



**INSTITUTO
FEDERAL**

Paraná

Campus
Paranavaí

Algoritmos e Estruturas de Dados I

Professores:

Ayslan Trevizan Possebom

Rafael Henrique Dalegrave Zottesso

Ano Letivo 2023

Ambiente Virtual de Aprendizagem (AVA)

Ambiente virtual de aprendizagem é um software, geralmente disponibilizado na web, que permite aos professores fornecerem conteúdos (apostilas, slides, videoaulas, avaliações, etc.) aos seus alunos, assim como acompanhar o progresso destes alunos durante o decorrer da disciplina.

Para acessar o AVA do IFPR, utilize um navegador web (ex: Google Chrome, Firefox, Internet Explorer) e abra o site <https://ava.ifpr.edu.br>.

No canto superior direito da tela, clique no texto (Acessar). Digite seus dados de acesso (CPF e senha cadastrada) e clique no botão Acessar para ter acesso ao sistema.

Contato com os professores

Para entrar em contato com os professores, utilize os seguintes meios de comunicação:

Email: Professor Ayslan: ayslan.possebom@ifpr.edu.br

Professor Rafael: rafael.zottesso@ifpr.edu.br

AVA: Na barra superior, clique no ícone de mensagem, pesquise pelo nome do professor (Ayslan Possebom ou Rafael Zottesso), digite e envie a mensagem.

Livros na Biblioteca ou Biblioteca Virtual

ASCENCIO, Ana F. G.; CAMPOS, Edilene A. V. Fundamentos da Programação de Computadores: algoritmos, Pascal, C/C++ e Java. Editora Pearson, 2007.

FORBELLONE, Andre L. V.; EBERSPÄCHER, Henri F. Lógica de Programação: a construção de algoritmos e estruturas de dados. 3a ed. Editora Pearson, 2005.

GUEDES, Sergio. Lógica de Programação Algorítmica. Editora Pearson, 2014.

PUGA, Sandra; RISSETTI, Gerson. Lógica de Programação e Estruturas de Dados. 3a ed. Editora Pearson, 2016.

BORGES, Luiz E. Python para Desenvolvedores. Editora Novatec, 2014.

RAMALHO, Luciano. Aprenda a Programar. Site Python Brasil. Acessado em 23/03/2022.

Ementa da Disciplina

Introdução à Lógica de Programação. Tipos de Dados e de Variáveis. Operadores Aritméticos, Relacionais e Lógicos. Expressões. Estruturas de Decisão e Controle. Conceitos de Programação Estruturada e Modular. Variáveis. Procedimentos e Funções. Recursividade.

Como devo estudar

Sugere-se o conjunto de atividades a seguir como forma de organização e sequência para o estudo individual dos alunos:

1. **ler o material disponibilizado** pelo professor;
2. **busca na internet** sobre textos e outros materiais;
3. **vídeos** no Youtube ou outros sites (vídeos sobre os assuntos “Algoritmos”, “Lógica de Programação” e “Linguagem Python”);
4. **resolução dos exercícios** propostos; e
5. participação nos **atendimentos acadêmicos**.

É de extrema importância que o estudante tente solucionar os problemas sem utilizar respostas prontas na Internet ou com outros alunos.

Sumário

1. Introdução aos Algoritmos	1
Conceitos iniciais	2
Preparação do ambiente de estudo	2
Criando um arquivo Python no VSCode	3
Informações importantes	3
Apresentando valores na tela do usuário	3
Criando Variáveis	4
Fornecendo dados ao programa pelo Teclado	8
Obtendo os tipos de dados das variáveis	9
Inserindo comentários no algoritmo	10
2. Cálculos mais avançados	13
Visão geral	14
Biblioteca Math	15
Biblioteca Random	16
3. Tomando decisões no código	18
Valor booleano	19
Estrutura condicional simples	20
Estrutura condicional composta	21
Estrutura condicional aninhada	22
Estrutura condicional com múltiplas alternativas	23
4. Estruturas de Repetição	27
Repetição em códigos	28
COMANDO: for i in range(inicio, fim)	29
COMANDO: while(condição)	30
6. Estrutura de Dados Homogênea Unidimensional	34
Arrays Unidimensionais (Vetores)	35
Percorrendo os vetores	37
Criando arrays já com valores inicializados	38
7. Ordenação de vetores	44
Ordenação de Vetores	45
Preliminares	45
Algoritmo de Ordenação por Bolha	46
Algoritmo de Ordenação por Inserção	49
Algoritmo de Ordenação por Seleção	51
8. Estrutura de Dados Homogênea Multidimensional	55
Arrays multidimensionais	56
Entendendo uma matriz	56
Declarando matrizes	57
Acessando valores dentro de uma matriz	59

9. Subprogramação	65
Modularização de algoritmos.....	66
Procedimentos	66
Funções	69
Criando nossas próprias bibliotecas.....	70
10. Arquivos em Disco	78
Arquivos	79
11. Estrutura de Dados Heterogênea.....	86
Registros.....	87
Vetores de Registros	89
Registros em parâmetros de subalgoritmos	90
Registros em bibliotecas	91
12. Acessando Banco de Dados.....	95
Banco de Dados.....	96
Conectando no banco de dados MySQL	98
Finalizando a conexão com o banco de dados.....	99
Executando comandos SQL	99



1. Introdução aos Algoritmos

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Introduzir aos alunos os principais conceitos de algoritmos
- Resolver exercícios de programação sequencial aplicando lógica de programação e linguagem de programação
- Utilizar um Ambiente Integrado de Desenvolvimento
- Favorecer a compreensão e a elaboração de soluções de problemas algorítmicos de forma fluída

O conteúdo trabalhado neste capítulo aborda:

- Conceitos básicos sobre Lógica de Programação e Algoritmos
- Tipos de representações de Algoritmos
- Tipos de Dados
- Criação de variáveis e constantes
- Atribuição de valores
- Entrada e Saída de Dados (teclado e vídeo)
- Comentários
- Operadores aritméticos
- Criação de expressões matemáticas

Conceitos iniciais

Um **algoritmo** se refere a uma descrição das etapas que devem ser executadas para se resolver um determinado problema, ou seja, consiste em uma “sequência ordenada e finita” de ações que deverão ser executadas.

Como exemplo de um algoritmo (sequência lógica de instruções) do dia-a-dia, considere o seguinte trecho de instruções relacionados ao problema: como entrar em contato pelo whatsapp com o professor de Algoritmos para se tirar uma dúvida?

1. Encontrar um telefone
2. O telefone **não tem** o Whatsapp instalado?
 - a. SIM. Voltar ao passo (1)
3. O número do professor de Algoritmos **não está** cadastrado?
 - a. SIM. Então cadastre o número do professor no telefone
4. Abrir o Whatsapp
5. Selecionar o contato do professor
6. Digitar a mensagem contendo a pergunta e enviá-la ao professor
7. O professor **respondeu** a sua pergunta?
 - a. SIM. Sua dúvida **não foi** resolvida?
 - i. SIM. Voltar ao passo (6)
 - b. NÃO. Aguardar um tempo e voltar ao passo (6)
8. Tem **mais algum assunto** para tratar com o professor?
 - a. SIM. Voltar ao passo (6)
 - b. NÃO. Fechar o Whatsapp
9. Continuar estudando

Observe que existe uma sequência de ações a serem executadas. Caso alguma das ações esteja fora de ordem, o resultado do algoritmo será incorreto e o problema não terá sido resolvido com sucesso.

Na programação de sistemas computacionais, os algoritmos são descritos por meio de instruções (comandos) em uma linguagem de programação. Usaremos a linguagem de programação Python para desenvolver nossos algoritmos. Para digitar os comandos da linguagem Python e construir nossos programas, utilizaremos um software conhecido como IDE (Integrated Development Environment – Ambiente de Desenvolvimento Integrado). Este tipo de software fornece um ambiente preparado para digitação de códigos e também para a execução destes códigos de forma simplificada. Para esta disciplina, utilizaremos a IDE chamada Visual Studio Code.

Preparação do ambiente de estudo

Precisamos instalar o *Interpretador Python* (programa responsável por traduzir os códigos na linguagem Python para códigos que são executados no sistema operacional). Para isso, precisamos acessar o site do Python¹. Após fazer o download, basta instalá-lo no computador.

Precisamos instalar o Visual Studio Code² para criar nossos projetos e escrever nossos algoritmos utilizando a linguagem Python. Após fazer o download, seguir o processo de instalação. Ao abrir o Visual Studio Code, precisamos instalar uma extensão (um recurso adicional dentro do software que permite a ele entender como trabalhar com a linguagem) chamada Python (*Python extension for Visual Studio Code*).

¹ Disponível em: <https://www.python.org> ou buscar por Python no Google.

² Disponível em: <https://code.visualstudio.com> ou buscar por VSCode no Google.

Criando um arquivo Python no VSCode

Cada algoritmo que formos desenvolver será um novo arquivo no computador. Utilizando o Visual Studio Code (conhecido popularmente como VSCode), para criar um novo arquivo precisamos clicar no menu **File**, escolher o item **New File**. Um novo arquivo em branco (vazio) será exibido na tela. Para salvar este arquivo, basta escolher o menu **File** e o item **Save**. O arquivo deve ser salvo com a extensão **.py**. Por exemplo: **exercicio01.py**

Para testar se o ambiente está funcionando corretamente, digite o seguinte código no arquivo e salve-o.

```
print("Olá Mundo!")
```

Para executar o programa que contém o código acima, basta clicar com o botão direito do mouse sobre a área “branca” (no código do arquivo) e escolher no menu a opção “Run Python File in Terminal”. O resultado da execução do algoritmo será exibido na área de Terminal, geralmente abaixo do código fonte digitado.

Informações importantes

Na linguagem Python, as instruções são fornecidas sem indentação de código, ou seja, a partir da margem esquerda (início da linha), não podemos fornecer qualquer “espaço em branco” até o comando a ser executado pela linguagem.

Ao fornecer uma “indentação”, estamos criando um “bloco de instruções”. Cada bloco de instruções ocupa geralmente um espaço de 4 caracteres em branco (pressionar a barra de espaços 4 vezes para então escrever os comandos). Por exemplo:

```
x = 10

print("Valor de x: ", x)

if (x > 0):
    print("Valor de x positivo")
else:
    print("Valor de x negativo")

print("Fim do algoritmo")
```

Observe que todos os comandos que são escritos no início de cada linha pertencem ao bloco de instruções principal do programa. O comando **if** definiu um novo bloco, portanto, todos os comandos que pertencem a este bloco devem estar indentados. O comando **else** também definiu um novo bloco de instruções e, portanto, todos os comandos que pertencem a este bloco devem estar “mais a frente” (indentados). O último comando **print("Fim do algoritmo")** pertence novamente ao bloco principal de instruções.

Apresentando valores na tela do usuário

Utilizamos o comando **print()** para exibir uma informação na tela do usuário. Observe que, diferentemente de outras linguagens de programação tais como Java, PHP ou C/C++, não devemos colocar um ; (ponto e vírgula) no fim da instrução. Em Python, cada instrução é colocada individualmente em uma linha no código.

Observe também que existe um par de parênteses após **print**. Dentro dos parênteses precisamos colocar os parâmetros para a função **print** irá utilizar para executar sua ação no algoritmo, ou seja, os valores que iremos fornecer para que a função possa exibí-los na tela do computador.

Para exibir na tela um valor do tipo String, ou seja, um conjunto de caracteres alfanuméricos, podemos colocar uma mensagem utilizando “aspas duplas”.

```
print("Bob Esponja Calça Quadrada")
```


Para exibir na tela um valor do tipo char, ou seja, apenas um único caractere alfanumérico, colocamos o caractere entre aspas simples.

```
print('x');
```

OBSERVAÇÃO: Na linguagem Python, não existe distinção entre Strings e caracteres. Strings também podem ser representadas por aspas 'simples' ou '''triplas''' . Entretanto, para manter uma padronização com outras linguagens de programação, usaremos aspas "duplas" para mensagens (String) e aspas simples para representar um único caractere alfanumérico (ex: 'x').

Para exibir na tela um valor do tipo boolean, ou seja, uma situação onde só existe dois possíveis valores, podemos informar os valores *True* ou *False* para representar uma situação em que é verdadeiro ou uma situação que é falsa. Observe que a linguagem Python é *case-sensitive* (sensível ao caso), ou seja, ela diferencia letras minúsculas de maiúsculas. Desta forma, os valores *True* e *False* devem iniciar com letras maiúsculas.

```
print(True)
print(False)
```

Para exibir na tela um número int, ou seja, um número inteiro positivo ou negativo que não possui parte fracionária, basta fornecer qual número será exibido.

```
print(80)
```

Para exibir na tela um número float, ou seja, um número ponto-flutuante positivo ou negativo que possui parte fracionária, basta fornecer qual número será exibido. Aqui vale observar que para a separação da parte inteira da fracionária utilizamos o . (ponto). Por exemplo, se o preço de um produto for R\$ 1,99, representamos este número por 1.99.

```
print(1.99)
```

Podemos criar frases para serem exibidas na tela do computador. Para isso, separamos por vírgula cada valor a ser exibido, criando uma lista de valores.

```
print("Meu nome é " , "He-Man e tenho " , 15 , " anos")
print("Idade:" , 15 , " e peso:" , 60.8 , "kg.")
```

Neste primeiro exemplo, a String "Meu nome é " será exibida, seguida pela outra String "He-Man e tenho " e também pelo número inteiro 15 e uma outra String " anos". Desta forma, visualizaremos a mensagem "Meu nome é He-Man e tenho 15 anos" exibida na tela.

No segundo exemplo, será exibido a String "Idade:" seguida do número inteiro 15, da String "e peso:" acompanhada do número ponto-flutuante 60.8 e da String "kg.". O resultado que será exibido no monitor será "Idade: 15 e peso: 60.8kg."

Criando Variáveis

Uma variável consiste em um **nome fictício** que damos para uma **posição de memória** do computador. Esta variável deve possuir um **tipo de dado**, ou seja, ela deve representar um valor que seja inteiro, ponto-flutuante, String, caractere, booleano, ou algum outro tipo de dado disponível na linguagem de programação. De forma resumida, usamos variáveis na programação para conseguir **armazenar temporariamente alguns dados durante a execução do programa**.

Por exemplo, se precisamos obter dois números e efetuar a soma destes números, precisamos de duas variáveis, uma para cada número, e mais uma variável para armazenar a soma dos valores das outras duas variáveis.

Na linguagem Python, o tipo de dado de uma variável depende do valor que aquela variável armazena. Por exemplo, se o valor da variável for um número inteiro, então a variável é do tipo inteiro. Se o valor da variável for um valor do tipo String, então a variável assume o tipo String, e assim por diante.

A sintaxe (ou seja, o jeito correto de se escrever os comandos) para criar variáveis é:

<NOME DA VARIÁVEL> = <VALOR DA VARIÁVEL>

Os tipos de dados que iremos utilizar em nossas aulas são:

int	para números inteiros
float	para números ponto-flutuantes
String	para conjuntos de caracteres (textos, frases)
bool	para valores booleanos

Atenção: valores **int** e **float** representam números. Portanto, o resultado de cálculos matemáticos pode ser guardado nestes tipos.

Atenção: **2** é diferente de **2.0** que é diferente de **"2"** ou **'2'**. O primeiro é um número **inteiro**, o segundo é um número **ponto-flutuante**, o terceiro e o quarto são **String** formadas por um único caractere alfanumérico.

O **nome da variável** geralmente indica o que aquele valor significa na execução do algoritmo. Por exemplo:

Nome = "Maria da Silva"

Indica que foi criada uma variável chamada **nome** e esta variável contém uma sequência de caracteres (é uma **String**). O conteúdo que esta variável irá guardar representa um nome.

Por boas práticas de programação, os programadores estabelecem que os nomes das variáveis devem sempre começar com letra minúscula e, caso seja uma palavra composta, as primeiras letras de cada palavra seriam maiúsculas. Por exemplo: **precoDoProduto**, **nomeCompleto**, **primeiroNome**. Outro uso seria o uso do caractere **_** (sublinhado) para separar as palavras: **preco_do_produto**, **nome_completo**, etc.

Regras para os nomes das variáveis:

- O nome da variável **NÃO** pode começar com números, mas pode conter números a partir do segundo caractere
 - 1nome INCORRETO
 - nome1 CORRETO
 - a1 CORRETO
- O nome da variável pode conter letras minúsculas, maiúsculas ou sublinhado, mas por convenção, a primeira letra deve ser minúscula:
 - nome_completo CORRETO
 - nomeCompleto CORRETO
 - _nome CORRETO
 - _1 CORRETO
- O nome da variável deve ser apenas uma palavra:
 - x CORRETO
 - aula1 CORRETO
 - meu nome INCORRETO
- Não podemos dar nomes para variáveis com palavras que possuem significado para a linguagem de programação, ou seja, não podemos usar **palavras-reservadas** da linguagem:
 - if INCORRETO. O comando if existe na linguagem
 - while INCORRETO. O comando while existe na linguagem
 - class INCORRETO. O comando class existe na linguagem
- **NÃO** podemos utilizar caracteres ou símbolos especiais, tais como @, !, %, #, entre outros

- @x INCORRETO
- aula! INCORRETO
- A linguagem Python é *case-sensitive* (sensível ao caso), ou seja, ela diferencia letras maiúsculas de minúsculas. Portanto **nome** é diferente de **nOme** ou **nomE** ou **nOMe**.

Variáveis são criadas atribuindo valores a elas (podemos **atribuir valores** a ela, ou seja, dizer ao algoritmo que a variável possui um valor). Para atribuir um valor a uma variável, usamos o operador de atribuição = seguido do valor a ser guardado nesta variável. Exemplos de criação de variáveis com atribuição de valores:

```
nome = "Mulher Maravilha"
idade = 26
portaEstaAberta = false
peso = 59.8
total = 5
```

Podemos modificar os valores das variáveis a qualquer momento durante a execução do nosso algoritmo. Por exemplo:

```
quantia = 8           #Aqui quantia vale 8
quantia = 10          #Aqui quantia agora vale 10
quantia = 2           #Agora quantia vale 2
```

Na primeira linha, a variável **quantia** foi criada e ela armazena o número 8. Na segunda linha, a variável **quantia** é atualizada e seu novo valor será 10. Por fim, a variável **quantia** vale 2.

Uma dica é: ao resolver um algoritmo, tente planejar quais variáveis serão necessárias. Crie todas estas variáveis logo no início do algoritmo atribuindo valores padrão a elas (ex: 0, 0.0, "", False, etc).

Podemos apresentar os valores das variáveis normalmente:

```
idade = 48
print(idade)
print("Valor da variável idade: ", idade)
```

Usando variáveis, também podemos efetuar cálculos matemáticos. Por exemplo:

```
x = 10
y = 20
z = a + b
print("Resultado: " , z)
```

Inicialmente, a variável **x** foi criada e inicializada com o valor 10. A variável **y** foi criada e inicializada com 20. A variável **z** foi criada e seu valor será 30. Por fim, a mensagem "Resultado: 30" será exibida na tela do usuário.

Dica: Uma forma alternativa de se criar mensagens a serem exibidas ao usuário que contenham textos fixos misturados com valores de variáveis pode ser pela criação de uma String e, no local onde o valor de determinada variável deve ser exibido, colocar os caracteres **{}**. Ao fim da String, utilizamos **.format()** e colocamos as variáveis dentro dos parênteses.

Por exemplo:

```
nome = "Ayrton Senna"
```

```
profissao = "piloto de F1"
idade = 34
totalVitorias = 41

print("O esportista {} atua como {}. Possui {} anos de idade e já obteve {} vitórias".format(nome, profissao, idade, totalVitorias))
```

Observe que na String criada no comando print() existem quatro locais definidos por {}. Isto indica que naquele ponto da mensagem existe um valor a ser exibido. A sequência de valores para serem exibidos é definido após a String em **.format(valor1, valor2, valor3, valor4)**.

Para executar o programa no VSCode, basta clicar com o botão direito do mouse e escolher a opção *“Run Python File in Terminal”*.

Dica: Quando utilizamos atribuições de valores para as variáveis, inicialmente é calculado a expressão do lado da direita (após o sinal de =) e o resultado deste cálculo é guardado na variável do lado da esquerda

```
x = 20 * 2 + 10
```

Calcula-se primeiro (20*2) que resulta em 40 e depois soma-se 10, resultado em 50. Tendo-se 50 como resultado, então a variável x irá armazenar o valor 50.

Dica: Existe uma ordem de precedência para a realização de cálculos matemáticos. Esta ordem é:

1. Primeiro é executado o cálculo que estiver entre **parênteses**
2. Segundo é executado o cálculo que tiver **multiplicação** ou **divisão**
3. Terceiro é executado o cálculo que tiver **soma** ou **subtração**

Com base no exposto acima, observe que para se calcular a média entre dois números, primeiro deve ser executado a soma destes números e, com base neste resultado, é realizado a divisão por 2.

Exemplo: calcular a média entre 2 e 6. (1) calcula-se 2+6=8 e depois temos que 8/2 = 4.

Desta forma, é comum os alunos digitarem:

```
a = 2.0
b = 6.0
media = a + b / 2
```

Entretanto, observando as regras de precedência dos operadores, a divisão é executada primeiro, tendo-se que 6.0/2 = 3.0, e depois a soma é executada obtendo-se 2.0 + 3.0 = 5.0, sendo a resposta incorreta ao cálculo. Para corrigir estes erros, precisamos prestar atenção quanto às prioridades de cálculo. Como a soma deve ser executada antes da divisão, então precisamos colocar parênteses na expressão:

```
a = 2.0
b = 6.0
media = (a + b) / 2
```

Desta forma, como os parênteses são executados primeiro, temos que 2.0 + 6.0 = 8.0. Em seguida, a divisão será executada 8.0 / 2 = 4.0 e obtemos a resposta correta.

Na linguagem Python, os **operadores aritméticos**, ou seja, os sinais utilizados para se fazer cálculos matemáticos são:

+ Soma

-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão (parte inteira do resto da divisão)
**	Expoente (a**b faz o cálculo de a ^b)
//	Divisão arredondada para baixo (9//2 = 4 ou 9.0//2.0 = 4.0)

O resultado do cálculo matemático sempre será do maior tipo de dado utilizado na expressão matemática. Por exemplo, a soma de um inteiro com um inteiro, resulta em um valor inteiro.

```
a = 10
b = 20
c = a + b           //temos que 10 + 20 = 30
```

O resultado do cálculo envolvendo um inteiro com um ponto-flutuante sempre será ponto-flutuante, visto que armazenam valores com casas decimais.

```
a = 10
b = 20.0
c = a + b           //temos que 10 + 20.0 = 30.0
```

A divisão de dois números (independente se eles são inteiros ou ponto-flutuantes) sempre será um número ponto-flutuante.

```
int a = 10
int b = 20
int c = a / b        //temos que 10 / 20 = 0.5
```

Fornecendo dados ao programa pelo Teclado

Para fornecer entrada de dados aos nossos algoritmos, ou seja, para que o usuário consiga digitar valores e estes valores sejam processados, iremos utilizar o comando **input("pergunta")**. Ao usar esse comando, a "pergunta" será exibida na tela do computador e o sistema ficará aguardando que o usuário digite algo no teclado e pressione a tecla ENTER.

Quando o usuário digitar algum valor, este valor digitado deve ser salvo em algum lugar, caso contrário, perderemos a informação dada pelo usuário. Desta forma, **sempre que o usuário for digitar algum valor** para o programa, este valor deve ser salvo em alguma variável.

Uma observação importante a fazer é que devemos garantir que o usuário tenha digitado o valor correspondente ao tipo de dado que se espera. Por exemplo, se o usuário esta digitando o preço de um produto, este valor deve ser ponto-flutuante. Se o usuário esta digitando a quantia de alunos de uma turma, este valor deve ser inteiro, e assim por diante. Desta forma, precisamos converter o que o usuário digitou para o tipo de dado correto, antes de ser atribuído o valor a uma variável. Para fazer estas conversões, usamos as seguintes funções:

int(valor)	converte o valor para números inteiros
float(valor)	converte o valor para números ponto-flutuantes
bool(valor)	converte o valor para booleano (True ou False)

str(valor) converte o valor para String

Desta forma, o seguinte comando faz a pergunta ao usuário, converte o valor digitado para um número inteiro e, em seguida, guarda na variável o valor correto convertido:

```
idade = int( input("Digite a sua idade: ") )
```

Como na matemática, o processamento ocorre de dentro dos parênteses para fora. No caso, a função **input("Digite a sua idade: ")** será executada primeiro. A mensagem será apresentada na tela e ficará aguardando o usuário digitar uma informação. Ao pressionar ENTER, a função foi finalizada. Em seguida, existe a função **int(valor_digitado)** para ser executada que fará a conversão do que o usuário digitou para um número inteiro. Por fim, o valor inteiro será armazenado na variável **idade**.

Outros exemplos:

```
nome = str(input("Digite o seu nome: "))
preco = float(input("Digite o preço do produto: "))
sexo = bool(input("Digite True ou Falso para masc ou fem:"))
total = int(input("Quantos alunos estão online agora? "))
```

Dica: para quem não conseguiu ainda instalar o Python e o VSCode, podemos utilizar alguns sistemas online, tais como:

- <https://repl.it/languages/python3>
 - https://www.onlinegdb.com/online_python_compiler
 - <https://www.programiz.com/python-programming/online-compiler/>
-

Obtendo os tipos de dados das variáveis

Como vimos, os tipos de dados primitivos na linguagem Python são int (para inteiros), float (para valores ponto-flutuantes), str (para valores string) e bool (para valores booleanos). Valores caracteres são tratados como Strings. Para verificarmos se as variáveis possuem os tipos de dados corretos com relação ao seu conteúdo, podemos usar a função **type(variável)**.

Por exemplo:

```
idade = 18
preco = 8.99
nome = "Carlos Eduardo"
letra = 'b'
arCondicionadoLigado = False

tipoIdade = type(idade)
print("Idade", idade, " é do tipo ", tipoIdade)

tipoPreco = type(preco)
print("Preco", preco, " é do tipo ", tipoPreco)

tipoNome = type(nome)
print("Nome", nome, " é do tipo ", tipoNome)

print("Letra", letra, " é do tipo ", type(letra))

print("O ar condicionado está com valor", arCondicionadoLigado, " e
é do tipo ", type(arCondicionadoLigado))
```

Ao executar o programa cima, a saída exibida ao usuário será:

```
Idade 18 é do tipo <class 'int'>
Preço 8.99 é do tipo <class 'float'>
Nome Carlos Eduardo é do tipo <class 'str'>
Letra b é do tipo <class 'str'>
O ar condicionado está com valor False e é do tipo <class 'bool'>
```

Inserindo comentários no algoritmo

Os comentários são utilizados pelo programador para documentar o algoritmo. Eles servem para anotar o que uma função faz, ou o que um conjunto de funções executa, ou outras explicações quem podem auxiliá-lo a lembrar ou explicar determinado assunto.

Ao executar o algoritmo, os comentários são ignorados completamente pelo interpretador (compilador) ao converter os códigos para linguagem de máquina e serem executados pelo sistema operacional.

Temos dois tipos de comentários que podem ser inseridos nos códigos:

- Comentário de uma única linha
- Comentário de múltiplas linhas

Os comentários de uma linha são inseridos utilizando o símbolo #. Tudo o que vier, naquela linha, após o símbolo # será ignorado. Geralmente utilizamos comentários de uma linha para ignorar a execução de uma instrução, ou então para inscrever o que aquela instrução irá executar.

