

IPOG

Lógica de Programação

EDITORIA IPOG

Todos os direitos quanto ao conteúdo desse material didático são reservados ao(s) autor(es). A reprodução total ou parcial dessa publicação por quaisquer meios, seja eletrônico, mecânico, fotocópia, de gravação ou outros, somente será permitida com prévia autorização do IPOG.

IP5p Instituto de Pós-Graduação e Graduação – IPOG

Lógica de Programação. / Autor: Thiago Santana Lemes.

240f. :il.

ISBN:

1. Algoritmo 2. Tecnologia 3. Lógica 4. Linguagem de programação 5. Processamento de dados.

CDU: 005

[illegible]

IPOG

Sede

Instituto de Pós Graduação e Graduação

<http://www.ipog.edu.br>

Av. T-1 esquina com Av. T-55 N. 2.390
- Setor Bueno - Goiânia-GO. Telefone
(0xx62) 3945-5050

SUMÁRIO

APRESENTAÇÃO	6
OBJETIVOS	7
UNIDADE 1 CONCEITOS INICIAIS DE LÓGICA DE PROGRAMAÇÃO	9
1.1 AMBIENTE DE DESENVOLVIMENTO NO GOOGLE COLAB	10
1.2. CARACTERIZAÇÃO DE UM ALGORITMO, TIPOS DE DADOS, CONSTANTES E COMENTÁRIOS	11
1.3. OPERADORES ARITMÉTICOS, RELACIONAIS E LÓGICOS	14
1.4. COMANDOS DE ENTRADA E SAÍDA	17
UNIDADE 2 ESTRUTURAS DE DECISÃO E REPETIÇÃO	19
2.1. ESTRUTURA CONDICIONAL SIMPLES	20
2.2. ESTRUTURA CONDICIONAL COMPOSTA	21
2.3. ESTRUTURAS DE REPETIÇÃO	22
2.4. EXPRESSÕES ARITMÉTICAS	27
UNIDADE 3 ESTRUTURAS DE DADOS	29
3.1. VETORES UNIDIMENSIONAIS	30
3.2. VETORES MULTIDIMENSIONAIS	33
UNIDADE 4 MODULARIZAÇÃO	37
4.1. REAPROVEITAMENTO DE CÓDIGO E DEFINIÇÃO DE UMA FUNÇÃO	38
4.2. IMPLEMENTAÇÃO DE CÁLCULOS MATEMÁTICOS POR MEIO DE FUNÇÕES	40
4.3. UTILIZANDO FUNÇÕES DENTRO DE FUNÇÕES	41
UNIDADE 5 PROJETO DE ALGORITMO	44
5.1. ESTUDO DE PROBLEMAS	45

5.2. DESENVOLVIMENTO DE PROJETOS DE ALGORITMOS	45
5.3. EXERCÍCIOS.....	63
FINALIZAR.....	65
Sobre o autor	66
Referências Bibliográficas	67

APRESENTAÇÃO

Seja bem-vindo ou bem-vinda à disciplina de Lógica de Programação.

É com grande entusiasmo que iniciamos esta jornada juntos em busca do entendimento dos fundamentos que regem a arte da programação. Neste curso, exploraremos os princípios básicos que são essenciais para qualquer programador, independentemente de sua área de atuação. Desde os conceitos mais elementares até os desafios mais complexos, estaremos aqui para guiar vocês em cada passo do caminho. Não se preocupem se tudo parecer um pouco confuso no início. A lógica de programação é como um idioma novo que estamos aprendendo a dominar, e com prática e dedicação, tenho certeza de que todos alcançarão um excelente nível de proficiência.

Estejam abertos para perguntas, debates e desafios. A troca de conhecimento é essencial para o nosso crescimento individual e coletivo. Não hesitem em compartilhar suas ideias e dúvidas, pois é assim que aprendemos melhor. Lembrem-se sempre: o sucesso vem da persistência, da curiosidade e do trabalho árduo. Estou aqui não apenas como seu professor, mas também como um parceiro em sua jornada de aprendizado.

Desejo a todos um semestre repleto de descobertas, conquistas e, acima de tudo, muito aprendizado. Estou animado para embarcar nesta jornada com vocês. Vamos começar!

Boa leitura e estudos!

Prof. Dr. Thiago Santana Lemes

OBJETIVOS

OBJETIVO GERAL

Equipar os estudantes com uma compreensão robusta dos conceitos fundamentais de lógica de programação, incluindo estruturas de controle, estruturas de dados básicas, e princípios de algoritmos, capacitando-os a analisar problemas e desenvolver soluções eficazes através da programação.

OBJETIVOS ESPECÍFICOS

- Introduzir conceitos básicos de programação, incluindo variáveis, tipos de dados, operadores aritméticos e lógicos;
- Ensinar o uso de estruturas de controle de fluxo, como condicionais (*if-else*) e laços de repetição (*for*, *while*);
- Apresentar estruturas de dados básicas, como *arrays* (vetores e matrizes);
- Fomentar a habilidade de desenvolver algoritmos para resolver problemas;
- Inculcar boas práticas de programação, como a escrita de código limpo, comentários, uso de convenções de nomes, e desenvolvimento de códigos modulares e reutilizáveis.

Conheça como esse conteúdo foi organizado!

Unidade 1: Introdução à Lógica de Programação

Unidade 2: Estruturas de decisão e repetição

Unidade 3: Estruturas de dados

Unidade 4: Modularização

Unidade 5: Projeto de algoritmo

UNIDADE 1 CONCEITOS INICIAIS DE LÓGICA DE PROGRAMAÇÃO

É com grande prazer que damos início a essa jornada de aprendizado e descobertas. Nesta unidade, teremos a oportunidade de explorar os conceitos iniciais do fascinante universo da lógica de programação.

OBJETIVOS DA UNIDADE 1

Ao final dos estudos, você deverá ser capaz de:

- Definição de conceitos básicos de lógica.
- Configuração do ambiente.

1.1 AMBIENTE DE DESENVOLVIMENTO NO GOOGLE COLAB

O Google fornece uma plataforma de desenvolvimento para projetos de ciência de dados e inteligência artificial chamada Colab. Esse ambiente é gratuito para utilização, podendo fornecer recursos computacionais extras por meio de pagamento. A Figura 1.1 abaixo mostra a página inicial que pode ser acessada por meio do link <https://colab.research.google.com>.

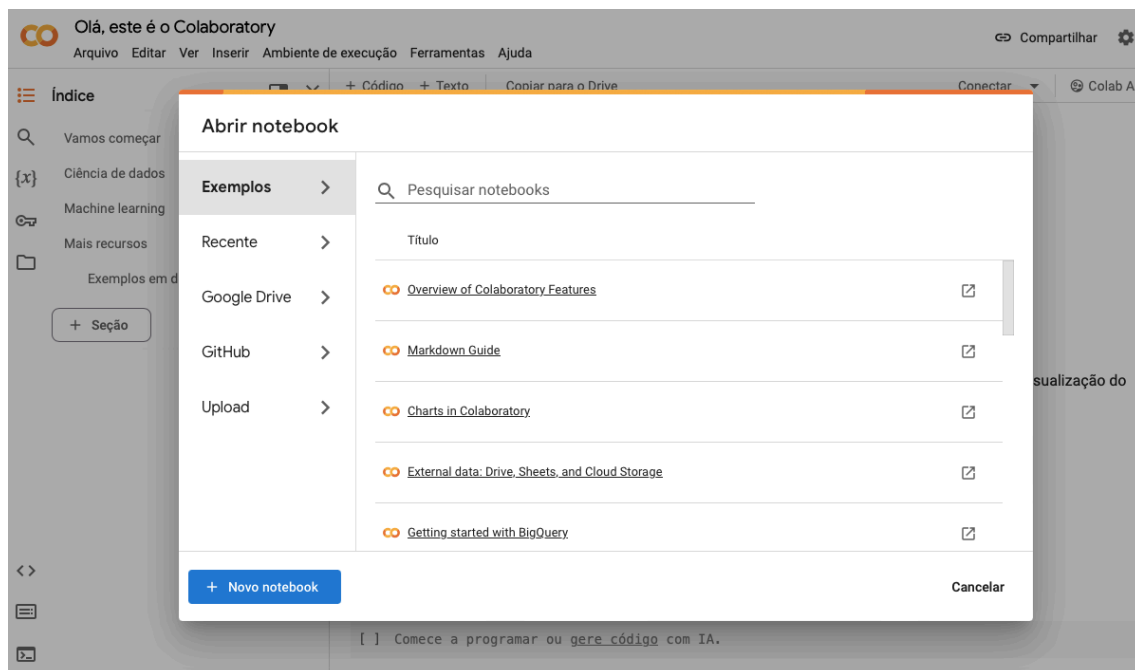
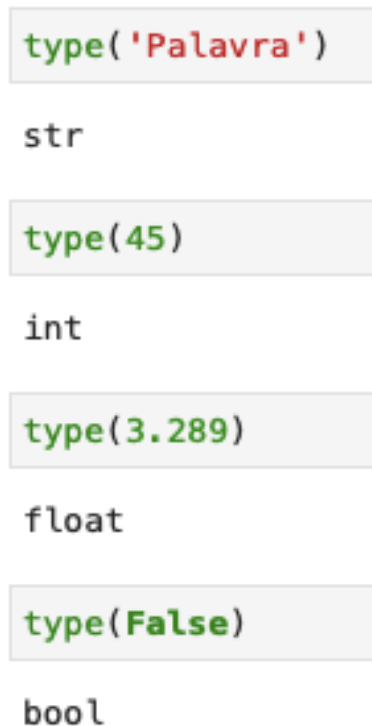


Figura 1.1: Página inicial do Google Colab

A plataforma possui diversos exemplos e tutoriais e todos estão acessíveis já na aba Exemplos como mostrado na Figura 1.1. Para iniciar um projeto basta clicar no botão azul Novo notebook. Todos os códigos que serão apresentados nesse texto podem ser desenvolvidos no ambiente do Google Colab. A vantagem é que as principais bibliotecas da linguagem Python já estão instaladas nesse ambiente, facilitando muito o início do desenvolvimento de um projeto. Obviamente é possível instalar bibliotecas caso elas não estejam disponíveis de início no ambiente.

