal (iniciada com **0o**) ou hexadecimal (iniciada com **0x**).
 - Lógico (ou Booleano): (**bool**)
 - 0 é **False** e 1 é **True**.

Tipos Básicos (Embutidos)

- Tipos básicos são imutáveis
- Tipos Integrais
 - Inteiro: (**int**)
 - Pode ter centenas de dígitos, limitado apenas pela memória do computador;
 - O padrão é decimal, mas pode-se usar outras bases como binária (iniciada com **0b**), octal (iniciada com **0o**) ou hexadecimal (iniciada com **0x**).
 - Lógico (ou Booleano): (**bool**)
 - 0 é **False** e 1 é **True**.
- Tipos de Ponto-Flutuante
 - Número de Ponto-Flutuante: (**float**)
 - Número Complexo: (**complex**)
 - Representado por um par de números de ponto-flutuante.

Tipos Básicos (Embutidos)

- Tipos básicos são imutáveis
- Tipos Integrais
 - Inteiro: (**int**)
 - Pode ter centenas de dígitos, limitado apenas pela memória do computador;
 - O padrão é decimal, mas pode-se usar outras bases como binária (iniciada com **0b**), octal (iniciada com **0o**) ou hexadecimal (iniciada com **0x**).
 - Lógico (ou Booleano): (**bool**)
 - 0 é **False** e 1 é **True**.
- Tipos de Ponto-Flutuante
 - Número de Ponto-Flutuante: (**float**)
 - Número Complexo: (**complex**)
 - Representado por um par de números de ponto-flutuante.
- String (**str**)
 - Representada por uma sequência de caracteres Unicode, iniciada e terminada por aspas simples ou duplas.

Conversão de Tipos

- String, booleano ou ponto-flutuante para inteiro: **int**
- String, booleano ou inteiro para ponto-flutuante: **float**
- String, inteiro ou ponto-flutuante para booleano: **bool**
- Booleano, inteiro ou ponto-flutuante para String: **str**

Comando de Atribuição

- Comando mais importante de uma linguagem imperativa.

Comando de Atribuição

- Comando mais importante de uma linguagem imperativa.
- Em um comando de atribuição, uma variável recebe o resultado da avaliação de uma expressão.

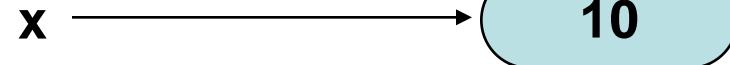
$\text{saldo} = \text{deposito} - \text{saque}$



Linguagem de Tipagem Dinâmica

Referência para

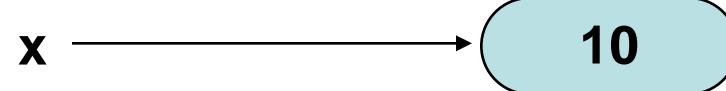
- $x = 10$



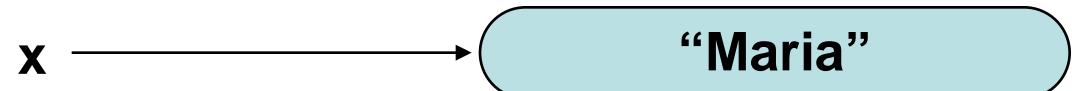
Linguagem de Tipagem Dinâmica

Referência para

- $x = 10$



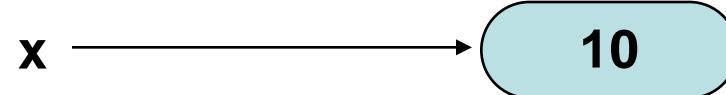
- $x = "Maria"$



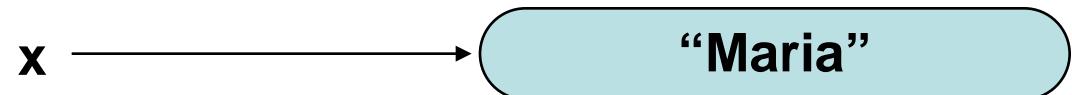
Linguagem de Tipagem Dinâmica

Referência para

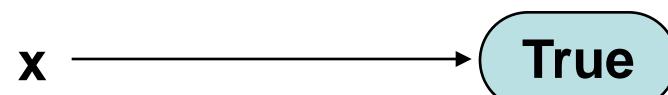
- $x = 10$



- $x = "Maria"$



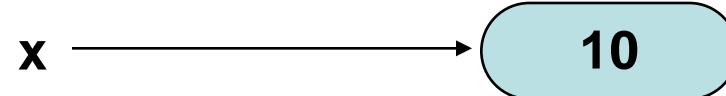
- $x = \text{True}$



Linguagem de Tipagem Dinâmica

Referência para

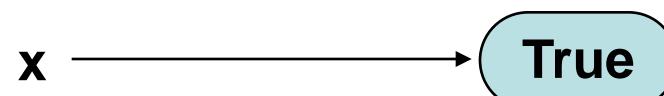
- $x = 10$



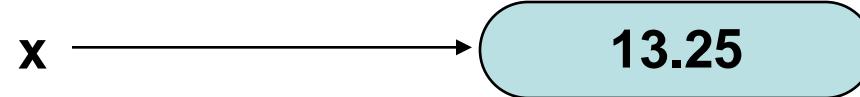
- $x = "Maria"$



- $x = \text{True}$



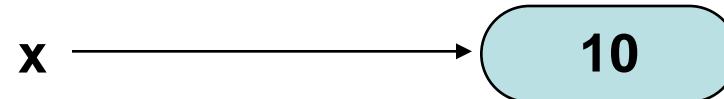
- $x = 13.25$



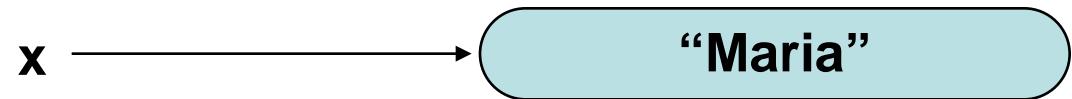
Linguagem de Tipagem Dinâmica

Referência para

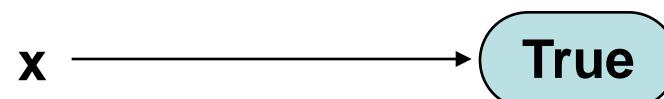
- $x = 10$



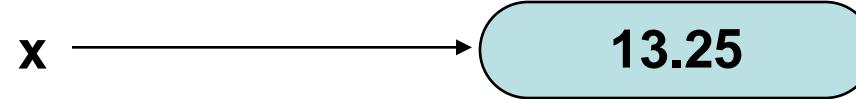
- $x = \text{"Maria"}$



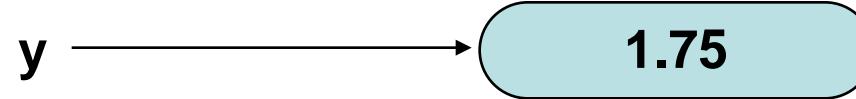
- $x = \text{True}$



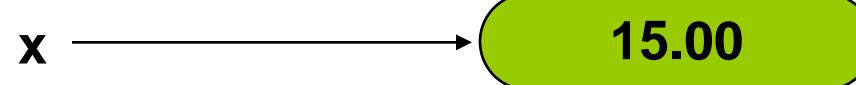
- $x = 13.25$



- $y = 1.75$

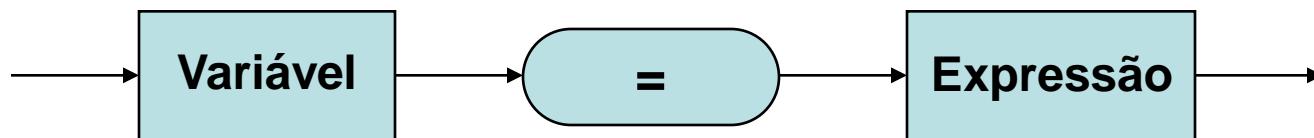


- $x = x+y$



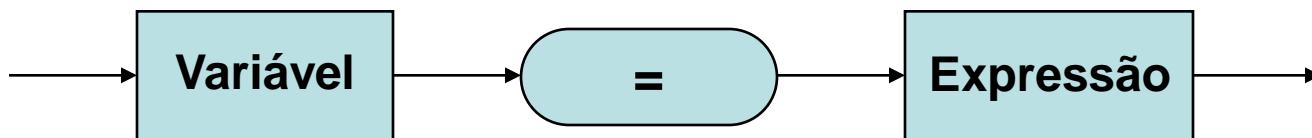
Diagramas Sintáticos

- Atribuição Simples:

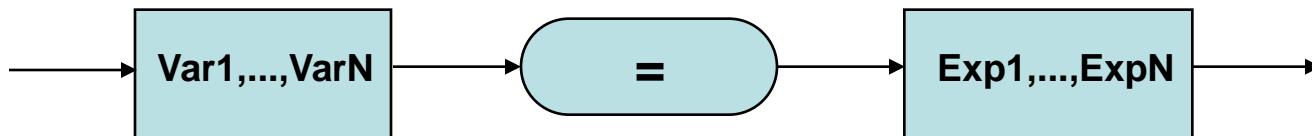


Diagramas Sintáticos

- Atribuição Simples:

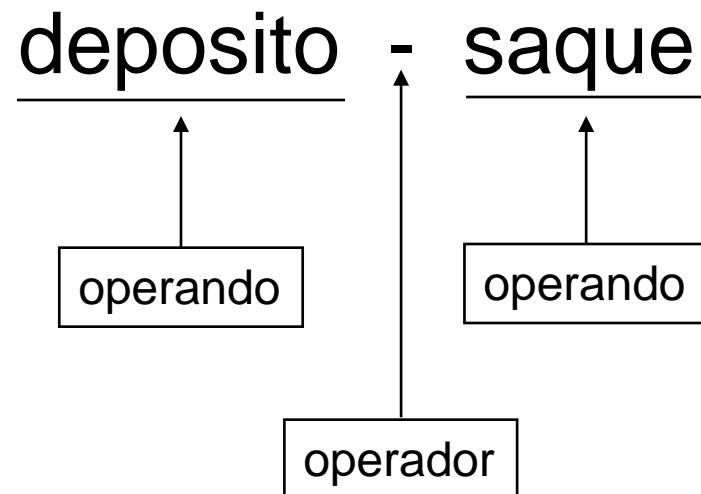


- Atribuição Múltipla:



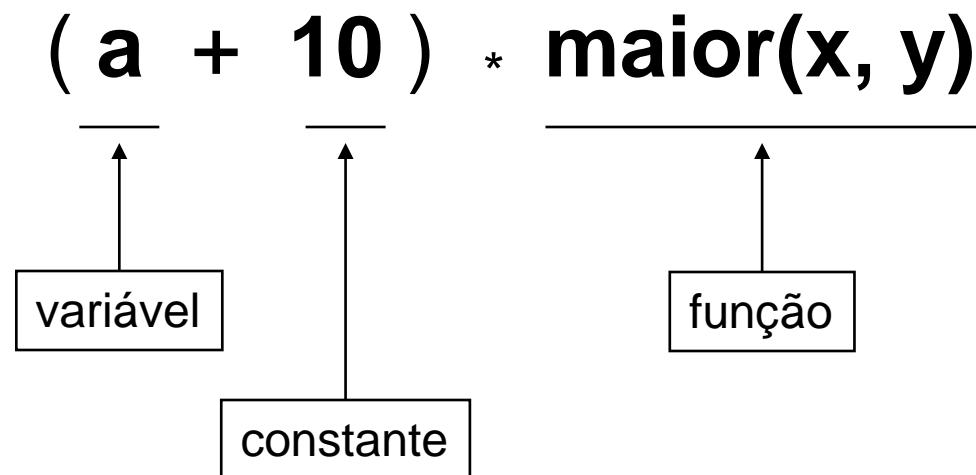
Expressão

- Uma expressão especifica o cálculo de um valor.
- É definida por operando(s) e operador(es).



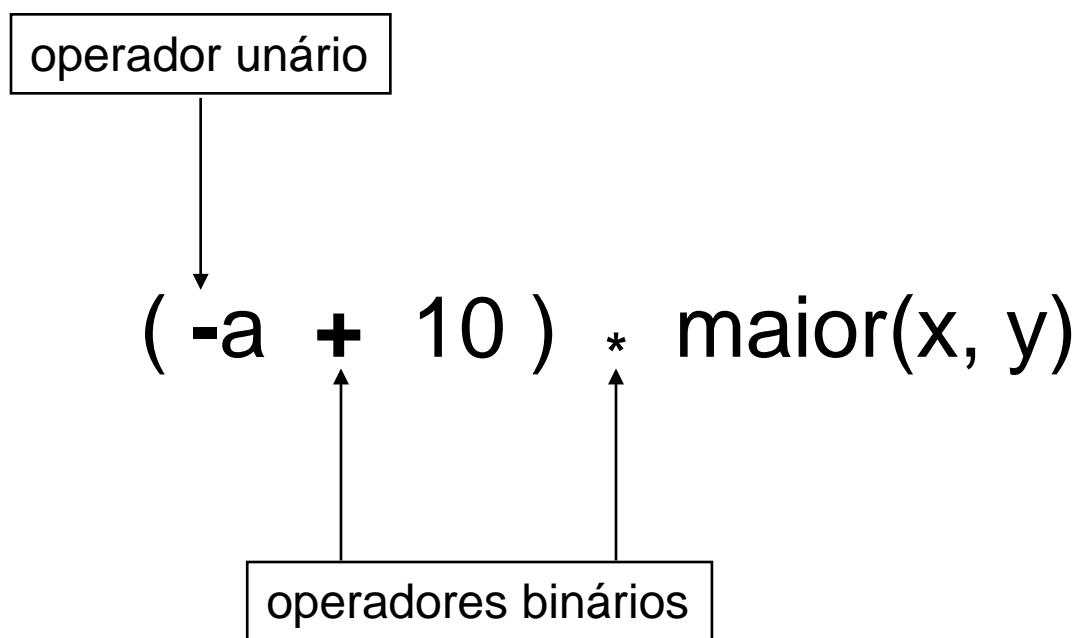
Operando

- Operando pode ser uma constante, uma variável ou um resultado de função.



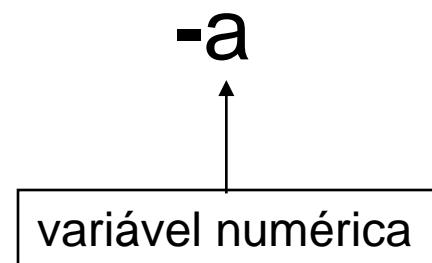
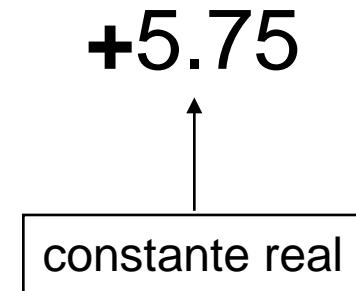
Operador

- Operador pode ser unário ou binário, dependendo se admite um operando ou dois operandos, respectivamente.



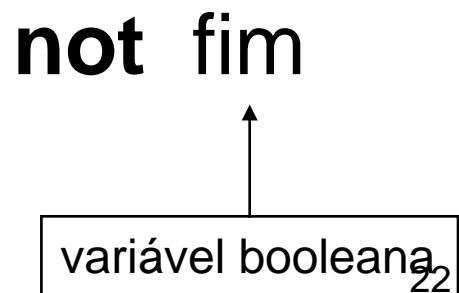
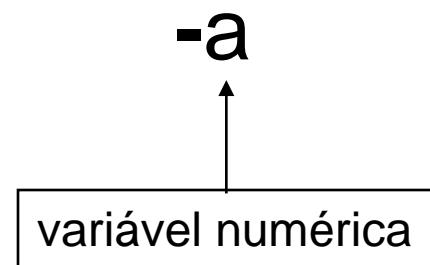
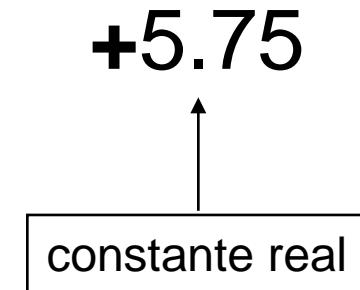
Operadores Unários

- Operador numérico positivo (+)
 - Operando deve ser numérico
- Operador numérico negativo (-)
 - Operando deve ser numérico



Operadores Unários

- Operador numérico positivo (+)
 - Operando deve ser numérico
- Operador numérico negativo (-)
 - Operando deve ser numérico
- Operador lógico negação (**not**)
 - Operando deve ser booleano



Operadores Binários Aritméticos

- Soma (+)
 - Subtração (-)
- } operadores aditivos

Operadores Binários Aritméticos

- Soma (+)
 - Subtração (-)
- }
- operadores
aditivos
-
- Produto (*)
 - Divisão de ponto flutuante (/)
 - Divisão inteira (//)
 - Resto da divisão inteira (%)
- }
- operadores
multiplicativos

Operadores Binários Aritméticos

- Soma (+)
 - Subtração (-)
- }
- operadores
aditivos
-
- Produto (*)
 - Divisão de ponto flutuante (/)
 - Divisão inteira (//)
 - Resto da divisão inteira (%)
- }
- operadores
multiplicativos
-
- Potenciação (**)

Operadores Binários Aritméticos

- Soma (+)
 - Subtração (-)
- }
- operadores
aditivos
-
- Produto (*)
 - Divisão de ponto flutuante (/)
 - Divisão inteira (//)
 - Resto da divisão inteira (%)
- }
- operadores
multiplicativos
-
- Potenciação (**)

$$(5.75 + (a \% b) - 7) / 8.1$$

26

Operadores Binários Lógicos

- Disjunção lógica ou soma lógica (**or**)

Operadores Binários Lógicos

- Disjunção lógica ou soma lógica (**or**)
- Conjunção lógica ou produto lógico (**and**)

Operadores Binários Lógicos

- Disjunção lógica ou soma lógica (**or**)
- Conjunção lógica ou produto lógico (**and**)

(p or q) and r

Operadores Binários Relacionais

- Igual a (`==`)
- Diferente de (`!=`)

O resultado de uma operação relacional é um valor booleano.

Operadores Binários Relacionais

- Igual a (**$==$**)
- Diferente de (**$!=$**)

O resultado de uma operação relacional é um valor booleano.

$(2 + 2) == 5$ é falso!

$a == 5$

$x != y$

Operadores Binários Relacionais

- Igual a ($==$)
- Diferente de ($!=$)
- Maior que ($>$)
- Menor que ($<$)
- Maior ou igual a (\geq)
- Menor ou igual a (\leq)

O resultado de uma operação relacional é um valor booleano.

$(2 + 2) == 5$ é falso!

$(2 + 2) \leq 5$ é verdadeiro!

$a == 5$

$x != y$

$a \leq 6 + c$

Precedência dos Operadores

1. Expressões entre Parênteses () – Maior Prioridade;
2. Potenciação (**);
3. Unários (+, -);
4. Binários Multiplicativos (*, /, %, //);
5. Binários Aditivos (+, -);
6. Relacionais (==, !=, <, >, <=, >=);
7. Lógico **not**;
8. Lógico **and**;
9. Lógico **or** – Menor Prioridade.

Precedência dos Operadores

1. Expressões entre Parênteses () – Maior Prioridade;
2. Potenciação (**);
3. Unários (+, -);
4. Binários Multiplicativos (*, /, %, //);
5. Binários Aditivos (+, -);
6. Relacionais (==, !=, <, >, <=, >=);
7. Lógico **not**;
8. Lógico **and**;
9. Lógico **or** – Menor Prioridade.

$5.75 + a\%b - 7/8.1$ equivale a $5.75 + (a\%b) - (7/8.1)$

Precedência dos Operadores

$5 * a \% b / 8.1$

equivale a

$((5 * a) \% b) / 8.1$

Comandos de Saída Padrão

- **print()**
 - Pula para a próxima linha na saída padrão (vídeo).

Comandos de Saída Padrão

- **print()**
 - Pula para a próxima linha na saída padrão (vídeo).
- **print(expressão)**
 - Escreve na saída padrão (vídeo) o resultado da avaliação da expressão;
 - Ao final, pula para a próxima linha.



Comandos de Saída Padrão

Comandos de Saída Padrão

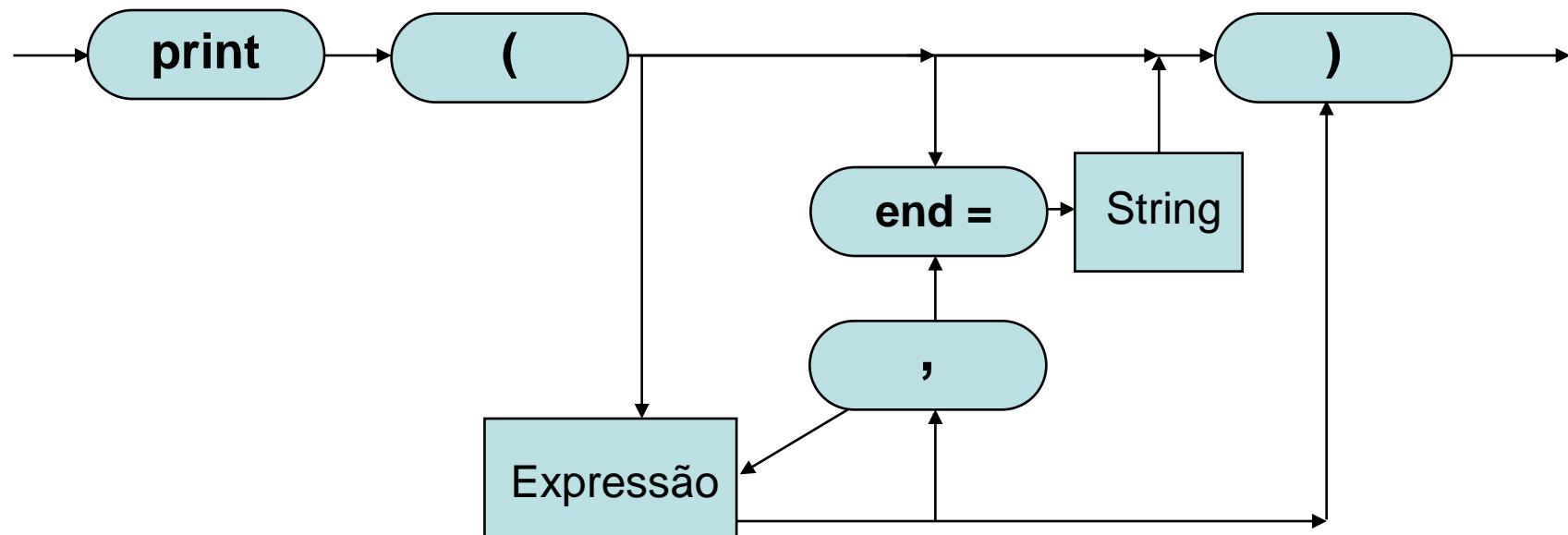
- **print(expressão, end = *término*)**
 - Escreve na saída padrão (vídeo) o resultado da avaliação da expressão;
 - Ao final, escreve a String de *término*.

Comandos de Saída Padrão

- **print(expressão, end = término)**
 - Escreve na saída padrão (vídeo) o resultado da avaliação da expressão;
 - Ao final, escreve a String de *término*.

- **print(exp1, exp2, ..., expN, end = término)**
 - Escreve na saída padrão (vídeo) o resultado da avaliação de cada expressão *expNum*;
 - Um espaço em branco é escrito entre cada par de *expNum*;
 - Ao final, escreve a String de *término*.

Diagrama Sintático do Comando *print*



Expressões Formatadas (operador %(...))

- Sintaxe da expressão formatada:

$$texto = "... \%formato1 \dots \%formatoN..." \%(exp1, \dots, expN)$$

- Formatos mais comuns:

d (inteiro), **f** (número com ponto flutuante) e **s** (String)

Expressões Formatadas (operador %(...))

- Sintaxe da expressão formatada:

$$texto = "...%formato1 ... %formatoN..." \%(exp1, \dots, expN)$$

- Formatos mais comuns:

d (inteiro), **f** (número com ponto flutuante) e **s** (String)

- Exemplos:

```
-msg = "A média dos números é %4.2f" % (82432.923421)  
print(msg)
```

- Escreve “A média dos números é 82432.92” e pula de linha

Expressões Formatadas (operador %(...))

- Sintaxe da expressão formatada:

texto = "...%formato1 ... %formatoN..." %(exp1, ..., expN)

- Formatos mais comuns:

d (inteiro), **f** (número com ponto flutuante) e **s** (String)

- Exemplos:

-msg = “A média dos números é %4.2f” % (82432.923421)
print(msg)

- Escreve “A média dos números é 82432.92” e pula de linha

-print("%f + %f = %4.1f" % (15,7.8313, 15+7.8313))
• Escreve “15.000000 + 7.831300 = 22.8” e pula de linha

Expressões Formatadas (operador %(...))

- Sintaxe da expressão formatada:

texto = "...%formato1 ... %formatoN..." %(exp1, ..., expN)

- Formatos mais comuns:

d (inteiro), **f** (número com ponto flutuante) e **s** (String)

- Exemplos:

-msg = “A média dos números é %4.2f” % (82432.923421)
print(msg)

- Escreve “A média dos números é 82432.92” e pula de linha

-print(“%f + %f = %4.1f” % (15,7.8313, 15+7.8313))

- Escreve “15.000000 + 7.831300 = 22.8” e pula de linha

-print(“%d + %d = %d” % (5.89, 7.83, 5.89+7.83), **end** = “!!!”)

- Escreve “5 + 7 = 13!!!” (apenas a parte inteira) e não pula de linha

Comandos de Entrada Padrão

- **input()**

- Comando que aguarda o usuário fornecer, pela entrada padrão (teclado), um valor expresso por uma sequência de caracteres, e o retorna.
- Este comando tem o efeito de suspender a execução do programa até que o usuário escreva sua entrada e pressione a tecla <enter>.
- Exemplo:

```
aluno = input()
```

Comandos de Entrada Padrão

- **input()**

- Comando que aguarda o usuário fornecer, pela entrada padrão (teclado), um valor expresso por uma sequência de caracteres, e o retorna.
- Este comando tem o efeito de suspender a execução do programa até que o usuário escreva sua entrada e pressione a tecla <enter>.
- Exemplo:

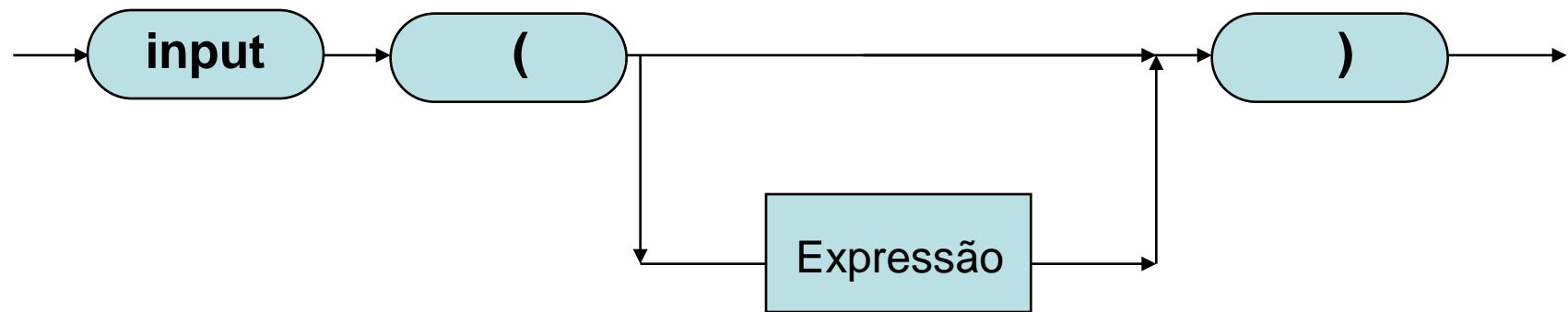
```
aluno = input()
```

- **input(*mensagem*)**

- Comando que escreve a expressão *mensagem* na saída padrão (vídeo) e aguarda, via interrupção, que o usuário escreva sua resposta, composta de uma sequência de caracteres digitados, e pressione a tecla <enter>.
- Exemplo:

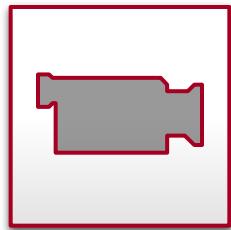
```
aluno = input("Digite o nome do aluno: ")
```

Diagrama Sintático do Comando *input*



Exemplos de Aplicação dos Conteúdos Vistos

Clique no botão para assistir ao tutorial:



Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:





Aula 2

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo Apresentado:

- Variáveis
- Tipos
- Comandos Básicos



Aula 3

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Controle
 - Sequência
 - Estruturas de Seleção
 - Estruturas de Repetição



Sequência de Comandos

Em Python, para construirmos uma sequência de comandos, devemos colocá-los um por linha, um abaixo do outro, mantendo-os na mesma tabulação (na mesma indentação).

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Bloco de Comandos (Suite)

Em Python, um bloco de comandos contendo um ou mais comandos é chamado de suite. No exemplo abaixo, temos uma suite com cinco comandos em sequência.

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Comentários

- Comentários simples são iniciados com o caractere #
- Comentários com múltiplas linhas podem ser delimitados em seu início e término por:
 - Três aspas simples, isto é: ““ e ””
 - Três aspas duplas, isto é: ““““ e ”””” (os docStrings)

““

Programa que lê do teclado um par de números inteiros e escreve na tela a soma e o produto dos números lidos.

””

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Comentários

- Comentários simples são iniciados com o caractere #
- Comentários com múltiplas linhas podem ser delimitados em seu início e término por:
 - Três aspas simples, isto é: ““ e ””
 - Três aspas duplas, isto é: “““ e ””” (os docStrings)

```
““
```

Programa que lê do teclado um par de números inteiros e escreve na tela a soma e o produto dos números lidos.

```
””
```

```
x = int(input("Digite o primeiro valor: "))      # Lê o primeiro número
y = int(input("Digite o segundo valor: "))        # Lê o segundo número
soma = x + y                                      # Calcula a soma
prod = x * y                                      # Calcula o produto
print("A Soma =", soma, "e o Produto =", prod)  # Escreve resultados
```



Execução do Programa

Exemplo:

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Execução do Programa

Exemplo:

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Execução:

```
Digite o primeiro valor: 2 <enter>
```

Execução do Programa

Exemplo:

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Execução:

```
Digite o primeiro valor: 2 <enter>
Digite o segundo valor: 13 <enter>
```

Execução do Programa

Exemplo:

```
x = int(input("Digite o primeiro valor: "))
y = int(input("Digite o segundo valor: "))
soma = x + y
prod = x * y
print("A Soma =", soma, "e o Produto =", prod)
```

Execução:

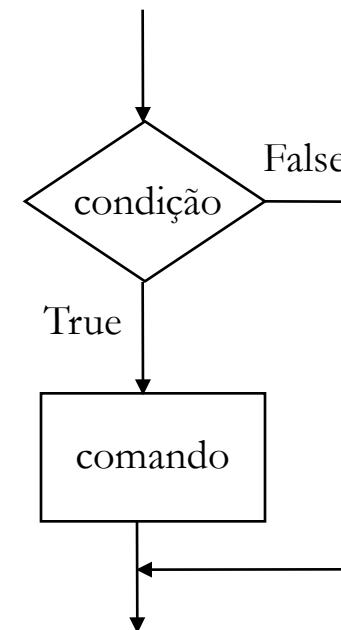
```
Digite o primeiro valor: 2 <enter>
Digite o segundo valor: 13 <enter>
A Soma = 15 e o Produto = 26 <enter>
```

Estrutura de Seleção com Um Ramo (*if*)

A estrutura de seleção com um ramo **if** é utilizada quando se deseja executar um comando (ou uma suite) apenas no caso de uma condição ser satisfeita.

```
if <condição>:  
    <comando>
```

<condição> representa uma expressão booleana, ou seja, expressão cujo resultado é verdadeiro (True) ou falso (False).



Estrutura de Seleção com Um Ramo (*if*)

Exemplo:

```
valor = float(input("Entre com um valor: "))
if valor>0:
    print(valor, "é maior do que zero.")
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplo:

```
valor = float(input("Entre com um valor: "))
if valor>0:
    print(valor, "é maior do que zero.")
```

Execução:

```
Entre com um valor: 4.7 <enter>
4.7 é maior do que zero. <enter>
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplo:

```
salario = float(input("Diga seu salário atual: "))
if salario>10000:
    inps = salario * 0.10
    impostoRenda = salario * 0.15
    print("Valor do INPS:", inps, "e do Imposto de Renda:", impostoRenda)
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplo:

```
salario = float(input("Diga seu salário atual: "))
if salario>10000:
    inps = salario * 0.10
    impostoRenda = salario * 0.15
    print("Valor do INPS:", inps, "e do Imposto de Renda:", impostoRenda)
```

Execução:

```
Diga seu salário atual: 11000 <enter>
Valor do INPS: 1100.0 e do Imposto de Renda: 1650.0 <enter>
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplos de condições compostas:

```
if (valor >= 10) and (valor <= 20):  
    print(valor, "ocorre dentro do intervalo [10,20]")
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplos de condições compostas:

```
if (valor >= 10) and (valor <= 20):
    print(valor, "ocorre dentro do intervalo [10,20]")
```

```
if 10<=valor<= 20:
    print(valor, "ocorre dentro do intervalo [10,20]")
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplos de condições compostas:

```
if (valor >= 10) and (valor <= 20):  
    print(valor, "ocorre dentro do intervalo [10,20]")
```

```
if 10<=valor<= 20:  
    print(valor, "ocorre dentro do intervalo [10,20]")
```

```
if (valor < 0) or (valor > 30):  
    print(valor, "ocorre fora do intervalo [0,30]")
```

Estrutura de Seleção com Um Ramo (*if*)

Exemplos de condições compostas:

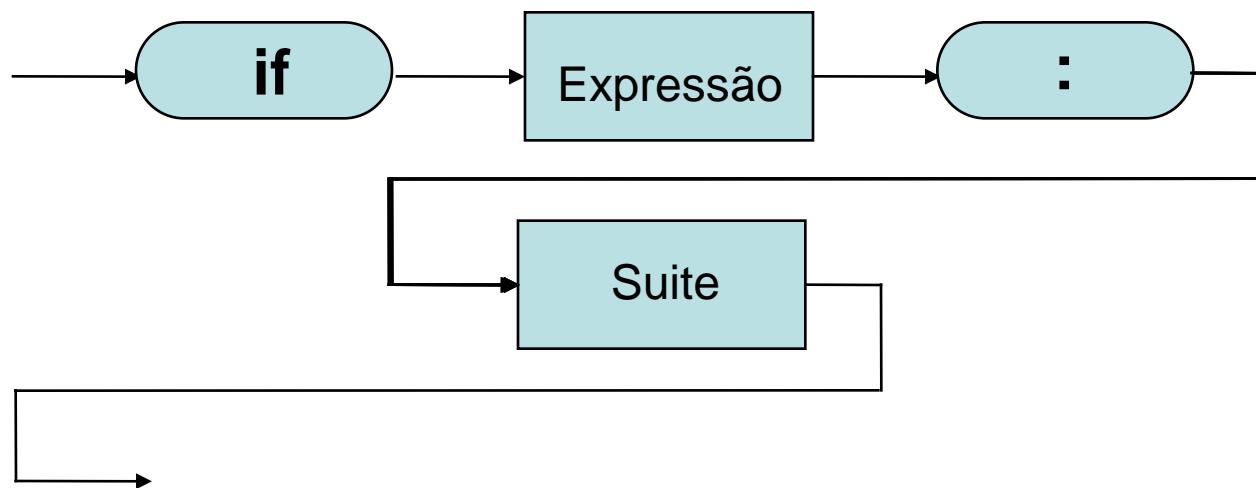
```
if (valor >= 10) and (valor <= 20):  
    print(valor, "ocorre dentro do intervalo [10,20]")
```

```
if 10<=valor<= 20:  
    print(valor, "ocorre dentro do intervalo [10,20]")
```

```
if (valor < 0) or (valor > 30):  
    print(valor, "ocorre fora do intervalo [0,30]")
```

```
if ((valor >= 0) and (valor < 10)) or ((valor > 20) and (valor <= 30)):  
    print(valor, "ocorre dentro do intervalo [0,10) ou em (20,30]")
```

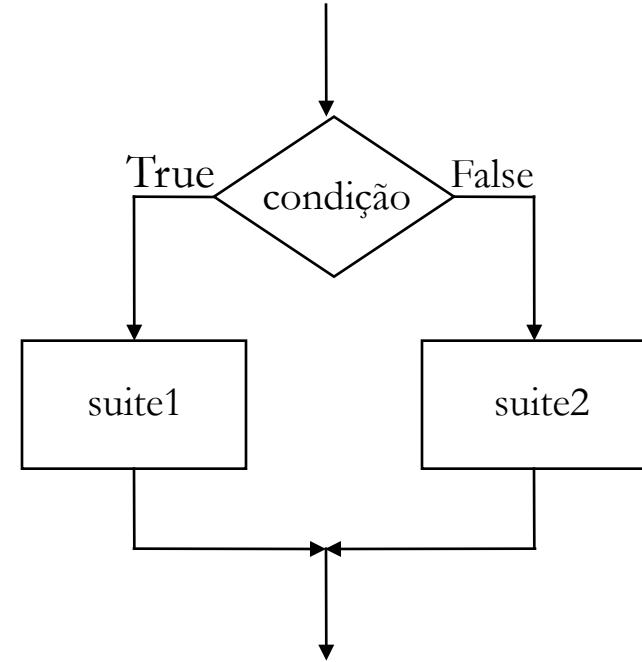
Diagrama Sintático da Seleção *if*



Estrutura de Seleção com Dois Ramos (**if-else**)

A estrutura de seleção **if-else** é utilizada quando se deseja executar uma entre duas suites, dependendo do resultado da avaliação de uma condição.

```
if <condição>:  
    <suite1>  
  
else:  
    <suite2>
```



Estrutura de Seleção com Dois Ramos (*if-else*)

Exemplo:

```
valor = float(input("Entre com um valor:"))
if valor>0:
    print(valor, "é maior do que zero.")
else:
    print(valor, "é menor ou igual a zero.")
```

Estrutura de Seleção com Dois Ramos (*if-else*)

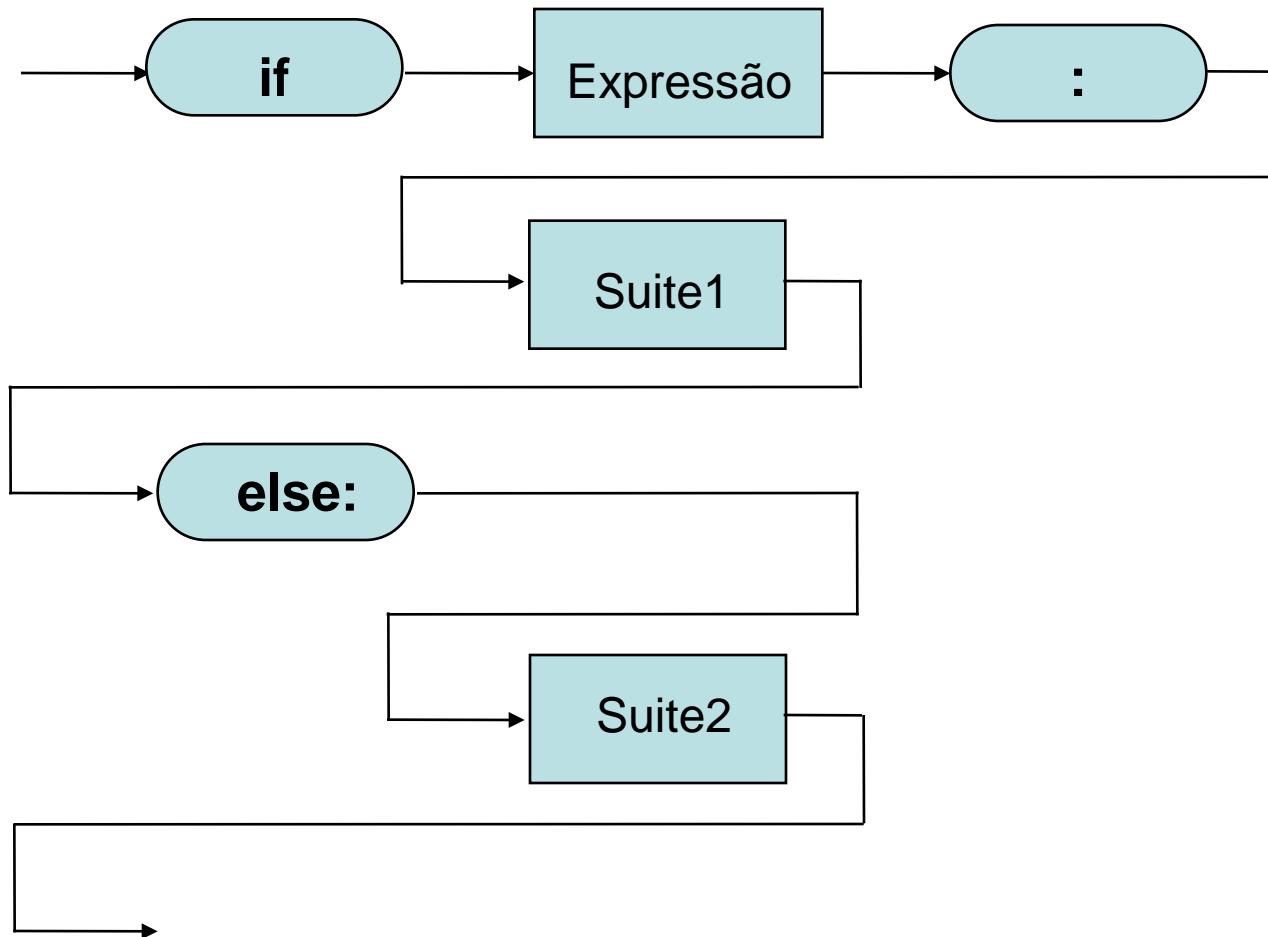
Exemplo:

```
valor = float(input("Entre com um valor:"))
if valor>0:
    print(valor, "é maior do que zero.")
else:
    print(valor, "é menor ou igual a zero.")
```

Execução:

```
Entre com um valor: -13.22 <enter>
-13.22 é menor ou igual a zero.<enter>
```

Diagrama Sintático de Seleção *if-else*



Estrutura de Seleção Aninhadas

O programa a seguir calcula o abono salarial que uma empresa concederá aos seus funcionários com mais de um ano de tempo de serviço. Os que têm menos de dez anos ganharão abono de 10%. Os demais ganharão de 25%.

```
"""
Abono Salarial
"""

salario = float(input("Diga seu salário atual: "))
tempo = int(input("Diga quantos anos completos tem de serviço: "))
if tempo<1:
    print("Seu salário se mantém o mesmo:", salario)
else:
    if tempo<10:
        salario = salario * 1.10
    else:
        salario = salario * 1.25
    print("Seu novo salário, com abono:", salario)
```

Estrutura de Seleção Aninhadas

O programa a seguir calcula o abono salarial que uma empresa concederá aos seus funcionários com mais de um ano de tempo de serviço. Os que têm menos de dez anos ganharão abono de 10%. Os demais ganharão de 25%.

```
"""
Abono Salarial
"""

salario = float(input("Diga seu salário atual: "))
tempo = int(input("Diga quantos anos completos tem de serviço: "))
if tempo<1:
    print("Seu salário se mantém o mesmo:", salario)
else:
    if tempo<10:
        salario = salario * 1.10
    else:
        salario = salario * 1.25
    print("Seu novo salário, com abono:", salario)
```

Estrutura de Seleção Aninhadas

O programa a seguir calcula o abono salarial que uma empresa concederá aos seus funcionários com mais de um ano de tempo de serviço. Os que têm menos de dez anos ganharão abono de 10%. Os demais ganharão de 25%.

```
""
```

Abono Salarial

```
""
```

```
salario = float(input("Diga seu salário atual: "))
tempo = int(input("Diga quantos anos completos tem de serviço: "))
if tempo<1:
    print("Seu salário se mantém o mesmo:", salario)
else:
    if tempo<10:
        salario = salario * 1.10
    else:
        salario = salario * 1.25
    print("Seu novo salário, com abono:", salario)
```

Estrutura de Seleção com Múltiplos Ramos (if-elif ou if-elif-else)

As estruturas de seleção **if-elif** e **if-elif-else** são utilizadas quando se deseja executar uma entre várias suites, dependendo dos valores das expressões.

```
if <expressao1>:  
    <suite1>  
elif <expressao2>:  
    <suite2>  
    ...  
elif <expressaoN>:  
    <suiteN>
```

Estrutura de Seleção com Múltiplos Ramos (if-elif ou if-elif-else)

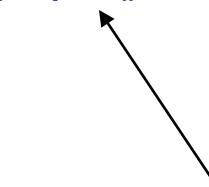
As estruturas de seleção **if-elif** e **if-elif-else** são utilizadas quando se deseja executar uma entre várias suites, dependendo dos valores das expressões.

```
if <expressao1>:  
    <suite1>  
elif <expressao2>:  
    <suite2>  
...  
elif <expressaoN>:  
    <suiteN>
```

```
if <expressao1>:  
    <suite1>  
elif <expressao2>:  
    <suite2>  
...  
elif <expressaoN>:  
    <suiteN>  
else:  
    <suiteCasoContrário>
```

Estrutura de Seleção com Múltiplos Ramos

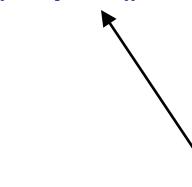
```
valores = input("Entre com dois inteiros positivos: ").split()  
x = int(valores[0])      # primeira substring da lista  
y = int(valores[1])      # segunda substring da lista  
op = input("Informe o operador (+, -, *, / ou **): ")  
if op=="+":  
    resultado = x + y  
elif op=="-":  
    resultado = x - y  
elif op=="*":  
    resultado = x * y  
elif op=="/":  
    resultado = x / y  
elif op=="**":  
    resultado = x ** y  
print(x, op, y, "=", resultado)
```



Operação a ser vista que divide a string, considerando o espaço em branco como separador, em uma lista de substrings.

Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()  
x = int(valores[0])      # primeira substring da lista  
y = int(valores[1])      # segunda substring da lista  
op = input("Informe o operador (+, -, *, / ou **): ")  
if op=="+":  
    resultado = x + y  
elif op=="-":  
    resultado = x - y  
elif op=="*":  
    resultado = x * y  
elif op=="/":  
    resultado = x / y  
elif op=="**":  
    resultado = x ** y  
print(x, op, y, "=", resultado)
```



Operação a ser vista que divide a string, considerando o espaço em branco como separador, em uma lista de substrings.

Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()  
x = int(valores[0])      # primeira substring da lista  
y = int(valores[1])      # segunda substring da lista  
op = input("Informe o operador (+, -, *, / ou **): ")  
if op=="+":  
    resultado = x + y  
elif op=="-":  
    resultado = x - y  
elif op=="*":  
    resultado = x * y  
elif op=="/":  
    resultado = x / y  
elif op=="**":  
    resultado = x ** y  
print(x, op, y, "=", resultado)
```

Operação a ser vista que divide a string, considerando o espaço em branco como separador, em uma lista de substrings.

Execução:

Entre com dois inteiros positivos: 2 9 <enter>

Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()  
x = int(valores[0])      # primeira substring da lista  
y = int(valores[1])      # segunda substring da lista  
op = input("Informe o operador (+, -, *, / ou **): ")  
  
if op=="+":  
    resultado = x + y  
elif op=="-":  
    resultado = x - y  
elif op=="*":  
    resultado = x * y  
elif op=="/":  
    resultado = x / y  
elif op=="**":  
    resultado = x ** y  
  
print(x, op, y, "=", resultado)
```

Operação a ser vista que divide a string, considerando o espaço em branco como separador, em uma lista de substrings.

Execução:

```
Entre com dois inteiros positivos: 2 9 <enter>  
Informe o operador (+, -, *, / ou **): ** <enter>
```

Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()  
x = int(valores[0])      # primeira substring da lista  
y = int(valores[1])      # segunda substring da lista  
op = input("Informe o operador (+, -, *, / ou **): ")  
if op=="+":  
    resultado = x + y  
elif op=="-":  
    resultado = x - y  
elif op=="*":  
    resultado = x * y  
elif op=="/":  
    resultado = x / y  
elif op=="**":  
    resultado = x ** y  
print(x, op, y, "=", resultado)
```

Operação a ser vista que divide a string, considerando o espaço em branco como separador, em uma lista de substrings.

Execução:

```
Entre com dois inteiros positivos: 2 9 <enter>  
Informe o operador (+, -, *, / ou **): ** <enter>  
2 ** 9 = 512 <enter>
```

33

Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()
x = int(valores[0])
y = int(valores[1])
op = input("Informe o operador (+, -, *, / ou **): ")
if op=="+":
    resultado = x + y
elif op=="-":
    resultado = x - y
elif op=="*":
    resultado = x * y
elif op=="/":
    resultado = x / y
elif op=="**":
    resultado = x ** y
else:
    resultado = None
if resultado == None:
    print(op, ": Operador inexistente!!")
else:
    print(x, op, y, "=", resultado)
```

Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()
x = int(valores[0])
y = int(valores[1])
op = input("Informe o operador (+, -, *, / ou **): ")
if op=="+":
    resultado = x + y
elif op=="-":
    resultado = x - y
elif op=="*":
    resultado = x * y
elif op=="/":
    resultado = x / y
elif op=="**":
    resultado = x ** y
else:
    resultado = None
if resultado == None:
    print(op, ": Operador inexistente!!")
else:
    print(x, op, y, "=", resultado)
```

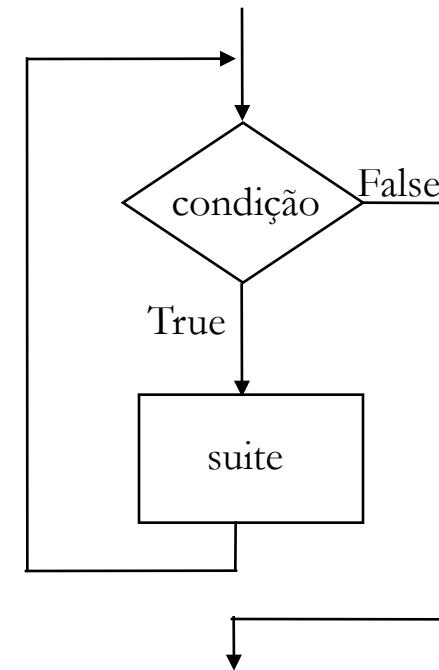
Estrutura de Seleção com Múltiplos Ramos

```
valores = input("Entre com dois inteiros positivos: ").split()
x = int(valores[0])
y = int(valores[1])
op = input("Informe o operador (+, -, *, / ou **): ")
if op=="+":
    resultado = x + y
elif op=="-":
    resultado = x - y
elif op=="*":
    resultado = x * y
elif op=="/":
    resultado = x / y
elif op=="**":
    resultado = x ** y
else:
    resultado = None
if resultado == None:
    print(op, ": Operador inexistente!!")
else:
    print(x, op, y, "=", resultado)
```

Estrutura de Repetição Indefinida (**while**)

A estrutura de repetição **while** é utilizada quando se deseja executar uma suite enquanto uma condição for verdadeira (zero ou mais vezes).

```
while <condição>:  
    <suite>
```



Estrutura de Repetição Indefinida (*while*)

Exemplo:

```
indice = 1          # inicializa variável
while indice <= 10:    # testa condição
    print(indice, end=" ")
    indice = indice + 1  # incrementa a variável que controla a repetição
    print()               # pula linha no vídeo
```

Estrutura de Repetição Indefinida (*while*)

Exemplo:

```
indice = 1          # inicializa variável
while indice <= 10:    # testa condição
    print(indice, end=" ")
    indice = indice + 1  # incrementa a variável que controla a repetição
print()              # pula linha no vídeo
```

Execução:

```
1 2 3 4 5 6 7 8 9 10 <enter>
```

Outro Exemplo: Cálculo do Fatorial

$N! = N * (N-1) * (N-2) * \dots * 2 * 1,$ para $N > 0.$
 $N! = 1,$ para $N = 0.$

```
num = int(input("Digite um valor inteiro e positivo: "))  
i = 1  
fat = 1  
while i <= num:  
    fat = fat * i  
    i = i + 1  
print("O fatorial de", num, "=", fat)
```

Outro Exemplo: Cálculo do Fatorial

$$N! = N * (N-1) * (N-2) * \dots * 2 * 1, \quad \text{para } N > 0.$$
$$N! = 1, \quad \text{para } N = 0.$$

```
num = int(input("Digite um valor inteiro e positivo: "))

i = 1
fat = 1

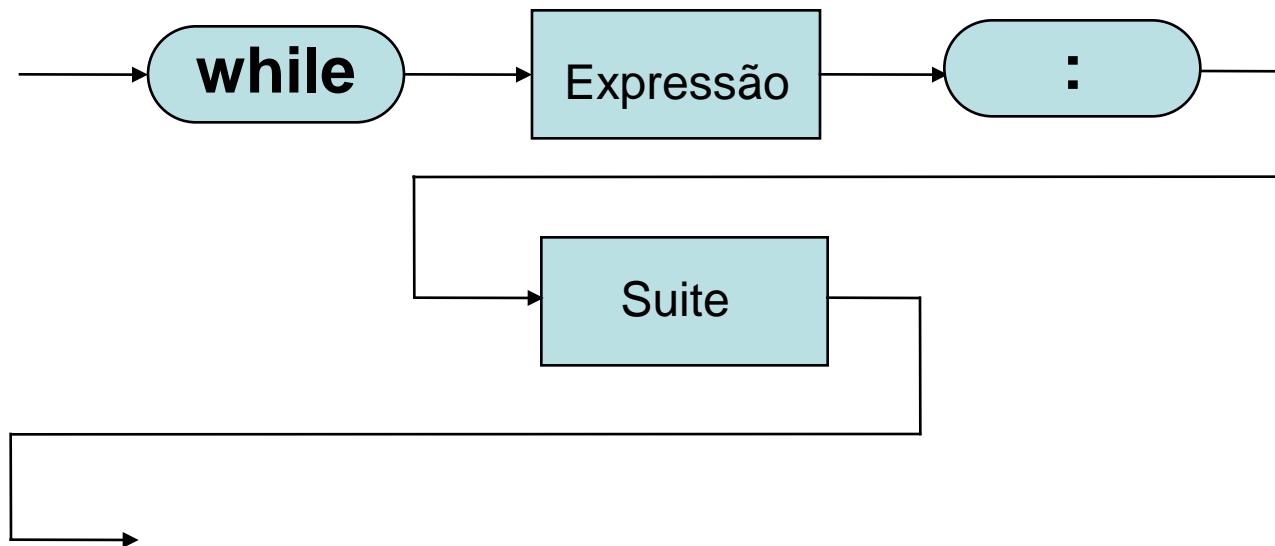
while i <= num:
    fat = fat * i
    i = i + 1

print("O fatorial de", num, "=", fat)
```

Execução:

```
Digite valor inteiro e positivo: 5 <enter>
O fatorial de 5 = 120 <enter>
```

Diagrama Sintático de Repetição while



Estrutura de Repetição Definida (**for**)

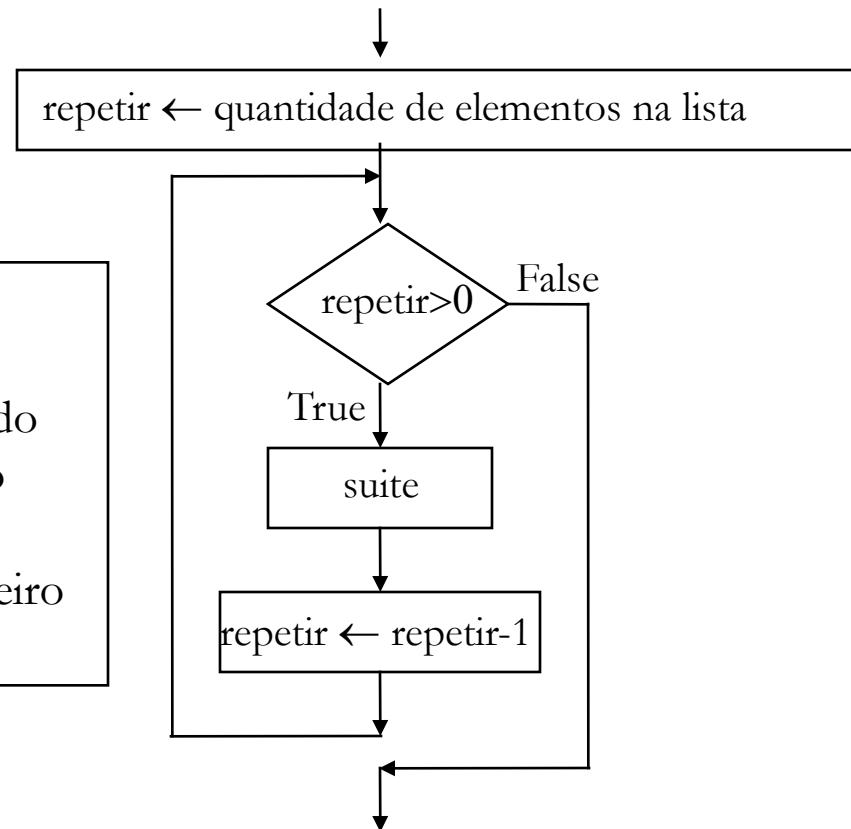
A estrutura de repetição **for** é utilizada quando se deseja executar uma mesma suite um número específico de vezes, enumerado em uma lista de valores.

```
for <var> in <lista de valores>:  
    <suite>
```

Estrutura de Repetição Definida (for)

```
for <var> in <lista>:  
    <suite>
```

A cada ciclo da repetição, `<var>` assume o valor referenciado pela posição do elemento na lista. Isto é, o valor referenciado progressivamente do primeiro até o último elemento.