Quanto o comentário a ser feito é grande, com textos explicativos maiores, ou então para comentar um conjunto maior de linhas de código, ele é feito inserindo três aspas (ou simples, ou duplas) para iniciar o comentário, e novamente três aspas (igual ao que foi usado no início) no fim do comentário. Todo o "conteúdo entre as três aspas" será ignorado ao executar o algoritmo.

Exemplo:

```
idade = 18    #criamos variável idade e atribuímos inteiro 18

"""
A função print() é usada para
exibir uma informação na tela do
computador. A mensagem a ser exibida deve
vir dentro dos parênteses
"""
print("A idade informada foi", idade)
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Após resolver os exercícios, aguarde um dia e tente resolvê-los novamente você mesmo, mas agora sem utilizar as respostas e dicas como consulta. Se conseguiu resolver todos os exercícios sozinho, então você está aprendendo o conteúdo!

Exercício 01: Crie um algoritmo que seja capaz de apresentar na tela do computador os seus dados pessoais, tais como nome completo, cidade em que mora e o nome de seus pais.

Exercício 02: Indique qual é o tipo de dado para cada valor a seguir:

- a) 2
- b) "Aula de Java"
- c) "p"
- d) 8.6
- e) 0
- f) 0.0
- g) 'x'
- h) True
- i) 789456
- j) -859.4512
- k) ""
- l) ' '
- m) False

Exercício 03: Crie um algoritmo que seja capaz de apresentar na tela o seu nome completo (uma String), a sua idade (um número inteiro), a primeira letra do seu nome (um char), a sua altura (um número ponto-flutuante) e o sexo (use True para masculino ou False para feminino).

Exercício 04: Crie variáveis (com valores imaginários) que sejam capazes de armazenar os seguintes valores:

- a. O seu nome completo
- b. A sua idade
- c. O número de biscoitos Passatempo que o supermercado possui em estoque
- d. O preço dos biscoitos Passatempo no supermercado
- e. Se o ar condicionado está ligado ou desligado (repare que existe apenas 2 valores)
- f. O nome da sua cidade
- g. A sigla do seu estado
- h. O total de habitantes que existe na sua cidade
- i. A distância entre duas cidades (repare que podem existir km e metros)
- j. A sua altura
- k. O seu peso
- l. O dia do seu nascimento
- m. O mês do seu nascimento
- n. A rua em que você mora
- o. O número da sua casa
- p. A quantidade de teclas que um teclado de computador possui
- q. Se a porta está aberta ou fechada
- r. A primeira letra do seu nome

Exercício 05: Crie as variáveis x e y que recebam dois valores inteiros informados pelo usuário. Crie também a variável z, que receberá a soma das variáveis x e y. Exiba na tela o valor da variável z.

Exercício 06: Você está dirigindo um carro e precisa abastecer. Ao chegar no posto de combustíveis, você repara no preço da gasolina (por litro) e na quantia de dinheiro que você possui. Com estes dados, você pode calcular quantos litros de combustível você poderá abastecer o seu carro. Implemente um algoritmo que solicite ao usuário pelo preço atual do litro da gasolina e também solicite ao usuário pela quantia de dinheiro que ele possui. Calcule e apresente na tela quantos litros de gasolina ele poderá abastecer.

Exercício 07: Você é funcionário de uma empresa e recebe um salário mensal. Seu chefe chega até você e diz que você terá um aumento no seu salário. Um algoritmo pode ser desenvolvido para se calcular este novo salário tendo-se como base o salário atual do funcionário e a porcentagem de aumento que este salário será atualizado. Implemente este algoritmo, solicitando ao usuário pelo salário atual do funcionário e pela porcentagem de aumento. Calcule e apresente o novo salário do funcionário.

Exercício 08: Qual é o resultado das seguintes operações matemáticas:

- $2 * 3 + 5$
- $2 + 6 / 2$
- $5 * 8 + 4 / 2$
- $5 + 8 - 1 / 3$
- $5 + 8 - 1.0 / 3$
- $(5 + 8 - 1) / 3$
- $(5 + 8 - 1) / 3.0$
- $4 / 3 + 8.0 / 2.0$

Exercício 09: Um cálculo realizado constantemente, em especial para quem deseja viajar ao exterior ou ir fazer compras no Paraguai é a conversão entre os valores do Real e do Dólar. Tendo-se o valor atual do Dólar e uma quantia em Reais, podemos calcular quantos dólares poderemos comprar. Para otimizar este processo de conversão, elabore um algoritmo que solicite ao usuário pela cotação atual do dólar (ex: 5.65) e pela quantia de reais que se deseja converter (ex: 170.00). O algoritmo deverá apresentar ao usuário quantos dólares ele poderá comprar.

Exemplo: Se U\$1,00 custa R\$5,65 e se tivermos R\$170,00, então poderemos comprar U\$30,08.

Exercício 10: Na física, as temperaturas são medidas de acordo com algumas escalas numéricas, tais como graus Célsius ou graus Farenheit. Como existe uma equivalência entre estas unidades de medida, elas podem ser convertidas uma na outra e vice-versa. Para se converter uma temperatura Farenheit para Celsius, podemos utilizar a fórmula $C = (5.0 / 9.0) * (F - 32)$.

Exemplo: Se a temperatura for de 50°F, então temos que ela está a 10°C

Crie um algoritmo que solicite ao usuário por um grau na temperatura Farenheit, calcule e apresente ao usuário a correspondência deste grau na escala Celsius.

Exercício 11: Algumas atividades esportivas são medidas, por um cronômetro, para determinar o tempo que o atleta utilizou para concluir uma prova. Os cronômetros podem registrar horas, minutos e segundos.

Implemente um algoritmo que solicite ao usuário pela quantia de horas, minutos e segundos que um atleta levou para concluir uma prova. O algoritmo deve calcular e apresentar ao usuário todo o tempo que o atleta utilizou em segundos (converta horas, minutos e segundos para apenas segundos).

Exercício 12: Desenvolva um algoritmo que solicite ao usuário por um número inteiro. Em seguida, o algoritmo deve apresentar na tela uma sequência de números em ordem crescente, incluindo os dois números anteriores ao número digitado, o número digitado e os dois números após o valor digitado.

Exemplo: Ao digitar o número 3, os números 1 2 3 4 5 deverão ser apresentados na tela.

Objetivos de Aprendizagem

- Resolver exercícios de programação sequencial aplicando lógica de programação e linguagem de programação
- Utilizar um Ambiente Integrado de Desenvolvimento
- Favorecer a compreensão e a elaboração de soluções de problemas algorítmicos de forma fluída
- Utilizar entrada de dados, processamento de dados e saída de dados
- Processar dados com cálculos matemáticos e números aleatórios

- Cálculos matemáticos utilizando a classe Math
- Geração de números aleatórios com a classe Random

Visão geral

Ao desenvolvermos algoritmos sequenciais, geralmente fazemos cálculos sobre os dados informados ao algoritmo. Para fazer cálculos matemáticos, vimos que os operadores aritméticos podem ser utilizados.

Relembrando os **operadores aritméticos**, eles são:

+	Soma
-	Subtração
*	Multiplicação
**	Potência (exponenciação)
/	Divisão
//	Parte inteira da divisão
%	Resto da divisão (sempre será um número inteiro)

Entretanto, estes cálculos são os mais básicos.

DICA: observar a prioridade de execução dos operadores.

Exponenciação tem maior precedência. Em seguida, a Multiplicação, Divisão e Resto são avaliados primeiro, da esquerda para a direita

Soma e subtração são avaliados em seguida, da esquerda para a direita

Atribuição de valores (=) para variáveis será executado por último, após já obter o resultado do cálculo.

Necessidade do uso de **parênteses** para garantir que o cálculo seja executado conforme a fórmula desejada.

DICA: O resultado obtido em uma expressão matemática sempre terá o **tipo de dado** do maior tipo utilizado nesta expressão.

Exemplo: $5 + 3$ gera como resultado 8 (soma de números inteiros gera como resultado número inteiro)

Exemplo: $5 + 3.0$ gera como resultado 8.0 (como a fórmula envolveu um número ponto-flutuante, então o resultado será ponto-flutuante)

Exemplo: $5 / 2$ gera como resultado 2.5 (divisões gera como resultado número ponto-flutuante)

Exemplo: $5.0 / 2$ gera como resultado 2.5 (divisões gera como resultado número ponto-flutuante)

Exemplo: $3.0 + (5 / 2)$ gera como resultado 5.5 (observe que primeiro será executado $5/2$ tendo como resultado 2.5; em seguida executará da esquerda para a direita o cálculo $3.0 + 2.5$ que é igual a 5.5).

Se os cálculos necessários envolverem logaritmos, raízes quadradas, potências ou mesmo o sorteio de números aleatórios, precisamos utilizar uma estrutura mais avançada.

Utilizaremos inicialmente a biblioteca **math** para cálculos matemáticos mais avançados e a biblioteca **random** para a geração de números aleatórios.

Biblioteca Math

Se o problema do algoritmo exigir que funções matemáticas mais avançadas sejam calculadas, tais como o cálculo do seno, cosseno, tangente, logaritmo, etc, a linguagem Python oferece uma biblioteca repleta de funções matemáticas. O nome desta biblioteca é **math**. Para que possamos utilizá-la em nosso programa, logo no início do código-fonte, precisamos importá-la, ou seja, informar ao Python que nosso programa utiliza funções que estão definidas fora do nosso arquivo com os códigos de programação. Para importar:

```
import math
```

Para o uso da biblioteca math, consulte alguns sites na Internet e na bibliografia sugerida. Alguns exemplos:

Para calcular a raiz quadrada de algum número, podemos utilizar a função **math.sqrt(valor)**. Exemplo:

```
import math
x = math.sqrt(9)
print(x)
```

Para calcular uma potência (base elevado ao expoente), podemos utilizar a função **math.pow(base, expoente)**.

```
import math
a=4
b=5
x = math.pow(a,b)
print(a, " elevado a ", b, " = ", x)
```

Outras funções interessantes: **math.ceil(valor)** arredonda um número para cima; **math.floor(valor)** arredonda um número para baixo; **math.log(valor)** para calcular o logaritmo de um número; **math.log(valor, base)** para calcular o logaritmo de um número indicando qual será a base utilizada; **math.cos(valor)** para calcular cosseno; etc.

Aprenda mais sozinho:

Busque na Internet qual é o comando da classe **Math** que pode ser utilizado para:

- Arredondar um número
- Calcular o logaritmo de um número
- Calcular o cosseno de um número
- Obter o valor de π (pi)

Desenvolva um algoritmo de exemplo que exiba na tela o resultado do uso destas funções.

Biblioteca Random

Além da biblioteca de funções matemáticas, também é importante sabermos como gerar números aleatórios que são sorteados ao acaso. Para gerar números aleatórios, precisamos importar no início do algoritmo a biblioteca **random**:

```
import random
```

Para sortear um número aleatório, chamamos a função **random.randint(inicio,fim)**. Por exemplo, se quisermos que seja sorteado um número aleatório entre 12 e 30, usamos **random.randint(12, 30)**. Exemplo:

```
import random

x = int(input("Digite um número entre 10 e 15: "))

y = random.randint(10, 15)

if(x == y):

    print("Você acertou o número sorteado")

else:

    print("Você errou. O número sorteado foi", y)
```

Neste exemplo, o usuário digitará um número no teclado e o valor digitado será armazenado na variável **x**. Em seguida, o computador irá sortear um número entre 10 e 15. Depois será feito um teste para verificar se o número que o usuário digitou é o mesmo número sorteado. Em caso positivo, a mensagem “Você acertou o número sorteado” será exibida na tela. Em caso negativo, a mensagem “Você errou. O número sorteado foi x” será exibida na tela.

Dica: a função **random.randint(inicio, fim)** sorteia números inteiros no intervalo fornecido do início até o fim. Caso desejarmos sortear números ponto-flutuantes, podemos utilizar a função **random.uniform(inicio, fim)**.

Dica: se tivermos uma lista de valores e desejarmos sortear algum destes valores desta lista, podemos utilizar a função **random.choice([lista, de, valores])**. Por exemplo:

```
import random

frutas = ["abacaxi", "morango", "goiaba", "tangerina"]

sorteio = random.choice(frutas)

print("Fruta sorteada: ", sorteio)
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Exercício 01: Imagine que você esteja desenvolvendo um aplicativo para telefones celulares para auxiliar motoristas. Este aplicativo tem o objetivo de calcular e apresentar a média de consumo de combustível utilizado por um veículo. Sabe-se que para este cálculo, precisamos conhecer a quilometragem inicial do veículo antes da viagem e a quilometragem final do veículo depois da viagem, assim como a quantidade de combustível que foi abastecida nesta viagem. Implemente um algoritmo capaz de fornecer esta solução..

Exercício 02: Você e seus amigos adoram se reunir nos finais de semana para uma confraternização e, nestes encontros, sempre fazem churrasco. Nesta organização da confraternização, geralmente precisamos estabelecer a quantidade de produtos que iremos comprar (ex: quantidade de refrigerantes, sucos, carnes, saladas, pães, etc.). Para facilitar a organização da confraternização, você ficou responsável por implementar um programa que seja capaz de calcular a quantidade de carne que será necessária. Para isso, precisamos conhecer a quantidade de homens e o consumo médio (em Kg) que os homens fazem da carne, assim como a quantidade de mulheres e o consumo médio (em Kg) de carne para as mulheres. Para garantir que não faltará carne na confraternização, precisamos comprar 20% de carne a mais. Implemente um programa que seja capaz de apresentar a quantidade de carne a ser comprada para uma confraternização conforme apresentado.

Exercício 03: Escreva um algoritmo que solicite ao usuário por um número inteiro. Apresente na tela o cubo deste número (utilize uma função matemática para o cálculo do cubo).

Exercício 04: Você é o dono de um restaurante e neste restaurante você cobra de seus clientes uma taxa de serviço de atendimento do garçom. Como você está estudando algoritmos, teve a ideia de criar um programa de computador para facilitar os cálculos. Sabe-se que a partir do valor total de uma conta, você cobra 10% a mais de seus clientes e repassa este valor aos seus garçons. Implemente um algoritmo que solicite ao usuário pelo valor total da conta e apresente na tela o valor total da conta, quanto será cobrado do cliente e quanto o garçom irá receber por este atendimento.

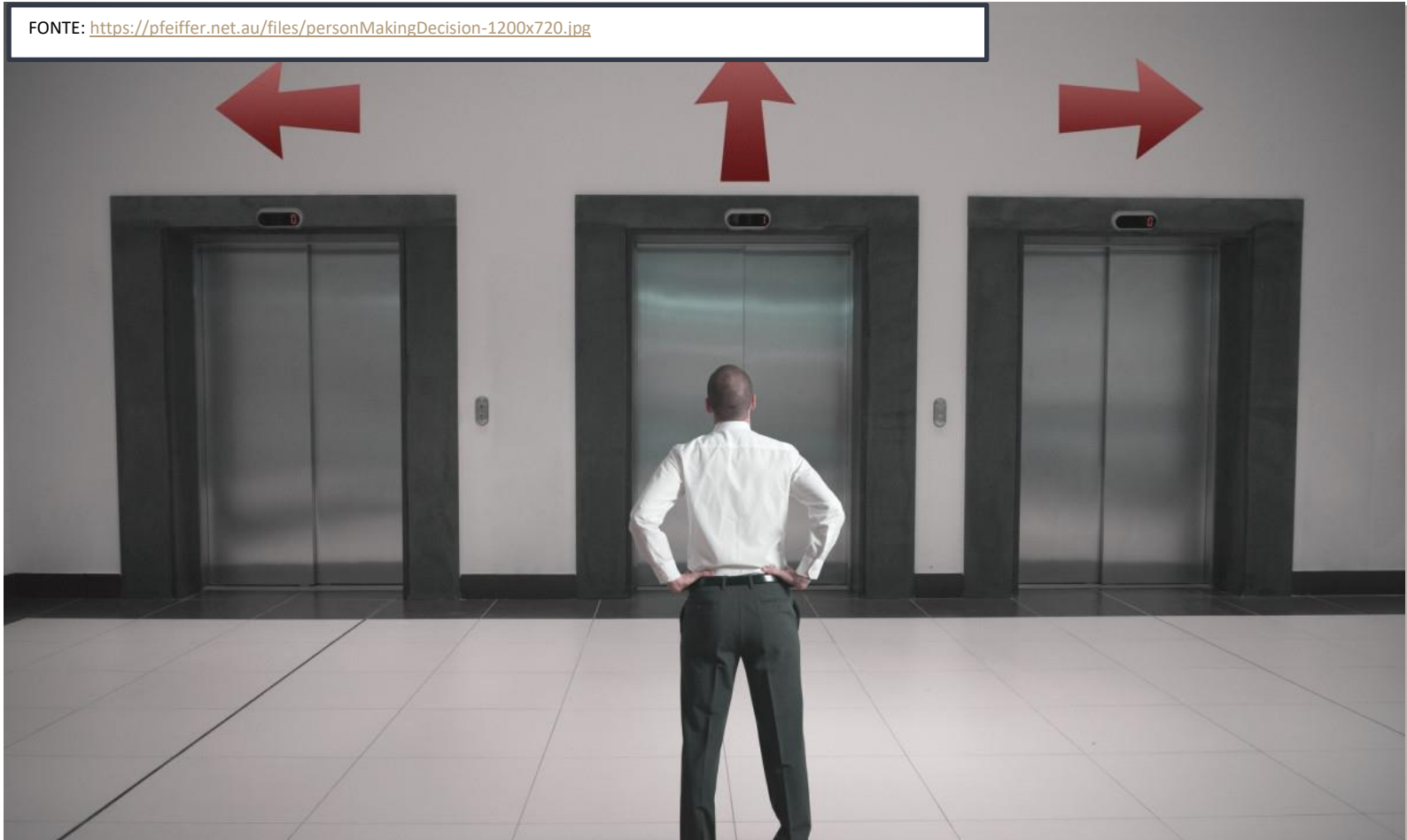
Exercício 05: Você adora jogar dados. Sabe-se que os dados possuem 6 lados. Cada vez que você joga um dado, um valor aleatório entre 1 e 6 é obtido. Para simplificar este jogo de dados, implemente um programa de computador que seja capaz de sortear 5 números inteiros (equivalente a jogar o dado 5 vezes) entre 0 a 5. Apresente na tela os valores sorteados.

Exercício 06: Vimos que os números obtidos por `random.randint(X, Y)` sorteiam valores aleatórios que vão de X até Y. Neste sentido, imagine uma situação onde devemos obter números aleatórios entre 5 e 25. Neste intervalo de valores, incluindo o 5 e o 25, temos um total de 21 valores possíveis (5, 6, 7, 8, ... 24, 25).

Implemente um algoritmo que solicite ao usuário por um número mínimo e um número máximo. O algoritmo deve sortear um número aleatório neste intervalo e apresentar o número sorteado ao usuário.

Exercício 07: No meio financeiro, frequentemente medimos o salário de uma pessoa com base no valor do salário mínimo atual definido pelo governo. Você pode facilitar a vida dos empresários neste cálculo. Implemente um algoritmo que solicite ao usuário pelo valor do salário mínimo atual e o valor do salário do funcionário de uma empresa. Calcule e apresente na tela quantos salários mínimos o funcionário recebe.

Exercício 08: Implemente um algoritmo que solicite ao usuário por três números inteiros, A, B e C. Após o usuário informar os três números, apresente-os na tela. Em seguida, o algoritmo deve trocar os valores das variáveis, fazendo com que a variável A receba o valor de B, a variável B tenha o valor de C e a variável C possua o valor de A. Após a troca dos valores das variáveis, apresente novamente os valores das variáveis atualizadas na tela.



3. Tomando decisões no código

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Resolver exercícios de programação sequencial aplicando lógica de programação e linguagem de programação
- Utilizar um Ambiente Integrado de Desenvolvimento
- Processar dados conforme situações que envolvam decisões sobre quais instruções deverão ser executadas

O conteúdo trabalhado neste capítulo aborda:

- Estrutura de seleção simples
- Estrutura de seleção múltipla
- Estrutura de seleção encadeada

Valor booleano

Ao desenvolvermos algoritmos mais aprimorados, é frequente o uso de tipos de dados booleanos, ou seja, o uso de valores **True** ou **False** (observe que T e F se escreve com letra maiúscula). O tipo booleano é utilizado em situações onde temos apenas duas possibilidades. Por exemplo: O ar condicionado está ligado? (a resposta seria apenas Sim ou Não), a porta está fechada? (a resposta poderia ser Sim ou Não), o aluno é maior de idade? (a resposta seria apenas Sim ou Não), o produto do mercado está caro? (a resposta seria apenas Sim ou Não), etc. Adotamos um padrão onde **True** (verdadeiro) seria a resposta Sim, enquanto o valor **False** (falso) representa a resposta Não. Portanto, se você possui uma altura de 1,90 metros e alguém perguntar: Você é alto? A resposta possivelmente seria True.

Ao trabalharmos com valores booleanos, geralmente utilizamos os **operadores relacionais**. Para que servem os operadores relacionais? Para fazermos comparações entre valores. Os operadores relacionais são:

Descrição	Operador
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=
Diferente de	!=
Igual a	==

O resultado do uso de operadores relacionais (efetuar comparações de valores) sempre será um valor booleano True ou False. Por exemplo:

```
x = 10
y = 20
print( x < y )      #True, pois 10 < 20 é verdadeiro
print( x <= y )     #True
print( x > y )      #False
print( x >= y )     #False
print( x == y )     #False, pois 10 == 20 é falso (10 não é igual a 20)
print( x != y )     #True, pois 10 != 20 é verdadeiro (10 é diferente de 20)
```

Ainda falando sobre o tipo de dado booleano, também temos os **operadores lógicos**. Estes operadores são usados, em especial, quando temos uma comparação que envolve muitos valores. Por exemplo: Você irá para a praia nas férias apenas se você tiver passado de ano (a resposta é sim ou não) e, ao mesmo tempo, tenha o dinheiro necessário para curtir as férias na praia (a resposta é sim ou não). Caso alguma das respostas seja não (ou você não tenha passado de ano, ou você não tenha o dinheiro suficiente), então você não poderá ir para a praia nas férias. Os operadores relacionais são:

Descrição	Operador
E	and
Ou	or
Não	not()

Como usamos os operadores lógicos sempre para unir expressões lógicas (booleanas, valores True ou False), o resultado também será um valor lógico (True ou False). Para saber se o resultado da expressão lógica é verdadeiro ou falso, utilizamos as **tabelas-verdade**.

Tabela-verdade para and

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Tabela-verdade para or

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Tabela-verdade para not

A	not(A)
True	False
False	True

Desta forma, considere a variável **a = 10** e **b = 20**:

```
a = 10
b = 20

print( a > 0 and b > 30 )
```

O Código acima exibirá na tela o valor False, visto que:

- para **a>0** temos **10 > 0** e o resultado da comparação será **True**.
- para **b>30** temos **20 > 30** e o resultado da comparação será **False**.
- **True and False** temos que o resultado de **True and False** é sempre **False**.

Os operadores relacionais e os operadores lógicos são extremamente importantes quando desenvolvemos algoritmos. Portanto, precisamos conhecê-los “de cor”, em especial os resultados das tabelas-verdade. Usamos estes operadores na tomada de decisões, no controle de repetições, e muitas outras situações ao criar diferentes estruturas de dados e estruturas de controle nos algoritmos.

Estrutura condicional simples

Utilizamos a estrutura condicional simples, ou estrutura de decisão simples, apenas quando queremos que um conjunto de instruções seja executado caso uma determinada condição seja satisfeita. Por exemplo: vamos para a escola todos os dias, mas apenas se hoje tiver prova agendada, faremos a prova. Ou seja, o padrão do algoritmo é “ir para a escola todos os dias”. Mas “se hoje tiver prova, então vamos para a escola normalmente e, além disso, faremos a prova”. Repare que a ação de fazer uma prova depende de uma condição. É uma ação que apenas será executada caso uma condição (comparação de valores) for satisfeita, no caso, verificar se hoje temos prova ou não.

O comando **if(condição)**: é utilizado para este fim: tomar decisões se um bloco de instruções será ou não executado. Sempre que uma condição for verdadeira (**True**), então o bloco será executado. Um **bloco de instruções** é definido quando temos o símbolo de início de bloco : (dois pontos) e, nas linhas abaixo, existe uma indentação (tabulação do código, deixar uma margem do começo da linha até o início da instrução). Desta forma, todas as instruções que estão indentadas fazem parte deste bloco de instruções. Por exemplo:

```
x = 10

if(x > 0):

    print("Valor maior que zero")

print("Fim do programa")
```

No exemplo acima, temos que a variável `x` possui o valor 10. Na segunda linha temos uma comparação, verificando **se `x` for maior que zero, então** o bloco de instruções do comando `if` será executado. Como `10 > 0` resulta em um valor **True**, então o bloco será executado e a mensagem “Valor maior que zero” será exibida na tela do computador. Por fim, será exibida a mensagem “Fim do programa”.

Se no exemplo, a primeira linha tivesse a instrução `x = 0`, então o teste feito na segunda linha `if(x>0)` utilizaria a comparação `0 > 0`, resultando em um valor **False**. Portanto, o bloco de instruções não será executado e o algoritmo pula para a primeira instrução fora do bloco, exibindo apenas a mensagem “Fim do programa”. Portanto, o fluxo de execução das instruções ocorre conforme Figura ao lado. Caso a condição seja satisfeita, um conjunto de instruções será executado, caso contrário, este conjunto será ignorado e o algoritmo segue seu fluxo normalmente nas instruções definidas em seguida.

Outro exemplo:

```
preco = 500.89

valor_a_pagar = preco

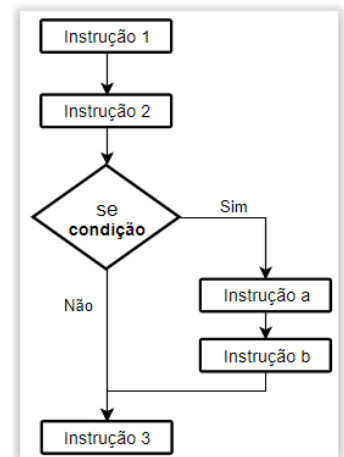
if(preco > 1000):

    desconto = 0.9 #10 de desconto

    valor_a_pagar = preco * desconto

print("O preço final foi de R${}".format(valor_a_pagar))

print("Fim do programa")
```



Neste exemplo, inicialmente a variável `preco` vale 500.89 e a variável `valor_a_pagar` vale o mesmo valor que `preco` (logo, 500.89). A terceira linha faz um teste verificando **se o valor de `preco` é maior que 1000**, em uma situação onde, se o preço de um produto custar mais que R\$1000,00 então será fornecido um desconto de 10% no valor a pagar. Como `500.85 > 100` resulta em um valor **False**, então o algoritmo pula para a primeira instrução após o bloco de instruções, exibindo as mensagens “O preço final foi de R\$500.89” e “Fim do programa”.

Se o valor do preço fosse definido como, por exemplo, `preco = 2000`, então o teste `2000 > 1000` resulta em um valor **True** e, desta forma, o bloco de instruções será executado, alterando o valor da variável **valor_a_pagar** oferecendo um desconto de 10% sobre o preço do produto. Por fim, as mensagens “O preço final foi de R\$1800.0” e “Fim do programa” serão exibidas.

Estrutura condicional composta

A estrutura de decisão apresentada anteriormente é bastante simples: só define um bloco de instruções para ser executado **se** uma condição for satisfeita. Entretanto, a maioria das situações envolve duas direções: ou executar uma ação caso a condição seja satisfeita (True), ou executar outra ação diferente caso a condição não seja satisfeita (False). Por exemplo, se um medicamento for aplicado a uma criança até 4 anos, então a dose deve ser de 4ml. Em contrapartida, se a criança tiver mais que 4 anos ou já for um adulto, a dose pode ser de 8ml. Observe que o valor da dose muda dependendo do caso.