1.2. CARACTERIZAÇÃO DE UM ALGORITMO, TIPOS DE DADOS, CONSTANTES E COMENTÁRIOS

Um algoritmo pode ser entendido como uma sequência de passos ou regras previamente definidas para executar uma tarefa. Imagine que ir ao supermercado pode ser visto como um algoritmo. Existem alguns passos que normalmente se repetem a cada nova compra: definir a lista de compras, entrar no carro, dirigir até o mercado, fazer as compras, passar no caixa e pagar a conta, retornar para casa. Obviamente podem existir etapas distintas, mas no geral, é algo próximo a isso. A lógica nos conduz na implementação e interpretação de algoritmos. Em geral, no mundo da programação, existem vários tipos de dados, e variações de acordo com a linguagem de programação. Especificamente em Python, para os números, temos os inteiros e os pontos flutuantes, que são *int* e *float* respectivamente. Para as letras, caracteres em geral e palavras (texto) temos o tipo *str*. O tipo *bool* é caracterizado por definir o que é verdadeiro e o que é falso. A Figura 1.2 abaixo ilustra o comando *type* do Python analisando alguns tipos.



```
type('Palavra')  
str  
  
type(45)  
int  
  
type(3.289)  
float  
  
type(False)  
bool
```

Figura 1.2: Tipos de dados em Python

```
palavra = 'laranja'

numero = 34

escolha = True

type(palavra)

str

type(numero)

int

type(escolha)

bool

numero = 5.6

type(numero)

float
```

Figura 1.3: Definição de variáveis e seus tipos em Python

As três primeiras células da Figura 1.3 definem as variáveis “palavra”, “numero” e “escolha”. Perceba o uso do símbolo “=” na atribuição de valor à uma variável (Mueller, 2020). Na sequência o comando *type* verifica os tipos de cada uma (*str*, *int*, *bool*). Veja que a penúltima célula atribui o valor 5.6 à variável “numero” e a última célula verifica seu tipo novamente, que agora passa a ser *float*. O Python se destaca por sua simplicidade e facilidade no aprendizado. As definições são feitas sem o uso de “;”, comum em quase

todas as linguagens de programação. Além disso, o tipo da variável é automaticamente inferido pelo Python, não sendo necessária uma declaração explícita, e quando é alterado, a própria linguagem se encarrega de compreender isso (veja o que acontece com “numero”). Por convenção, constantes em Python são declaradas utilizando letras maiúsculas, veja: $\text{PI} = 3.14$, $\text{E} = 2.718$.

Muitas vezes se faz necessário inserir comentários no código, de maneira que o Python ignore tais instruções. Para isso basta colocar # e escrever na frente. É possível também fazer comentário em blocos utilizando três aspas duplas em sequência “””. A Figura 1.4 abaixo ilustra os comentários no código.

```
#Aqui é necessário fazer um comentário de exemplo  
variavel_numero = 56.7  
variavel_nome = 'Nome da Pessoa'  
"""  
Além disso, é possível colocar os comentários  
dentro de blocos.  
"""
```

Figura 1.4: Comentários em Python

1.3. OPERADORES ARITMÉTICOS, RELACIONAIS E LÓGICOS

Os operadores aritméticos em Python são:

- Adição: +
- Subtração: -
- Multiplicação: *
- Divisão: /
- Divisão inteira: //
- Módulo (resto da divisão): %
- Potenciação: **

Esses operadores são basicamente o que se vê em matemática e, portanto, autoexplicativos com exceção dos três últimos. A Divisão inteira divide um número por outro e trunca o resultado para o número inteiro mais próximo, o módulo divide um número por outro e apresenta no resultado o resto da divisão e a potenciação, que também se vê na matemática, calcula o resultado de um número elevado ao outro. A Figura 1.5 a seguir ilustra a utilização dos operadores. Obviamente o uso de 4 e 2 foi proposital (qualquer número pode ser utilizado) até para verificar o efeito dos operadores de divisão, divisão inteira e resto da divisão.

<code>4+2</code>
16
<code>4-2</code>
-1
<code>4*2</code>
56
<code>4/2</code>
2.0
<code>4//2</code>
2
<code>4%2</code>
0
<code>4**2</code>
16

Figura 1.5: Operadores aritméticos

Os operadores relacionais são:

- Igual a: `==`
- Diferente de: `!=`
- Maior que: `>`
- Maior ou igual a: `>=`
- Menor que: `<`
- Menor ou igual a: `<=`

Eles basicamente estabelecem a relação entre as parcelas ao fazerem as comparações. A Figura 1.6 mostra alguns exemplos de utilização dos operadores relacionais.

```
: 'nome' == 'palavra'
: False

: 2 != 8
: True

: 1 > 2
: False

: 4 >= 4
: True

: numero1 = 9
: numero2 = 10

: numero1 < numero2
: True

: 8 <= 8
: True
```

Figura 1.6: Exemplos de utilização de operadores relacionais

Os operadores lógicos são:

- E lógico: and
- Ou lógico: or
- Não lógico: not

Utilizar os operadores lógicos em conjunto com os demais dá muita flexibilidade para modelar diversas situações. Veja a Figura 1.7 a seguir alguns exemplos de utilização.


```
| var1 = 'fruta'
| var2 = 'árvore'

| var1 == 'fruta' and var2 == 'manga'

| False

| var1 == 'fruta' or var2 == 'manga'

| True

| var1 == 'fruta' and var2 == 'árvore'

| True

| not var1 == 'manga' and var2 == 'árvore'

| True
```

Figura 1.7: Exemplos de utilização dos operadores lógicos

Obviamente todas as situações apresentadas nas figuras anteriores são muito simples e tem propósito didático. Situações mais complexas irão surgir ao longo do texto e o uso desses operadores fará ainda mais sentido.

1.4. COMANDOS DE ENTRADA E SAÍDA

A construção de algoritmos naturalmente demanda por entradas e saídas de dados. Em Python contamos com dois comandos básicos nesse sentido: *print* e *input*.

- *print*: Imprime na tela uma informação
- *input*: Solicita a entrada uma informação ao usuário

Os comandos *print* e *input* fazem parte de um conjunto de palavras reservadas que fazem parte da linguagem de programação Python. Tais

palavras são utilizadas para acionar determinadas ações ou implementações. Ao longo do curso veremos diversas delas.

Veja a Figura 1.8 abaixo para ter uma ideia melhor da utilização desses comandos.

```
variavel_nome = input('Insira um nome:')  
variavel_numero = float(input('Insira um número:'))  
print('\n')  
print(f'O nome digitado foi: {variavel_nome}\n')  
print(f'O número digitado foi: {variavel_numero}')
```

```
Insira um nome: Thiago  
Insira um número: 20
```

```
O nome digitado foi: Thiago
```

```
O número digitado foi: 20.0
```

Figura 1.8: Utilização de comandos *input* e *print*

O comando *input* recebe uma informação digitada pelo usuário e ele precisa de um argumento do tipo *str* que simplesmente possui uma mensagem explicando o que deve ser inserido. O comando *input* retorna *str* e por isso, na segunda linha, foi utilizado o comando *float()* passando o *input* entre parêntesis para que o retorno seja convertido em um número do tipo *float*. A terceira linha utiliza o `\n` como um caractere para produzir espaço em branco entre as linhas. Finalmente as duas últimas linhas da célula possuem os comandos *print* para imprimir o nome e o número digitados. Veja que uma letra *f* é inserida antes das aspas. Isso se deve a uma funcionalidade do Python conhecida como interpolação de *strings*. Exatamente no local do texto no qual uma variável deve aparecer, ela é inserida entre chaves, como visto nos dois últimos comandos *print*.

UNIDADE 2 ESTRUTURAS DE DECISÃO E REPETIÇÃO

Tivemos uma ideia básica a respeito do funcionamento da linguagem Python além de alguns exemplos simples tratando dos aspectos lógicos. A partir de agora iniciaremos as estruturas de decisão e repetição. Nessa parte o raciocínio lógico aparece de forma mais explícita e é possível produzir exemplos mais complexos que exigem mais do raciocínio.

Espero que o leitor esteja empolgado para continuar a jornada e mergulhar um pouco mais fundo no mundo da tecnologia.