Estrutura de Repetição Definida (*for*)

Exemplo:

```
for item in [3,4,5,6,7,8,9]: # item assume progressivamente os valores na lista
    print(item, end=" ")      # cada valor de item é escrito na tela, separado por um espaço
    print()                   # pula linha (imprime "\n")
```

Estrutura de Repetição Definida (*for*)

Exemplo:

```
for item in [3,4,5,6,7,8,9]: # item assume progressivamente os valores na lista
    print(item, end=" ")      # cada valor de item é escrito na tela, separado por um espaço
print()                      # pula linha (imprime "\n")
```

Execução:

```
3 4 5 6 7 8 9 <enter>
```

Listas de Valores para uma Repetição **for**

- Para criar uma lista com uma progressão aritmética de elementos, bastante comum em repetições **for**, podemos utilizar a função range

Listas de Valores para uma Repetição **for**

- Para criar uma lista com uma progressão aritmética de elementos, bastante comum em repetições **for**, podemos utilizar a função `range`
- **range(*valor limite*)**
Cria uma lista com progressão aritmética de itens de razão 1, iniciada pelo valor 0 e terminada no valor que antecede o *limite*
 - Exemplo: `range(5)` cria a lista [0, 1, 2, 3, 4]

Listas de Valores para uma Repetição **for**

- Para criar uma lista com uma progressão aritmética de elementos, bastante comum em repetições **for**, podemos utilizar a função **range**
- **range(*valor limite*)**
Cria uma lista com progressão aritmética de itens de razão 1, iniciada pelo valor 0 e terminada no valor que antecede o *limite*
 - Exemplo: **range(5)** cria a lista [0, 1, 2, 3, 4]
- **range(*valor inicial*, *valor limite*)**
Cria uma lista com progressão aritmética de itens de razão 1, iniciada pelo *valor inicial* e terminada no valor que antecede o *limite*
 - Exemplo: **range(8, 13)** cria a lista [8, 9, 10, 11, 12]

Listas de Valores para uma Repetição **for**

- Para criar uma lista com uma progressão aritmética de elementos, bastante comum em repetições **for**, podemos utilizar a função **range**
- **range(*valor limite*)**
Cria uma lista com progressão aritmética de itens de razão 1, iniciada pelo valor 0 e terminada no valor que antecede o *limite*
 - Exemplo: **range(5)** cria a lista [0, 1, 2, 3, 4]
- **range(*valor inicial*, *valor limite*)**
Cria uma lista com progressão aritmética de itens de razão 1, iniciada pelo *valor inicial* e terminada no valor que antecede o *limite*
 - Exemplo: **range(8, 13)** cria a lista [8, 9, 10, 11, 12]
- **range(*valor inicial*, *valor limite*, *avanço*)**
Cria uma lista com progressão aritmética de itens de razão *avanço*, iniciada pelo *valor inicial* e terminada no valor que antecede o *limite*
 - Exemplo: **range(1, 30, 5)** cria a lista [1, 6, 11, 16, 21, 26]
 - Exemplo: **range(5, -14, -3)** cria a lista [5, 2, -1, -4, -7, -10, -13]

Outro Exemplo: Cálculo do Fatorial

$N! = N * (N-1) * (N-2) * \dots * 2 * 1,$ para $N > 0.$
 $N! = 1,$ para $N = 0.$

```
num = int(input("Digite valor inteiro e positivo: "))  
fat = 1  
for i in range(1,num+1):  
    fat = fat * i  
print("O fatorial de", num, "=", fat)
```

Outro Exemplo: Cálculo do Fatorial

$$\begin{aligned} N! &= N * (N-1) * (N-2) * \dots * 2 * 1, && \text{para } N > 0. \\ N! &= 1, && \text{para } N = 0. \end{aligned}$$

```
num = int(input("Digite valor inteiro e positivo: "))

fat = 1

for i in range(1,num+1):
    fat = fat * i

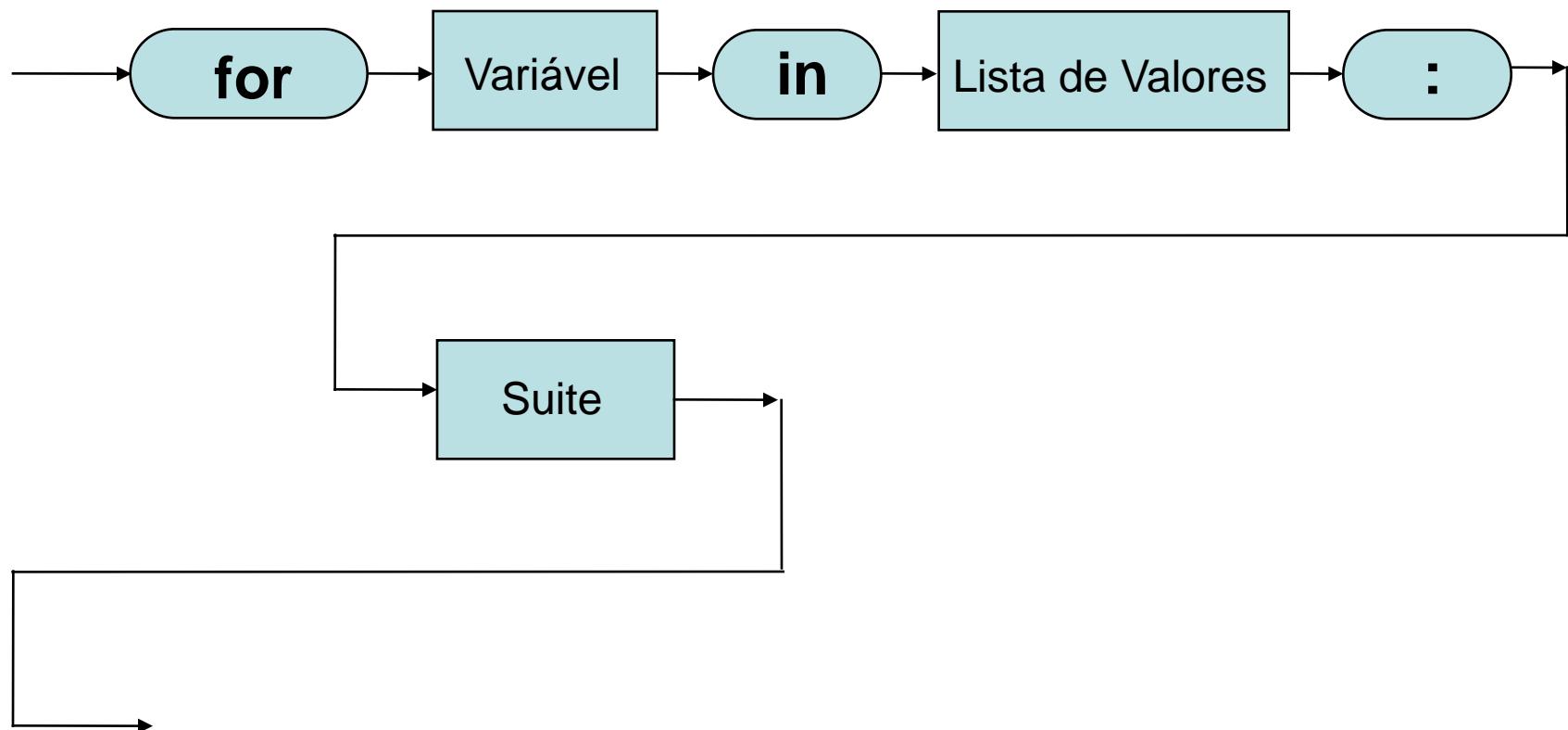
print("O fatorial de", num, "=", fat)
```

Execução:

```
Digite valor inteiro e positivo: 5 <enter>

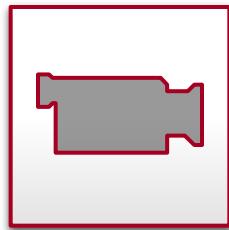
O fatorial de 5 = 120 <enter>
```

Diagrama Sintático de Repetição for



Exemplos de Aplicação dos Conteúdos Vistos

Clique no botão para assistir ao tutorial:



Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:



Aula 3

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo apresentado:

- Estruturas de Controle
 - Estrutura de Sequência
 - **suite**
 - Estruturas de Seleção
 - **if, if-else, if-elif, if-elif-else**
 - Estruturas de Repetição
 - **while e for**



Aula 4

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Subprogramação:
 - Funções
 - Passagem de Parâmetros
 - Recursividade

Subprogramação: Funções

- A utilização de funções permite que:
 - Diferentes partes do programa possam ser desenvolvidas e testadas separadamente;
 - Partes do código possam ser reutilizadas em diferentes pontos do programa;
 - Programas complexos possam ser montados a partir de unidades menores já desenvolvidas e testadas.

Subprogramação: Funções

- A utilização de funções permite que:
 - Diferentes partes do programa possam ser desenvolvidas e testadas separadamente;
 - Partes do código possam ser reutilizadas em diferentes pontos do programa;
 - Programas complexos possam ser montados a partir de unidades menores já desenvolvidas e testadas.
- Trata-se de um grupo de sentenças (suite), comando(s) e/ou estrutura(s) de controle (e funções internas), ao qual é atribuído um nome, que após a sua execução retorna um valor.

Subprogramação: Funções

- A utilização de funções permite que:
 - Diferentes partes do programa possam ser desenvolvidas e testadas separadamente;
 - Partes do código possam ser reutilizadas em diferentes pontos do programa;
 - Programas complexos possam ser montados a partir de unidades menores já desenvolvidas e testadas.
- Trata-se de um grupo de sentenças (suite), comando(s) e/ou estrutura(s) de controle (e funções internas), ao qual é atribuído um nome, que após a sua execução retorna um valor.
- Sua ativação se dá através de seu nome ou de expressões que o contenha.

Declaração de Uma Função em Python

```
def nomeEscolhido(lista de parâmetros):  
    suite do corpo da função
```

A *lista de parâmetros* pode ter zero ou mais parâmetros, separados por vírgulas.

Declaração de Uma Função em Python

```
def nomeEscolhido(lista de parâmetros):  
    suite do corpo da função
```

A *lista de parâmetros* pode ter zero ou mais parâmetros, separados por vírgulas.

A *suite do corpo da função* deve possuir zero ou mais retornos de valores, expressos por

return *valor apropriado*

Caso nenhum valor seja retornado, corresponde a

return ou **return None**

Esboço de Um Programa Contendo Funções

Sempre que possível, as funções devem ser declaradas antes de serem utilizadas. Portanto, devem ficar acima do programa principal ou de outras funções que as utilize.

```
# Programa Completo
# Subprogramas
def nomeFun1(<listaParametros1>):
    <corpo nomeFun1>
...
def nomeFunN(<listaParametrosN>):
    <corpo nomeFunN>
```

Esboço de Um Programa Contendo Funções

Sempre que possível, as funções devem ser declaradas antes de serem utilizadas. Portanto, devem ficar acima do programa principal ou de outras funções que as utilize.

```
# Programa Completo
# Subprogramas
def nomeFun1(<listaParametros1>):
    <corpo nomeFun1>
...
def nomeFunN(<listaParametrosN>):
    <corpo nomeFunN>
# Programa Principal
utiliza nomeFun1(...) ou/e nomeFunN(...)
utiliza nomeFunN(...) ou/e nomeFunN(...)
```

Esboço de Um Programa Contendo Funções

Sempre que possível, as funções devem ser declaradas antes de serem utilizadas. Portanto, devem ficar acima do programa principal ou de outras funções que as utilize.

```
# Programa Completo
# Subprogramas
def nomeFun1(<listaParametros1>):
    <corpo nomeFun1>
...
def nomeFunN(<listaParametrosN>):
    <corpo nomeFunN>
# Programa Principal
utiliza nomeFun1(...) ou/e nomeFunN(...)
utiliza nomeFunN(...) ou/e nomeFunN(...)
```

Portanto, um **programa** passa a ser composto de um **programa principal** e vários outros **subprogramas**, definidos em seu início.

Programa com Variáveis Globais e Funções

Um programa pode conter variáveis globais. No entanto, o uso de variáveis globais não é considerada uma boa prática de programação, por dificultar a legibilidade e compreensão dos códigos. Desta forma, não mais utilizaremos este conceito, após esta aula.

Programa Completo com Variáveis Globais

```
global var1
```

```
...
```

```
global varM
```

Subprogramas

```
def nomeFun1(<listaParametros1>):
```

```
    <corpo nomeFun1>
```

```
...
```

```
def nomeFunN(<listaParametrosN>):
```

```
    <corpo nomeFunN>
```

Programa Principal

```
utiliza var1 e/ou varM e/ou nomeFun1(...) e/ou nomeFunN(...)
```

```
utiliza var1 e/ou varM e/ou nomeFun1(...) e/ou nomeFunN(...)
```

Variáveis Globais vs. Passagem de Parâmetros

Veremos dois programas, um utilizando variáveis globais e outro utilizando passagem de parâmetros, que acessam funções que calculam a multiplicação de dois números através de somas sucessivas.

- A função **mult** retorna o resultado da multiplicação das variáveis **x** e **y**, através do cálculo de sucessivas somas.
- As chamadas **mult()** ou **mult(x, y)** representarão, além da ativação da função, o valor a ser retornado.

Variáveis Globais (Escopo)

```
# Programa Completo com Variáveis Globais
global x
global y
# Subprogramas
def mult():
    z = 0
    for u in range(x):
        z = z + y
    return z
# Programa Principal
x = 20
y = 200
w = mult() + 10
# Neste ponto, w vale 4010
print(w)
```

Variáveis Globais (Escopo)

```
# Programa Completo com Variáveis Globais
```

```
global x
```

```
global y
```

```
# Subprogramas
```

```
def mult():
```

```
    z = 0
```

```
    for u in range(x):
```

```
        z = z + y
```

```
    return z
```

```
# Programa Principal
```

```
x = 20
```

```
y = 200
```

```
w = mult() + 10
```

```
# Neste ponto, w vale 4010
```

```
print(w)
```

Variáveis Globais (Escopo)

```
# Programa Completo com Variáveis Globais
```

```
global x
```

```
global y
```

```
# Subprogramas
```

```
def mult():
```

```
    z = 0
```

```
    for u in range(x):
```

```
        z = z + y
```

```
    return z
```

```
# Programa Principal
```

```
x = 20
```

```
y = 200
```

```
w = mult() + 10
```

```
# Neste ponto, w vale 4010
```

```
print(w)
```

Variáveis Globais (Escopo)

```
# Programa Completo com Variáveis Globais
```

```
global x
```

```
global y
```

```
# Subprogramas
```

```
def mult():
```

```
    z = 0
```

```
    for u in range(x):
```

```
        z = z + y
```

```
    return z
```

```
# Programa Principal
```

```
x = 20
```

```
y = 200
```

```
w = mult() + 10
```

```
# Neste ponto, w vale 4010
```

```
print(w)
```



A função **mult** é ativada nesta atribuição e retorna o valor 4000

Passagem de Parâmetros

Caso uma função deva ser aplicada a diferentes operandos, ou seja, a diferentes valores de entrada, a definição desta função deve conter parâmetros.

Passagem de Parâmetros

Programa Completo sem Variáveis Globais

Subprogramas

```
def mult(x,y):
    z = 0
    for u in range(x):
        z = z + y
    return z
```

Programa Principal

```
w = mult(20, 200) + 10
# Neste ponto, w vale 4010
print(w)
w = mult(10, 100) + 20
# Neste ponto, w vale 1020
print(w)
```

Passagem de Parâmetros

Programa Completo sem Variáveis Globais

Subprogramas

```
def mult(x,y):  
    z = 0  
    for u in range(x):  
        z = z + y  
    return z
```

Observe que as variáveis **x** e **y**, que antes eram variáveis globais, agora são parâmetros da função **mult** e só são utilizadas dentro desta.

Programa Principal

```
w = mult(20, 200) + 10
```

Neste ponto, w vale 4010

```
print(w)
```

```
w = mult(10, 100) + 20
```

Neste ponto, w vale 1020

```
print(w)
```

Passagem de Parâmetros

Programa Completo sem Variáveis Globais

Subprogramas

```
def mult(x,y):  
    z = 0  
    for u in range(x):  
        z = z + y  
    return z
```

Observe que as variáveis **x** e **y**, que antes eram variáveis globais, agora são parâmetros da função **mult** e só são utilizadas dentro desta.

Programa Principal

```
w = mult(20, 200) + 10  
# Neste ponto, w vale 4010  
  
print(w)  
  
w = mult(10, 100) + 20  
# Neste ponto, w vale 1020  
  
print(w)
```

Na **primeira ativação** da função **mult**, os parâmetros **x** e **y** recebem os valores 20 e 200

Passagem de Parâmetros

Programa Completo sem Variáveis Globais

Subprogramas

```
def mult(x,y):  
    z = 0  
    for u in range(x):  
        z = z + y  
    return z
```

Observe que as variáveis **x** e **y**, que antes eram variáveis globais, agora são parâmetros da função **mult** e só são utilizadas dentro desta.

Programa Principal

```
w = mult(20, 200) + 10  
# Neste ponto, w vale 4010  
print(w)  
  
w = mult(10, 100) + 20  
# Neste ponto, w vale 1020  
print(w)
```

Na **primeira ativação** da função **mult**, os parâmetros **x** e **y** recebem os valores 20 e 200

Na **segunda ativação** da função **mult**, os parâmetros **x** e **y** recebem os valores 10 e 100

Passagem de Parâmetros em Python

- Passagem de parâmetro por valor:
 - No início da função, os parâmetros sempre são inicializados com a cópia das referências (ponteiros) para os valores passados na ativação da função
 - Valores de tipos básicos são imutáveis
 - Valores de tipos estruturados (a serem vistos) são mutáveis

Passagem de Parâmetros em Python

- Passagem de parâmetro por valor:
 - No início da função, os parâmetros sempre são inicializados com a cópia das referências (ponteiros) para os valores passados na ativação da função
 - Valores de tipos básicos são imutáveis
 - Valores de tipos estruturados (a serem vistos) são mutáveis
 - Os valores podem vir de
 - Constantes
 - Variáveis
 - Resultados de funções
 - Ou seja, resultados de expressões

Passagem de Parâmetros em Python

- Passagem de parâmetro por valor:
 - No início da função, os parâmetros sempre são inicializados com a cópia das referências (ponteiros) para os valores passados na ativação da função
 - Valores de tipos básicos são imutáveis
 - Valores de tipos estruturados (a serem vistos) são mutáveis
 - Os valores podem vir de
 - Constantes
 - Variáveis
 - Resultados de funções
 - Ou seja, resultados de expressões
 - Observe estas diferentes ativações da função **mult**:

`w = mult(20, 200) + 10`

`w = mult(a, b) + 10`

`w = 13 + mult(a+b, mult(c,d))`

Tipos Mutáveis e Imutáveis como Parâmetro

Referência para tipo mutável
Referências para tipos imutáveis

Programa Completo

```
# Subprograma
def trocar(valores, pos1, pos2):      # se possível, modifica o conteúdo de duas células
    if 0<=pos1<len(valores) and 0<=pos2<len(valores):
        temp = valores[pos1]
        valores[pos1] = valores[pos2]
        valores[pos2] = temp
    return None
```

Programa Principal

```
amigas = ["Maria", "Regina", "Eliana", "Angelica"] # vetor com 4 strings – próximas aulas
trocar(amigas, 3, 1)
# Neste ponto, amigas = ["Maria", "Angelica", "Eliana", "Regina"]

trocar(amigas, 0, 2)
# Neste ponto, amigas = ["Eliana", "Angelica", "Maria", "Regina"]
```

Tipos Mutáveis e Imutáveis como Parâmetro

Programa Completo

Subprograma

```
def trocar(valores, pos1, pos2):      # se possível, modifica o conteúdo de duas células
    if 0<=pos1<len(valores) and 0<=pos2<len(valores):
        temp = valores[pos1]
        valores[pos1] = valores[pos2]
        valores[pos2] = temp
    return None
```

Programa Principal

```
amigas = ["Maria", "Regina", "Eliana", "Angelica"] # vetor com 4 strings – próximas aulas
trocar(amigas, 3, 1)
# Neste ponto, amigas = ["Maria", "Angelica", "Eliana", "Regina"]

trocar(amigas, 0, 2)
# Neste ponto, amigas = ["Eliana", "Angelica", "Maria", "Regina"]
```

Referência para tipo mutável

Referências para tipos imutáveis

Esta passagem de parâmetro ocorre como se o primeiro argumento (a variável **amigas** – também chamado “parâmetro real”) substituisse o parâmetro dentro do escopo do função (o “parâmetro formal” **valores**).

Tipos Mutáveis e Imutáveis como Parâmetro

Programa Completo

Subprograma

```
def trocar(valores, pos1, pos2):      # se possível, modifica o conteúdo de duas células
    if 0<=pos1<len(valores) and 0<=pos2<len(valores):
```

```
        temp = valores[pos1]
        valores[pos1] = valores[pos2]
        valores[pos2] = temp
```

```
    return None
```

Referência para tipo mutável

Referências para tipos imutáveis

Programa Principal

```
amigas = ["Maria", "Regina", "Eliana", "Angelica"] # vetor com 4 strings – próximas aulas
trocar(amigas, 3, 1)
```

Neste ponto, amigas = ["Maria", "Angelica", "Eliana", "Regina"]

```
trocar(amigas, 0, 2)
```

Neste ponto, amigas = ["Eliana", "Angelica", "Maria", "Regina"]

Esta passagem de parâmetro ocorre como se o primeiro argumento (a variável **amigas** – também chamado “parâmetro real”) substituisse o parâmetro dentro do escopo do função (o “parâmetro formal” **valores**).

Esta é a forma de se modificar, dentro de uma função, conteúdos de variáveis de um programa principal.

Funções Como Parâmetros

Programa Completo

Subprogramas

```
def fatorial(num):
```

```
    if num==0:
```

```
        return 1
```

```
    else:
```

```
        return num*fatorial(num-1)
```

```
def fib(num):
```

```
    if 1<=num<=2:
```

```
        return 1
```

```
    else:
```

```
        return fib(num-1)+fib(num-2)
```

```
def soma(f, n):    # Esta função soma os n primeiros valores de uma dada função f
```

```
    parcial = 0
```

```
    for ind in range(1, n+1):
```

```
        parcial = parcial + f(ind)
```

```
    return parcial
```

Programa Principal

```
total = soma(fatorial,10) + soma(fib,10)
print(total)
```



Funções Como Parâmetros

Programa Completo

Subprogramas

```
def fatorial(num):
    if num==0:
        return 1
    else:
        return num*fatorial(num-1)
```

```
def fib(num):
    if 1<=num<=2:
        return 1
    else:
        return fib(num-1)+fib(num-2)
```

def soma(f, n): # Esta função soma os n primeiros valores de uma dada função f

parcial = 0

for ind **in** range(1, n+1):

 parcial = parcial + f(ind)

return parcial

Programa Principal

```
total = soma(fatorial,10) + soma(fib,10)
print(total)
```

28



Funções Como Parâmetros

Programa Completo

Subprogramas

```
def fatorial(num):
    if num==0:
        return 1
    else:
        return num*fatorial(num-1)
```

```
def fib(num):
    if 1<=num<=2:
        return 1
    else:
        return fib(num-1)+fib(num-2)
```

```
def soma(f, n):      # Esta função soma os n primeiros valores de uma dada função f
    parcial = 0
    for ind in range(1, n+1):
        parcial = parcial + f(ind)
    return parcial
```

Programa Principal

```
total = soma(fatorial,10) + soma(fib,10)
print(total)
```

29

Funções Como Parâmetros

Programa Completo

Subprogramas

```
def fatorial(num):
    if num==0:
        return 1
    else:
        return num*fatorial(num-1)
```

```
def fib(num):
    if 1<=num<=2:
        return 1
    else:
        return fib(num-1)+fib(num-2)
```

```
def soma(f, n):    # Esta função soma os n primeiros valores de uma dada função f
    parcial = 0
    for ind in range(1, n+1):
        parcial = parcial + f(ind)
    return parcial
```

Programa Principal

```
total = soma(fatorial,10) + soma(fib,10)
print(total)
```



Funções Como Parâmetros

Programa Completo

Subprogramas

```
def fatorial(num):
    if num==0:
        return 1
    else:
        return num*fatorial(num-1)
```

```
def fib(num):
    if 1<=num<=2:
        return 1
    else:
        return fib(num-1)+fib(num-2)
```

```
def soma(f, n):    # Esta função soma os n primeiros valores de uma dada função f
    parcial = 0
    for ind in range(1, n+1):
        parcial = parcial + f(ind)
    return parcial
```

Programa Principal

```
total = soma(fatorial,10) + soma(fib,10)
print(total)
```

31

Funções Como Parâmetros

Programa Completo

Subprogramas

```
def fatorial(num):
    if num==0:
        return 1
    else:
        return num*fatorial(num-1)
```

```
def fib(num):
    if 1<=num<=2:
        return 1
    else:
        return fib(num-1)+fib(num-2)
```

```
def soma(f, n):      # Esta função soma os n primeiros valores de uma dada função f
    parcial = 0
    for ind in range(1, n+1):
        parcial = parcial + f(ind)
    return parcial
```

Programa Principal

```
total = soma(fatorial,10) + soma(fib,10)
print(total)
```

32

Escopo de um Identificador em Python

Os parâmetros, as variáveis locais e as funções declaradas internamente a uma função definem identificadores que são **locais** a esta função, isto é: têm **escopo local**.

Estes identificadores não podem ser utilizados fora da respectiva função, isto é, não são **visíveis** em outra parte do programa.

Escopo de um Identificador em Python

- A região de validade de um identificador é chamada de escopo do identificador.

Escopo de um Identificador em Python

- A região de validade de um identificador é chamada de escopo do identificador.
- Um identificador é chamado de global se o seu escopo é todo o programa.
 - Isto é: o programa principal e os seus subprogramas.

Escopo de um Identificador em Python

- A região de validade de um identificador é chamada de escopo do identificador.
- Um identificador é chamado de global se o seu escopo é todo o programa.
 - Isto é: o programa principal e os seus subprogramas.
- O escopo de um identificador é dito local se ele é válido apenas na função que é definido.

Escopo de um Identificador em Python

Programa Completo

global x

x escopo global

Subprogramas

def p(n):

y = 13

n escopo local a p

y escopo local a p

Neste ponto, x, y e n são conhecidos

return x+y+n

def q(m):

z = 3

m escopo local a q

z escopo local a q

Neste ponto, x, z e m são conhecidos

return m**z - x*m

Programa Principal

x = 26

Neste ponto, apenas x, p e q são conhecidos

print(p(x), q(2*x+p(3)))

Escopo de um Identificador em Python

Programa Completo

global x # x escopo global

Subprogramas

```
def p(n):          # n escopo local a p
    y = 13         # y escopo local a p
    # Neste ponto, x, y e n são conhecidos
    return x+y+n
```

def q(m): # m escopo local a q

z = 3 # z escopo local a q

Neste ponto, x, z e m são conhecidos

return mz - x*m**

Programa Principal

x = 26

Neste ponto, apenas x, p e q são conhecidos

print(p(x), q(2*x+p(3)))

Escopo de um Identificador em Python

Programa Completo

global x # x escopo global

Subprogramas

```
def p(n):          # n escopo local a p
    y = 13         # y escopo local a p
    # Neste ponto, x, y e n são conhecidos
    return x+y+n
```

```
def q(m):          # m escopo local a q
```

```
    z = 3           # z escopo local a q
```

```
    # Neste ponto, x, z e m são conhecidos
```

```
    return m**z - x*m
```

Programa Principal

x = 26

Neste ponto, apenas x, p e q são conhecidos

```
print(p(x), q(2*x+p(3)))
```

Escopo de um Identificador em Python

Programa Completo

global x # x escopo global

Subprogramas

def p(n): # n escopo local a p

 y = 13 # y escopo local a p

 # Neste ponto, x, y e n são conhecidos

 return x+y+n

def q(m): # m escopo local a q

 z = 3 # z escopo local a q

 # Neste ponto, x, z e m são conhecidos

 return m**z - x*m

Programa Principal

x = 26

Neste ponto, apenas x, p e q são conhecidos

print(p(x), q(2*x+p(3)))

Escopo de um Identificador em Python

Programa Completo

global x

Variável Global x

Subprograma

def colisao(m):

x = 8

z = 13

Neste ponto, as variáveis x e z locais a q
e o parâmetro m são conhecidos.

x = x + 1

A variável local x foi alterada.

print(x, z, m*2) # escreve 9, 13 e 2000

return None

Programa Principal

Neste ponto, apenas variável global x é conhecida, além do nome da função.

x = 57

colisao(1000)

print(x) # escreve 57

No caso de existirem dois identificadores, definidos em escopos diferentes, com o mesmo nome x, a ocorrência do nome x estará referenciando aquele com o escopo mais local.

Escopo de um Identificador em Python

Programa Completo

global x

Variável Global x

Subprograma

def colisao(m):

x = 8

z = 13

Neste ponto, as variáveis x e z locais a q
e o parâmetro m são conhecidos.

x = x + 1

A variável local x foi alterada.

print(x, z, m*2) # escreve 9, 13 e 2000

return None

No caso de existirem dois identificadores, definidos em escopos diferentes, com o mesmo nome x, a ocorrência do nome x estará referenciando aquele com o escopo mais local.

Programa Principal

Neste ponto, apenas variável global x é conhecida, além do nome da função.

x = 57

colisao(1000)

print(x) # escreve 57

Escopo de um Identificador em Python

Programa Completo

global x

Variável Global x

Subprograma

def colisao(m):

x = 8

z = 13

Variável Local x

Neste ponto, as variáveis x e z locais a q
e o parâmetro m são conhecidos.

x = x + 1

A variável local x foi alterada.

print(x, z, m*2) # escreve 9, 13 e 2000

return None

No caso de existirem dois identificadores, definidos em escopos diferentes, com o mesmo nome x, a ocorrência do nome x estará referenciando aquele com o escopo mais local.

Programa Principal

Neste ponto, apenas variável global x é conhecida, além do nome da função.

x = 57

colisao(1000)

print(x) # escreve 57

Escopo de um Identificador em Python

```
# Programa Completo
# Subprograma
def calcula(x,y):
    # Função Interna
    def cubo(z):
        return z**3
    #
    return cubo(x) - y
# Programa Principal
x = calcula(10,20)
# Neste ponto, x vale (10*10*10)-20.
print(x) # escreve 980
```

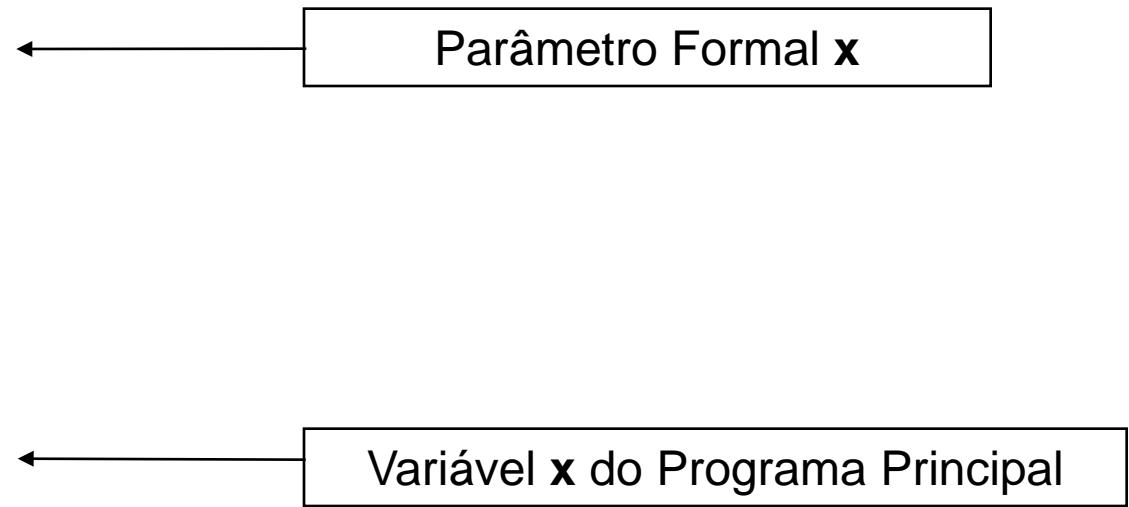
Escopo de um Identificador em Python

```
# Programa Completo
# Subprograma
def calcula(x,y):
    # Função Interna
    def cubo(z):
        return z**3
    #
    return cubo(x) - y
# Programa Principal
x = calcula(10,20)
# Neste ponto, x vale (10*10*10)-20.
print(x) # escreve 980
```



Escopo de um Identificador em Python

```
# Programa Completo
# Subprograma
def calcula(x,y):
    # Função Interna
    def cubo(z):
        return z**3
    #
    return cubo(x) - y
# Programa Principal
x = calcula(10,20)
# Neste ponto, x vale (10*10*10)-20.
print(x) # escreve 980
```



Ativação de Funções

Quando uma função é chamada, duas tarefas são executadas:

- a) Criação de um espaço de memória para as variáveis locais e parâmetros; e
- b) Passagem efetiva de parâmetros.

Ativação de Funções

- Durante a execução de um programa:
 - Uma área especial de memória, organizada em forma de pilha (“stack”), é utilizada para armazenar os valores
 - das variáveis locais e
 - parâmetros das funções;
 - Outra área é organizada para manter as variáveis globais do programa principal.
- Quando o programa principal é iniciado, um espaço é criado para manter as variáveis globais.
- Sempre que uma função é ativada, um espaço contendo os valores das variáveis locais e dos parâmetros é reservado no topo da pilha.
- Ao fim de uma função, seu espaço é automaticamente eliminado do topo da pilha, podendo ser reutilizado.

Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y)
```

```
print(z)
```

Pilha de Ativação e Registro de Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y)
```

```
print(z)
```

Pilha de Ativação e Registro de Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

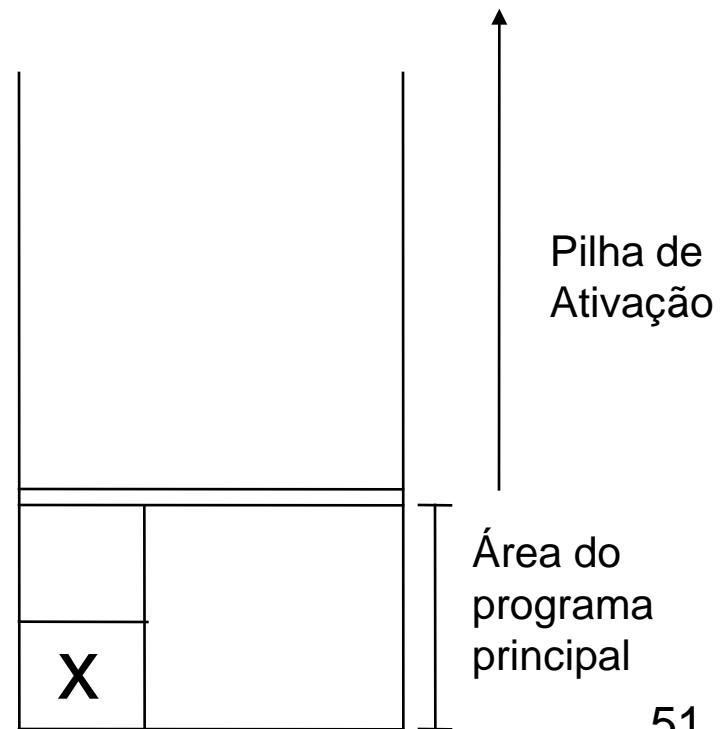
Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y)
```

```
print(z)
```



Pilha de Ativação e Registro de Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

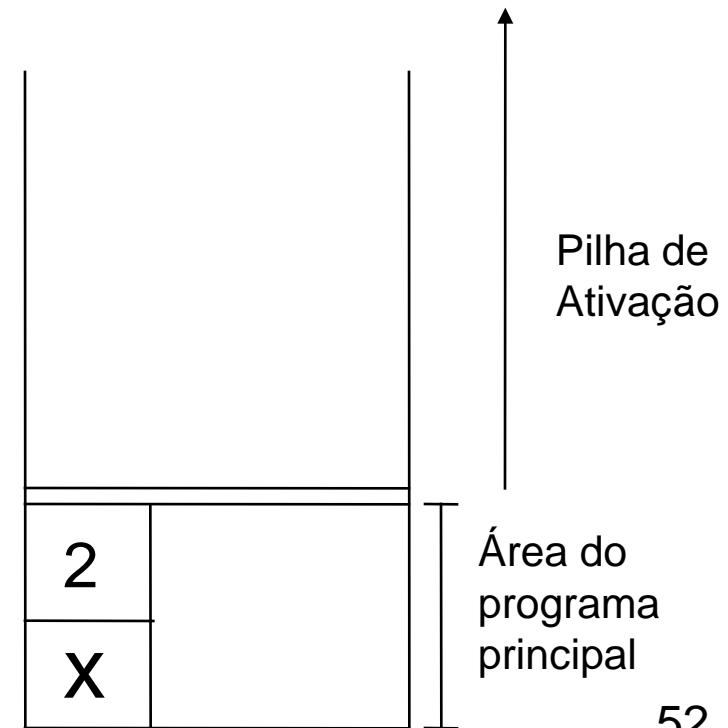
Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y)
```

```
print(z)
```



Pilha de Ativação e Registro de Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

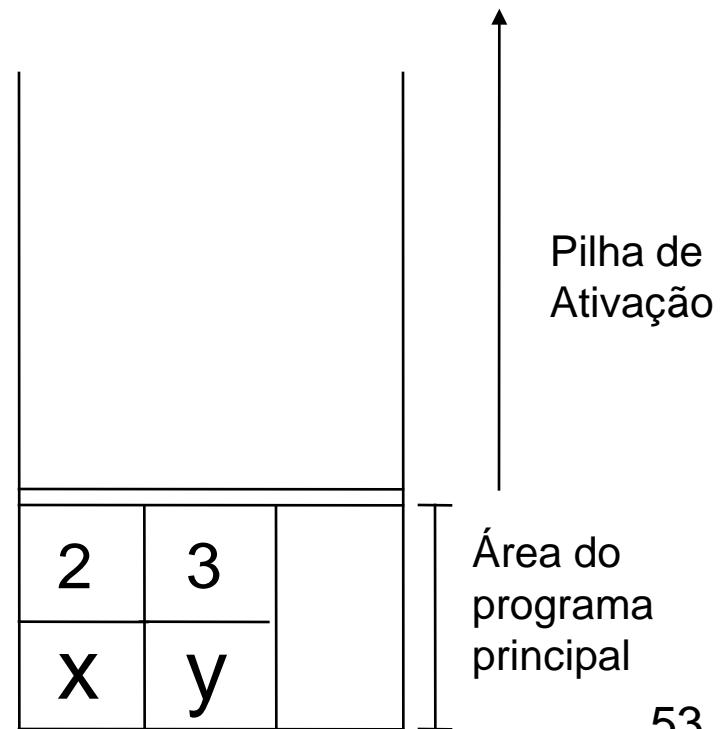
Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y)
```

```
print(z)
```



Pilha de Ativação e Registro de Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

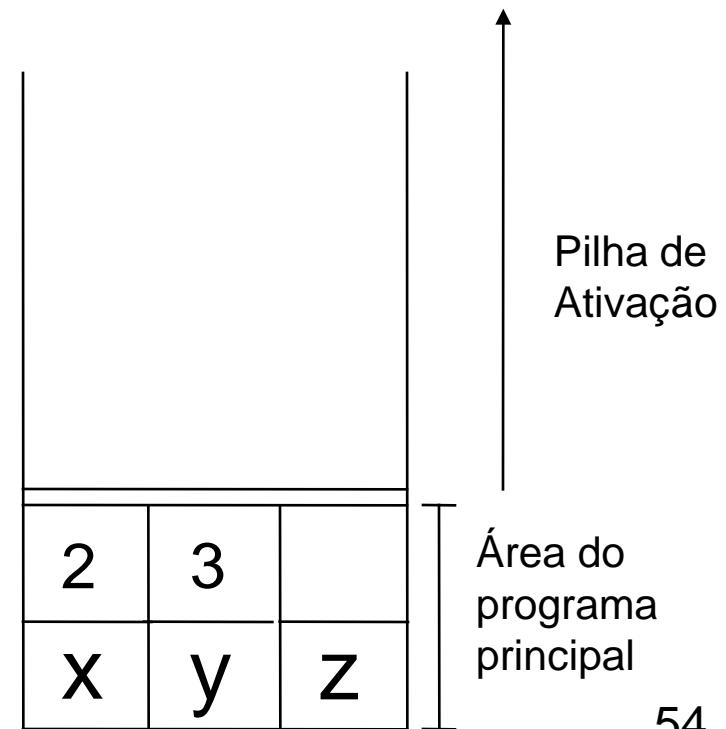
Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y) ←
```

```
print(z)
```



Pilha de Ativação e Registro de Ativação de Funções

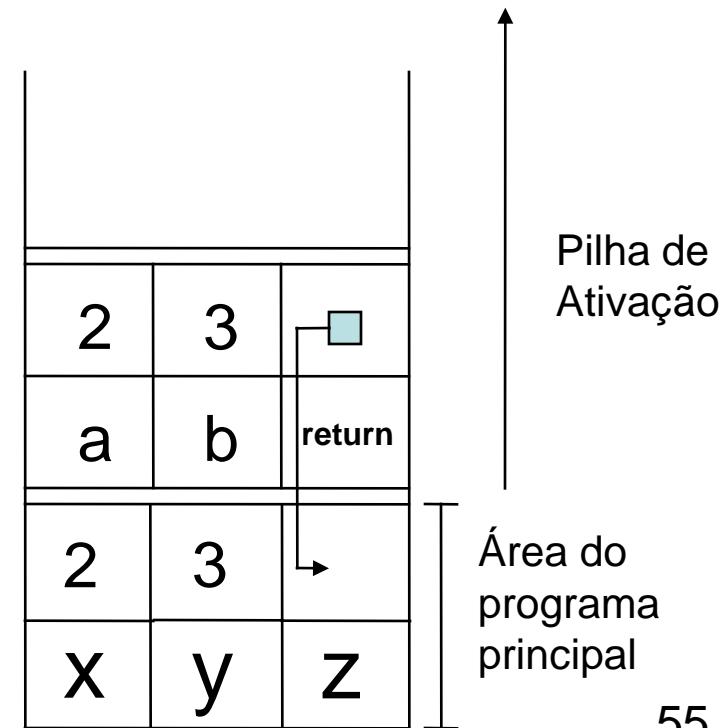
Programa Completo

Subprograma

```
def soma(a,b): ←  
    return a + b
```

Programa Principal

```
x = 2  
y = 3  
z = soma(x,y)  
print(z)
```



Pilha de Ativação e Registro de Ativação de Funções

Programa Completo

Subprograma

```
def soma(a,b):  
    return a + b
```

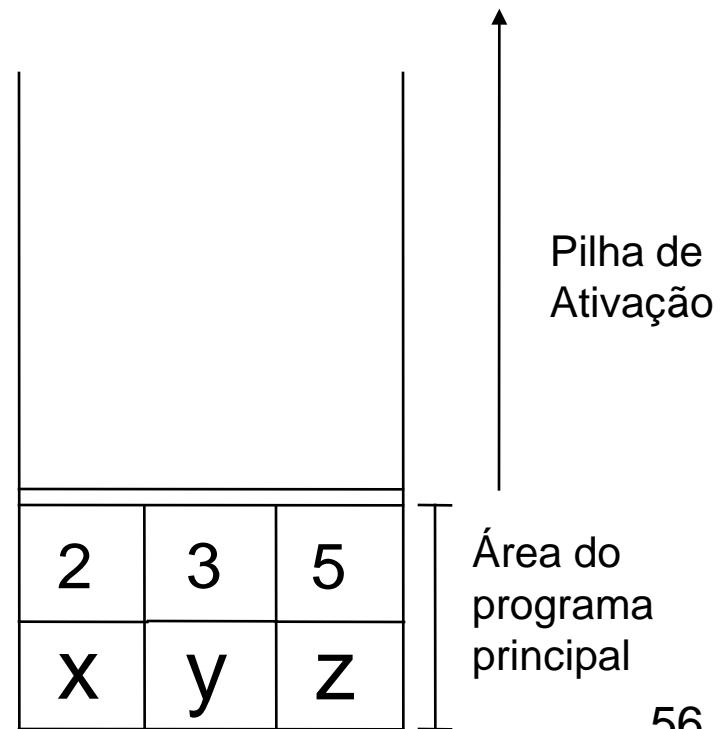
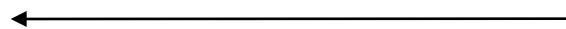
Programa Principal

```
x = 2
```

```
y = 3
```

```
z = soma(x,y)
```

```
print(z)
```



Recursividade

- Uma função é chamada recursiva quando possui no seu corpo uma chamada a ela própria.
- Um exemplo comum de utilização de recursividade é o cálculo da função fatorial de um número natural.

$$n! = \begin{cases} 1 & , \text{ se } n = 0; \text{ \{caso base\}} \\ n * (n-1)! & , \text{ se } n > 0. \text{ \{expressão de recorrência\}} \end{cases}$$

$$3! = 3 * 2! = 3 * 2 * 1! = 3 * 2 * 1 * 0! = 3 * 2 * 1 * 1 = 6$$

Recursividade

$$n! = \begin{cases} 1 & , \text{ se } n = 0; \text{ \{caso base\}} \\ n * (n-1)! & , \text{ se } n > 0. \text{ \{expressão de recorrência\}} \end{cases}$$

```
def fat(n):
    if n == 0:                      # condição de parada
        return 1
    else:
        return n*fat(n-1)           # chamada recursiva
```

Recursividade

Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```

Recursividade

Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3) ←
print(x)
```

Área do
programa
principal

60

X

Recursividade

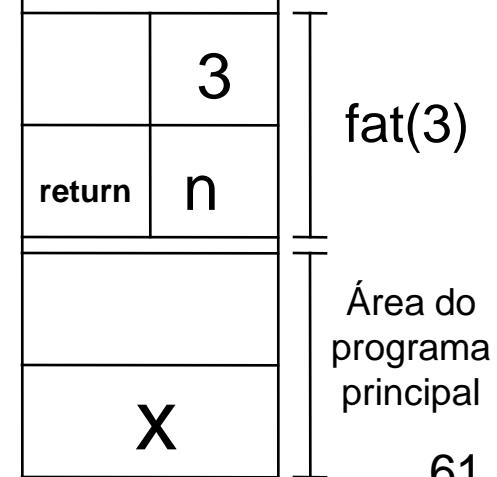
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

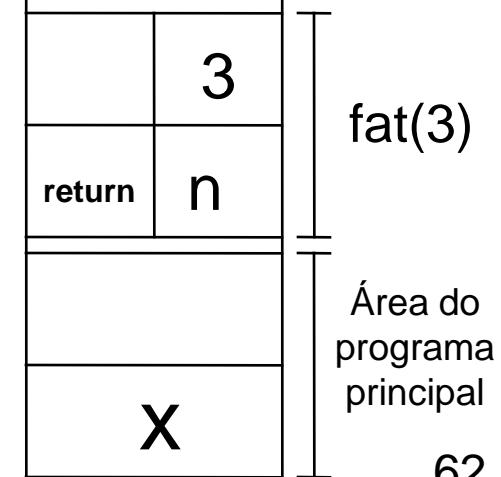
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



62

Recursividade

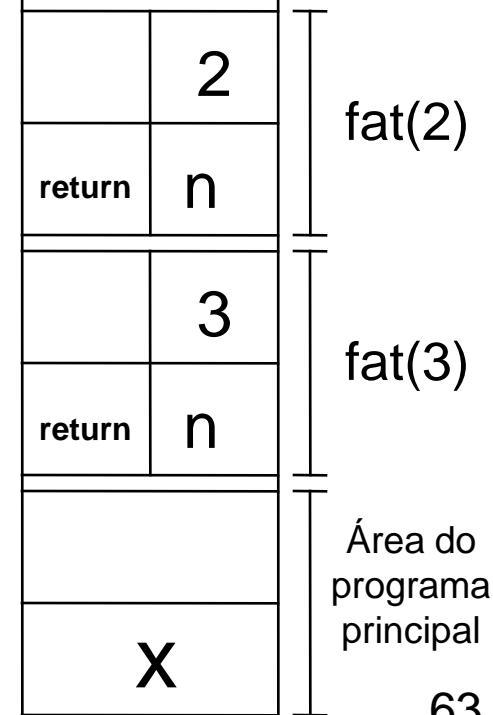
Programa Completo

Subprograma