Esta situação poderia ser implementada facilmente com a estrutura condicional simples. Entretanto, precisaríamos definir duas condições: uma para verificar se a idade da criança é menor ou igual a 4 anos, e outra estrutura para verificar se a idade da criança é maior que 4 anos:

```
idade = 3
dose = 0
if(idade <= 4):
    dose = 4
if(idade > 4):
    dose = 8

print("A dose da criança de {} anos é de {}ml".format(idade, dose))
print("Fim do programa")
```

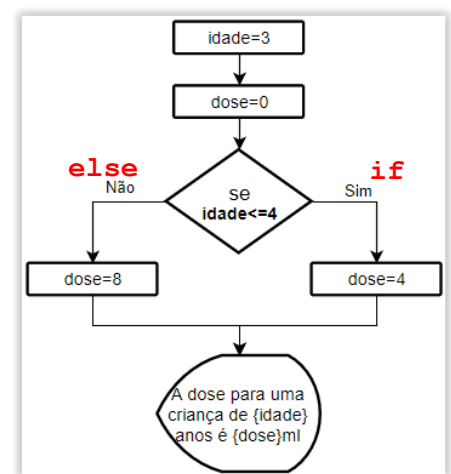
Para simplificar esta situação de termos que executar diversas comparações para resolver os nossos problemas de tomada de decisão, podemos utilizar os comandos **if(condição):** e **else:**. Sempre que uma condição for verdadeira (True), então o bloco do **if(condição):** será executado. Caso seja falso (False), o bloco do **else:** será executado.

OBSERVAÇÃO: a condição é feita apenas 1 vez. E ela é feita dentro dos parênteses do comando **if**. O comando **else** não possui nenhuma comparação, já que ele será executado apenas se a condição já feita no **if** for False.

Por exemplo:

```
idade = 3
dose = 0
if(idade <= 4):
    dose = 4
else:
    dose = 8

print("A dose da criança de {} anos é de {}ml".format(idade, dose))
print("Fim do programa")
```



Como o valor da variável idade é 3, então a comparação realizada no comando **if** para $3 \leq 4$ resulta em um valor True. Desta forma, a dose receberá o valor 4 (o bloco definido pelo **else** não será executado, já que executou o **if**). No caso do valor da idade ser, por exemplo, 10, então o teste $10 \leq 4$ será False e o **if** não será executado, fazendo então que o bloco do **else** seja executado.

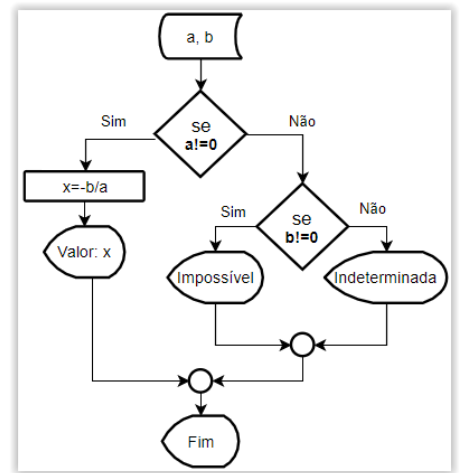
Estrutura condicional aninhada

Podemos ter diversos comandos **if/else** dentro de comandos **if/else**, ou seja, incluímos comandos de decisão dentro do bloco de instruções de comandos **if/else**. Por exemplo: para resolver uma equação do 1º grau com a expressão $ax + b = 0$, então temos que $x = \frac{-b}{a}$ se o valor de a for diferente de zero. Caso contrário a solução pode ser impossível de se obter (se o valor de b for diferente de zero) ou indeterminada (se ambos a e b forem zero).

```

a = int(input("Digite valor de a:"))
b = int(input("Digite valor de b:"))
if(a != 0):
    x = -b / a
    print("Valor: ", x)
else:
    if(b != 0):
        print("Solução impossível")
    else:
        print("Solução indeterminada")
print("Fim")

```

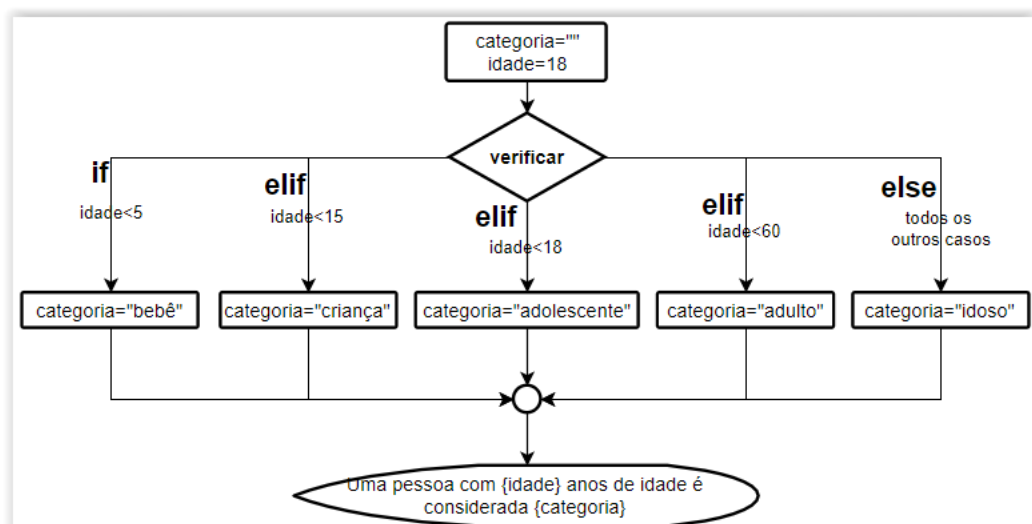


Estrutura condicional com múltiplas alternativas

Considere a seguinte situação: você liga para um tele-atendimento da operadora de telefonia móvel. O sistema informatizado te oferece um conjunto de opções: digite 1 para contratar serviços, digite 2 para cancelar o serviço, digite 3 para falar com um de nossos atendentes, digite 4 para.....

Nestes sistemas, temos vários “blocos de instruções” para serem executados e apenas um deles será selecionado para execução. Esta situação necessita da tomada de decisão sobre qual bloco será executado e esta decisão possui múltiplas alternativas. Como o comando **if(condição)**: possui apenas uma condição e um bloco de instruções, caso a condição seja falsa, o bloco a ser executado será o **else**. Precisamos de uma estrutura intermediária, que permita fazer outras condições além daquela estabelecida no comando **if**.

Para resolver este problema, temos o comando **elif(condição)**. Este comando é usado igual o comando **if**: precisa de parênteses e uma condição cujo resultado seja True ou False. Caso a condição seja satisfeita, o seu bloco de instruções será executado. Colocamos este comando entre os comandos **if** e **else**.



Por exemplo:

```
categoria = ""
idade = 18
if(idade < 5):
    categoria = "bebê"
elif(idade < 12):
    categoria = "criança"
elif(idade < 18):
    categoria = "adolescente"
elif(idade < 60):
    categoria = "adulto"
else:
    categoria = "idoso"
print("Uma pessoa com {} anos de idade é considerado {}".format(idade,
    categoria))
```

A variável **categoria** é usada para armazenar um texto que indica qual a categoria de idade que uma pessoa pertence. A variável **idade** representa a idade de uma pessoa. O primeiro comando da tomada de decisão **sempre será** o comando **if(condição)**. Caso a condição seja satisfeita, então este bloco será executado. Se a condição for False, então o próximo comando **elif(condição)** será verificado. Caso o valor da nova condição seja True, então este será o bloco escolhido para ser executado. No caso do valor ser False, então o algoritmo segue testando o próximo **elif(condição)**. No caso de nenhuma das condições anteriores serem satisfeitas, apenas neste caso, o bloco do comando **else** será executado. Como no exemplo acima a variável idade tem o valor 18, então o primeiro teste (comando if) será False, o próximo comando a ser testado (comando elif) também será False, o terceiro teste será feito (comando elif) e também receberá resultado False e, por fim, o próximo teste será feito na condição do último comando elif obtendo como resultado True e, portanto, este bloco será executado, atribuindo à variável categoria o valor “adulto”.

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Exercício 01: Um jogo bem tradicional entre os jovens é o jogo de par ou ímpar. Neste jogo, as pessoas esticam um braço com uma quantia aleatória de dedos esticados. A somatória dos números dos dois jogadores indica quem foi o vencedor: quem escolheu o número par e quem escolheu o número ímpar. Implemente um algoritmo para jogar contra o computador. Você deve informar um número inteiro ao algoritmo e o computador deve sortear um número inteiro entre 0 e 5. Você deve informar ao computador se você deseja ser o “par” ou o “ímpar”. O computador deve emitir a mensagem indicando se você ganhou ou se perdeu o jogo.

Exercício 02: Faça um programa que peça um valor e mostre na tela se o valor é positivo, negativo ou igual a zero.

Exercício 03: Faça um programa que peça ao aluno o conceito dele na disciplina (A, B, C ou D). Se o conceito for A, B ou C apresente “Aprovado”, se não, apresente “Reprovado”. Se o usuário informar um conceito inválido, exiba uma mensagem informando que o conceito é inexistente. Utilize operadores lógicos para fazer as comparações necessárias.

Exercício 04: Faça um Programa receba o valor de x , calcule e imprima o valor de $f(x)$ que será:

$$f(x) = \begin{cases} \frac{1}{2-x}, & x < 2 \\ \frac{x^3}{2-\sqrt{x}}, & x \geq 2 \end{cases}$$

Utilize funções matemáticas para o cálculo da raiz quadrada e da potência.

Exercício 05: Uma calculadora pode ser feita via programação. Considere um algoritmo que faça a leitura de 2 números e de um operador matemático (+, -, * ou /). O algoritmo deve efetuar o cálculo indicado no operador com os dois números informados e apresentar o resultado da operação. Exemplo: $x = 10$, $y = 20$, sinal = '+'. O resultado da operação deverá ser $z = 30$.

Exercício 06: Um jogo muito conhecido entre as crianças é o Pedra-Papel-Tesoura. Neste jogo, a pessoa simboliza a pedra com a mão fechada, o papel com a mão aberta, e a tesoura com a mão fechada e os dedos indicador e médio estendidos. As regras para este jogo são: Pedra ganha de Tesoura (já que a pedra pode amassar e quebrar a tesoura); Tesoura ganha do Papel (já que a tesoura pode cortar o papel) e Papel ganha da Pedra (já que o papel pode embrulhar a pedra). Com base neste jogo, implemente um algoritmo que solicite ao usuário por um número inteiro onde 1 representa Pedra, 2 representa Papel e 3 representa Tesoura. O computador é o jogador adversário e, portanto, deve sortear um número entre 1 e 3. Informe ao usuário se ele venceu ou perdeu o jogo.

Exercício 07: Implemente um algoritmo que seja capaz de solicitar ao usuário por um número (x). O algoritmo deve ser capaz de calcular e imprimir o valor do seguinte cálculo:

$$f(x) = \begin{cases} \frac{5x+3}{\sqrt{x^2-16}}, & x \geq 10 \\ \frac{x}{\log(x^3)}, & x < 10 \end{cases}$$

Se o valor de x for maior ou igual a 10, então o cálculo resultará em $\frac{5x+3}{\sqrt{x^2-16}}$. Se o valor de x for menor que 10, então o cálculo a ser feito será $\frac{x}{\log(x^3)}$.

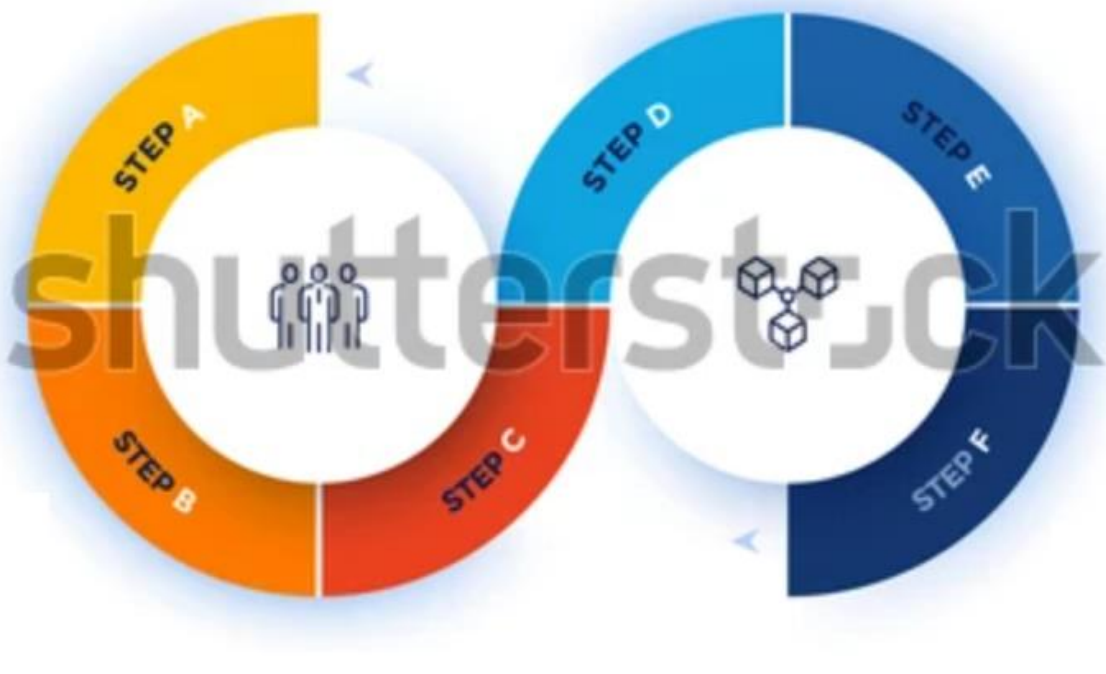
Exercício 08: Frequentemente os sistemas computacionais precisam lidar com datas, tais como data de cadastro, data de uma compra realizada, data de nascimento de um cliente, entre outras datas. As datas são formadas por dia, mês e ano. Implemente um algoritmo que faça a leitura de um dia, do mês e do ano e

apresente uma mensagem na tela informando se a data é válida ou não. DICA: verificar se o mês está entre 1 e 12 e verificar se o dia é um número válido para àquele mês (verificar se o ano é bissexto).

Exercício 09: Um algoritmo deve ser desenvolvido, de forma que o usuário informe o nome de uma pessoa, a idade desta pessoa e o sexo desta pessoa (considere *true* para masculino e *false* para feminino). O sistema deve informar se a pessoa pode ser aceita em um determinado grupo ou não. Para que a pessoa possa ser aceita no grupo, considere:

- Mulher com mais de 30 anos
- Homem com mais de 25 anos
- Homens ou mulheres com idades entre 10 e 15 anos

Para todos os outros casos, a pessoa não será aceita no grupo.



4. Estruturas de Repetição

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Compreender como funciona uma estrutura de repetição;
- Compreender como funciona uma estrutura de repetição com o operador “while”;
- Compreender como funciona uma estrutura de repetição com o operador “for”;
- Utilizar estruturas de decisão e controle com “while” e “for”;
- Criar estruturas de repetição contadas e controladas;
- Utilizar variáveis de controle;
- Resolver exercícios práticos.

O conteúdo trabalhado neste capítulo aborda:

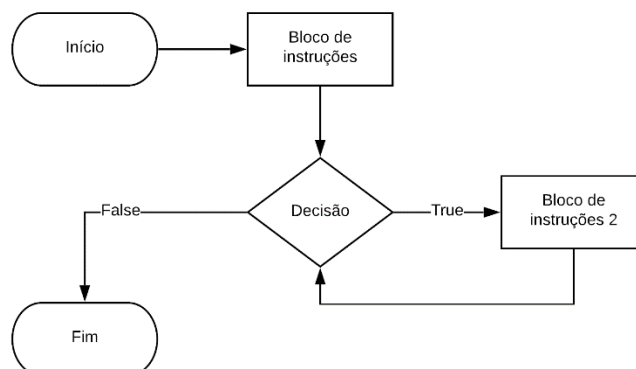
- Estrutura de repetição contada
- Estruturas de repetição com condições

Repetição em códigos

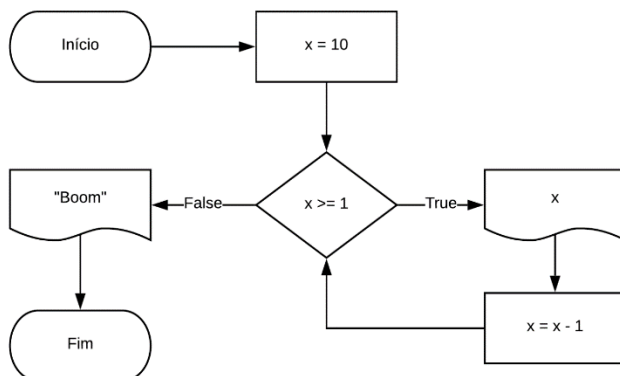
Laços de repetições: podemos criar uma estrutura para que as tarefas sejam executadas **repetidamente**. As repetições podem ser executadas eternamente ou podem se encerrar quando o objetivo definido para sua conclusão seja atingido.

Para que uma repetição ocorra, deve ocorrer um teste para verificação de condição. Neste teste, quando a condição for verdadeira (valor lógico True), então o código de um bloco de instruções é executado.

A Figura abaixo apresenta um exemplo de fluxograma que retrata um algoritmo com laço de repetição. Diferente do “if” que tem seu bloco de instruções executado uma vez só, um laço de repetição executa aquele bloco de instruções repetidamente até que sua condição se torne falsa. Em certos casos, é necessário alterar o valor da variável que está presente na condição.



A figura abaixo ilustra um fluxograma de um algoritmo que faz uma contagem regressiva. Primeiro é definido um valor inicial para “x” e enquanto “x” tem um valor maior ou igual a 1, o valor dele é apresentado na tela e depois é diminuído “1”. Esse processo se repete até que a condição do laço de repetição se torne falsa e, assim, o programa passa a executar os demais blocos de instruções presentes fora do laço de repetição. E ao executar o programa temos os resultados ao lado.



```
PS C:\Users\zotte\Downloads> python .\teste.py
10
9
8
7
6
5
4
3
2
1
Boom
PS C:\Users\zotte\Downloads> |
```

O código do exemplo acima, em Python, poderia ser:

```
x = 10
while(x >= 0):
    print(x)
    x = x - 1
print("Boom")
```

COMANDO: *for i in range(inicio, fim)*

Existem, basicamente, dois comandos para trabalhar com laços de repetição: o “for” e o “while”, que veremos a seguir.

Quando utilizamos o “for” para criar um laço de repetição, precisamos fornecer três coisas:

1. uma variável: `i`
2. um valor inicial para o “range”: `1`
3. um valor final para o “range”: `10`

Nesse caso, o “range” é uma função que cria uma sequência numérica para auxiliar o “for”. O “range” cria uma sequência de 1 a 9 e **não** de 1 a 10. Sempre é criada uma sequência em que o valor final dela não inclui o valor final definido na função.

O “for” é indicado para aqueles problemas que devem ser executados “n” vezes e você sabe qual o valor desse “n”. Por exemplo: fazer uma contagem de 1 a 10; fazer a soma dos números entre 1 e 100; fazer a tabuada do 1 ao 10 para o 5.

Exemplo 1: laço de repetição que conta de 1 a 9.

```
for i in range(1, 10):  
    print ("O valor de i é", i)
```

```
PS C:\Users\zotte\Downloads> python .\teste.py  
O valor de i é 1  
O valor de i é 2  
O valor de i é 3  
O valor de i é 4  
O valor de i é 5  
O valor de i é 6  
O valor de i é 7  
O valor de i é 8  
O valor de i é 9  
PS C:\Users\zotte\Downloads> |
```

Resultado:

Exemplo 2: Fazer a tabuada de algum número informado pelo usuário:

```
num = int(input("Digite um número para tabuada:"))  
for i in range(1, 10):  
    print (num, "x", i, "=", num*i)
```

```
PS C:\Users\zotte\Downloads> python .\teste.py  
Digite um número para tabuada:7  
7 x 1 = 7  
7 x 2 = 14  
7 x 3 = 21  
7 x 4 = 28  
7 x 5 = 35  
7 x 6 = 42  
7 x 7 = 49  
7 x 8 = 56  
7 x 9 = 63  
PS C:\Users\zotte\Downloads> |
```

Resultado:

COMANDO: *while*(condição)

Usamos o comando **while** para definir um bloco de repetição. Sempre precisamos fornecer a condição (entre parênteses, de preferência) que fará o teste para verificar quando a repetição deve ser finalizada.

Exemplo 1: laço de repetição que entra em looping, nunca será finalizado.

```
print("Programa que utiliza repetição.")
while(True):
    print("Devo prestar atenção na aula!")
print("Após sair do while(), o código continua aqui.")
```

Exemplo 2: laço de repetição “contado”. Estes laços são usados para controlar quantas vezes um bloco de instruções deverá ser executado. Precisa de: (1) uma inicialização, geralmente representado por uma variável que será usada como contador; (2) uma condição que é colocada no laço while; e (3) um incremento para aumentar/diminuir o valor do contador; geralmente colocado na última linha do bloco do while.

```
print("Programa que utiliza repetição.")
cont = 0
while(cont < 10):
    print("Vamos contar: ", cont)
    cont = cont + 1
print("Fim do programa")
```

Exemplo 3: exibir menus ao usuário. Podemos utilizar o while para exibir menus ao usuário. Desta forma, o sistema pode ser construído de forma “navegável”, saindo apenas quando o usuário escolher a opção apropriada.

```
opcao = 0
while (opcao != 5):
    print("Sistema: cadastro de clientes")
    print("1: cadastrar")
    print("2: alterar")
    print("3: excluir")
    print("4: consultar")
    print("5: sair do sistema")

    opcao = int(input("Digite uma opção: "))

    if (opcao == 1):
        print("Escolheu cadastrar cliente")
    elif (opcao == 2):
        print("Escolheu alterar dados")
    elif (opcao == 3):
        print("Escolheu excluir dados")
    elif (opcao == 4):
        print("Escolheu consultar dados")
    elif (opcao == 5):
        print("Escolheu sair do sistema")
    else:
        print("Opção inválida. Tente novamente.")
    print("\n\n")
print("O restante do programa continua aqui.")
```


Exemplo 4: forçar o usuário a digitar um número válido. Garantir que o valor informado seja válido para a execução do programa.

```
numero = int(input("Informe um número inteiro positivo"))
while(numero < 0):
    numero = int(input("Eu já disse, informe um número inteiro positivo"))
print("OK, o número informado é positivo: ", numero)
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Após resolver os exercícios, aguarde um dia e tente resolvê-los novamente você mesmo, mas agora sem utilizar as respostas e dicas como consulta. Se conseguiu resolver todos os exercícios sozinho, então você está aprendendo o conteúdo!

Usando o comando **while**

Exercício 01: Faça algoritmos em Python, utilizando o *while*, que:

- Apresente na tela os números de 1 a 100.
- Faça a soma dos números de 1 a 100 e apresente somente o resultado.
- Apresente na tela somente os números pares entre 50 e 100.
- Apresente na tela somente os números ímpares entre 1 e 50.
- Apresente na tela somente a soma dos números pares entre 1 e 100.
- Apresente na tela os números de X a Y (semelhante aos anteriores, porém peça para o usuário informar os valores de X e de Y em vez de fazer com 1 e 100).
- Faça a soma dos números de X a Y e apresente somente o resultado.
- Apresente na tela somente os números ímpares entre X e Y.

Exercício 02: Na matemática, o fatorial de um número natural n , representado por $n!$, é o produto de todos os inteiros positivos menores ou iguais a n . Por exemplo: o fatorial de 5 é representado por $5!$ que é igual a $5 \times 4 \times 3 \times 2 \times 1$. Faça um programa que peça um número para o usuário e apresente na tela seu fatorial.

Exercício 03: Faça um programa que mostre o menu a seguir, receba a opção do usuário e os dados necessários para executar cada operação. O programa será executado repetidamente até que o usuário passe o número informado para sair do programa (opção). Se o usuário informar uma opção inválida, apresente uma mensagem informando isso.

```
===== Menu Principal =====
1. Par ou ímpar?
2. Potência X
3. Raiz quadrada
4. Sair
```

Exercício 04: O Sr. Manoel Joaquim expandiu seus negócios para além dos negócios de 1,99 e agora possui uma loja de conveniências. Faça um programa que implemente uma caixa registradora. O programa deverá receber um número desconhecido de valores referentes aos preços das mercadorias. O valor "0" (zero) deve ser informado pelo operador para indicar o final da compra. O programa deve então mostrar o total da compra e perguntar o valor em dinheiro que o cliente forneceu, para então calcular e mostrar o valor do troco. A saída deve ser conforme o exemplo abaixo:

```
Lojas Tabajara
Produto 1: R$ 2.20
Produto 2: R$ 5.80
Produto 3: R$ 10.00
Produto 4: R$ 0
Total: R$ 18.00
Dinheiro: R$ 20.00
Troco: R$ 2.00
```

Usando o comando **for**

Exercício 05: Faça um programa em Python utilizando o *for* (um programa pra cada um), que:

- Apresente os números de 1 a 100 (um por linha).
- Apresente os números de 100 a 1 (um por linha).
- Apresente os números pares de 1 a 100 (um por linha).

- d) Apresente os números ímpares de 1 a 100 (um por linha).
- e) Faça a soma dos números de 1 a 100 e ao final mostre apenas a soma total.
- f) Faça a soma dos números de X a Y (informados pelo usuário), desde que X seja menor que Y, e apresente o valor total (semelhante ao anterior).
- g) Faça a multiplicação dos números de 1 a j (fatorial) e mostre o resultado final. Exemplo: Se j = 5 você deve calcular $1 * 2 * 3 * 4 * 5 = 120$

Exercício 06: Faça um programa que leia 5 números e informe apenas o maior número.

Exercício 07: Faça um programa que leia 5 números e informe a soma e a média dos números.

Exercício 08: Faça um programa que imprima na tela apenas os números ímpares entre 1 e 50.

Exercício 09: O Sr. Manoel Joaquim possui uma grande loja de artigos de R\$ 1,99, com cerca de 10 caixas. Para agilizar o cálculo de quanto cada cliente deve pagar ele desenvolveu uma tabela que contém o número de itens que o cliente comprou e ao lado o valor da conta. Desta forma a atendente do caixa precisa apenas contar quantos itens o cliente está levando e olhar na tabela de preços. Você foi contratado para desenvolver o programa que monta esta tabela de preços, que conterá os preços de 1 até 50 produtos, conforme o exemplo abaixo:

```
Lojas Quase Dois - Tabela de preços
1 - R$ 1.99
2 - R$ 3.98
...
50 - R$ 99.50
```

Exercício 10: Utilizando o laço de repetição *for*, faça um programa que apresente as tabuadas do 1 ao 10 para um número informado pelo usuário.

Exercício 11: Utilizando o laço de repetição *for*, faça um programa que apresente as tabuadas do X a Y para um número

Usando o comando **for** ou **while**

Exercício 12: Departamento Estadual de Meteorologia lhe contratou para desenvolver um programa utilizando laço de repetição (*for* ou *while*) que leia 10 temperaturas para calcular e informar na tela:

- a menor temperatura
- a maior temperatura
- a média das temperaturas

Exercício 13: Considerando a fórmula abaixo para calcular o valor de H, faça um programa que peça ao usuário qual o termo final (N) e calcule o valor de H:

$$H = 1 + 1/2 + 1/3 + 1/4 + \dots + 1/N$$

Exercício 14: Faça um programa para calcular descontos e juros. Para isso, apresente o menu abaixo para o usuário e sempre que ele escolher uma opção, peça novamente qual o valor a ser utilizado na operação. O programa só deverá ser finalizado quando a opção 6 for informada.