OBJETIVOS DA UNIDADE 2

Ao final dos estudos, você deverá ser capaz de:

- Implementar código contendo blocos lógicos com aspectos de decisão e repetição
- Compreender a modelagem de situações mais complexas

2.1. ESTRUTURA CONDICIONAL SIMPLES

Diversas situações reais em nosso cotidiano exigem a tomada de decisão e na lógica de programação não é diferente. As instruções que viabilizam a tomada de decisão no algoritmo são o *if* e o *else* (Manzano & Oliveira, 2019). O *if*, em tradução literal “se” avalia se uma condição é satisfeita, e caso positivo executa o que vem logo abaixo. Caso contrário o que está imediatamente abaixo do *else* (“senão”) será executado. Vamos iniciar essa parte imaginando a situação em que um sistema de cobrança avalia se o pagamento foi realizado até a data de vencimento ou não para liberar um código de pagamento. Caso a data seja inferior ou igual à data de vencimento a cobrança é feita sem o acréscimo de juros e, caso contrário, 2.5% de juros irão incidir sobre o valor original. Veja o trecho de código abaixo que implementa a lógica para resolver essa situação.

```
data_vencimento = 28
data_atual = 29

if data_atual <= data_vencimento:
    print('O valor a ser pago é R$200,00\n')
else:
    juros = 200*0.025
    novo_valor = 200 + juros
    print(f'O valor a ser pago é {novo_valor}\n')
```

No trecho acima temos a data atual (pagamento) superior à data de vencimento e por isso teremos a incidência de juros. A terceira linha possui a instrução *if* que verifica se a data atual é menor ou igual à data de vencimento e caso isso seja verdade, o código passa para a próxima linha que imprime o valor original de R\$200,00. Caso contrário, a instrução *else* apresenta o que deve ser feito. Uma simples expressão matemática calcula o percentual e na sequência aplica a soma que então é impressa na tela como o resultado.

Observe que em Python não há necessidade do uso de “;”. Além disso, veja a indentação (espaço em branco) logo abaixo do *if* e do *else*. Python entende que tudo que está imediatamente alinhado com essa indentação está

dentro do comando acima. Além disso é necessário inserir ":" ao final de cada linha de *if* e *else*.

Veja mais um exemplo abaixo:

```
nome_usuario = 'Thiago'
horario_limite = 18

if nome_usuario == 'Thiago' and horario_limite <= 18:
    print('Acesso liberado!')
else:
    print('Acesso negado!')
```

No trecho de código acima além de fazer uma comparação com um texto (*str*) uma segunda condição é necessária (horário limite). O acesso só é liberado caso as duas condições sejam satisfeitas ao mesmo tempo, caso contrário o acesso é negado. As possibilidades são inúmeras no caso de estruturas condicionais e cabe ao leitor estudar e se aprofundar.

2.2. ESTRUTURA CONDICIONAL COMPOSTA

Para expandir o conceito visto na seção anterior a respeito de estrutura condicional iremos aumentar um pouco a complexidade da tomada de decisão. Imagine novamente a situação do sistema de cobrança. Caso a data de pagamento seja inferior ou igual à data de vencimento, a cobrança é feita sem o acréscimo de juros, caso o pagamento esteja em atraso de até cinco dias, uma taxa de 2,5% irá incidir sobre o valor original. Finalmente, caso o atraso seja superior a cinco dias, uma taxa de 5,25% irá incidir sobre o valor principal. Para resolver essa situação precisamos montar uma estrutura que seja capaz de avaliar três cenários distintos, sendo eles, pagamento em dias, atraso de até cinco dias e atraso superior a cinco dias. Aqui entra uma nova parte da lógica de estrutura condicional com o *elif*. Essa palavra é uma abreviação para *else if* e significa que existe uma outra possibilidade. Veja o trecho de código abaixo.

```
data_vencimento = 25
data_atual = 31
```

```

if data_atual <= data_vencimento:
    print('O valor a ser pago é R$200,00\n')
elif data_atual - data_vencimento <= 5:
    juros = 200*0.025
    novo_valor = 200 + juros
    print(f'O valor a ser pago é {novo_valor}\n')
else:
    juros = 200*0.0525
    novo_valor = 200 + juros
    print(f'O valor a ser pago é {novo_valor}\n')

```

Agora temos uma terceira possibilidade para a decisão. Dessa forma, se estiver dentro do prazo correto (antes do vencimento) não há cobrança de taxa, se o atraso for de até 5 dias, a taxa é 2,5% e todas as outras possibilidades vão para a terceira parte. É possível colocar diversas condições, portanto, poderia existir o prazo entre 5 e 10 dias de atraso, entre 10 e 15 dias de atraso e por aí vai. Tudo depende do problema que está sendo implementado.

2.3. ESTRUTURAS DE REPETIÇÃO

Estruturas de repetição são extremamente úteis e fazem parte da rotina diária dos profissionais que utilizam linguagens de programação. Aqui, iremos aprender como utilizar *for* e *while* que são as palavras reservadas da linguagem Python para implementar os blocos de repetição (Lambert, 2022).

FOR

Vamos apresentar um exemplo para facilitar o entendimento. Imagine uma situação na qual tem-se um conjunto de anotações de um supermercado que representam a quantidade de dias restantes para determinados produtos perderem a validade. O gerente preocupado com a situação solicita uma investigação na lista no intuito de obter tudo que irá vencer em 7 dias ou menos. O trecho de código abaixo ilustra duas implementações. Na primeira, todos os valores são impressos e na segunda, apenas os valores menores ou iguais a sete.

```
lista_de_vencimento_de_produtos = [5, 12, 31, 60, 8, 7, 45, 98, 6, 108]

for item in lista_de_vencimento_de_produtos:
    print(f'Quantidade de dias para o vencimento do produto - {item}')
```

A primeira linha do trecho de código acima introduz uma estrutura de dados comum em Python, conhecida como lista. Ela é definida por colchetes “[]”. Dessa forma, temos a `lista_de_vencimento_de_produtos` contendo os dias restantes para o vencimento de cada produto. Para fins de simplicidade do código, não iremos inserir os nomes dos produtos. Na segunda linha do trecho de código, o comando `for` aparece e em tradução literal significa para. Na prática essa linha quer dizer o seguinte: para cada item da `lista_de_vencimento_de_produtos` faça. A instrução encerra a linha com “:” e tudo que está abaixo do comando `for`, indentado será compreendido como um comando a ser executado. Nesse caso, os itens da lista são impressos na tela com uma mensagem. Observe o conceito de interpolação de *strings* já mencionado anteriormente com o item entre chaves na terceira linha do código. Veja o resultado na Figura 2.1 abaixo:

```
1 lista_de_vencimento_de_produtos = [5, 12, 31, 60, 8, 7, 45, 98, 6, 108]
2
3 for item in lista_de_vencimento_de_produtos:
4     print(f'Quantidade de dias para o vencimento do produto - {item}')
```

Quantidade de dias para o vencimento do produto - 5
Quantidade de dias para o vencimento do produto - 12
Quantidade de dias para o vencimento do produto - 31
Quantidade de dias para o vencimento do produto - 60
Quantidade de dias para o vencimento do produto - 8
Quantidade de dias para o vencimento do produto - 7
Quantidade de dias para o vencimento do produto - 45
Quantidade de dias para o vencimento do produto - 98
Quantidade de dias para o vencimento do produto - 6
Quantidade de dias para o vencimento do produto - 108

Figura 2.1: Implementação do comando `for` para impressão de valores em uma lista

A Figura 2.2 abaixo implementa a situação descrita inicialmente no texto que é verificar produtos com data de vencimento igual ou inferior a sete.

```
1 lista_de_vencimento_de_produtos = [5, 12, 31, 60, 8, 7, 45, 98, 6, 108]
2
3 for item in lista_de_vencimento_de_produtos:
4     if item <= 7:
5         print(f'Produto próximo do vencimento: {item}')
```

Produto próximo do vencimento: 5
Produto próximo do vencimento: 7
Produto próximo do vencimento: 6

Figura 2.2: Implementação do comando *for* juntamente com *if* para testar produtos de acordo com os dias restantes para vencimento

Na Figura 2.2 o comando *for* (também conhecido como laço de repetição) é combinado com uma estrutura de decisão (*if*) para avaliar se cada elemento da lista é menor ou igual a 7. Veja que apenas os números que se encaixam nessa condição são impressos como resultado.