```
def fat(n):  
    if n == 0:  
        return 1  
    else:  
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)  
print(x)
```



Recursividade

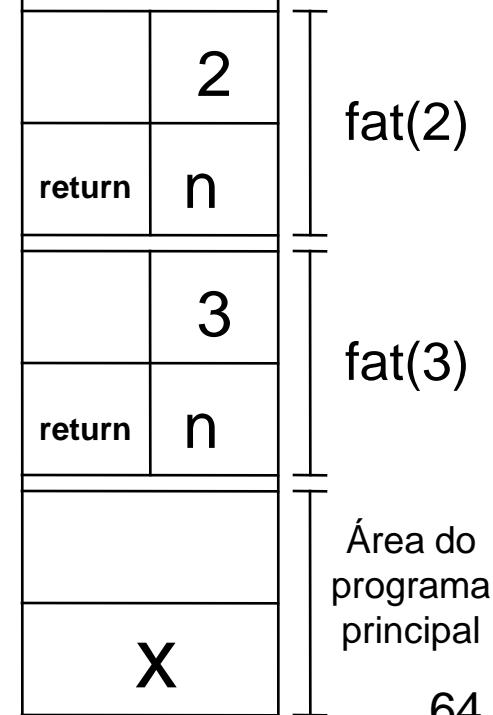
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



64

Recursividade

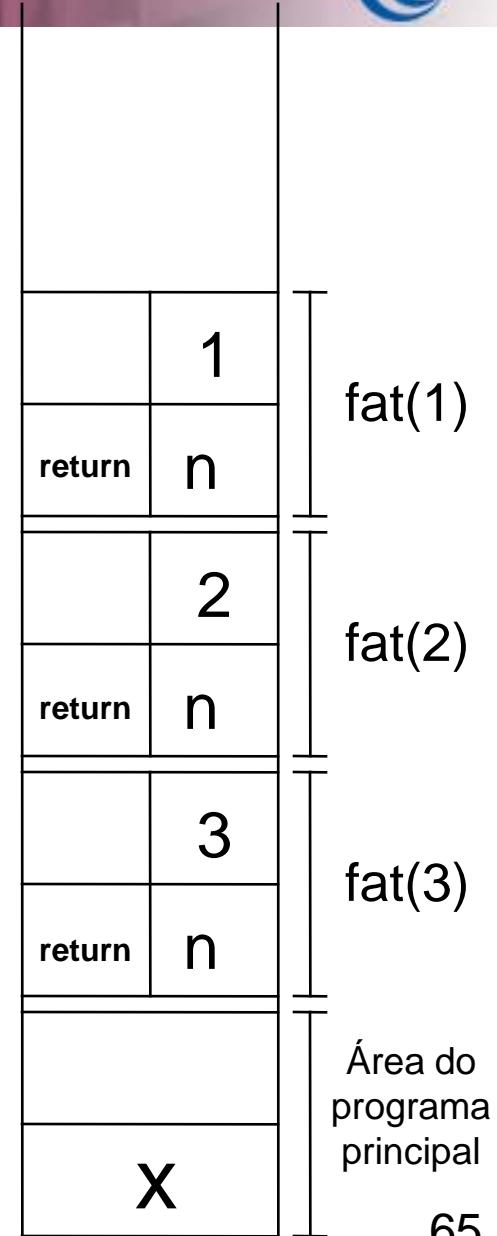
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

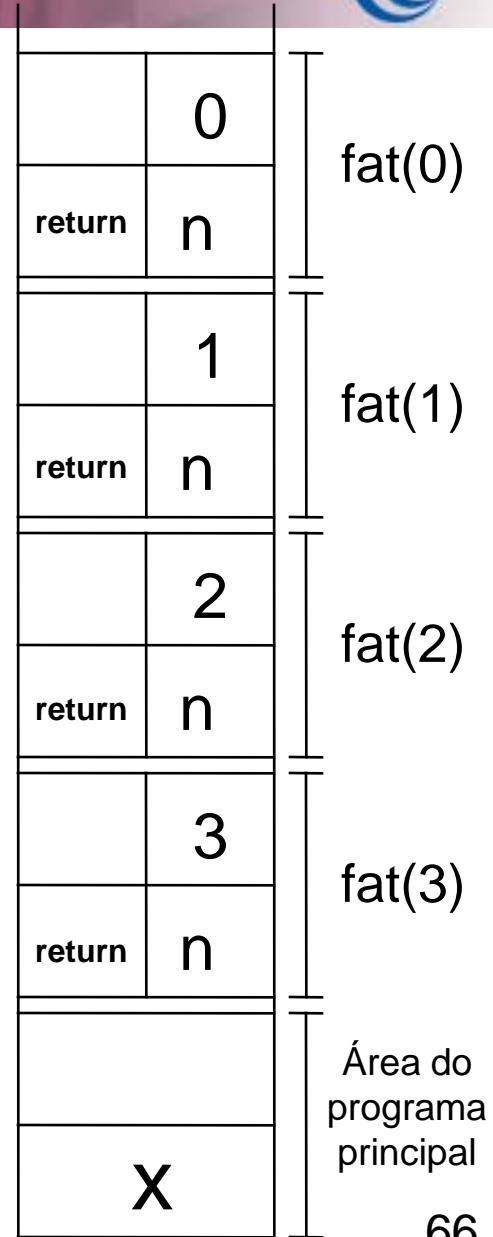
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

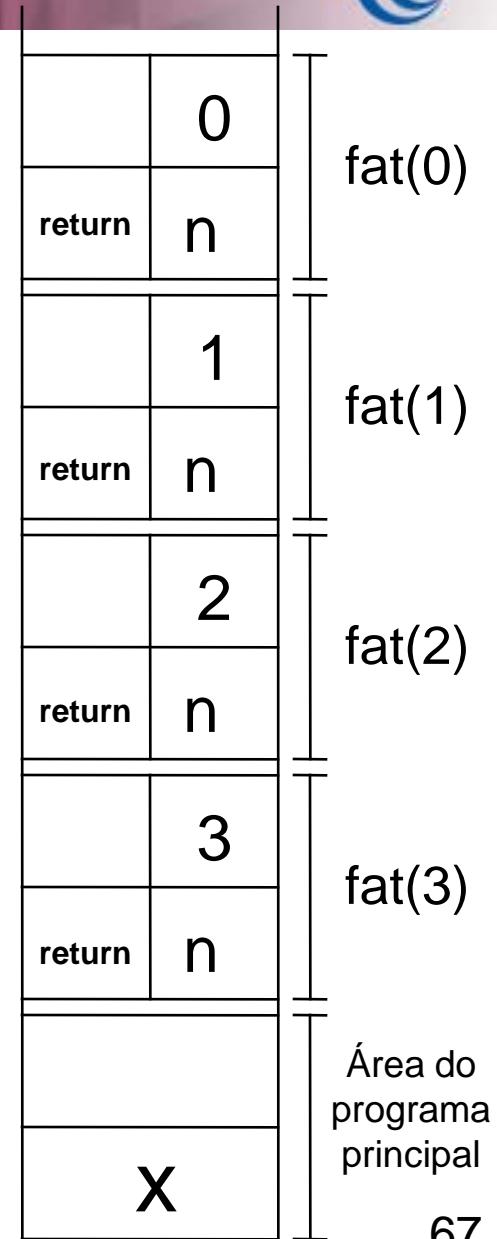
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

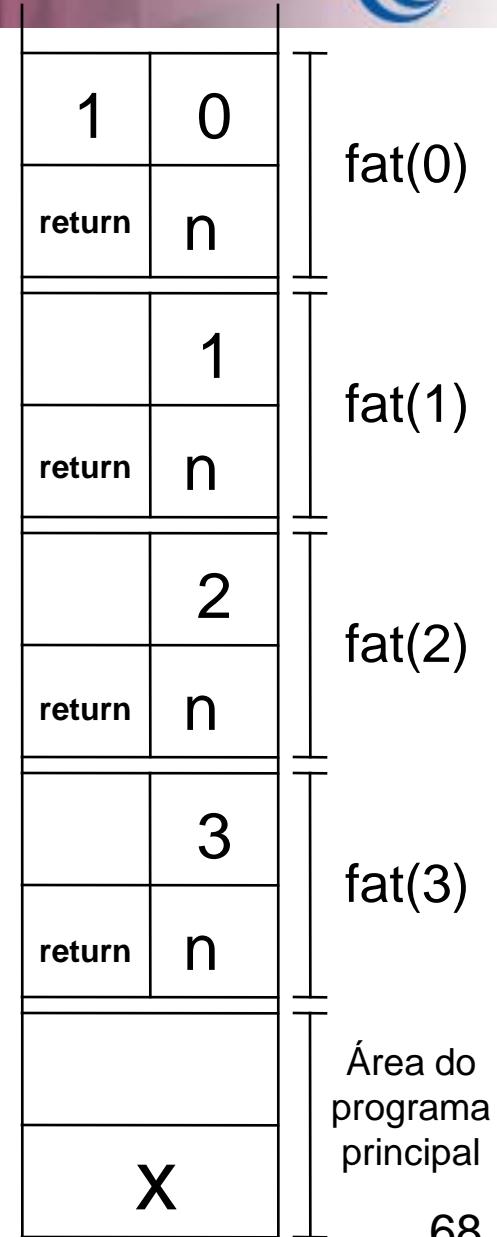
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

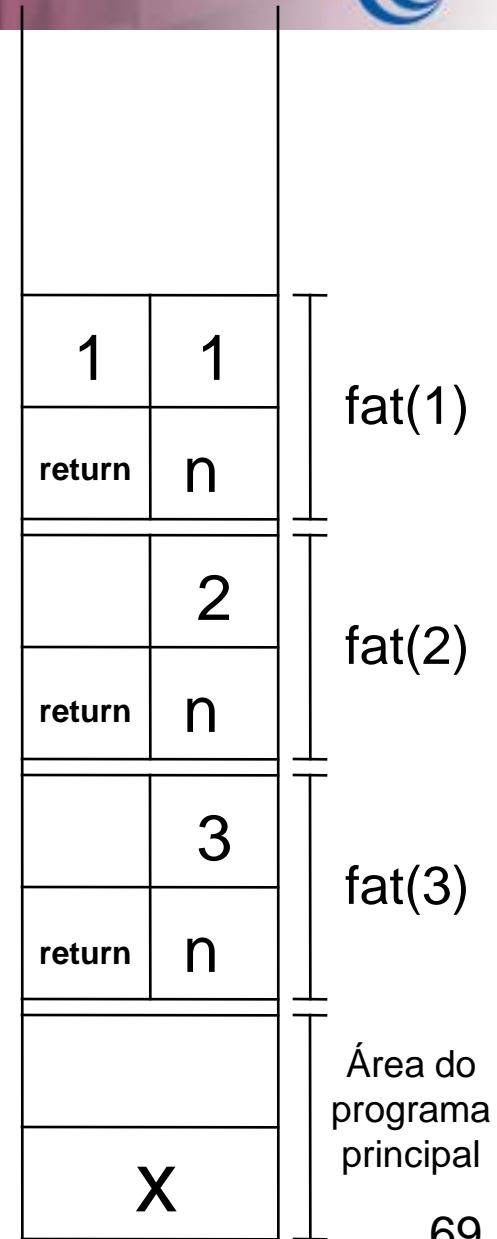
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

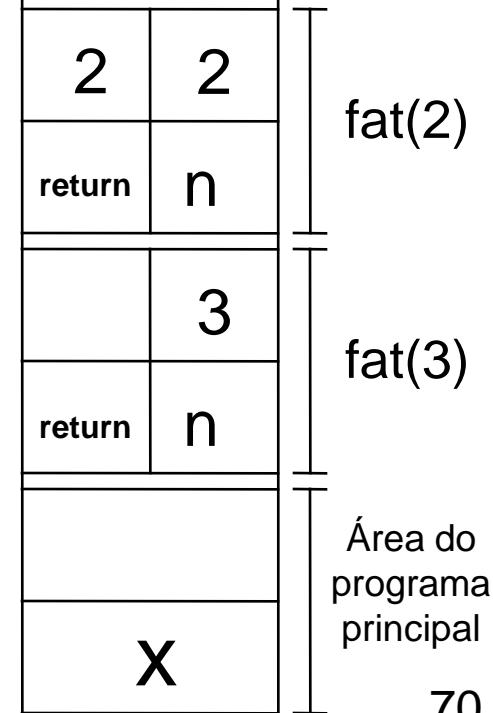
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



Recursividade

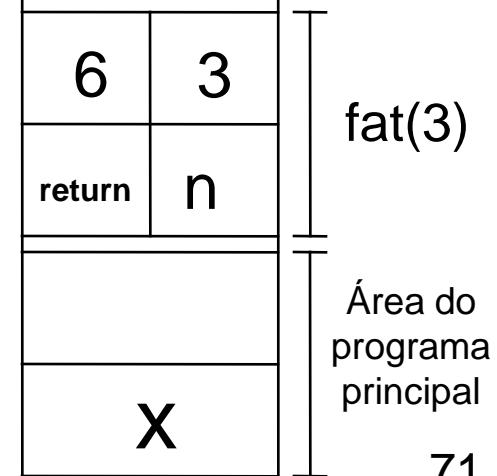
Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```



71

Recursividade

Programa Completo

Subprograma

```
def fat(n):
    if n == 0:
        return 1
    else:
        return n*fat(n-1)
```

Programa Principal

```
x = fat(3)
print(x)
```

6

X

Área do
programa
principal

72

Recursividade

- Trata-se, também, de uma forma de repetição da execução de um determinado trecho de código.
- Apesar de nenhum comando explícito de repetição ter sido utilizado, na prática o código anterior executa um produto de N termos.
- A função fatorial poderia também ser definida por:

$$n! = \begin{cases} 1 & , \text{ se } n = 0; \\ 1 * 2 * \dots * n & , \text{ se } n > 0. \end{cases}$$

Implementação Iterativa para o Fatorial

$$n! = \begin{cases} 1 & , \text{ se } n = 0; \\ 1 * 2 * \dots * n & , \text{ se } n > 0. \end{cases}$$

```
def fat(n):
    p = 1
    for ind in range(1,n+1):
        p = p * ind
    return p
```

Recursividade Mútua

- Em alguns casos, pode ser necessário que dois subprogramas se chamem reciprocamente (recursividade mútua).

```
# Programa Completo
```

```
# Subprogramas
```

```
def flip(n):
```

```
    print("Flip")
```

```
    if n>0:
```

```
        flop(n-1)
```

```
def flop(n):
```

```
    print("Flop")
```

```
    if n>0:
```

```
        flip(n-1)
```

```
# Programa Principal
```

```
flip(5)
```

Recursividade Mútua

- Em alguns casos, pode ser necessário que dois subprogramas se chamem reciprocamente (recursividade mútua).

Programa Completo

Subprogramas

def flip(*n*):

 print("Flip")

if *n*>0:

 flop(*n*-1)

def flop(*n*):

 print("Flop")

if *n*>0:

 flip(*n*-1)

Programa Principal

flip(5)

Saída:

Flip

Flop

Flip

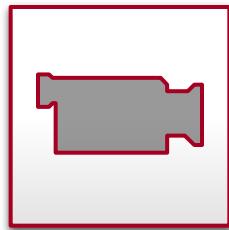
Flop

Flip

Flop

Exemplos de Aplicação dos Conteúdos Vistos

Clique no botão para assistir ao tutorial:



Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:



Aula 4

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo Apresentado:

- Subprogramação:
 - Funções
 - Passagem de Parâmetros
 - Passagem por Valor,
 - Passagem de Função
 - Recursividade

Aula 5

Professores:

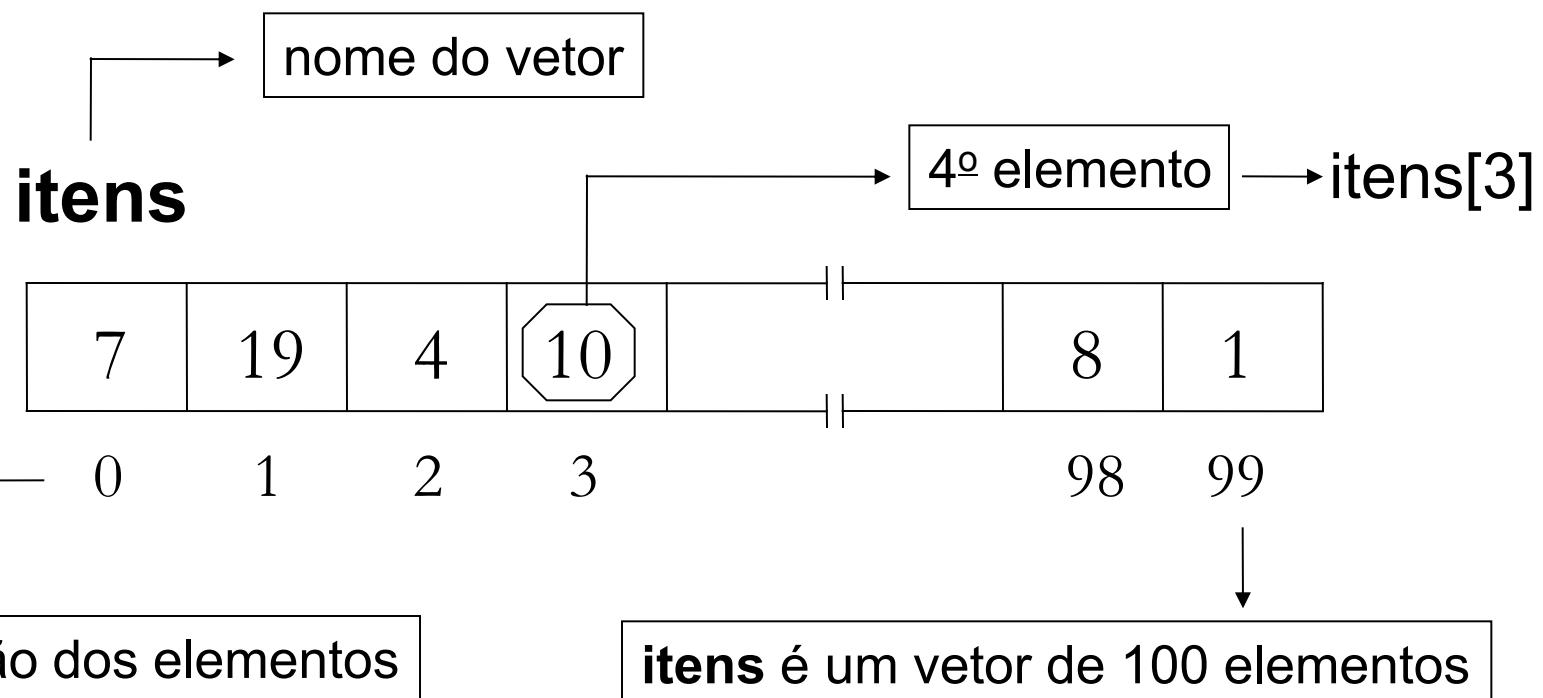
Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

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

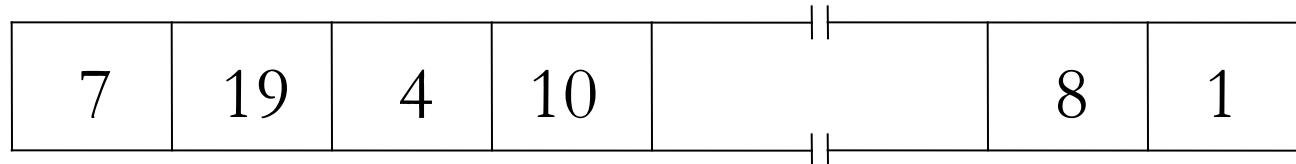
Vetor

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.

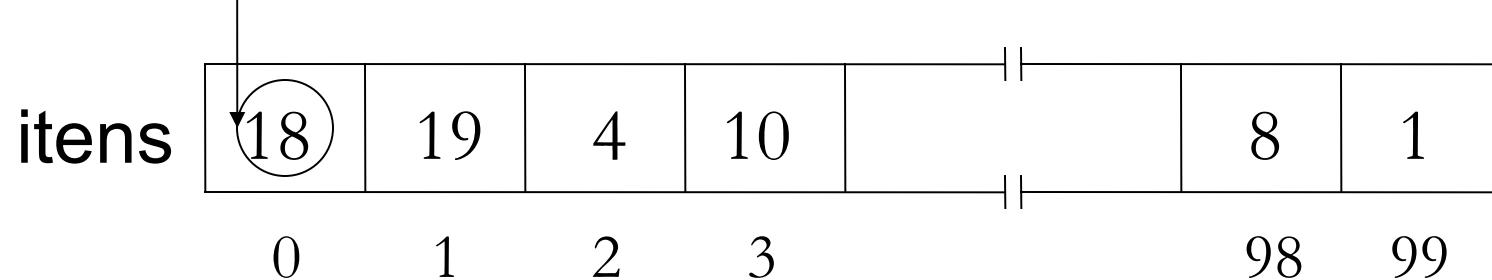
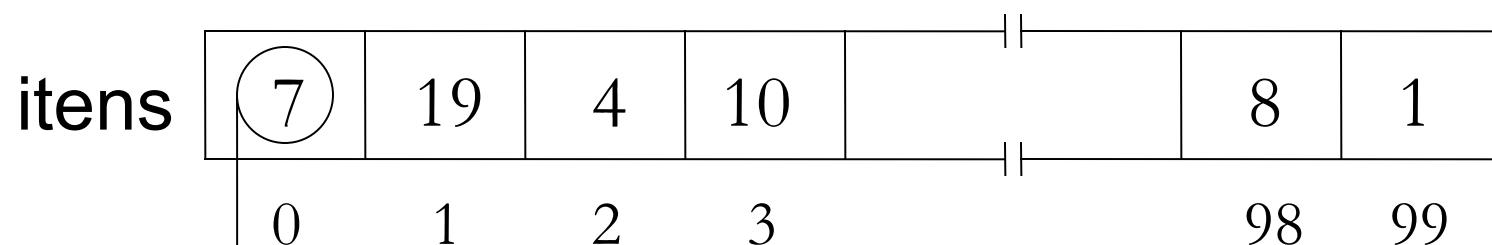


O índice ou seletor do elemento indica sua ordem (posição)

3

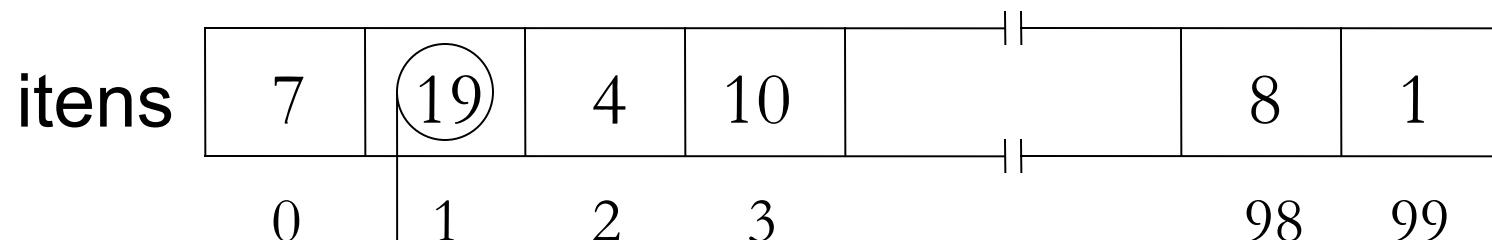
Vetor

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.



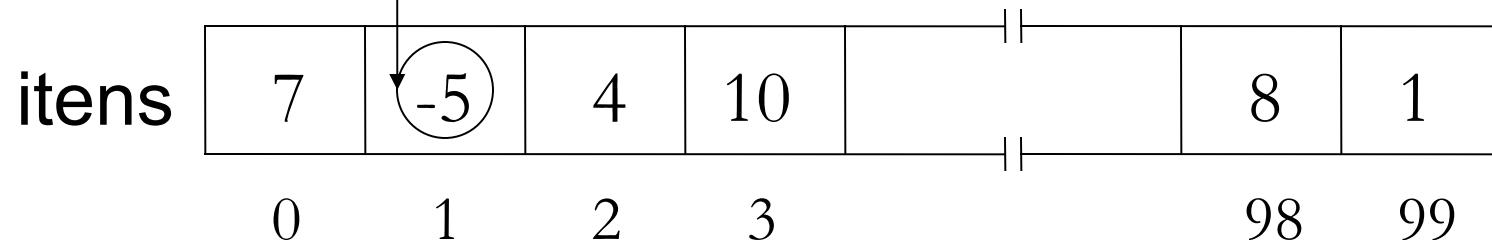
$$i = 1$$

$$\text{itens}[i] = \text{itens}[i-1]*2 - \text{itens}[\text{itens}[99]]$$

$$14$$

$$1$$

$$19$$



Vetor

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

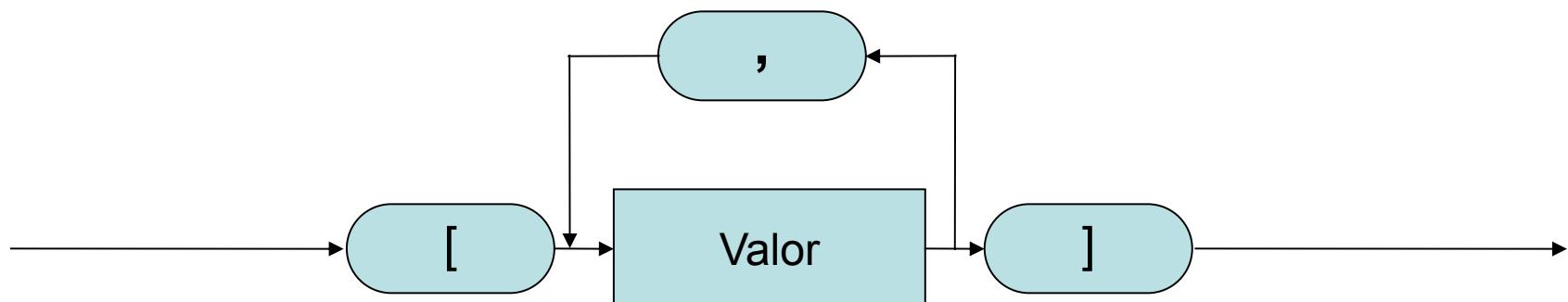
Vetor

- 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

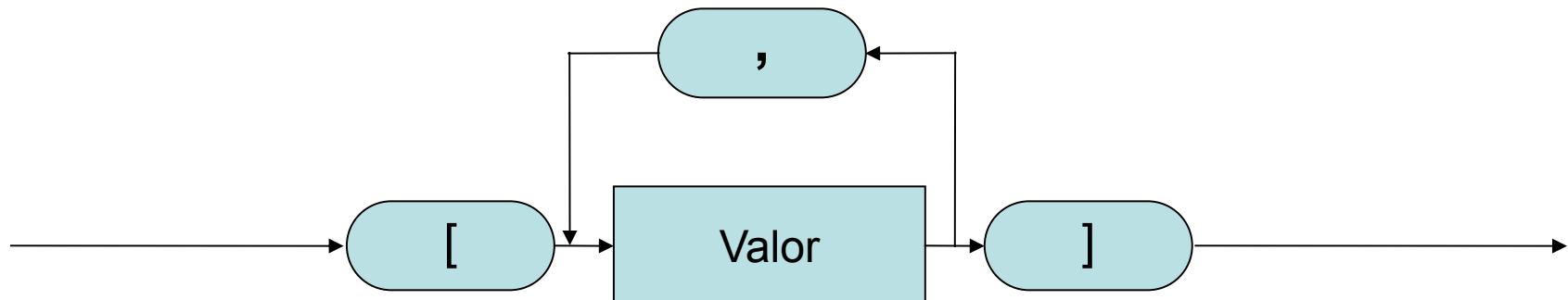
- Características:
 - Trata-se de uma estrutura homogênea, isto é, formada por elementos de um mesmo tipo, chamado tipo base;
 - Todos os elementos da estrutura são igualmente acessíveis, ou seja, o tipo de procedimento para acessar qualquer elemento é igual;
 - Cada elemento da estrutura tem um nome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

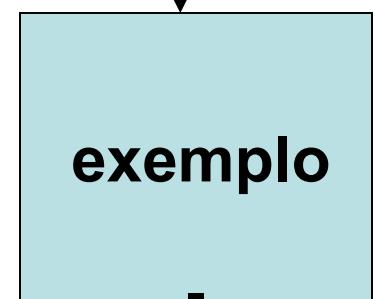
Metodologia para a Solução:

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)
```

Teclado



Vídeo

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

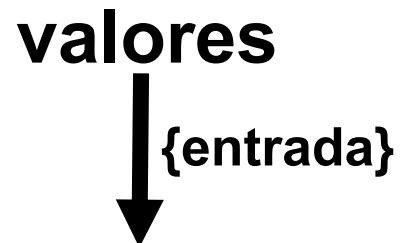
# Programa Principal
TAM = 10
numeros = [0.0]*TAM
ler(numeros)
escrever(numeros)
ordenar(numeros)
escrever(numeros)
```

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)
```



`escrever`

`Vídeo`

23

Metodologia para a Solução:

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

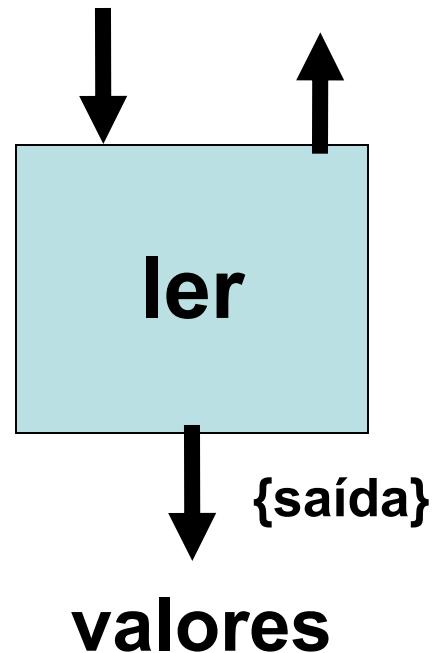
Subprogramas

```
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
```

Programa Principal

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

Teclado Vídeo



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
```

ordenar

{entrada/saída}

valores

```

# 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

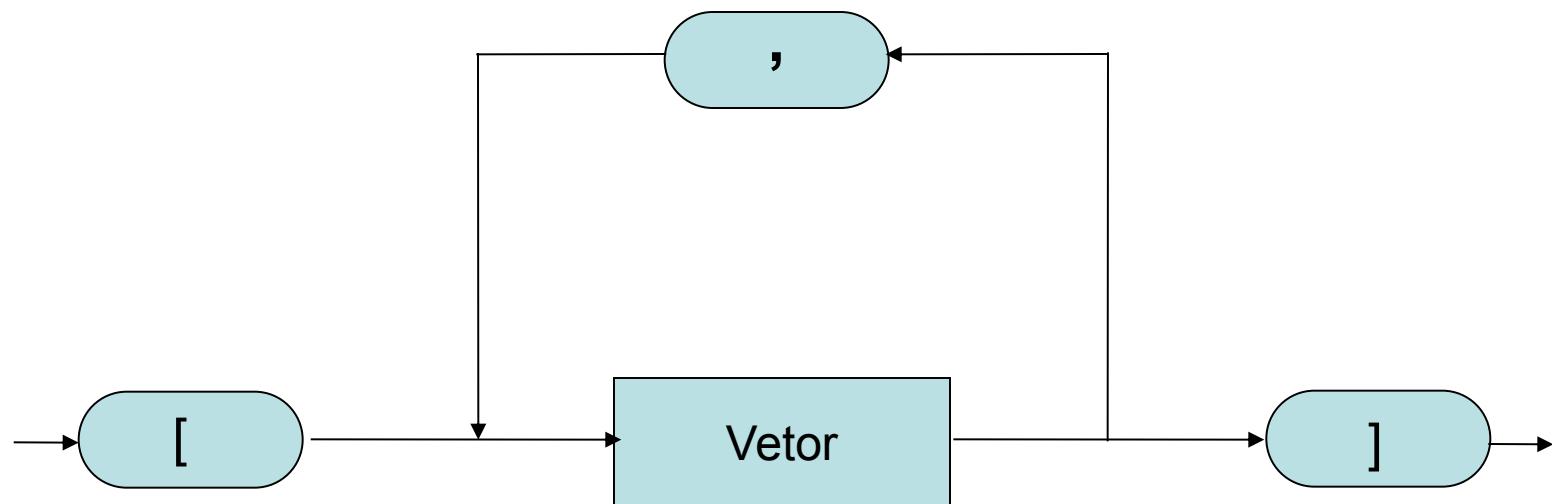
Entrada de Valores de um Vetor: Elemento a Elemento.

```
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]]
```

| | | colunas | | | | |
|--------|---|---------|-------|-------|------|-------|
| | | 0 | 1 | 2 | 3 | 4 |
| linhas | 0 | True | False | False | True | True |
| | 1 | True | True | False | True | False |

O primeiro índice representa a linha e o segundo a coluna. 31

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]]
```

| | | colunas | | | | |
|--------|---|---------|-------|-------|------|-----------------|
| | | 0 | 1 | 2 | 3 | 4 |
| linhas | 0 | True | False | False | True | True |
| | 1 | True | True | False | True | False |
| | | | | | | → celulas[1][2] |

O primeiro índice representa a linha e o segundo a coluna. 32

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)
```

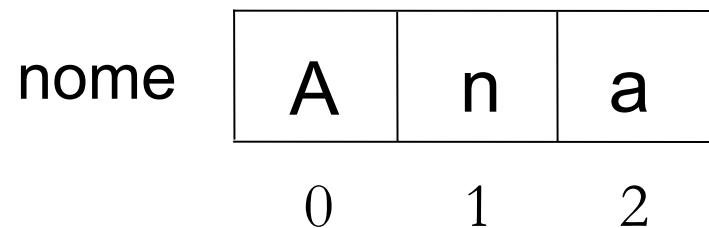
String, ou Cadeia de Caracteres

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

String, ou Cadeia de Caracteres

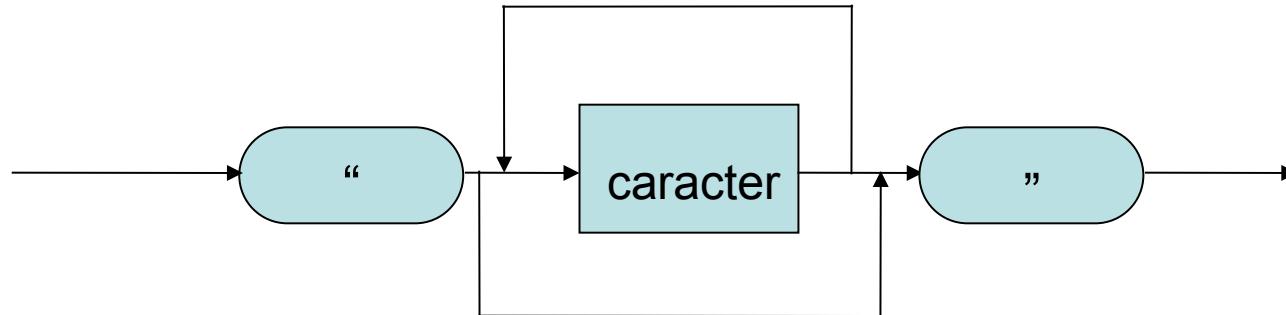
- Um objeto do tipo **str** representa uma cadeia de caracteres, de tamanho e valor imutáveis.
- Declaração e atribuição:

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.

52

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+{} =".format("", 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.

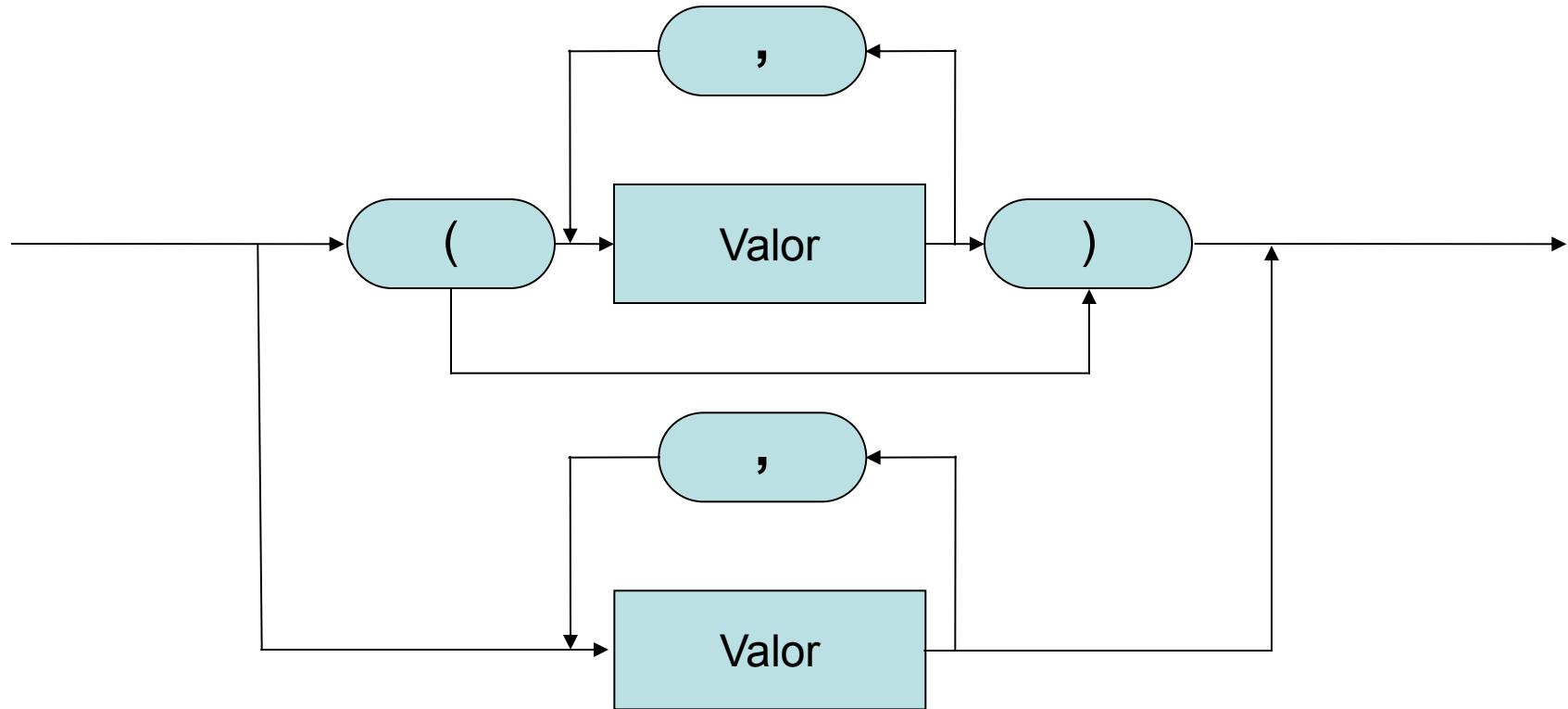
Tupla

- 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.

68

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.

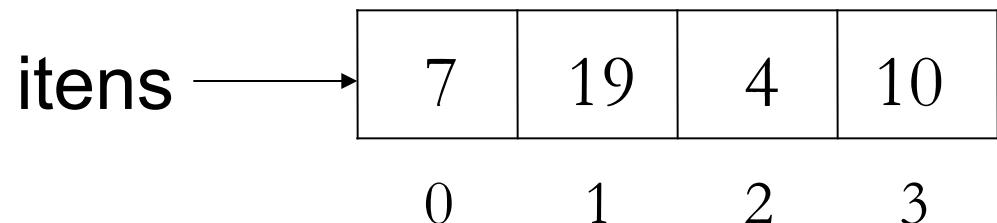
```
valores = ("abacaxi", 500, 4.99)  
print(valores)  
  
valores = "abacaxi", 500, 4.99  
print(valores)
```

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 Itens

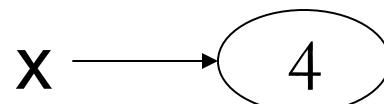
O índice de um elemento de uma tupla pode ser uma variável ou uma expressão.



$$i = 1$$

$$x = \text{itens}[i-1]*2 - \text{itens}[3]$$

$$\begin{array}{r} 14 \\ - 10 \\ \hline 4 \end{array}$$



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: $<$, \leq , \equiv , \neq , $>$ ou \geq

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: $<$, \leq , \equiv , \neq , $>$ ou \geq

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:



Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Aula 5

Conteúdo:

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

Aula 6

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Algoritmos de Busca
 - Busca Simples
 - Busca com Sentinel
 - Busca Binária
- Busca do Menor e Maior Elementos
- Noções de Complexidade de Algoritmos

Algoritmos de Busca

- O problema de busca é caracterizado pela procura de um determinado elemento em um grupo de elementos do mesmo tipo.
 - Exemplos:
 - Encontrar o registro de um cliente entre todos os registros dos clientes de um banco, para que seja possível informar seu saldo.
 - Encontrar o registro de um aluno dentre todos os registros dos alunos de uma universidade, para que seja possível gerar o seu histórico escolar.

Algoritmos de Busca

- O problema de busca é caracterizado pela procura de um determinado elemento em um grupo de elementos do mesmo tipo.
 - Exemplos:
 - Encontrar o registro de um cliente entre todos os registros dos clientes de um banco, para que seja possível informar seu saldo.
 - Encontrar o registro de um aluno dentre todos os registros dos alunos de uma universidade, para que seja possível gerar o seu histórico escolar.
- Algoritmos de busca visam resolver o problema da busca e são, portanto, bastante utilizados em computação.

Algoritmos de Busca

- O problema de busca é caracterizado pela procura de um determinado elemento em um grupo de elementos do mesmo tipo.
 - Exemplos:
 - Encontrar o registro de um cliente entre todos os registros dos clientes de um banco, para que seja possível informar seu saldo.
 - Encontrar o registro de um aluno dentre todos os registros dos alunos de uma universidade, para que seja possível gerar o seu histórico escolar.
- Algoritmos de busca visam resolver o problema da busca e são, portanto, bastante utilizados em computação.
- Necessita-se de algoritmos eficientes de busca.
 - De forma simplificada, um algoritmo eficiente é aquele que realiza sua tarefa em tempo computacional reduzido.

Algoritmos de Busca

- O tempo de execução de um algoritmo de busca depende do tamanho do conjunto de elementos a serem consultados.
 - É mais rápido procurar um elemento em um grupo de 100 do que em um grupo de 100.000.

Algoritmos de Busca

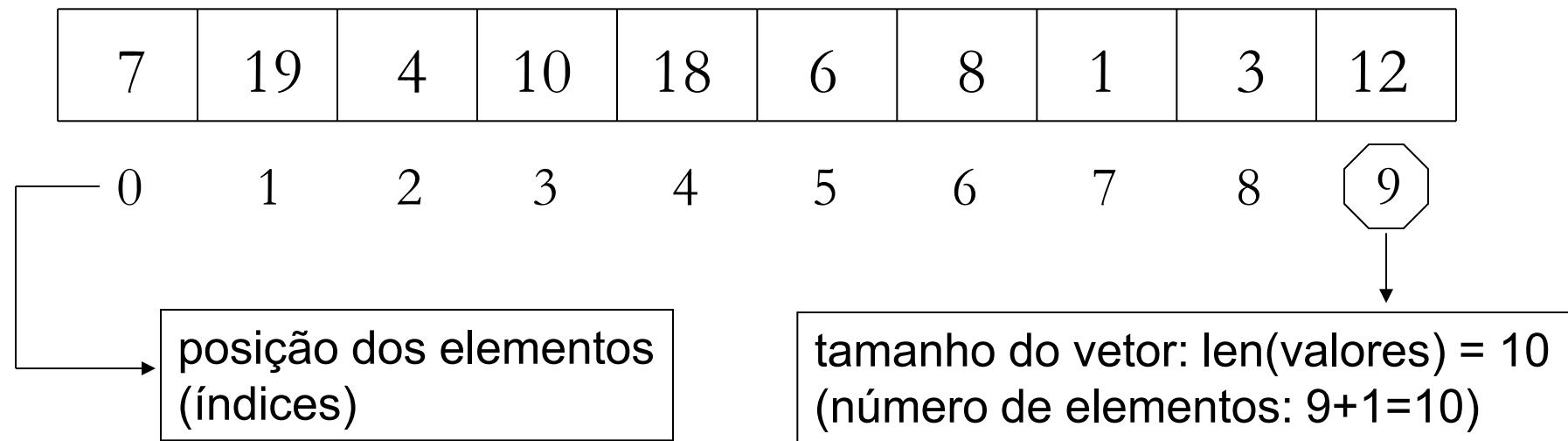
- O tempo de execução de um algoritmo de busca depende do tamanho do conjunto de elementos a serem consultados.
 - É mais rápido procurar um elemento em um grupo de 100 do que em um grupo de 100.000.
- Além disso, a eficiência do processo de busca depende do algoritmo adotado.
 - Exploraremos o conceito de complexidade de algoritmos no sentido de tentar avaliar a eficiência dos algoritmos estudados.

Algoritmos de Busca

- Sem perda de generalidade, o problema será atacado considerando-se que:
 1. Os elementos a serem percorridos são numéricos, distintos e estão armazenados em um vetor;
 2. O número de elementos define o tamanho do vetor;
 3. O elemento procurado pode não estar no vetor (no conjunto de elementos).

Algoritmos de Busca

numeros: contém o grupo de elementos



Exemplo 1) Elemento procurado: 18 Resposta: posição 4

Exemplo 2) Elemento procurado: 5 Resposta: não encontrado

Exemplo

Subprogramas

...

Programa Principal de Busca

```
numeros = [0]*10    # Cria o vetor numeros zerado, com tamanho n = 10
preencher(numeros)
```

dado: o elemento a ser procurado

```
dado = int(input("Escolha valor a ser procurado: "))
```

onde: o local no vetor onde dado foi encontrado ou -1 se não encontrado

```
onde = buscaElemento(numeros, dado)
```

```
escreverResposta(dado, onde)
```

Exemplo (continuação)

```
# Subprogramas
def preencher(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return None
def buscaElemento(valores, procurado):
    ...
def escreverResposta(valor, pos):
    if pos<0:
        print(valor,"não foi encontrado")
    else:
        print(valor, "foi encontrado na posição", pos)
    return
# Programa Principal de Busca
numeros = [0]*10
preencher(numeros)
dado = int(input("Escolha valor a ser procurado: "))
onde = buscaElemento(numeros, dado)
escreverResposta(dado, onde)
```

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
|---|----|---|----|----|---|---|---|---|----|



procurado: 18
local: -1

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: -1
índice: 0

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: -1
índice: 1

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: -1
índice: 2

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

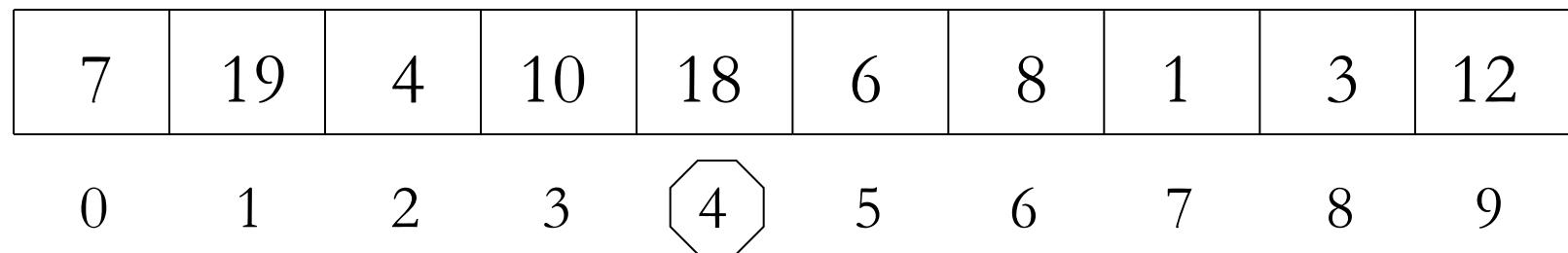
| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: -1
índice: 3

Busca Simples (utilizando for)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos



procurado: 18
local: 4
índice: 4

Busca Simples (utilizando for)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: 4
índice: 5

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: 4
índice: 6

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: 4
índice: 7

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: 4
índice: 8

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos

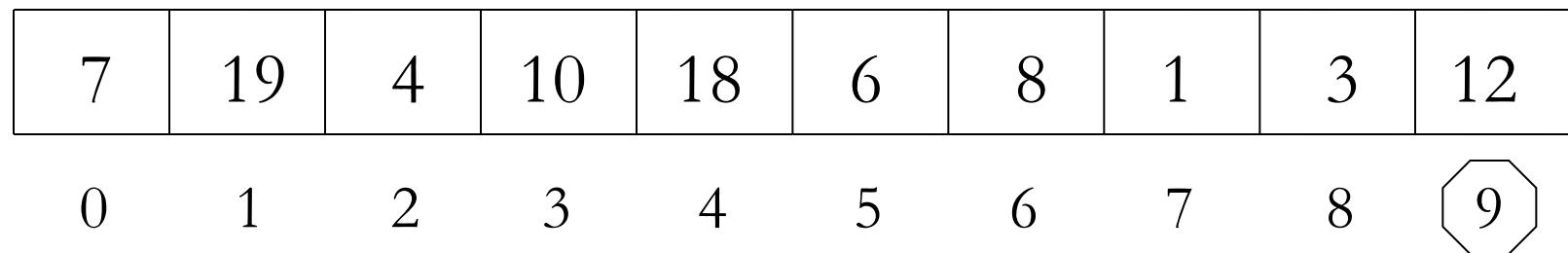
| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 18
local: 4
índice: 9

Busca Simples (utilizando *for*)

Todas as posições do vetor são comparadas com o valor procurado.

valores: contém o grupo de elementos



procurado: 18
local: 4
índice: 9

Resultado

Subprogramas

```
def preencher(valores):
```

```
    ...
```

```
def buscaElemento(valores, procurado):
```

```
    local = -1
```

```
    for indice in range(len(valores)):
```

```
        if valores[indice]==procurado:
```

```
            local = indice
```

```
    return local
```

```
def escreverResposta(valor, pos):
```

```
    ...
```

Programa Principal de Busca

```
numeros = [0]*10
```

```
preencher(numeros)
```

```
dado = int(input("Escolha valor a ser procurado: "))
```

```
onde = buscaElemento(numeros, dado)
```

```
escreverResposta(dado, onde)
```

Busca Simples (utilizando *for*)

- Todas as posições do vetor são comparadas com o valor procurado, mesmo quando este é encontrado antes do fim do vetor.

Busca Simples (utilizando *for*)

- Todas as posições do vetor são comparadas com o valor procurado, mesmo quando este é encontrado antes do fim do vetor.
- Nitidamente, esta é uma busca ineficiente.

Busca Simples (utilizando *for*)

- Todas as posições do vetor são comparadas com o valor procurado, mesmo quando este é encontrado antes do fim do vetor.
- Nitidamente, esta é uma busca ineficiente.
- O algoritmo avalia necessariamente **n** elementos, onde **n** é o tamanho do vetor. Portanto, sua complexidade é da ordem de **n**, representado por $O(n)$.

Busca Simples (utilizando **for** com saída rápida)

- Utilizando-se sub-programação, não se considera uma má prática de programação se realizar um **return** dentro de uma repetição.
- O subprograma *buscaElemento* pode ter sua eficiência melhorada quando encontra o elemento no vetor, não necessitando repetir todos os índices previstos no **for**:

```
def buscaElemento(valores, procurado):
    for indice in range(len(valores)):
        if valores[indice]==procurado:
            return indice          # retorna ao encontrar o local
    return -1                      # retorna -1 se terminar sem encontrar
```

Busca Simples (utilizando while)

Neste algoritmo, o processo de busca é interrompido quando o elemento procurado é encontrado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
|---|----|---|----|----|---|---|---|---|----|



procurado: 4
local: -1
índice: 0

Busca Simples (utilizando while)

Neste algoritmo, o processo de busca é interrompido quando o elemento procurado é encontrado.

valores: contém o grupo de elementos

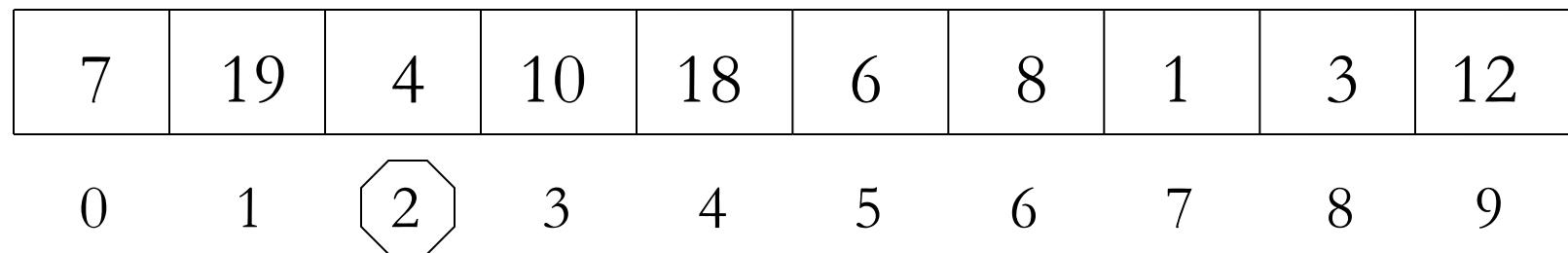
| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 4
local: -1
índice: 1

Busca Simples (utilizando while)

Neste algoritmo, o processo de busca é interrompido quando o elemento procurado é encontrado.

valores: contém o grupo de elementos



procurado: 4
local: -1
índice: 2

Busca Simples (utilizando while)

Neste algoritmo, o processo de busca é interrompido quando o elemento procurado é encontrado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

procurado: 4
local: 2
índice: 2

Busca Simples (utilizando while)

Neste algoritmo, o processo de busca é interrompido quando o elemento procurado é encontrado.

valores: contém o grupo de elementos

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
|---|----|---|----|----|---|---|---|---|----|

0 1 2 3 4 5 6 7 8 9



procurado: 4
local: 2
índice: 10

Resultado

Subprogramas

```
def preencher(valores):
```

```
...
```

```
def buscaElemento(valores, procurado):      # primeira solução
```

```
    local = -1
```

```
    indice = 0
```

```
    while indice < len(valores):
```

```
        if valores[indice] != procurado:
```

```
            indice = indice + 1
```

```
        else:
```

```
            local = indice
```

```
            indice = len(valores)
```

```
    return local
```

```
def escreverResposta(valor, pos):
```

```
...
```

Programa Principal de Busca

```
numeros = [0]*10
```

```
preencher(numeros)
```

```
dado = int(input("Escolha valor a ser procurado: "))
```

```
onde = buscaElemento(numeros, dado)
```

```
escreverResposta(dado, onde)
```

Subprogramas

```
def preencher(valores):
```

```
    ...
```

```
def buscaElemento(valores, procurado):      # segunda solução
```

```
    indice = 0
```

```
    while indice < len(valores):
```

```
        if valores[indice] != procurado:
```

```
            indice = indice + 1
```

```
        else:
```

```
            return indice    # retorna ao encontrar local
```

```
    return -1                # retorna -1 se terminar sem encontrar
```

```
def escreverResposta(valor, pos):
```

```
    ...
```

Programa Principal de Busca

```
numeros = [0]*10
```

```
preencher(numeros)
```

```
dado = int(input("Escolha valor a ser procurado: "))
```

```
onde = buscaElemento(numeros, dado)
```

```
escreverResposta(dado, onde)
```

Busca Simples (utilizando while)

- Em média, este algoritmo executa $n/2$ vezes o corpo da repetição **while**.

Busca Simples (utilizando while)

- Em média, este algoritmo executa $n/2$ vezes o corpo da repetição **while**.
- Em algumas vezes, o dado será encontrado logo na primeira posição, em outras, na última posição.

Busca Simples (utilizando while)

- Em média, este algoritmo executa $n/2$ vezes o corpo da repetição **while**.
- Em algumas vezes, o dado será encontrado logo na primeira posição, em outras, na última posição.
- No pior caso, o algoritmo avalia n elementos. Portanto, sua complexidade também é da ordem de n , representado por $O(n)$.

Busca com Sentinela (utilizando *while*)

- O algoritmo anterior poderia executar menos comparações, se não houvesse a necessidade de evitar ultrapassar o fim do vetor, caso o elemento procurado não se encontre no conjunto.

Busca com Sentinel (utilizando while)

- O algoritmo anterior poderia executar menos comparações, se não houvesse a necessidade de evitar ultrapassar o fim do vetor, caso o elemento procurado não se encontre no conjunto.
- Propõe-se então alocar uma posição a mais no vetor e inserir forçadamente o elemento procurado nesta posição.

Busca com Sentinel (utilizando while)

- O algoritmo anterior poderia executar menos comparações, se não houvesse a necessidade de evitar ultrapassar o fim do vetor, caso o elemento procurado não se encontre no conjunto.
- Propõe-se então alocar uma posição a mais no vetor e inserir forçadamente o elemento procurado nesta posição.
- Desta forma, necessariamente o elemento procurado será encontrado. Se for encontrado na posição $n+1$, significa que o elemento não pertence ao conjunto original.

Busca com Sentinel (utilizando while)

- O algoritmo anterior poderia executar menos comparações, se não houvesse a necessidade de evitar ultrapassar o fim do vetor, caso o elemento procurado não se encontre no conjunto.
- Propõe-se então alocar uma posição a mais no vetor e inserir forçadamente o elemento procurado nesta posição.
- Desta forma, necessariamente o elemento procurado será encontrado. Se for encontrado na posição $n+1$, significa que o elemento não pertence ao conjunto original.
- Apesar de executar menos comparações, o algoritmo avalia, no pior caso, $n+1$ elementos. Portanto, sua complexidade também é da ordem de n , representado por $O(n)$.

Subprogramas

```
def preencher(valores):
```

```
...
```

```
def buscaElemento(valores, procurado):
```

```
    indice = 0
```

```
    while valores[indice]!=procurado:
```

```
        indice = indice + 1
```

```
    if indice==len(valores)-1:
```

```
        local = -1          # local do sentinel, informa que não achou
```

```
    else:
```

```
        local = indice
```

```
return local
```

```
def escreverResposta(valor, pos):
```

```
...
```

Programa Principal de Busca

```
numeros = [0]*10
```

```
preencher(numeros)
```

```
dado = int(input("Escolha valor a ser procurado: "))
```

```
numeros.append(dado)    # coloca o procurado no final: o sentinel
```

```
onde = buscaElemento(numeros, dado)
```

```
escreverResposta(dado, onde)
```

Busca Binária

- Os algoritmos apresentados até o momento foram projetados sem levar em consideração se os elementos do vetor encontram-se ordenados ou não.

Busca Binária

- Os algoritmos apresentados até o momento foram projetados sem levar em consideração se os elementos do vetor encontram-se ordenados ou não.
- Caso os elementos se encontrem ordenados, algoritmos mais eficientes podem ser implementados.

Busca Binária

- Os algoritmos apresentados até o momento foram projetados sem levar em consideração se os elementos do vetor encontram-se ordenados ou não.
- Caso os elementos se encontrem ordenados, algoritmos mais eficientes podem ser implementados.
- No algoritmo chamado busca binária, cada passo divide o espaço de busca em dois grupos até encontrar o elemento sendo procurado.