```
===== Descontos e Juros =====
1. Desconto de 10%
2. Desconto de 20%
3. Desconto de 50%
4. Juros de 2%
5. Juros de 5%
6. Sair
```



6. Estrutura de Dados Homogênea Unidimensional

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Compreender como funciona uma estrutura de dados homogênea;
- Aplicação de estruturas de repetição para preenchimento das estruturas de dados homogêneas, bem como exibição dos seus valores e processamento dos dados;
- Compreender a importância e o uso dos índices;
- Resolver exercícios práticos.

O conteúdo trabalhado neste capítulo aborda:

- Declaração de arrays unidimensionais (vetores)
- Preencher vetores com valores aleatórios e com valores fornecidos pelo usuário
- Percorrer o vetor para exibição de valores, preenchimento de valores e processamento dos dados armazenados no vetor

Arrays Unidimensionais (Vetores)

Iniciaremos o estudo das estruturas de dados básicas. Iniciaremos pela Estrutura de Dados Homogênea Unidimensional, também conhecidas como **Arrays Unidimensionais**, ou, mais popularmente como **Vetores**. Na linguagem Python, estas estruturas são também conhecidas como **Listas**.

Um vetor consiste em um “conjunto de dados”, ou seja, temos uma única variável e dentro desta variável podemos ter diversos valores armazenados. Observe que uma variável primitiva (string, int, float, boolean, etc.) armazena apenas um único valor em um determinado instante ($x = 10$ faz com que a variável x armazene o valor inteiro 10). Já um vetor é uma única variável que pode armazenar mais de um valor.

Os vetores são estruturas de dados homogêneas visto que todos os valores contidos no vetor pertencem a um mesmo tipo de dado. Por exemplo: se criarmos um vetor de números inteiros, então este vetor conterá apenas números inteiros. Se o vetor for de strings, conterá vários valores String. Caso o vetor seja booleano, então todos os valores neste vetor devem ser booleanos. Os vetores também são unidimensionais, pois possuem apenas uma única dimensão. Entenderemos melhor o conceito de dimensões quando estudarmos as estruturas bidimensionais (conhecidas como **matrizes**) ou n-dimensionais.

Para criarmos um vetor, precisamos informar quantos valores terão dentro da variável. A sintaxe para criar um vetor é:

```
vetor1 = [0] * tamanho
```

Na linha acima, uma variável chamada **vetor1** foi criada. Dentro desta variável haverá **tamanho** valores. Todas as posições destes valores dentro da variável serão inicializadas com o valor 0. Por exemplo:

```
vetor1 = [0] * 8
```

Cria uma variável chamada **vetor1** e esta variável será configurada com a seguinte estrutura;

vetor1	0	0	0	0	0	0	0	0
---------------	---	---	---	---	---	---	---	---

Observe que dentro da variável **vetor1** existem oito valores. Por padrão, todos foram inicializados como 0, pois na sua criação, foi dito que seriam 8 números [0].

Se digitarmos os comandos:

```
vetor1 = [0] * 8
print(vetor1)
```

podemos observar que a saída apresentada será: [0, 0, 0, 0, 0, 0, 0, 0].

Um conceito muito importante ao trabalharmos com vetores é o de **índice do vetor**. O índice corresponde a um *endereço* que indica qual é a *posição* dentro do vetor que desejamos acessar. O valor do índice deve vir dentro dos colchetes. Por exemplo:

```
vetor1[3] = 10
```

Com a instrução acima, estamos acessando a variável **vetor1** na posição de índice 3, e estamos guardando o número 10 dentro desta posição na variável.

Outro detalhe muito importante a saber sobre os índices: o **primeiro índice** de um vetor (a posição do primeiro número dentro da variável) é o número 0. Portanto, a posição do **último número** dentro do vetor possui índice $\langle \text{tamanho} \rangle - 1$. Por exemplo, se criarmos um vetor com tamanho 8, então o primeiro índice será o 0 e o último índice será o 7. Desta forma, o vetor **vetor1** possui a seguinte estrutura:

vetor1

Valores:								
Índices:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Considere o seguinte trecho de algoritmo:

```
idades = [0] * 10  
idades[3] = 5  
idades[8] = 50  
idades[0] = 37
```

Inicialmente o vetor **idades** foi criado e este vetor contém 10 posições para guardar valores. Em seguida, o número 5 foi atribuído à posição [3] do vetor.

idades

Valores:	0	0	0	5	0	0	0	0	0	0
Índices:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Em seguida, o número 50 foi atribuído ao vetor em sua posição de índice [8]:

idades

Valores:	0	0	0	5	0	0	0	0	50	0
Índices:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Por fim, o vetor **idades**, em seu índice [0], guardará o valor 37:

idades

Valores:	37	0	0	5	0	0	0	0	50	0
Índices:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Se digitarmos o comando:

```
print(idades)
```

podemos observar que a saída apresentada será: [37, 0, 0, 5, 0, 0, 0, 0, 0, 50, 0].

Podemos apresentar ao usuário os valores do vetor normalmente como fazemos com qualquer variável. Basta fornecer o índice do vetor para acessarmos o valor que desejamos exibir:

```
print( idades[8] )
```

Também podemos fazer cálculos matemáticos com os valores dos vetores igual fazemos com variáveis primitivas. Por exemplo, o cálculo $5 + 50 + 2$ será feito e o valor 57 será atribuído à variável **soma**:

```
soma = idades[3] + idades[8] + 2;
```

Podemos também digitar valores para as posições nos vetores, exatamente igual fazemos com as variáveis primitivas. Só precisamos lembrar de colocar o índice onde o valor digitado será salvo. No exemplo a seguir, o

usuário pode digitar um valor e ele será convertido para número inteiro para, então, ser salvo na variável `idades` na posição de índice [4].

```
idades[4] = int(input("Informe um valor: "))
```

Percorrendo os vetores

Tendo-se a seguinte instrução:

```
nomes = [""] * 10
```

Imagine que precisamos digitar todos os valores do vetor **nomes**. Desta forma, temos o seguinte conjunto de instruções:

```
nomes[0] = str(input("Informe um nome: "))
nomes[1] = str(input("Informe um nome: "))
nomes[2] = str(input("Informe um nome: "))
nomes[3] = str(input("Informe um nome: "))
nomes[4] = str(input("Informe um nome: "))
nomes[5] = str(input("Informe um nome: "))
nomes[6] = str(input("Informe um nome: "))
nomes[7] = str(input("Informe um nome: "))
nomes[8] = str(input("Informe um nome: "))
nomes[9] = str(input("Informe um nome: "))
```

Agora imagine que desejamos exibir na tela todos os valores do vetor. Teríamos o seguinte conjunto de instruções:


```
print(nomes[0])
print(nomes[1])
print(nomes[2])
print(nomes[3])
print(nomes[4])
print(nomes[5])
print(nomes[6])
print(nomes[7])
print(nomes[8])
print(nomes[9])
```

Observe que todas as linhas de entrada de dados, bem como as linhas de saída de dados são parecidas. A única diferença está no valor dentro dos colchetes, indicando a posição em que os valores estão sendo acessados dentro do nosso vetor.


Sempre que precisarmos percorrer nosso vetor, tanto para digitar dados para todas as posições, como para apresentar na tela todos os valores deste vetor, ou mesmo quando precisarmos fazer cálculos envolvendo os valores deste vetor, podemos melhorar o código utilizando uma estrutura de repetição.

Geralmente utilizamos o comando **for** para percorrer todos os valores de um vetor. A variável utilizada para contar as repetições é colocada **dentro dos colchetes**. Desta forma, conseguimos acessar, sequencialmente, todos os valores do vetor. Por exemplo:

```
for cont in range(0, 10):  
    nomes[cont] = str(input("Informe um nome: "))
```



```
for cont in range(0, 10):  
    print("Valor: ", nomes[cont])
```



Com o uso da estrutura de repetição, o código é reduzido de tamanho, facilitando o acesso a todos os dados contidos em nosso vetor.

Uma forma interessante para se utilizar a estrutura de repetição para percorrer todas as posições de um vetor é conseguir obter o tamanho do vetor (número de valores presentes em um vetor). Para conseguirmos saber qual é o tamanho de um vetor, podemos usar a função **len(vetor)**. Ao utilizar esta função, estamos obtendo qual o tamanho foi fornecido a este vetor no momento de sua criação. Por exemplo: podemos criar um vetor de números inteiros chamado **idades** e que possui espaço para guardarmos 15 valores.

```
idades = [0] * 15
```

Se durante a execução do algoritmo precisamos “relembrar” qual foi o tamanho que demos ao vetor no momento em que ele foi criado (15), podemos utilizar **len(idades)**. Por exemplo:

```
for cont in range(0, len(idades)):  
    print("Valor: ", idades[cont])
```

Ao utilizar a função **len(vetor)** nos laços de repetição, caso o algoritmo mude e o vetor **idades** não tenha mais 15 endereços, mas sim 200, não precisamos modificar o restante do nosso algoritmo, visto que a repetição continua indo até o tamanho máximo de endereços possíveis:

```
nomes = ["" ] * 10
```

```
{ for cont in range(0, len(nomes)):  
    nomes[cont] = str(input("Informe um nome: "))
```

Vetor nomes com
10 posições

Mesmo laço de
repetição para
diferentes
tamanhos de
vetor

```
nomes = ["" ] * 200
```

```
{ for cont in range(0, len(nomes)):  
    nomes[cont] = str(input("Informe um nome: "))
```

Novo vetor nomes
com 200 posições

Criando arrays já com valores **inicializados**

Quando já sabemos quais são os dados que o vetor conterá, podemos criar a variável do vetor já preenchido com estes dados. Para isso, podemos declarar a variável de vetor e atribuir os valores, na sequência em que

os valores estarão dentro dos índices do vetor. Como é um conjunto de valores (uma lista de valores), estes dados devem estar entre colchetes e serem separados por vírgula.

Por exemplo:

```
vetor = [45, 23, 80, 0, -8]

print(vetor)

print("Tamanho do vetor: " , len(vetor))
```

Inicialmente a variável **vetor** foi criada. Esta variável contém uma lista de valores (repare que foi criada usando colchetes) e, dentro dos colchetes, uma lista de valores foi informada.

vetor

Valores:	45	23	80	0	-8
Índices:	[0]	[1]	[2]	[3]	[4]

Outros exemplos:

a) Criar um vetor já preenchido com valores String:

```
nomes = ["Rafael", "Ayslan", "Frank", "Willian", "André", "Marcelo"]
```

O vetor criado foi:

nomes

Valores:	"Rafael"	"Ayslan"	"Frank"	"Willian"	"André"	"Marcelo"
Índices:	[0]	[1]	[2]	[3]	[4]	[5]

b) Criar um vetor já preenchido com valores booleanos:

```
luzesLigadas = [True, False, False, False, True]
```

O vetor criado foi:

luzesLigadas

Valores:	True	False	False	False	True
Índices:	[0]	[1]	[2]	[3]	[4]

c) Criar um vetor já preenchido com valores ponto-flutuantes:

```
precos = [45.98, 23.00, 80.9, 88652.09, 8.12]
```

O vetor criado foi:

precos

Valores:	45.98	23.00	80.9	88652.09	8.12
Índices:	[0]	[1]	[2]	[3]	[4]

Neste momento, podemos começar a resolver problemas que juntam as estruturas de decisão (if / elif / else) e repetição (for / while) para processar os dados que estarão armazenados em nossos vetores. Os exercícios exigirão um raciocínio lógico mais apurado. Então, não esqueçam de: **PLANEJAMENTO!** Não saiam resolvendo os exercícios de imediato. Raciocinem sobre o que precisa ser feito (passo a passo) para resolver os problemas, quais variáveis adicionais serão necessárias para guardar os valores, e por aí vai.

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Exercício 01: Você precisa desenvolver um aplicativo que seja capaz de guardar as notas de 10 alunos de uma turma. As notas vão de 0,0 a 10,0. **Crie o vetor** que seja capaz de representar esta situação.

Exercício 02: Declare um vetor que seja capaz de armazenar o nome de 85 alunos de uma escola.

Exercício 03: Imagine um sistema computacional que precisa controlar o status dos ar-condicionados de uma empresa. A empresa contém 8 salas com ar-condicionado. O status é representado por `true` para quando o ar-condicionado de uma sala está ligado e `false` quando está desligado. Crie um vetor em que cada posição represente uma sala da empresa e que armazene o status de cada ar-condicionado.

Exercício 04: Considere o seguinte vetor:

`precos = [0.0] * 5`

- Qual é a instrução para armazenar o preço R\$ 1500,00 na primeira posição do vetor?
- Qual é a instrução para armazenar o preço R\$ 8,99 na última posição do vetor?
- Qual é a instrução para exibir na tela o preço que está na terceira posição do vetor?

Exercício 05: Implemente um algoritmo que seja capaz de solicitar por 10 nomes ao usuário. Em seguida, o algoritmo deve mostrar na tela os 10 nomes informados. Armazene os 10 nomes em uma única variável.

Exercício 06: Considere o seguinte vetor:

`alturasAlunos = [0.0] * 40`

Como fazer para exibir na tela o número de valores que a variável `alturasAlunos` possui?

Exercício 07: Implemente um algoritmo que seja capaz de armazenar o nome e a idade de 10 alunos de uma turma. Solicite os nomes e as idades de cada aluno ao usuário e, em seguida, apresente na tela um relatório contendo o nome e a idade informada.

DICA: um vetor para os nomes e um vetor para as idades.

DICA: o nome no índice [0] possui a idade no índice [0].

DICA: utilize um laço de repetição para solicitar o nome e a idade e um laço de repetição para apresentar os dados.

DICA: a saída dos dados deve ser um relatório contendo a mensagem

“O aluno NOME[posição] possui IDADE[posição] anos”

Exercício 08: Implemente um algoritmo que seja capaz de armazenar em uma variável 10 números inteiros. Estes números deverão ser sorteados pelo computador aleatoriamente, no intervalo entre 0 e 100. Após obter todos os números, apresente na tela todos os números sorteados.

Exercício 09: Considere o seguinte vetor:

`precos = [0.0] * 5`

Implemente um algoritmo que tenha este vetor, solicite ao usuário por todos os preços (todos os valores deste vetor), e, ao final, apresente na tela a soma de todos os valores contidos neste vetor.

Exercício 10: Implemente um algoritmo que contenha um vetor com 10 números inteiros. O algoritmo deve solicitar ao usuário pelo preenchimento deste vetor. Após o preenchimento, o algoritmo deve encontrar qual foi o maior número informado pelo usuário e apresentá-lo na tela.

Exercício 11: Seja um vetor declarado e inicializado como segue:

precos = [8, 8.9, 7.89, 9, 1, 0.89, 11]

Implemente um algoritmo que contenha este vetor. Em seguida, o algoritmo deve encontrar qual é o menor número dentro do vetor e apresentar ao usuário qual é o índice (a posição dentro do vetor) em que este menor número se encontra.

Exercício 12: Crie um algoritmo que contenha um vetor de números inteiros contendo 50 posições. O algoritmo deve sortear números aleatórios para preencher este vetor, com valores variando entre 0 e 500. Após preencher o vetor, o algoritmo deve calcular e apresentar ao usuário quantos números pares foram sorteados e estão presentes neste vetor.

Exercício 13: Um algoritmo deve conter um vetor de inteiros com 10 posições que é preenchido com valores aleatórios entre 0 e 10. Em seguida, o algoritmo deve solicitar ao usuário por um número. Por fim, o algoritmo deve emitir uma mensagem informando se o número digitado está presente ou não neste vetor.

Exercício 14: Implemente um algoritmo que contenha dois vetores. Um armazenará os nomes dos alunos de uma turma. Outro armazenará as notas obtidas por cada aluno em uma dada disciplina.

Observe que o aluno [0] tirou a nota armazenada em [0] e assim por diante.

O algoritmo deve ser capaz de solicitar os nomes e as notas dos alunos ao usuário. Em seguida, o algoritmo deve apresentar um relatório com as mensagens da seguinte maneira:

Aluno 1 [0]	100 [0]
Aluno 2 [1]	90 [1]
Aluno 3 [2]	45 [2]
...	
Aluno n [n-1]	82 [n-1]

Exercício 15: Um programa de computador deve ser capaz de solicitar ao usuário pelo nome de 10 cidades e também a população de cada uma destas cidades. Implemente um programa que crie os vetores e solicite os dados ao usuário. Por exemplo:

cidade[0] = "Paranavaí"	população[0] = 89500
cidade[1] = "Terra Rica"	população[1] = 15256
cidade[2] = "Paranacity"	população[2] = 11361
....

Após o usuário preencher os valores dos dois vetores, o algoritmo deve ser capaz de apresentar na tela qual é a maior cidade informada pelo usuário (cidade que possui a maior população).

Exercício 16: Faça um programa que solicite ao usuário pelo tamanho de um vetor de números inteiros. Após o usuário informar o tamanho, o programa deve criar o vetor no tamanho informado. Em seguida, o usuário deverá digitar todos os dados deste vetor.

Em seguida, o algoritmo deve exibir na tela todos os dados informados pelo usuário a este vetor.

Por fim, o algoritmo deve exibir na tela todos os dados informados pelo usuário, mas em ordem inversa (do último valor até o primeiro valor).

Exercício 17: Um programa de computador deve ser desenvolvido para obter o preço de vinte televisores que um hipermercado comercializa. Estes valores devem ser armazenados em um vetor com os dados fornecidos pelo usuário.

Em seguida, o algoritmo deve informar ao usuário qual é a média dos preços de todos os televisores contidos no vetor, assim como o maior preço e o menor preço.

Exercício 18: Implemente um algoritmo que contenha um vetor com 20 posições para números inteiros. O algoritmo deve preencher este vetor com valores aleatórios entre 0 e 500.

Por fim, apresente na tela:

- Quantos números sorteados são ímpares

- Qual foi o maior número sorteado
- Qual foi o menor número sorteado
- Qual foi a média de todos os números sorteados



7. Ordenação de vetores

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Criação de vetores e manipulação de vetores
- Uso de estruturas de repetição e decisão para manipulação de vetores
- Entendimento da lógica envolvida nos algoritmos de ordenação de vetores
- Resolver exercícios práticos.

O conteúdo trabalhado neste capítulo aborda:

- Ordenação de vetores utilizando os algoritmos Bolha, Inserção e Seleção
- Entender a diferença e o desempenho de cada algoritmo de ordenação

Ordenação de Vetores

Os computadores lidam o tempo todo com muitos dados. O processamento de dados consiste em ler ou obter um conjunto de dados, realizar uma série de cálculos sobre estes dados e fornecer algum tipo de resultado com estes cálculos que seja útil para alguma finalidade.

Neste bloco de atividades, iremos estudar alguns algoritmos que fazem o “processamento de dados” de uma lista de dados (**vetores**). Frequentemente existem situações onde um conjunto de dados precisam estar ordenados (alfabeticamente, do maior para o menor, etc.). Imagine um sistema computacional que precisa exibir os nomes dos clientes em ordem alfabética, ou uma lista que exibe os conceitos dos alunos da nota máxima até a nota mínima, ou mesmo um sistema de vendas online em que os produtos podem ser listados do menor preço para o maior preço. Para conseguir fazer esta **ordenação** dos dados, existem diversos algoritmos diferentes que podem ser usados.

A ordenação dos dados é o assunto a ser estudado neste bloco de atividades. Considere um conjunto de dados armazenados em um vetor, tal como o representado abaixo:

10	8	6	15	18	20	1
----	---	---	----	----	----	---

Para o processamento destes dados, imagine que estes dados precisam ser organizados em ordem numérica crescente, ou seja, do menor número para o maior número. Após esta ordenação, o vetor teria a seguinte configuração:

1	6	8	10	15	18	20
---	---	---	----	----	----	----

A função dos **algoritmos de ordenação** é justamente esta: a partir de um vetor preenchido com dados, colocar estes dados em ordem seguindo algum critério (do menor para o maior valor, do maior para o menor, etc.).

Iremos estudar sobre três algoritmos básicos de ordenação de vetor:

- Ordenação de vetor utilizando o método **bolha**;
- Ordenação de vetor utilizando o método de **inserção**;
- Ordenação de vetor utilizando o método de **seleção**.

Bora estudar? Essa matéria é bem legal.

Preliminares

Vocês se lembram quando começamos falar sobre variáveis? Provavelmente você resolveu um algoritmo que dizia o seguinte: Tendo-se duas variáveis A e B, faça com que a variável A passe a valer o valor de B, e B passe a valer o valor de A.

Esta *troca* de valores entre as variáveis é o princípio básico para os algoritmos de ordenação. Inicialmente, os alunos fazem o seguinte:

```
a = 10
b = 20
print("A:", a, "e B:", b)

a = b
b = a
print("A:", a, "e B:", b)
```

O que ocorre é o seguinte: inicialmente temos que a variável **a** vale 10 e a variável **b** vale 20 e a mensagem “A: 10 e b: 20” será exibida na tela. Em seguida, com a instrução **a = b** modificamos o valor da variável **a** fazendo

com que **a = 20**. Até aí tudo bem, o valor de **a** recebeu o valor de **b** e o algoritmo está correto. Mas observe a próxima instrução: **b = a**. Aqui, como o valor da linha anterior disse que o valor de **a** é 20, então temos que **b = 20**. Logo, não fizemos a troca dos valores. O valor da variável **a** foi perdido.

Para resolver este problema, precisamos de uma terceira variável, uma variável *auxiliar*. Por que esta variável é necessária? Para guardar um valor que será perdido no algoritmo. Se guardarmos este valor em algum lugar, então podemos recuperá-lo depois quando for necessário. Observe como o algoritmo funcionaria:

```
a = 10
b = 20
print("A:", a, "e B:", b)

aux = a
a = b
b = aux
print("A:", a, "e B:", b)
```

O que ocorreu é o seguinte: inicialmente temos que a variável **a** vale 10 e a variável **b** vale 20 e a mensagem “A: 10 e b: 20” será exibida na tela. Em seguida, com a instrução **aux = a** dizemos que o valor de **a** será salvo na variável **aux**. Ou seja, neste momento temos o valor 10 tanto na variável **a** quanto na variável **aux**. Em seguida, temos que **a = b**, ou seja, **a = 20**. Entretanto, para concluir a troca, temos que **b = aux**, ou seja, desta vez dissemos que **b = 10** e, portanto, foi feita a troca dos valores. Inicialmente **a = 10** e **b = 20**. No fim do algoritmo, temos que **a = 20** e **b = 10**.

Algoritmo de Ordenação por **Bolha**

O algoritmo de ordenação por **bolha** utiliza justamente o método de troca de valores de duas variáveis. O que este algoritmo faz?

Ele compara dois valores adjacentes (um valor e o próximo valor) da lista. Por exemplo, considere um vetor chamado **vetor** criado e inicializado com os seguintes dados:

vetor = [10, 8, 6, 15, 18, 20, 1]

10	8	6	15	18	20	1
----	---	---	----	----	----	---

O primeiro valor do vetor é 10. Temos que **vetor[0] = 10**

O segundo valor do vetor é 8. Portanto, **vetor[1] = 8**

Precisamos fazer uma comparação entre estes dois valores. Ou seja, comparamos 10 e 8, ou, mais precisamente, **vetor[0]** e **vetor[1]**, visto que são valores adjacentes, vizinhos.

Observe que $10 > 8$. Logo, o primeiro valor é maior que o segundo valor. Se o vetor precisa ser ordenado do menor número para o maior número, então existe um problema aqui. É necessário fazer a troca destes valores. O resultado do vetor após a troca:

```
aux = vetor[0]
vetor[0] = vetor[1]
vetor[1] = aux
```

Após fazer esta troca, o vetor estará configurado da seguinte maneira:

8	10	6	15	18	20	1
---	----	---	----	----	----	---

Como no primeiro passo fizemos a comparação do primeiro valor com o segundo valor, o próximo passo é fazer a comparação do segundo valor com o terceiro valor. Observe que temos na segunda posição o valor 10

(vetor[1] = 10) e na terceira posição o valor 6 (vetor[2] = 6). Como $10 > 6$, então temos um novo problema. Os valores não estão do menor para o maior e precisam ser novamente trocados.