WHILE

Outra maneira de avaliar situações de maneira repetida é o comando *while*, que em tradução literal significa enquanto. De forma simples, quer dizer, enquanto algo acontecer, repita as instruções. Veja a implementação da situação dos dias de vencimento dos produtos utilizando *while* no trecho de código abaixo (Figura 2.3).

```
1 lista_de_vencimento_de_produtos = [5, 12, 31, 60, 8, 7, 45, 98, 6, 108]
2
3 indice = 0
4
5 while indice < len(lista_de_vencimento_de_produtos):
6     if lista_de_vencimento_de_produtos[indice] <= 7:
7         print(f'Produto próximo do vencimento: {lista_de_vencimento_de_produtos[indice]}')
8     indice += 1
```

Produto próximo do vencimento: 5
Produto próximo do vencimento: 7
Produto próximo do vencimento: 6

Figura 2.3: Implementação do comando *while* juntamente com *if* para testar produtos de acordo com os dias restantes para vencimento

Para utilizar o *while* é necessário definir um critério de parada, o que torna o comando bastante flexível. Nesse caso o critério foi o tamanho da lista de valores, ou seja, enquanto o índice definido na linha 3 for menor do que o tamanho da lista o Python mantém a execução ativa. Na linha 8 o índice é incrementado (valor atual + 1) em 1 a cada rodada. Em resumo, esse bloco da Figura 2.3 avalia os elementos da lista de valores, um a um (cada rodada do *while*), e quando são menores ou iguais a 7, são impressos, na sequência a variável índice é incrementada em 1 e o fluxo segue até que a variável índice tenha o valor 10 (quantidade de valores na lista, ou seja, seu tamanho).

Uma observação importante é que a maioria das linguagens de programação, incluindo Python, o índice de estruturas como listas iniciam em 0 e não em 1. O índice (index) é uma maneira de achar em qual posição o elemento está na lista. Na linha 6 da Figura 2.3 é mostrada a variável `lista_de_vencimento_de_produtos` com dois colchetes na frente e a variável índice dentro “[índice]”. Essa forma de representação é o que indica a posição de um elemento na lista. Portanto, se a variável índice, em determinado momento tiver o valor 4, indica que estamos interessados em obter o elemento da lista que está na posição 4. Em nosso exemplo, o elemento que ocupa a posição 4 na lista é o 8 (Figura 2.3 na linha 1), lembrando que o índice começa em zero.

Um outro exemplo de utilização do *while* é exibido no trecho de código na Figura 2.4 abaixo.

```

1 flag = True
2 indice = 0
3 lista_de_vencimento_de_produtos = [5, 12, 31, 60, 8, 7, 45, 98, 6, 108]
4
5 while flag == True:
6     print(f'Quantidade de dias para o vencimento do produto - {lista_de_vencimento_de_produtos[indice]}')
7     indice += 1
8
9     if indice == len(lista_de_vencimento_de_produtos):
10        flag = False

```

Quantidade de dias para o vencimento do produto - 5
 Quantidade de dias para o vencimento do produto - 12
 Quantidade de dias para o vencimento do produto - 31
 Quantidade de dias para o vencimento do produto - 60
 Quantidade de dias para o vencimento do produto - 8
 Quantidade de dias para o vencimento do produto - 7
 Quantidade de dias para o vencimento do produto - 45
 Quantidade de dias para o vencimento do produto - 98
 Quantidade de dias para o vencimento do produto - 6
 Quantidade de dias para o vencimento do produto - 108

Figura 2.4: Implementação do comando *while* juntamente com *if* e uma variável do tipo *bool* para exibir os elementos em uma lista

Na prática, o código exibido na Figura 1.9 anteriormente é suficiente para exibir os valores da lista, mas para fins ilustrativos, a Figura 1.12 apresenta uma outra maneira de fazer. Aqui, uma variável do tipo *bool* é criada na linha 1 e o *while* verifica se essa variável permanece como verdadeira (*True*) para continuar sua execução (linha 5). A linha 9 analisa o tamanho da lista, e em caso de a variável *índice* ter o mesmo valor que o tamanho da lista, a condição é verificada e a variável *flag* é alterada para *False* (linha 10). O restante do código contém a lógica de impressão de valores contidos na lista. Em termos práticos, esse código dá voltas e faz mais do que é necessário para exibir os valores, mas vale reforçar, foi apenas um exemplo para ilustrar a combinação das estruturas já vistas até aqui no texto.

2.4. EXPRESSÕES ARITMÉTICAS

Expressões aritméticas no Python seguem exatamente a mesma ideia já vista nas aulas de matemática na escola (ensino fundamental). Python compreende a precedência de operadores e obviamente é possível utilizar parêntesis para completar a lógica desejada no que tange às operações que devem ser realizadas antes. Relembre a fórmula de Bháskara na Equação 2.1 abaixo:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{Equação 2.1}$$

O trecho de código abaixo implementa as operações necessárias para solucionar essa equação. Aqui faremos uso da biblioteca *math*, que adiciona capacidades ao Python.

```
import math

a = 4
b = 2
c = -1

delta = b**2 - 4*a*c

x1 = (-b + math.sqrt(delta))/(2*a)
x2 = (-b - math.sqrt(delta))/(2*a)
```

No trecho acima, a primeira linha importa a biblioteca *math*, os elementos a, b e c de uma equação do segundo grau são definidos. A seguir, o termo que fica dentro da raiz quadrada (Equação 2.1) pode ser calculado de maneira separada (delta). Veja que nenhum parêntesis precisa ser inserido aqui, pois o Python sabe a potenciação e o produto devem ser resolvidos antes da subtração.

As duas últimas linhas (x1 e x2) exibem o cálculo final (parte positiva e parte negativa) com o auxílio de parêntesis para especificar que a tudo que está dentro será dividido por 2a. A Figura 2.5 abaixo ilustra alguns trechos de código que fazem uso de expressões aritméticas no Python.

```
1 2 * 4 - 9 + 5 / 3
0.6666666666666667

1 2 * 4 - (9 + 5) / 3
3.333333333333333

1 5 ** 3 + 5
130

1 12 / 3 - 10 + math.sqrt(9) + 4 ** 3
61.0
```

Figura 2.5: Algumas expressões aritméticas em Python

UNIDADE 3 ESTRUTURAS DE DADOS

Nesse momento já somos capazes de aplicar diversas operações e utilizar várias estruturas lógicas para tomada de decisão e para repetições de instruções. Vamos agora aprofundar na utilização de estruturas de dados, que são blocos chave para construção de sistemas mais complexos.

OBJETIVOS DA UNIDADE 3

Ao final dos estudos, você deverá ser capaz de:

- Implementar lógicas que façam uso de vetores e matrizes
- Realizar operações com essas estruturas

3.1. VETORES UNIDIMENSIONAIS

Vetores unidimensionais, também conhecidos como *arrays*, são estruturas de dados focadas em organizar coleções de informações. Em Python a maneira mais simples e direta de organizar dados dessa forma é utilizar as listas (a seção 2.3 apresentou o uso de listas de forma muito simplificada). As listas podem ser definidas por colchetes, como por exemplo: `[3, 4, 5]`, `['banana', 'morango']`, `[True, False, False, True]`.

A ideia dessas estruturas pode ser vista na Figura 3.1 abaixo na qual temos oito posições numeradas de 0 a 7. Em cada posição um valor é armazenado.

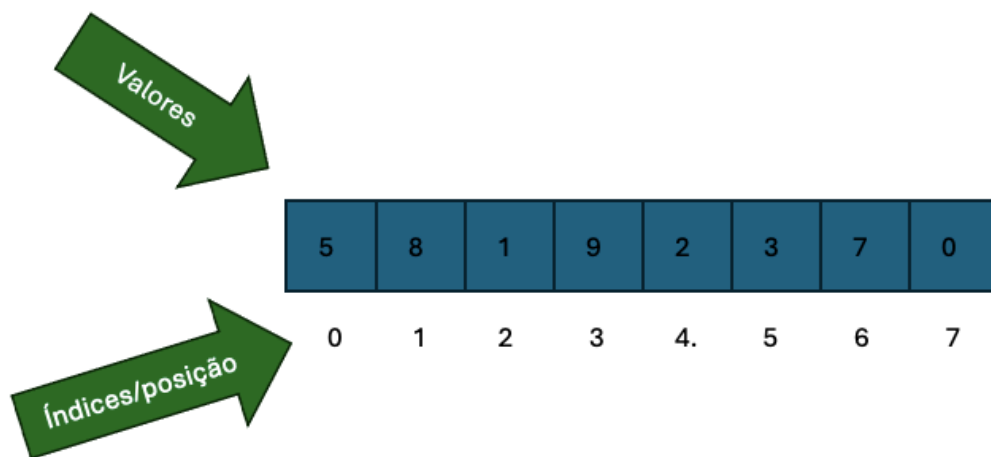


Figura 3.1: Estrutura de um vetor unidimensional (*array*)

Para se ter acesso a um valor da lista basta utilizar seu índice (index). Suponha que a lista acima seja nomeada como “numeros”. Dessa forma, poderia ser definida da seguinte maneira: `numeros = [5, 8, 1, 9, 2, 3, 7, 0]`. Para acessar o número 3 por exemplo, basta fazer `numeros[5]` (veja o uso dos colchetes com o índice dentro). A lista é uma estrutura muito dinâmica em Python possuindo diversas operações e manipulações bastante facilitadas. O objetivo desse texto não é aprofundar em todas as características das listas, mas dar uma base para o leitor se aperfeiçoar posteriormente. Para adicionar um novo elemento à lista, basta utilizar o método `append()`, passando entre os parêntesis o valor a ser incluído. Por exemplo, para incluir o número 13, faz-se `numeros.append(13)` e para remover um elemento utiliza-se o método `remove()`, `numeros.remove(1)`. Vale ressaltar que o método `remove()` retorna um erro no caso em que o elemento solicitado a ser removido não existe na lista. Para alterar um elemento da lista, por exemplo, inserir no lugar do número 5 na primeira posição o número 13, basta fazer `numeros[0] = 13` (observe que a primeira posição é representada pelo índice 0).