Busca Binária

Os **n** elementos encontram-se ordenados no vetor.

valores

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 13 | 14 | 15 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

↑
inicio

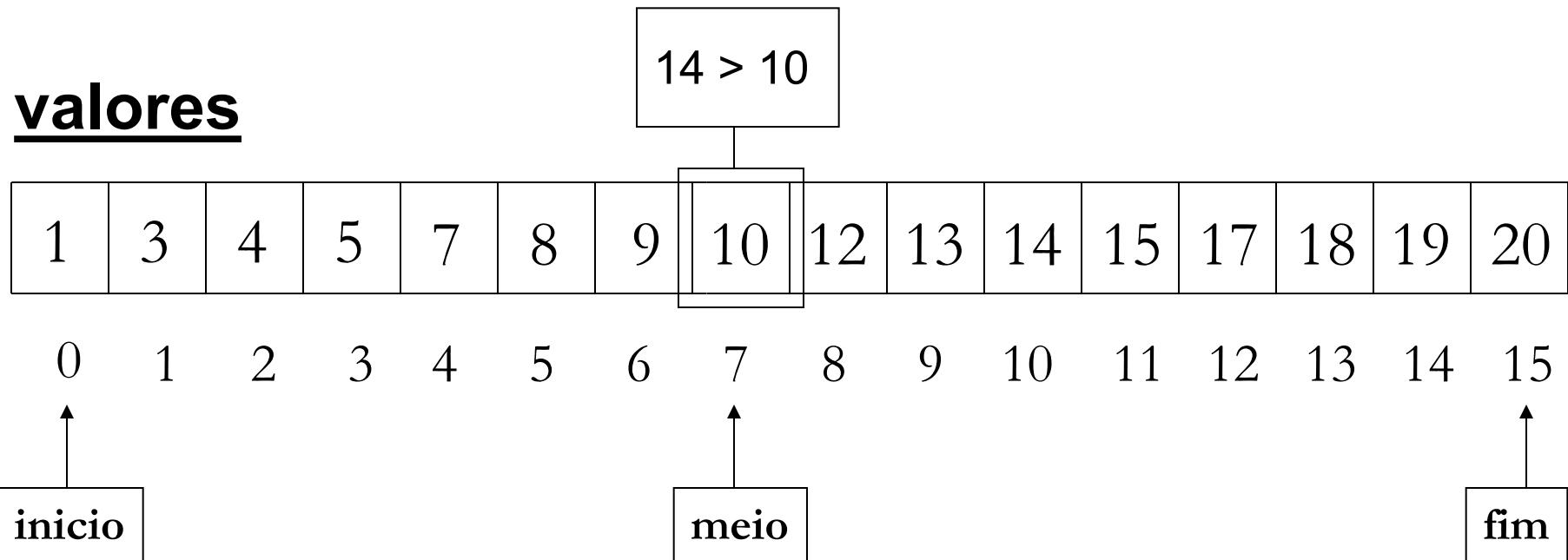
↑
fim

procurado: 14

Busca Binária

procurado: 14

valores



Número de elementos comparados: 1

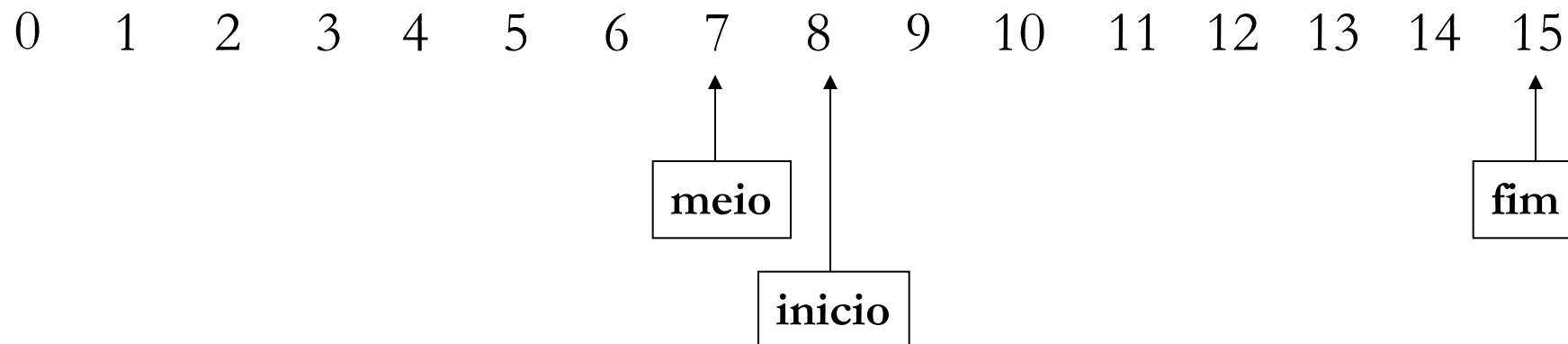
47

Busca Binária

procurado: 14

valores

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 13 | 14 | 15 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|



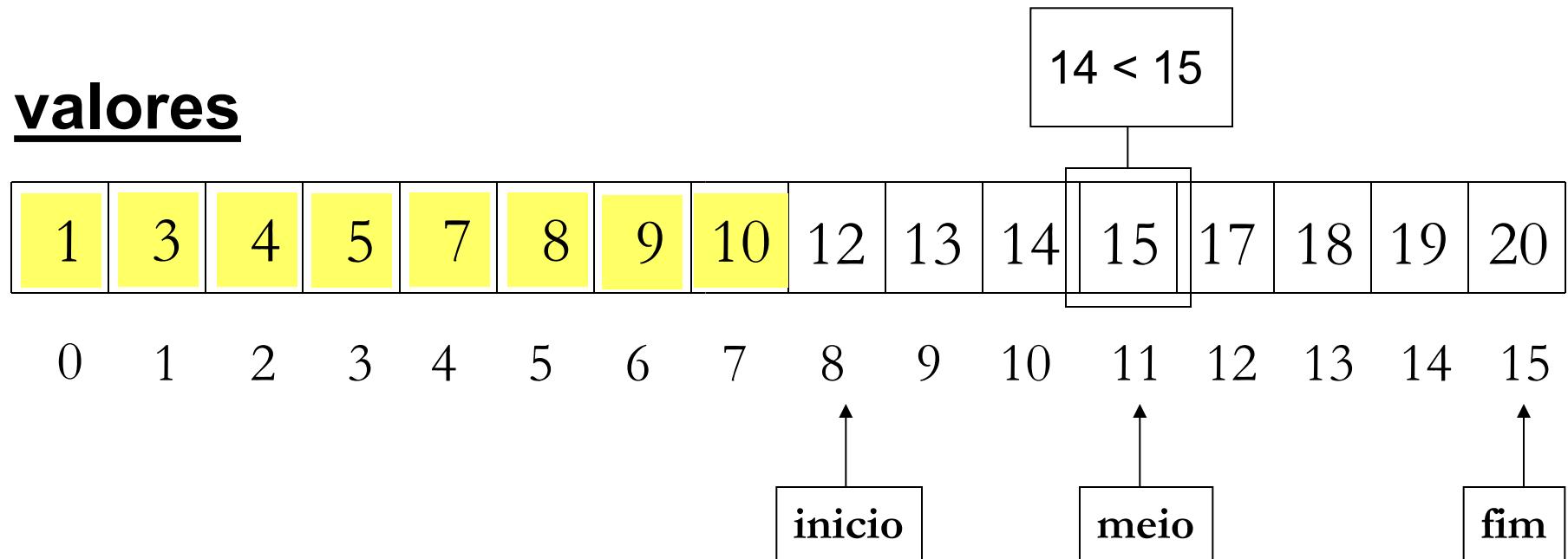
Número de elementos comparados: 1

48

Busca Binária

procurado: 14

valores



Número de elementos comparados: 2

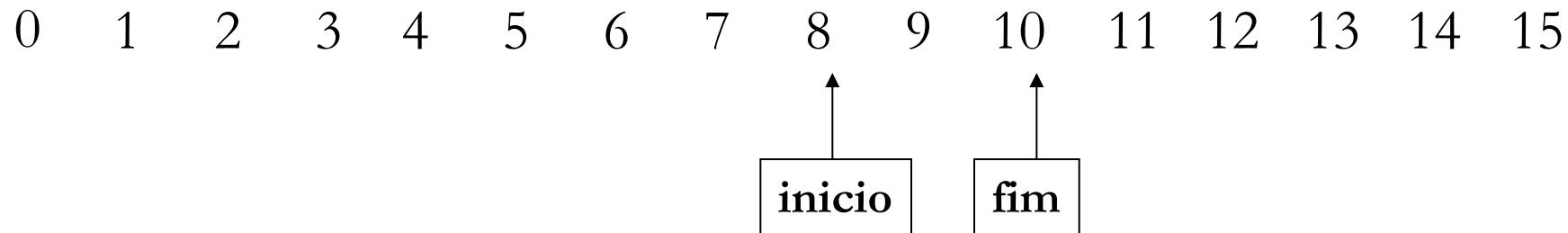
49

Busca Binária

procurado: 14

valores

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 13 | 14 | 15 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|



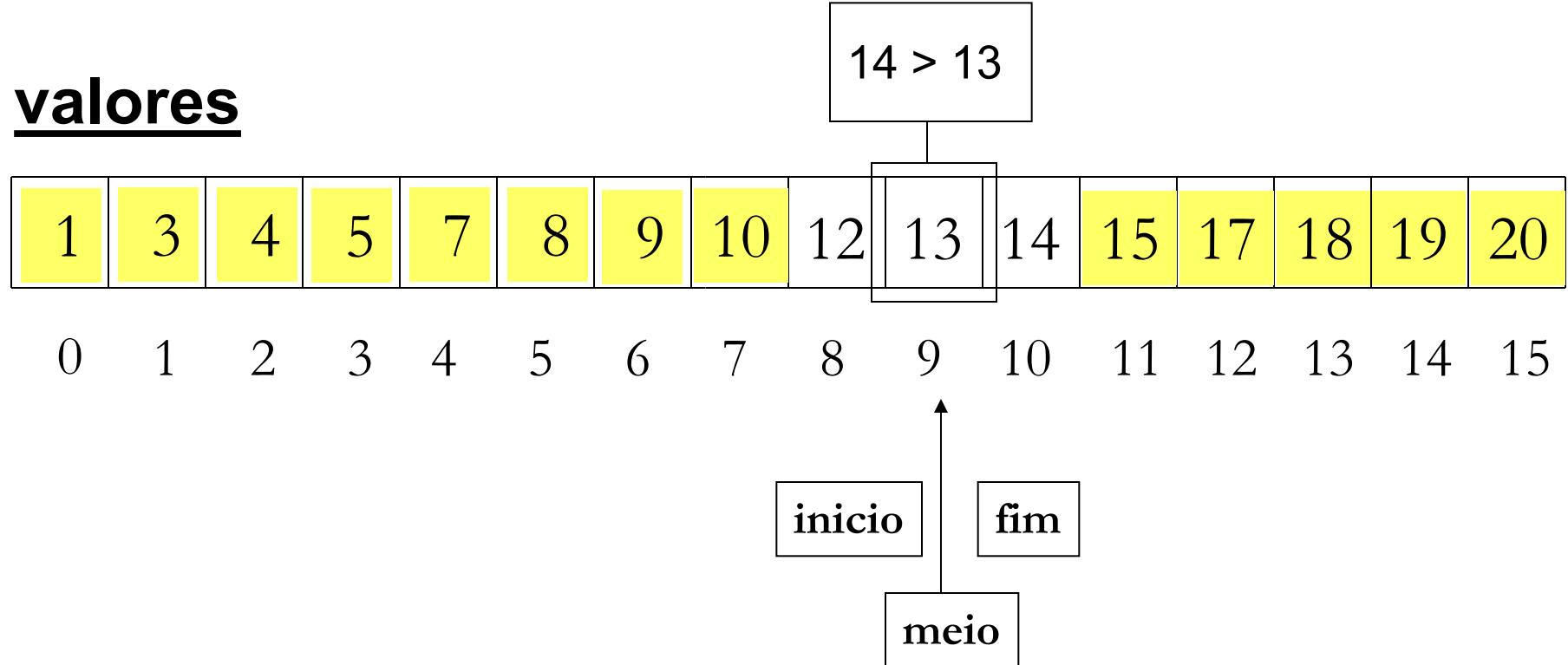
Número de elementos comparados: 2

50

Busca Binária

procurado: 14

valores



Número de elementos comparados: 3

51

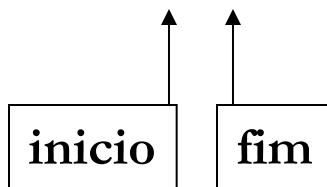
Busca Binária

procurado: 14

valores

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| 1 | 3 | 4 | 5 | 7 | 8 | 9 | 10 | 12 | 13 | 14 | 15 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



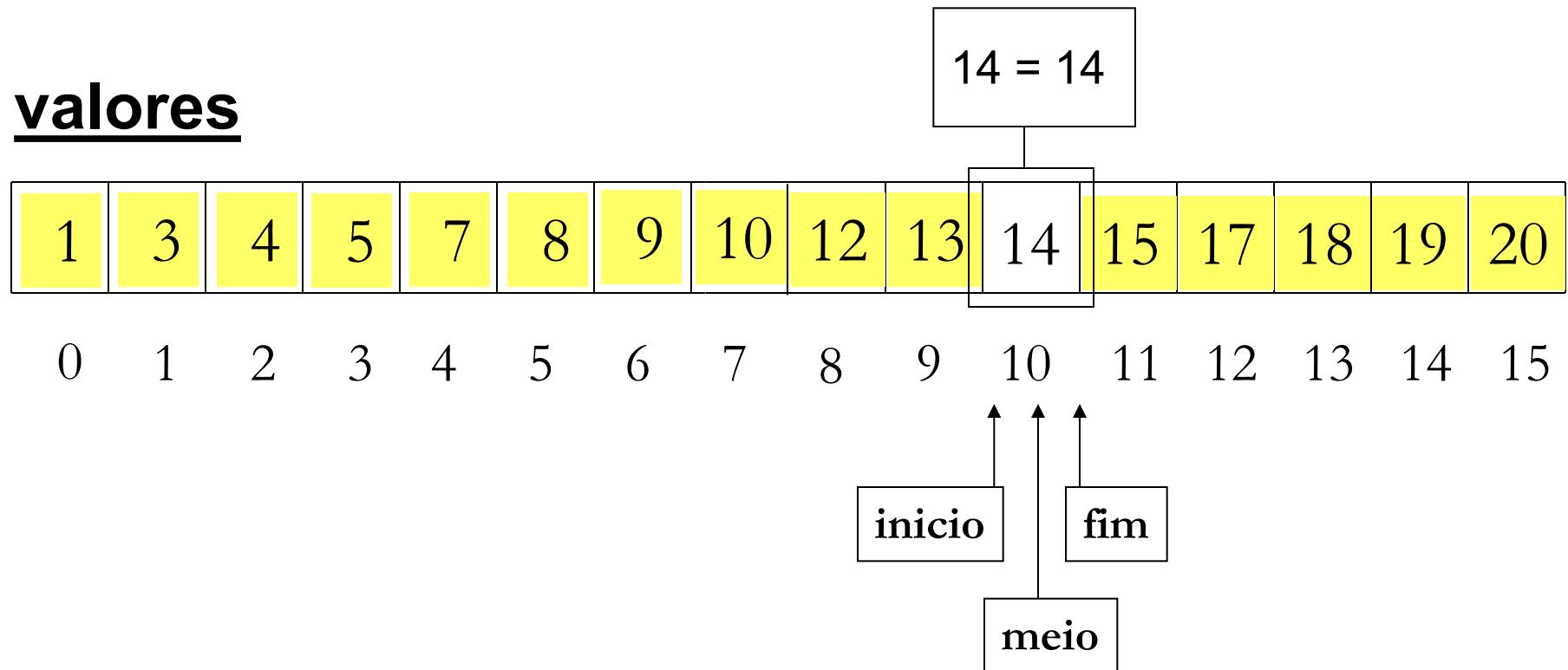
Número de elementos comparados: 3

52

Busca Binária

procurado: 14

valores



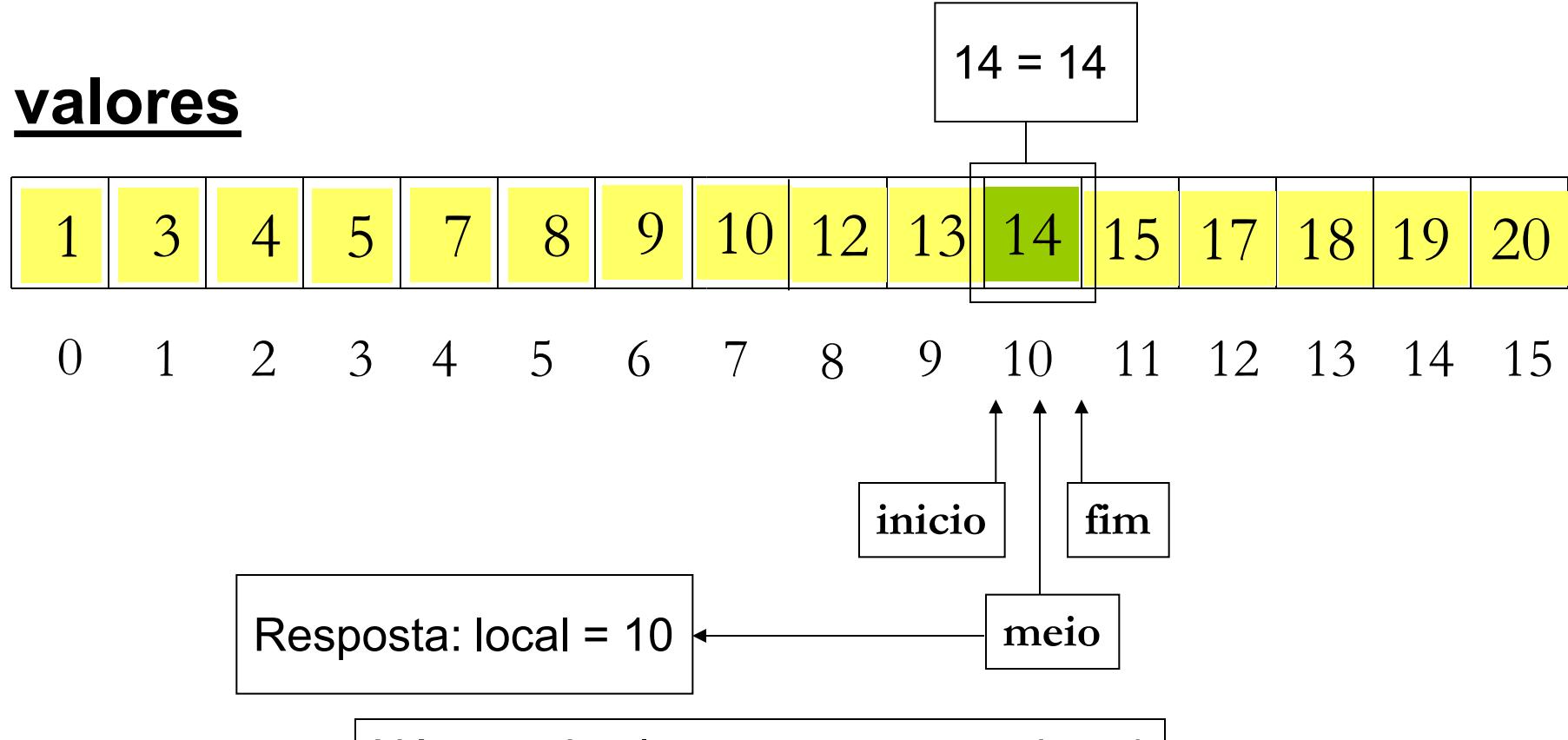
Número de elementos comparados: 4

53

Busca Binária

procurado: 14

valores



Número de elementos comparados: 4

Busca Binária

```
def buscaElemento(valores, procurado):
    inicio = 0
    fim = len(valores)-1
    meio = (inicio + fim) // 2
    while (inicio < fim) and (procurado != valores[meio]):
        if procurado > valores[meio]:
            inicio = meio + 1
        else:
            fim = meio - 1
            meio = (inicio + fim) // 2
    if procurado != valores[meio]:
        local = -1
    else:
        local = meio
    return local
```

Busca Binária

- Na busca binária, a cada passo, divide-se o espaço de busca em dois até que o elemento procurado seja encontrado.

Busca Binária

- Na busca binária, a cada passo, divide-se o espaço de busca em dois até que o elemento procurado seja encontrado.
- Considerando-se um vetor de 16 posições, no máximo 4 divisões podem ser feitas. E portanto, no máximo 4 elementos do vetor serão comparados com o elemento procurado.

Busca Binária

- Na busca binária, a cada passo, divide-se o espaço de busca em dois até que o elemento procurado seja encontrado.
- Considerando-se um vetor de 16 posições, no máximo 4 divisões podem ser feitas. E portanto, no máximo 4 elementos do vetor serão comparados com o elemento procurado.
- Observe que se o vetor for de 32 posições (vetor duplicado), no máximo 5 elementos do vetor serão acessados (apenas um a mais). Ou ainda, se o vetor tiver 32.000 posições, apenas 15 avaliações serão necessárias.

Busca Binária

- Na busca binária, a cada passo, divide-se o espaço de busca em dois até que o elemento procurado seja encontrado.
- Considerando-se um vetor de 16 posições, no máximo 4 divisões podem ser feitas. E portanto, no máximo 4 elementos do vetor serão comparados com o elemento procurado.
- Observe que se o vetor for de 32 posições (vetor duplicado), no máximo 5 elementos do vetor serão acessados (apenas um a mais). Ou ainda, se o vetor tiver 32.000 posições, apenas 15 avaliações serão necessárias.
- Na prática, se o vetor tiver n posições, a busca binária avaliará, no pior caso, $\log_2(n)$ elementos. Portanto, sua complexidade é da ordem de $\log(n)$, representado por $O(\log(n))$.

Busca do Menor e Maior Elementos

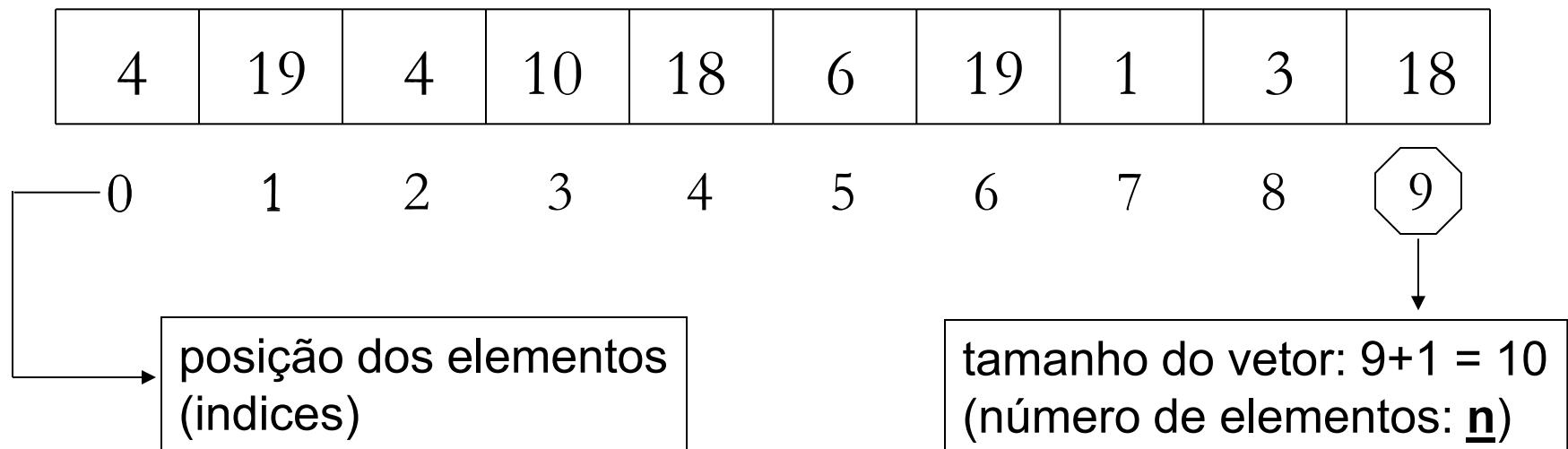
- Este problema é caracterizado pela procura do menor e do maior elementos em um grupo de elementos do mesmo tipo.

Busca do Menor e Maior Elementos

- Este problema é caracterizado pela procura do menor e do maior elementos em um grupo de elementos do mesmo tipo.
- Sem perda de generalidade, o problema será atacado considerando-se que:
 - Os elementos a serem percorridos são numéricos e estão armazenados em um vetor;
 - Estes elementos podem não ser distintos;
 - O número de elementos define o tamanho do vetor.

Busca do Menor e Maior Elementos

valores: contém o grupo de elementos



Resposta: Menor Elemento: 1
 Maior Elemento: 19

Busca do Menor e Maior Elementos

Subprogramas

```
def preencher(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return
def buscarMenorMaiorElementos(valores):
    ...
def escrever(infos):
    print("O menor elemento =", infos[0], "e o maior elemento =", infos[1])
    return None
```

Programa Principal de Busca do Menor e do Maior Elementos

```
numeros = [0]*10
preencher(numeros)
extremos = buscarMenorMaiorElementos(numeros)
escrever(extremos)
```

Busca do Menor e Maior Elementos (continuação)

Subprogramas

```
def preencher(valores):
```

```
    ...
```

```
def buscarMenorMaiorElementos(valores):
```

```
    menor = valores[0]
```

```
    maior = valores[0]
```

```
    for indice in range(1,len(valores)):
```

```
        if menor > valores[indice]:
```

```
            menor = valores[indice]
```

```
        elif maior < valores[indice]:
```

```
            maior = valores[indice]
```

```
    return [menor, maior]
```

```
def escrever(infos):
```

```
    ...
```

Programa Principal de Busca do Menor e do Maior Elementos

```
numeros = [0]*10
```

```
preencher(numeros)
```

```
extremos = buscarMenorMaiorElementos(numeros)
```

```
escrever(extremos)
```

Busca do Menor e Maior Elementos

- O algoritmo apresentado encontra o menor e o maior elementos em um vetor de n elementos.

Busca do Menor e Maior Elementos

- O algoritmo apresentado encontra o menor e o maior elementos em um vetor de n elementos.
- Dado que cada um dos n elementos é avaliado necessariamente uma vez, a sua complexidade é da ordem de n , representado por $O(n)$.

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:



Aula 6

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Algoritmos de Busca
 - Busca Simples
 - Busca com Sentinel
 - Busca Binária
- Busca do Menor e Maior Elementos
- Noções de Complexidade de Algoritmos

Aula 7

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Moda
- Algoritmos de Ordenação
 - Método da Seleção (*SelectionSort*)
 - Método da Bolha (*BubbleSort*)
 - Método da Partição (*QuickSort*)
- Noções de Complexidade de Algoritmos

Busca pela Moda de um Vetor

- A moda de um vetor é o elemento que aparece com mais frequência neste vetor.

Busca pela Moda de um Vetor

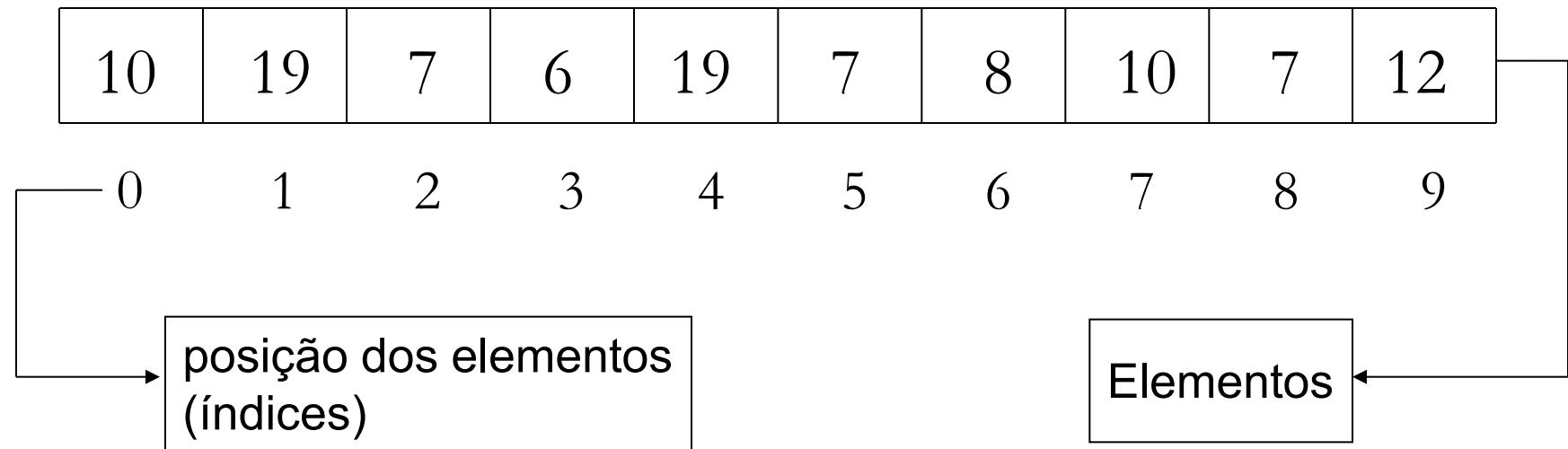
- A moda de um vetor é o elemento que aparece com mais frequência neste vetor.
- O problema da busca pela moda consiste em encontrar este elemento no vetor.

Busca pela Moda de um Vetor

- A moda de um vetor é o elemento que aparece com mais frequência neste vetor.
- O problema da busca pela moda consiste em encontrar este elemento no vetor.
- Sem perda de generalidade, o problema será atacado considerando-se que:
 - Os elementos do vetor são numéricos;
 - Mais de um elemento pode aparecer o mesmo número de vezes e ser moda. Nesse caso, o que aparecer primeiro no vetor será apresentado como resposta.

Busca pela Moda de um Vetor

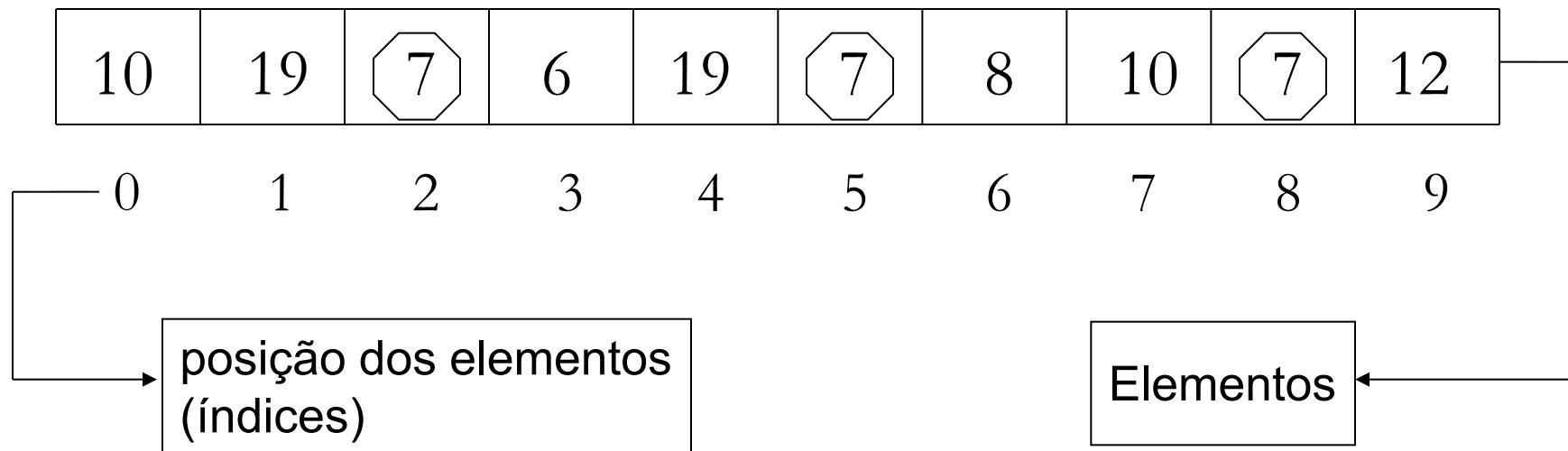
valores: contém um grupo de inteiros



Qual é a Moda deste vetor **valores**?

Busca pela Moda de um Vetor

valores: contém um grupo de inteiros



Moda deste vetor **valores**: elemento 7 (que ocorre 3 vezes).

Busca pela Moda de um Vetor (Exemplo)

Subprogramas

```
def preenche(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return None
```

```
def buscaModa(valores):
    ...
```

Programa Principal de Busca o Elemento da Moda

```
numeros = [0]*10
preenche(numeros)
moda = buscaModa(numeros)
```

```
print("A moda do vetor elemento da é:", moda)
```

Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores

| | | | | | | | | | | |
|--|----|----|---|---|----|---|---|----|---|----|
| | 10 | 19 | 7 | 6 | 19 | 7 | 8 | 10 | 7 | 12 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

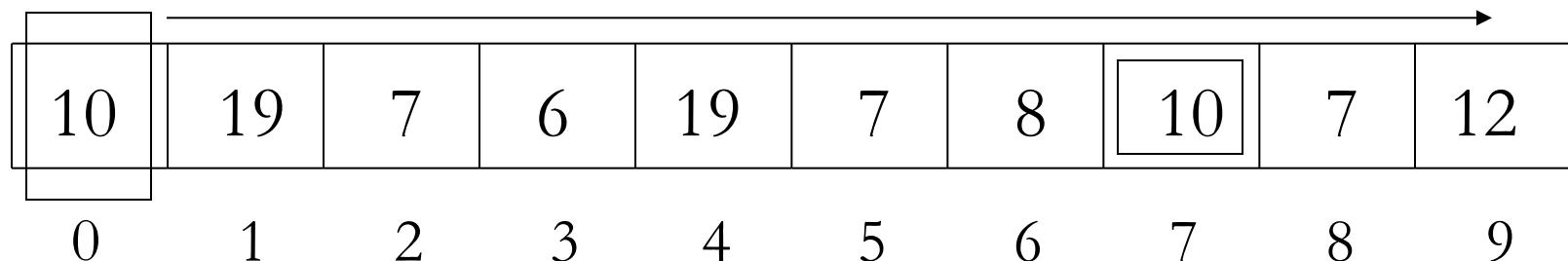
auxiliar (contém a frequência de cada elemento do valores)

| | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

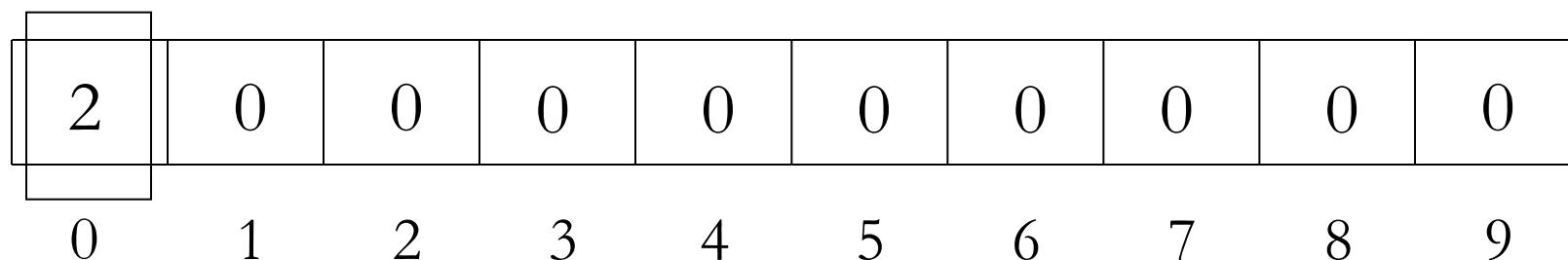
Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores



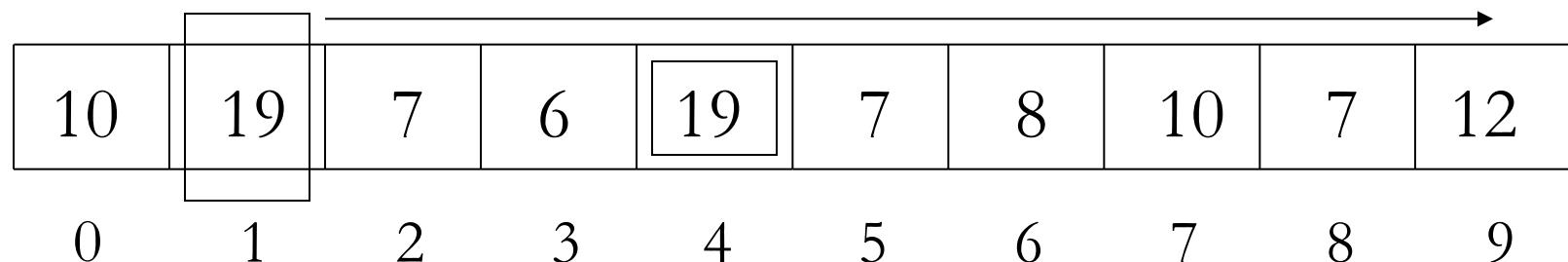
auxiliar (contém a frequência de cada elemento do valores)



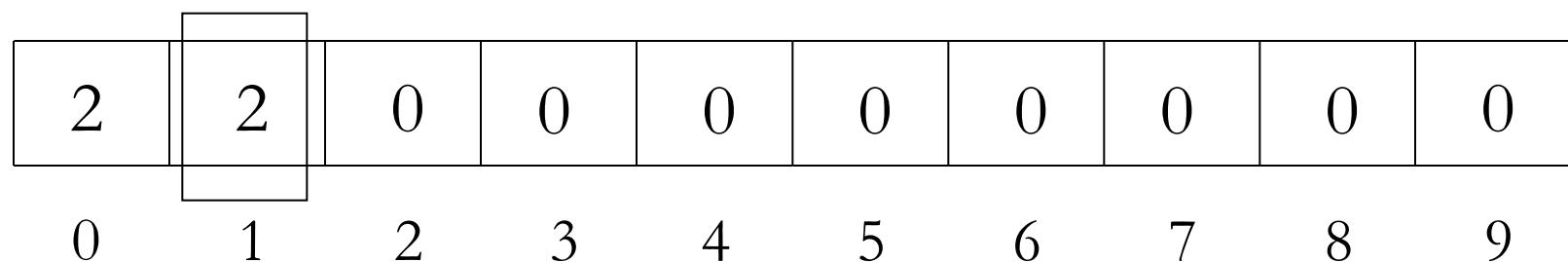
Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores



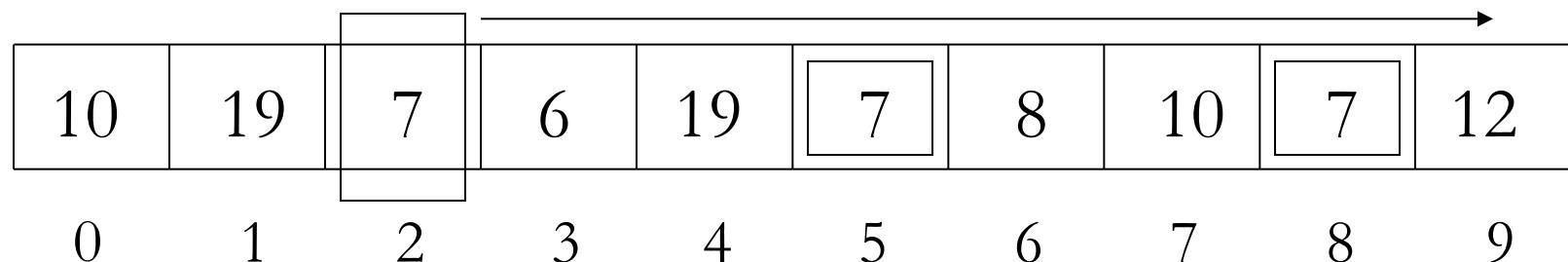
auxiliar (contém a frequência de cada elemento do valores)



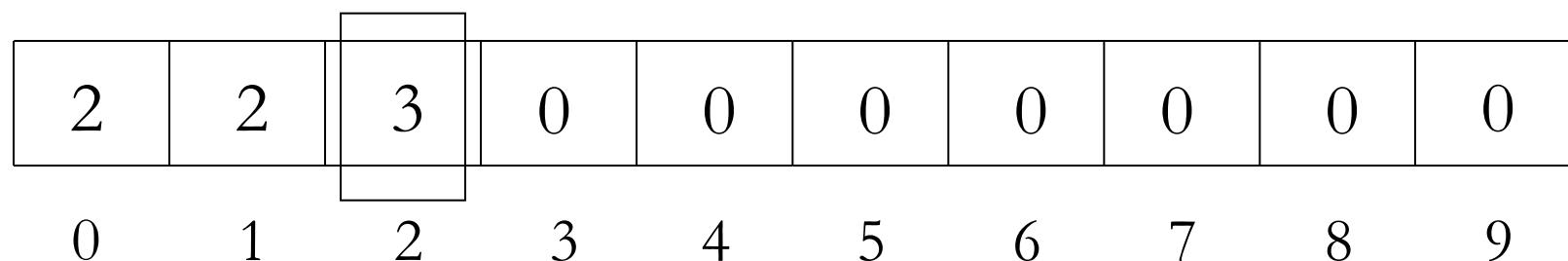
Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores



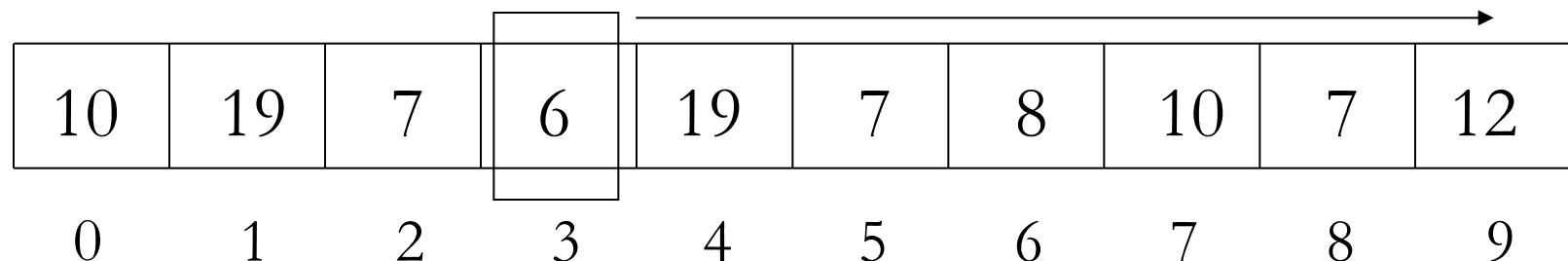
auxiliar (contém a frequência de cada elemento do valores)



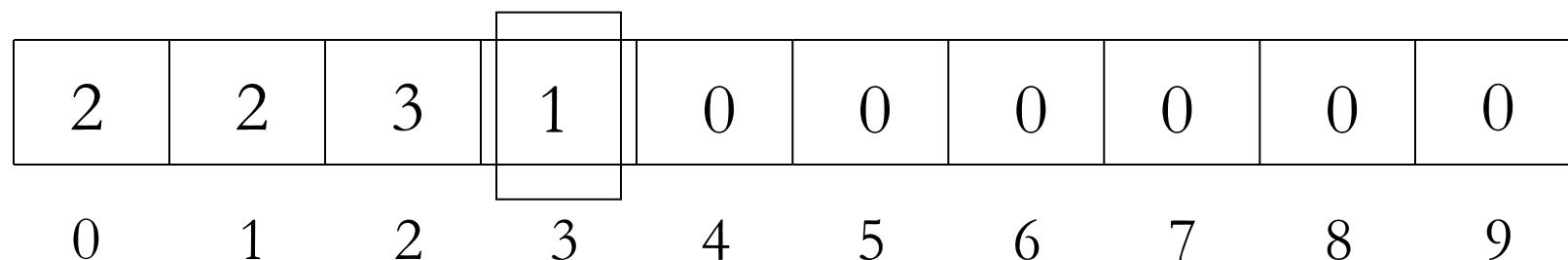
Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores



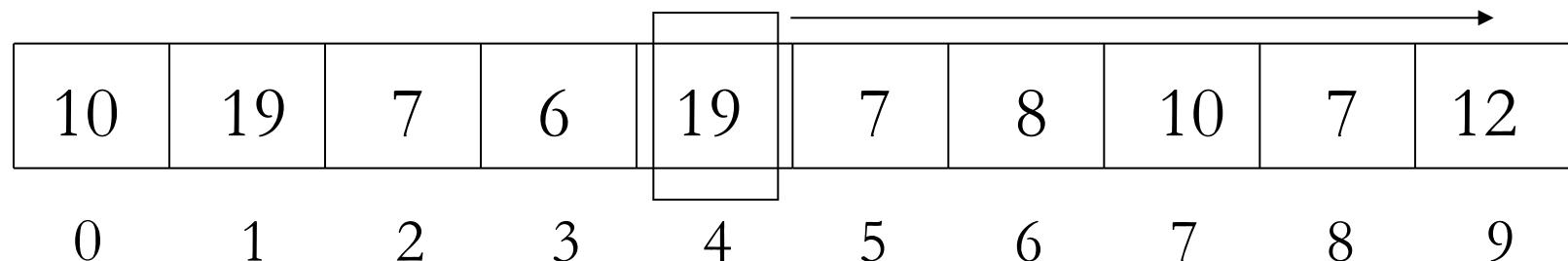
auxiliar (contém a frequência de cada elemento do valores)



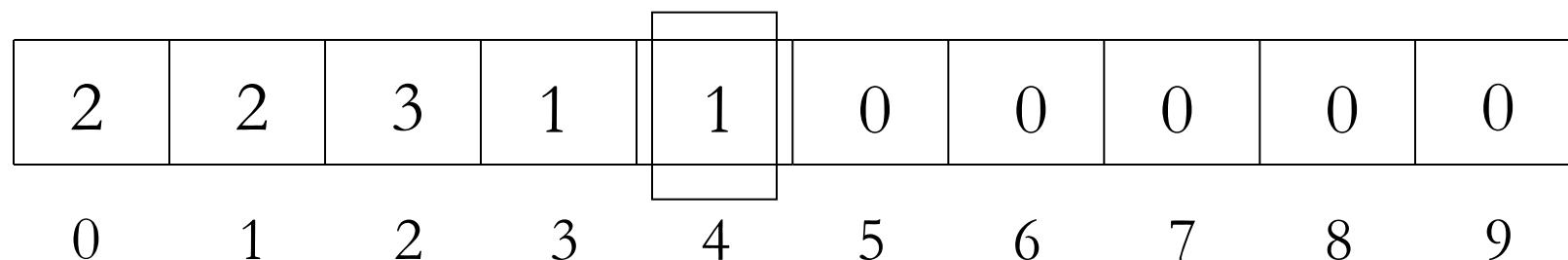
Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores



auxiliar (contém a frequência de cada elemento do valores)

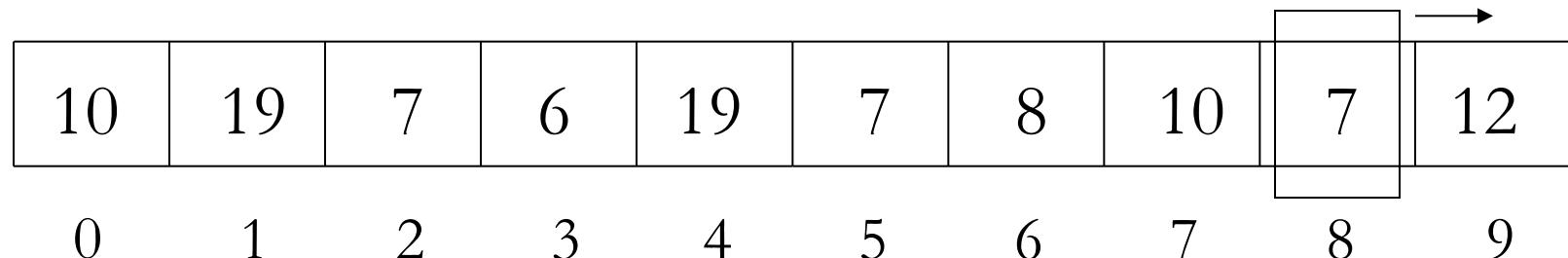


Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

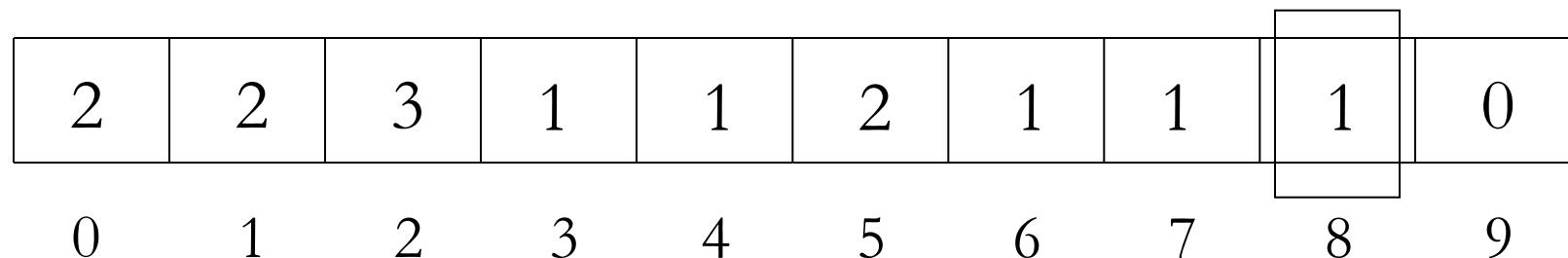
valores

■ ■ ■



auxiliar

■ ■ ■



Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.

valores

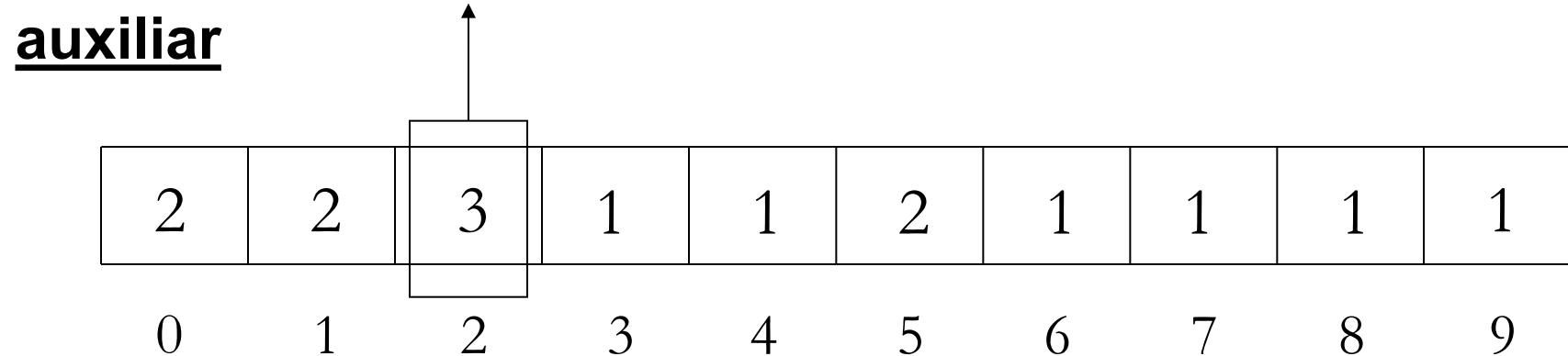
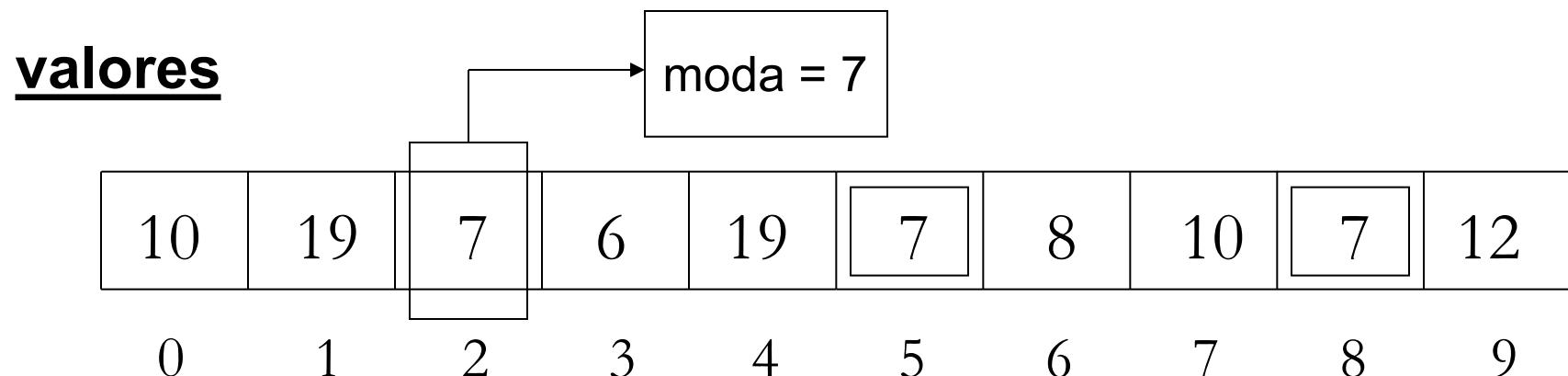
| | | | | | | | | | |
|----|----|---|---|----|---|---|----|---|----|
| 10 | 19 | 7 | 6 | 19 | 7 | 8 | 10 | 7 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

auxiliar

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Busca Simples pela Moda

Todas as posições do vetor são comparadas com os elementos seguintes do vetor.



```

# Subprogramas
def preenche(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return None

def buscaModa(valores):
    auxiliar = [0]*len(valores)
    for indice in range(len(valores)):          # calcula as frequências
        auxiliar[indice] = 1
        for varre in range(indice+1, len(valores)):
            if valores[varre]==valores[indice]:
                auxiliar[indice] += 1
    ondeModa = 0
    for i in range(1, len(auxiliar)):          # localiza a maior frequência
        if auxiliar[i] > auxiliar[ondeModa]:
            ondeModa = i
    return valores[ondeModa]                  # retorna o valor da moda

```

Programa Principal de Busca o Elemento da Moda

```

numeros = [0]*10
preenche(numeros)
moda = buscaModa(numeros)
print("A moda do vetor elemento da é:", moda)

```

Subprogramas

```
def preenche(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return None

def buscaModa(valores):
    auxiliar = [0]*len(valores)
    for indice in range(len(valores)):          # calcula as frequências
        auxiliar[indice] = 1
        for varre in range(indice+1, len(valores)):
            if valores[varre]==valores[indice]:
                auxiliar[indice] += 1
    ondeModa = 0
    for i in range(1, len(auxiliar)):           # localiza a maior frequência
        if auxiliar[i] > auxiliar[ondeModa]:
            ondeModa = i
    return valores[ondeModa]                   # retorna o valor da moda
```

Programa Principal de Busca o Elemento da Moda

```
numeros = [0]*10
preenche(numeros)
moda = buscaModa(numeros)
print("A moda do vetor elemento da é:", moda)
```

Subprogramas

```
def preenche(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return None

def buscaModa(valores):
    auxiliar = [0]*len(valores)
    for indice in range(len(valores)):          # calcula as frequências
        auxiliar[indice] = 1
        for varre in range(indice+1, len(valores)):
            if valores[varre]==valores[indice]:
                auxiliar[indice] += 1
    ondeModa = 0
    for i in range(1, len(auxiliar)):           # localiza a maior frequência
        if auxiliar[i] > auxiliar[ondeModa]:
            ondeModa = i
    return valores[ondeModa]                    # retorna o valor da moda
```

Programa Principal de Busca o Elemento da Moda

```
numeros = [0]*10
preenche(numeros)
moda = buscaModa(numeros)
print("A moda do vetor elemento da é:", moda)
```

Busca Simples pela Moda

- No algoritmo anterior, há basicamente duas estruturas de controle **for** em sequência.
 - Neste caso, o primeiro **for**, que possui o maior custo computacional, determina a complexidade do algoritmo.

Busca Simples pela Moda

- No algoritmo anterior, há basicamente duas estruturas de controle **for** em sequência.
 - Neste caso, o primeiro **for**, que possui o maior custo computacional, determina a complexidade do algoritmo.
- O primeiro **for** executa, para cada elemento do vetor, comparações com os elementos seguintes.
 - Desta forma, aproximadamente $n(n-1)/2$ elementos são acessados.
 - Ou seja, o número de elementos avaliados é da ordem de n^2 . Portanto sua complexidade é $O(n^2)$.

Busca pela Moda de um Vetor Ordenado

Neste caso, considera-se que os elementos encontram-se ordenados de forma crescente no vetor.

valores (ordenado)

| | | | | | | | | | |
|---|---|---|---|---|----|----|----|----|----|
| 6 | 7 | 7 | 7 | 8 | 10 | 10 | 12 | 19 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Moda deste vetor: elemento 7 (que ocorre 3 vezes).

Busca pela Moda de um Vetor Ordenado

```
def buscaModa(valores):
    moda = valores[0]
    ind = 0
    frequencia = 1
    while ind < len(valores) - 1:
        ind = ind + 1
        if valores[ind] == valores[ind-frequencia]:
            moda = valores[ind]
            frequencia = frequencia + 1
    return moda
```

No algoritmo acima, o vetor é percorrido apenas uma vez. Desta forma, o número de elementos avaliados é da ordem de n . Portanto a complexidade do algoritmo é $O(n)$.

Algoritmos de Ordenação

- O problema da ordenação é caracterizado pela organização de um conjunto de elemento do mesmo tipo segundo um critério de ordenação.