```
aux = vetor[1]
vetor[1] = vetor[2]
vetor[2] = aux
```

Após esta nova troca, o vetor ficará da seguinte maneira:

8	6	10	15	18	20	1
---	---	----	----	----	----	---

O processo continua fazendo a comparação agora do terceiro valor com o quarto valor. Temos que **vetor[2]=10** e que **vetor[3] = 15**. Se compararmos $10 > 15$, temos uma resposta *false*. Neste sentido, estes dois números não precisam ser trocados, visto que os números 10 e 15 já se encontram ordenados. Portanto, o vetor permanece da mesma forma:

8	6	10	15	18	20	1
---	---	----	----	----	----	---

Agora precisamos comparar o número 15 com o 18. Fazemos a pergunta: $15 > 18$? Como a resposta foi *false* novamente, então os números 15 e 18 já estão ordenados. O vetor resultante é o mesmo:

8	6	10	15	18	20	1
---	---	----	----	----	----	---

Novamente precisamos fazer o teste do valor 18 com o valor 20. Comparando $18 > 20$, obtemos *false* novamente como resposta. Portanto, o vetor, neste ponto, também não precisa ser modificado, visto que 18 e 20 já estão ordenados.

8	6	10	15	18	20	1
---	---	----	----	----	----	---

Por fim, temos que comparar **vetor[5]=20** com **vetor[6]=1**. Ao verificar se $20 > 1$, então o resultado foi *true* e, neste ponto, temos um problema, visto que os dois números não estão ordenados. Devemos fazer esta troca novamente:

```
aux = vetor[5]
vetor[5] = vetor[6]
vetor[6] = aux
```

Como resultado, o vetor terá a seguinte configuração:

8	6	10	15	18	1	20
---	---	----	----	----	---	----

Observe que o processo sempre faz a comparação de dois números consecutivos:

Primeiro com segundo. Se precisar trocar valores, troque.

Segundo com terceiro. Se precisar trocar valores, troque.

Terceiro com quarto. Se precisar trocar valores, troque.

E assim por diante.

Observe também que, ao comparar todos os valores, o vetor não está em ordem! Este processo de comparação deve ser executado novamente, desde o início.

Comparar $8 > 6$, precisamos trocar de lugar.

6	8	10	15	18	1	20
---	---	----	----	----	---	----

Comparar $8 > 10$, não trocamos os valores neste momento.

6	8	10	15	18	1	20
---	---	----	----	----	---	----

Comparamos $10 > 15$, não precisamos trocar os valores neste momento.

6	8	10	15	18	1	20
---	---	----	----	----	---	----

Comparamos $15 > 18$, não precisamos trocar os valores neste momento.

6	8	10	15	18	1	20
---	---	----	----	----	---	----

Comparamos $18 > 1$. Aqui temos um problema e, portanto, precisamos fazer a troca dos valores:

6	8	10	15	1	18	20
---	---	----	----	---	----	----

Para concluir, comparamos $18 > 20$. Aqui não precisamos trocar os valores. O vetor resultante é:

6	8	10	15	1	18	20
---	---	----	----	---	----	----

Observe neste momento que o vetor ainda não se encontra ordenado.

Neste caso, precisamos realizar todo o processo de comparação novamente.

6>8 não troca	6	8	10	15	1	18	20
8>10 não troca	6	8	10	15	1	18	20
10>15 não troca	6	8	10	15	1	18	20
15>1 troca	6	8	10	15	1	18	20
	6	8	10	1	15	18	20
15>18 não troca	6	8	10	1	15	18	20
18>20 não troca	6	8	10	1	15	18	20

Observando novamente o vetor, temos que ele ainda não se encontra ordenado. O processo é reiniciado novamente.

6>8 não troca	6	8	10	1	15	18	20
8>10 não troca	6	8	10	1	15	18	20
10>1 troca	6	8	10	1	15	18	20
	6	8	1	10	15	18	20
10>15 não troca	6	8	1	10	15	18	20
15>18 não troca	6	8	1	10	15	18	20
18>20 não troca	6	8	1	10	15	18	20

Observando novamente o vetor, temos que ele ainda não se encontra ordenado. O processo é reiniciado novamente.

6>8 não troca	6	8	1	10	15	18	20
8>1 troca	6	8	1	10	15	18	20

	6	1	8	10	15	18	20
8>10 não troca	6	1	8	10	15	18	20
10>15 não troca	6	1	8	10	15	18	20
15>18 não troca	6	1	8	10	15	18	20
18>20 não troca	6	1	8	10	15	18	20

Observando novamente o vetor, temos que ele ainda não se encontra ordenado. O processo é reiniciado novamente.

6>1 troca	6	1	8	10	15	18	20
	1	6	8	10	15	18	20
6>8 não troca	1	6	8	10	15	18	20
8>10 não troca	1	6	8	10	15	18	20
10>15 não troca	1	6	8	10	15	18	20
15>18 não troca	1	6	8	10	15	18	20
18>20 não troca	1	6	8	10	15	18	20

Observando novamente o vetor, temos que neste momento, sim. Agora o vetor já se encontra ordenado.

Observe o funcionamento do método de ordenação por bolha:

- Compara-se um valor com o próximo. Se precisar fazer a troca de posição, troque;
- Repita o processo anterior até que todos os valores tenham sido comparados;
- Reinicie o processo. Os dois itens anteriores deverão ser executados n-1 vezes, sendo n o tamanho do vetor. Ou seja, se o vetor tem tamanho 7, o processo de comparação deve ser executado 6 vezes.

Algoritmo de Ordenação por Inserção

O algoritmo de ordenação por inserção (*Insertion Sort*) funciona de forma análoga ao que fazemos quando estamos ordenando as cartas de um baralho. Neste algoritmo, cada valor é considerado um a um, e são inseridos em sua respectiva posição entre os valores já ordenados.

Usando a ordenação de cartas de baralho como exemplo, o processo de ordenação seria o seguinte:

- Lê uma carta
 - Se for a primeira carta lida, então ela já se encontra em seu devido lugar
 - Lê outra carta
- Encontra a posição desta nova carta (ou será antes da primeira carta ou depois)
- Lê a próxima carta e encontra a posição desta nova carta (será a primeira carta, a segunda carta ou a terceira carta?)

- Lê a próxima carta e encontra a sua posição
- Lê a próxima carta e encontra a sua posição
- O processo segue até que a última carta é lida e colocada em sua devida posição.

O algoritmo funciona da seguinte maneira:

Percorrer o vetor:

- 1) Da esquerda para a direita
- 2) Encontrou elemento for a de ordem
 - a. Posiciona o elemento em seu local
 - b. Leva os “demais” valores uma casa pra frente

Considere o seguinte vetor:

vetor = [10, 8, 6, 15, 18, 20, 1]

10	8	6	15	18	20	1
----	---	---	----	----	----	---

O primeiro valor, automaticamente, já se encontra em seu devido lugar:

10	8	6	15	18	20	1
----	---	---	----	----	----	---

A partir do segundo valor, o algoritmo deve encontrar a sua posição ideal. Para isso, seguem-se os seguintes passos:

- **Passo 1:** encontrar o lugar do número 8. Ele deve vir antes do 10.

8 → 10		6	15	18	20	1
--------	--	---	----	----	----	---

Coloca o número 8 no seu devido lugar e o número 10 é enviado uma casa para frente.

8	10	6	15	18	20	1
---	----	---	----	----	----	---

- **Passo 2:** encontra o lugar do número 6. Ele deve vir antes do número 8.

6 → 8	10		15	18	20	1
-------	----	--	----	----	----	---

Coloca o número 6 no seu devido lugar e os números 8 e 10 são enviados uma casa para frente.

6	8	10	15	18	20	1
---	---	----	----	----	----	---

- **Passo 3:** encontra o lugar do número 15. Ele já está em seu devido lugar.

6	8	10	15	18	20	1
---	---	----	----	----	----	---

- **Passo 4:** encontra o lugar do número 18. Ele já está em seu devido lugar.

6	8	10	15	18	20	1
---	---	----	----	----	----	---

- **Passo 5:** encontra o lugar do número 20. Ele já está em seu devido lugar.

6	8	10	15	18	20	1
---	---	----	----	----	----	---

- **Passo 6:** encontra o lugar do número 1. Ele deve vir antes do número 6.

1 → 6	8	10	15	18	20	
-------	---	----	----	----	----	--

Coloca o número 1 no seu devido lugar e os números 6, 8, 10, 15, 18 e 20 são enviados uma casa para frente.

1	6	8	10	15	18	20
---	---	---	----	----	----	----

Algoritmo de Ordenação por Seleção

O algoritmo de ordenação por seleção (*Selection Sort*) é considerado um dos algoritmos de seleção mais simples.

Este algoritmo funciona da seguinte maneira:

- O vetor é lido da Esquerda para a Direita
- Encontrar o menor valor do vetor
- Trocar de posição este menor valor do vetor com o primeiro valor do vetor
- Encontrar o segundo menor valor do vetor
- Trocar de posição este segundo menor valor do vetor com o segundo valor do vetor
- Encontrar o terceiro menor valor do vetor
- Trocar este terceiro menor valor com o valor que esta na terceira posição do vetor
- ... (continuar encontrando o quarto menor, quinto menor, etc.) ... até que todos os valores estejam em ordem.

Considere o seguinte vetor:

vetor = [10, 8, 6, 15, 18, 20, 1]

10	8	6	15	18	20	1
----	---	---	----	----	----	---

Passo 1: Encontrar o menor valor do vetor [10, 8, 6, 15, 18, 20, 1]. Este valor é o 1. Colocar ele na primeira posição do vetor, trocando os valores:

1	8	6	15	18	20	10
---	---	---	----	----	----	----

Passo 2: Encontrar o segundo menor valor do vetor (o menor valor entre a segunda e a última posições, ou seja, [8, 6, 15, 18, 20, 10]). Este valor é o 6. Colocar ele na segunda posição do vetor, trocando os valores:

1	6	8	15	18	20	10
---	---	---	----	----	----	----

Passo 3: Encontrar o terceiro menor valor do vetor (encontrar o menor valor entre a terceira e a última posições, ou seja, o menor entre [8, 15, 18, 20, 10]). Este valor é o 8. Como ele já se encontra em sua devida posição, não é necessário realizar a troca dele com ele mesmo!

Passo 4: Encontrar o quarto menor valor do vetor (ou seja, encontrar o menor valor da posição atual até o fim do vetor [15, 18, 20, 10]). O menor valor é o 10. Logo, precisamos trocar o valor 10 com o valor (15) da quarta posição.

1	6	8	10	18	20	15
---	---	---	----	----	----	----

Passo 5: Encontrar o quinto menor valor do vetor (encontrar o menor valor entre a quinta posição e a última posição do vetor, ou seja, o menor de [18, 20, 15]). O menor valor é o 15 e, portanto, devemos trocar ele de posição com o número 18 que está ocupando a casa do quinto número.

1	6	8	10	15	20	18
---	---	---	----	----	----	----

Passo 6: Encontrar o sexto menor valor do vetor (encontrar o menor valor entre a sexta posição e a última posição do vetor, ou seja, o menor de [20, 18]). O menor valor é o 18 e, portanto, devemos colocar ele em sua devida posição, trocando os valores.

1	6	8	10	15	18	20
---	---	---	----	----	----	----

Como só restou o sétimo valor, último valor do vetor, então ele não precisa ser verificado ou trocado com ninguém, pois já ocupa sua posição.

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Exercício 01: Utilizando o algoritmo de ordenação por Bolha (*Bubble Sort*) onde um *ciclo* de trocas corresponde a fazer a comparação de todos os valores do vetor com os seus adjacentes. Considere o seguinte vetor:

valores = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Considere ainda que o vetor “valores” precise ser ordenado de forma crescente. Considerando o “pior caso” (onde o vetor está completamente desordenado, na ordem inversa a que se deseja), quantos ciclos de troca serão necessários para ordenar o vetor valores?

Exercício 02: Utilizando o seguinte vetor:

precos = [1.99, 10.99, 10.50, 8.50, 7.49, 1599.45, 1400.00]

Apresente o “passo a passo” para a ordenação deste vetor utilizando o algoritmo Bubble Sort.

Exercício 03: Utilizando o algoritmo de ordenação por Inserção (*Insertion Sort*) onde um *passo* (ou *ciclo*) de trocas corresponde a encontrar o local exato de um valor e deslocar todos os próximos valores uma casa para frente. Considere o seguinte vetor:

vetor = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Considere ainda que o primeiro valor do vetor já se encontra em sua devida posição e o primeiro passo se inicia com a busca pela posição correta do segundo valor do vetor. Considerando o “pior caso” (onde o vetor está completamente desordenado, na ordem inversa a que se deseja), quantos passos serão necessários para ordenar o vetor valores?

Exercício 04: Utilizando o seguinte vetor:

precos = [1.99, 10.99, 10.50, 8.50, 7.49, 1599.45, 1400.00]

Apresente o “passo a passo” para a ordenação deste vetor utilizando o algoritmo Insertion Sort.

Exercício 05: Utilizando o algoritmo de ordenação por Seleção (*Selection Sort*) onde um *passo* de trocas corresponde a encontrar o menor valor e colocar em sua posição atual representada pelo passo, trocando os valores. Considere o seguinte vetor:

vetor = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Considerando o “pior caso” (onde o vetor está completamente desordenado, na ordem inversa a que se deseja), quantos passos serão necessários para ordenar o vetor valores?

Exercício 06: Utilizando o seguinte vetor:

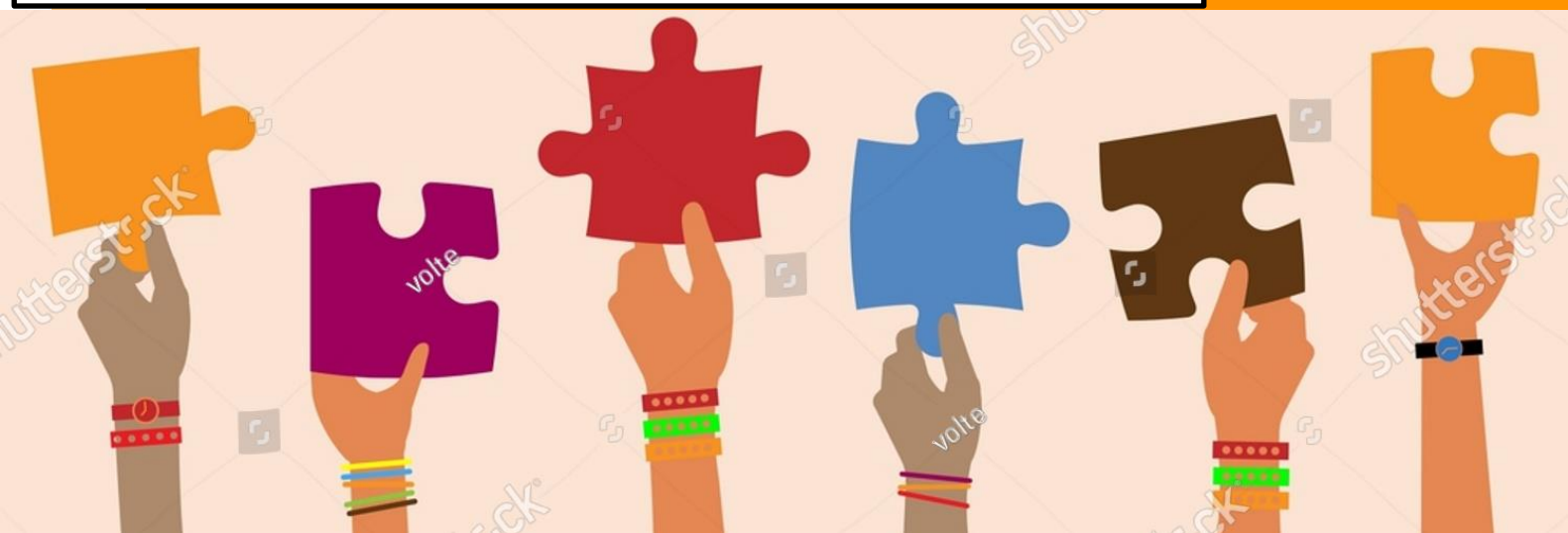
precos = [1.99, 10.99, 10.50, 8.50, 7.49, 1599.45, 1400.00]

Apresente o “passo a passo” para a ordenação deste vetor utilizando o algoritmo Selection Sort.

Exercício 07: Implemente um algoritmo que solicite ao usuário pelo tamanho de um vetor de números inteiros. O algoritmo deve ser capaz de criar este vetor. O algoritmo deve solicitar ao usuário por todos os dados deste vetor. Apresente os dados informados pelo usuário em ordem crescente. Utilize o algoritmo de ordenação por Bolha.

Exercício 08: Implemente um algoritmo que solicite ao usuário pelo tamanho de um vetor de números ponto-flutuantes. O algoritmo deve ser capaz de criar este vetor e solicitar por todos os números ao usuário para o preenchimento deste vetor. Aplique o algoritmo de Ordenação por Inserção para ordenar os dados deste vetor. Por fim, apresente o vetor ordenado ao usuário.

Exercício 09: Implemente um algoritmo que solicite ao usuário pelo tamanho de um vetor de números inteiros e pelos limites inferior e superior (menor valor possível e maior valor possível para preencher este vetor). O algoritmo deve ser capaz de criar este vetor e preenchê-lo com valores aleatórios entre os limites inferior e superior. Exiba o vetor com os valores. Aplique o algoritmo de Ordenação por Seleção para ordenar os dados deste vetor. Por fim, apresente o vetor ordenado ao usuário.



8. Estrutura de Dados Homogênea Multidimensional

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Entender o funcionamento das estruturas de dados homogêneas multidimensionais
- Implementar algoritmos que utilizem estruturas de dados homogêneas bidimensionais (matrizes).

O conteúdo trabalhado neste capítulo aborda:

- Criação de matrizes
- Utilizar laços de repetição e decisão aninhados para percorrer matrizes.

Arrays multidimensionais

Quando temos uma estrutura de dados homogênea multidimensional, temos o que chamamos de Arrays multidimensionais. Os arrays multidimensionais funcionam da mesma forma que os arrays unidimensionais. Entretanto, eles possuem mais de uma dimensão.

Em especial, quando um array possui duas dimensões, dizemos que temos um array bidimensional ou uma **matriz**. A estrutura é chamada de homogênea visto que todos os valores armazenados pertencem ao mesmo tipo de dado, ou seja, se a matriz for de números inteiros, então todos os valores devem ser números inteiros, e assim por diante.

Lembre-se: array unidimensional é chamado de **vetor** e array bidimensional é chamado de **matriz**.

Para entendermos o sentido de uma dimensão, podemos ter uma visualização de um vetor e de uma matriz.

Vetor:	5	3	0	1	40	12
--------	---	---	---	---	----	----

Matriz	5	1	10	0	61	20
	60	45	100	2	88	30
	3	33	92	16	7	40
	12	25	2	73	0	50

Observe que um vetor possui apenas “uma linha” (ou “uma coluna”).

Observe que uma matriz possui um conjunto de linhas, e, para cada linha, temos um novo conjunto de valores.

Podemos dizer que uma matriz é um vetor de vetores, onde temos um vetor (ex: conjunto de linhas) e, dentro de cada linha, temos outro vetor (ex: colunas).

Entendendo uma matriz

Como comparação de dimensões, considere um **vetor** de números inteiros:

```
vet = [0] * 6
```

Temos a estrutura criada:

vet	0	0	0	0	0	0
indices	[0]	[1]	[2]	[3]	[4]	[5]

Para percorrer este vetor, posição por posição, como ele possui apenas uma dimensão, então utilizamos geralmente um comando **for** para que o contador forneça o índice do vetor:

```
vet = [0] * 6
for cont in range(0, len(vet)):
    vet[cont] = int(input("Digite o valor da posição [{}]:".format(cont)))
```

Entretanto, quando temos uma matriz, temos a seguinte estrutura:

Matriz	Índices	[0]	[1]	[2]	[3]	[4]	[5]
	[0]	5	1	10	0	61	20
	[1]	60	45	100	2	88	30
	[2]	3	33	92	16	7	40
	[3]	12	25	2	73	0	50

Observe que para cada valor na tabela (matriz), temos um conjunto de dois índices para localizar este valor. Temos o índice que representa a linha e temos o índice que representa a coluna.

Por exemplo, temos o número 100 na linha com índice [1] e coluna com índice [2].

Observe que na linha de índice [1] temos o conjunto de dados da primeira dimensão:

Índices	[0]	[1]	[2]	[3]	[4]	[5]
[1]	60	45	100	2	88	30

Desta forma, temos um outro array que faz parte da segunda dimensão. Com isso, podemos acessar a posição de índice [2] para localizarmos o valor 100.

Índices	[0]	[1]	[2]	[3]	[4]	[5]
[1]	60	45	100	2	88	30

Como uma matriz consiste em uma tabela, podemos utilizar matrizes em diferentes situações para representar nossos problemas, tais como:

- Um tabuleiro de jogos de damas ou xadrez;
- Uma cartela de jogo de loterias;
- Uma cartela de jogo de bingo;
- Matrizes estudadas na matemática (cálculo de determinantes, diagonal principal, etc.);
- Todos os pontos de cores de uma imagem em pixels;
- Um boletim escolar com as notas de um aluno, onde cada linha pode representar uma determinada disciplina e cada coluna um conceito obtido naquela disciplina;
- Entre tantos outros exemplos.

Declarando matrizes

Quando já conhecemos todos os dados de uma matriz, ela pode ser criada já preenchida, da mesma forma que um vetor. Só precisamos lembrar que uma matriz é um vetor de vetores. Portanto:

Na criação de um vetor temos:

```
vet = [5, 3, 4, 6, 8]
```

que criará a estrutura:

vet	5	3	4	6	8
índices	[0]	[1]	[2]	[3]	[4]

Para a criação de uma matriz temos:

```
matriz = [ [10, 12, 15], [8, 5, 9], [50, 0, 31], [-8, 3, -10] ]
```

que criará a estrutura:

matriz	Índices	[0]	[1]	[2]
	[0]	10	12	15
	[1]	8	5	9
	[2]	50	0	31
	[3]	-8	3	-10

Para acessar um valor na matriz, precisamos indicar os dois índices (o da linha e o da coluna). Por exemplo, o número 12 se encontra na linha de índice [0] e na coluna de índice [1].

Na prática, podemos observar que a matriz é um vetor e cada elemento do vetor possui outro vetor:

<table><tr><td>10</td><td>12</td><td>15</td></tr><tr><td>[0]</td><td>[1]</td><td>[2]</td></tr></table>	10	12	15	[0]	[1]	[2]	<table><tr><td>8</td><td>5</td><td>9</td></tr><tr><td>[0]</td><td>[1]</td><td>[2]</td></tr></table>	8	5	9	[0]	[1]	[2]	<table><tr><td>50</td><td>0</td><td>31</td></tr><tr><td>[0]</td><td>[1]</td><td>[2]</td></tr></table>	50	0	31	[0]	[1]	[2]	<table><tr><td>-8</td><td>3</td><td>-10</td></tr><tr><td>[0]</td><td>[1]</td><td>[2]</td></tr></table>	-8	3	-10	[0]	[1]	[2]
10	12	15																									
[0]	[1]	[2]																									
8	5	9																									
[0]	[1]	[2]																									
50	0	31																									
[0]	[1]	[2]																									
-8	3	-10																									
[0]	[1]	[2]																									
[0]	[1]	[2]	[3]																								

Observe que se acessarmos o vetor no índice [0], seu conteúdo é outro vetor:

10	12	15
[0]	[1]	[2]

E se acessarmos novamente a posição de índice [1], temos o número 12.

Quando temos o tamanho da matriz, mas não sabemos quais serão os dados desta matriz, então podemos criá-la da seguinte forma:

```
matriz = [0] * linhas
for i in range(0, linhas):
    matriz[i] = [0] * colunas
```

Por exemplo, considere a criação de uma matriz de números inteiros contendo 4 linhas e 3 colunas:

```
linhas = 4
colunas = 3

#declarar a matriz com linhas x colunas
matriz = [0] * linhas
for i in range(0, linhas):
    matriz[i] = [0] * colunas
```

Com a instrução acima, criamos a matriz:

mat	[0]	[1]	[2]
[0]	0	0	0
[1]	0	0	0
[2]	0	0	0
[3]	0	0	0

Para criarmos uma matriz de números ponto-flutuantes com 5 colunas e 2 linhas, temos::

```

linhas = 2
colunas = 5
matriz = [0] * linhas
for i in range(0, linhas):
    matriz[i] = [0.0] * colunas

```

Com a instrução acima, criamos a matriz:

mat	[0]	[1]	[2]	[3]	[4]
[0]	0.0	0.0	0.0	0.0	0.0
[1]	0.0	0.0	0.0	0.0	0.0

Da mesma forma, podemos criar matrizes do tipo String ou Booleanas, tais como:

```

linhas = 3
colunas = 3
matrizString = [""] * linhas
for i in range(0, linhas):
    matrizString[i] = [""] * colunas

linhas = 10
colunas = 8
matrizBooleana = [False] * linhas
for i in range(0, linhas):
    matrizBooleana[i] = [False] * colunas

```

Acessando valores dentro de uma matriz

Como uma matriz possui duas dimensões, então precisaremos de dois laços de repetição **for** aninhados para podermos percorrer a matriz. O primeiro comando **for** pode ser utilizado para que a variável contadora consiga posicionar linha por linha. O segundo comando **for** pode ser utilizado para que a variável contadora consiga posicionar coluna por coluna.

Por exemplo, considere uma matriz de tamanho 4x5, ou seja, possui 4 linhas e 5 colunas:

```

linhas = 5
colunas = 4
matriz = [0] * linhas
for i in range(0, linhas):
    matriz[i] = [0] * colunas

for i in range (0, len(matriz)):
    for j in range (0, len(matriz[i])):
        print("[", i, "][", j, "]= ", matriz[i][j])

```

Inicialmente a variável **i** começa com o valor 0 e o laço de repetição será executado. Como a variável **i** representa uma linha, então estamos nos referindo à linha de índice 0.

Dentro dele temos outro laço de repetição onde a variável **j** começa com 0 representando que estamos na coluna de índice 0. Neste momento, estamos na linha 0 e na coluna 0. Precisamos finalizar a execução do laço mais interno, portanto, após uma mensagem ser exibida na tela, a variável **j** é atualizada, indicando que estamos indo para a coluna de índice 1.

Após a execução deste ciclo, o valor de `j` será atualizado para 2, depois para 3 e, finalmente, para 4, indicando que estamos na quinta coluna da tabela (observe que começamos com 0 indicando a primeira coluna).

Como o laço de repetição mais interno foi finalizado, então o laço mais externo será retomado, atualizando o valor de `i` para 1 indicando que estamos na segunda linha da tabela. Nesta execução, temos o laço interno novamente, onde a variável `j` inicializará novamente com 0 indicando a primeira coluna e seguirá com os valores 1, 2, 3 e 4.

A repetição retorna ao laço mais externo onde `i` será atualizado novamente com o valor 2 indicando que estamos na terceira linha, enquanto a variável `j` percorrerá novamente valor por valor nas colunas.

A repetição retorna no laço mais externo onde `i` será atualizado para 3 (quarta linha) e o processo de percorrer coluna por coluna é executado novamente.

Outra instrução importante neste código se refere a `matriz[i][j]`. Observe que para acessar um valor individualmente dentro da matriz, precisamos fornecer o endereço completo deste valor, ou seja, precisamos indicar qual linha `[i]` e qual coluna `[j]` estamos acessando. Os índices são sempre números inteiros e cada índice é colocado em um par de colchetes:

```
matriz[linha][coluna]
```

Desta forma, considere a seguinte matriz:

```
linhas = 4
colunas = 5
matriz = [0] * linhas
for i in range(0, linhas):
    matriz[i] = [0] * colunas
```

que pode ser representada graficamente por:

matriz	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0

A instrução

```
matriz[2][3] = 40
```

faz com que o número 40 seja atribuído à matriz na linha 2 e na coluna 3:

matriz	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	40	0
[3]	0	0	0	0	0

A instrução

```
matriz[0][0] = 55
```

faz com que o número 55 seja atribuído à matriz na linha 0 e na coluna 0:

matriz	[0]	[1]	[2]	[3]	[4]
[0]	55	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	40	0
[3]	0	0	0	0	0

A instrução

```
matriz[1][1] = int(input("Forneça um número:"))
```

faz com que o algoritmo exiba uma mensagem “Forneça um número:” na tela do computador e aguarda até que o usuário informe um número. Imaginando que o usuário digitou o número 199, este número será atribuído à matriz na linha 1 com a coluna 1:

matriz	[0]	[1]	[2]	[3]	[4]
[0]	55	0	0	0	0
[1]	0	199	0	0	0
[2]	0	0	0	40	0
[3]	0	0	0	0	0

Podemos exibir valores da matriz individualmente fornecendo a mesma notação. Por exemplo, a instrução

```
print(matriz[1][1])
```

exibirá na tela o valor 199.

Desta forma, quando desejarmos percorrer a matriz, posição por posição, sempre precisaremos de dois laços de repetição. Um representa o posicionamento da linha (a cada repetição estaremos em uma linha diferente), enquanto que o outro laço de repetição representa o posicionamento da coluna (a cada repetição estaremos em uma coluna diferente).