Todas as operações aritméticas citadas anteriormente na seção 2.1 podem ser utilizadas com vetores (listas). O detalhe fica por conta de acessar os elementos por meio dos índices e então realizar as operações. O trecho de código apresentado na Figura 3.2 abaixo mostra a implementação do cálculo da média dos valores em uma lista.

```
1 notas = [6.5, 6.0, 7.8, 7.5]
2 soma = 0
3 for i in range(len(notas)):
4     soma += notas[i]
5 media = soma/len(notas)
6 print(f'A média é: {media}')
```

A média é: 6.95

Figura 3.2: Cálculo da média dos valores contidos em uma lista

A linha 1 define a lista com os números, a linha 2 define uma variável soma iniciando em 0 que servirá como auxílio para calcular a média. Nas linhas 3 e 4 um laço for é utilizado para percorrer os elementos da lista. Observe o comando `len()` que calcula o tamanho da lista (nesse caso é 4). Portanto aqui, o índice `i` que é apenas uma variável auxiliar para utilização do for, irá variar de 0 a 3, graças ao comando `range()`. A variável soma é incrementada em cada rodada pelo valor das notas em cada posição. Na prática, significa que a soma irá acumular cada valor da lista ($6.5 + 6.0 + 7.8 + 7.5$). Finalmente a linha 5 calcula a média dividindo o valor da soma dos elementos pelo total de elementos na lista e a linha 6 imprime o resultado.

Vale destacar uma observação muito importante em relação ao comando for e sua utilização para percorrer elementos de listas. Na seção 2.3 foi apresentada uma maneira de percorrer a lista que obtém o elemento contido nela de maneira direta e na Figura 3.2 acima, os elementos são acessados por meio de suas posições (índice). A Figura 3.3 ajuda a compreender melhor o conceito ao apresentar outra maneira de reproduzir o cálculo da média dos valores na lista.

```
1 notas = [6.5, 6.0, 7.8, 7.5]
2 soma = 0
3 for i in notas:
4     soma += i
5 media = soma/len(notas)
6 print(f'A média é: {media}')
```

A média é: 6.95

Figura 3.3: Cálculo da média dos valores contidos em uma lista utilizando os índices

Comparando a metodologia na Figura 3.2 e na Figura 3.3 é possível ver que a grande diferença é que na primeira ao fazer uso dos métodos *range()* e *len()* o laço de repetição for acessa os valores da lista de acordo com seus índices, ou seja, *notas[i]*, uma vez que a variável *i* inicia em 0 e termina em 4, enquanto isso, na Figura 3.3 o acesso aos valores da lista é feito de forma direta (linha 3). Ainda vale destacar que o Python, ao trabalhar com valores em uma faixa (*range*), sempre para um elemento antes do último, ou seja, se temos uma faixa de 0 a 4, o último elemento acessado será o 3, o que faz sentido, visto que uma lista com tamanho quatro inicia seu índice em 0 e termina em 3.

3.2. VETORES MULTIDIMENSIONAIS

Os vetores multidimensionais, também conhecidos como matrizes, são um formato muito utilizado na computação. É muito natural representar informações em formato matricial, visto que qualquer tabela do cotidiano apresenta dados e é facilmente interpretada. Em Python é possível combinar listas para se obter o comportamento de matrizes (duas dimensões – linha e coluna). A Figura 3.4 a seguir mostra um trecho de código que cria uma lista com duas dimensões.

```
1 notas = [[7.8, 9.9], [6.0, 6.6], [5.4, 7.6]]
2
3 for l in range(3):
4     print(notas[l])
```

[7.8, 9.9]
[6.0, 6.6]
[5.4, 7.6]

```
1 for l in range(3):
2     for c in range(2):
3         print(notas[l][c])
```

7.8
9.9
6.0
6.6
5.4
7.6

Figura 3.4: Criação de uma lista com duas dimensões e impressão de valores

A primeira célula da Figura 3.4 acima, na linha 1 define uma matriz, que na prática é uma lista contendo outras listas. Veja que os colchetes externos definem uma lista maior que é composta pelas listas menores contendo os valores. As linhas 3 e 4 da primeira célula utilizam o *for* para imprimir em formato matricial, mais natural para nós. Além disso, a segunda célula utiliza dois laços *for* para imprimir os valores da matriz, sendo o primeiro *for* para percorrer as linhas (0 a 3) e o segundo *for* para percorrer as colunas (0 a 2). Perceba a utilização de dois índices na frente de *notas* na linha 3 da segunda célula. O primeiro índice indica a linha e o segundo a coluna. A título de exemplo, pode-se imaginar a matriz de notas da seguinte maneira: as linhas representando alunos diferentes e as colunas representando as disciplinas português e matemática.

Para adicionar ou remover elementos nesse formato, utilizando os métodos *append()* e *remove()* é necessário lidar com as listas menores dentro da lista maior, o que pode ser um pouco problemático. Para a alteração (edição) de valores, basta encontrar a posição por meio dos índices de linha e coluna e atribuir um novo valor (por exemplo: `notas[1][1] = 9.9`).

Vamos verificar agora a média de cada coluna da matriz `notas`, lembrando que a primeira coluna se refere à português e a segunda coluna matemática.

```
1 notas = [[7.8, 9.9], [6.0, 6.6], [5.4, 7.6]]
2
3 soma_portugues = 0
4 soma_matematica = 0
5
6 for l in range(3):
7     soma_portugues += notas[l][0]
8     soma_matematica += notas[l][1]
9
10 media_portugues = soma_portugues/3
11 media_matematica = soma_matematica/3

1 print(round(media_portugues,2))
6.4

1 print(round(media_matematica,2))
8.03
```

Figura 3.5: Cálculo da média dos valores por coluna na matriz

A Figura 3.5 acima ilustra o uso de uma estrutura de repetição (`for`) para percorrer as linhas da matriz e somar os elementos. As linhas 7 e 8 da primeira célula incrementam os valores das somas para as disciplinas. Veja que a linha 7 soma os valores da primeira coluna (índice 0) e a linha 8 soma os valores da segunda coluna (índice 1). As linhas 10 e 11 calculam as médias para cada coluna. Nas duas células finais o comando *print()* é utilizado para exibir os resultados com o auxílio do comando *round()* que arredonda o resultado (aqui em 2 casas decimais de acordo com o argumento passado logo após a média).

Da mesma maneira que foi descrito no caso de vetores unidimensionais, aqui para os multidimensionais, todas as operações aritméticas são perfeitamente aplicáveis desde que os elementos sejam acessados da maneira correta por meio de seus índices. O exemplo de média da Figura 3.5 é uma das diversas possibilidades de se operar com matrizes.

Nesse ponto do material vale destacar que listas não são a única maneira de utilizar estruturas de dados em Python. Existem algumas bibliotecas que implementam funcionalidades excepcionais para desempenhar tarefas com vetores e matrizes, dentre elas, destaco a *Numpy* (Harris et al., 2020) que é uma biblioteca especialista em álgebra linear. Essa biblioteca é capaz de fazer diversas operações matemáticas e facilita bastante o trabalho e por isso é amplamente utilizada pelos cientistas de dados.

UNIDADE 4 MODULARIZAÇÃO

Chegamos no ponto em que boa parte dos conceitos de algoritmos e lógica de programação foram explorados. Agora é hora de juntar todos esses elementos de forma a tornar os códigos mais padronizados. Nessa unidade, exploraremos os princípios da modularização de códigos. Tornar os códigos modulares facilita muito sua manutenção e seu entendimento, além de deixar os algoritmos mais organizados.

OBJETIVOS DA UNIDADE 4

Ao final dos estudos, você deverá ser capaz de:

- Implementar algoritmos modulares por meio de funções
- Utilizar as funções para realizar operações e tarefas específicas

4.1. REAPROVEITAMENTO DE CÓDIGO E DEFINIÇÃO DE UMA FUNÇÃO

Em diversos casos ao desenvolver algoritmos certos tipos de tarefas são repetitivas e aparecem diversas vezes ao longo do código. A repetição de instruções que fazem a mesma coisa pode ser vista como um desperdício de tempo e de código. Criar uma função que execute uma tarefa e acionar essa função sempre que necessário é mais inteligente, visto que propicia uma organização melhor no código além de economizar linhas escritas e facilitar a alteração da lógica em questão. Uma segunda maneira de enxergar a questão da modularização é a divisão de tarefas complexas em várias tarefas menores. Dessa forma, fica mais fácil fazer reparos no código e entender a lógica como um todo. A título de exemplo, imagine um sistema que calcula uma taxa referente a um pagamento. Essa taxa é utilizada em várias partes do sistema e por isso, caso o cálculo fosse escrito em todos esses pontos separadamente, imagine a dor de cabeça para alterar no caso de mudança do percentual dessa taxa. Cada ponto do código deveria ser editado. O primeiro e mais comum problema nesse caso é o esquecimento de edição em um determinado ponto do sistema. Por outro lado, se uma função para realizar o cálculo for implementada, e chamada nesses diversos pontos do sistema, em caso de alteração do percentual da taxa, basta mudar dentro da função, uma vez que sua chamada permaneceria a mesma. Fazendo uma outra analogia, a função é como um motor, que será modificado em seu interior, o restante do carro não se interessa pelo motor em si, apenas faz uso dele.