Algoritmos de Ordenação

- O problema da ordenação é caracterizado pela organização de um conjunto de elemento do mesmo tipo segundo um critério de ordenação.
- Sem perda de generalidade, o problema será atacado considerando-se que:
 - Os elementos a serem ordenados são numéricos e estão armazenados em um vetor;
 - Deseja-se ordenar os elementos não decrescentemente:
$$\text{se } i < j, \text{ então } \text{valores}[i] \leq \text{valores}[j].$$

Algoritmos de Ordenação

Entrada:

Vetor **valores** que contém um conjunto de elementos.

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Saída:

Vetor **valores** com o conjunto de elementos ordenados.

| | | | | | | | | | |
|---|---|---|---|---|---|----|----|----|----|
| 1 | 3 | 4 | 6 | 7 | 8 | 10 | 12 | 18 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Algoritmos de Ordenação

Subprogramas

```
def preenche(valores):
    for ind in range(len(valores)):
        valores[ind] = int(input("Elemento["+str(ind)+"]="))
    return None

def ordena(valores):
    ...
    return None
```

Programa Principal para Ordenar Vetor

```
numeros = [0]*10
preenche(numeros)
print("Vetor Lido:", numeros)
ordena(numeros)
print("Vetor Ordenado:", numeros)
```

Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição **i** do vetor **valores**, o algoritmo procura pelo **i**-ésimo menor elemento e o coloca na posição **i**.

valores:

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
|---|----|---|----|----|---|---|---|---|----|

0 1 2 3 4 5 6 7 8 9

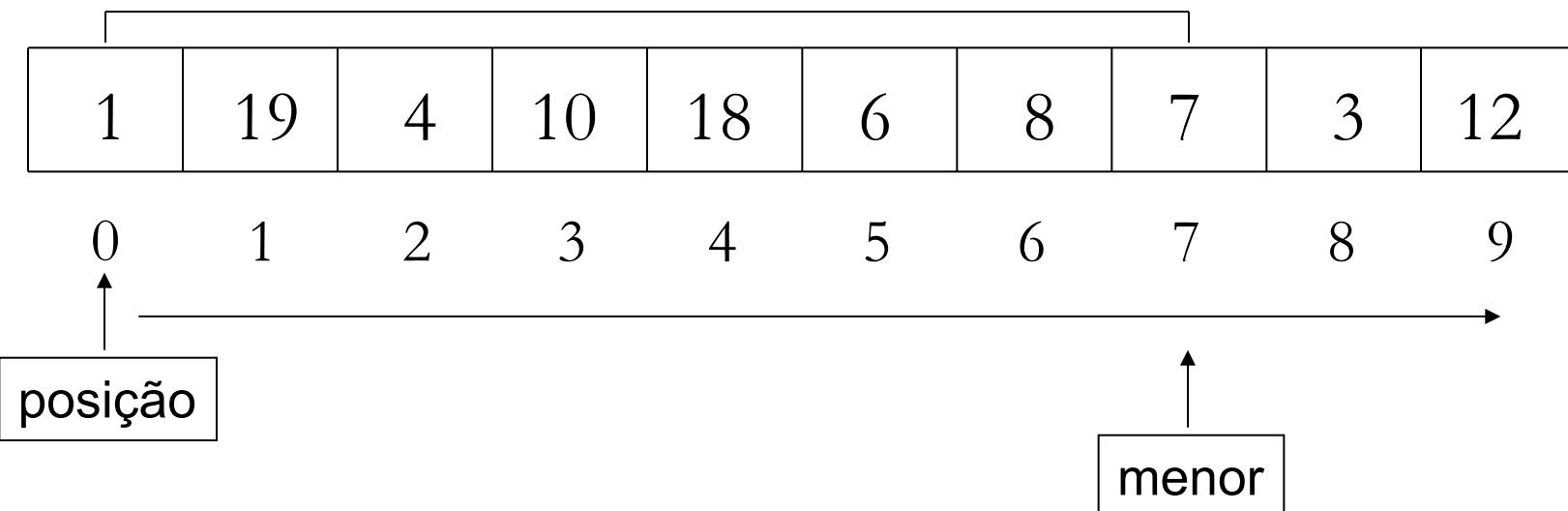
↑
posição

↑
menor

Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:



Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 1 | 19 | 4 | 10 | 18 | 6 | 8 | 7 | 3 | 12 |
|---|----|---|----|----|---|---|---|---|----|

0 1 2 3 4 5 6 7 8 9

↑
posição

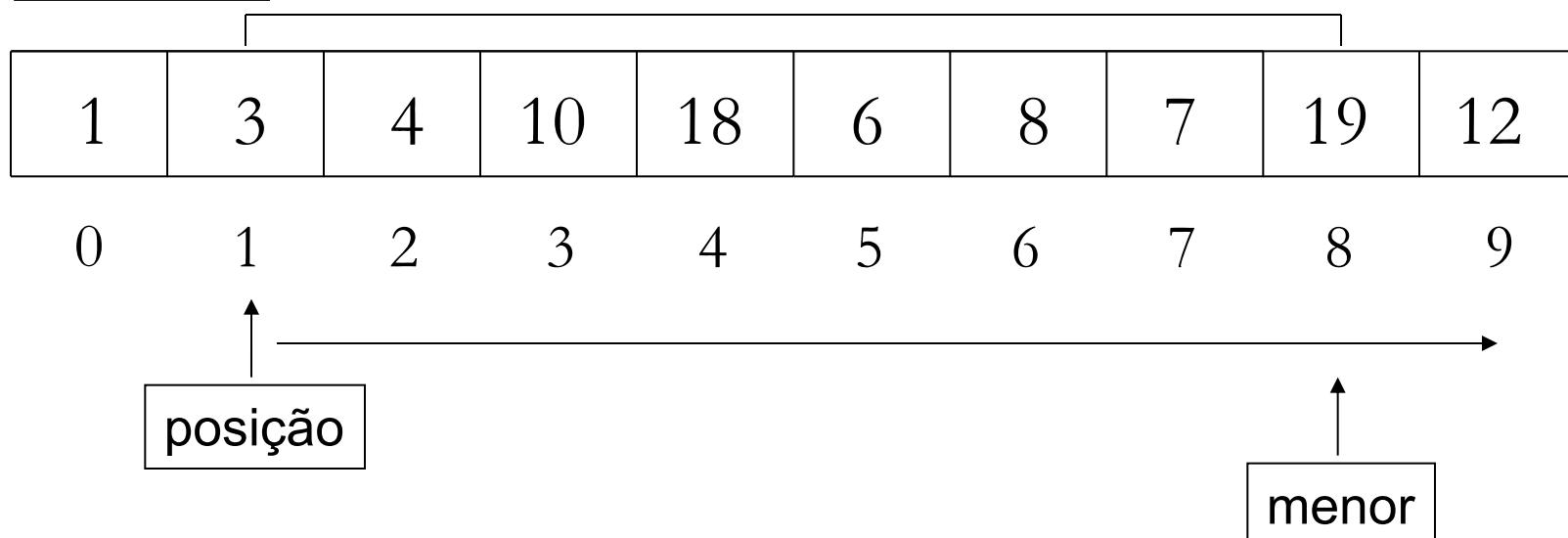
↑
menor

31

Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:



Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:

| | | | | | | | | | |
|---|---|---|----|----|---|---|---|----|----|
| 1 | 3 | 4 | 10 | 18 | 6 | 8 | 7 | 19 | 12 |
|---|---|---|----|----|---|---|---|----|----|

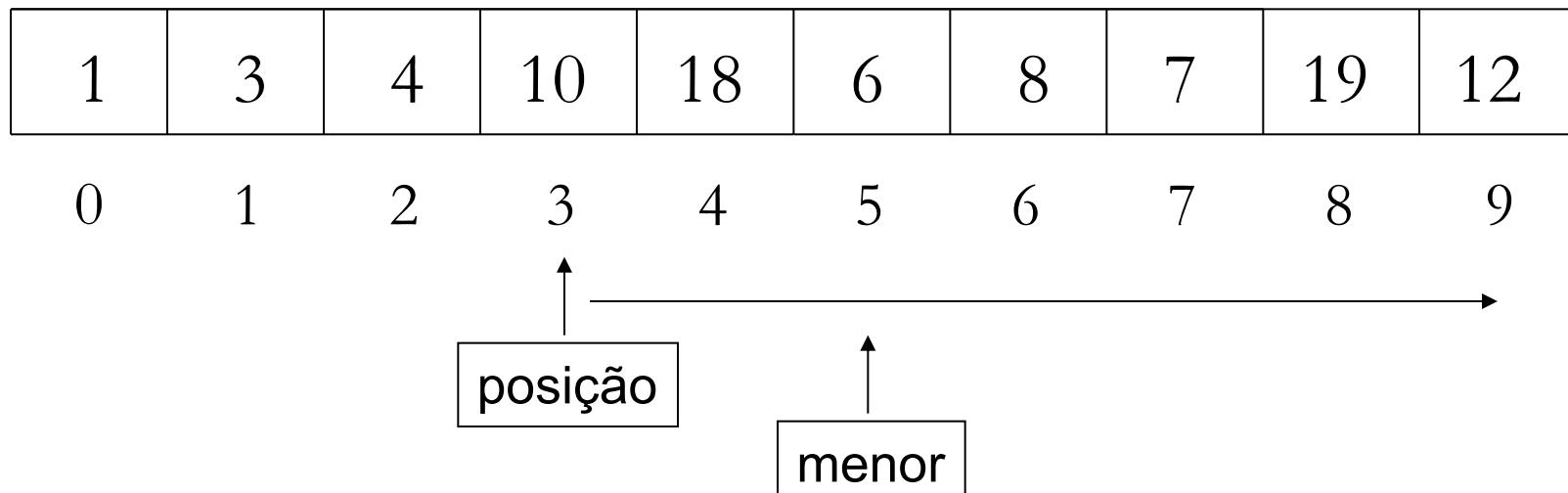
0 1 2 3 4 5 6 7 8 9



Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

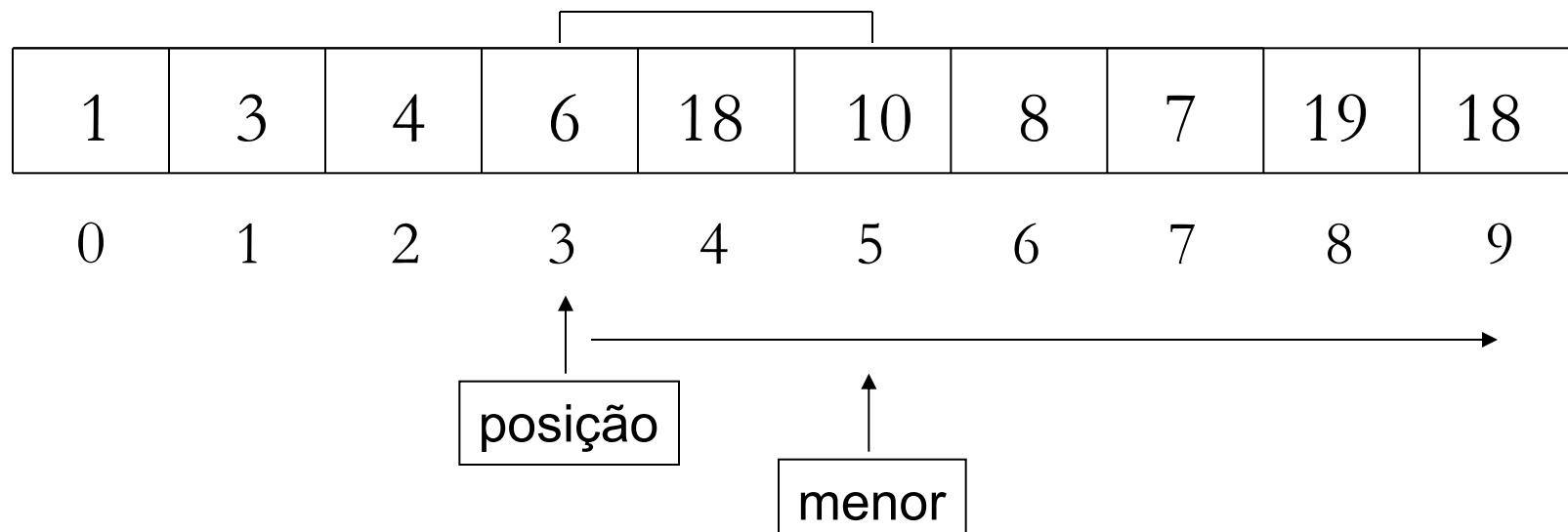
valores:



Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:



Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:

| | | | | | | | | | |
|---|---|---|---|---|---|----|----|----|----|
| 1 | 3 | 4 | 6 | 7 | 8 | 10 | 12 | 19 | 18 |
|---|---|---|---|---|---|----|----|----|----|

0 1 2 3 4 5 6 7 8 9



• • •

36

Ordenação pelo Método da Seleção (*SelectionSort*)

Para cada posição i do vetor **valores**, o algoritmo procura pelo i -ésimo menor elemento e o coloca na posição i .

valores:

| | | | | | | | | | |
|---|---|---|---|---|---|----|----|----|----|
| 1 | 3 | 4 | 6 | 7 | 8 | 10 | 12 | 18 | 19 |
|---|---|---|---|---|---|----|----|----|----|

0 1 2 3 4 5 6 7 8 9

• • •

↑
Posição

↑
Menor

37

```
# Operação que troca o conteúdo de duas células do vetor.
```

```
def trocar(vals, posX, posY):
```

```
    temp = vals[posX]
```

```
    vals[posX] = vals[posY]
```

```
    vals[posY] = temp
```

```
    return None
```

```
# Operação que encontra e retorna o local do menor elemento do vetor,
```

```
# considerando as células a partir de um dado início.
```

```
def selecionarMenor(vals, inicio):
```

```
    localMenor = inicio
```

```
    for pos in range(inicio+1, len(vals)):
```

```
        if vals[pos]<vals[localMenor]:
```

```
            localMenor = pos
```

```
    return localMenor
```

```
# Método da Seleção
```

```
def ordenar(valores):
```

```
    for ind in range(len(valores)-1):
```

```
        menor = selecionarMenor(valores, ind)
```

```
        trocar(valores, ind, menor)
```

```
    return None
```

```
# Operação que troca o conteúdo de duas células do vetor.
```

```
def trocar(vals, posX, posY):  
    temp = vals[posX]  
    vals[posX] = vals[posY]  
    vals[posY] = temp  
    return None
```

```
# Operação que encontra e retorna o local do menor elemento do vetor,  
# considerando as células a partir de um dado início.
```

```
def selecionarMenor(vals, inicio):  
    localMenor = inicio  
    for pos in range(inicio+1, len(vals)):  
        if vals[pos]<vals[localMenor]:  
            localMenor = pos  
    return localMenor
```

```
# Método da Seleção
```

```
def ordenar(valores):  
    for ind in range(len(valores)-1):  
        menor = selecionarMenor(valores, ind)  
        trocar(valores, ind, menor)  
    return None
```

```
# Operação que troca o conteúdo de duas células do vetor.
```

```
def trocar(vals, posX, posY):
    temp = vals[posX]
    vals[posX] = vals[posY]
    vals[posY] = temp
    return None
```

```
# Operação que encontra e retorna o local do menor elemento do vetor,
# considerando as células a partir de um dado início.
```

```
def selecionarMenor(vals, inicio):
    localMenor = inicio
    for pos in range(inicio+1, len(vals)):
        if vals[pos]<vals[localMenor]:
            localMenor = pos
    return localMenor
```

```
# Método da Seleção
```

```
def ordenar(valores):
    for ind in range(len(valores)-1):
        menor = selecionarMenor(valores, ind)
        trocar(valores, ind, menor)
    return None
```

Ordenação pelo Método da Seleção (*SelectionSort*)

- No algoritmo anterior, há basicamente duas estruturas de repetição **for** aninhadas.

Ordenação pelo Método da Seleção (*SelectionSort*)

- No algoritmo anterior, há basicamente duas estruturas de repetição **for** aninhadas.
- A mais externa executa, para cada elemento do vetor, comparações com os elementos seguintes e em seguida uma troca.
 - Desta forma, aproximadamente $n(n-1)/2$ elementos são acessados.

Ordenação pelo Método da Seleção (*SelectionSort*)

- No algoritmo anterior, há basicamente duas estruturas de repetição **for** aninhadas.
- A mais externa executa, para cada elemento do vetor, comparações com os elementos seguintes e em seguida uma troca.
 - Desta forma, aproximadamente $n(n-1)/2$ elementos são acessados.
- Ou seja, o número de elementos avaliados é da ordem de n^2 .
 - Portanto sua complexidade é $O(n^2)$.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Primeira Iteração Externa

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| ↑ | ↑ | | | | | | | | |

7 < 19

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Primeira Iteração Externa

| | | | | | | | | | |
|---|----|---|----|----|---|---|---|---|----|
| 7 | 19 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | ↑ | ↑ | | | | | | | |

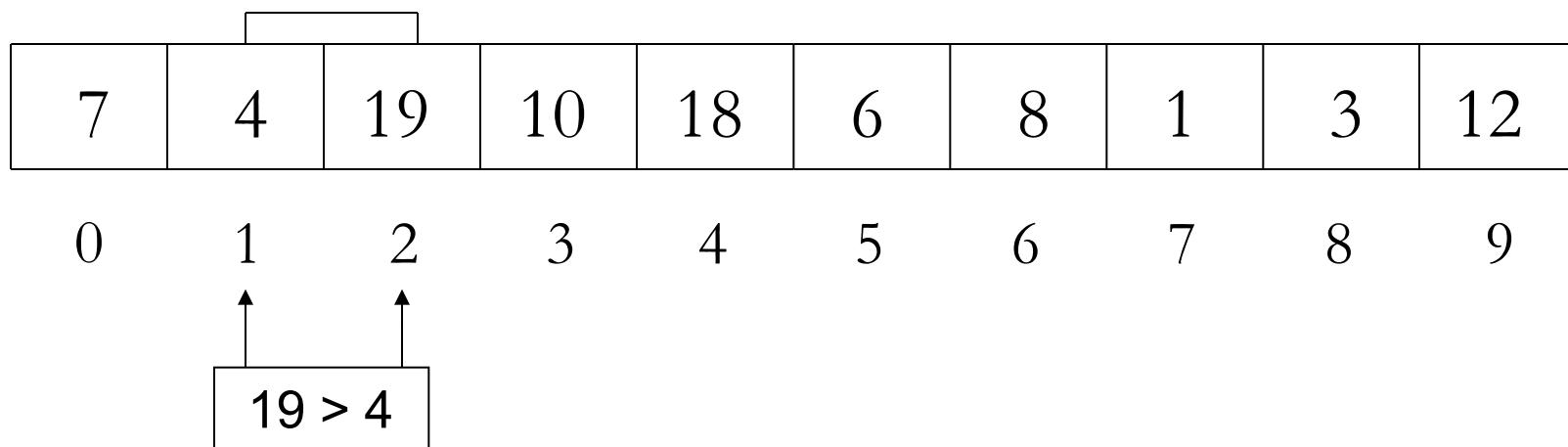
19 > 4

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Primeira Iteração Externa



Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Primeira Iteração Externa

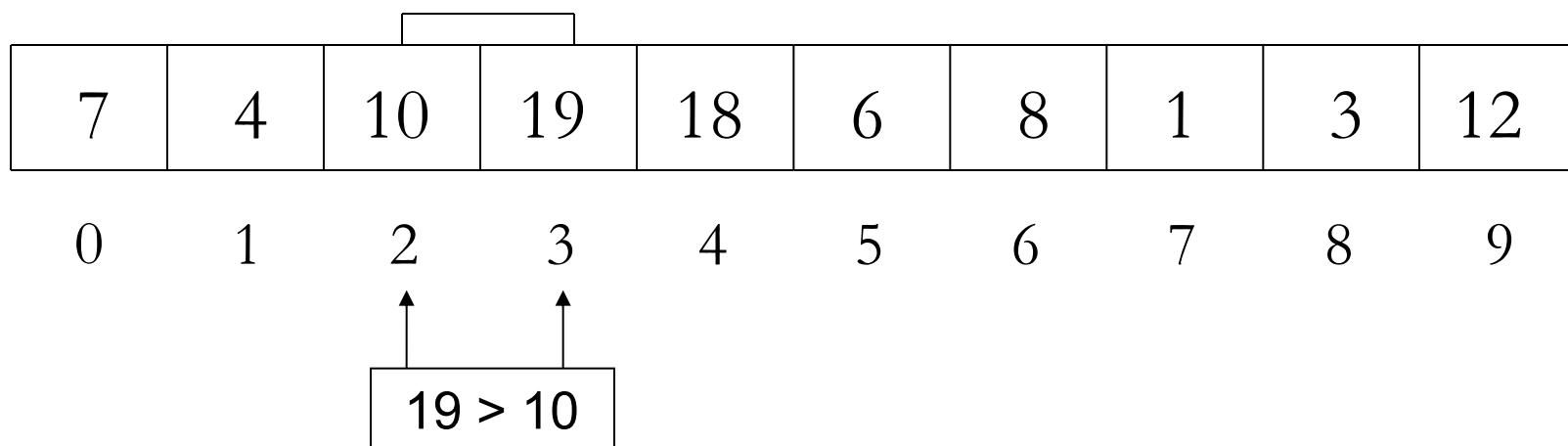
| | | | | | | | | | |
|---|---|----|----|--|---|---|---|---|----|
| 7 | 4 | 19 | 10 | 18 | 6 | 8 | 1 | 3 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | | ↑ | ↑ | <div style="border: 1px solid black; padding: 5px;">19 > 10</div> | | | | | |

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Primeira Iteração Externa



Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

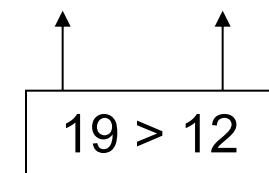
valores:

Primeira Iteração Externa

| | | | | | | | | | |
|---|---|----|----|---|---|---|---|----|----|
| 7 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 19 | 12 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

■ ■ ■

$19 > 12$

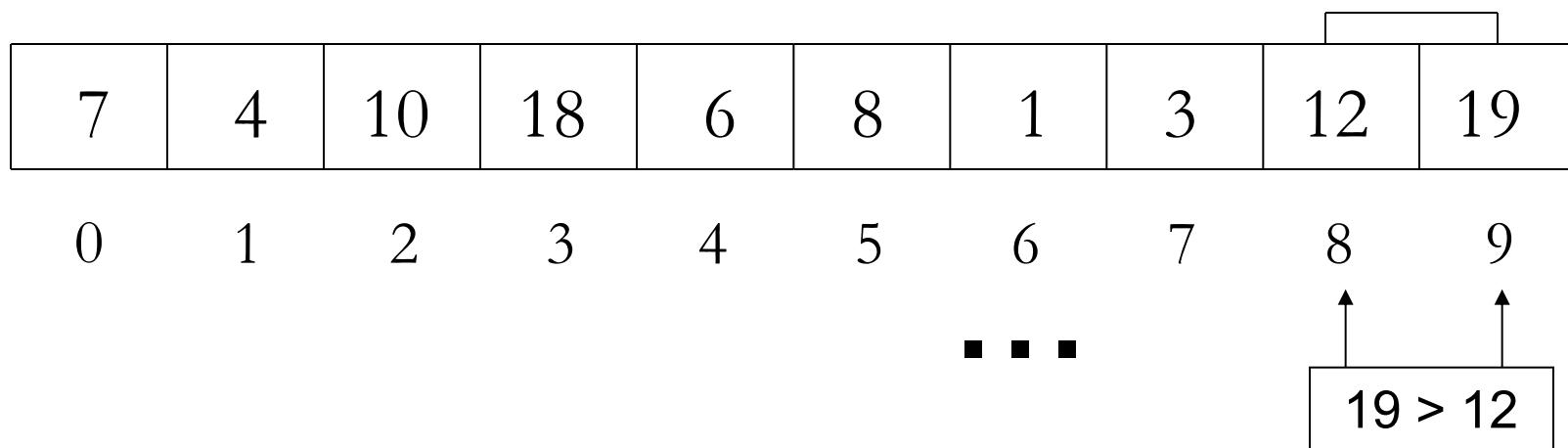


Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Primeira Iteração Externa



Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa

| | | | | | | | | | |
|---|---|----|----|---|---|---|---|----|----|
| 7 | 4 | 10 | 18 | 6 | 8 | 1 | 3 | 12 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

↑ ↑

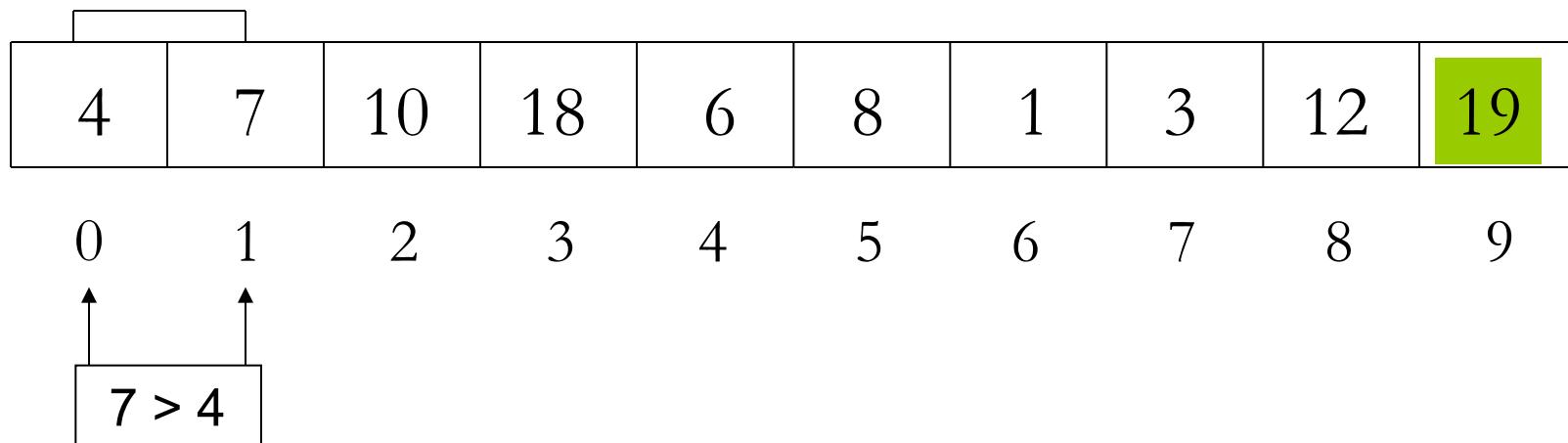
$7 > 4$

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa



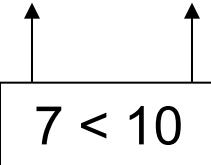
Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa

| | | | | | | | | | |
|---|---|----|----|---|---|---|---|----|----|
| 4 | 7 | 10 | 18 | 6 | 8 | 1 | 3 | 12 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

 7 < 10

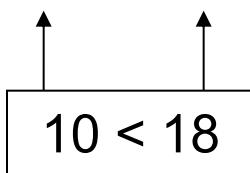
Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa

| | | | | | | | | | |
|---|---|----|----|---|---|---|---|----|----|
| 4 | 7 | 10 | 18 | 6 | 8 | 1 | 3 | 12 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |


10 < 18

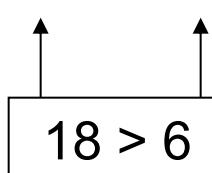
Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa

| | | | | | | | | | |
|---|---|----|----|---|---|---|---|----|----|
| 4 | 7 | 10 | 18 | 6 | 8 | 1 | 3 | 12 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

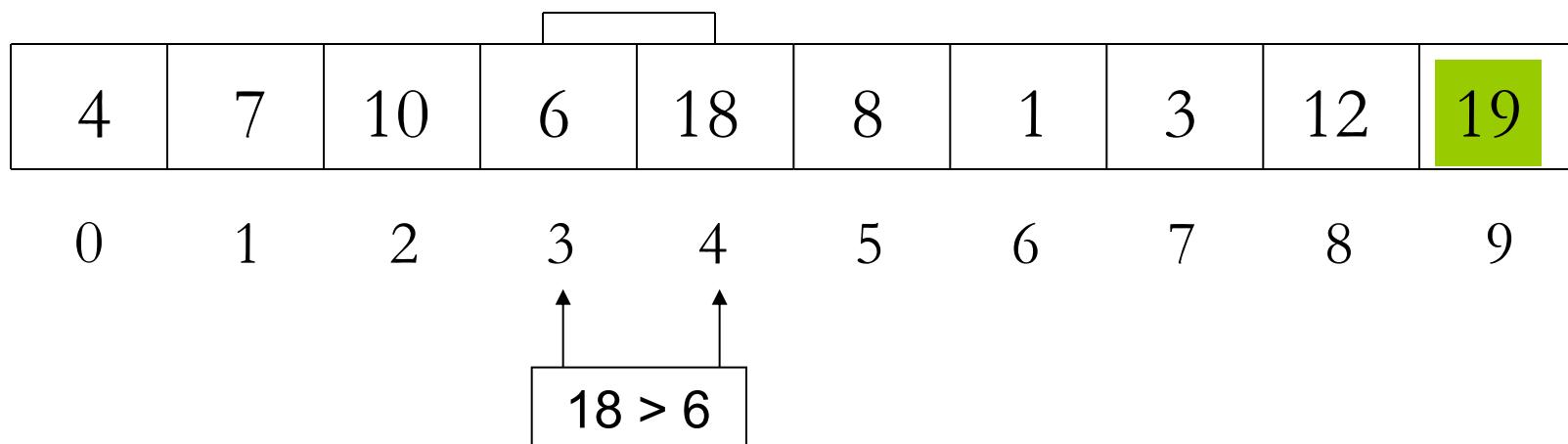

18 > 6

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa



Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa

| | | | | | | | | | |
|---|---|----|---|---|---|---|----|----|----|
| 4 | 7 | 10 | 6 | 8 | 1 | 3 | 18 | 12 | 19 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

■ ■ ■

$18 > 12$

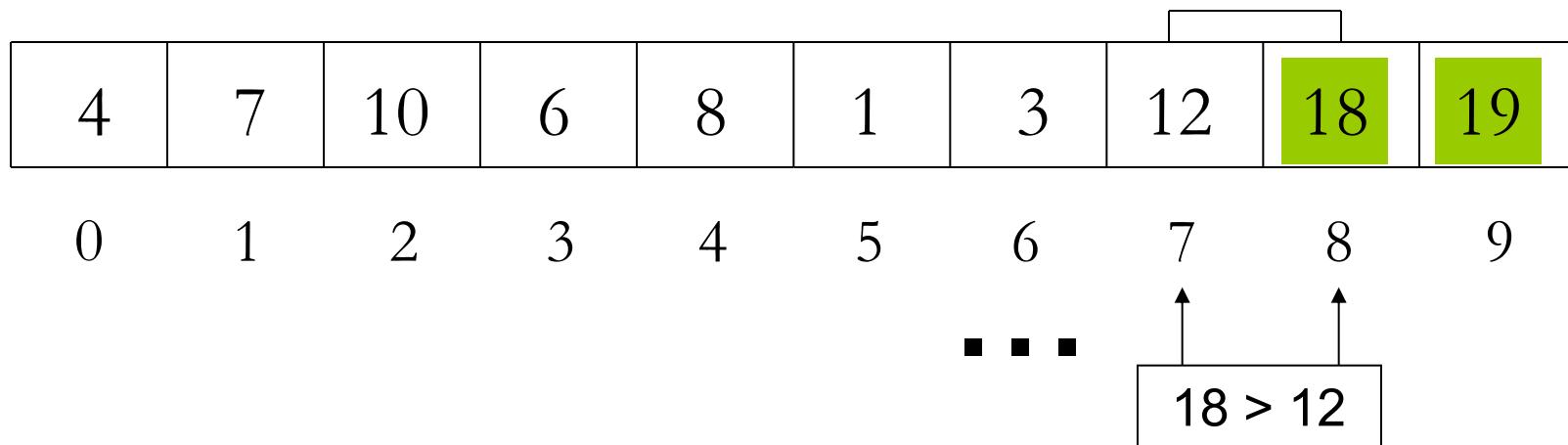
The diagram illustrates the second pass of bubble sort on the array `valores`. The array elements are shown in boxes, with indices from 0 to 9 below. The 8 at index 4 and the 12 at index 8 are highlighted with green boxes. Arrows point from these boxes to a comparison box at the bottom containing the inequality `18 > 12`. Ellipses between indices 6 and 7 indicate that the comparison continues through the remaining elements of the array.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Segunda Iteração Externa

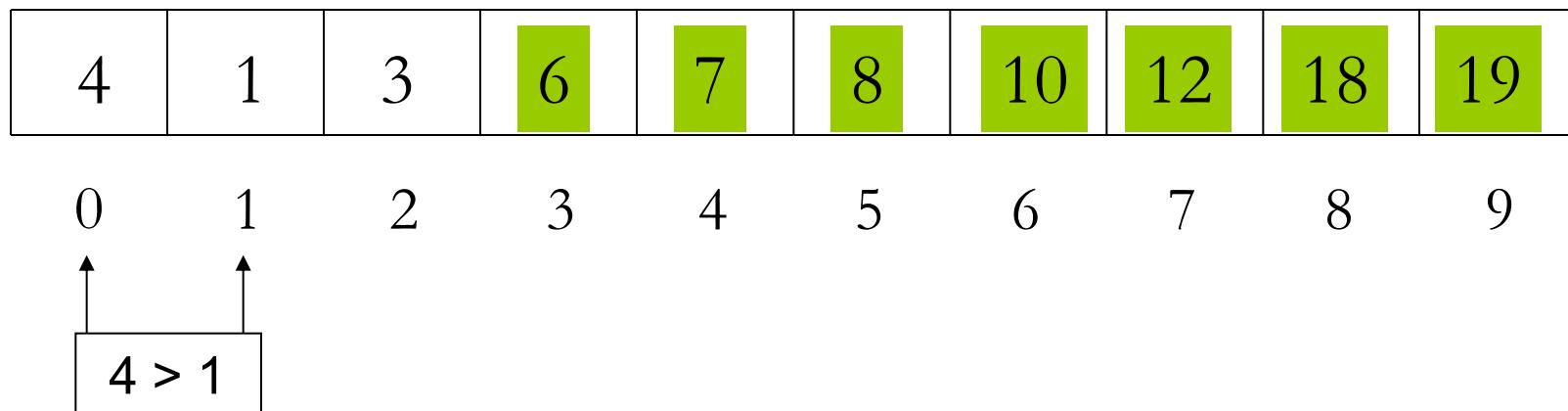


Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

■ ■ ■ Oitava Iteração Externa

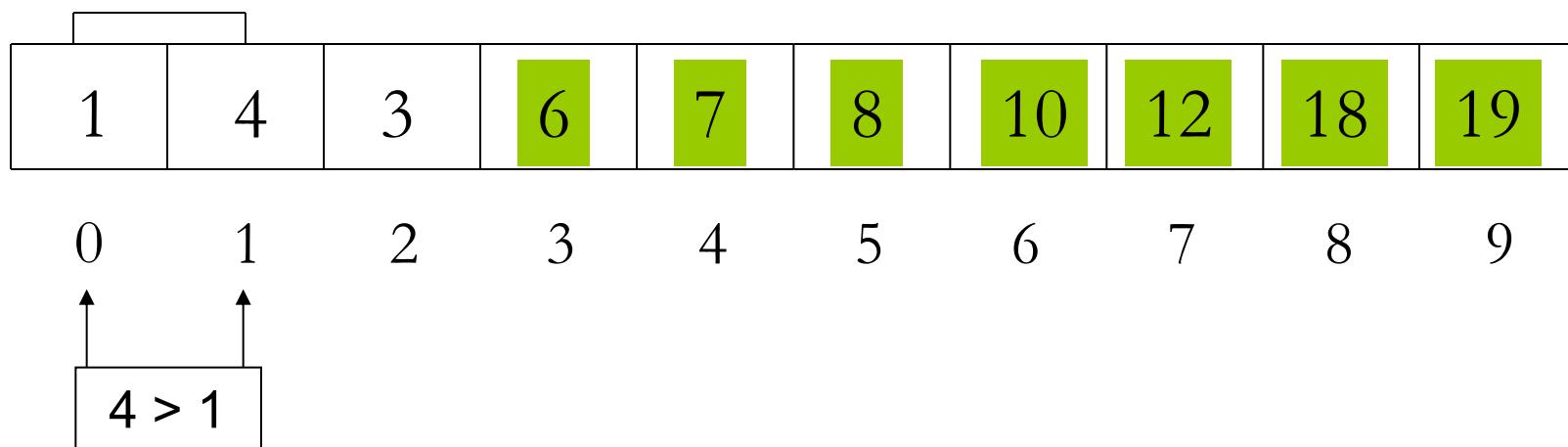


Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

■ ■ ■ Oitava Iteração Externa

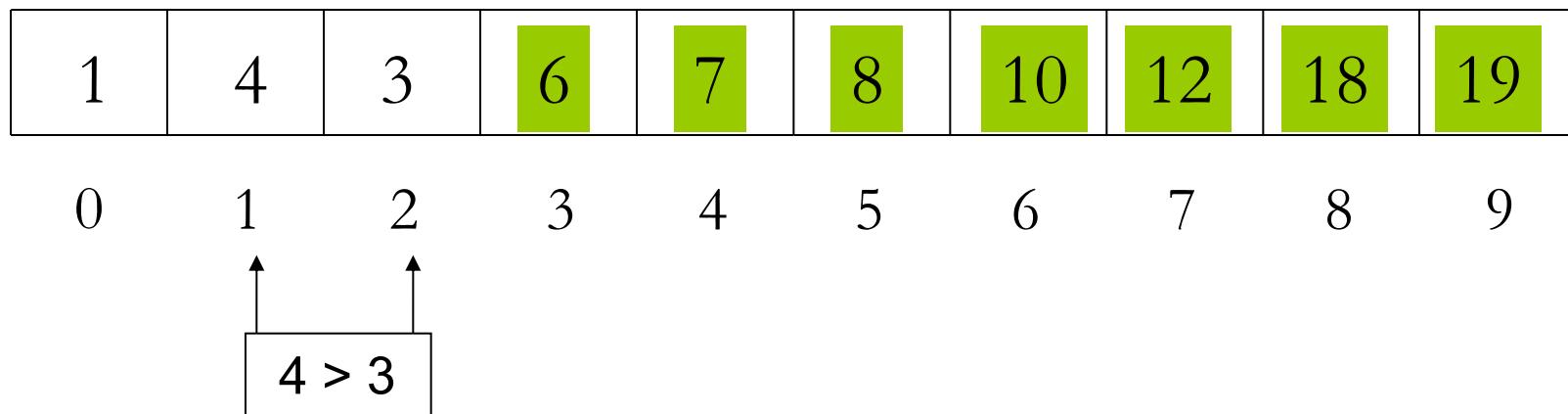


Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

■ ■ ■ Oitava Iteração Externa

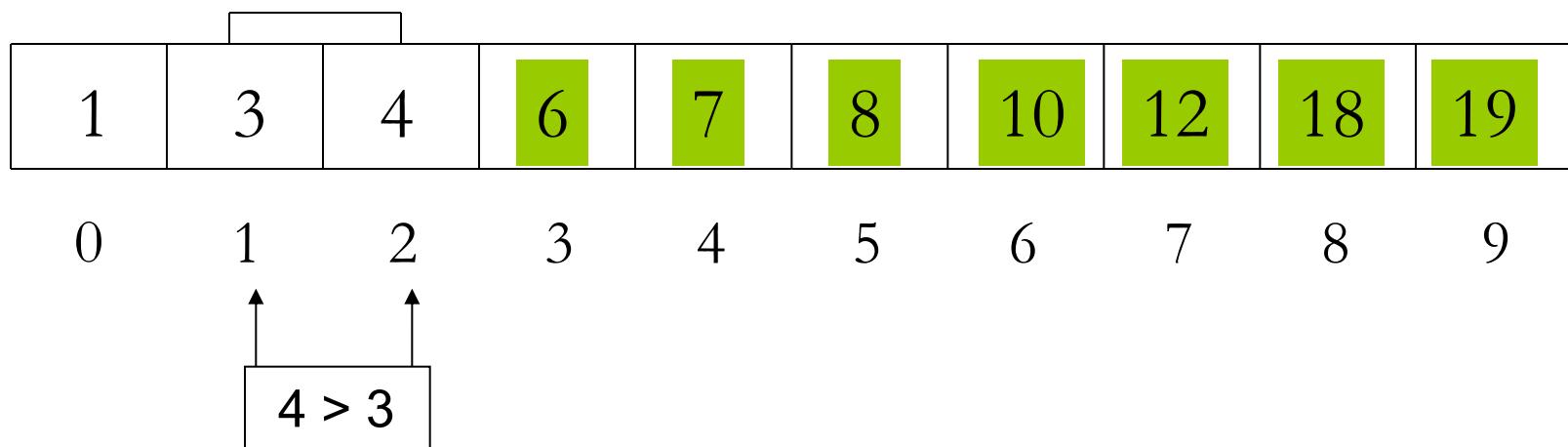


Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

■ ■ ■ Oitava Iteração Externa

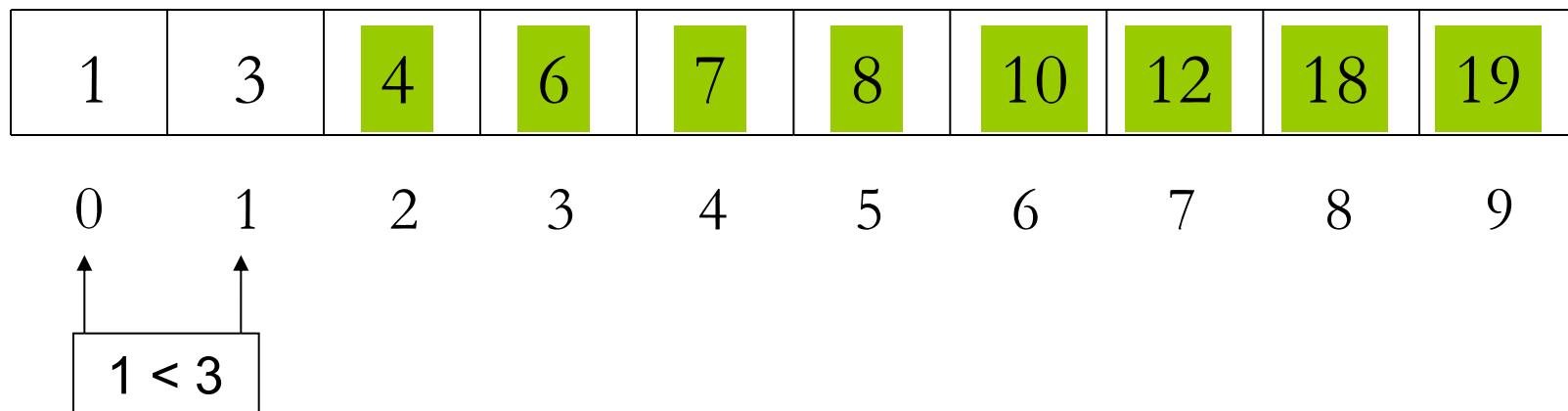


Ordenação pelo Método da Bolha (*BubbleSort*)

- Esta estratégia executa $n-1$ iterações, controlada por uma repetição mais externa.
- Em cada uma delas, via repetição interna,
 - percorre-se todo o vetor comparando cada par de elementos $\text{valores}[i]$ e $\text{valores}[i+1]$ e
 - trocando-os de posição caso $\text{valores}[i] > \text{valores}[i+1]$.

valores:

Nona (última) Iteração Externa



Ordenação pelo Método da Bolha (*BubbleSort*)

```
# Operação que troca o conteúdo de duas células do vetor.
```

```
def trocar(vals, posX, posY):
```

```
    temp = vals[posX]
```

```
    vals[posX] = vals[posY]
```

```
    vals[posY] = temp
```

```
    return None
```

```
# Método da Bolha
```

```
def ordenar(valores):
```

```
    for tamanho in range(len(valores)-1,0,-1):
```

```
        for i in range(tamanho):
```

```
            if valores[i]>valores[i+1]:
```

```
                trocar(valores, i, i+1)
```

```
    return None
```

Ordenação pelo Método da Bolha (*BubbleSort*)

- Novamente, neste algoritmo, há basicamente duas estruturas de repetição **for** aninhadas.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Novamente, neste algoritmo, há basicamente duas estruturas de repetição **for** aninhadas.
- A mais externa, executado $n-1$ vezes, representa as iterações. Na i -ésima iteração, $n-i$ comparações são feitas.
 - Como no algoritmo anterior, aproximadamente $n(n-1)/2$ comparações são realizadas.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Novamente, neste algoritmo, há basicamente duas estruturas de repetição **for** aninhadas.
- A mais externa, executado $n-1$ vezes, representa as iterações. Na i -ésima iteração, $n-i$ comparações são feitas.
 - Como no algoritmo anterior, aproximadamente $n(n-1)/2$ comparações são realizadas.
- Desta forma, o custo computacional do método da bolha também é da ordem de n^2 .
 - Portanto sua complexidade é $O(n^2)$.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Nesta implementação, o algoritmo para na primeira iteração em que não houver nenhuma troca.

Ordenação pelo Método da Bolha (*BubbleSort*)

- Nesta implementação, o algoritmo para na primeira iteração em que não houver nenhuma troca.
- No pior caso, n iterações serão realizadas, o que não muda a complexidade de pior caso do algoritmo.

Ordenação pelo Método da Bolha (*BubbleSort*)

```
# Operação que troca o conteúdo de duas células do vetor.
```

```
def trocar(vals, posX, posY):  
    temp = vals[posX]  
    vals[posX] = vals[posY]  
    vals[posY] = temp  
    return None
```

```
# Outra forma do Método da Bolha – com saída rápida
```

```
def ordenar(valores):  
    tamanho = len(valores)-1  
    troquei = True  
    while troquei:  
        troquei = False  
        for i in range(tamanho):  
            if valores[i]>valores[i+1]:  
                trocar(valores, i, i+1)  
                troquei = True  
    tamanho -= 1  
    return None
```

Ordenação pelo Método da Partição (QuickSort)

- Trata-se de um método de ordenação eficiente e de natureza recursiva.

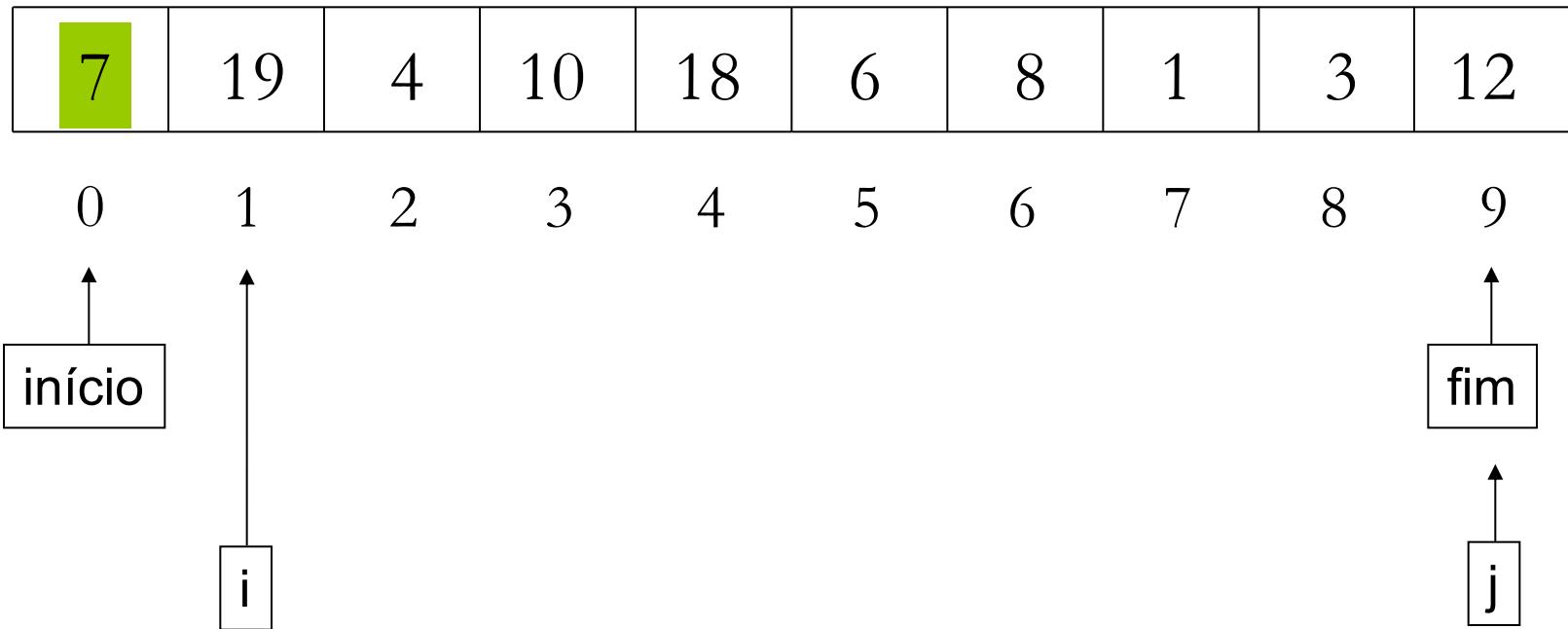
Ordenação pelo Método da Partição (QuickSort)

- Trata-se de um método de ordenação eficiente e de natureza recursiva.
- Em cada ativação do algoritmo:
 1. Um elemento do vetor é escolhido e é denominado pivô;
 2. Todos os elementos do vetor são re-arrumados, ocupando as primeiras células do vetor os elementos menores que o pivô, o pivô ocupa sua posição definitiva, seguido pelos valores maiores ou iguais a ele.
 3. Aplicamos a mesma ideia, com chamadas recursivas, aos subvetores, também chamados de partições, com tamanho maiores que 1, contendo elementos menores que o pivô, e com valores maiores ou iguais ao pivô, respectivamente.
 4. Quando todos os subvetores tiverem tamanhos menores ou iguais a um, o vetor original estará ordenado.