```
for linha in range(0, len(matriz)):
    for coluna in range(0, len(matriz[linha])):
        print( matriz[linha][coluna] )
```

Dica: observe que no laço de repetição mais interno que controla o posicionamento das colunas, o contador iniciará com 0 (indicando o índice da primeira coluna) e irá até o tamanho de **matriz[linha]**. Não podemos esquecer de fornecer a dimensão da linha que estaremos percorrendo, visto que em Python, cada linha pode ter um tamanho diferente de colunas. Como adotamos um padrão que sirva para diversas linguagens de programação, as matrizes possuem o mesmo número de colunas para todas as linhas.

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Exercício 01: O que é uma matriz?

Exercício 02: Explique como acessar um elemento dentro de uma matriz.

Exercício 03: Crie uma matriz chamada A possuindo dimensão 3x3 com os seguintes valores:

10	20	15
9	8	7
1	3	5

Apresente os valores da matriz A na tela do computador, valor por valor, individualmente, utilizando laços de repetição.

Exercício 04: Implemente um algoritmo que contenha uma matriz B de tamanho 4x4. O algoritmo deve solicitar pelos dados de todas as células da matriz e, ao final, apresentá-la ao usuário.

Exercício 05: Implemente um algoritmo que solicite ao usuário por quantas linhas e quantas colunas uma determinada matriz chamada C deve possuir. Em seguida, crie a matriz com a quantidade de linhas e colunas informadas. Solicite ao usuário pelos dados para preencher a matriz e, ao fim, apresente os valores informados.

Exercício 06: Crie uma matriz de tamanho 3x4. Solicite ao usuário pelos dados para preencher esta matriz e, ao fim, apresente a soma de todos os números presentes na matriz.

Exercício 07: Crie uma matriz de tamanho 5x3. Solicite ao usuário pelos dados para preencher esta matriz. Após preencher a matriz, informe ao usuário (cada item pode ser desenvolvido em um algoritmo independente):

- A somatória dos valores para cada linha
- A somatória dos valores para cada coluna
- A somatória de todos os valores da matriz
- O maior valor para cada linha
- O maior valor para cada coluna
- O menor valor para cada linha
- O menor valor para cada coluna
- O maior e o menor valor da matriz
- A média aritmética dos valores da matriz

Exercício 08: Crie uma matriz quadrática possuindo um tamanho informado pelo usuário. Em seguida, preencha esta matriz com dados solicitados ao usuário. Por fim, apresente:

- Os elementos da diagonal principal
- Os elementos da diagonal secundária
- A soma dos elementos da diagonal principal
- A soma dos elementos da diagonal secundária
- O maior e o menor elemento da diagonal principal
- O maior e o menor elemento da diagonal secundária

Exercício 09: Implemente um algoritmo que seja capaz de solicitar ao usuário pelo tamanho que uma matriz deve possuir.

Em seguida, o algoritmo deve preencher esta matriz com valores sorteados aleatoriamente entre 0 e 200.

Apresente a matriz ao usuário. O algoritmo deve ser capaz de fazer a soma de todos os valores da matriz e dividir a soma dos valores pelo maior valor encontrado na matriz.

Exercício 10: Sejam duas matrizes $A[4][6]$ e $B[6][4]$. Desenvolva um algoritmo que, tendo-se as matrizes A e B preenchidas, seja capaz de gerar a matriz C contendo o produto matricial de A por B.

DICA: para calcular o produto de matrizes, precisamos fazer os seguintes cálculos:

Cada elemento de $C[i][j]$ é obtido pela soma dos produtos dos elementos correspondentes da i-ésima linha de A pelos elementos da j-ésima coluna de B. Ou seja:

Seja $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ e $B = \begin{bmatrix} -1 & 3 \\ 4 & 2 \end{bmatrix}$. Temos que $C[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0]$

Logo, $C[0][0] = (1 * -1) + (2 * 4) = 7$ (soma de cada item da linha 0 de A multiplicado por cada item da coluna 0 de B).

Exercício 11: Implemente um algoritmo que receba uma matriz $A[10][10]$ e realize as seguintes trocas:

- a linha 2 com a linha 8;
- a coluna 4 com a coluna 10;
- a diagonal principal com a secundária;
- a linha 5 com a coluna 10;

Exercício 12: Implemente um algoritmo que receba uma matriz $B[6][4]$ e apresente a soma dos valores das linhas pares.

Exercício 13: Implemente um algoritmo que receba uma matriz C de tamanho 3x3. Solicite ao usuário por um número inteiro. Altere todos os valores da matriz, somando o valor atual de cada célula com o número informado.

Exercício 14: Crie um algoritmo que contenha uma matriz do tamanho 4x4 com valores sorteados aleatoriamente entre 0 e 200. Em seguida, apresente todos os valores acima da diagonal principal e todos os valores abaixo da diagonal principal.

Dica:

180	45	103	80
98	100	89	1
145	7	19	199
77	36	29	179

Diagonal principal: Fundo verde

Valores acima da diagonal principal: Fundo amarelado

Valores abaixo da diagonal principal: Fundo avermelhado

Exercício 15: Elabore um algoritmo que solicite ao usuário por quantas linhas e quantas colunas uma matriz deve ter. Em seguida, preencha esta matriz com valores sorteados aleatoriamente (entre 0 e 200). Apresente ao final quantos destes valores são maiores que 100.

Exercício 16: Seja uma matriz $A[5][5]$ com valores sorteados aleatoriamente entre 0 e 1000. Solicite ao usuário por um número e apresente na tela se este número informado está presente ou não na matriz.

Exercício 17: Faça um programa para gerar automaticamente números entre 0 e 99 de uma cartela de bingo. Sabendo que cada cartela deverá conter 5 linhas de 5 números, gere estes dados de modo a não ter números repetidos dentro das cartelas. O programa deve exibir na tela a cartela gerada.

Exercício 18: Leia uma matriz 10 x 3 com as notas de 10 alunos em 3 provas. Em seguida, escreva o número de alunos cuja pior nota foi na prova 1, o número de alunos cuja pior nota foi na prova 2, e o número de alunos cuja pior nota foi na prova 3. Em caso de empate das piores notas de um aluno, o critério de desempate é arbitrário, mas o aluno deve ser contabilizado apenas uma vez.

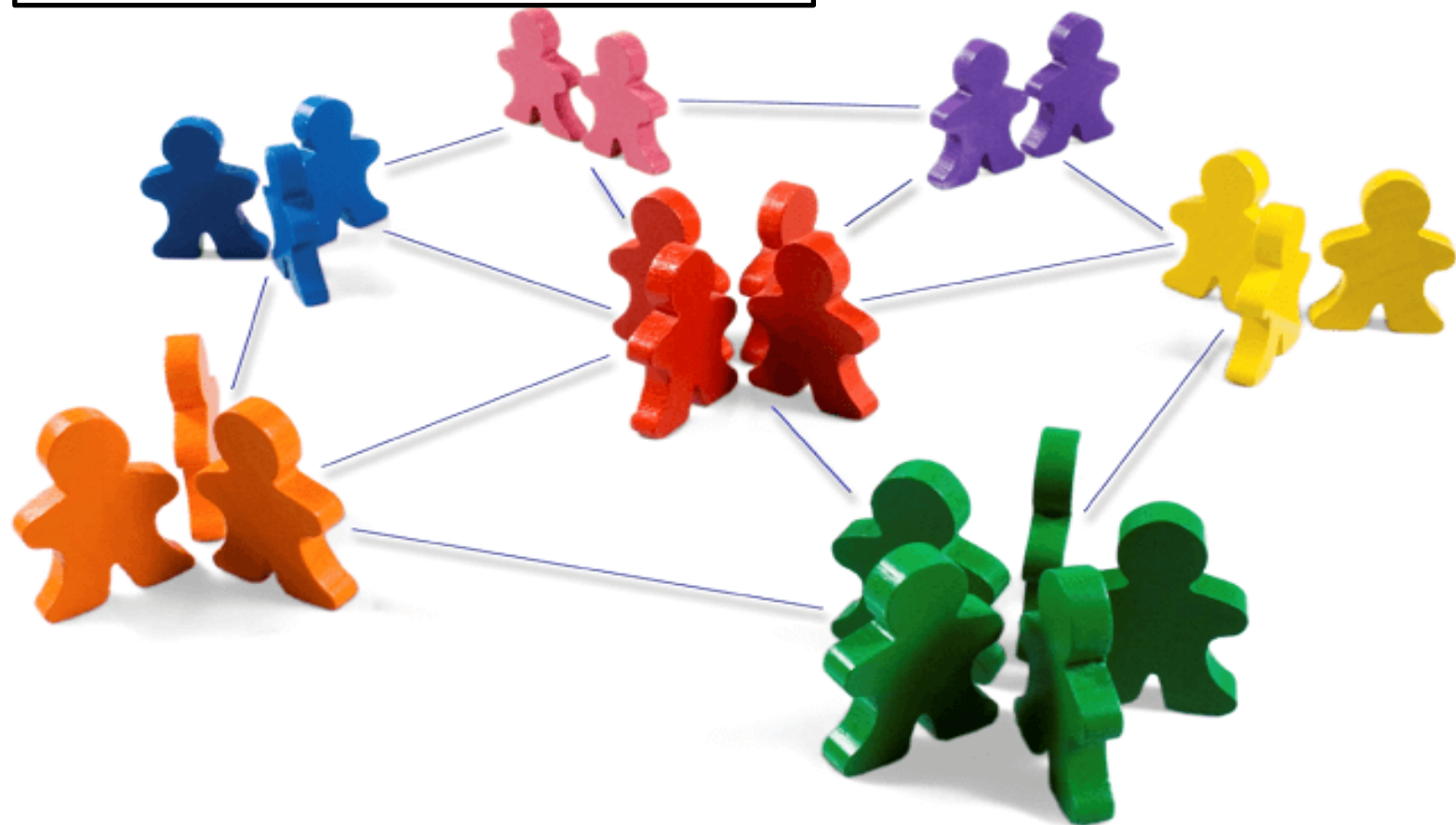
Exercício 19: Faça um programa que leia uma matriz de 5 linhas e 4 colunas contendo as seguintes informações sobre alunos de uma disciplina:

- Primeira coluna: número de matrícula
- Segunda coluna: primeira nota obtida em uma avaliação (0 a 100)
- Terceira coluna: segunda nota obtida em uma avaliação (0 a 100)
- Quarta coluna: média das notas obtidas

O algoritmo deve solicitar pela matrícula do aluno e pelas duas notas obtidas. Em seguida, deve calcular a média final das notas. Apresente a tabela com todos os dados ao usuário. Apresente o aluno que conseguiu tirar a maior nota (maior média), considerando que só um aluno consiga a maior nota).

Exercício 20: Um jogo da velha é formado por um tabuleiro contendo 3 linhas e 3 colunas. Considere que o jogador seja representado pelo valor 1 e que o computador seja representado pelo valor -1. Considere que o valor 0 indique a casa que ainda não foi preenchida.

Desenvolva um algoritmo que seja capaz de representar um jogo da velha. O sistema deve solicitar ao usuário pela linha e pela coluna até que a casa informada esteja vazia (valor 0). Em seguida, o sistema deverá sortear uma linha e uma coluna até que a casa sorteada esteja vazia (valor 0). Repita o processo até que um jogador vença a partida ou até que a matriz seja totalmente preenchida.



9. Subprogramação

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Divisão do algoritmo em módulos
- Implementação de procedimentos e funções
- Passagem de parâmetros
- Retorno de valores
- Distinção entre variáveis locais e variáveis globais.

O conteúdo trabalhado neste capítulo aborda:

- Modularização do código em subalgoritmos (procedimentos e funções)
- Criação de biblioteca externa de funções e impotação de códigos.

Modularização de algoritmos

Subalgoritmo, também conhecidos como Rotina ou Módulo consiste em um conjunto de códigos (bloco de instruções) que representa alguma ação do sistema (realiza alguma tarefa específica). O uso de subalgoritmos torna a leitura do código mais intuitiva e sempre que uma determinada ação precisar ser repetida, o código não precisa ser refeito em outros trechos do programa.

Uma grande vantagem de se utilizar subalgoritmos é a possibilidade de dividir o código em módulos (partes). Desta forma, fica mais fácil para testar os códigos desenvolvidos (o teste é feito por módulos).

Outra vantagem é a possibilidade de redução de códigos, visto que uma vez desenvolvido, o bloco de instruções pode ser chamado diversas vezes.

Os subalgoritmos são desenvolvidos para executar algumas tarefas específicas. Por exemplo: somar dois números, solicitar dados e efetuar transformações nestes dados, etc.

Exemplos de subalgoritmos disponíveis na linguagem Python:

- **print():** usado para exibir informações na tela. Você sabe como ela foi implementada? Existe um código dentro de `print()` que é responsável por receber os parâmetros e executar a ação de exibir informações em vídeo. Se tivermos várias informações para serem exibidas (vários comandos `print()`), estas ações não precisam ser replicadas. Basta chamar a rotina `print()`.
- **math.sqrt():** usado para calcular a raiz quadrada de um valor. Sempre que precisar efetuar o cálculo da raiz quadrada, basta chamar esta rotina.
- **input():** esta rotina trata da entrada de dados pelo teclado. Sempre que precisarmos da ação de digitar valores para fornecer entradas ao programa, podemos chama-la.
- **len(vetor):** esta rotina é responsável por calcular quantos elementos um vetor possui.
- **int(valor):** esta rotina é responsável por converter um valor para um número inteiro.

Podemos ter dois tipos de rotinas: **Procedimentos e Funções**.

Procedimentos

Os procedimentos são subalgoritmos responsáveis por executar um conjunto de instruções (conjunto de comandos). Estes comandos são executados sempre que o procedimento for chamado.

Para definir um procedimento, utilizamos a palavra **def** seguida do nome do procedimento e um par de parênteses.

Sintaxe:

```
def nomeFuncao() :  
    conjunto de instruções
```

Os procedimentos devem ser criados antes de serem chamados.

DICA: criar todos os procedimentos no início do código

Exemplo:

Criar um procedimento chamado `exibirMenu` que exibe um conjunto de opções ao usuário:

```
def exibirMenu() :  
    print("MENU DE OPÇÕES:")  
    print("1. incluir")  
    print("2. alterar")  
    print("3. excluir")  
    print("4. sair")
```

Para chamar o procedimento e reutilizá-lo sempre que necessário, basta fornecer ao código o nome do procedimento.

Exemplo:

```
exibirMenu()
```

Se precisar exibir o menu mais de uma vez, o código não precisa ser refeito. Basta chamar o procedimento quantas vezes forem necessárias.

Exemplo:

```
def exibirMenu() :  
    print("MENU DE OPÇÕES:")  
    print("1. incluir")  
    print("2. alterar")  
    print("3. excluir")  
    print("4. sair")  
  
exibirMenu()  
exibirMenu()  
exibirMenu()
```

Produz como saída:

```
MENU DE OPÇÕES:  
1. incluir  
2. alterar  
3. excluir  
4. sair  
MENU DE OPÇÕES:  
1. incluir  
2. alterar  
3. excluir  
4. sair  
MENU DE OPÇÕES:  
1. incluir  
2. alterar  
3. excluir  
4. sair
```

Os procedimentos podem receber **parâmetros**. Parâmetros são valores que são passados para o procedimento. Sempre que um procedimento precisar de valores para ser executado, estes valores devem ser fornecidos nos parâmetros. Os parâmetros sempre vem dentro dos parênteses.


```
def exibirMultiplicacao(a, b):
    mult = a * b
    print("Multiplicando", a, "e", b, "temos", mult)

exibirMultiplicacao(2, 3)
exibirMultiplicacao(5, 5)
exibirMultiplicacao(10, 210)
```

O resultado apresentado será:

```

Multiplicando 2 e 3 temos 6
Multiplicando 5 e 5 temos 25
Multiplicando 10 e 210 temos 2100

```

As variáveis **a** e **b** fornecidas no parâmetro do procedimento, assim como a variável **mult** criada dentro do procedimento, são chamadas de **Variáveis Locais**. Uma variável local só existe (só é visível) dentro do módulo em que ela foi criada. Fora deste módulo, ela não tem efeito.

Exemplo:

```
def exibirMultiplicacao(a, b):
    mult = a * b
    print("Multiplicando", a, "e", b, "temos", mult)

exibirMultiplicacao(2, 3)
exibirMultiplicacao(5, 5)
exibirMultiplicacao(10, 210)

print(a)
```

Neste exemplo, o valor da variável **a** será exibido ao usuário na última linha do código. Entretanto, o código não pertence ao bloco do procedimento. Um erro será gerado:

```

print(a)
NameError: name 'a' is not defined

```

Além das variáveis locais, podemos ter **Variáveis Globais**. As variáveis globais são visíveis tanto no programa principal quanto nos blocos dos procedimentos. Elas podem ser utilizadas durante todo o programa.

```

aula = "Algoritmos"

def exibirMensagem():
    print("A aula agora é", aula)

exibirMensagem()
print("Acessando a mesma variável aula=", aula)
```

Acessando a variável **aula** dentro do procedimento

Acessando a variável **aula** fora do procedimento

Repare que para que o procedimento consiga acessar a variável global, ela deve ter sido definida antes do procedimento. Outra característica desta abordagem é que a variável é apenas leitura, ou seja, ela apenas pode ser utilizada para cálculos ou exibições e seu valor não pode ser alterado.

Caso a variável global tenha sido definida após o procedimento, podemos utilizar a palavra **global** para indicar que a variável a ser acessada foi criada fora do escopo do procedimento. Além disso, informando ao procedimento que a variável é global, o conteúdo desta variável pode ser modificado dentro do procedimento.

Exemplo: acessando variável global definida antes do bloco do procedimento.

```
aula = "Algoritmos"

def exibirMensagem():
    global aula
    print("A aula agora é", aula)
    aula = "Java"
    print(aula)

exibirMensagem()
print("Acessando a mesma variável aula=", aula)
```

Exemplo: acessando variável global definida após o bloco do procedimento.

```
def exibirMensagem():
    global aula
    print("A aula agora é", aula)
    aula = "Java"
    print(aula)

aula = "Algoritmos"

exibirMensagem()
print("Acessando a mesma variável aula=", aula)
```

Funções

As funções são semelhantes aos procedimentos. Entretanto, uma função sempre possui um valor, ou seja, toda função executa alguma ação ou cálculo, e o resultado desta ação é retornada ao local onde a função foi chamada.

Para informar a função que sua execução terminou e que um valor deve ser retornado, utilizamos o comando **return** seguido pelo valor a ser retornado.

Exemplo:

```
def lerInteiro():
    x = int(input("Informe número:"))
    return x
```

Ao executar a função **lerInteiro()**, o sistema solicitará um valor ao usuário, converterá este valor para int e o valor será atribuído a variável local **x**. Em seguida, o valor de **x** será retornado para o local da chamada da função.

Para chamar a função, diferentemente dos procedimentos que não retornam valores, o valor retornado deve ser utilizado para algum fim (atribuído a uma variável ou usado diretamente no parâmetro de outra função/procedimento).

Exemplo:

```
valor1 = lerInteiro()

print("O inteiro informado foi", lerInteiro())
```

Exemplo de uso de função:

```
def lerInteiro():
    x = int(input("Informe número:"))
    return x

valor1 = lerInteiro()
print("O inteiro informado foi", valor1)

valor2 = lerInteiro()
print("O inteiro informado foi", valor2)

print("O inteiro informado foi", lerInteiro())
```

As funções também podem receber valores nos parâmetros e também podem acessar variáveis globais, da mesma forma que os procedimentos.

```
def lerInteiro():
    x = int(input("Digite um número inteiro"))
    return x

def calcularSoma(a, b):
    return a + b

print("2 + 3 = ", calcularSoma(2, 3))

soma = calcularSoma(12, 5)
print("12 + 5 = ", soma)

a = lerInteiro()
b = lerInteiro()

soma2 = calcularSoma(a, b)
print("A soma dos números informados foi", soma2)

print("A outra soma foi", calcularSoma(lerInteiro(), lerInteiro()))
```

Criando nossas próprias bibliotecas

Vimos que para utilizar funções matemáticas, precisamos importar a biblioteca math:

```
import math
```

Vimos que para utilizar funções para números aleatórios, precisamos importar a biblioteca random:

```
import random
```

A linguagem Python possui um conjunto vasto de bibliotecas disponíveis. Uma lista extensa destas bibliotecas pode ser obtida em <https://docs.python.org/3/library/>. Muitas outras bibliotecas podem ser instaladas como adicional da linguagem, tais como as bibliotecas comentadas em <https://terminalroot.com.br/2019/12/as-30-melhores-bibliotecas-e-pacotes-python-para-iniciantes.html>.

Uma biblioteca consiste em um conjunto de módulos, ou seja, é um conjunto de variáveis, constantes, procedimentos, funções, além de diversas outras estruturas de dados. Esta biblioteca pode ser reutilizada sempre que os recursos oferecidos por esta biblioteca forem necessários em seus algoritmos.

Para criarmos nossas próprias bibliotecas, precisamos criar um arquivo adicional com a extensão **.py** e o código dentro deste arquivo será a implementação das funções e procedimentos utilizando a linguagem Python. Por exemplo, imagine um arquivo chamado **minhasfuncoes.py** que contém:

- Função **somar(a,b)** responsável por efetuar a soma de dois números e retornar esta soma
- Função **maior(a,b)** que retorna **True** caso o valor de **a** seja maior do que o valor de **b** e **False** em caso contrário
- Procedimento **exibirMenu()** que exibe um menu de ações ao usuário
- Uma função **lerOpcao()** que faz a leitura de um valor inteiro e retorna o valor digitado

Estas funções são apresentadas abaixo:

```
Arquivo: minhasfuncoes.py
opcao = 0

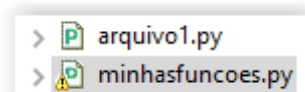
def somar(a, b):
    r = a + b
    return r

def maior(a, b):
    if (a > b):
        return True
    else:
        return False

def lerOpcao():
    op = int(input("OPÇÃO: "))

def exibirMenu():
    print("Menu de opções:")
    print("1. Somar dois números")
    print("2. Comparar dois números")
    global opcao
    opcao = lerOpcao()
```

Para utilizarmos a nossa biblioteca, basta colocarmos o arquivo **minhasfuncoes.py** na mesma pasta junto ao arquivo do nosso algoritmo. Por exemplo, caso o arquivo do nosso algoritmo se chamar **arquivo1.py**, os arquivos **minhasfuncoes.py** e **arquivo1.py** devem estar juntos, na mesma pasta.



No arquivo do código-fonte em Python do algoritmo (arquivo1.py), basta importarmos a nossa biblioteca, fornecendo o nome do arquivo, sem a extensão py.

Para chamar as funções desta biblioteca, basta utilizar **nomeDaBiblioteca.nomeDaFuncao()**.

Por exemplo:

```
import minhasfuncoes

minhasfuncoes.exibirMenu()
if (minhasfuncoes.opcao == 1):
    v1 = int(input("Informe o 1. valor:"))
    v2 = int(input("Informe o 2. valor:"))
    x = minhasfuncoes.somar(v1, v2)
    print("Resultado:", x)
elif (minhasfuncoes.opcao == 2):
    v1 = int(input("Informe o 1. valor:"))
    v2 = int(input("Informe o 2. valor:"))
    if (minhasfuncoes.maior(v1, v2)):
        print(v1, "é maior que", v2)
    else:
        print(v2, "é maior que", v1)
else:
    print("Opção inválida")
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

PROCEDIMENTOS

Exercício 01: O que é um procedimento?

Exercício 02: Qual é a sintaxe para se criar um procedimento?

Exercício 03: Seja `acao()` um procedimento. Durante o algoritmo, como chamar este procedimento para que seu código seja executado?

Exercício 04: O que são parâmetros? Para que eles servem?

Exercício 05: Seja `acao(a, b, c)` um procedimento que recebe três números inteiros como parâmetro. Como chamar este procedimento para que seu código seja executado?

Exercício 06: As variáveis declaradas nos parâmetros de um procedimento, assim como as variáveis declaradas dentro do procedimento, são variáveis chamadas de Locais ou Globais? Por que?

Exercício 07: Qual é a diferença entre variáveis locais de variáveis globais?

Exercício 08: Seja `idade = 20` uma variável definida no algoritmo. Como fazer para que esta variável possa ser modificada dentro de um procedimento?

Exercício 09: Implemente um procedimento que seja capaz de exibir a mensagem “Olá Pessoal”. Chame este procedimento 4x.

Exercício 10: Implemente um procedimento que seja capaz de solicitar por um número e apresentar se este número é par ou ímpar. Chame este procedimento 5x.

Exercício 11: Implemente um algoritmo que contenha um procedimento responsável por exibir o seguinte menu ao usuário:

MENU DE OPÇÕES:

1. Somar dois números
2. Subtrair dois números
3. Dividir dois números
4. Multiplicar dois números 5. Sair

No corpo principal do algoritmo, o menu deverá ser exibido e a opção deverá ser solicitada ao usuário. O algoritmo deve emitir uma mensagem “opção inválida” caso o usuário informe um valor diferente das opções do menu. Garanta que o algoritmo seja finalizado apenas quando a opção Sair for informada. (OBS: não precisa implementar as ações do menu, apenas o Sair).

Exercício 12: (continuação 11) Implemente os seguintes procedimentos:

- **somar():** este procedimento deverá solicitar por dois números e apresentar a soma dos dois números informados.
- **subtrair():** este procedimento deverá solicitar por dois números e apresentar a subtração dos dois números
- **dividir(a, b):** este procedimento deverá realizar a divisão do maior número pelo menor número informado nos parâmetros e apresentar ao usuário. Solicitar dois números no algoritmo principal e informa-los ao procedimento.
- **multiplicar(a, b):** este procedimento deverá multiplicar os dois valores informados nos parâmetros e apresentar o resultado ao usuário. Solicitar os dois valores no algoritmo principal e informa-los ao procedimento.

Chame os procedimentos correspondentes às ações informadas no menu.

Exercício 13: Crie um procedimento que receba um valor inteiro positivo. O procedimento deve calcular o fatorial deste número e apresentá-lo ao usuário.

Exercício 14: Crie um procedimento que receba como parâmetro o número total de alunos de uma turma, a quantidade de notas obtidas em uma disciplina e o nome da disciplina. O procedimento deverá solicitar o total de notas, por aluno e calcular a sua média aritmética. Crie também um segundo procedimento que receba os dados informados e o nome da disciplina, sendo que este procedimento deverá apresentar ao usuário uma tabela contendo as três notas e a média, para a disciplina informada.

Exercício 15: Faça um procedimento que receba, por parâmetro, uma matriz A(5,5) já preenchida. O procedimento deve atualizar o valor de uma variável global chamada soma com a soma de todos os valores da matriz. Apresente a soma no corpo principal do algoritmo.

Exercício 16: Implemente um algoritmo que seja capaz de solicitar pelos valores das variáveis A, B e C. Implemente um módulo que seja capaz de acessar estas variáveis e fazer com que A receba o valor de B, B receba o valor de C e C receba o valor de A. Apresente os valores de A, B e C antes de chamar o módulo implementado, e após a chamada do módulo.

Exercício 17: Escreva um procedimento chamado **exibirLinha** que apresente na tela uma linha contendo *. Implemente um procedimento chamado **exibirVetor** que seja capaz de exibir os valores de um vetor informado no parâmetro. A exibição do vetor deverá ser:

```
*****                                exibirLinha
DADOS DO VETOR
*****                                exibirLinha
...                                  dados do vetor
...
...
...
```

Implemente também um procedimento chamado **lerVetor** que crie o vetor, faça a leitura dos dados do vetor e chame a exibição do vetor.

Exercício 18: Faça um procedimento que receba como parâmetro um número inteiro no intervalo de 1 a 9 e mostre a seguinte tabela de multiplicação (no exemplo, n = 7):

1						
2	4					
3	6	9				
4	8	12	16			
5	10	15	20	25		
6	12	18	24	30	36	
7	14	21	28	35	42	49

FUNÇÕES

Exercício 19: Elabore uma função chamada **fatorial** que receba um número inteiro positivo como parâmetro e retorne o fatorial deste número.

Exercício 20: Elabore uma função que receba 4 números inteiros nos parâmetros e retorne o maior dos números.

Exercício 21: Elabore uma função que receba um valor numérico nos parâmetros e retorne True caso o valor seja positivo e False caso seja negativo. Efetue testes diretamente no algoritmo principal chamando esta função dentro do comando if.

Exercício 22: Elabore um algoritmo que faça o que se pede:

- No programa principal, solicite ao usuário pelo número N de valores a ser informado;
- Implemente um módulo que receba o número N de valores por parâmetro. Este módulo deve ser capaz de fazer a leitura de N números e solicitar ao usuário por um número a ser buscado dentro deste conjunto de valores. Por fim, apresente uma mensagem indicando se o valor buscado está presente ou não no conjunto (usando o tópico abaixo);
- Implemente um módulo que receba o conjunto de números e o número a ser buscado por parâmetro. O módulo deverá retornar True caso o valor buscado esteja presente no conjunto, ou False caso contrário.

Exercício 23: Elabore um programa que contenha uma rotina que receba três notas de um aluno como parâmetros e uma letra. Se a letra for A, a rotina deverá calcular a média Aritmética das notas do aluno. Se a letra for P, a rotina deverá calcular a média Ponderada (com pesos 5, 3 e 2). A média deverá ser retornada ao programa principal para, então, ser mostrada.