Em Python, uma função pode ser definida pela palavra reservada *def* seguida pelo nome da função e os parêntesis que irão delimitar os argumentos solicitados pela função. A Figura 4.1 abaixo ilustra a definição e utilização de uma função simples para calcular a soma de 3 números.

```
1 def somar_numeros(numero1, numero2, numero3):  
2  
3     soma = numero1 + numero2 + numero3  
4  
5     return soma
```

```
1 soma = somar_numeros(5, 1, 9)
```

```
1 print(soma)
```

15

Figura 4.1: Definição de uma função que soma três números e retorna o resultado

A primeira linha da primeira célula na Figura 4.1 define a função e tudo que está entre os parêntesis são parâmetros que devem ser fornecidos à função para que ela possa ser utilizada. A linha 3 realiza o cálculo, que na prática é a tarefa principal dessa função. Finalmente na linha 5 temos a palavra reservada *return* que indica que a função irá retornar a soma calculada. Uma função não necessariamente precisa ter um retorno, ela pode simplesmente realizar uma tarefa e imprimir o resultado. A segunda célula faz a chamada à função, passando três números para que ela realize a soma (5, 1 e 9). Finalmente, a última célula imprime o resultado da função que foi armazenado na variável soma. Um detalhe importante a esclarecer é que tudo que está dentro da função só existe quando ela executa sua tarefa, ou seja, se por acaso, após o cálculo da soma dos números não houvesse o retorno, não seria possível obter o resultado pois. Além disso, a variável soma definida dentro da função é independente da variável soma da segunda célula na qual a função foi chamada. Na verdade, o resultado poderia ter qualquer outro nome, soma é obviamente mais intuitivo.

4.2. IMPLEMENTAÇÃO DE CÁLCULOS MATEMÁTICOS POR MEIO DE FUNÇÕES

Funções são excelentes ferramentas para implementar cálculos matemáticos e organizar o racional. Imagine o cálculo de área das diversas figuras geométricas existentes. Por exemplo, uma das maneiras estudadas na escola para calcular a área de um triângulo é por meio da Equação 4.1 abaixo.

$$A = \frac{b \cdot h}{2} \quad \text{Equação 4.1}$$

Nessa equação b é o comprimento da base e h é o valor da altura do triângulo. Um outro exemplo interessante é o cálculo da área de um círculo que pode ser realizado por meio da Equação 4.2 abaixo.

$$A = \pi r^2 \quad \text{Equação 4.2}$$

Nessa equação π é a famosa constante matemática que possui o valor aproximado de 3,14 e r é o comprimento do raio. A Figura 4.2 apresenta um trecho de código que implementa as funções.

```
1 import math
2
3 def calcular_area_do_triangulo(base, altura):
4
5     area = (base * altura) / 2
6
7     return area
8
9 def calcular_area_do_circulo(raio):
10
11     area = math.pi * raio ** 2
12
13     return area
14
15 area_triangulo = calcular_area_do_triangulo(3, 6)
16 area_circulo = calcular_area_do_circulo(2)
17
18 print(area_triangulo)
19 print('\n')
20 print(round(area_circulo,2))
21
22 9.0
23
24 12.57
```

Figura 4.2: Implementação de funções para cálculo de área de figuras geométricas

A linha 1 da primeira célula mostrada na Figura 4.2 importa a biblioteca *math*. A função que calcula a área do triângulo é definida da linha 3 até a linha 7 e a função que calcula a área do círculo vai da linha 9 até a linha 13. Observe as operações matemática realizadas nas linhas 5 e 11 da primeira célula, com destaque para a linha 11 que faz uso da biblioteca *math* e define o pi por meio de *math.pi*. A segunda célula faz a chamada às duas funções (linha 1 e linha 2) armazenando os resultados dos cálculos das áreas nas variáveis *area_triangulo* e *area_circulo* respectivamente. A última célula imprime os resultados. Um ponto que chama a atenção é a nomenclatura das funções. Perceba que não pode haver espaços e por isso o uso do símbolo *_* para separar as palavras. Além disso, não se utilizam acentos ou cedilha em nomes de funções.

4.3. UTILIZANDO FUNÇÕES DENTRO DE FUNÇÕES

Funções podem ser utilizadas umas dentro das outras. Isso é uma prática muito comum. Aproveitando os exemplos construídos na seção 4.2, vamos implementar uma função um pouco mais complexa que utiliza as outras funções para cálculo de área. Imagine uma função mais abrangente que calcula a área de uma figura geométrica dado seu nome. Veja a Figura 4.3 abaixo.

```

1 import math
2
3 def calcular_area_do_triangulo(base, altura):
4     area = (base * altura) / 2
5     return area
6
7 def calcular_area_do_circulo(raio):
8
9     area = math.pi * raio ** 2
10
11     return area
12
13 def calcular_area(nome_figura, base=0, altura=0, raio=0):
14
15     if nome_figura == 'triangulo':
16         if base != 0 and altura != 0:
17             area = calcular_area_do_triangulo(base, altura)
18         else:
19             area = 0
20             print('As dimensões do triângulo não foram informadas!')
21     elif nome_figura == 'circulo':
22         if raio != 0:
23             area = calcular_area_do_circulo(raio)
24         else:
25             area = 0
26             print('As dimensões do círculo não foram informadas!')
27     else:
28         area = 0
29
30     return area
31
32

```

Figura 4.3: Implementação de uma função que utiliza duas outras funções

A partir da linha 15 na exibida na Figura 4.3 é implementada a função `calcular_area` que é mais abrangente e solicita o nome da figura e suas dimensões. Com base nisso ela faz a chamada à uma das duas outras funções implementadas anteriormente. Obviamente essa função é mais complexa e exige um aprofundamento maior do leitor, mas nada impossível. Em primeiro lugar, na linha 15, a definição da função faz uso de elementos padrão (*default*) para as dimensões. Isso garante que se a chamada for para calcular a área do círculo, os parâmetros de base e altura não precisam ser preenchidos, não causando um erro por falta de parâmetros a serem inseridos. Além disso, a função utiliza estruturas de decisão, já explicadas anteriormente, para validar o nome da figura (linha 17 e linha 23) e qual o cálculo correto a ser feito.

Além disso, uma vez escolhida a figura, é necessário testar se as dimensões fornecidas são diferentes de 0. Veja, não faz sentido calcular área de uma figura que não possui dimensões. Outra questão é a inversão de parâmetros, no caso de solicitar o nome círculo e passar as dimensões de triângulo. Todos esses detalhes necessitam de validação e os comandos *if*, *elif* e *else* ajudam a tratar. As linhas 22 e 28 imprimem mensagens nos casos em que há falta de dimensões para calcular a área das figuras. Os testes para verificar se as dimensões são diferentes de zero (!=) são feitos nas linhas 18 e 24.

Esse uso de funções torna o código bastante organizado e modular, uma vez que deixa tarefas bem delimitadas para cada função. O leitor deve avançar nos estudos de funções para conseguir implementar lógicas ainda mais complexas e obviamente úteis à sua rotina.

UNIDADE 5 PROJETO DE ALGORITMO

Nessa unidade daremos sequência ao estudo da lógica de programação de forma a consolidar todo o conhecimento visto nas unidades anteriores. A ideia é utilizar as estruturas aprendidas, criar módulos e implementar algoritmos um pouco mais complexos.

OBJETIVOS DA UNIDADE 5

Ao final dos estudos, você deverá ser capaz de:

- Implementar algoritmos mais complexos que resolvam problemas
- Aplicar diversos conceitos de lógica de programação de maneira mais integrada

5.1. ESTUDO DE PROBLEMAS

O estudo de lógica de programação e algoritmos culmina inevitavelmente na solução de diversos tipos de problemas e situações. É comum a implementação de soluções ser precedida por uma etapa de modelagem na qual são discutidas metodologias e abordagens. Em especial para algoritmos e linguagens de programação existe uma etapa de construção de diagramas que são bastante úteis para visualizar o fluxo de operação do sistema. Isso não será abordado nesse texto, mas recomenda-se que o leitor procure materiais a respeito.