Ordenação pelo Método da Partição (QuickSort)

valores:

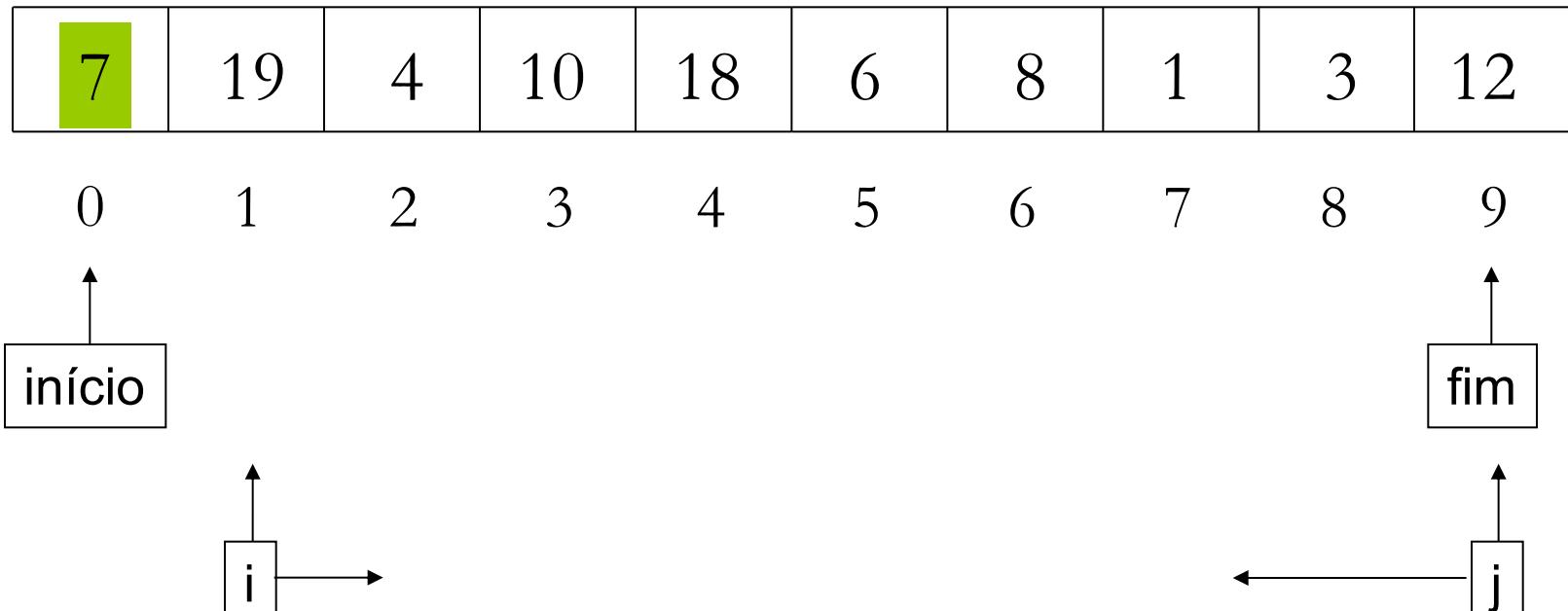
pivo = valores[início] = 7



Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7



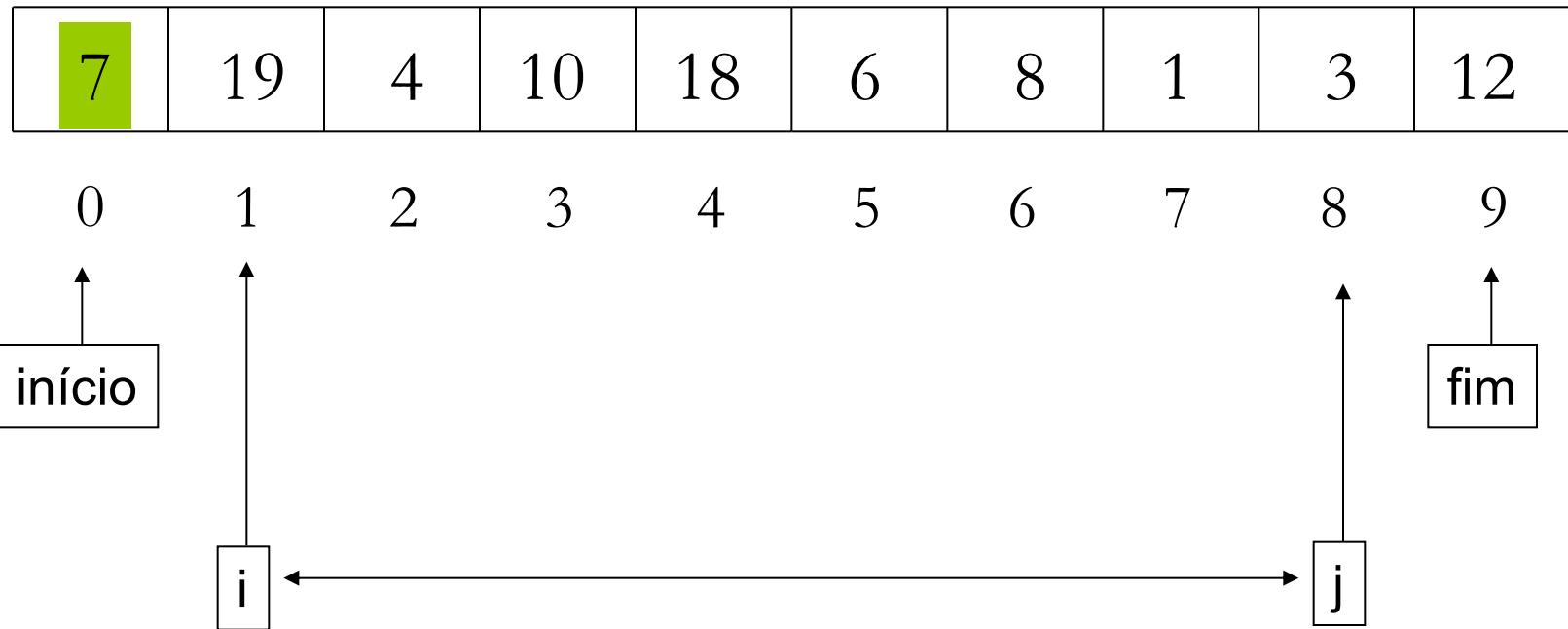
Enquanto $i < fim$ e $valores[i] < \text{pivo}$

Enquanto $j > \text{início}$ e $valores[j] \geq \text{pivo}$

Ordenação pelo Método da Partição (QuickSort)

valores:

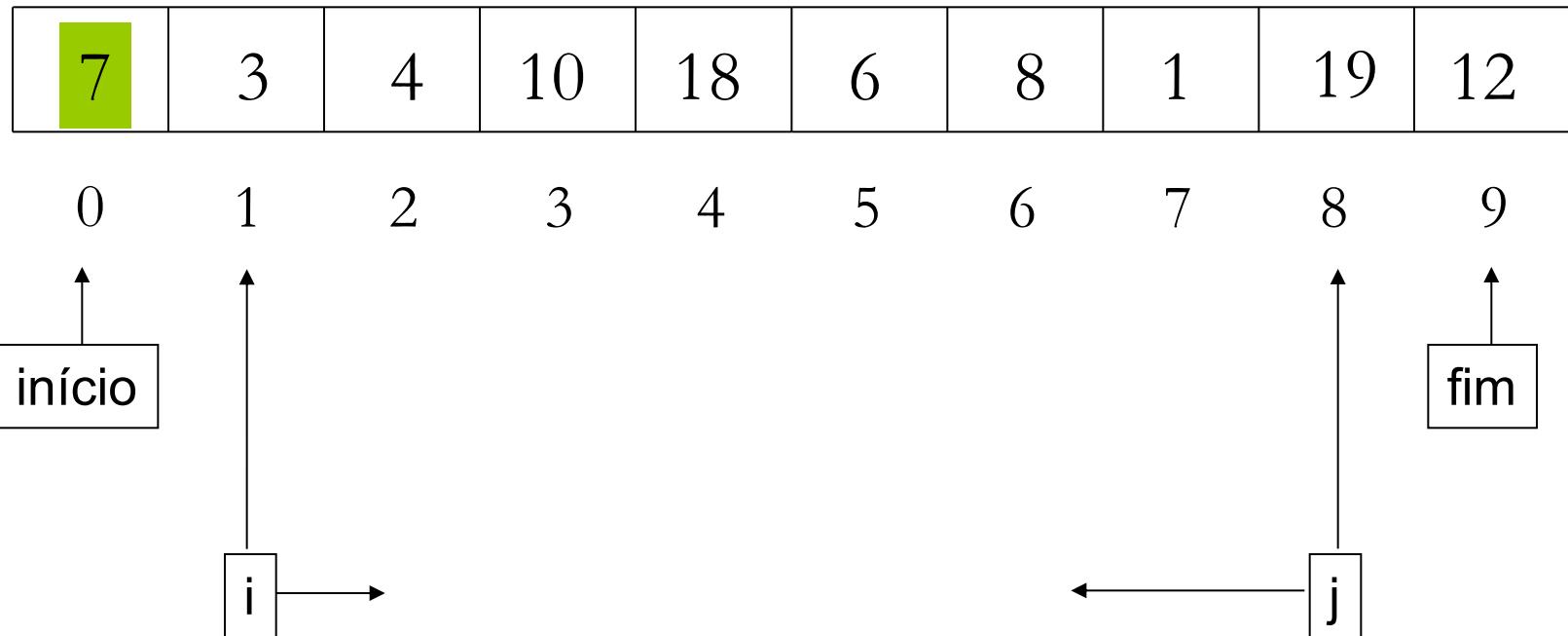
pivo = valores[início] = 7



Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7



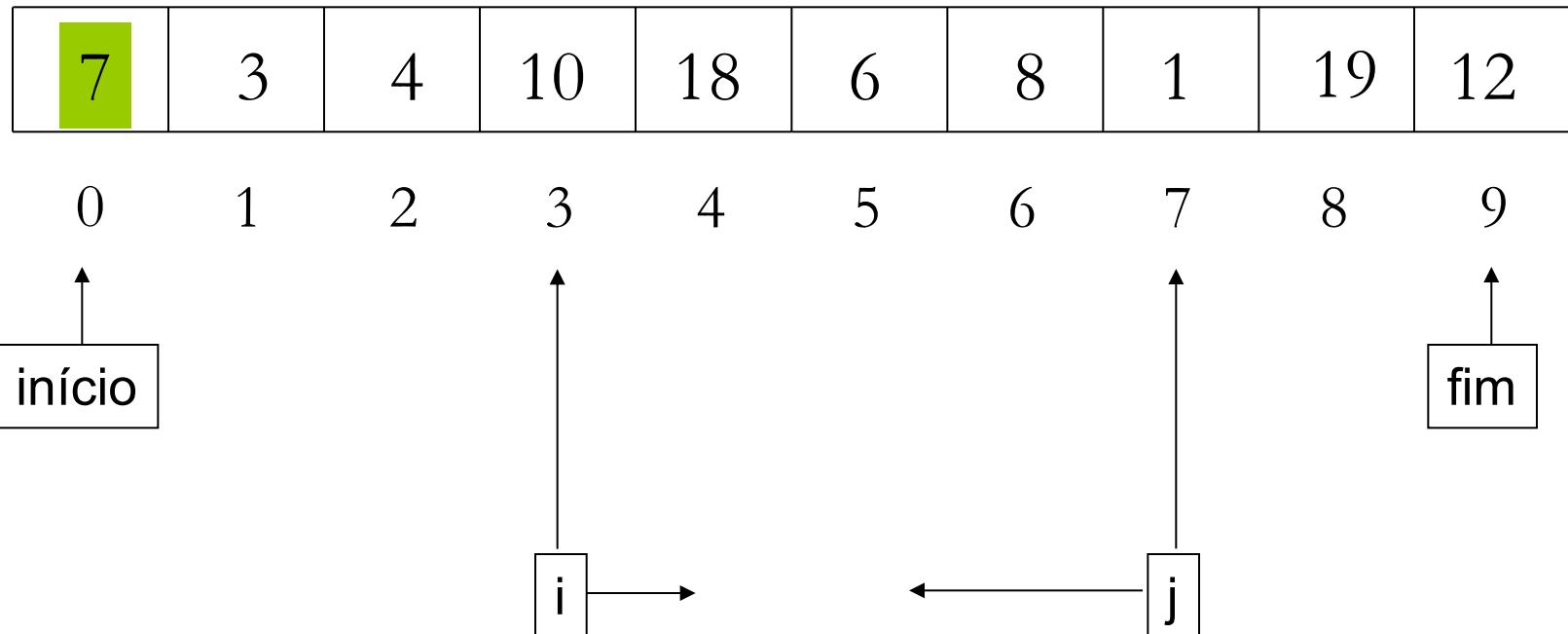
Enquanto $i < fim$ e $valores[i] < \text{pivo}$

Enquanto $j > \text{início}$ e $valores[j] >= \text{pivo}$

Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7



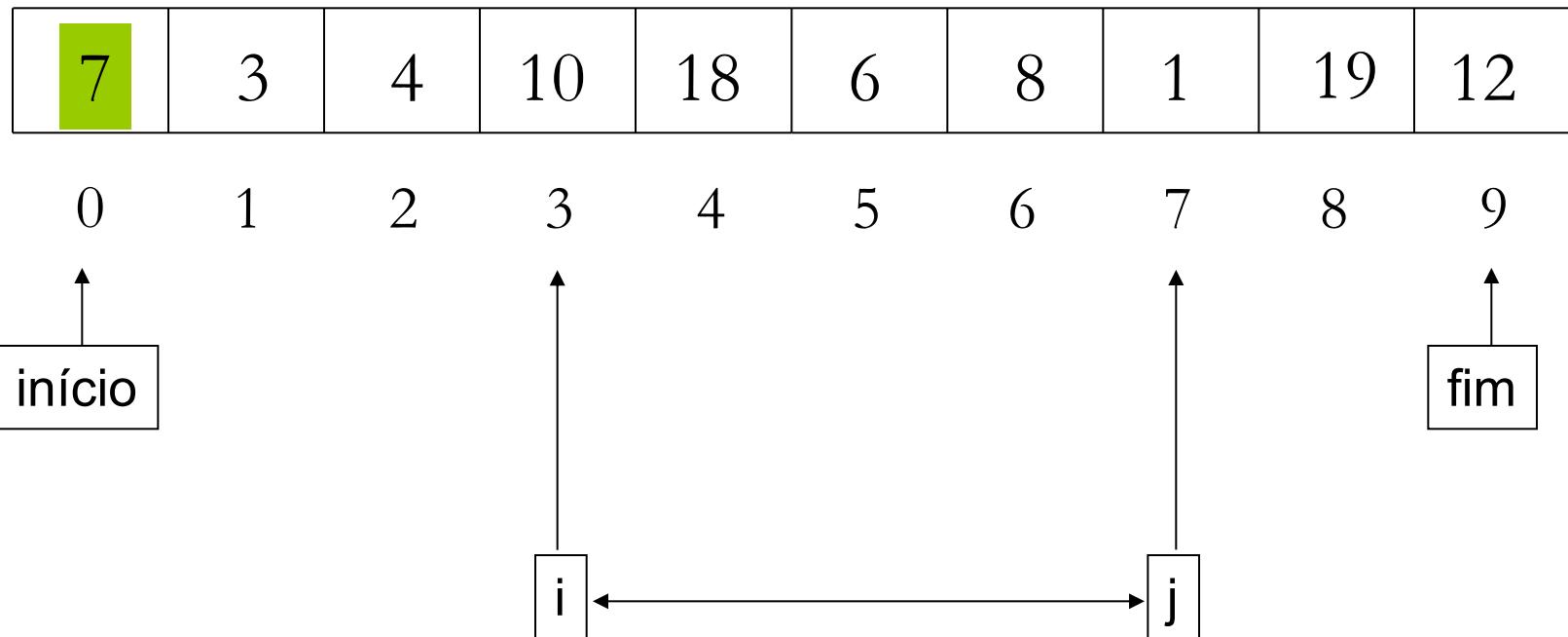
Enquanto $i < fim$ e $valores[i] < \text{pivo}$

Enquanto $j > \text{início}$ e $valores[j] \geq \text{pivo}$

Ordenação pelo Método da Partição (QuickSort)

valores:

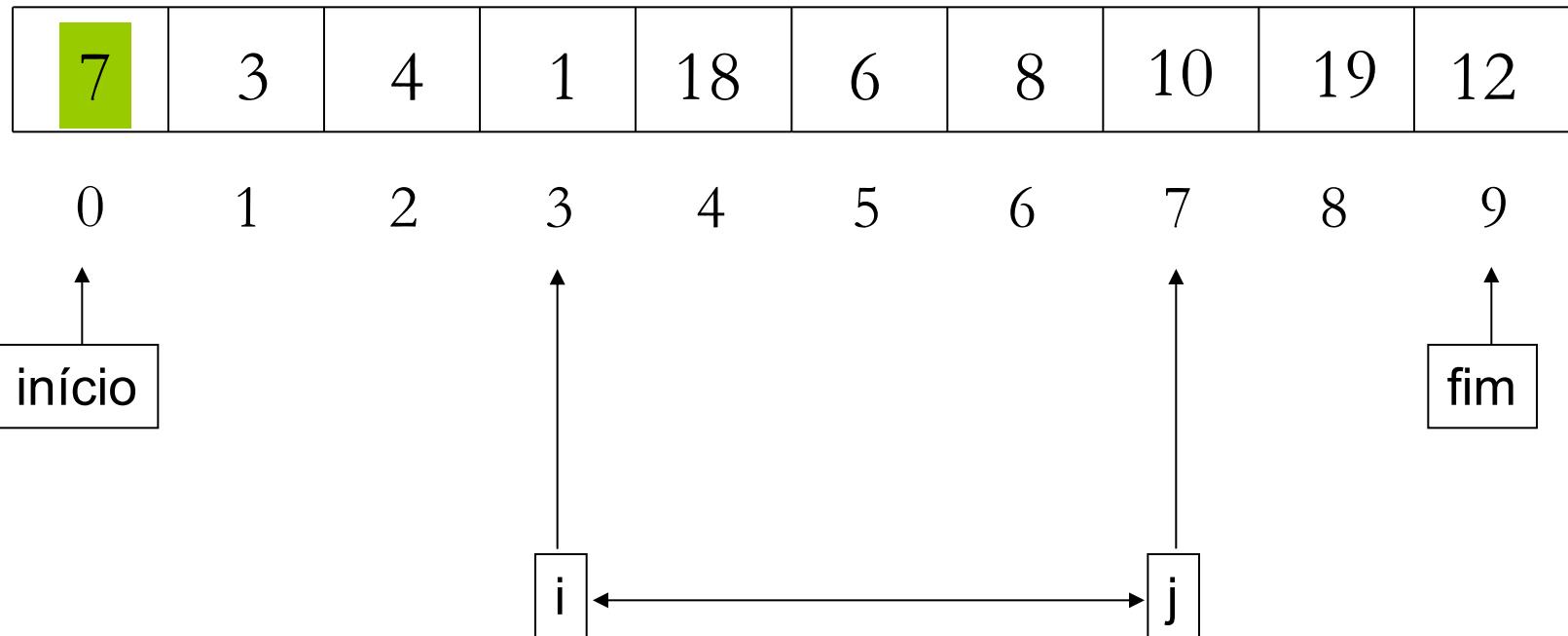
pivo = valores[início] = 7



Ordenação pelo Método da Partição (QuickSort)

valores:

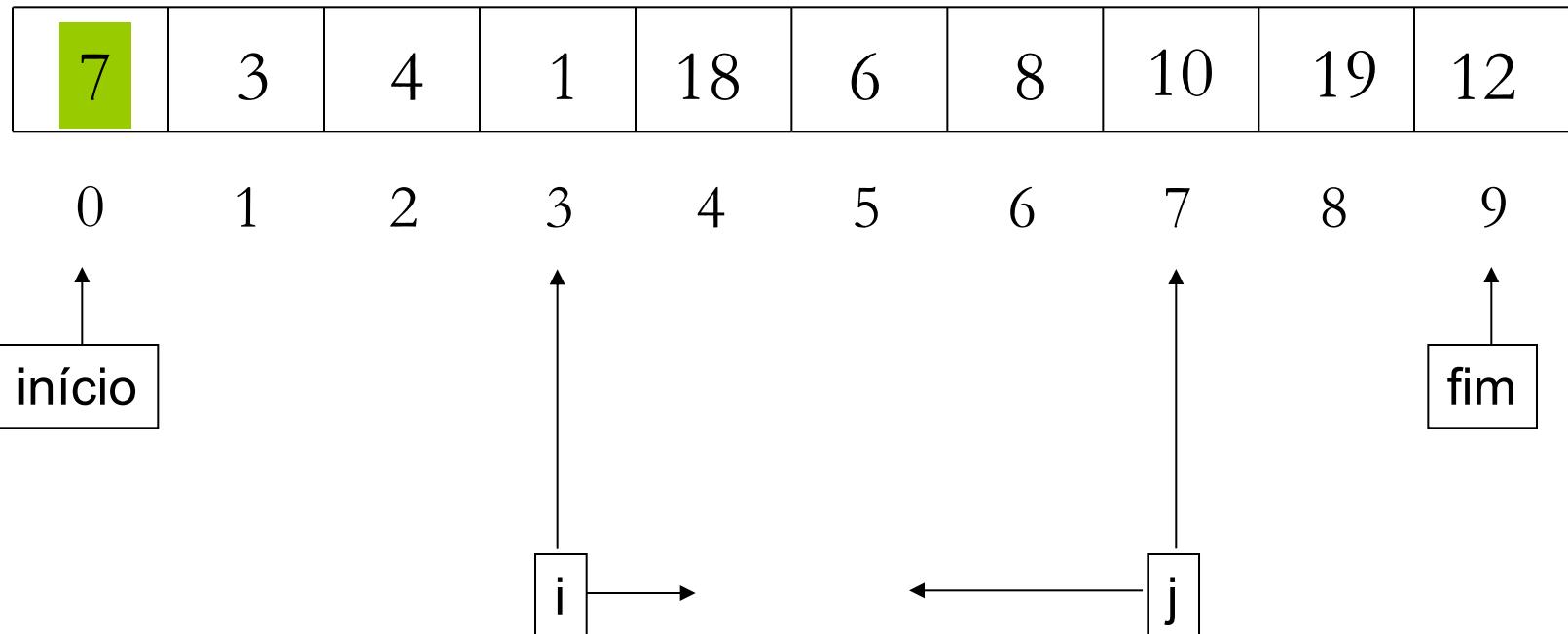
pivo = valores[início] = 7



Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7



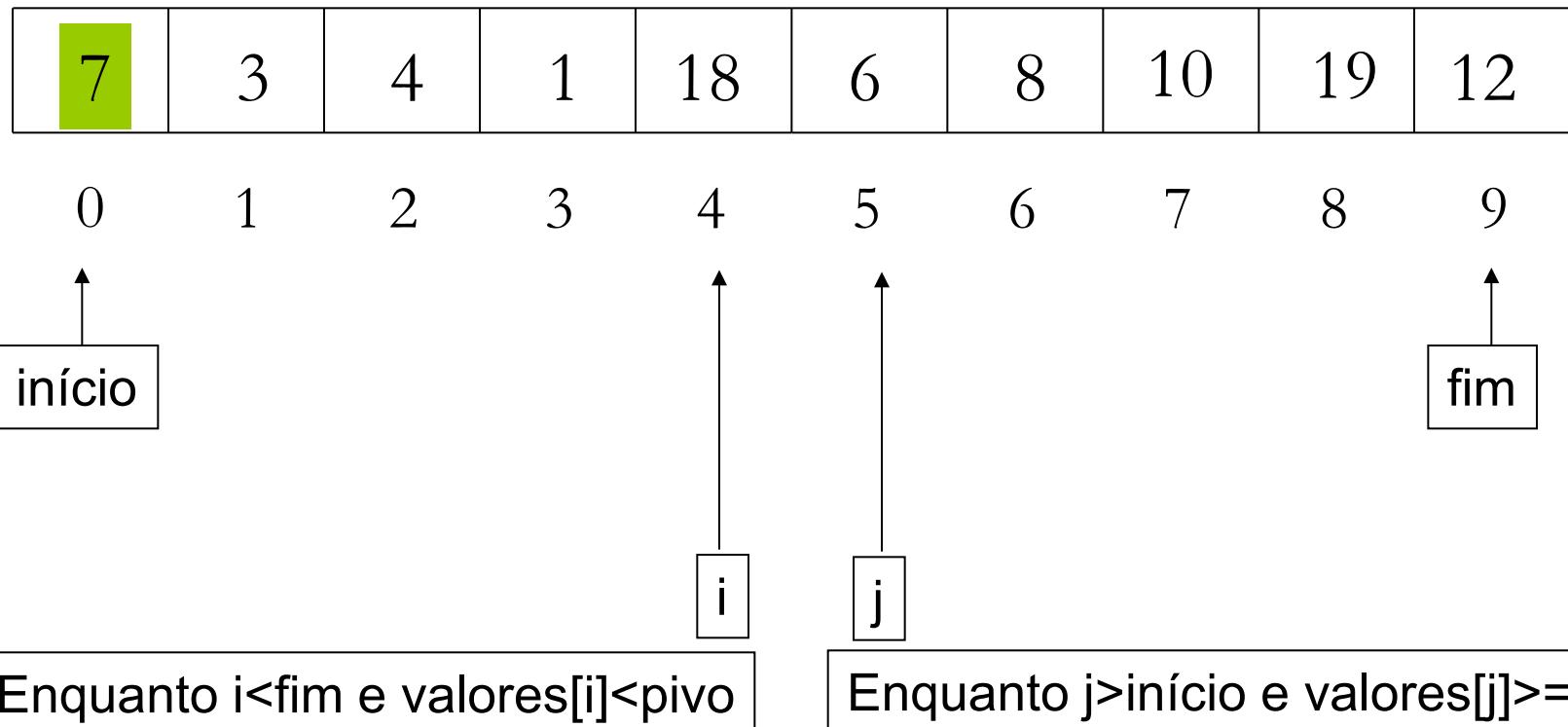
Enquanto $i < \text{fim}$ e $\text{valores}[i] < \text{pivo}$

Enquanto $j > \text{início}$ e $\text{valores}[j] >= \text{pivo}$

Ordenação pelo Método da Partição (QuickSort)

valores:

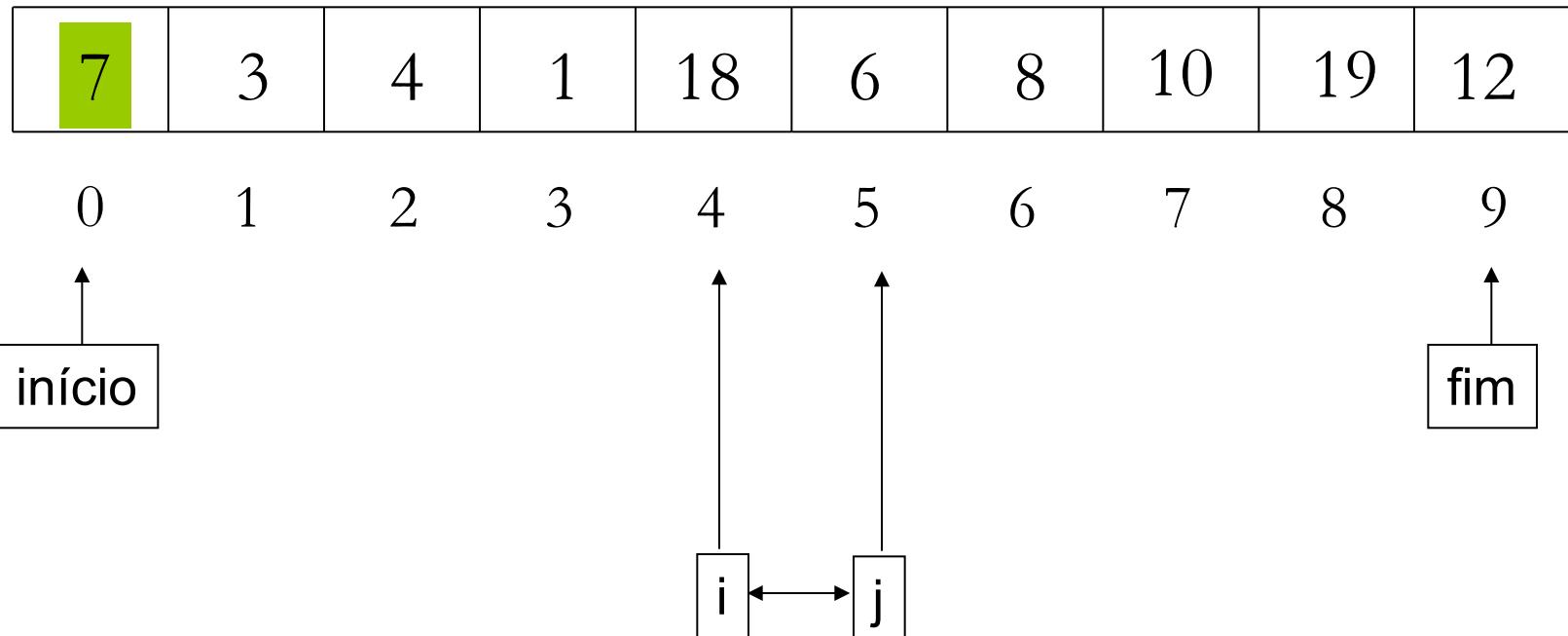
pivo = valores[início] = 7



Ordenação pelo Método da Partição (QuickSort)

valores:

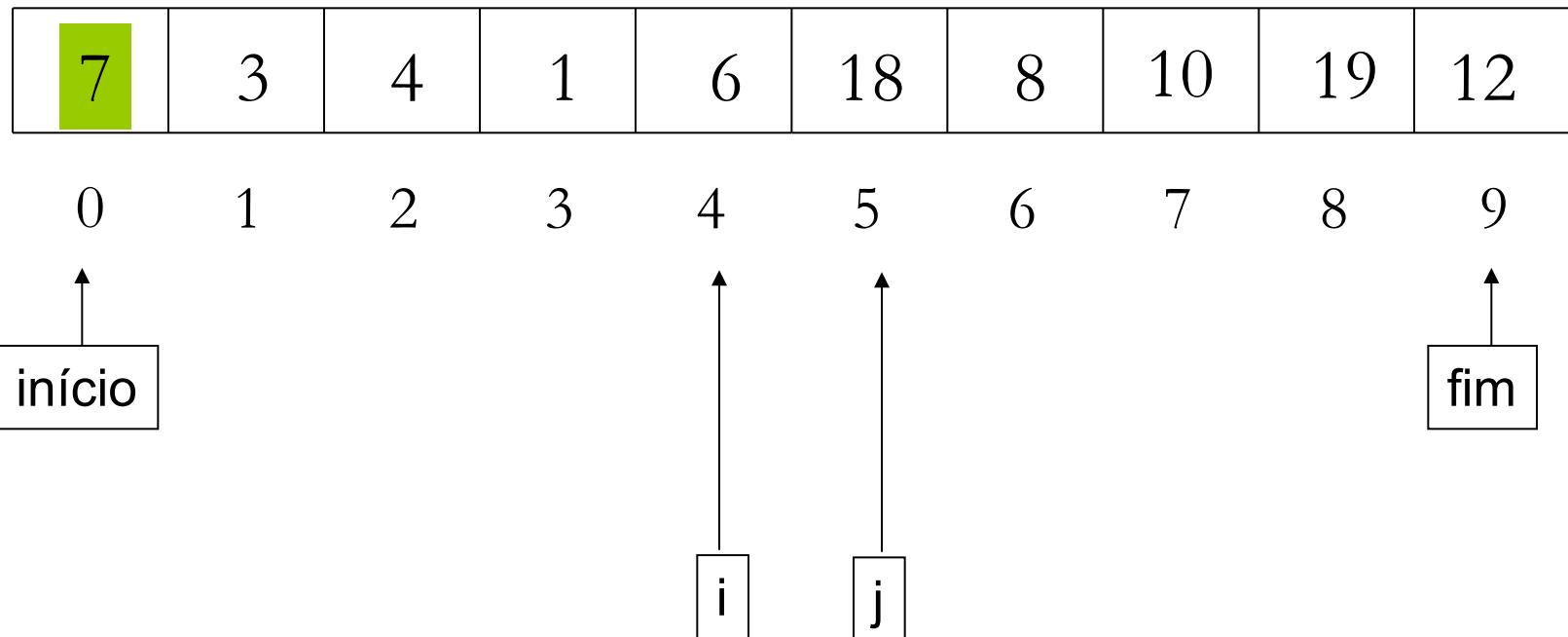
$\text{pivo} = \text{valores}[\text{início}] = 7$



Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7



Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7

| | | | | | | | | | |
|---|---|---|---|---|----|---|----|----|----|
| 7 | 3 | 4 | 1 | 6 | 18 | 8 | 10 | 19 | 12 |
|---|---|---|---|---|----|---|----|----|----|

0 1 2 3 4 5 6 7 8 9

↑
início

↑
i → ← j

↑
fim

Enquanto $i < fim$ e $\text{valores}[i] < \text{pivo}$

Enquanto $j > \text{início}$ e $\text{valores}[j] >= \text{pivo}$

Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7

| | | | | | | | | | |
|---|---|---|---|---|----|---|----|----|----|
| 7 | 3 | 4 | 1 | 6 | 18 | 8 | 10 | 19 | 12 |
|---|---|---|---|---|----|---|----|----|----|

0 1 2 3 4 5 6 7 8 9

↑
início

↑
j ↑
i

↑
fim

Dado que $j < i$, encerra-se a repetição. Em seguida coloca-se o pivô na sua posição definitiva e ativa-se recursivamente o algoritmo para os sub-vetores $V[0,j-1]$ e $V[j+1,9]$.

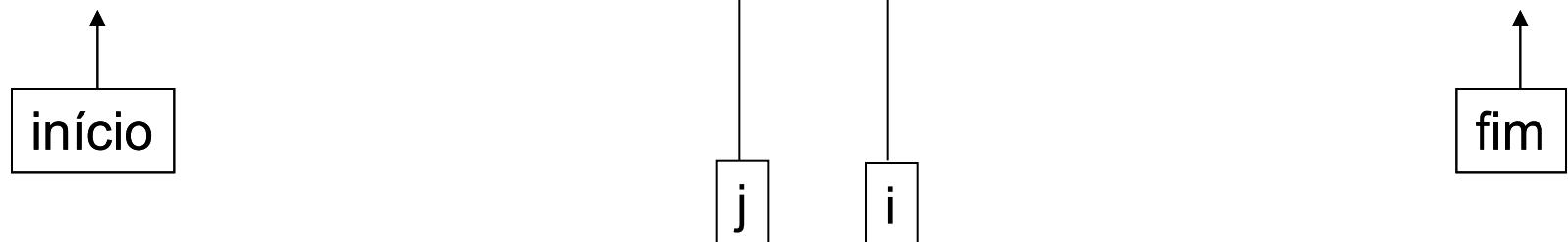
Ordenação pelo Método da Partição (QuickSort)

valores:

pivo = valores[início] = 7

| | | | | | | | | | |
|---|---|---|---|---|----|---|----|----|----|
| 6 | 3 | 4 | 1 | 7 | 18 | 8 | 10 | 19 | 12 |
|---|---|---|---|---|----|---|----|----|----|

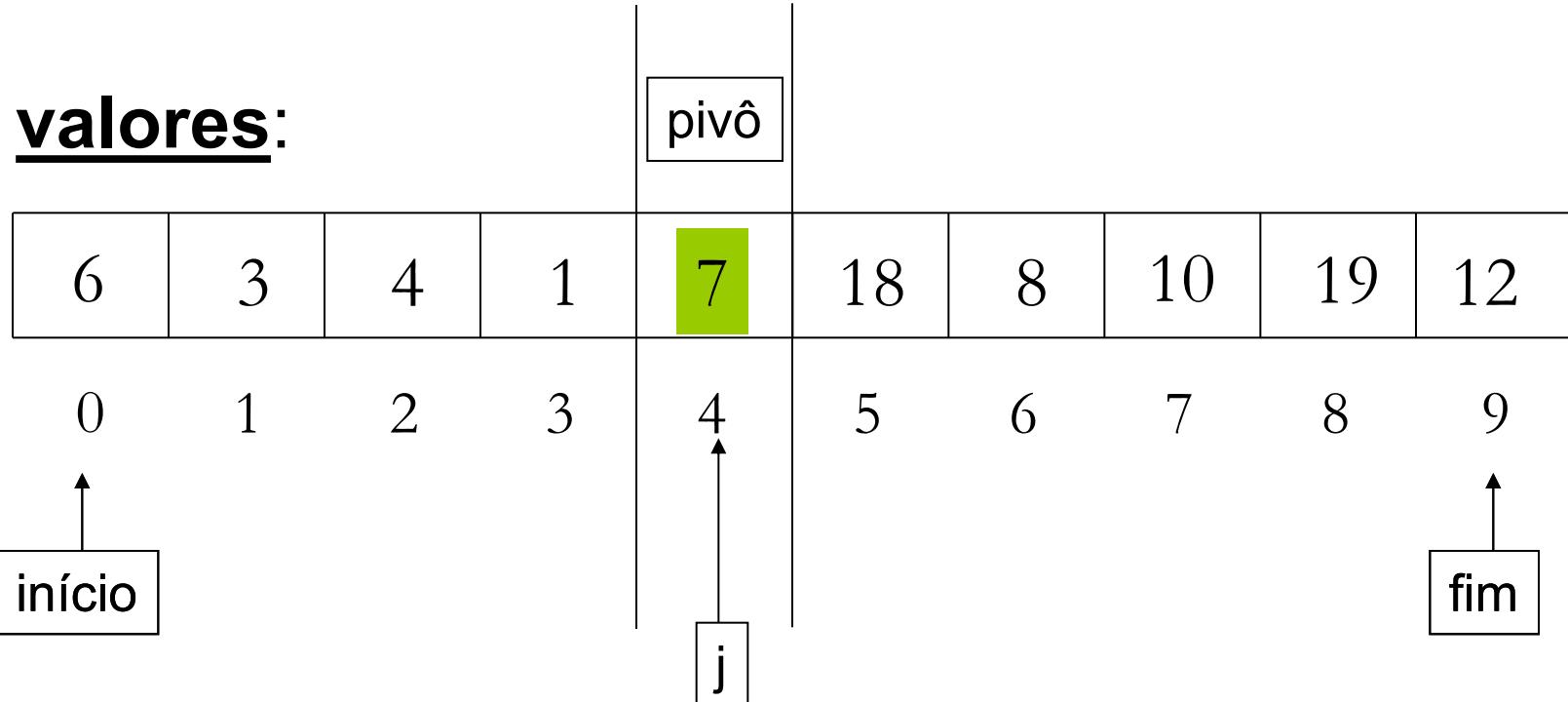
0 1 2 3 4 5 6 7 8 9



Dado que $j < i$, encerra-se a repetição. Em seguida coloca-se o pivô na sua posição definitiva e ativa-se recursivamente o algoritmo para os sub-vetores $V[0,j-1]$ e $V[j+1,9]$.

Ordenação pelo Método da Partição (QuickSort)

valores:



Observe que todos os elementos em $\text{valores}[0..j-1]$ são menores que o pivô e os elementos em $\text{valores}[j+1..9]$ são maiores ou iguais ao pivô.

```
def particiona(vals, inicio, fim):
    pivo = vals[inicio]
    i = inicio+1
    j = fim
    while i < j:
        while i<fim and vals[i] < pivo:
            i += 1
        while j>inicio and vals[j] >= pivo:
            j -= 1
        if i < j:
            trocar(vals, i, j)
    if pivo>vals[j]:
        trocar(vals, inicio, j)
    return j
def quickSort(vals, inicio, fim):
    if inicio < fim:
        posPivo = particiona(vals,inicio,fim)
        quickSort(vals,inicio,posPivo-1)
        quickSort(vals,posPivo+1,fim)
    return None
def ordena (valores):
    quickSort(valores, 0, len(valores)-1)
    return None
```

```

def particiona(vals, inicio, fim):
    pivo = vals[inicio]
    i = inicio+1
    j = fim
    while i < j:
        while i<fim and vals[i] < pivo:
            i += 1
        while j>inicio and vals[j] >= pivo:
            j -= 1
        if i < j:
            trocar(vals, i, j)
    if pivo>vals[j]:
        trocar(vals, inicio, j)
    return j

def quickSort(vals, inicio, fim):
    if inicio < fim:
        posPivo = particiona(vals,inicio,fim)
        quickSort(vals,inicio,posPivo-1)
        quickSort(vals,posPivo+1,fim)
    return None

def ordena (valores):
    quickSort(valores, 0, len(valores)-1)
    return None

```

```

def particiona(vals, inicio, fim):
    pivo = vals[inicio]
    i = inicio+1
    j = fim
    while i < j:
        while i<fim and vals[i] < pivo:
            i += 1
        while j>inicio and vals[j] >= pivo:
            j -= 1
        if i < j:
            trocar(vals, i, j)
    if pivo>vals[j]:
        trocar(vals, inicio, j)
    return j

```

```

def quickSort(vals, inicio, fim):
    if inicio < fim:
        posPivo = particiona(vals,inicio,fim)
        quickSort(vals,inicio,posPivo-1)
        quickSort(vals,posPivo+1,fim)
    return None
def ordena (valores):
    quickSort(valores, 0, len(valores)-1)
    return None

```

Ordenação pelo Método da Partição (QuickSort)

- No pior caso, este algoritmo executará aproximadamente $n(n-1)/2$ comparações.
 - Isto acontecerá quando o valor pivô gerar sempre dois subvetores, sendo um deles vazio e o outro subvetor com os demais elementos a menos do pivô, com $n-1$ elementos.

Ordenação pelo Método da Partição (QuickSort)

- No pior caso, este algoritmo executará aproximadamente $n(n-1)/2$ comparações.
 - Isto acontecerá quando o valor pivô gerar sempre dois subvetores, sendo um deles vazio e o outro subvetor com os demais elementos a menos do pivô, com $n-1$ elementos.
- Portanto, também se trata de um algoritmo $O(n^2)$.

Ordenação pelo Método da Partição (QuickSort)

- No pior caso, este algoritmo executará aproximadamente $n(n-1)/2$ comparações.
 - Isto acontecerá quando o valor pivô gerar sempre dois subvetores, sendo um deles vazio e o outro subvetor com os demais elementos a menos do pivô, com $n-1$ elementos.
- Portanto, também se trata de um algoritmo $O(n^2)$.
- Porém, é um algoritmo mais eficiente do que os demais apresentados, pois, na média, executará $n \log_2(n)$, onde o fator $\log_2(n)$ se deve a divisão recursiva do vetor original.

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:



Aula 7

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Moda
- Algoritmos de Ordenação
 - Método da Seleção (*SelectionSort*)
 - Método da Bolha (*BubbleSort*)
 - Método da Partição (*QuickSort*)
- Noções de Complexidade de Algoritmos



Aula 8

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Dados
 - Listas
 - Listas de Listas

Lista

- Uma lista é uma sequência ordenada pelo índice, de zero ou mais referências a objetos (ponteiros para objetos).

Lista

- Uma lista é uma sequência ordenada pelo índice, de zero ou mais referências a objetos (ponteiros para objetos).
- É uma estrutura de dado recursiva
 - Quando tem zero elementos é representada pela lista vazia: [].
 - Quando tem um ou mais elementos, pode ser representada por uma sequência, fechada por colchetes ([e]), de um ou mais elementos separados por vírgulas.
 - O primeiro elemento da lista está na posição zero.

Lista

- Uma lista é uma sequência ordenada pelo índice, de zero ou mais referências a objetos (ponteiros para objetos).
- É uma estrutura de dado recursiva
 - Quando tem zero elementos é representada pela lista vazia: [].
 - Quando tem um ou mais elementos, pode ser representada por uma sequência, fechada por colchetes ([e]), de um ou mais elementos separados por vírgulas.
 - O primeiro elemento da lista está na posição zero.
- Listas são mutáveis, portanto podem receber novos elementos, substituir elementos existentes ou remover antigos elementos.

Lista

Exemplos:

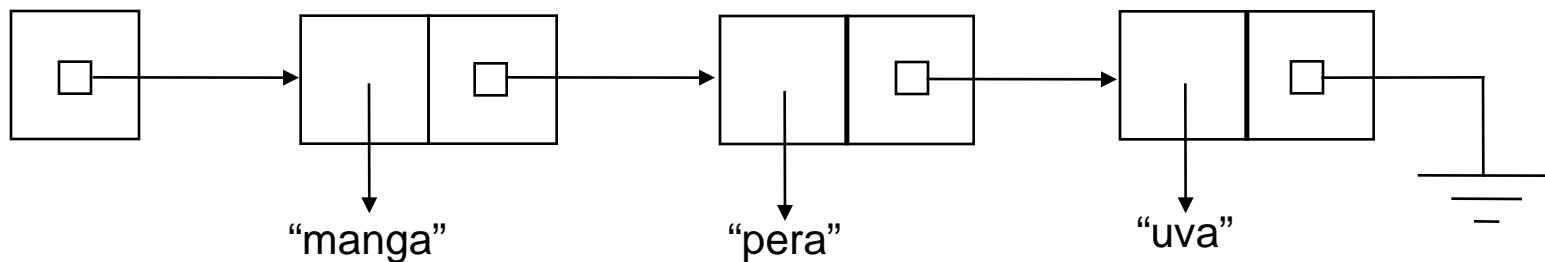
salada = []

salada



salada = ["manga", "pera", "uva"]

salada



Lista (Operações)

- Operações para inclusão de novos elementos:
 - **append(*novoElemento*)**: anexa o *novoElemento* no final da lista;
 - **insert(*pos*, *novoElemento*)**: insere o *novoElemento* na posição *pos* da lista. Caso a lista tenha menos que *pos* elementos, o *novoElemento* é inserido no final da lista.

Lista (Operações)

- Operações para inclusão de novos elementos:
 - **append(*novoElemento*)**: anexa o *novoElemento* no final da lista;
 - **insert(*pos*, *novoElemento*)**: insere o *novoElemento* na posição *pos* da lista. Caso a lista tenha menos que *pos* elementos, o *novoElemento* é inserido no final da lista.
- Exemplos:

```
salada = ["manga", "pera", "uva"]
```

```
salada.append("banana")
# salada = ["manga", "pera", "uva", "banana"]
```

```
salada.insert(2, "goiaba")
# salada = ["manga", "pera", "goiaba", "uva", "banana"]
```

Exemplo

Criando uma lista de números com quantidade de números aleatórios e um intervalo de valores escolhidos pelo usuário.

Subprogramas

```
def preencher(listaElems, qtd, min, max):
    from random import randint
    for item in range(qtd):
        listaElems.append(randint(min,max))
    return None
```

Programa Principal

```
qtdNumeros = int(input("A Lista deve ter quantos valores?: "))
minimo = int(input("Menor valor da faixa: "))
maximo = int(input("Maior valor da faixa: "))
numeros = []
preencher(numeros, qtdNumeros, minimo, maximo)
print(numeros)
```

Lista (Operações)

- Operações para remoção de elementos:
 - **pop()**: retorna e remove o último elemento da lista, o mais a direita.
 - Lança uma exceção “IndexError: pop from empty list”, se aplicado a uma lista vazia.

Lista (Operações)

- Operações para remoção de elementos:
 - **pop()**: retorna e remove o último elemento da lista, o mais a direita.
 - Lança uma exceção “IndexError: pop from empty list”, se aplicado a uma lista vazia.
 - **pop(pos)**: retorna e remove o item na posição *pos* da lista.
 - Lança uma exceção “IndexError: pop index out of range”, se aplicado a uma posição inexistente.

Lista (Operações)

- Operações para remoção de elementos:
 - **pop()**: retorna e remove o último elemento da lista, o mais a direita.
 - Lança uma exceção “IndexError: pop from empty list”, se aplicado a uma lista vazia.
 - **pop(pos)**: retorna e remove o item na posição *pos* da lista.
 - Lança uma exceção “IndexError: pop index out of range”, se aplicado a uma posição inexistente.
 - **remove(x)**: remove a primeira ocorrência do item *x*, da esquerda para a direita.
 - Lança uma exceção “ValueError” se *x* não for encontrado.

Lista (Operações)

- Outras operações úteis sobre listas:
 - *lista[pos]*: retorna o elemento da *lista* na posição *pos*.
 - **len(*lista*)**: retorna o comprimento da *lista*.
 - *lista.count(elemento)*: retorna quantas vezes o *elemento* ocorre na *lista*.
 - *lista.sort()*: ordena o conteúdo da *lista*, se os elementos forem todos do mesmo tipo.
 - Caso contrário levanta uma exceção “`TypeError: unorderable types`”.

Exemplo

Crie uma lista com 100 números aleatórios no intervalo 0 a 40. Remova da lista todos os valores duplicados e mostre seu conteúdo.

```
# Subprogramas
def preencher(listaElems, qtd, min, max):
    ...
def removerDuplicatas(elems):
    indice = 0
    while indice<len(elems):
        if elems.count(elems[indice])==1:
            indice += 1
        else:
            elems.remove(elems[indice])
    return None
# Programa Principal
numeros = []
preencher(numeros, 100, 0, 40)
print(numeros)
removerDuplicatas(numeros)
print(numeros)
```

Lista (Operações)

- Fatiamento de Listas:
 - `antiga[posInicio : posAposFim]`: retorna um nova lista composta de referências para os elementos existentes, iniciando-se por elemento na posição `posInicio` e finalizando por elemento na posição anterior ao `posAposFim`.

Lista (Operações)

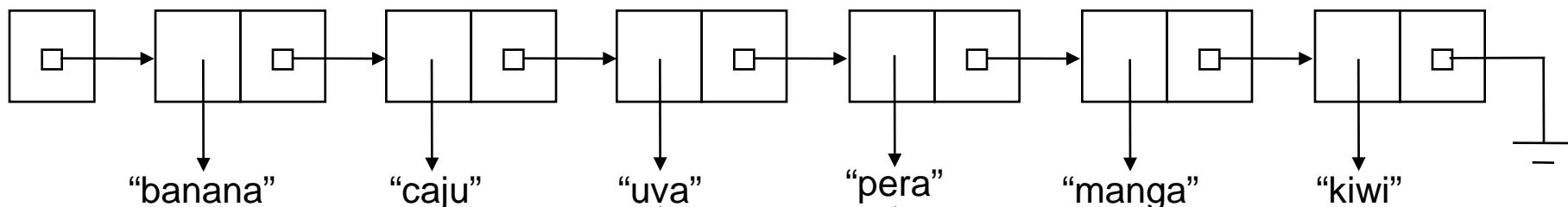
- Fatiamento de Listas:

- `antiga[posInicio : posAposFim]`: retorna um nova lista composta de referências para os elementos existentes, iniciando-se por elemento na posição `posInicio` e finalizando por elemento na posição anterior ao `posAposFim`.

- Exemplo:

```
saladaComposta = ["banana", "caju", "uva", "pera", "manga", "kiwi"]
saladaSimples = saladaComposta[1:4] # lista ["caju", "uva", "pera"]
```

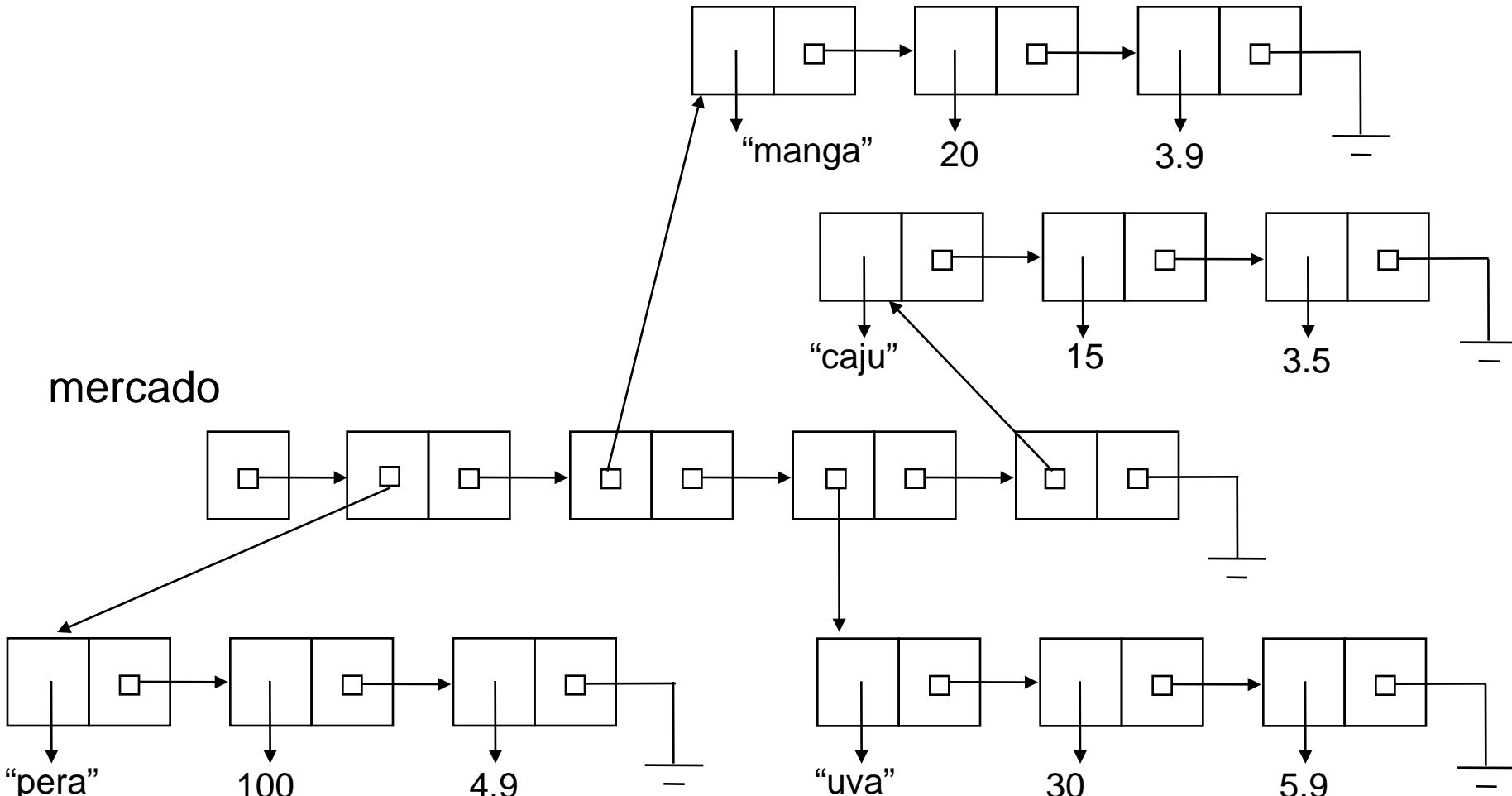
`saladaComposta`



`saladaSimples`

Lista de Listas

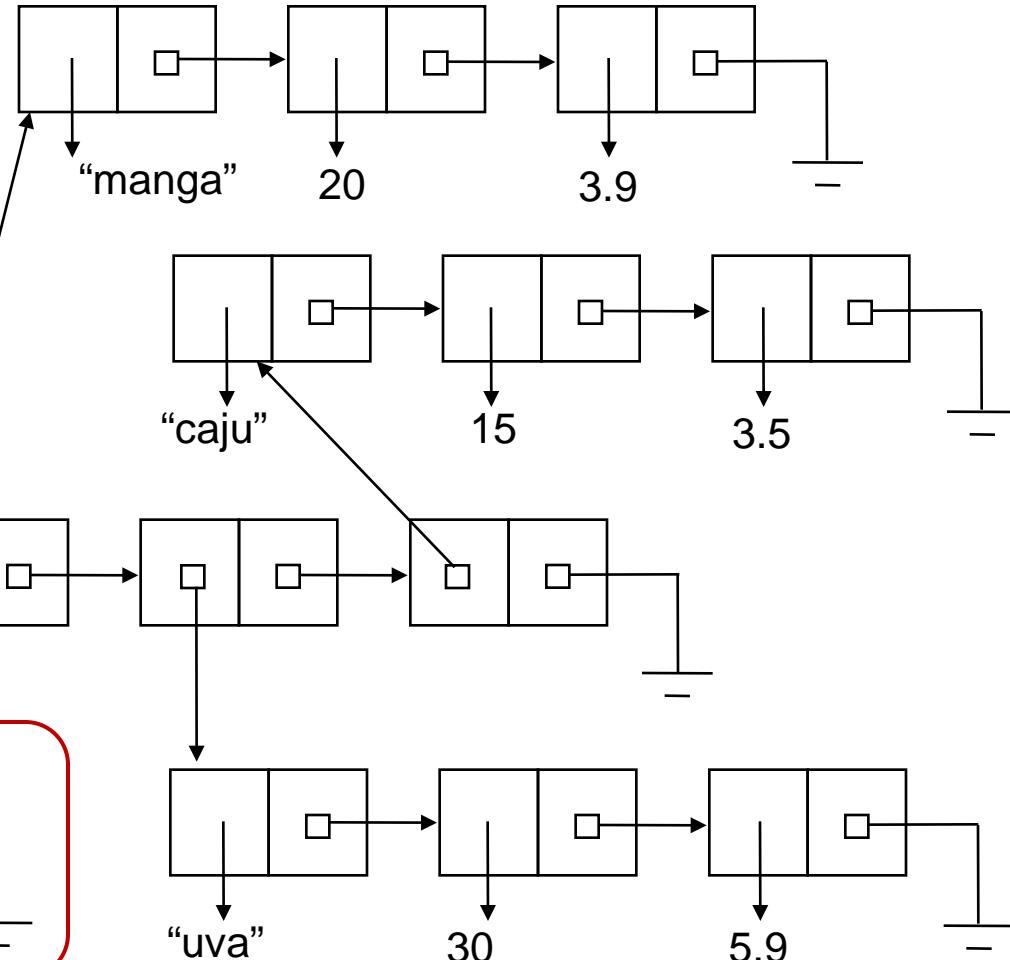
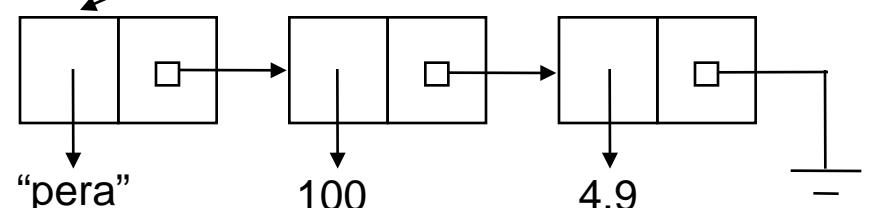
```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]
```