Caso a letra seja diferente de A ou P, a rotina deverá retornar -1.

Exercício 24: Implemente uma rotina chamada **fatorial** que receba um valor inteiro como parâmetro e retorne o fatorial deste número. Implemente também uma rotina chamada **calcularS** que receba como parâmetro um valor inteiro e positivo N e retorne o valor de S, obtido pelo seguinte cálculo:

$$S = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{N!}$$

Solicite o valor de N e apresente o valor de S no programa principal.

Exercício 25: Foi realizada uma pesquisa sobre algumas características físicas de cinco habitantes de uma região. Foram coletados os seguintes dados de cada habitante: sexo, cor dos olhos (A: azuis, C: castanhos), cor dos cabelos (L: louros, P: pretos, C: castanhos) e idade.

- a) Implemente um procedimento que leia estes dados, armazenando-os em vetores (vetores declarados como variáveis globais)
- b) Faça uma função que determine e retorne ao programa principal a média de idade das pessoas com olhos castanhos e cabelos pretos
- c) Faça uma função que calcule e retorne ao programa principal a maior idade entre os habitantes
- d) Faça uma função que determine e devolva ao programa principal a quantidade de indivíduos de sexo feminino com idade entre 18 e 35 anos (inclusive) e que tenham olhos azuis e cabelos louros.

Exercício 26: As palavras “ovo”, “Ana”, “Arara”, “mamam”, “ralar”, “rir”, “salas”, “aibofobia” são palíndromos, ou seja, são palavras que podem ser lidas do início ao fim ou do fim ao início. Podemos ter frases palíndromos (desconsiderando espaços em branco), tais como “o galo ama o lago”, “a base do teto desaba”, “a cara rajada da jararaca”, “ato idiota”, “anotaram a data da maratona”, “a gorda ama a droga”, “a grama é amarga”, “a sacada da casa”. Implemente uma função chamada **ePalindromo** que receba como parâmetro uma String. Esta função deverá retornar True caso o parâmetro seja um palíndromo, ou False, caso contrário.

Exercício 27: Elabore uma rotina que receba dois vetores A e B de números inteiros como parâmetro. A rotina deverá devolver ao programa principal um terceiro vetor C formado pela união dos dois primeiros. Apresente o vetor C no programa principal.

Exercício 28: Considere um sistema informatizado para cadastro de alunos. Ao executar o sistema, o seguinte menu deverá ser exibido ao usuário:

MENU:

1. Incluir novo aluno

2. Alterar aluno
 3. Excluir aluno
 4. Consultar aluno
 5. Exibir todos os alunos
 6. Sair do sistema
- OPÇÃO: ____

Considere que possam existir no máximo 20 alunos (um vetor contendo o nome de 20 alunos no máximo declarado como variável global).

- Implemente uma função responsável por apresentar o menu ao usuário e retornar a opção informada.
- Implemente uma função chamada **incluirAluno** que solicite pelo nome de um aluno e inclua-o na lista de alunos (caso seja possível)
- Implemente um procedimento chamado **alterarAluno** que solicite pela posição do aluno a ser alterado na lista de nomes e pelo novo nome. Atualize a lista, na posição informada, com o novo nome informado.
- Implemente um procedimento chamado **excluirAluno** que solicite pela posição do aluno a ser excluído na lista de nomes. Atualize a lista, na posição informada, removendo o conteúdo armazenado (atribuindo uma String vazia). Também faça uma readequação da lista, mantendo os nomes cadastrados no início da lista e as posições vazias ao final.
- Implemente uma função chamada **consultarAluno** que receba no parâmetro o nome de um aluno e retorne True caso o nome esteja cadastrado, False caso contrário.
- Implemente um procedimento chamado **exibirAlunosCadastrados** que exiba todos os alunos cadastrados (não exibir as posições vazias sem alunos)

O programa principal deve ser capaz de exibir o menu e executar as ações necessárias, até que o usuário solicite a saída do sistema.

OBS: caso seja necessário, outras variáveis globais podem ser utilizadas.

Exercício 29: Implemente uma função que receba nos parâmetros um vetor numérico. A função deverá retornar ao programa principal o número de valores pares presentes neste vetor.

Exercício 30: Faça um programa que contenha uma rotina capaz de receber nos parâmetros dois valores numéricos e um símbolo. Este símbolo representará a operação que se deseja efetuar com os números (+, -, *, /, %). O resultado do cálculo deve ser apresentado no programa principal.

Exercício 31: Faça uma rotina que receba três valores representando horas, minutos e segundos e retorne o número de segundos.

Exemplo: 2h, 40min, 10seg correspondem a 9610 segundos.

Exercício 32: Implemente um programa que contenha uma rotina capaz de calcular o peso ideal de uma pessoa. Esta rotina deverá receber nos parâmetros o sexo da pessoa e a altura atual. Sabe-se que o peso ideal é calculado da seguinte maneira;

$$\text{Peso_homem} = 72,7 * \text{altura} - 58$$

$$\text{Peso_mulher} = 62,1 * \text{altura} - 44,7$$

Exercício 33: Implemente um programa que contenha os seguintes requisitos:

- Um vetor de números inteiros contendo 15 posições declarado no programa principal como variável global
- Um procedimento capaz de preencher o vetor com números aleatórios entre 0 e 500
- Uma função que retorne o maior número sorteado no vetor
- Uma função que retorne o menor número sorteado no vetor
- Uma função que retorne a média aritmética dos números sorteados no vetor
- Uma função que retorne a posição do maior número sorteado no vetor

O programa deve ser capaz de apresentar ao usuário todos os dados das funções.

Exercício 34: Criar uma função que calcule o número de combinações de n elementos p a p. A fórmula da combinação é a seguinte:

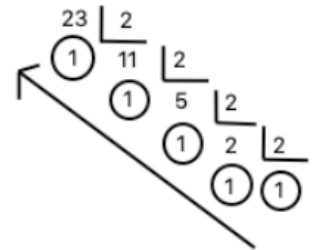
$$C_p^n = \frac{n!}{p! * (n - p)!}$$

Os valores de n e p deverão ser solicitados no bloco principal do algoritmo e informados nos parâmetros da função. O resultado deverá ser exibido no algoritmo principal.

Exercício 35: Elabore uma função que converta um número qualquer da base 10 para qualquer base entre 2 e 10, inclusive.

Exemplo de conversão para a base 2:

Converter 23 (base 10) para base 2 gera o valor 10111





10. Arquivos em Disco

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Entendimento sobre conceitos de arquivos e diretórios/pastas
- Entendimento sobre caminho absoluto e caminho relativo
- Acesso a arquivos em disco.

O conteúdo trabalhado neste capítulo aborda:

- Manipular arquivos em disco (criação, leitura e acesso a dados em arquivos em disco)

Arquivos

Um arquivo consiste em um conjunto de dados que ficam armazenados permanentemente em uma unidade de armazenamento (HD, SSD, pendrive, CD, DVD, etc.), ou seja, é uma área em um disco onde podemos ler e armazenar dados. A manipulação do disco é feita pelo Sistema Operacional, desta forma, não precisamos nos preocupar em como os espaços livres no disco serão gerenciados.

Arquivos podem ser utilizados como entrada e saída de dados dos programas. Os programas podem salvar os seus dados em arquivos e também podem fazer a leitura destes dados salvos.

Para acessar um arquivo, precisamos fornecer o nome do arquivo, juntamente com o caminho (diretório) onde o arquivo se encontra. Também precisamos fornecer o modo como o arquivo será aberto (se é para escrita de dados e/ou para leitura de dados).

Exemplo de caminho para um arquivo: **C:\ifpr\aula\algoritmos\arquivo.txt**

Modos de abertura de arquivos mais comuns:

- r** **read** (apenas para leitura de dados)
- w** **write** (apenas para a criação de arquivo novo e escrita de conteúdos)
- a** **append** (apenas para a criação de arquivo novo ou abertura de arquivo existente, porém permite atualizar o conteúdo de um arquivo caso ele já exista).

Em Python, utilizamos a função **open("arquivo.txt", "modo")** para abrir um arquivo. Esta função retorna como valor um *objeto* do tipo *file* (arquivo). Desta forma, podemos ter:

```
arquivo = open("arquivo.txt", "w")
```

Após a abertura do arquivo, dados podem ser lidos ou salvos neste arquivo. Concluindo as ações sobre este arquivo, ele precisa ser fechado. Ao fechar um arquivo, os recursos do sistema operacional reservados para este arquivo serão liberados.

Em Python, para fechar um arquivo aberto, utilizamos o procedimento

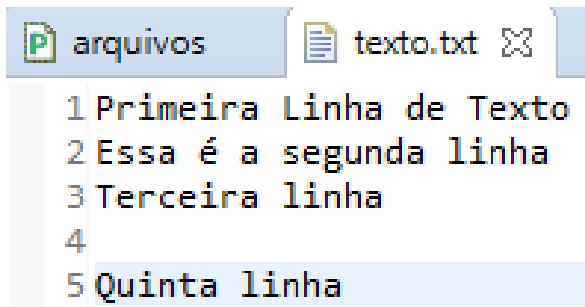
```
arquivo.close()
```

Podemos obter algumas informações dos arquivos:

- arquivo.name** obter o nome do arquivo
- arquivo.mode** obter o modo de abertura do arquivo
- arquivo.closed** obter se o arquivo esta fechado (True se o arquivo estiver fechado, False se o arquivo estiver aberto).

Exemplo:

Considere um arquivo chamado *texto.txt* que esteja na mesma pasta que o arquivo do código-fonte em Python.



O seguinte código irá produzir:

```
arquivo = open("texto.txt", "r")

print("Nome do arquivo: ", arquivo.name)
print("Modo de abertura: ", arquivo.mode)
print("O arquivo está fechado? ", arquivo.closed)

arquivo.close()

print("O arquivo está fechado? ", arquivo.closed)
```

```
Nome do arquivo: texto.txt
Modo de abertura: r
O arquivo está fechado? False
O arquivo está fechado? True
```

Resultado da execução:

Para fazer a leitura dos dados no arquivo podemos utilizar a função

`arquivo.readline()`.

Esta função sempre retornará uma String contendo o conteúdo de uma linha do arquivo.

OBS: uma linha é finalizada quando o ENTER (caractere “\n”) for encontrado.

Exemplo:

```
arquivo = open("texto.txt", "r")

linha = arquivo.readline()
print(linha)

arquivo.close()
```

Apresentará como resultado:

Primeira Linha de Texto

Caso seja necessário ler todas as linhas do arquivo, podemos colocar em um laço de repetição:

```
arquivo = open("texto.txt", "r", encoding="utf-8")

linha = arquivo.readline()

while (linha):
    print(linha)
    linha = arquivo.readline()

arquivo.close()
```

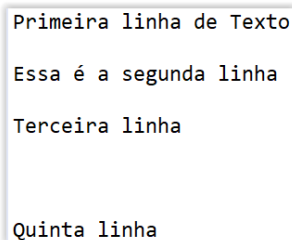
No código acima, o algoritmo faz a leitura de uma linha. Se houver conteúdo, ele é apresentado na tela e uma nova linha é lida. Apresentará como resultado:

```
Primeira Linha de Texto  
  
Essa é a segunda linha  
  
Terceira linha  
  
  
Quinta linha
```

Podemos utilizar a função **arquivo.readlines()**. Esta função faz a leitura de todo o conteúdo do arquivo de uma só vez. Cada linha do arquivo é armazenada em uma posição de um Array (vetor) e este vetor é retornado. Desta forma, podemos percorrer o vetor para acessar o conteúdo de cada linha do arquivo.

```
arquivo = open("texto.txt", "r", encoding="utf-8")  
  
conteudo = arquivo.readlines()  
  
for linha in range(0, len(conteudo)):  
    print(conteudo[linha])  
  
arquivo.close()
```

Saída:



```
Primeira linha de Texto  
  
Essa é a segunda linha  
  
Terceira linha  
  
  
Quinta linha
```

ALTERNATIVA:

```
arquivo = open("texto.txt", "r", encoding="utf-8")  
  
conteudo = arquivo.readlines()  
  
for conteudoLinha in conteudo:  
    print(conteudoLinha)  
  
arquivo.close()
```

Para escrever dados em um arquivo, podemos abri-lo em modo de escrita (**w**) ou de atualização (**a**).

Caso o arquivo não exista, ele será criado.

No modo de escrita (**w**), caso o arquivo já exista, ele será substituído por um arquivo novo, totalmente em branco.

No modo de atualização (**a**), caso o arquivo já exista, ele será aberto para que conteúdos adicionais sejam inseridos no arquivo.

Utilizamos o procedimento

```
arquivo.write("conteúdo")
```

para escrever um conteúdo no arquivo aberto. Esta função, sempre que utilizada, insere o novo conteúdo ao final do arquivo. Para que uma nova linha seja inserida, o caractere “\n” deve ser escrito no arquivo.

Existe também o procedimento

```
arquivo.writelines("conteúdo \nde várias\nlinhas")
```

Com este procedimento, após a inserção de um conteúdo, uma quebra de linha é adicionada automaticamente.

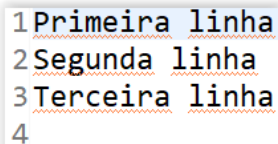
Exemplo 1:

```
arquivo = open("prova.txt", "w", encoding="utf-8")

arquivo.write("Primeira linha\n")
arquivo.write("Segunda linha\n")
arquivo.write("Terceira linha\n")

arquivo.close()
```

Criará o arquivo prova.txt e seu conteúdo será:



Exemplo 2:

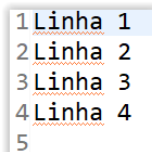
```
arquivo = open("prova.txt", "w", encoding="utf-8")

vetor = ["Linha 1\n", "Linha 2\n", "Linha 3\n", "Linha 4\n"]

arquivo.writelines(vetor)

arquivo.close()
```

Criará o arquivo prova.txt e gravará todo o conteúdo do vetor de uma só vez.:



Exemplo 3: Salvará no arquivo tudo o que o usuário digitar, até que ele digite ENTER para uma String vazia.

```
arquivo = open("texto.txt", "w", encoding="utf-8")

while(True):
    texto = input("Forneça um texto: ")
    if(texto == ""):
        break
    arquivo.write(texto+"\n")

arquivo.close()
```

RESULTADO:

```
Forneça um texto: Primeiro Texto
Forneça um texto: Segundo Texto
Forneça um texto: Terceiro Texto
Forneça um texto: Quarto Texto
Forneça um texto:
```

```
1Primeiro Texto
2Segundo Texto
3Terceiro Texto
4Quarto Texto
5
```

Outras operações que podem ser realizadas com arquivos:

- Listar o conteúdo da pasta atual do programa:

```
import os

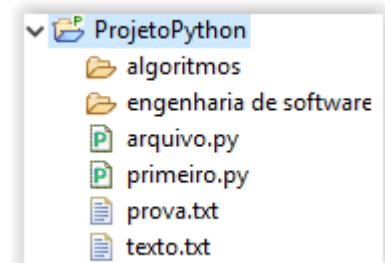
listaDiretorio = os.listdir(".")

for item in listaDiretorio:
    print(item)
```

- Criando diretórios e subdiretórios:

```
import os

os.mkdir("algoritmos")
os.mkdir("engenharia de software")
```



- Acessar diretórios:

```
import os

caminhoAtual = os.getcwd()
print("O caminho atual é", caminhoAtual)

os.chdir("algoritmos")

caminhoAtual = os.getcwd()
print("O caminho atual é", caminhoAtual)
```

```
O caminho atual é D:\eclipse\workspace\ProjetoPython
O caminho atual é D:\eclipse\workspace\ProjetoPython\algoritmos
```

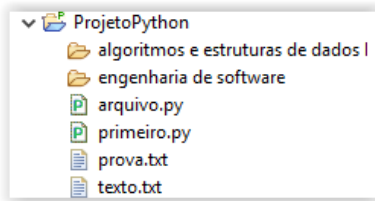
Pode-se trocar de diretórios conforme comandos do Terminal. Exemplos:

- `..` retorna um nível na árvore de diretórios
- `../c` retorna um nível na árvore de diretórios e acessa a pasta c

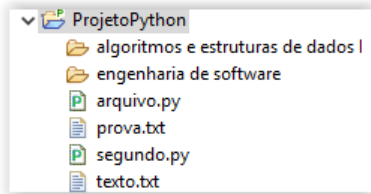
- Alteração no nome de arquivos e no nome de diretórios:

```
import os

os.rename("algoritmos", "algoritmos e estruturas de dados I")
```

```
import os
os.rename("primeiro.py", "segundo.py")
```



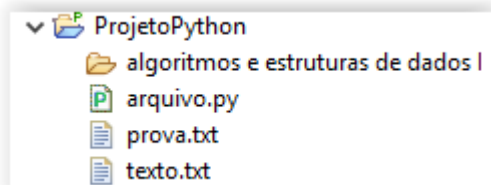
- Excluir arquivos e diretórios:

```
import os

#excluir diretórios: rmdir("nome do diretório")
#excluir arquivos: remove("nome do arquivo")

os.rmdir("engenharia de software")

os.remove("segundo.py")
```



- Testar para verificar se um arquivo ou diretório já existe ou não

```
import os

if(os.path.exists("prova.txt")):
    print("o arquivo já existe")
else:
    print("O arquivo não existe e pode ser criado")
```

- Obter dados sobre o arquivo (atributos do arquivo):

```
import os
import time
import sys

arquivo = open("prova.txt", "r", encoding="utf-8")
print("Nome do arquivo: ", arquivo.name)
print("Tamanho em bytes: ", os.path.getsize(arquivo.name))
print("Arquivo criado em: ", time.ctime(os.path.getctime(arquivo.name)))
print("Arquivo modificado em: ", time.ctime(os.path.getmtime(arquivo.name)))
```

```
print("Arquivo acessado em: ", time.ctime(os.path.getatime(arquivo.name)))
```

```
Nome do arquivo: prova.txt
Tamanho em bytes: 36
Arquivo criado em: Mon Oct 7 17:58:48 2019
Arquivo modificado em: Mon Oct 7 18:01:35 2019
Arquivo acessado em: Mon Oct 7 18:01:36 2019
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Após resolver os exercícios, aguarde um dia e tente resolvê-los novamente você mesmo, mas agora sem utilizar as respostas e dicas como consulta. Se conseguiu resolver todos os exercícios sozinho, então você está aprendendo o conteúdo!

Exercício 01: Implemente um programa que, ao ser executado, solicite ao usuário pelo nome de dois arquivos que contenham textos. O programa deve comparar se os dois arquivos são iguais.

Exercício 02: Implemente um programa que solicite ao usuário pelo nome de um arquivo. Apresente na tela quantas linhas existem neste arquivo.

Exercício 03: Desenvolva um programa que solicite faça a abertura de um arquivo e faça uma cópia do conteúdo deste arquivo para um novo arquivo

Exercício 04: Considere o seguinte requisito de software: Um sistema deve ser capaz de abrir um arquivo de texto. O usuário deve informar um caractere e o sistema deve apresentar quantas vezes o caractere fornecido aparece no texto. Desenvolva um aplicativo que seja capaz de representar este requisito.

Exercício 05: Implemente um algoritmo que faça a leitura de um número N que representa quantos valores devem ser sorteados aleatoriamente (entre 0 e 100). O algoritmo deve sortear os N números aleatórios armazenando-os em um vetor. Por fim, os dados deste vetor devem ser salvos em um arquivo texto. Ao salvar o arquivo texto, a primeira linha do arquivo deve conter a quantidade de números sorteados.

Exercício 06: Considere que um arquivo chamado **exercício.txt** esteja salvo no disco com o seguinte conteúdo:

```
8
20
30
50
1 7
28
47
14
```

A primeira linha do arquivo (8) indica que existem 8 números a serem analisados (20, 30, 50, 1, 7, 28, 47, 14).

Implemente um algoritmo que contenha uma função **chamada abrirArquivo(nome)** que receba no parâmetro o nome do arquivo a ser aberto. Esta função deve ser capaz de abrir o arquivo e preencher um vetor com os valores indicados conforme descrição acima, retornando este vetor preenchido.

Implemente uma função chamada **obterMaior(vetor)** que receba o vetor com os números lidos. Esta função deverá retornar o maior dos números contidos no vetor.

No **programa principal**, solicite o nome do arquivo, obtenha o vetor com os dados e apresente o maior dos números deste vetor (chamando as respectivas funções criadas).

Exercício 07: Considere que no disco exista um arquivo chamado **prova.txt**. Desenvolva um algoritmo que seja capaz de renomear este arquivo para que ele se chame **exercício.txt**.



11. Estrutura de Dados Heterogênea

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Entendimento sobre as estruturas de dados heterogêneas
- Entendimento sobre os conceitos de objeto e atributos (registro e campos)
- Implementação de algoritmos que utilizam estruturas de dados heterogêneas

O conteúdo trabalhado neste capítulo aborda:

- Definir registros (classes) e campos (atributos)
- Criação de variáveis (objetos) do tipo de dado definido pelo programador
- Acessar campos dos registros

Registros

Uma estrutura de dados homogênea (unidimensional **vetor** ou multidimensional **matriz**), também conhecida como Array, permite o armazenamento de uma lista de valores, onde todos os valores pertencem ao mesmo tipo de dado, utilizando-se um único nome de variável. Para se acessar cada valor desta lista, precisamos informar o índice deste array. Por exemplo, podemos ter um vetor de números inteiros, ou um vetor de valores ponto-flutuante, ou mesmo um vetor de Strings.

OBSERVAÇÃO: Na linguagem Python, como as variáveis não são tipadas, um vetor consiste em uma lista de valores, independentemente do tipo de dado de cada valor. Nesta disciplina, adotamos um padrão que serve para a maioria das linguagens de programação.

Quando falamos em estrutura de dados heterogênea, nos referimos a uma estrutura que pode conter diversos valores, e cada valor pode ter um tipo de dado diferente. Como exemplo, uma estrutura pode armazenar um valor do tipo String, outro valor do tipo inteiro, outro valor do tipo ponto-flutuante, etc. É comum chamarmos as estruturas de dados heterogêneas de Registro (em especial em linguagens como Pascal) ou mesmo de estrutura (struct em linguagem C/C++).

Um registro representa um “novo tipo de dado” que é criado pelo programador. Ao criarmos variáveis deste tipo de dado, podemos acessar cada informação deste registro individualmente.

Exemplo: considere um registro (novo tipo de dado) chamado de Data. Uma data possui as seguintes informações: dia, mês e ano.

```
registro Data {
    dia : integer
    mes : integer
    ano : integer
}
```

As variáveis **dia**, **mes** e **ano** são conhecidas como “**campos**” do registro. Desta forma, o registro Data possui três campos.

Após definir um tipo de estrutura (registro), podemos declarar variáveis deste tipo:

```
dataNascimento : Data
```

Desta forma, uma variável chamada dataNascimento foi criada e ela é do tipo Data, ou seja, dentro da variável dataNascimento, temos 3 campos disponíveis para armazenar valores. Para acessar individualmente cada campo, utilizamos o operador ponto . (.):

```
dataNascimento.dia = 05
dataNascimento.mes = 11
dataNascimento.ano = 2019
```

As linguagens de programação atuais são conhecidas como Linguagens Orientadas a Objetos (exemplo: Java, C++, Dart, Python, Object Pascal, etc.). Nestas linguagens, geralmente podemos definir os registros por meio de uma estrutura conhecida como **Classe**.

OBSERVAÇÃO:

os conceitos de **Classes** e **Objetos** utilizados na **Programação Orientada a Objetos** não serão tratados aqui. Usaremos classes/objetos para trabalhar como **registros** e desenvolvimento do **raciocínio lógico** em **Algoritmos**.

Uma **classe** pode ser vista como um novo tipo de dado que possui um conjunto de dados (ou seja, uma espécie de registro) e um conjunto de operações em uma só estrutura (trabalharemos apenas com dados e não com operações neste momento). Chamamos de “**atributo**” todos os **campos** que pertencem a uma classe.

Relembrando:

- Registro → Classe
- Campo do registro → Atributo da classe

Um **objeto** pode ser visto como uma variável que é do tipo de uma classe (chamamos de instância de uma classe).

Relembrando:

- Objeto → Variável que é do tipo de uma Classe.

A implementação de sistemas utilizando registros/classes permite organizar o código e definir funções específicas para tratamento dos dados das classes.

Em Python, podemos criar uma classe conforme o exemplo a seguir.

```
class Data:
    def __init__(self):
        self.atributo1 = valor_inicial
        self.atributo2 = valor_inicial
        self.atributo3 = valor_inicial
```

Considere que precisamos implementar um sistema e, nesta implementação, uma Data precise ser definida, já que podemos ter a Data de Nascimento de uma pessoa, a Data de Matrícula de um aluno, a Data de uma Avaliação na faculdade, a Data do Início das Aulas, a Data do início das férias escolares, entre outras datas. Desta forma, podemos definir Data como uma classe e criarmos diversas variáveis do tipo Data, uma para cada informação desejada. Para todas as datas, temos o dia, o mês e o ano.

```
class Data:
    def __init__(self):
        self.dia = 0
        self.mes = 0
        self.ano = 0

dataNascimento = Data()

print(dataNascimento.dia, "/", dataNascimento.mes, "/", dataNascimento.ano)

dataNascimento.dia = 5
dataNascimento.mes = 11
dataNascimento.ano = 2019

print(dataNascimento.dia, "/", dataNascimento.mes, "/", dataNascimento.ano)
```

No exemplo acima, uma classe chamada **Data** foi criada. Três atributos (dia, mes e ano) foram definidos com os seus valores iniciais (valores padrão).

```
class Data:
    def __init__(self):
        self.dia = 0
        self.mes = 0
        self.ano = 0
```

Uma variável chamada **dataNascimento** foi criada e esta variável é do tipo **Data**, ou seja, **dataNascimento** é um **objeto**. Observe que precisa de **()** após o **nome da classe**.

```
dataNascimento = Data()
```

Podemos acessar cada atributo utilizando o **operador ponto**. Para atribuir valores:

```
dataNascimento.dia = 5
dataNascimento.mes = 11
dataNascimento.ano = 2019
```

Para exibir valores, também utilizamos o operador ponto ao acessar cada valor individualmente:

```
print(dataNascimento.dia)
```

Vetores de Registros

Podemos ter ainda um **vetor** (ou matriz) **de registros** (classes).

Exemplo: precisamos representar um conjunto de alunos de uma faculdade.

Os alunos devem possuir em seu registro o nome e a idade.