5.2. DESENVOLVIMENTO DE PROJETOS DE ALGORITMOS

Para ilustrar um primeiro exemplo, vamos juntar alguns raciocínios das unidades anteriores. Na seção 4.2 algumas funções foram implementadas para calcular a área de figuras planas. Vamos expandir um pouco o racional para um pequeno sistema que é capaz de calcular a área de qualquer uma das principais figuras estudadas na geometria plana: quadrilátero, triângulo, círculo, trapézio e losango. Além disso, esse sistema também é capaz de calcular as raízes de uma equação do segundo grau. E por fim, ele também é capaz de calcular a distância entre dois pontos representados em um plano cartesiano. Esse sistema precisa de um menu inicial que apresenta as opções ao usuário e, de acordo com a escolha, irá realizar o cálculo solicitado e imprimir a resposta. O trecho de código abaixo ilustra a implementação completa desse pequeno sistema.

```
import math

def calcular_area_do_triangulo(base, altura):
    area = (base * altura) / 2

    return round(area, 2)
```

```

def calcular_area_do_circulo(raio):
    area = math.pi * raio ** 2

    return round(area, 2)

def calcular_area_do_trapezio(base_maior, base_menor, altura):
    area = ((base_maior + base_menor) * altura) / 2

    return round(area, 2)

def calcular_area_do_loosango(diagonal_maior, diagonal_menor):
    area = (diagonal_maior * diagonal_menor) / 2

    return round(area, 2)

def calcular_area_do_quadrilatero(lado1, lado2):
    area = lado1 * lado2

    return round(area, 2)

def calcular_raizes_bhaskara(a, b, c):
    delta = b ** 2 - 4 * a * c

    if delta > 0:
        print('A equação possui duas raízes reais distintas!\n')
        x1 = (- b + math.sqrt(delta)) / (2 * a)
        x2 = (- b - math.sqrt(delta)) / (2 * a)
        print('O resultado é:')
        print(f'x1 = {round(x1, 2)}')
        print(f'x2 = {round(x2, 2)}')
    elif delta == 0:
        print('A equação possui duas raízes reais iguais!\n')
        x1 = (- b + math.sqrt(delta)) / (2 * a)
        x2 = x1
        print('O resultado é:')
        print(f'x1 = {round(x1, 2)}')
        print(f'x2 = {round(x2, 2)}')
    else:
        print('A equação não possui raízes reais!\n')
        x1_real = - b / (2 * a)
        x1_imaginario = math.sqrt(-delta) / (2 * a)
        x2_real = - b / (2 * a)
        x2_imaginario = - math.sqrt(-delta) / (2 * a)
        print('O resultado é:')
        print(f'x1 = {x1_real}+{x1_imaginario}i')
        print(f'x2 = {x2_real}{x2_imaginario}i')

def calcular_distancia_entre_dois_pontos(x1, y1, x2, y2):
    distancia = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

    return round(distancia, 2)

```

```

def menu():
    print(
        '\n 0 - Encerrar \n 1 - Triângulo \n 2 - Círculo \n 3 -
        Trapézio \n 4 - Losango \n 5 - Quadrilátero \n 6 - Bháskara \n 7 -
        Distância entre dois pontos')

    opcao = int(input('Informe a opção desejada:'))

    return opção

def executar_algoritmo():
    while True:

        opcao = menu()

        if opcao == 1:
            base = float(input('Insira o valor da base:'))
            altura = float(input('Insira o valor da altura:'))
            print('\n')
            area = calcular_area_do_triangulo(base, altura)
            print(f'A área do triângulo é: {area}\n')
        elif opcao == 2:
            raio = float(input('Insira o valor do raio:'))
            print('\n')
            area = calcular_area_do_circulo(raio)
            print(f'A área do círculo é: {area}\n')
        elif opcao == 3:
            base_maior = float(input('Insira o valor da base maior:'))
            base_menor = float(input('Insira o valor da base menor:'))
            altura = float(input('Insira o valor da altura:'))
            print('\n')
            area = calcular_area_do_trapezio(base_maior, base_menor,
altura)
            print(f'A área do trapézio é: {area}\n')
        elif opcao == 4:
            diagonal_maior = float(input('Insira o valor da diagonal
maior:'))
            diagonal_menor = float(input('Insira o valor da diagonal
menor:'))
            print('\n')
            area = calcular_area_do_losango(diagonal_maior,
diagonal_menor)
            print(f'A área do losango é: {area}\n')
        elif opcao == 5:
            lado1 = float(input('Insira o valor do lado 1:'))
            lado2 = float(input('Insira o valor do lado 2:'))
            print('\n')
            area = calcular_area_do_quadrilatero(lado1, lado2)
            print(f'A área do quadrilátero é: {area}\n')
        elif opcao == 6:
            a = float(input('Insira o valor de a:'))
            b = float(input('Insira o valor de b:'))
            c = float(input('Insira o valor de c:'))
            print('\n')
            calcular_raizes_bhaskara(a, b, c)
        elif opcao == 7:
            x1 = float(input('Insira o valor de x1:'))
            y1 = float(input('Insira o valor de y1:'))
            x2 = float(input('Insira o valor de x2:'))
            y2 = float(input('Insira o valor de y2:'))

```

```

        print('\n')
        distancia = calcular_distancia_entre_dois_pontos(x1, y1,
x2, y2)
        print(f'A distância entre os dois pontos é:
{distancia}\n')
        elif opcao == 0:
            print('\n')
            print('Sistema encerrado!\n')
            break

    else:
        print('\n')
        print('Opção inválida!\n')
        print('Sistema encerrado!\n')
        break

executar_algoritmo()

```

O trecho de código acima é todo modularizado, o que facilita o entendimento e a organização. Em primeiro lugar, existe uma função para calcular a área de cada uma das figuras planas descritas. Na sequência, uma função para calcular as raízes de uma equação do segundo grau por meio da fórmula de Bháskara é implementada. Por fim, uma função que calcula a distância entre dois pontos no plano cartesiano fecha a principal ideia desse algoritmo, que é ter uma função para fazer cada tarefa. Perceba que cada uma das funções citadas possui seus argumentos, de acordo com o cálculo executado. Na sequência existe uma função *menu()* que tem o objetivo de apresentar as opções disponíveis ao usuário, coletar a escolha e retornar. Finalmente, a função *executar_algoritmo()* direciona a opção escolhida para uma sequência de *if-elif-else* que irão fazer as chamadas às funções corretas, calcular o que está sendo solicitado e imprimir os resultados. Todo o sistema funciona dentro de um *while* (utilizando o *True*) que permanece em execução constante até que a opção 0 seja digitada para encerrar ou uma opção inválida seja inserida. Essas paradas são feitas por meio do comando *break*. A Figura 5.1 abaixo mostra o menu de opções do algoritmo e sua saída.


```
executar_algoritmo()

0 - Encerrar
1 - Triângulo
2 - Círculo
3 - Trapézio
4 - Losango
5 - Quadrilátero
6 - Bháskara
7 - Distância entre dois pontos
Informe a opção desejada: 1
Insira o valor da base: 5
Insira o valor da altura: 6

A área do triângulo é: 15.0

0 - Encerrar
1 - Triângulo
2 - Círculo
3 - Trapézio
4 - Losango
5 - Quadrilátero
6 - Bháskara
7 - Distância entre dois pontos
Informe a opção desejada: |↵ for history
```

Figura 5.1: Menu principal do algoritmo e saída de resultado da área do triângulo

Observe que na parte superior da Figura 5.1 a opção 1 foi digitada no menu de opções e na sequência os valores 5 e 6 foram inseridos. Por fim a área do triângulo é exibida. O menu principal aparece novamente aguardando uma nova opção. A Figura 5.2 abaixo ilustra a escolha 6 que leva ao uso da fórmula de Bháskara. É possível ver que os valores de x_1 e x_2 , que são as raízes da equação, são números complexos.

```
0 - Encerrar
1 - Triângulo
2 - Círculo
3 - Trapézio
4 - Losango
5 - Quadrilátero
6 - Bháskara
7 - Distância entre dois pontos
Informe a opção desejada: 6
Insira o valor de a: 1
Insira o valor de b: -14
Insira o valor de c: 50
```

A equação não possui raízes reais!

```
0 resultado é:
x1 = 7.0+1.0i
x2 = 7.0-1.0i
```

```
0 - Encerrar
1 - Triângulo
2 - Círculo
3 - Trapézio
4 - Losango
5 - Quadrilátero
6 - Bháskara
7 - Distância entre dois pontos
```

Informe a opção desejada: ↩ for history

Figura 5.2: Menu principal do algoritmo e saída de resultado para as raízes da equação de segundo grau

O menu principal continua sendo exibido aguardando uma opção até que 0 seja digitado para encerrar o sistema ou então uma opção inválida ser digitada, que também leva ao encerramento.

Vamos analisar agora um exemplo disponível em (Farrer et al., 1999) no enunciado abaixo.