Lista de Listas

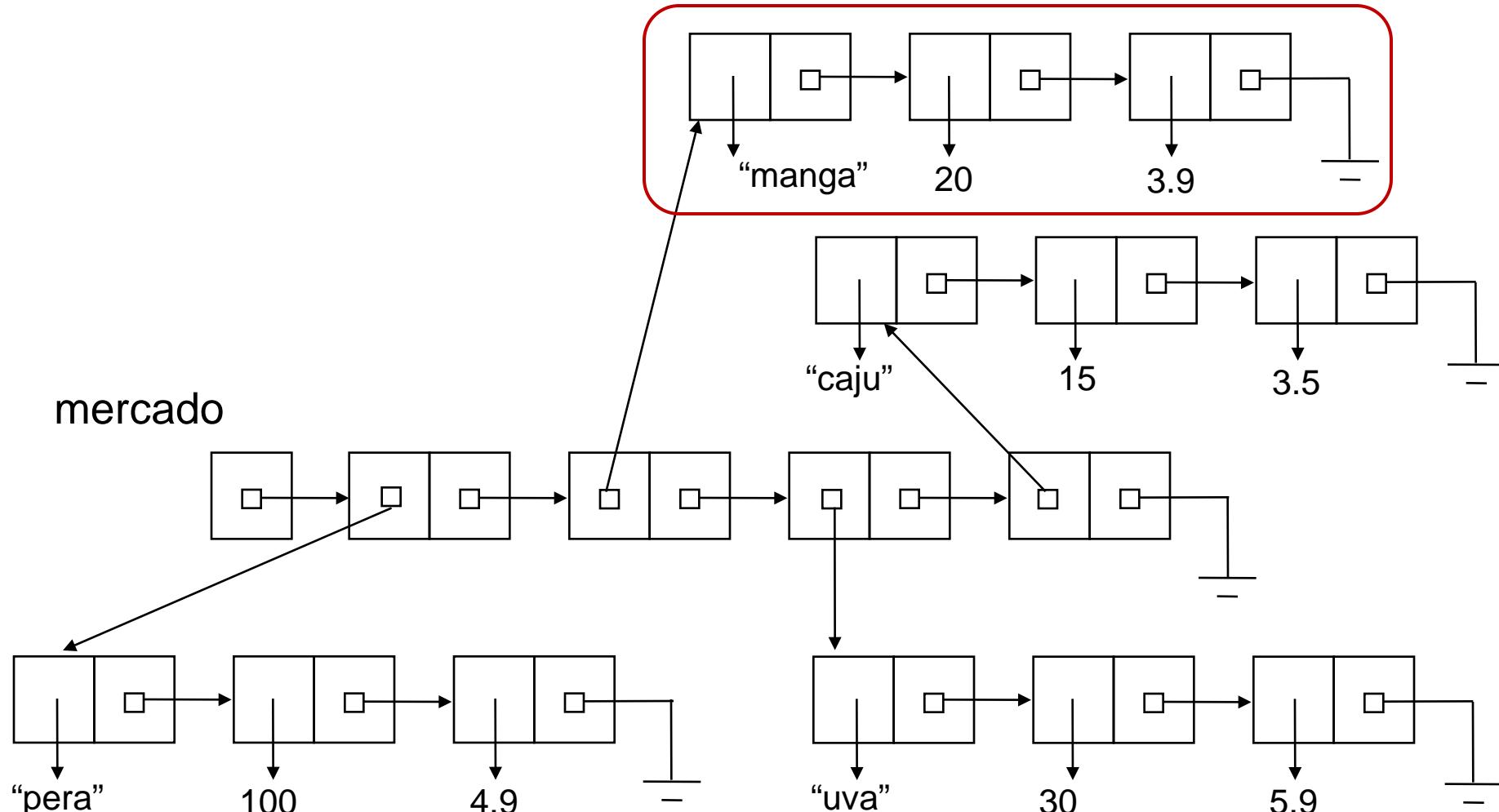
```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]
```

mercado



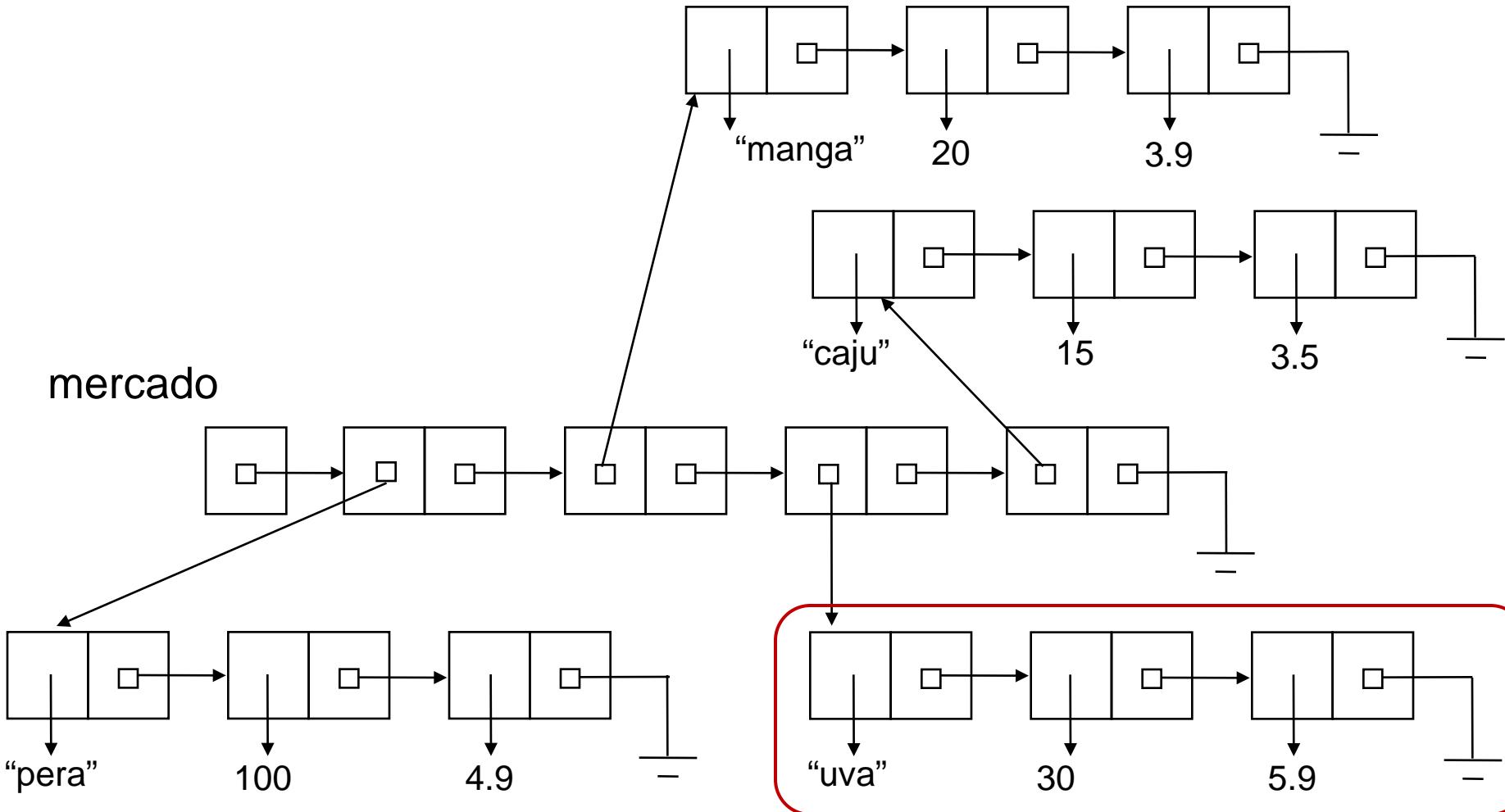
Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]
```



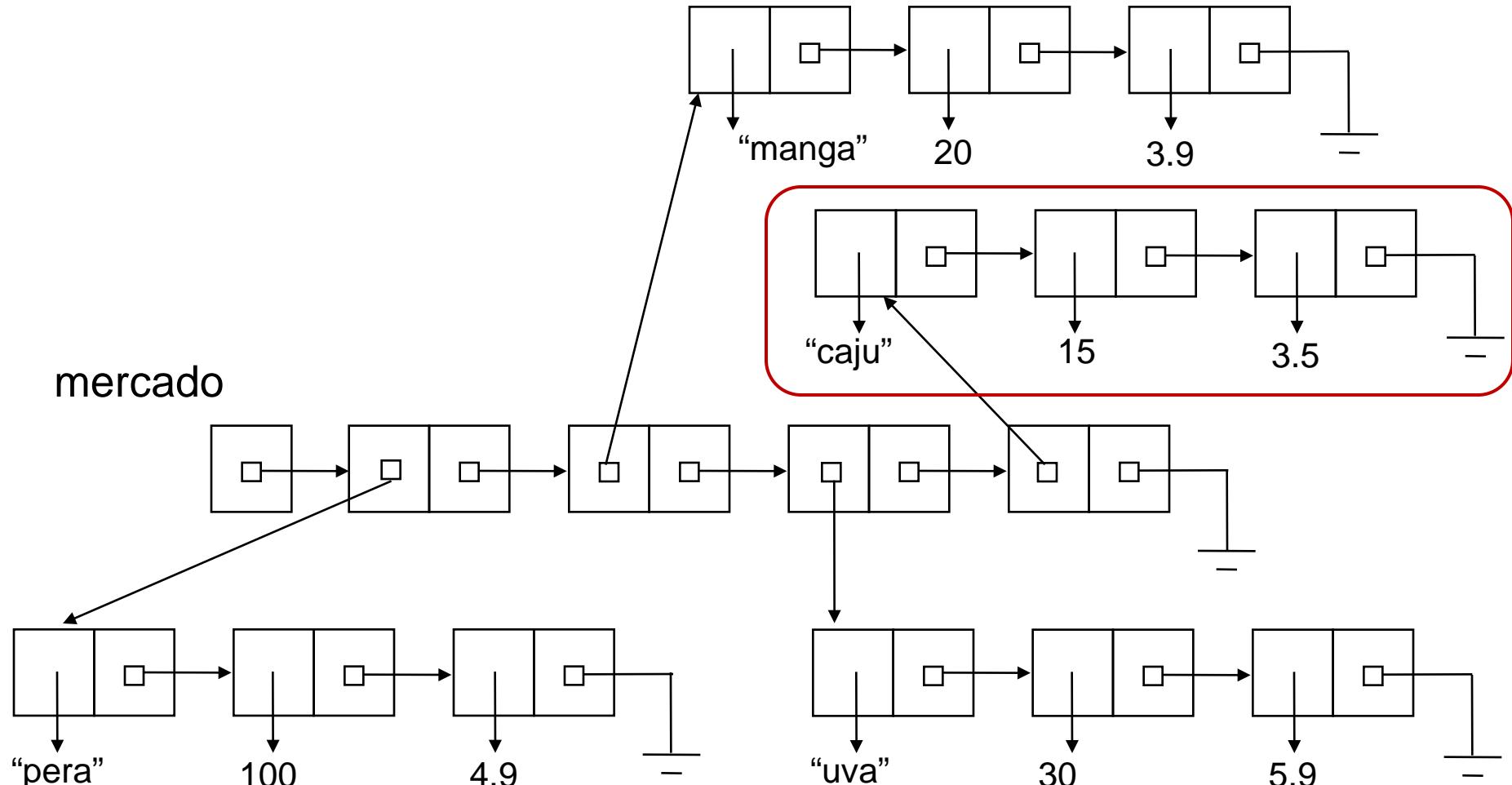
Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]
```



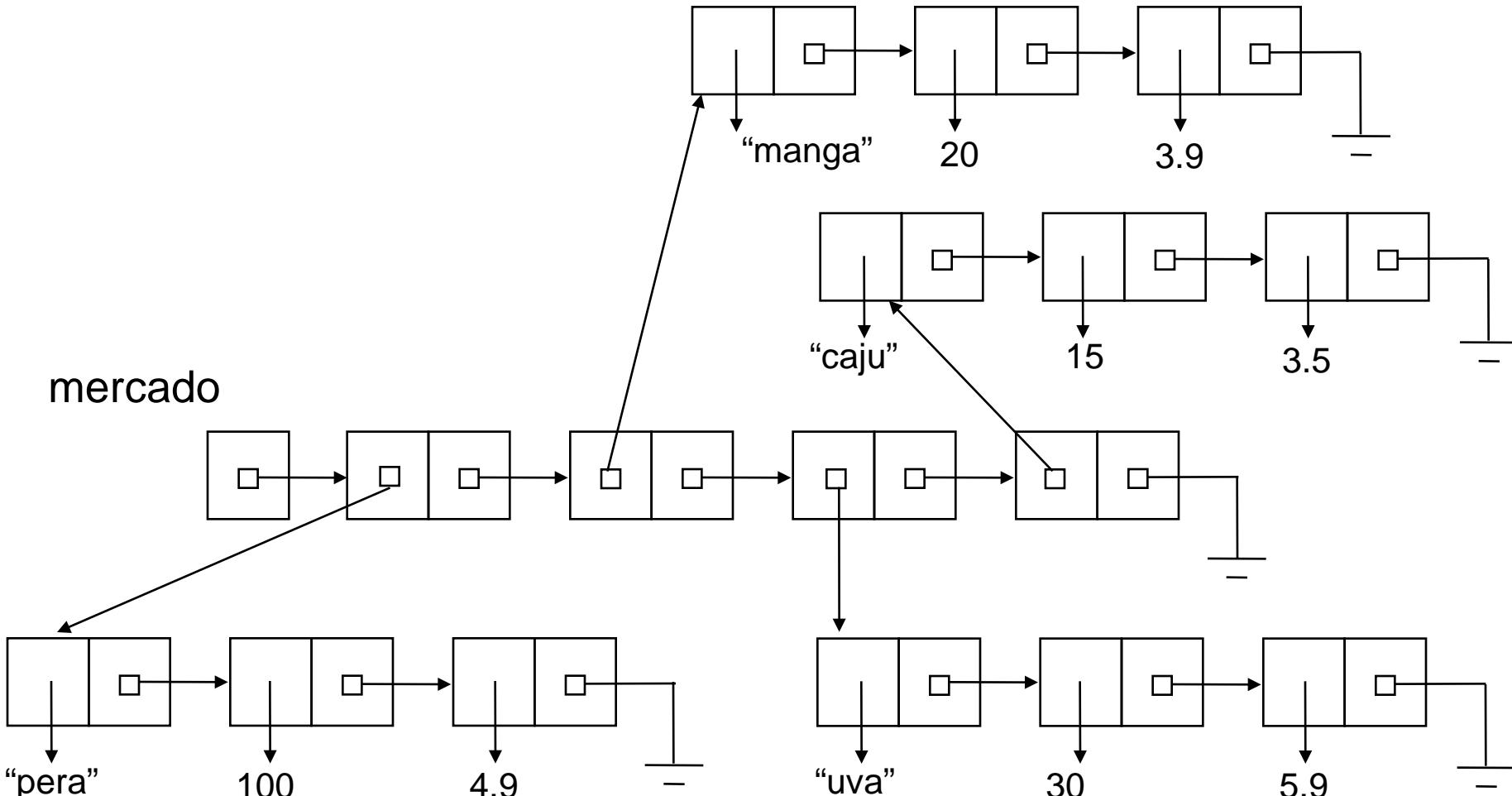
Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]
```



Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]
```



Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]  
print(mercado)
```

```
mercado[1][2] *= 0.5 # manga pela metade do preço  
print(mercado)
```

Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]  
print(mercado)
```

```
mercado[1][2] *= 0.5 # manga pela metade do preço  
print(mercado)
```

```
mercado[3][1] -= 10 # caju com dez quilos a menos  
print(mercado)
```

Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]  
print(mercado)
```

```
mercado[1][2] *= 0.5 # manga pela metade do preço  
print(mercado)
```

```
mercado[3][1] -= 10 # caju com dez quilos a menos  
print(mercado)
```

```
mercado.remove(["uva",30,5.9]) # o produto uva é removido do mercado  
print(mercado)
```

Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]  
print(mercado)
```

```
mercado[1][2] *= 0.5 # manga pela metade do preço  
print(mercado)
```

```
mercado[3][1] -= 10 # caju com dez quilos a menos  
print(mercado)
```

```
mercado.remove(["uva",30,5.9]) # o produto uva é removido do mercado  
print(mercado)
```

```
mercado.insert(1, ["kiwi", 200, 1.99]) # o produto kiwi foi inserida  
print(mercado)
```

Lista de Listas

```
mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30, 5.9], ["caju", 15, 3.5]]  
print(mercado)
```

```
mercado[1][2] *= 0.5 # manga pela metade do preço  
print(mercado)
```

```
mercado[3][1] -= 10 # caju com dez quilos a menos  
print(mercado)
```

```
mercado.remove(["uva",30,5.9]) # o produto uva é removido do mercado  
print(mercado)
```

```
mercado.insert(1, ["kiwi", 200, 1.99]) # o produto kiwi foi inserida  
print(mercado)
```

```
# mercado = [["pera", 100, 4.9], ["kiwi", 200, 1.99], ["manga", 20, 1.95], ["caju", 5, 3.5]]
```

Lista (*Lisp-Like*)

- No sentido de exercitar a elaboração de subprogramas recursivos, apresentamos as três operações primitivas do paradigma de programação funcional da linguagem Lisp (*List Processing*):
 - car(dados): é a operação seletora que retorna o primeiro elemento de uma lista dados;
 - cdr(dados): é a operação seletora que retorna o resto da lista dados, isto é, retorna uma lista com todos os elementos da lista dados, exceto pelo primeiro;
 - cons(item, dados): é a operação construtora que retorna uma lista que contém o item como primeiro elemento, seguido pela lista dados.

Lista (*Lisp-Like*)

- No sentido de exercitar a elaboração de subprogramas recursivos, apresentamos as três operações primitivas do paradigma de programação funcional da linguagem Lisp (*List Processing*):
 - car(dados): é a operação seletora que retorna o primeiro elemento de uma lista dados;
 - cdr(dados): é a operação seletora que retorna o resto da lista dados, isto é, retorna uma lista com todos os elementos da lista dados, exceto pelo primeiro;
 - cons(item, dados): é a operação construtora que retorna uma lista que contém o item como primeiro elemento, seguido pela lista dados.
- Implementação em Python das operações do Lisp:

```
def car(dados):  
    return dados[0]  
def cdr(dados):  
    return dados[1:len(dados)]  
def cons(item, dados):  
    return [item]+ dados
```

Processamento Recursivo de Listas (*Lisp-Like*)

- Exemplo:
 - Utilizando as operações seletoras **car** e **cdr**, faça uma função recursiva que some o conteúdo de uma lista de números recebida como parâmetro.

Processamento Recursivo de Listas (*Lisp-Like*)

- Exemplo:
 - Utilizando as operações seletoras **car** e **cdr**, faça uma função recursiva que some o conteúdo de uma lista de números recebida como parâmetro.
- Implementação:

```
def soma(dados):  
    if dados == [ ]:  
        return 0  
    else:  
        return car(dados) + soma(cdr(dados))
```

Lista de Listas (*Lisp-Like*)

- Os elementos de uma lista podem ser listas.
 - Portanto, precisamos saber o estado de cada elemento da lista.
 - Para isto, as operações abaixo identificam se um item é uma lista ou se é átomo, isto é, não é uma lista, respectivamente:

```
def ehLista(item):  
    return isinstance(item, list)
```

```
def ehAtomo(item):  
    return not ehLista(item)
```

Lista de Listas (*Lisp-Like*)

- Os elementos de uma lista podem ser listas.
 - Portanto, precisamos saber o estado de cada elemento da lista.
 - Para isto, as operações abaixo identificam se um item é uma lista ou se é átomo, isto é, não é uma lista, respectivamente:

```
def ehLista(item):
    return isinstance(item, list)
```

```
def ehAtomo(item):
    return not ehLista(item)
```

- Com estes estados, podemos processar recursivamente uma lista de lista de números e encontrar o valor da soma de todos os valores numéricos:

```
def soma(dados):
    if dados == []:
        return 0
    else:
        if ehAtomo(car(dados)):
            return car(dados) + soma(cdr(dados))
        else:
            return soma(car(dados)) + soma(cdr(dados))
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:





Aula 8

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estruturas de Dados
 - Listas
 - Listas de Listas



Aula 9

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Persistência de Dados
 - Arquivo Texto

Arquivos

- Os programas apresentados neste curso até o momento são chamados **interativos**.
 - Programas interativos leem os dados de entrada do teclado e apresentam os dados de saída na tela.
 - Este tipo de programação é utilizada quando poucos dados são processados ou quando necessitam de interação humana.

Arquivos

- Os programas apresentados neste curso até o momento são chamados **interativos**.
 - Programas interativos leem os dados de entrada do teclado e apresentam os dados de saída na tela.
 - Este tipo de programação é utilizada quando poucos dados são processados ou quando necessitam de interação humana.
- Quando grandes quantidades de dados são processadas, **arquivos** são utilizados para armazenar os dados de entrada e os de saída.
 - Estes são chamados, respectivamente, arquivos de entrada e arquivos de saída.

Arquivo Texto

- Há basicamente dois tipos de arquivo: **texto** e **binário**.
 - Arquivos binários serão vistos nas próximas aulas.

Arquivo Texto

- Há basicamente dois tipos de arquivo: **texto** e **binário**.
 - Arquivos binários serão vistos nas próximas aulas.
- Um **arquivo texto** é uma sequência de caracteres, organizada em linhas, que reside em uma área de armazenamento (e.g., disco rígido, pen drive, CD/DVD) sob um mesmo nome.

Arquivo Texto

- Há basicamente dois tipos de arquivo: **texto** e **binário**.
 - Arquivos binários serão vistos nas próximas aulas.
- Um **arquivo texto** é uma sequência de caracteres, organizada em linhas, que reside em uma área de armazenamento (e.g., disco rígido, pen drive, CD/DVD) sob um mesmo nome.
- Arquivos texto podem ser criados, visualizados e alterados por editores ou processadores de texto.
 - O arquivo de entrada de um programa pode ser criado em um editor de texto e o arquivo de saída pode ser consultado utilizando-se também um editor.

Arquivo Texto

- Um arquivo texto é armazenado em disco como uma sequência de caracteres.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|--|---|---|---|---|--|---|---|---|---|----|---|---|---|--|---|---|---|---|---|---|----|----|
| 1 | 0 | 0 | | C | a | b | o | | F | r | i | o | \n | 2 | 6 | 1 | | L | a | g | u | n | a | \n | \0 |
|---|---|---|--|---|---|---|---|--|---|---|---|---|----|---|---|---|--|---|---|---|---|---|---|----|----|

Arquivo Texto

- Um arquivo texto é armazenado em disco como uma sequência de caracteres.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|--|---|---|---|---|--|---|---|---|---|----|---|---|---|--|---|---|---|---|---|---|----|----|
| 1 | 0 | 0 | | c | a | b | o | | F | r | i | o | \n | 2 | 6 | 1 | | L | a | g | u | n | a | \n | \0 |
|---|---|---|--|---|---|---|---|--|---|---|---|---|----|---|---|---|--|---|---|---|---|---|---|----|----|

52 bytes para UNICODE

- O arquivo acima contém 26 caracteres, entre eles, dígitos, brancos, letras e caracteres especiais: “\n” e “\0”.

Arquivo Texto

- Um arquivo texto é armazenado em disco como uma sequência de caracteres.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|--|---|---|---|---|--|---|---|---|---|----|---|---|---|--|---|---|---|---|---|---|----|----|
| 1 | 0 | 0 | | c | a | b | o | | F | r | i | o | \n | 2 | 6 | 1 | | L | a | g | u | n | a | \n | \0 |
|---|---|---|--|---|---|---|---|--|---|---|---|---|----|---|---|---|--|---|---|---|---|---|---|----|----|

52 bytes para UNICODE

- O arquivo acima contém 26 caracteres, entre eles, dígitos, brancos, letras e caracteres especiais: “\n” e “\0”.
- O caractere “\n” indica fim de linha e o caractere “\0” indica fim do arquivo.

1 decimal 49

A decimal 65

\n decimal 10

\0 decimal 0

Abrindo um Arquivo de Texto

- Em Python, antes de ser utilizado, um arquivo texto precisa ser associado a um nome no diretório de arquivos e ser aberto, via operação **open**:

*variável = **open**(caminho do arquivo)*

ou

*variável = **open**(caminho do arquivo, modo)*

Abrindo um Arquivo de Texto

- Em Python, antes de ser utilizado, um arquivo texto precisa ser associado a um nome no diretório de arquivos e ser aberto, via operação **open**:

variável = open(caminho do arquivo)

ou

variável = open(caminho do arquivo, modo)

- Os modos de operação de um arquivo são:
“r” : apenas leitura (se omitido = “r”);

Abrindo um Arquivo de Texto

- Em Python, antes de ser utilizado, um arquivo texto precisa ser associado a um nome no diretório de arquivos e ser aberto, via operação **open**:

variável = open(caminho do arquivo)

ou

variável = open(caminho do arquivo, modo)

- Os modos de operação de um arquivo são:
 - “r” : apenas leitura (se omitido = “r”);
 - “w” : apenas escrita;

Abrindo um Arquivo de Texto

- Em Python, antes de ser utilizado, um arquivo texto precisa ser associado a um nome no diretório de arquivos e ser aberto, via operação **open**:

variável = open(caminho do arquivo)

ou

variável = open(caminho do arquivo, modo)

- Os modos de operação de um arquivo são:
 - “r” : apenas leitura (se omitido = “r”);
 - “w” : apenas escrita;
 - “a” : escrita no final do arquivo;

Abrindo um Arquivo de Texto

- Em Python, antes de ser utilizado, um arquivo texto precisa ser associado a um nome no diretório de arquivos e ser aberto, via operação **open**:

variável = open(caminho do arquivo)

ou

variável = open(caminho do arquivo, modo)

- Os modos de operação de um arquivo são:

“r” : apenas leitura (se omitido = “r”);

“w” : apenas escrita;

“a” : escrita no final do arquivo;

“r+” : leitura e escrita (não visto aqui).

Abrindo um Arquivo de Texto

```
dados = open("teste.txt", "r")
```

Caso o arquivo exista: abre para leitura o arquivo “teste.txt”, e coloca a cabeça de leitura sobre o primeiro caractere da primeira linha.

Caso ele não exista: causa erro **FileNotFoundException**.

Abrindo um Arquivo de Texto

```
dados = open("teste.txt", "r")
```

Caso o arquivo exista: abre para leitura o arquivo “teste.txt”, e coloca a cabeça de leitura sobre o primeiro caractere da primeira linha.

Caso ele não exista: causa erro **FileNotFoundException**.

```
dados = open("teste.txt", "w")
```

Caso o arquivo exista: apaga seu conteúdo antigo e coloca a cabeça de escrita no início do arquivo.

Caso ele não exista: cria o arquivo no diretório e coloca a cabeça de escrita no início do arquivo.

Abrindo um Arquivo de Texto

```
dados = open("teste.txt", "r")
```

Caso o arquivo exista: abre para leitura o arquivo “teste.txt”, e coloca a cabeça de leitura sobre o primeiro caractere da primeira linha.

Caso ele não exista: causa erro **FileNotFoundException**.

```
dados = open("teste.txt", "w")
```

Caso o arquivo exista: apaga seu conteúdo antigo e coloca a cabeça de escrita no início do arquivo.

Caso ele não exista: cria o arquivo no diretório e coloca a cabeça de escrita no início do arquivo.

```
dados = open("teste.txt", "a")
```

Caso o arquivo exista: abre o arquivo para escrita e coloca a cabeça de escrita no fim do arquivo. Isto é: pronto a anexar novas informações no seu final.

Caso ele não exista: cria o arquivo no diretório e coloca a cabeça de escrita no fim do arquivo, que neste caso é igual ao início. 17

Fechando um Arquivo de Texto

A operação **close()** permite que um arquivo texto seja fechado. Sempre que não for mais ser utilizado, um arquivo deve ser fechado.

```
dados = open("exemplo.txt", "w")
```

...

Nome interno
do arquivo.

```
dados.close()
```

Este comando garante que todos os dados gravados no arquivo estarão efetivamente no diretório. A partir deste ponto, o arquivo não poderá mais ser utilizado.

O Método **readline()**

A operação **readline()**, aplicada sobre um arquivo texto aberto, retorna uma linha completa do arquivo, incluindo o fim de linha: “\n”. A cabeça de leitura avança para a próxima linha. Uma string vazia é retornada quando o fim de arquivo é encontrado.

```
dados = open("exemplo.txt", "r")
```

```
linha = dados.readline()
```

```
print(linha, end="")
```

```
dados.close()
```

O Método *readline()*

A operação **readline()**, aplicada sobre um arquivo texto aberto, retorna uma linha completa do arquivo, incluindo o fim de linha: “\n”. A cabeça de leitura avança para a próxima linha. Uma string vazia é retornada quando o fim de arquivo é encontrado.

```
dados = open("exemplo.txt", "r")
linha = dados.readline()
print(linha, end="")
dados.close()
```

A partir deste ponto,
o arquivo
“exemplo.txt”
pode ser lido.
O primeiro caracter
a ser lido será o
primeiro caracter do
arquivo (onde
estará inicialmente
a “cabeça de
leitura”).

O Método *readline()*

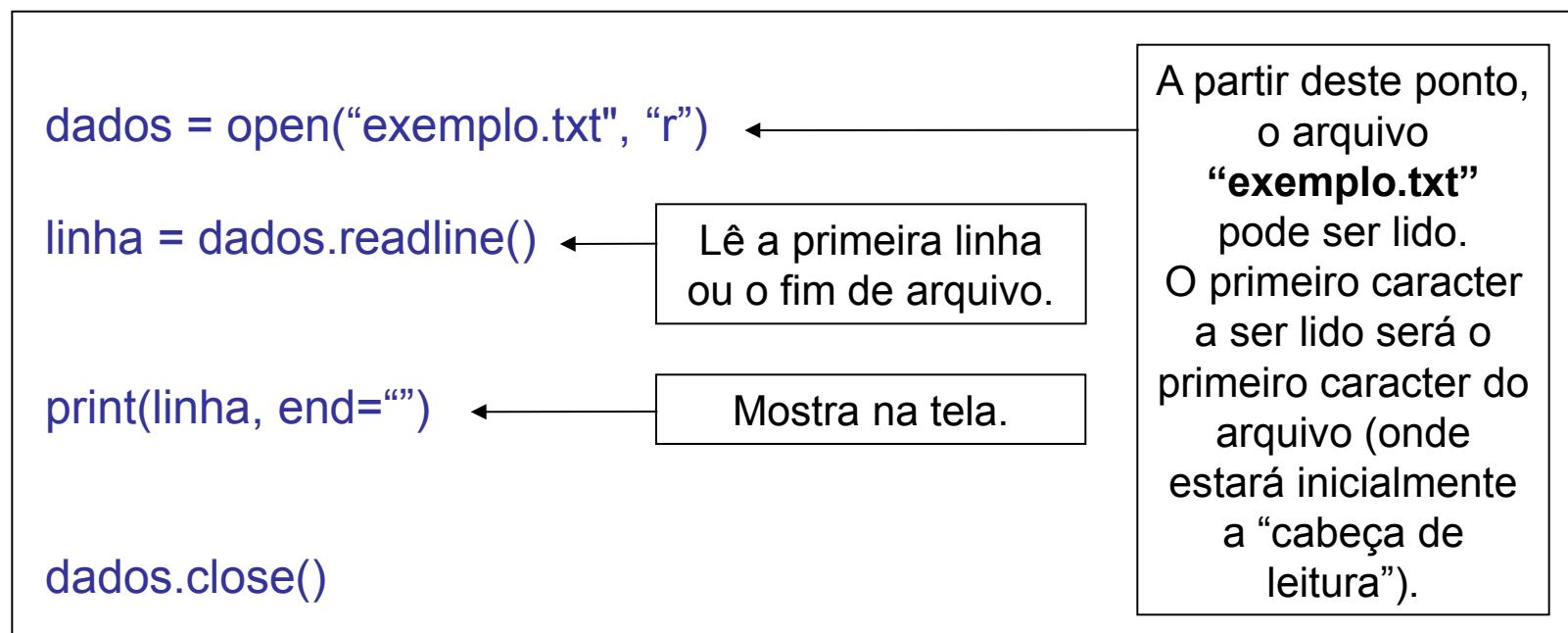
A operação **readline()**, aplicada sobre um arquivo texto aberto, retorna uma linha completa do arquivo, incluindo o fim de linha: “\n”. A cabeça de leitura avança para a próxima linha. Uma string vazia é retornada quando o fim de arquivo é encontrado.

```
dados = open("exemplo.txt", "r")  
  
linha = dados.readline() ← Lê a primeira linha  
ou o fim de arquivo.  
  
print(linha, end="")  
  
dados.close()
```

A partir deste ponto,
o arquivo
“exemplo.txt”
pode ser lido.
O primeiro caracter
a ser lido será o
primeiro caracter do
arquivo (onde
estará inicialmente
a “cabeça de
leitura”).

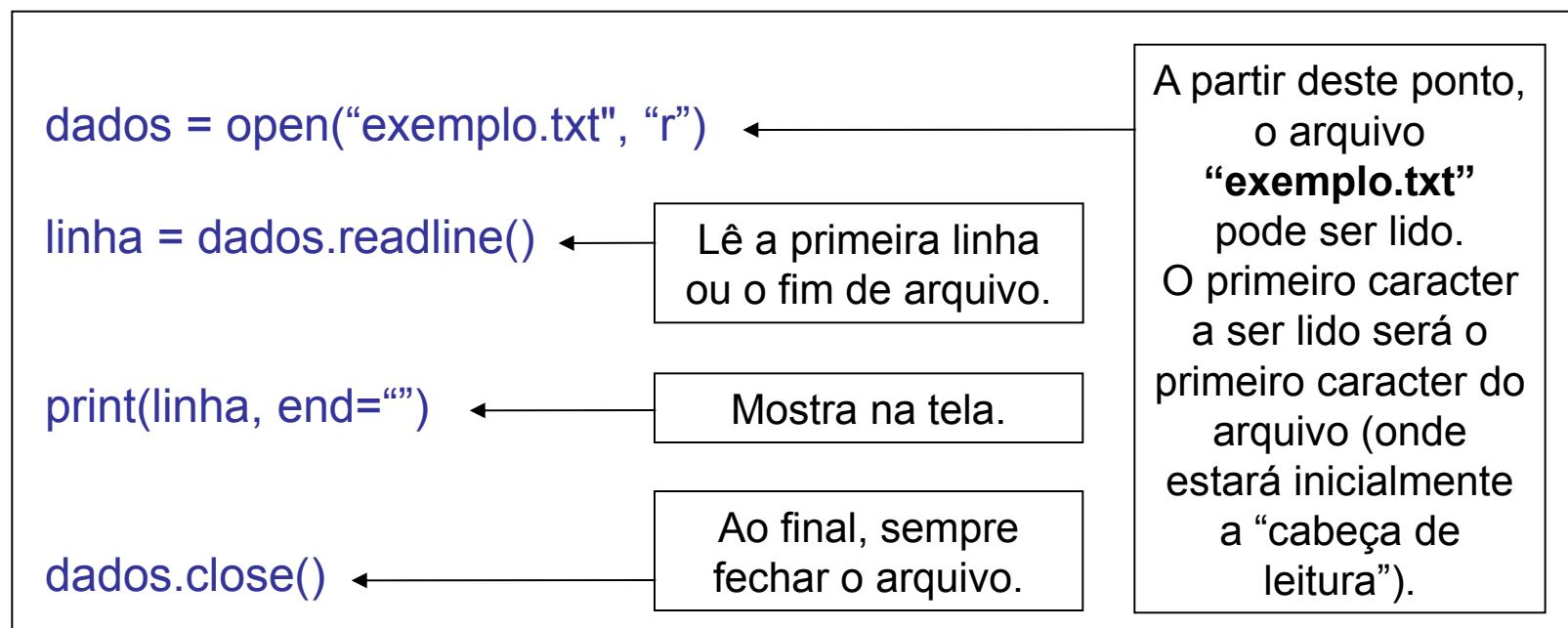
O Método *readline()*

A operação **readline()**, aplicada sobre um arquivo texto aberto, retorna uma linha completa do arquivo, incluindo o fim de linha: “\n”. A cabeça de leitura avança para a próxima linha. Uma string vazia é retornada quando o fim de arquivo é encontrado.



O Método *readline()*

A operação **readline()**, aplicada sobre um arquivo texto aberto, retorna uma linha completa do arquivo, incluindo o fim de linha: “\n”. A cabeça de leitura avança para a próxima linha. Uma string vazia é retornada quando o fim de arquivo é encontrado.



Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r")

linha = dados.readline()

while linha != "":
    print(linha, end="")

    linha = dados.readline()

dados.close()
```

Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r") ← A partir deste ponto,  
                                o arquivo  
nomeArquivo pode  
                                ser lido.  
  
linha = dados.readline()  
  
while linha != "":  
  
    print(linha, end="")  
  
    linha = dados.readline()  
  
dados.close()
```

Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r") ← A partir deste ponto,  
                                o arquivo  
                                nomeArquivo pode  
                                ser lido.  
                                O primeiro caracter  
                                a ser lido será o  
                                primeiro caracter do  
                                arquivo (onde  
                                estará inicialmente  
                                a "cabeça de  
                                leitura").  
  
linha = dados.readline() ← Lê primeira linha  
  
while linha != "":  
  
    print(linha, end="")  
  
    linha = dados.readline()  
  
dados.close()
```

Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r") ← A partir deste ponto,  
                                o arquivo  
                                nomeArquivo pode  
                                ser lido.

linha = dados.readline() ← Lê primeira linha

while linha != "": ← Enquanto não é o fim

    print(linha, end="")

    linha = dados.readline()

dados.close()
```

A partir deste ponto,
o arquivo
nomeArquivo pode
ser lido.
O primeiro caracter
a ser lido será o
primeiro caracter do
arquivo (onde
estará inicialmente
a “cabeça de
leitura”).

Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r") ← A partir deste ponto,  
                                o arquivo  
                                nomeArquivo pode  
                                ser lido.

linha = dados.readline() ← Lê primeira linha  
while linha != "": ← Enquanto não é o fim
    print(linha, end="")
    linha = dados.readline() ← Mostra na tela

dados.close()
```

Apartir deste ponto, o arquivo **nomeArquivo** pode ser lido. O primeiro caractere a ser lido será o primeiro caractere do arquivo (onde estará inicialmente a “cabeça de leitura”).

Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r") ← A partir deste ponto,  
                                o arquivo  
                                nomeArquivo pode  
                                ser lido.

linha = dados.readline() ← Lê primeira linha  
while linha != "": ← Enquanto não é o fim
    print(linha, end="") ← Mostra na tela
    linha = dados.readline() ← Lê próxima linha

dados.close()
```

Lê primeira linha

Enquanto não é o fim

Mostra na tela

Lê próxima linha

A partir deste ponto, o arquivo **nomeArquivo** pode ser lido. O primeiro caracter a ser lido será o primeiro caracter do arquivo (onde estará inicialmente a “cabeça de leitura”).

Lendo o Arquivo Texto com o Método *readline()*

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")
```

```
dados = open(nomeArquivo, "r")
```



Lê primeira linha

```
while linha != "":
```

Enquanto não é o fim

```
print(linha, end="")
```

Mostra na tela

```
linha = dados.readline()
```

Lê próxima linha

```
dados.close()
```

Ao final, sempre fechar o arquivo.

A partir deste ponto, o arquivo **nomeArquivo** pode ser lido. O primeiro caracter a ser lido será o primeiro caracter do arquivo (onde estará inicialmente a “cabeça de leitura”).

Lendo com o Iterador sobre o Arquivo

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r")

for linha in dados:

    print(linha, end="")

dados.close()
```

Lendo com o Iterador sobre o Arquivo

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")  
  
dados = open(nomeArquivo, "r") ←  
  
for linha in dados:  
  
    print(linha, end="")  
  
dados.close()
```

A partir deste ponto, o arquivo **nomeArquivo** pode ser lido. O primeiro caracter a ser lido será o primeiro caracter do arquivo (onde estará inicialmente a “cabeça de leitura”).

Lendo com o Iterador sobre o Arquivo

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")  
  
dados = open(nomeArquivo, "r") ←  
  
for linha in dados: ← Iterando linha a linha  
    print(linha, end="")  
  
dados.close()
```

A partir deste ponto, o arquivo **nomeArquivo** pode ser lido. O primeiro caracter a ser lido será o primeiro caracter do arquivo (onde estará inicialmente a “cabeça de leitura”).

Lendo com o Iterador sobre o Arquivo

O programa abaixo pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")
```

```
dados = open(nomeArquivo, "r")
```

```
for linha in dados:
```

```
    print(linha, end="")
```

```
dados.close()
```

Iterando linha a linha sobre o conteúdo de um arquivo texto.

Ao final, sempre fechar o arquivo.

A partir deste ponto, o arquivo **nomeArquivo** pode ser lido.

O primeiro caracter a ser lido será o primeiro caracter do arquivo (onde estará inicialmente a “cabeça de leitura”).

Lendo Todas as Linhas para uma Lista

O programa abaixo, que funciona apenas para pequenos arquivos, pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r")

linhas = dados.readlines()

for linha in linhas:

    print(linha, end="")

dados.close()
```

Lendo Todas as Linhas para uma Lista

O programa abaixo, que funciona apenas para pequenos arquivos, pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r")

linhas = dados.readlines()

for linha in linhas:

    print(linha, end="")

dados.close()
```

Restrição (ERRO): Será que o arquivo cabe na memória principal?

Lendo Todas as Linhas para uma Lista

O programa abaixo, que funciona apenas para pequenos arquivos, pede ao usuário que escolha um nome de arquivo, existente em seu diretório, e exibe seu conteúdo na tela.

```
nomeArquivo = input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(nomeArquivo, "r")

linhas = dados.readlines() ← Restrição (ERRO): Será que o arquivo cabe na memória principal?

for linha in linhas: ← Iterando linha a linha sobre a lista de linhas.

    print(linha, end="")

dados.close()
```

Produzindo Arquivo Texto: o Método `write()`

Para escrever uma sequência de caracteres em um arquivo texto, no modo “w” ou “a”, podemos utilizar o método **write(desejada)**. Que escreverá a String **desejada** a partir do ponto em que a cabeça de escrita do arquivo estiver posicionada. Ao final, a cabeça de escrita ficará posicionada após o último caractere da String **desejada**.

```
dados = open("teste.txt", "w")
```

Abre o arquivo “teste.txt”
em modo de escrita.

```
dados.write("qualquer dado pode ser escrito.")
```

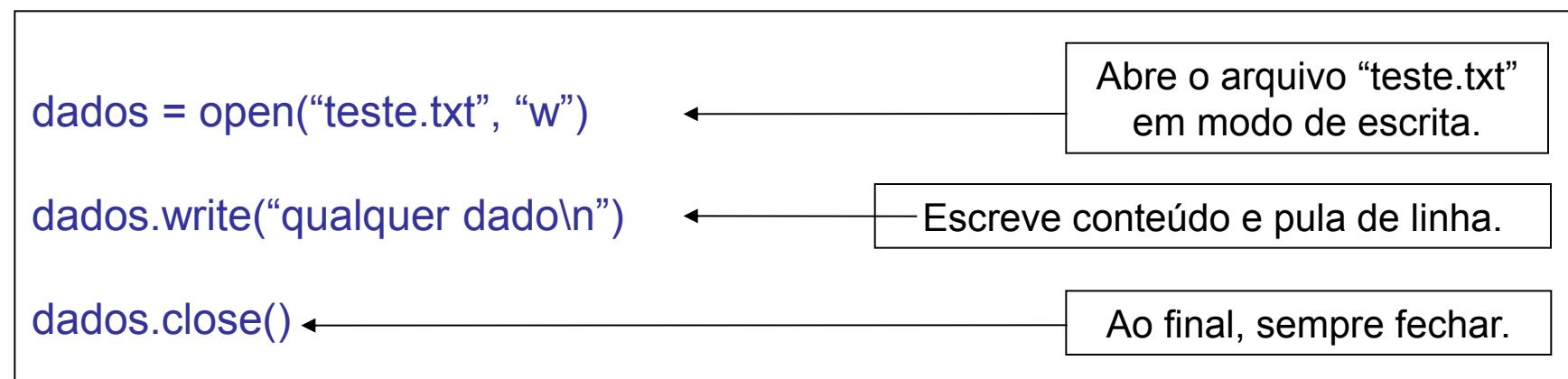
Escreve conteúdo.

```
dados.close()
```

Ao final, sempre fechar.

Produzindo Arquivo Texto: `write()` com “\n”

Para escrever uma linha de texto, necessitamos colocar o caractere que representa o fim de linha. Esse caractere é o “\n”.



Criando um Arquivo Texto

O programa abaixo pede ao usuário que escolha um nome de arquivo e quantidade de linhas que deseja escrever, em seguida os seus conteúdos são lidos do teclado e escritos no arquivo.

```
nomeArquivo = input("Digite o nome do arquivo que deseja criar: ")

quantasLinhas = int(input("Quantas linhas: "))

dados = open(nomeArquivo, "w")

for i in range(quantasLinhas):

    nova = input("Linha " + str(i+1) + ": ")

    dados.write(nova + "\n")

dados.close()
```

Criando um Arquivo Texto

O programa abaixo pede ao usuário que escolha um nome de arquivo e quantidade de linhas que deseja escrever, em seguida os seus conteúdos são lidos do teclado e escritos no arquivo.

```
nomeArquivo = input("Digite o nome do arquivo que deseja criar: ")
```

```
quantasLinhas = int(input("Quantas linhas: "))
```

```
dados = open(nomeArquivo, "w")
```

```
for i in range(quantasLinhas):
```

```
    nova = input("Linha " + str(i+1) + ": ")
```

```
    dados.write(nova + "\n")
```

```
dados.close()
```

A partir deste ponto, o arquivo **nomeArquivo** pode ser escrito.

O primeiro caracter a ser escrito será o primeiro caracter do arquivo (onde estará inicialmente a “cabeça de escrita”).

Criando um Arquivo Texto

O programa abaixo pede ao usuário que escolha um nome de arquivo e quantidade de linhas que deseja escrever, em seguida os seus conteúdos são lidos do teclado e escritos no arquivo.

```
nomeArquivo = input("Digite o nome do arquivo que deseja criar: ")  
  
quantasLinhas = int(input("Quantas linhas: "))  
  
dados = open(nomeArquivo, "w") ← A partir deste ponto, o arquivo nomeArquivo pode ser escrito.  
  
for i in range(quantasLinhas): ← O primeiro caracter a ser escrito será o primeiro caracter do arquivo (onde estará inicialmente a "cabeça de escrita").  
  
    nova = input("Linha " + str(i+1) + ": ")  
  
    dados.write(nova + "\n") ← Escreve conteúdo e pula para a próxima linha.  
  
dados.close()
```

Criando um Arquivo Texto

O programa abaixo pede ao usuário que escolha um nome de arquivo e quantidade de linhas que deseja escrever, em seguida os seus conteúdos são lidos do teclado e escritos no arquivo.

```
nomeArquivo = input("Digite o nome do arquivo que deseja criar: ")  
  
quantasLinhas = int(input("Quantas linhas: "))  
  
dados = open(nomeArquivo, "w") ← A partir deste ponto, o arquivo nomeArquivo pode ser escrito.  
  
for i in range(quantasLinhas): ← O primeiro caracter a ser escrito será o primeiro caracter do arquivo (onde estará inicialmente a "cabeça de escrita").  
  
    nova = input("Linha " + str(i+1) + ": ")  
  
    dados.write(nova + "\n") ← Escreve conteúdo e pula para a próxima linha.  
  
dados.close() ← Ao final, sempre fechar.
```

Criando um Arquivo Texto de Pontos 2D

O programa abaixo cria um arquivo, chamado “pontos.txt”, com 30 pontos bidimensionais (2D), com coordenadas aleatórias (x,y) no intervalo 0 a 400. Ao final, mostra na tela o conteúdo do arquivo gerado.

```
# Subprogramas
def criaArqPts(nome, qtd, min, max):
    from random import randint
    arq = open(nome, "w")
    for pos in range(qtd):
        arq.write(str(randint(min,max))+" "+str(randint(min, max))+"\n")
    arq.close()
    return None
def mostra(nome):
    arq = open(nome, "r")
    for pt in arq:
        print(pt, end="")
    arq.close()
    return None
# Programa Principal – Cria e Mostra Arquivo de Pontos 2D
criaArqPts("pontos.txt", 30, 0, 400)
mostra("pontos.txt")
```

Criando um Arquivo Texto de Pontos 2D

O programa abaixo cria um arquivo, chamado “pontos.txt”, com 30 pontos bidimensionais (2D), com coordenadas aleatórias (x,y) no intervalo 0 a 400. Ao final, mostra na tela o conteúdo do arquivo gerado.

Subprogramas

```
def criaArqPts(nome, qtd, min, max):
    from random import randint
    arq = open(nome, "w")
    for pos in range(qtd):
        arq.write(str(randint(min,max))+" "+str(randint(min, max))+"\n")
    arq.close()
    return None

def mostra(nome):
    arq = open(nome, "r")
    for pt in arq:
        print(pt, end="")
    arq.close()
    return None
```

Programa Principal – Cria e Mostra Arquivo de Pontos 2D

```
criaArqPts("pontos.txt", 30, 0, 400)
mostra("pontos.txt")
```

Criando um Arquivo Texto de Pontos 2D

O programa abaixo cria um arquivo, chamado “pontos.txt”, com 30 pontos bidimensionais (2D), com coordenadas aleatórias (x,y) no intervalo 0 a 400. Ao final, mostra na tela o conteúdo do arquivo gerado.

Subprogramas

```
def criaArqPts(nome, qtd, min, max):
    from random import randint
    arq = open(nome, "w")
    for pos in range(qtd):
        arq.write(str(randint(min,max))+" "+str(randint(min, max))+"\n")
    arq.close()
    return None

def mostra(nome):
    arq = open(nome, "r")
    for pt in arq:
        print(pt, end="")
    arq.close()
    return None

# Programa Principal – Cria e Mostra Arquivo de Pontos 2D
criaArqPts("pontos.txt", 30, 0, 400)
mostra("pontos.txt")
```

Criando um Arquivo Texto de Pontos 2D

O programa abaixo cria um arquivo, chamado “pontos.txt”, com 30 pontos bidimensionais (2D), com coordenadas aleatórias (x,y) no intervalo 0 a 400. Ao final, mostra na tela o conteúdo do arquivo gerado.

Subprogramas

```
def criaArqPts(nome, qtd, min, max):
    from random import randint
    arq = open(nome, "w")
    for pos in range(qtd):
        arq.write(str(randint(min,max))+" "+str(randint(min, max))+"\n")
    arq.close()
    return None

def mostra(nome):
    arq = open(nome, "r")
    for pt in arq:
        print(pt, end="")
    arq.close()
    return None
```

Programa Principal – Cria e Mostra Arquivo de Pontos 2D

```
criaArqPts("pontos.txt", 30, 0, 400)
mostra("pontos.txt")
```

Processando um Arquivo Texto de Pontos 2D

O programa abaixo faz a leitura de um arquivo, chamado “pontos.txt”, com pontos bidimensionais (2D), com coordenadas (x,y). Calcula e escreve o centroide de todos os pontos lidos.

```
# Subprogramas
def centroide(nome):
    arquivo = open(nome, "r")
    qtdPts = 0
    somaX = 0
    somaY = 0
    for coordenada in arquivo:
        partes = coordenada.split()
        somaX += float(partes[0])
        somaY += float(partes[1])
        qtdPts+=1
    arquivo.close()
    if qtdPts == 0:
        print(arquivo.name, "- vazio!!!")
    else:
        print("Ponto calculado: (", somaX/qtdPts, ", ", somaY/qtdPts, ".)")
    return None
# Programa Principal – Calcula e escreve o centroide de pontos.
centroide("pontos.txt")
```

Copiando um Arquivo Texto

Subprogramas

```
def mostra(nome):
    infos = open(nome, "r")
    for linha in infos:
        print(linha.strip())
    infos.close()
    return None

def copiar(nomeOrigem, nomeDestino):
    orig = open(nomeOrigem, "r")
    dest = open(nomeDestino, "w")
    for linha in orig:
        dest.write(linha)
    orig.close()
    dest.close()
    return None

# Programa Principal
nomes = input("Escreva os nomes dos arquivos, original e destino: ").split()
mostra(nomes[0])
copiar(nomes[0], nomes[1])
mostra(nomes[1])
```

Copiando um Arquivo Texto

Subprogramas

```
def mostra(nome):  
    infos = open(nome, "r")  
    for linha in infos:  
        print(linha.strip())  
    infos.close()  
    return None  
  
def copiar(nomeOrigem, nomeDestino):  
    orig = open(nomeOrigem, "r")  
    dest = open(nomeDestino, "w")  
    for linha in orig:  
        dest.write(linha)  
    orig.close()  
    dest.close()  
    return None
```

Programa Principal

```
nomes = input("Escreva os nomes dos arquivos, original e destino: ").split()  
mostra(nomes[0])  
copiar(nomes[0], nomes[1])  
mostra(nomes[1])
```

Copiando um Arquivo Texto

Subprogramas

```
def mostra(nome):  
    infos = open(nome, "r")  
    for linha in infos:  
        print(linha.strip())  
    infos.close()  
    return None
```

```
def copiar(nomeOrigem, nomeDestino):  
    orig = open(nomeOrigem, "r")  
    dest = open(nomeDestino, "w")  
    for linha in orig:  
        dest.write(linha)  
    orig.close()  
    dest.close()  
    return None
```

Programa Principal

```
nomes = input("Escreva os nomes dos arquivos, original e destino: ").split()  
mostra(nomes[0])  
copiar(nomes[0], nomes[1])  
mostra(nomes[1])
```

Copiando um Arquivo Texto

Subprogramas

```
def mostra(nome):  
    infos = open(nome, "r")  
    for linha in infos:  
        print(linha.strip())  
    infos.close()  
    return None  
  
def copiar(nomeOrigem, nomeDestino):  
    orig = open(nomeOrigem, "r")  
    dest = open(nomeDestino, "w")  
    for linha in orig:  
        dest.write(linha)  
    orig.close()  
    dest.close()  
    return None
```

Programa Principal

```
nomes = input("Escreva os nomes dos arquivos, original e destino: ").split()  
mostra(nomes[0])  
copiar(nomes[0], nomes[1])  
mostra(nomes[1])
```

Copiando um Arquivo Texto

Subprogramas

```
def mostra(nome):  
    infos = open(nome, "r")  
    for linha in infos:  
        print(linha.strip())  
    infos.close()  
    return None
```

```
def copiar(nomeOrigem, nomeDestino):
```

```
    orig = open(nomeOrigem, "r")  
    dest = open(nomeDestino, "w")  
    for linha in orig:  
        dest.write(linha)  
    orig.close()  
    dest.close()  
    return None
```

Programa Principal

```
nomes = input("Escreva os nomes dos arquivos, original e destino: ").split()  
mostra(nomes[0])  
copiar(nomes[0], nomes[1])  
mostra(nomes[1])
```

Copiando um Arquivo Texto

Subprogramas

```
def mostra(nome):  
    infos = open(nome, "r")  
    for linha in infos:  
        print(linha.strip())  
    infos.close()  
    return None  
  
def copiar(nomeOrigem, nomeDestino):  
    orig = open(nomeOrigem, "r")  
    dest = open(nomeDestino, "w")  
    for linha in orig:  
        dest.write(linha)  
    orig.close()  
    dest.close()  
    return None
```

Programa Principal

```
nomes = input("Escreva os nomes dos arquivos, original e destino: ").split()  
mostra(nomes[0])  
copiar(nomes[0], nomes[1])  
mostra(nomes[1])
```

Anexando uma Nova Linha ao Final de um Arquivo

```
nome = input("Diga o nome do arquivo que deseja anexar linha ao final: ")  
  
arquivo = open(nome, "a")  
  
novaLinha = input("Diga a nova linha: ")  
  
arquivo.write(novaLinha + "\n")  
  
arquivo.close()
```

Erros de Entrada e Saída para Arquivos Texto

- Nos programas vistos até agora, um erro de entrada e saída que ocorra fará seu programa terminar em estado de erro (abortará).

Erros de Entrada e Saída para Arquivos Texto

- Nos programas vistos até agora, um erro de entrada e saída que ocorra fará seu programa terminar em estado de erro (abortará).
- É possível se evitar este término abrupto do programa pelo uso de tratamento de exceções, que será visto em aulas futuras.
 - Lá veremos que os erros de entrada e saída que ocorrerem dentro de uma região do código onde as exceções são tratadas não abortarão o programa.

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:





Aula 9

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Persistência de Dados
 - Arquivo Texto

Aula 10

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estrutura de Dados
 - Conjunto (set)

Conjunto

- Python, assim como várias linguagens, inclui um tipo de dado chamado conjunto (set), que é uma estrutura de dados mutável, desordenada e sem elementos repetidos.

Conjunto

- Python, assim como várias linguagens, inclui um tipo de dado chamado conjunto (set), que é uma estrutura de dados mutável, desordenada e sem elementos repetidos.
- O uso básico deste tipo de dado se dá quando se necessita de teste de pertinência de um elemento em um conjunto ou eliminação de dados duplicados.

Conjunto

- Python, assim como várias linguagens, inclui um tipo de dado chamado conjunto (set), que é uma estrutura de dados mutável, desordenada e sem elementos repetidos.
- O uso básico deste tipo de dado se dá quando se necessita de teste de pertinência de um elemento em um conjunto ou eliminação de dados duplicados.
- Estruturas do tipo conjunto suportam as operações matemáticas:
 - A pertinência de um elemento a um conjunto,
 - A união de dois conjuntos,
 - A interseção de dois conjuntos,
 - A diferença de dois conjuntos,
 - etc.

Conjunto (set)

- Em Python, uma variável pode ser um conjunto contendo elementos de tipo(s) imutável(áveis), tais como números inteiros e de ponto flutuante, Strings e Tuplas.

Conjunto (set)

- Em Python, uma variável pode ser um conjunto contendo elementos de tipo(s) imutável(áveis), tais como números inteiros e de ponto flutuante, Strings e Tuplas.
- Diferentemente de vetores, conjuntos não têm seus elementos acessados por índice.

Conjunto (set)

- Em Python, uma variável pode ser um conjunto contendo elementos de tipo(s) imutável(áveis), tais como números inteiros e de ponto flutuante, Strings e Tuplas.
- Diferentemente de vetores, conjuntos não têm seus elementos acessados por índice.
- No entanto, conjuntos são iteráveis, podendo seus elementos serem acessados por uma estrutura **for**.

Conjunto (set)

- Em Python, uma variável pode ser um conjunto contendo elementos de tipo(s) imutável(áveis), tais como números inteiros e de ponto flutuante, Strings e Tuplas.
- Diferentemente de vetores, conjuntos não têm seus elementos acessados por índice.
- No entanto, conjuntos são iteráveis, podendo seus elementos serem acessados por uma estrutura **for**.
- Além disso, um conjunto pode ser escrito diretamente no vídeo via comando **print**.