```
class Aluno:
    def __init__(self):
        self.nome = ""
        self.idade = 0

listaAlunos = [Aluno()] * 5
```

No exemplo acima, a variável **listaAlunos** é um **vetor** que armazenará valores do tipo **Aluno** e que contém **5** posições, ou seja, para acessar cada aluno individualmente, precisamos informar o seu índice. Por exemplo:

```
listaAlunos[0]
listaAlunos[1]
listaAlunos[2]
```

Para cada posição dentro do vetor, precisamos informar que o conteúdo será um Aluno, portanto, **antes de utilizarmos o vetor** para armazenar dados ou para exibir dados, precisamos inicializar cada aluno dentro de cada posição do vetor:

```

#Preencher o vetor com registros (classes), um aluno por posição no
vetor

for cont in range(0, len(listaAlunos)):
    listaAlunos[cont] = Aluno()

```

A partir do momento que conseguimos acessar o aluno fornecendo o índice do vetor e já exibir objetos em cada índice, podemos utilizar o **operador ponto (.)** para acessar os atributos:

```

listaAlunos[0].nome
listaAlunos[0].idade
listaAlunos[1].nome
listaAlunos[1].idade

```

Exemplo completo:

```

#criar a classe/registro com seus campos/atributos
class Aluno:
    def __init__(self):
        self.nome = ""
        self.idade = 0

#criar um vetor que armazenará 5 alunos
listaAlunos = [Aluno()] * 5

#inicializar cada aluno dentro do vetor
for cont in range(0, len(listaAlunos)):
    listaAlunos[cont] = Aluno()
    listaAlunos[cont].nome = input("Qual o nome do aluno? ")
    listaAlunos[cont].idade = int(input("Qual a idade do aluno? "))

#exibir os dados de cada aluno
for cont in range(0, len(listaAlunos)):
    print(listaAlunos[cont].nome, listaAlunos[cont].idade, "\n")

```

Registros em parâmetros de subalgoritmos

Podemos passar objetos como parâmetro para funções e procedimentos e também retornar objetos nas funções:

```

#criar a classe/registro com seus campos/atributos
class Aluno:
    def __init__(self):
        self.nome = ""
        self.idade = 0

#procedimento que recebe um aluno como parâmetro
#e exibe os dados deste aluno
def exibirAluno(aluno):
    print("Nome: ", aluno.nome)
    print("Idade: ", aluno.idade)

```

```

#função que cria um aluno e retorna o aluno criado
def obterAluno():
    a = Aluno()
    a.nome = input("Qual o nome do aluno? ")
    a.idade = int(input("Qual a idade do aluno? "))
    return a

#criar um vetor que armazenará 5 alunos
listaAlunos =[Aluno()]*5

#inicializar cada aluno dentro do vetor
#com o aluno criado na função
for cont in range(0, len(listaAlunos)):
    listaAlunos[cont] = obterAluno()

#exibir os dados de cada aluno
#enviando o aluno para o procedimento
for cont in range(0, len(listaAlunos)):
    exibirAluno(listaAlunos[cont])

```

Registros em bibliotecas

Da mesma forma que podemos criar variáveis, constantes, procedimentos e funções em nossas próprias bibliotecas, também podemos criar bibliotecas de registros/classes. Este recurso é muito útil para a organização das classes da nossa aplicação e será mais explorado em uma disciplina específica sobre **Programação Orientada a Objetos**.

Como exemplo, considere uma biblioteca que possua uma classe chamada **Produto**. Cada produto precisa de um nome, de um preço de venda e de uma quantidade em estoque. A biblioteca também contém um procedimento que é responsável por exibir um determinado produto em específico, recebido em seu parâmetro. A classe e o procedimento poderiam ser implementados como a seguir:

Arquivo: **produto.py**

```

class Produto:
    def __init__(self):
        self.nome = ""
        self.precoVenda = 0.0
        self.qtde = 0

def exibirProduto(prod):
    print("Nome: ", prod.nome)
    print("Preço de venda: ", prod.precoVenda)
    print("Quantidade em estoque: ", prod.qtde)

```

Para utilizar nossa biblioteca, basta importa-la no nosso algoritmo antes da criação da variável que representará um produto. Em seguida, podemos criar quantos produtos desejarmos, acessando seus atributos individualmente por meio do operador ponto.

Arquivo: exercicio1.py

```
from produto import Produto
from produto import exibirProduto

p1 = Produto()
p1.nome = "Biscoito Passatempo"
p1.precoVenda = 2.49
p1.qtde = 20

p2 = Produto()
p2.nome = "Arroz Integral"
p2.precoVenda = 25.00
p2.qtde = 10

exibirProduto(p1)
exibirProduto(p2)
```

Dica: observe no exemplo acima que a instrução de importação foi

```
from produto import Produto
```

e que na instrução para criar um produto, utilizamos

```
p1 = Produto()
```

Isso faz com que a classe Produto esteja disponível como se fosse local no algoritmo (como se a classe estivesse definida dentro do mesmo arquivo sendo programado).

Se tivéssemos importado apenas o conteúdo do arquivo da biblioteca, então precisaríamos acessar a classe dentro do arquivo:

```
import produto

p1 = produto.Produto()
p1.nome = "Biscoito Passatempo"
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Após resolver os exercícios, aguarde um dia e tente resolvê-los novamente você mesmo, mas agora sem utilizar as respostas e dicas como consulta. Se conseguiu resolver todos os exercícios sozinho, então você está aprendendo o conteúdo!

Exercício 01: Um telefone celular precisa ser representado em um programa de computador. Sabe-se que este telefone possui uma marca, um modelo, um ano de fabricação e um preço. Implemente uma estrutura para representar os telefones celulares. Crie três celulares, solicite os dados ao usuário para cada um e apresente-os na tela.

Exercício 02: Um programa de computador deve ser capaz de armazenar dados sobre as partes de uma casa. Como a casa possui vários cômodos e cada cômodo possui uma porta, precisa-se de uma estrutura capaz de representar as portas da casa. Sabe-se que toda porta deve ter registrado a sua largura, sua altura, seu peso e o seu material (ex: madeira, alumínio, etc.). Crie três objetos que representem portas (ex: porta da sala, porta

do quarto e porta da cozinha). Preencha os dados da porta solicitando-os ao usuário. Em seguida, apresente os dados das portas.

Exercício 03: Escreva um modelo para representar uma lâmpada que está à venda em um supermercado. Que atributos devem ser representados por este modelo?

Exercício 04: Considere que uma lâmpada possua os seguintes estados: apagada e acesa. Implemente uma estrutura capaz de representar o estado da lâmpada.

Implemente um método (função ou procedimento) chamado `ascenderLuz` que receba a lâmpada por parâmetro e faça com que a lâmpada fique acesa. Implemente o método `pagarLuz` que receba a lâmpada por parâmetro e faça com que a lâmpada fique apagada. Implemente o método `exibirLuz` que receba a lâmpada como parâmetro e apresente uma mensagem informando se a lâmpada está acesa ou apagada.

DICA: Em Python, objetos são passados “por referência” nos parâmetros dos métodos. Ou seja, se um objeto for recebido no parâmetro de um método, todas as alterações feitas neste objeto dentro do método serão automaticamente modificadas no objeto original fornecido ao chamar o método.

Exercício 05: Imagine uma lâmpada que possa ter três estados: apagada, acesa e meia-luz. Estes estados são representados por um valor de 0 a 100, onde 0 representa o estado apagado, de 80 a 100 representa o estado aceso e de 1 a 79 representa o estado meia-luz.

Implemente um programa capaz de representar uma lâmpada conforme o enunciado.

Implemente uma função chamada `criarLampada` que crie uma lâmpada e solicite a intensidade da lâmpada, retornando o objeto criado.

Implemente um procedimento chamado `exibirLampada` que recebe uma lâmpada como parâmetro e apresente na tela se a lâmpada está apagada, acesa ou em meia-luz.

Exercício 06: Uma agência bancária deve ser capaz de representar contas bancárias. Sabe-se que para as contas bancárias precisa-se conhecer a data de abertura (formada pelo dia, mês e ano), o cliente (formado pelo nome e pela data de nascimento) e pelo saldo.

Implemente um programa que seja capaz de representar uma conta bancária. Faça uma análise de quais classes (registros) serão necessários e a relação entre estes registros.

Implemente um método chamado `lerData` que crie um objeto de uma `Data` e faça a leitura do dia, mês e ano, retornando o objeto criado.

Implemente um método chamado `lerCliente` que crie um objeto de um `Cliente` e faça a leitura do nome e da data de nascimento, retornando o objeto criado.

Implemente um método chamado `lerConta` que crie uma conta bancária e faça a leitura de todos os valores da conta, retornando o objeto criado. Implemente um método chamado `exibirContaBancaria` que receba uma conta bancária como parâmetro e exiba na tela todos os dados da conta bancária.

No algoritmo principal, chame os métodos necessários para que uma conta bancária possa ser obtida e exibida na tela.

Exercício 07: Uma pessoa possui um nome, uma data de nascimento (dia, mês e ano) e uma hora de nascimento (hora e minuto). Implemente um programa que seja capaz de representar uma pessoa (utilizando registros).

Exercício 08: Uma televisão deve ser capaz de armazenar seu status (ligada ou desligada) e o número do canal a ser exibido.

Implemente um registro capaz de representar esta situação.

Implemente um método chamado `aumentarCanal` que receba uma televisão como parâmetro e aumente o canal em uma unidade (ir para o próximo canal), caso a TV esteja ligada.

Implemente um método chamado `diminuirCanal` que receba uma televisão como parâmetro e diminua o canal em uma unidade (ir para o canal anterior), caso a TV esteja ligada.

Implemente um método capaz de exibir o estado atual da TV (se ela está ligada e o canal atual, ou se ela está desligada). Considere que a TV possa ter canais representados de 0 a 99 (canais fora desta faixa não podem ser exibidos pela TV).

Exercício 09: (continuação ex. 8) Adicione na classe `Televisão` os atributos que representam a marca da TV e o número de polegadas que a TV possui.

Implemente um método chamado `criarTV` que crie um objeto da televisão e retorne o objeto criado.

Implemente um método chamado `lerDadosTV` que receba como parâmetro uma TV e faça a leitura dos dados iniciais da TV solicitando-os ao usuário.

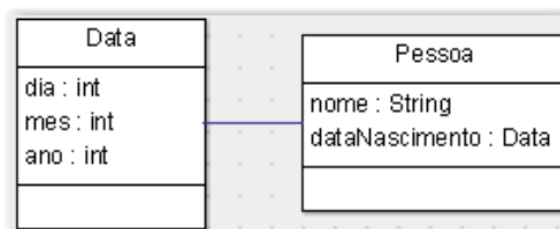
Implemente um método chamado `exibirTV` que exiba todos os dados da TV.

Exercício 10: Implemente uma classe chamada `Matematica`. Esta classe deve possuir dois atributos representando dois valores numéricos e um atributo representando uma operação matemática.

Implemente um método chamado `getCalculo` que receba um objeto do tipo `Matematica` como parâmetro, efetue a operação matemática (atributo que representa a operação) com os dois valores numéricos do objeto e retorne o resultado.

Teste seus códigos.

Exercício 11: Considere as seguintes classes:



O diagrama representa uma pessoa que possui um nome e uma data de nascimento formada pelo dia, mês e ano.

Implemente as classes conforme o diagrama.

Implemente um método chamado `calcularIdade` que receba uma pessoa por parâmetro, calcule e retorne a idade da pessoa.

Implemente um método chamado `lerDados` que receba uma pessoa por parâmetro e faça a leitura dos dados desta pessoa.

Implemente um método chamado `exibirDados` que receba uma pessoa por parâmetro e exiba os dados desta pessoa, incluindo a sua idade atual.

No algoritmo principal, crie três objetos do tipo `Pessoa` e faça a leitura dos dados dos objetos:

- Albert Einstein (nascido em 14/3/1879)
- Isaac Newton (nascido em 4/1/1643)
- Você (seus dados pessoais)

Por fim, exiba os dados das três pessoas.



12. Acessando Banco de Dados

Objetivos de Aprendizagem

Este capítulo tem como objetivo:

- Entendimento sobre como acessar um Banco de Dados
- Executar consultas nas tabelas do banco de dados

O conteúdo trabalhado neste capítulo aborda:

- Acessar banco de dados
- Executar consultas em tabelas do banco de dados para inclusão, alteração e exclusão de dados
- Executar consultas em tabelas do banco de dados para seleção e exibição de dados

Banco de Dados

Utilizamos sistemas de banco de dados para armazenar os dados de nossas aplicações. Seja um sistema acadêmico que mantém todos os dados da instituição de ensino, assim como um sistema bancário ou sistema de vendas, tanto web quanto desktop, fazem uso de bancos de dados.

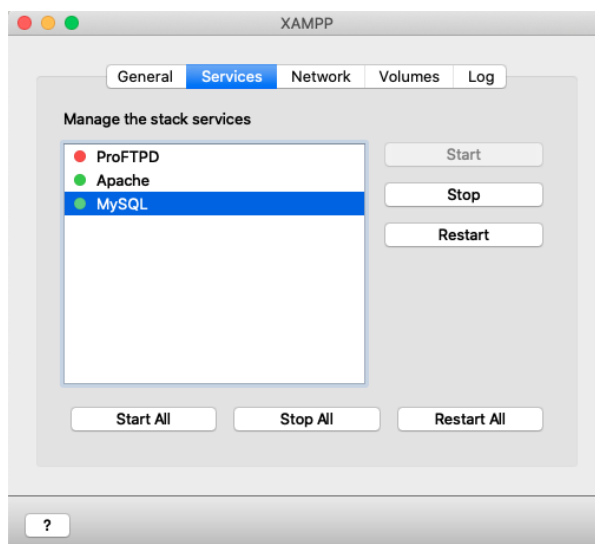
A grande vantagem da utilização de sistemas de banco de dados se refere à segurança dos dados armazenados. Também temos outras vantagens, tais como rapidez na atualização dos dados e na obtenção de informações destes dados, manter a integridade dos dados, evitar a redundância e a inconsistência dos dados.

Os Sistemas de Gerenciamento de Banco de Dados (SGBD) relacionais são sistemas que permitem o uso de linguagens específicas para a manipulação destes bancos de dados, tal como a linguagem Structured Query Language (SQL). Usando linguagem SQL, podemos fornecer um comando ao SGBD e ele cuida de atualizações e retorno de consultas automaticamente, sem a necessidade de manipular os dados diretamente em arquivos em disco.

Para que um algoritmo em Python possa lidar com bancos de dados, precisamos basicamente de:

- Importar a biblioteca responsável por fornecer funções para manipular o banco de dados. Cada SGBD possui uma biblioteca específica.
- Abrir a conexão com o banco de dados
- Criar um cursor que é o responsável por executar os comandos SQL no banco de dados, permitindo navegar e manipular os registros do banco de dados
- Fechar a conexão com o banco de dados que estava aberta

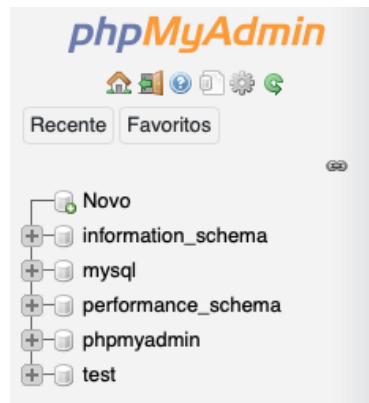
Vamos utilizar o SGBD **MySQL**. Para simplificar a instalação e as ferramentas necessárias para trabalhar com MySQL, basta instalar o software XAMPP³. Após a instalação, o XAMPP possui um Painel de Controle que permite iniciar/finalizar os serviços do servidor web Apache e do banco de dados MySQL.



Após inicializar o servidor Apache e o servidor do MySQL, podemos utilizar o software phpMyAdmin para manipular nossos bancos de dados. Para acessar o phpMyAdmin, basta abrir um navegador web (Internet Explorer, Firefox, Google Chrome, etc.) e acessar a página <http://localhost/phpmyadmin/>. A página responsável por gerenciar o banco de dados será aberta.

Para criar um novo banco de dados, podemos clicar no item “**Novo**” no painel lateral da esquerda.

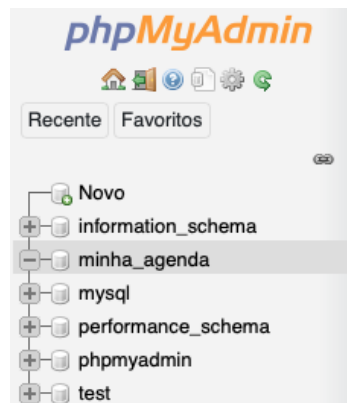
³ Disponível em https://www.apachefriends.org/pt_br/index.html



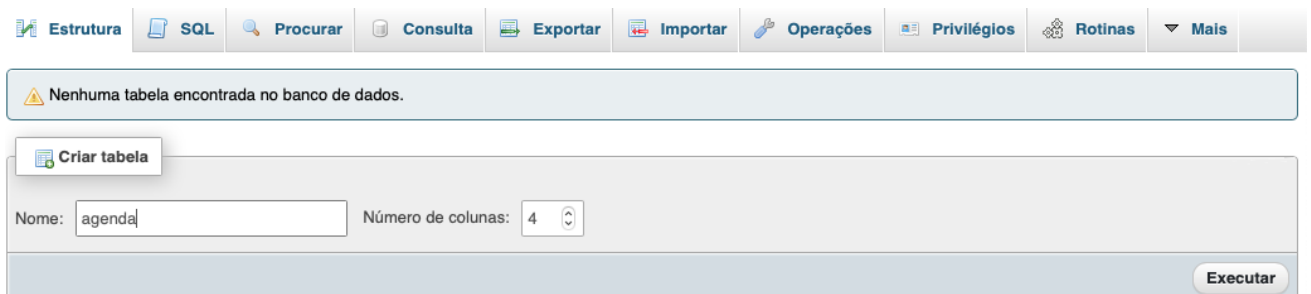
Em seguida, podemos informar o nome do nosso banco de dados a ser criado e clicar no botão Criar. Por exemplo: **minha_agenda**.



Observe no painel lateral da esquerda que o banco de dados criado será exibido.



Com o banco de dados criado, podemos criar nossas tabelas. Na área central, informe o nome da tabela **agenda**. A tabela armazenará 4 campos: campo de identificação **id**, o **nome** do contato, o endereço de **email** e **data de nascimento**. Desta forma, preencha o nome da tabela e a quantidade de campos, clicando no botão Executar.



Preencha os dados para cada campo:

- Campo **id**
 - Tipo: int
 - Índice: Primary

- Auto Incremento (AI): marcar a caixa
- Campo **nome**
 - Tipo: Varchar
 - Tamanho: 100
- Campo **email**
 - Tipo: Varchar
 - Tamanho: 50
- Campo **datanasc**
 - Tipo: Date

Nome	Tipo	Tamanho/Valores	Padrão	Colação	Atributos	Nulo	Índice
id	INT		Nenhum			<input type="checkbox"/>	PRIMARY <input checked="" type="checkbox"/>
nome	VARCHAR	100	Nenhum			<input type="checkbox"/>	
email	VARCHAR	50	Nenhum			<input type="checkbox"/>	
datanasc	DATE		Nenhum			<input type="checkbox"/>	

Clicar no botão Salvar.

A tabela é então criada dentro do banco de dados.

The screenshot shows the phpMyAdmin interface with the 'agenda' table selected. The table structure is as follows:

#	Nome	Tipo	Colação	Atributos	Nulo	Padrão	Comentários	Extra	Ação
1	id	int(11)			Não	Nenhum		AUTO_INCREMENT	Alterar, Eliminar, Mais
2	nome	varchar(100)	utf8mb4_general_ci		Não	Nenhum			Alterar, Eliminar, Mais
3	email	varchar(50)	utf8mb4_general_ci		Não	Nenhum			Alterar, Eliminar, Mais
4	datanasc	date			Não	Nenhum			Alterar, Eliminar, Mais

Com o banco de dados criado, podemos manipulá-lo utilizando comandos SQL por meio da linguagem Python.

Usaremos a biblioteca **MySQLdb** para conectar no banco de dados MySQL (existem outras bibliotecas disponíveis). Para instalar esta biblioteca, basta acessar o Prompt e Comandos (ou Terminal) do sistema operacional e instalá-la com o comando

pip install mysql-connector-python

Conectando no banco de dados MySQL

Para conectar em banco de dados MySQL, primeiramente precisamos importar a biblioteca necessária que disponibiliza de funções próprias para manipulação deste SGBD. Para isso, basta importar **mysql.connector**.

```
import mysql.connector
```

Quando as tabelas possuem dados do tipo Date ou DateTime, podemos importar a biblioteca

```
from datetime import date
```

Dica: não esqueça de colocar todas as instruções de importação no início do algoritmo.

Para criar a conexão com o banco de dados, precisamos informar o local onde o SGBD está instalado (um endereço IP do servidor de banco de dados), o nome do usuário e a senha para conectar no banco de dados e o nome do banco de dados que desejamos ter acesso.

Considerando que o SGBD está no mesmo computador onde estamos trabalhando, o servidor é o **localhost**. O usuário padrão de acesso é o **root** sem nenhuma senha. O banco de dados que criamos se chama **minha_agenda**.

```
#importar o driver do MySQL
import mysql.connector

#para tratar datas
from datetime import date

#conectar no banco de dados
banco = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database = "minha_agenda"
)

if(banco):
    print("Conectado com sucesso!")
else:
    print("Erro na conexão")
```

Finalizando a conexão com o banco de dados

Sempre que conectamos em um banco de dados para realizar alguma operação de atualização (inserir novos registros, alterar registros existentes ou excluir registros existentes, bem como efetuar consultas nos registros), a conexão deve ser aberta. O recurso consome memória e, por isso, precisa ser finalizado ao fim do algoritmo ou após a última operação com o banco de dados.

Para encerrar a conexão aberta com o banco de dados, utilizamos a função **nomeDoBanco.close()**.

```
#importar o driver do MySQL
import mysql.connector
#conectar no banco de dados
banco = mysql.connector.connect(host="localhost", user="root",
    password="", database = "minha_agenda")
#fechar a conexão com o banco de dados
banco.close()
```

Executando comandos SQL

Para executarmos comandos na linguagem SQL, precisamos primeiramente obter um **cursor**. O cursor representa uma transação qualquer no banco de dados (transação corresponde a qualquer operação de leitura ou escrita em um banco de dados).

Para se obter uma transação no banco de dados, precisamos guardar em uma variável o resultado da função **nomeDoBanco.cursor()**. Todo cursor aberto também consome recursos e deve ser finalizado no fim do algoritmo.

```
#importar o driver do MySQL
import mysql.connector

#conectar no banco de dados
banco = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database = "minha_agenda"
)

cursor = banco.cursor()

cursor.close()
banco.close()
```

Para executarmos consultas em SQL, utilizamos sempre o cursor criado. A consulta será sempre executada com o método:

```
cursor.execute("Comando na linguagem SQL", (lista de valores))
```

O comando SQL será uma string e todos os valores a serem inseridos deverão ser substituídos por %s. A sequência de valores em %s da string deverá ser fornecida na lista de valores

É importante lembrar que todas as instruções para inserir, alterar ou excluir dados no banco de dados devem ser sempre salvas após serem executadas, para manter o estado do banco de dados. Para isso, precisamos chamar a função **commit()** do banco de dados:

```
banco.commit()
```

Exemplo 1: Inserindo dados no banco de dados

```
import mysql.connector
from datetime import date

banco = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database = "minha_agenda"
)

#cria um cursor
cursor = banco.cursor()

#variáveis com valores para serem inseridos no BD
nome = "Isabela da Silva"
email = "isa@empresa.com"
data_nascimento = date(2002, 12, 3)

#comando SQL e lista de valores para serem inseridos
sql = ("INSERT INTO agenda (nome, email, datanasc) VALUES (%s, %s, %s)")
valores = (nome, email, data_nascimento)
```

```

#executa a SQL com os valores
cursor.execute(sql, valores)

#salva os dados no banco de dados
banco.commit()

cursor.close()
banco.close()

```

Exemplo 2: Inserindo vários registros de uma única vez.

Precisamos executar a instrução em SQL diversas vezes. Para isso, ao invés de chamar a função **execute**, utilizamos a função **executemany**.

```

import mysql.connector
from datetime import date

banco = mysql.connector.connect(host="localhost", user="root",
                                password="", database = "minha_agenda")

#cria um cursor
cursor = banco.cursor()

#comando SQL e lista de valores para serem inseridos
sql = ("INSERT INTO agenda (nome, email, datanasc) VALUES (%s, %s, %s)")
valores = [
    ("Mario", "mario@empresa.com", date(2000, 10, 4)),
    ("Carolina", "carol@empresa.com", date(2001, 9, 15)),
    ("Jair", "jair@empresa.com", date(2008, 12, 28)),
    ("Amanda", "amanda@empresa.com", date(2010, 1, 7))
]

#executa a SQL com os valores
cursor.executemany(sql, valores)

#salva os dados no banco de dados
banco.commit()

cursor.close()
banco.close()

```

Dica: para obter o **id** da última inserção no banco de dados, pode-se utilizar **cursor.lastrowid**

Dica: para obter o número de linhas atualizadas no banco de dados (número de linhas inseridas, número de linhas alteradas, número de linhas excluídas, ou mesmo o número de linhas retornadas em uma consulta, podemos utilizar **cursor.rowcount**.

Exemplo 3: selecionando os registros da tabela

Utilizamos a função **fetchall()** do cursor para obtermos os resultados retornados pela consulta.

Podemos percorrer o resultado obtido, linha por linha, para processar os dados ou para exibi-los na tela.

Podemos acessar cada coluna individualmente, para cada registro, como se fosse um vetor, fornecendo o índice do campo para o resultado de cada linha obtida como resposta. A sequência numérica dos índices depende da sequência obtida no comando SQL.

```
import mysql.connector
from datetime import date

banco = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database = "minha_agenda"
)

#cria um cursor
cursor = banco.cursor()

#comando SQL e lista de valores para serem inseridos
sql = ("SELECT * FROM agenda ORDER BY nome")

#executa a SQL com os valores
cursor.execute(sql)

#obter os resultados da seleção
resultados = cursor.fetchall()

print("Temos um total de ", cursor.rowcount, " resultados")

for linha in resultados:
    print(linha)
    print("ID: ", linha[0])
    print("    Nome: ", linha[1])
    print("    Email: ", linha[2])
    print("    Data de Nascimento: ", linha[3].strftime("%d/%m/%Y"))
    print()

cursor.close()
banco.close()
```

Vamos Praticar?

Tente resolver os seguintes algoritmos sozinho, individualmente. Caso não consiga, solicite auxílio aos colegas e aos professores, mas não obtenha as respostas. Tente conseguir dicas de como resolver.

Após resolver os exercícios, aguarde um dia e tente resolvê-los novamente você mesmo, mas agora sem utilizar as respostas e dicas como consulta. Se conseguiu resolver todos os exercícios sozinho, então você está aprendendo o conteúdo!

Exercício 01: Crie um banco de dados chamado **sistemavendas** e uma tabela chamada **produto**. Sabe-se que um produto possui um código de identificação, um nome, um valor de venda e uma data de validade.

Implemente uma biblioteca chamada **sistemaVendas** que contenha funções para incluir um novo produto no banco de dados e alterar um determinado produto no banco de dados em que estas funções recebam no parâmetro os dados do produto para serem incluídos e alterados. Crie também uma função para excluir um produto do banco de dados em que receba o código de identificação de um produto em seu parâmetro. Também disponibilize uma função que seja capaz de retornar uma lista de todos os produtos cadastrados.

No algoritmo principal, crie um menu de opções com as alternativas conforme exemplo:

MENU DE OPÇÕES

1. Incluir novo produto
2. Alterar produto existente
3. Excluir produto existente
4. Listar todos os produtos

OPÇÃO: ____

Se o usuário escolher a opção 1, então o sistema deve solicitar os dados do produto e chamar a função responsável pela inclusão dos dados no banco de dados.

Se o usuário escolher a opção 2, então o sistema deve solicitar os dados do produto e chamar a função para alterar os dados no banco de dados.

Se o usuário escolher a opção 3, então o sistema deve solicitar apenas o código de identificação do produto a ser excluído e chamar a função para excluir aquele produto do banco de dados.

Se o usuário escolher a opção 4, então o sistema deve listar na tela todos os resultados de produtos cadastrados.

Para qualquer outro valor de opção, o sistema deve exibir na tela uma mensagem dizendo que a opção é inválida.

Exercício 02: Crie um banco de dados que contenha ao menos duas tabelas. Popule o banco de dados utilizando qualquer ferramenta de conexão com banco de dados. Crie um algoritmo que seja capaz de exibir na tela resultados de consultas do tipo SELECT que possuam a cláusula INNER JOIN.