As funções *set()*, *add()*, *discard()* e *len()*

A função **set()** associa um conjunto vazio a uma variável.

```
escolhidos = set()  
print(escolhidos)
```

As funções **set()**, **add()**, **discard()** e **len()**

A função **set()** associa um conjunto vazio a uma variável.

```
escolhidos = set()  
print(escolhidos)
```

A função **add()** adiciona um elemento ao conjunto, caso o elemento ainda não ocorra no conjunto.

```
escolhidos = set()  
escolhidos.add(13)  
print(escolhidos)
```

As funções **set()**, **add()**, **discard()** e **len()**

A função **set()** associa um conjunto vazio a uma variável.

```
escolhidos = set()  
print(escolhidos)
```

A função **add()** adiciona um elemento ao conjunto, caso o elemento ainda não ocorra no conjunto.

```
escolhidos = set()  
escolhidos.add(13)  
print(escolhidos)
```

A função **discard()** retira um elemento do conjunto, caso o elemento esteja no conjunto.

```
escolhidos = {20, 11, 68, 93}  
escolhidos.discard(68)  
print(escolhidos)
```

As funções **set()**, **add()**, **discard()** e **len()**

A função **set()** associa um conjunto vazio a uma variável.

```
escolhidos = set()  
print(escolhidos)
```

A função **add()** adiciona um elemento ao conjunto, caso o elemento ainda não ocorra no conjunto.

```
escolhidos = set()  
escolhidos.add(13)  
print(escolhidos)
```

A função **discard()** retira um elemento do conjunto, caso o elemento esteja no conjunto.

```
escolhidos = {20, 11, 68, 93}  
escolhidos.discard(68)  
print(escolhidos)
```

A função **len()** retorna a cardinalidade do conjunto, isto é, seu tamanho.

```
escolhidos = {20, 11, 68, 93}  
print(len(escolhidos))
```

Criando um Conjunto de Nomes via Teclado

O programa abaixo faz a leitura de cinco nomes e cria um conjunto com até cinco nomes distintos digitados pelo usuário. A impressão do conteúdo do conjunto ocorre a cada tentativa de inclusão de nome.

```
escolhidos = set()
for i in range(5):
    nome = input("Digite nome: ")
    escolhidos.add(nome)
    print(escolhidos)
```

Criando um Conjunto de Nomes via Teclado

O programa abaixo faz a leitura de cinco nomes e cria um conjunto com até cinco nomes distintos digitados pelo usuário. A impressão do conteúdo do conjunto ocorre a cada tentativa de inclusão de nome.

```
escolhidos = set()  
for i in range(5):  
    nome = input("Digite nome: ")  
    escolhidos.add(nome)  
    print(escolhidos)
```

Criando um Conjunto de Nomes Diretamente

```
escolhidos = {"Maria", "Ana", "Giovanna", "Leandro", "Dante"}  
print(escolhidos)
```

Operadores para Conjuntos

UNIÃO: **s.union(t)** ou **s | t**

Retorna um novo conjunto resultante da união de dois conjuntos **s** e **t** é formado por todo elemento que pertence a **s** ou que pertence a **t** (ou a ambos).

Operadores para Conjuntos

UNIÃO: **s.union(t)** ou **s | t**

Retorna um novo conjunto resultante da união de dois conjuntos **s** e **t** é formado por todo elemento que pertence a **s** ou que pertence a **t** (ou a ambos).

$$x \in (\mathbf{s.union(t)}) \Leftrightarrow x \in \mathbf{s} \text{ OU } x \in \mathbf{t}$$

Operadores para Conjuntos

UNIÃO: **s.union(t)** ou **s | t**

Retorna um novo conjunto resultante da união de dois conjuntos **s** e **t** é formado por todo elemento que pertence a **s** ou que pertence a **t** (ou a ambos).

$$x \in (\mathbf{s.union(t)}) \Leftrightarrow x \in \mathbf{s} \text{ OU } x \in \mathbf{t}$$

$$\{1, 3, 4\}.union(\{1, 2, 4\}) = \{1, 2, 3, 4\}$$

$$\{1, 3\}.union(\{2, 4\}) = \{1, 2, 3, 4\}$$

$$\{\text{'A'}, \text{'C'}, \text{'E'}\}.union(\{\text{'B'}, \text{'C'}, \text{'D'}\}) = \{\text{'A'}, \text{'B'}, \text{'C'}, \text{'D'}, \text{'E'}\}$$

$$\{\text{'C'}\}.union(\{\text{'B'}, \text{'C'}, \text{'D'}\}) = \{\text{'B'}, \text{'C'}, \text{'D'}\}$$

Operadores para Conjuntos

INTERSEÇÃO: **s.intersection(t)** ou **s & t**

Retorna um novo conjunto resultante da interseção de dois conjuntos **s** e **t** é formado por todo elemento que pertence a **s** e que pertence a **t**.

Operadores para Conjuntos

INTERSEÇÃO: **s.intersection(t)** ou **s & t**

Retorna um novo conjunto resultante da interseção de dois conjuntos **s** e **t** é formado por todo elemento que pertence a **s** e que pertence a **t**.

$$x \in s.intersection(t) \Leftrightarrow x \in s \text{ e } x \in t$$

Operadores para Conjuntos

INTERSEÇÃO: **s.intersection(t)** ou **s & t**

Retorna um novo conjunto resultante da interseção de dois conjuntos **s** e **t** é formado por todo elemento que pertence a **s** e que pertence a **t**.

$$x \in s.intersection(t) \Leftrightarrow x \in s \text{ e } x \in t$$

$$\{1, 3, 4\}.intersection(\{1, 2, 4\}) = \{1, 4\}$$

$$\{1, 3\}.intersection(\{2, 4\}) = \{ \}$$

$$\{\text{'A'}, \text{'C'}, \text{'E'} \}.intersection(\{\text{'B'}, \text{'C'}, \text{'D'}\}) = \{\text{'C'}\}$$

$$\{\text{'C'} \}.intersection(\{\text{'B'}, \text{'C'}, \text{'D'}\}) = \{\text{'C'}\}$$

Operadores para Conjuntos

DIFERENÇA: **s.difference(t)** ou **s - t**

Retorna um novo conjunto resultante da diferença entre dois conjuntos **s** e **t**. O resultado é formado por todo elemento que pertence a **s** e que não pertence a **t**.

Operadores para Conjuntos

DIFERENÇA: **s.difference(t)** ou **s - t**

Retorna um novo conjunto resultante da diferença entre dois conjuntos **s** e **t**. O resultado é formado por todo elemento que pertence a **s** e que não pertence a **t**.

$$x \in (s.difference(t)) \Leftrightarrow x \in s \text{ e } x \notin t$$

Operadores para Conjuntos

DIFERENÇA: **s.difference(t)** ou **s - t**

Retorna um novo conjunto resultante da diferença entre dois conjuntos **s** e **t**. O resultado é formado por todo elemento que pertence a **s** e que não pertence a **t**.

$$x \in (\mathbf{s}.difference(\mathbf{t})) \Leftrightarrow x \in \mathbf{s} \text{ e } x \notin \mathbf{t}$$

$$\{1, 3, 4\}.difference(\{1, 2, 4\}) = \{3\}$$

$$\{1, 3, 4\} - \{1, 2, 4\} = \{3\}$$

$$\{1, 3\}.difference(\{2, 4\}) = \{1, 3\}$$

$$\{\text{'A'}, \text{'C'}, \text{'E'}\}.difference(\{\text{'B'}, \text{'C'}, \text{'D'}\}) = \{\text{'A'}, \text{'E'}\}$$

$$\{\text{'C'}\}.difference(\{\text{'B'}, \text{'C'}, \text{'D'}\}) = \{ \}$$

Operadores para Conjuntos

Exemplo: Utilizando os operadores que sobre conjuntos, declare variáveis do tipo conjunto para representarem: um ano, as férias de fim de ano, as férias de meio de ano, todas as férias e o período letivo de um ano.

Operadores para Conjuntos

Exemplo: Utilizando os operadores que sobre conjuntos, declare variáveis do tipo conjunto para representarem: um ano, as férias de fim de ano, as férias de meio de ano, todas as férias e o período letivo de um ano.

```
ano = {"jan", "fev", "mar", "abr", "mai", "jun", " jul", "ago", "set", "out", "nov", "dez"}  
  
feriasFimAno = {"jan", "fev", "dez"}  
  
feriasMeioAno = {"jul"}  
  
ferias = feriasFimAno.union(feriasMeioAno)  
  
periodoLetivo = ano.difference(ferias)
```

Operadores Relacionais para Conjuntos

IGUAL (==) e DIFERENTE (!=):

Sejam s e t dois conjuntos:

$s == t$ é verdadeiro $\Leftrightarrow s$ e t contêm os mesmos elementos;

$s != t$ é verdadeiro, em caso contrário.

Operadores Relacionais para Conjuntos

IGUAL (`==`) e DIFERENTE (`!=`):

Sejam **s** e **t** dois conjuntos:

s == t é verdadeiro \Leftrightarrow **s** e **t** contêm os mesmos elementos;

s != t é verdadeiro, em caso contrário.

`{1, 3} == {1, 3}` é verdadeiro `{1, 3} != {1, 3}` é falso

`{1, 3} == {3, 1}` é verdadeiro `{1, 3} != {3, 1}` é falso

`{1, 3} == {1, 2}` é falso `{1, 3} != {1, 2}` é verdadeiro

`{1, 3} == {}` é falso `{1, 3} != {}` é verdadeiro

27

Operadores Relacionais para Conjuntos

CONTÉM (\geq ou `issubset`) e
ESTÁ CONTIDO (\leq ou `issuperset`):

Sejam s e t dois conjuntos:

$s \leq t$ é verdadeiro \Leftrightarrow todo elemento de s está em t .

$s \geq t$ é verdadeiro \Leftrightarrow todo elemento de t está em s .

Operadores Relacionais para Conjuntos

CONTÉM (\geq ou **issubset**) e
ESTÁ CONTIDO (\leq ou **issuperset**):

Sejam **s** e **t** dois conjuntos:

s \leq **t** é verdadeiro \Leftrightarrow todo elemento de **s** está em **t**.

s \geq **t** é verdadeiro \Leftrightarrow todo elemento de **t** está em **s**.

$\{1, 3\} \leq \{1, 2, 3, 4\}$ é verdadeiro $\{1, 3\} \geq \{1, 2, 3, 4\}$ é falso

$\{1, 3\} \leq \{1, 3\}$ é verdadeiro $\{1, 3\} \geq \{1, 3\}$ é verdadeiro

$\{\} \leq \{1, 3\}$ é verdadeiro $\{\} \geq \{1, 3\}$ é falso

$\{1, 2, 3, 4\} \leq \{1, 3\}$ é falso $\{1, 2, 3, 4\} \geq \{1, 3\}$ é verdadeiro

$\{1, 3\} \leq \{\}$ é falso $\{1, 3\} \geq \{\}$ é verdadeiro

Operadores Relacionais para Conjuntos

PERTINÊNCIA (in):

Seja **s** um conjunto.

Seja **x** um elemento imutável.

Operadores Relacionais para Conjuntos

PERTINÊNCIA (in):

Seja **s** um conjunto.

Seja **x** um elemento imutável.

x in s é verdadeiro \Leftrightarrow **x** é um elemento de **s**.

3 in {1, 2, 3, 4} é verdadeiro

5 in {1, 2, 3, 4} é falso

1 in {} é falso

Exemplo: Este programa imprime o número de vogais, e dígitos existentes em uma frase.

```
# Subprograma
def contaVogaisDigitos (frase):
    vogais = {"A", "E", "I", "O", "U", "a", "e", "i", "o", "u"}
    digitos = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
    nVogais = 0
    nDigitos = 0
    for letra in frase:
        if letra in vogais:
            nVogais += 1
        elif letra in digitos:
            nDigitos+= 1
    print("Quantidade de Vogais:", nVogais)
    print("Quantidade de Dígitos:", nDigitos)
    return None

# Programa Principal
lida = input("Diga a frase: ")
contaVogaisDigitos(lida)
```

Exemplo: Este programa imprime o número de vogais, e dígitos existentes em uma frase.

```
# Subprograma
def contaVogaisDigitos (frase):
    vogais = {"A", "E", "I", "O", "U", "a", "e", "i", "o", "u"}
    digitos = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
    nVogais = 0
    nDigitos = 0
    for letra in frase:
        if letra in vogais:
            nVogais += 1
        elif letra in digitos:
            nDigitos+= 1
    print("Quantidade de Vogais:", nVogais)
    print("Quantidade de Dígitos:", nDigitos)
    return None
```

```
# Programa Principal
lida = input("Diga a frase: ")
contaVogaisDigitos(lida)
```

Exemplo: Este programa imprime o número de vogais, e dígitos existentes em uma frase.

Subprograma

```
def contaVogaisDigitos (frase):
    vogais = {"A", "E", "I", "O", "U", "a", "e", "i", "o", "u"}
    digitos = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
    nVogais = 0
    nDigitos = 0
    for letra in frase:
        if letra in vogais:
            nVogais += 1
        elif letra in digitos:
            nDigitos+= 1
    print("Quantidade de Vogais:", nVogais)
    print("Quantidade de Dígitos:", nDigitos)
    return None

# Programa Principal
lida = input("Diga a frase: ")
contaVogaisDigitos(lida)
```

Exemplo: Este programa imprime o número de vogais, e dígitos existentes em uma frase.

```
# Subprograma
def contaVogaisDigitos (frase):
    vogais = {"A", "E", "I", "O", "U", "a", "e", "i", "o", "u"}
    digitos = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
    nVogais = 0
    nDigitos = 0
    for letra in frase:
        if letra in vogais:
            nVogais += 1
        elif letra in digitos:
            nDigitos+= 1
    print("Quantidade de Vogais:", nVogais)
    print("Quantidade de Dígitos:", nDigitos)
    return None

# Programa Principal
lida = input("Diga a frase: ")
contaVogaisDigitos(lida)
```

Exemplo: Este programa imprime o número de vogais, e dígitos existentes em uma frase.

```
# Subprograma
def contaVogaisDigitos (frase):
    vogais = {"A", "E", "I", "O", "U", "a", "e", "i", "o", "u"}
    digitos = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}
    nVogais = 0
    nDigitos = 0
    for letra in frase:
        if letra in vogais:
            nVogais += 1
        elif letra in digitos:
            nDigitos+= 1
    print("Quantidade de Vogais:", nVogais)
    print("Quantidade de Dígitos:", nDigitos)
    return None

# Programa Principal
lida = input("Diga a frase: ")
contaVogaisDigitos(lida)
```

Exemplo: Este programa gera e imprime os números primos entre 2 e N, escolhido pelo usuário, usando o algoritmo chamado de “**Crivo de Eratóstenes**”.

```
# Subprogramas
def imprime(num, osPrimos):
    print("Primos entre 2 e", num, ":")
    for candidato in range(2, num+1):
        if candidato in osPrimos:
            print(candidato, end=" ")
    print()
    return None
def eratostenes(num):
    resposta = set()
    ...
    return resposta
# Programa Principal - Crivo de Eratóstenes
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Exemplo: Este programa gera e imprime os números primos entre 2 e N, escolhido pelo usuário, usando o algoritmo chamado de “**Crivo de Eratóstenes**”.

```
# Subprogramas
def imprime(num, osPrimos):
    print("Primos entre 2 e", num, ":")
    for candidato in range(2, num+1):
        if candidato in osPrimos:
            print(candidato, end=" ")
    print()
    return None
def eratostenes(num):
    resposta = set()
    ...
    return resposta
# Programa Principal - Crivo de Eratóstenes
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Exemplo: Este programa gera e imprime os números primos entre 2 e N, escolhido pelo usuário, usando o algoritmo chamado de “**Crivo de Eratóstenes**”.

```
# Subprogramas
def imprime(num, osPrimos):
    print("Primos entre 2 e", num, ":")
    for candidato in range(2, num+1):
        if candidato in osPrimos:
            print(candidato, end=" ")
    print()
    return None

def eratostenes(num):
    resposta = set()

    ...
    return resposta

# Programa Principal - Crivo de Eratóstenes
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Exemplo: Este programa gera e imprime os números primos entre 2 e N, escolhido pelo usuário, usando o algoritmo chamado de “**Crivo de Eratostenes**”.

```
# Subprogramas
def imprime(num, osPrimos):
    print("Primos entre 2 e", num, ":")
    for candidato in range(2, num+1):
        if candidato in osPrimos:
            print(candidato, end=" ")
    print()
    return None
def eratostenes(num):
    resposta = set()
    ...
    return resposta
# Programa Principal - Crivo de Eratostenes
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Subprogramas

```
def imprime(num, osPrimos):
    def eratostenes(num):
        resposta = set()          # inicializa resposta
        vazio = set()              # inicializa conjunto vazio
        crivo = set(range(2, num+1)) # constrói conjunto de 2 a num
        prox = 2
        while crivo != vazio:
            while not (prox in crivo):
                prox += 1
            resposta.add(prox) # ou resposta = resposta | {prox}
            j = prox
            while j <= num:
                crivo.discard(j) # ou crivo = crivo - {j}
                j += prox
        return resposta
```

Programa Principal - Crivo de Erastóstenes

```
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Subprogramas

```
def imprime(num, osPrimos):
    def eratostenes(num):
        resposta = set()          # inicializa resposta
        vazio = set()              # inicializa conjunto vazio
        crivo = set(range(2, num+1)) # constrói conjunto de 2 a num
        prox = 2
        while crivo != vazio:
            while not (prox in crivo):
                prox += 1
            resposta.add(prox) # ou resposta = resposta | {prox}
            j = prox
            while j <= num:
                crivo.discard(j) # ou crivo = crivo - {j}
                j += prox
        return resposta
# Programa Principal - Crivo de Erastóstenes
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Subprogramas

```
def imprime(num, osPrimos):
    def eratostenes(num):
        resposta = set()          # inicializa resposta
        vazio = set()              # inicializa conjunto vazio
        crivo = set(range(2, num+1)) # constrói conjunto de 2 a num
        prox = 2
        while crivo != vazio:
            while not (prox in crivo):
                prox += 1
            resposta.add(prox) # ou resposta = resposta | {prox}
            j = prox
            while j <= num:
                crivo.discard(j) # ou crivo = crivo - {j}
                j += prox
        return resposta
```

Programa Principal - Crivo de Erastóstenes

```
n = int(input("Diga o valor: "))
primos = eratostenes(n)
imprime(n, primos)
```

Este subprograma imprime o conjunto de características comuns a todos os indivíduos pertencentes a um subconjunto de uma determinada população. Cada indivíduo da população está associado a um identificador (0..MaxPop-1) e possui um conjunto de características.

Este subprograma imprime o conjunto de características comuns a todos os indivíduos pertencentes a um subconjunto de uma determinada população. Cada indivíduo da população está associado a um identificador (0..MaxPop-1) e possui um conjunto de características.

```
características = {"esporte", "tv", "cinema", "livro", "jornal", "teatro", "musica"}
```

```
def perfilComum(habitantes, características, grupo):
    comuns = características
    for ident in range(len(habitantes)):
        if ident in grupo:
            comuns = comuns & habitantes[ident]      # ou intersection
    print("As características em comum são:")
    for c in características:
        if c in comuns:
            print(c, end=" ")
    print()
return None
```

Subprograma

```
def perfilComum(habitantes, caracteristicas, grupo):
    comuns = caracteristicas
    for ident in range(len(habitantes)):
        if ident in grupo:
            comuns = comuns & habitantes[ident]
    print("As características em comum são:")
    for c in caracteristicas:
        if c in comuns:
            print(c, end=" ")
    print()
    return None
```

Programa Principal

```
caracteristicas = {"esporte", "tv", "cinema", "livro", "jornal", "teatro", "musica"}
alunos = [{"tv", "cinema", "livro"}, {"cinema", "musica"}, {"cinema", "tv", "teatro"}]
perfilComum(alunos, caracteristicas, {2, 0})
```

Subprograma

```
def perfilComum(habitantes, caracteristicas, grupo):
    comuns = caracteristicas
    for ident in range(len(habitantes)):
        if ident in grupo:
            comuns = comuns & habitantes[ident]
    print("As características em comum são:")
    for c in caracteristicas:
        if c in comuns:
            print(c, end=" ")
    print()
    return None
```

Programa Principal

```
caracteristicas = {"esporte", "tv", "cinema", "livro", "jornal", "teatro", "musica"}
alunos = [{"tv", "cinema", "livro"}, {"cinema", "musica"}, {"cinema", "tv", "teatro"}]
perfilComum(alunos, caracteristicas, {2, 0})
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:



Aula 10

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estrutura de Dados
 - Conjunto (set)



Aula 11

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estrutura de Dados
 - Dicionário (dict)

Dicionário (*dict*)

- O tipo de dado dicionário, é uma estrutura de dados eficiente e mutável, acessada por uma chave não repetida, que pode ser de qualquer tipo imutável, tal como número, String ou Tupla.

Dicionário (*dict*)

- O tipo de dado dicionário, é uma estrutura de dados eficiente e mutável, acessada por uma chave não repetida, que pode ser de qualquer tipo imutável, tal como número, String ou Tupla.
- O uso básico deste tipo de dado se dá quando se necessita de uma estrutura eficiente para armazenamento de pares **chave:valor**.

Dicionário (*dict*)

- O tipo de dado dicionário, é uma estrutura de dados eficiente e mutável, acessada por uma chave não repetida, que pode ser de qualquer tipo imutável, tal como número, String ou Tupla.
- O uso básico deste tipo de dado se dá quando se necessita de uma estrutura eficiente para armazenamento de pares **chave:valor**.
- Para se recuperar um **valor**, basta informar sua **chave**.

Dicionário (*dict*)

- Uma variável pode ser um dicionário contendo chave de tipo imutável, tal como números inteiros e de ponto flutuante, Strings e Tuplas, e valor de qualquer tipo.

Dicionário (*dict*)

- Uma variável pode ser um dicionário contendo chave de tipo imutável, tal como números inteiros e de ponto flutuante, Strings e Tuplas, e valor de qualquer tipo.
- Dicionários tem pares, compostos de **chave:valor**, iteráveis sobre a **chave**, podendo seus elementos serem acessados por uma estrutura **for**.

Dicionário (*dict*)

- Uma variável pode ser um dicionário contendo chave de tipo imutável, tal como números inteiros e de ponto flutuante, Strings e Tuplas, e valor de qualquer tipo.
- Dicionários tem pares, compostos de **chave:valor**, iteráveis sobre a **chave**, podendo seus elementos serem acessados por uma estrutura **for**.
- Além disso, um dicionário pode ser escrito diretamente no vídeo via comando **print**.

As funções *dict()*, *del()* e *len()*

A função **dict()** associa um dicionário vazio a uma variável.

```
pares = dict()          # ou pares = {}  
print(pares)
```

As funções **dict()**, **del()** e **len()**

A função **dict()** associa um dicionário vazio a uma variável.

```
pares = dict() # ou pares = {}  
print(pares)
```

Para adicionar um novo par ao dicionário, por exemplo um par número:string, basta atribuir o valor ao nome do dicionário seguido pela chave entre colchetes. Caso já exista esta chave, o valor é atualizado.

```
pares = dict()  
pares[13] = "Valor da Sorte"  
print(pares)
```

As funções `dict()`, `del()` e `len()`

A função `dict()` associa um dicionário vazio a uma variável.

```
pares = dict() # ou pares = {}  
print(pares)
```

Para adicionar um novo par ao dicionário, por exemplo um par número:string, basta atribuir o valor ao nome do dicionário seguido pela chave entre colchetes. Caso já exista esta chave, o valor é atualizado.

```
pares = dict()  
pares[13] = "Valor da Sorte"  
print(pares)
```

A função `del nomeDict[chave]` retira o par **chave:valor** do dicionário. Caso a **chave** não esteja no dicionário um erro ocorre - `KeyError`.

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
del pares[13] # ou pares.pop(13)  
print(pares)
```

As funções `dict()`, `del()` e `len()`

A função `dict()` associa um dicionário vazio a uma variável.

```
pares = dict() # ou pares = {}  
print(pares)
```

Para adicionar um novo par ao dicionário, por exemplo um par número:string, basta atribuir o valor ao nome do dicionário seguido pela chave entre colchetes. Caso já exista esta chave, o valor é atualizado.

```
pares = dict()  
pares[13] = "Valor da Sorte"  
print(pares)
```

A função `del nomeDict[chave]` retira o par **chave:valor** do dicionário. Caso a **chave** não esteja no dicionário um erro ocorre - `KeyError`.

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
del pares[13] # ou pares.pop(13)  
print(pares)
```

A função `len()` retorna o tamanho do dicionário.

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
print(len(pares))
```

Criando um Dicionário de nomes:fones

O programa abaixo faz a leitura de cinco nomes e telefones e cria um dicionário com até cinco nomes distintos digitados pelo usuário. Ocorre uma escrita do conteúdo do dicionário a cada tentativa de inclusão de **nome:fone**.

```
pares = dict() # ou pares = {}  
for i in range(5):  
    nome = input("Digite nome: ")  
    fone = input("Digite o telefone de "+nome+": ")  
    pares[nome] = fone  
print(pares)
```

Criando um Dicionário de nomes:fones

O programa abaixo faz a leitura de cinco nomes e telefones e cria um dicionário com até cinco nomes distintos digitados pelo usuário. Ocorre uma escrita do conteúdo do dicionário a cada tentativa de inclusão de **nome:fone**.

```
pares = dict() # ou pares = {}  
for i in range(5):  
    nome = input("Digite nome: ")  
    fone = input("Digite o telefone de "+nome+": ")  
    pares[nome] = fone  
print(pares)
```

Criando um Dicionário Diretamente

```
pares = {"Maria": "3456-7922", "Ana": "3214-2211", "Giovanna": "4564-1234"}  
print(pares)
```

Visualizações sobre Dicionário

d.items(): retorna uma visualização de todos os pares (chave, valor) de um dicionário **d**.

d.keys(): retorna uma visualização de todas as chaves de um dicionário **d**.

d.values(): retorna uma visualização de todos os valores de um dicionário **d**.

Iterando sobre Chaves ou Valores

```
for chave in pares:  
    print(chave, ":", pares[chave])
```

Iterando sobre Chaves ou Valores

```
for chave in pares:  
    print(chave, ":", pares[chave])
```

```
for chave, valor in pares.items():  
    print(chave, ":", valor)
```

Iterando sobre Chaves ou Valores

```
for chave in pares:  
    print(chave, “:”, pares[chave])
```

```
for chave, valor in pares.items():  
    print(chave, “:”, valor)
```

```
for chave in sorted(pares):  
    print(chave, “:”, pares[chave])
```

Iterando sobre Chaves ou Valores

```
for chave in pares:  
    print(chave, “:”, pares[chave])
```

```
for chave, valor in pares.items():  
    print(chave, “:”, valor)
```

```
for chave in sorted(pares):  
    print(chave, “:”, pares[chave])
```

```
for chave in sorted(pares.keys()):  
    print(chave, “:”, pares[chave])
```

Iterando sobre Chaves ou Valores

```
for chave in pares:  
    print(chave, “:”, pares[chave])
```

```
for chave, valor in pares.items():  
    print(chave, “:”, valor)
```

```
for chave in sorted(pares):  
    print(chave, “:”, pares[chave])
```

```
for chave in sorted(pares.keys()):  
    print(chave, “:”, pares[chave])
```

```
for valor in pares.values():  
    print(valor)
```

Exemplo de Aplicação de Dicionário

Supermercado de Produtos, onde cada Produto possui:

- (a) Código não repetido (a chave) – representado por String;
- (b) Descrição – representada por String;
- (c) Quantidade – representada por inteiro;
- (d) Preço – representado por número de ponto flutuante; e
- (e) Data de Validade – representada por Tupla
(dia, mês, ano).

Exemplo de Aplicação de Dicionário

Supermercado de Produtos, onde cada Produto possui:

- (a) Código não repetido (a chave) – representado por String;
- (b) Descrição – representada por String;
- (c) Quantidade – representada por inteiro;
- (d) Preço – representado por número de ponto flutuante; e
- (e) Data de Validade – representada por Tupla
(dia, mês, ano).

```
produtos = {}  
preencher(produtos, 5)  
mostrar(produtos)  
vender(produtos, {"xkk":3, "yzb":2})  
mostrar(produtos)
```

```

# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, “ – ”, valor)
    return None
def preencher(prods, entradas):
    for i in range(entradas):
        cod = input(“Código: ”)
        desc = input(“Descrição: ”)
        qtd = int(input(“Quantidade: ”))
        preco = float(input(“Valor: ”))
        data = input(“Limite de Validade - dd/mm/aa: ”)
        partes = data.split(“/”)
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None
def vender(prods, codsQtds):
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)

```

```

# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " - ", valor)
    return None

def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None

def vender(prods, codsQtds):
    return None

# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)

```

Subprogramas

```
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " - ", valor)
    return None

def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None

def vender(prods, codsQtds):
    return None

# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

```

# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, “ – ”, valor)
    return None

def preencher(prods, entradas):
    for i in range(entradas):
        cod = input(“Código: ”)
        desc = input(“Descrição: ”)
        qtd = int(input(“Quantidade: ”))
        preco = float(input(“Valor: ”))
        data = input(“Limite de Validade - dd/mm/aa: ”)
        partes = data.split(“/”)
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None

def vender(prods, codsQtds):
    return None

# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)

```

```

# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " - ", valor)
    return None

def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None

def vender(prods, codsQtds):
    return None

# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)

```

Subprogramas

...

```
def vender(prods, codsQtds):
    totalGasto = 0
    for chave in codsQtds:
        if chave not in prods:
            print(chave, "- Código Inexistente")
        else:
            item = prods[chave]
            if item[1]<codsQtds[chave]:
                print(chave, "- Quantidade Insuficiente")
            else:
                item[1]-=codsQtds[chave]
                prods[chave]=item
                print(item[0], "x", codsQtds[chave], "- Subtotal:", item[2]*codsQtds[chave])
                totalGasto += item[2]*codsQtds[chave]
    print("Total da Nota Fiscal:", totalGasto)
    return None
```

Programa Principal

```
produtos = {}
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

Consultando Dicionários

Se consultarmos diretamente uma chave inexistente em um dicionário, obteremos uma exceção e uma respectiva mensagem de erro: **KeyError**. Veja o exemplo abaixo:

Consultando Dicionários

Se consultarmos diretamente uma chave inexistente em um dicionário, obteremos uma exceção e uma respectiva mensagem de erro: **KeyError**. Veja o exemplo abaixo:

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
print(pares[51])          # escreve Boa Ideia  
print(pares[52])          # aborta a execução com KeyError: 52
```

Consultando Dicionários

Se consultarmos diretamente uma chave inexistente em um dicionário, obteremos uma exceção e uma respectiva mensagem de erro: **KeyError**. Veja o exemplo abaixo:

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
print(pares[51])          # escreve Boa Ideia  
print(pares[52])          # aborta a execução com KeyError: 52
```

Para se evitar este tipo de erro durante a consulta podemos utilizar a função **get**, descrita a seguir.

A Função `get()`

A função **get(chave)** retorna **None** se não existir aquela **chave** no dicionário, caso contrário retorna o respectivo **valor**.

```
pares = {}  
print(pares.get(44)) # escreve None
```

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
print(pares.get(13)) # escreve Valor da Sorte
```

A Função `get()`

A função **get(chave)** retorna **None** se não existir aquela **chave** no dicionário, caso contrário retorna o respectivo **valor**.

```
pares = {}  
print(pares.get(44)) # escreve None
```

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
print(pares.get(13)) # escreve Valor da Sorte
```

A função **get(chave,default)** retorna **default** se não existir aquela **chave** no dicionário, caso contrário retorna o respectivo **valor**.

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}  
print(pares.get(22, "Vazia")) # escreve Vazia  
print(pares.get(51, "Vazia")) # escreve Boa Ideia
```

Uso de Dicionário para Matriz Esparsa

Subprograma

```
def mostra(vs):                      # exibe a matriz organizadamente
    qtdLinhas = vs["Número de Linhas"]
    qtdColunas = vs["Número de Colunas"]
    for linha in range(qtdLinhas):
        for coluna in range(qtdColunas):
            valor = vs.get((linha,coluna),0)
            print("%4d"%valor, end=' ')
        print()                           # saída formatada - 4 espaços para cada valor
                                         # pula para a próxima linha
    return
```

Programa Principal

```
# valores mantém apenas as dimensões e os poucos números diferentes de zero
valores = {}
valores["Número de Linhas"] = 5      # chave string mantém a quantidade de linhas
valores["Número de Colunas"] = 10    # chave string mantém a quantidade de colunas
valores[(1,1)] = 13                  # chave tupla (1,1) com um valor não zero
valores[(0,8)] = -4                 # chave tupla (0,8) com um valor não zero
valores[(4,7)] = 75                  # chave tupla (4,7) com um valor não zero
print(valores)
mostra(valores)
```

Uso de Dicionário para Matriz Esparsa

Subprograma

```
def mostra(vs):                      # exibe a matriz organizadamente
    qtdLinhas = vs["Número de Linhas"]
    qtdColunas = vs["Número de Colunas"]
    for linha in range(qtdLinhas):
        for coluna in range(qtdColunas):
            valor = vs.get((linha,coluna),0)
            print("%4d"%valor, end=' ')    # saída formatada - 4 espaços para cada valor
            print()                      # pula para a próxima linha
    return
```

Programa Principal

```
# valores mantém apenas as dimensões e os poucos números diferentes de zero
valores = {}
valores["Número de Linhas"] = 5      # chave string mantém a quantidade de linhas
valores["Número de Colunas"] = 10     # chave string mantém a quantidade de colunas
valores[(1,1)] = 13                  # chave tupla (1,1) com um valor não zero
valores[(0,8)] = -4                  # chave tupla (0,8) com um valor não zero
valores[(4,7)] = 75                  # chave tupla (4,7) com um valor não zero
print(valores)
mostra(valores)
```

Uso de Dicionário para Matriz Esparsa

Subprograma

```
def mostra(vs):                      # exibe a matriz organizadamente
    qtdLinhas = vs["Número de Linhas"]
    qtdColunas = vs["Número de Colunas"]
    for linha in range(qtdLinhas):
        for coluna in range(qtdColunas):
            valor = vs.get((linha,coluna),0)
            print("%4d"%valor, end=' ')    # saída formatada - 4 espaços para cada valor
            print()                      # pula para a próxima linha
    return
```

Programa Principal

```
# valores mantém apenas as dimensões e os poucos números diferentes de zero
valores = {}
valores["Número de Linhas"] = 5          # chave string mantém a quantidade de linhas
valores["Número de Colunas"] = 10         # chave string mantém a quantidade de colunas
valores[(1,1)] = 13                      # chave tupla (1,1) com um valor não zero
valores[(0,8)] = -4                      # chave tupla (0,8) com um valor não zero
valores[(4,7)] = 75                      # chave tupla (4,7) com um valor não zero
print(valores)
mostra(valores)
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:





Aula 11

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estrutura de Dados
 - Dicionário (dict)



Aula 12

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Persistência de Dados
 - Arquivos Binários

Arquivos (conforme visto na Aula 9)

- Programas **interativos** leem os dados de entrada do teclado e apresentam os dados de saída na tela.
 - Este tipo de programação é utilizada quando poucos dados são processados ou quando necessitam de interação humana.

Arquivos (conforme visto na Aula 9)

- Programas **interativos** leem os dados de entrada do teclado e apresentam os dados de saída na tela.
 - Este tipo de programação é utilizada quando poucos dados são processados ou quando necessitam de interação humana.
- Quando grandes quantidades de dados são processadas, **arquivos** são utilizados para armazenar os dados de entrada e os de saída.
 - Estes são chamados, respectivamente, arquivos de entrada e arquivos de saída.

Arquivo Binário

- Há basicamente dois tipos de arquivo: texto e binário.

Arquivo Binário

- Há basicamente dois tipos de arquivo: texto e binário.
- Um **arquivo binário** é uma sequência de bytes que reside em uma área de armazenamento (e.g., disco rígido, pen drive, CD/DVD) sob um mesmo nome.
 - Essencialmente, arquivos texto também armazenam informação como uma sequência de bytes. Porém, esses são diretamente associados a caracteres.

Arquivo Binário

- Há basicamente dois tipos de arquivo: texto e binário.
- Um **arquivo binário** é uma sequência de bytes que reside em uma área de armazenamento (e.g., disco rígido, pen drive, CD/DVD) sob um mesmo nome.
 - Essencialmente, arquivos texto também armazenam informação como uma sequência de bytes. Porém, esses são diretamente associados a caracteres.
- Arquivos binários podem ser criados, editados e seu conteúdo recuperado por programas que conhecem a estrutura dos dados nele armazenados.

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função **arq = open(...)**

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função **arq = open(...)**

Fechamento: Fecha o arquivo associado a uma variável, liberando o recurso. Operação realizada através do método **arq.close()**

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função **arq = open(...)**

Fechamento: Fecha o arquivo associado a uma variável, liberando o recurso. Operação realizada através do método **arq.close()**

Leitura: Copia parte ou todo o conteúdo de um arquivo para a memória principal. Operação realizada através do método **arq.read(...)**

Escrita: Copia dados presentes na memória principal para dentro do arquivo. Operação realizada através do método **arq.write(...)**

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função **arq = open(...)**

Fechamento: Fecha o arquivo associado a uma variável, liberando o recurso. Operação realizada através do método **arq.close()**

Leitura: Copia parte ou todo o conteúdo de um arquivo para a memória principal. Operação realizada através do método **arq.read(...)**

Escrita: Copia dados presentes na memória principal para dentro do arquivo. Operação realizada através do método **arq.write(...)**

Navegação: Reposiciona e consulta a localização do cursor de escrita/leitura. Operações realizadas, respectivamente, através dos métodos **arq.seek(...)** e **arq.tell()**

Abertura e Fechamento de Arquivo Binário

A operação de abertura associa um arquivo a uma variável (e.g., **arq**), permitindo a manipulação de seu conteúdo. Ela é realizada pela função

arq = open(*nome*, *modo*)

- O parâmetro **nome** é um String que indica o local onde o arquivo reside.
- O parâmetro **modo** é um String que define como o arquivo deverá ser aberto. Ele pode assumir os valores: “rb”, “wb”, “r+b”, “w+b” e “a+b”

Abertura e Fechamento de Arquivo Binário

A operação de abertura associa um arquivo a uma variável (e.g., **arq**), permitindo a manipulação de seu conteúdo. Ela é realizada pela função

arq = open(*nome, modo*)

- O parâmetro **nome** é um String que indica o local onde o arquivo reside.
- O parâmetro **modo** é um String que define como o arquivo deverá ser aberto. Ele pode assumir os valores: “rb”, “wb”, “r+b”, “w+b” e “a+b”

O método **arq.close()** fecha o arquivo e libera o recurso. Todo arquivo aberto DEVE ser fechado após o uso!

```
# Abertura de arquivo para leitura, seguido pelo fechamento
try:
    arq = open("arquivo.bin", "rb")
    print("O arquivo está aberto.")
    arq.close()
except IOError:
    print("Erro ao abrir o arquivo.")
```

Modos de Abertura de Arquivo Binário

arq = **open**(nome, modo)

O parâmetro **modo** pode assumir os seguintes valores:

“rb”: Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

Modos de Abertura de Arquivo Binário

arq = **open**(nome, modo)

O parâmetro **modo** pode assumir os seguintes valores:

“rb”: Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

“wb”: Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

Modos de Abertura de Arquivo Binário

arq = **open**(nome, modo)

O parâmetro **modo** pode assumir os seguintes valores:

“rb”: Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

“wb”: Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

“r+b”: Abre um arquivo binário especificado para leitura e escrita. O arquivo deve existir previamente, podendo seu conteúdo ser alterado.

Modos de Abertura de Arquivo Binário

arq = **open**(nome, modo)

O parâmetro **modo** pode assumir os seguintes valores:

“rb”: Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

“wb”: Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

“r+b”: Abre um arquivo binário especificado para leitura e escrita. O arquivo deve existir previamente, podendo seu conteúdo ser alterado.

“w+b”: Abre um arquivo binário especificado para escrita e leitura. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

Modos de Abertura de Arquivo Binário

arq = **open**(nome, modo)

O parâmetro **modo** pode assumir os seguintes valores:

“rb”: Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

“wb”: Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

“r+b”: Abre um arquivo binário especificado para leitura e escrita. O arquivo deve existir previamente, podendo seu conteúdo ser alterado.

“w+b”: Abre um arquivo binário especificado para escrita e leitura. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

“a+b”: Abre um arquivo binário no modo *append* (concatenação). Caso o arquivo não exista, este será criado. Caso existam seus dados serão mantidos e o novo conteúdo será anexado ao fim do arquivo.

Abertura e Fechamento de Arquivo Binário

Utilize a diretiva **with-as** para simplificar o script e para garantir que o arquivo será fechado, mesmo em caso de exceção na manipulação

```
try:  
    arq = open("arquivo.bin", "rb")  
    try:  
        # Faça alguma manipulação no arquivo  
    finally:  
        arq.close()  
except IOError:  
    print("Erro ao abrir ou ao manipular o arquivo.")
```

... simplifica para ...

```
try:  
    with open("arquivo.bin", "rb") as arq:  
        # Faça alguma manipulação no arquivo  
    except IOError:  
        print("Erro ao abrir ou ao manipular o arquivo.")
```

Leitura de Arquivo Binário

A operação de leitura é sequencial. Ela copia um bloco de bytes contido no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A leitura é realizada pelo método

bloco = arq.read(tamanho)

- O parâmetro **tamanho** é opcional. Quando não informado, resulta na leitura de todos os bytes, da posição atual até o fim do arquivo. Quando informado, resulta na leitura da quantidade de bytes indicada.

Leitura de Arquivo Binário

A operação de leitura é sequencial. Ela copia um bloco de bytes contido no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A leitura é realizada pelo método

bloco = arq.read(tamanho)

- O parâmetro **tamanho** é opcional. Quando não informado, resulta na leitura de todos os bytes, da posição atual até o fim do arquivo. Quando informado, resulta na leitura da quantidade de bytes indicada.

Utilize o método **unpack(formato, bloco)**, do módulo **struct**, para desempacotar o bloco de bytes na forma de tipos primitivos

Leitura de Arquivo Binário

A operação de leitura é sequencial. Ela copia um bloco de bytes contido no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A leitura é realizada pelo método

bloco = arq.read(tamanho)

- O parâmetro **tamanho** é opcional. Quando não informado, resulta na leitura de todos os bytes, da posição atual até o fim do arquivo. Quando informado, resulta na leitura da quantidade de bytes indicada.

Utilize o método **unpack(formato, bloco)**, do módulo **struct**, para desempacotar o bloco de bytes na forma de tipos primitivos

No caso de String, utilize o método **decode(codificação)** para converter o bloco de bytes em caracteres

Exemplos de Leitura de Arquivo Binário

```
# Leitura de todos os bytes contidos em um arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        byte = arq.read(1)
        while byte != b"":
            # Faça alguma coisa com o byte lido
            byte = arq.read(1)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

```
import struct
# Leitura de um valor inteiro e um ponto flutuante com precisão dupla
try:
    with open("arquivo.bin", "rb") as arq:
        inteiro = struct.unpack("i", arq.read(4))[0]      # Note o acesso "[0]"
        real = struct.unpack("d", arq.read(8))[0]
        print("Valor inteiro:", inteiro, "Valor real:", real)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplos de Leitura de Arquivo Binário

```
# Leitura de um string binarizado que ocupa 16 bytes no arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        bloco = arq.read(16)
        texto = bloco.decode("utf-8") # Conversão do bloco em String
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

O parâmetro esperado pelo método **decode(codificação)** deve indicar qual o mapa de caracteres utilizado na conversão do bloco de bytes em String. Caso não seja informado, a codificação utilizada será “utf-8”.

Dentre as dezenas de codificações disponíveis, junto à “utf-8”, a codificação “ascii” é uma das mais comuns.

Escrita em Arquivo Binário

A operação de escrita é sequencial. Ela copia um bloco de bytes contido na memória principal (acessada por uma variável), para dentro de um bloco de mesmo tamanho no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A escrita é realizada pelo método

arq.write(*bloco*)

Escrita em Arquivo Binário

A operação de escrita é sequencial. Ela copia um bloco de bytes contido na memória principal (acessada por uma variável), para dentro de um bloco de mesmo tamanho no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A escrita é realizada pelo método

`arq.write(bloco)`

Utilize o método **pack(*formato*, *v1*, *v2*, ...)**, do módulo **struct**, para empacotar dentro de um bloco, conforme o formato especificado, os valores *v1*, *v2*, ...

Escrita em Arquivo Binário

A operação de escrita é sequencial. Ela copia um bloco de bytes contido na memória principal (acessada por uma variável), para dentro de um bloco de mesmo tamanho no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A escrita é realizada pelo método

`arq.write(bloco)`

Utilize o método **pack(formato, v1, v2, ...)**, do módulo **struct**, para empacotar dentro de um bloco, conforme o formato especificado, os valores *v1*, *v2*, ...

No caso de String, utilize o método **encode(codificação)** para converter os caracteres em um bloco de bytes

Exemplos de Escrita em Arquivo Binário

Cópia do conteúdo de um arquivo binário para outro

```
with open("origem.bin", "rb") as entrada:
```

```
    with open("destino.bin", "wb") as saída:
```

```
        byte = entrada.read(1)
```

```
        while byte != b"":
```

```
            saída.write(byte)
```

```
            byte = entrada.read(1)
```

import struct

Escrita de um valor inteiro e um ponto flutuante com precisão dupla

```
try:
```

```
    with open("arquivo.bin", "wb") as arquivo:
```

```
        bloco = struct.pack("i", 10)
```

```
        arquivo.write(bloco)
```

```
        bloco = struct.pack("d", 99.5)
```

```
        arquivo.write(bloco)
```

```
except IOError:
```

```
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Escrita em Arquivo Binário

Escrita de um string binarizado no arquivo

try:

```
    with open("arquivo.bin", "wb") as arq:  
        texto = "Sou aluno do CEDERJ"  
        bloco = texto.encode("utf-8") # Conversão do String em bloco de bytes  
        arq.write(bloco)
```

except IOError:

```
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Escrita em Arquivo Binário

Escrita de um string binarizado no arquivo

try:

```
    with open("arquivo.bin", "wb") as arq:  
        texto = "Sou aluno do CEDERJ"  
        bloco = texto.encode("utf-8") # Conversão do String em bloco de bytes  
        arq.write(bloco)
```

except IOError:

```
    print("Erro ao abrir ou ao manipular o arquivo.")
```

O parâmetro esperado pelo método **encode(codificação)** deve indicar qual o mapa de caracteres utilizado na conversão de String para bloco de bytes. Caso não seja informado, a codificação utilizada será “utf-8”.

Exemplo de Escrita em Arquivo Binário

```
# Escrita de um string binarizado no arquivo
```

```
try:
```

```
    with open("arquivo.bin", "wb") as arq:
```

```
        texto = "Sou aluno do CEDERJ"
```

```
        bloco = texto.encode("utf-8") # Conversão do String em bloco de bytes
```

```
        arq.write(bloco)
```

```
except IOError:
```

```
    print("Erro ao abrir ou ao manipular o arquivo.")
```

O parâmetro esperado pelo método **encode(codificação)** deve indicar qual o mapa de caracteres utilizado na conversão de String para bloco de bytes. Caso não seja informado, a codificação utilizada será “utf-8”.

Para evitar erros de interpretação de caracteres, o programador deve garantir que a mesma codificação seja utilizada tanto na escrita quanto na leitura do arquivo.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente n bytes a cada leitura ou escrita, onde n é a quantidade de bytes lidos ou escritos.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente n bytes a cada leitura ou escrita, onde n é a quantidade de bytes lidos ou escritos.

O endereço atual do cursor é retornado pelo método arq.**tell()**.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente n bytes a cada leitura ou escrita, onde n é a quantidade de bytes lidos ou escritos.

O endereço atual do cursor é retornado pelo método arq.**tell()**.

Utilize o método

arq.**seek(endereço relativo, origem)**

para posicionar o cursor no endereço desejado em relação à origem informada.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente n bytes a cada leitura ou escrita, onde n é a quantidade de bytes lidos ou escritos.

O endereço atual do cursor é retornado pelo método arq.**tell()**.

Utilize o método

arq.**seek(endereço relativo, origem)**

para posicionar o cursor no endereço desejado em relação à origem informada.

A origem é um parâmetro opcional que aceita os valores:

0: início do arquivo (padrão, caso origem não seja informada)

1: posição atual do cursor

2: fim do arquivo

Exemplos de Navegação em Arquivo Binário

```
import struct
# Verificando o avanço do cursor de leitura/escrita
try:
    with open("arquivo.bin", "rb") as arq:
        pos = arq.tell()
        print("O cursor está no endereço", pos)
        x = struct.unpack("i", arq.read(4))[0]
        pos = arq.tell()
        print("Após ler 4 bytes, ele foi para o endereço", pos)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

```
# Verificando o tamanho do arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        arq.seek(0, 2)
        tam = arq.tell()
        print("O arquivo possui", tam, "bytes")
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Navegação em Arquivo Binário

```
import struct

# Acesso randômico para leitura do conteúdo do arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        print("Os 10 primeiros valores inteiros armazenados são")
        for x in range(0, 10):
            print(struct.unpack("i", arq.read(4))[0])
        print("Da posição atual, voltarei o cursor para reler os 5 últimos valores")
        arq.seek(-(4 * 5), 1)
        for x in range(0, 5):
            print(struct.unpack("i", arq.read(4))[0])
        print("Agora voltei para o início do arquivo para reler o primeiro valor")
        arq.seek(0)
        print(struct.unpack("i", arq.read(4))[0])

    except IOError:
        print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Navegação em Arquivo Binário

```
import struct

# Subprograma para leitura dos “n” primeiros valores inteiros
def mostrarOsNPrimeirosValores(arq, n):
    print("Os primeiros valores contidos no arquivo são")
    arq.seek(0)
    for k in range(0, n):
        print(struct.unpack("i", arq.read(4))[0])

# Programa Principal para leitura e escrita com acesso randômico
try:
    with open("arquivo.bin", "w+b") as arq:
        print("Escrever os valores inteiro de 1 a 5 no arquivo")
        for x in range(1, 6):
            arq.write(struct.pack("i", x))
        mostrarOsNPrimeirosValores(arq, 5)
        print("Substituir o segundo valor inteiro contido no arquivo por 99")
        arq.seek(4)
        arq.write(struct.pack("i", 99))
        mostrarOsNPrimeirosValores(arq, 5)

    except IOError:
        print("Erro ao abrir ou ao manipular o arquivo.")
```

O Conceito de Registro

É conveniente organizar coleções de dados dentro de arquivos em unidades de tamanho fixo chamadas registros, mesmo que nem todo o espaço reservado para um registro seja efetivamente utilizado.

O Conceito de Registro

É conveniente organizar coleções de dados dentro de arquivos em unidades de tamanho fixo chamadas **registros**, mesmo que nem todo o espaço reservado para um registro seja efetivamente utilizado.

- Exemplo:

Armazenar os dados de um grupo de alunos reservando, para cada aluno, m bytes do arquivo, isto é, um registro de m bytes.

O Conceito de Registro

É conveniente organizar coleções de dados dentro de arquivos em unidades de tamanho fixo chamadas **registros**, mesmo que nem todo o espaço reservado para um registro seja efetivamente utilizado.

- Exemplo:

Armazenar os dados de um grupo de alunos reservando, para cada aluno, m bytes do arquivo, isto é, um registro de m bytes.

- Vantagens:

- Podemos saber quantos registros existem no arquivo, ao tomar o tamanho do arquivo e dividi-lo pelo tamanho do registro.

- Podemos calcular o endereço do início de cada registro utilizando
$$\text{endereço} = \text{chave do registro} \times \text{tamanho do registro}$$

e, assim, usar o método **seek(endereço)** para navegar entre os registros de maneira eficiente. A chave é um valor indexador que vai de zero à quantidade de registros, menos um.

O Conceito de Registro

Registros são compostos por **itens de dados**, também chamados de **campos**. É conveniente que os campos sejam preenchidos na mesma ordem em todo registro colocado em um arquivo.

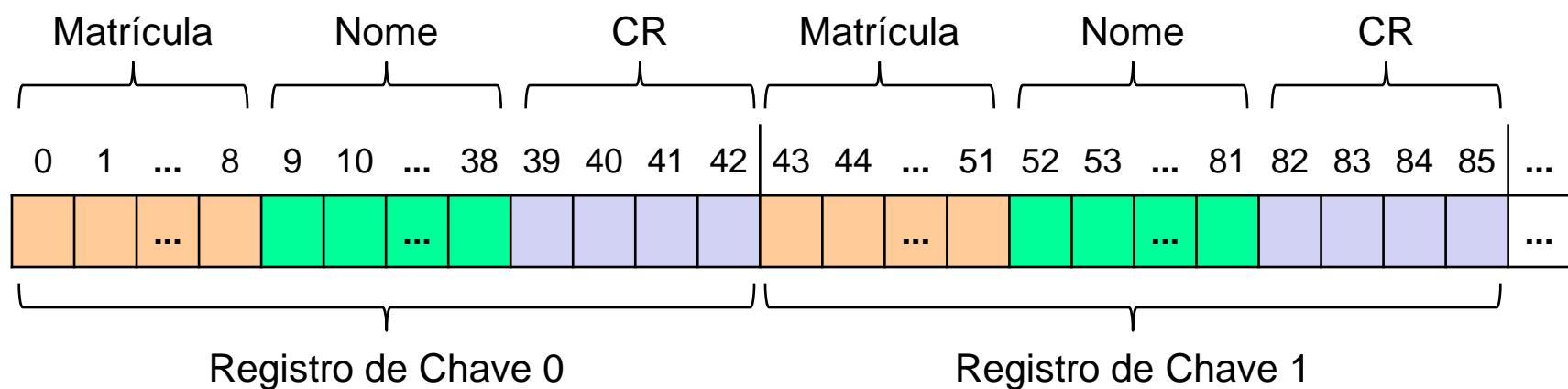
O Conceito de Registro

Registros são compostos por itens de dados, também chamados de campos. É conveniente que os campos sejam preenchidos na mesma ordem em todo registro colocado em um arquivo.

- Exemplo:

O registro de cada aluno contém o código de matrícula (texto com exatamente 9 caracteres UTF-8; 9 bytes), o nome (texto com no máximo 30 caracteres UTF-8; 30 bytes) e o CR (valor float; 4 bytes).

Logo, cada registro de aluno ocupa $m = 43$ bytes neste arquivo.



Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43      # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)      # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)          # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)         # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43      # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)      # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)          # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)         # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43      # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr
```

Programa Principal

```
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)      # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)          # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)         # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43      # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr
```

Programa Principal

```
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)      # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)          # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)         # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43      # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)      # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)          # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)         # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Forma Alternativa para Lidar com Registros

Python oferece mais de uma maneira elegante de empacotar e desempacotar dados para serem escritos ou lidos como registros de tamanho fixo em arquivos binários.

A mais simples é através de estruturas criadas via módulo **struct**:

```
NomeDaEstrutura = struct.Struct(formato)
```

Forma Alternativa para Lidar com Registros

Python oferece mais de uma maneira elegante de empacotar e desempacotar dados para serem escritos ou lidos como registros de tamanho fixo em arquivos binários.

A mais simples é através de estruturas criadas via módulo **struct**:

```
NomeDaEstrutura = struct.Struct(formato)
```

NomeDaEstrutura é uma variável que referencia um objeto Struct capaz de empacotar e desempacotar valores, conforme o formato especificado

formato é uma String que determina a sequência em que os dados serão empacotados pelo método **pack(v1, v2, ...)** ou desempacotados em uma tupla pelo método **unpack(bloco)**

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f") # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f") # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f") # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f") # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)      # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)          # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)         # chave 2

    arq.seek(2 * Aluno.size) ←
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:





Aula 12

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Persistência de Dados
 - Arquivos Binários