Em uma cidade do interior, sabe-se que, de janeiro a abril de 1976 (121) dias, não ocorreu temperatura inferior a 15°C nem superior a 40°C. Fazer um algoritmo que calcule e imprima:

- A menor temperatura ocorrida
- A maior temperatura ocorrida
- A temperatura média
- O número de dias nos quais a temperatura foi inferior à temperatura média

O trecho de código abaixo implementa toda a lógica para esse algoritmo.

```
import random

temperaturas = []
for i in range(121):
    temperaturas.append(random.randint(15, 40))

def calcular_media_de_temperatura(temperaturas):
    soma = 0

    for temp in temperaturas:
        soma += temp

    media = soma / len(temperaturas)

    return round(media, 2)

def calcular_dias_abaixo_da_media(temperaturas):
    temperatura_media = calcular_media_de_temperatura(temperaturas)

    contador = 0

    for temp in temperaturas:
        if temp < temperatura_media:
            contador += 1

    return contador

def menu():
    print(
        '\n 0 - Encerrar \n 1 - Menor temperatura \n 2 - Maior temperatura \n 3 - Temperatura média \n 4 - Quantidade de dias com temperatura abaixo da média \n')

    opcao = int(input('Informe a opção desejada:'))

    return opcao
```

```

def executar_analise_temperaturas():
    while True:

        opcao = menu()

        if opcao == 1:
            menor_temperatura = min(temperaturas)
            print('\n')
            print(f'A menor temperatura é: {menor_temperatura}\n')
        elif opcao == 2:
            maior_temperatura = max(temperaturas)
            print('\n')
            print(f'A maior temperatura é: {maior_temperatura}\n')
        elif opcao == 3:
            temperatura_media =
calcular_media_de_temperatura(temperaturas)
            print('\n')
            print(f'A temperatura média é: {temperatura_media}\n')
        elif opcao == 4:
            temperatura_abaixo_da_media =
calcular_dias_abaixo_da_media(temperaturas)
            print('\n')
            print(f'Houveram {temperatura_abaixo_da_media} com
temperatura abaixo da média\n')
        elif opcao == 0:
            print('\n')
            print('Sistema encerrado!\n')
            break
        else:
            print('\n')
            print('Opção inválida!\n')
            print('Sistema encerrado!\n')
            break

executar_analise_temperaturas()

```

Esse algoritmo utiliza o conceito de geração de números aleatórios. Veja na primeira linha do código a importação da biblioteca *random*. São gerados 121 números aleatórios entre 15 e 40. Todos esses números são adicionados a uma lista de temperaturas. A sequência é muito parecida com o exemplo anterior dos cálculos matemáticos. Algumas funções são criadas para modularizar o código. Uma função menu direciona o que o usuário pode fazer e a última função junta toda a lógica para executar o algoritmo. As figuras abaixo (Figura 5.3 e Figura 5.4) ilustram algumas operações com o algoritmo.

```
executar_analise_temperaturas()
```

```
0 - Encerrar  
1 - Menor temperatura  
2 - Maior temperatura  
3 - Temperatura média  
4 - Quantidade de dias com temperatura abaixo da média
```

Informe a opção desejada: 1

A menor temperatura é: 15

```
0 - Encerrar  
1 - Menor temperatura  
2 - Maior temperatura  
3 - Temperatura média  
4 - Quantidade de dias com temperatura abaixo da média
```

Informe a opção desejada: ↵ for history

Figura 5.2: Menu principal do algoritmo de temperaturas e saída de resultado para a temperatura mínima

```
0 - Encerrar  
1 - Menor temperatura  
2 - Maior temperatura  
3 - Temperatura média  
4 - Quantidade de dias com temperatura abaixo da média
```

Informe a opção desejada: 4

Houveram 60 com temperatura abaixo da média

```
0 - Encerrar  
1 - Menor temperatura  
2 - Maior temperatura  
3 - Temperatura média  
4 - Quantidade de dias com temperatura abaixo da média
```

Informe a opção desejada: ↵ for history

Figura 5.3: Menu principal do algoritmo de temperaturas e saída de resultado para temperaturas abaixo da média

Vamos analisar a situação a seguir. Precisamos implementar um algoritmo que solicite ao usuário para inserir números inteiros. O processo de inserção finaliza quando o usuário digita 0 (isso é verificado por meio de um *if*). De posse dos valores digitados em uma lista é possível verificar a quantidade de elementos inseridos e a média dos elementos. Veja o trecho de código abaixo.

```
import random

def calcular_media_de_valores(lista_valores):
    soma = 0

    for item in lista_valores:
        soma += item

    media = soma / len(lista_valores)

    return round(media, 2)

def menu():
    print(
        '\n 0 - Encerrar \n 1 - Inserir valores \n 2 - Quantidade de\n'
        'valores inseridos \n 3 - Média dos valores inseridos \n')

    opcao = int(input('Informe a opção desejada:'))

    return opcao
```

```

def executar():
    lista_numeros = []

    while True:

        opcao = menu()

        if opcao == 1:
            execucao = True
            while execucao:
                variavel = int(input('Insira um número inteiro'))
                if variavel != 0:
                    lista_numeros.append(variavel)
                else:
                    execucao = False
                    print('\n')
                    print('Inserção de números encerrada!\n')
            elif opcao == 2:
                quantidade_valores_inseridos = len(lista_numeros)
                print('\n')
                print(f'Foram inseridos {quantidade_valores_inseridos}
valores\n')
            elif opcao == 3:
                media = calcular_media_de_valores(lista_numeros)
                print('\n')
                print(f'A média dos valores inseridos é: {media}\n')
            elif opcao == 0:
                print('\n')
                print('Sistema encerrado!\n')
                break
            else:
                print('\n')
                print('Opção inválida!\n')
                print('Sistema encerrado!\n')
                break

executar()

```

A ideia desse algoritmo é muito semelhante aos exemplos apresentados anteriormente. Aqui existe um detalhe a mais que é a utilização de um *while* dentro do *while* principal. Isso se deve ao fato de o algoritmo ser dinâmico e permitir o usuário digitar quantos números ele desejar até inserir o 0. Esse fato de inserir de maneira dinâmica se deve ao comportamento das listas em Python que suportam a adição de elementos por meio do método *append()*. As figuras abaixo ilustram o menu do algoritmo e sua operação para facilitar o entendimento do leitor.

```
0 - Encerrar
1 - Inserir valores
2 - Quantidade de valores inseridos
3 - Média dos valores inseridos
```

```
Informe a opção desejada: 1
Insira um número inteiro 4
Insira um número inteiro 8
Insira um número inteiro 12
Insira um número inteiro 3
Insira um número inteiro 1
Insira um número inteiro 0
```

Inserção de números encerrada!

```
0 - Encerrar
1 - Inserir valores
2 - Quantidade de valores inseridos
3 - Média dos valores inseridos
```

Informe a opção desejada:

Figura 5.4: Menu principal do algoritmo de inserção de valores

```
0 - Encerrar
1 - Inserir valores
2 - Quantidade de valores inseridos
3 - Média dos valores inseridos
```

Informe a opção desejada: 2

Foram inseridos 5 valores

```
0 - Encerrar
1 - Inserir valores
2 - Quantidade de valores inseridos
3 - Média dos valores inseridos
```

Informe a opção desejada:

Figura 5.5: Menu principal do algoritmo de inserção de valores e contagem da quantidade de elementos digitados


```
0 - Encerrar
1 - Inserir valores
2 - Quantidade de valores inseridos
3 - Média dos valores inseridos
```

Informe a opção desejada: 3

A média dos valores inseridos é: 5.6

```
0 - Encerrar
1 - Inserir valores
2 - Quantidade de valores inseridos
3 - Média dos valores inseridos
```

Informe a opção desejada:

Figura 5.6: Menu principal do algoritmo de inserção de valores e cálculo da média dos valores inseridos

A Figura 5.4 mostra a escolha da opção 1 no menu principal e a inserção de cinco número inteiros (4, 8, 12, 3, 1). Na Figura 5.5 a opção 2 foi escolhida e o algoritmo exibe a contagem de números digitados (5). Finalmente na Figura 5.6 a média dos valores digitados é exibida.

Um outro exemplo muito interessante que pode ser implementado é um gerador de números aleatórios para simular a Mega-Sena. O código abaixo mostra a implementação.

```
import random

def inserir_numeros_no_volante():
    numeros = []
    executar = True

    while executar:

        numero = int(input('Insira um número:'))

        if numero >= 1 and numero <= 60 and numero not in numeros:
            numeros.append(numero)

            if len(numeros) == 6:
                executar = False
                print('Obrigado por inserir os 6 números!\n')
            else:
                print('Número digitado inválido! Digite novamente!\n')
                print('Lembre-se que o número deve estar entre 1 e 60 e não ter sido inserido anteriormente.\n')
```

```

    return numeros

def sortear_numeros():
    numeros = []

    for i in range(6):
        numero_sorteado = random.randint(1, 60)
        if numero_sorteado not in numeros:
            numeros.append(numero_sorteado)

    return numeros

def conferir_numeros_jogados_e_sorteados(numeros_jogados,
numeros_sorteados):
    acertos = []

    for numero in numeros_sorteados:
        if numero in numeros_jogados:
            acertos.append(numero)

    return acertos

def menu():
    print('ALGORITMO DA MEGA SENA\n')

    print('\n 0 - Encerrar \n 1 - Inserir números no volante \n 2 -
Realizar sorteio \n 3 - Conferir acertos \n')

    opcao = int(input('Informe a opção desejada:'))

    return opcao

def executar_mega_sena():
    numeros_jogados = []
    numeros_sorteados = []

    while True:

        opcao = menu()

        if opcao == 1:
            if len(numeros_sorteados) > 0:
                print('O sorteio já foi realizado! Você não pode jogar
após o sorteio!\n')
                numeros_sorteados = []
            else:
                numeros_jogados = inserir_numeros_no_volante()
                print('\n')
                print(f'Os números jogados são: {numeros_jogados}\n')
        elif opcao == 2:
            numeros_sorteados = sortear_numeros()
            print('\n')
            print(f'Os números sorteados são: {numeros_sorteados}\n')
        elif opcao == 3:
            if len(numeros_jogados) == 0:
                print('Você ainda não jogou nenhum número!\n')
            elif len(numeros_sorteados) == 0:
                print('O sorteio ainda não foi realizado!\n')
            else:

```

```

        acertos =
conferir_numeros_jogados_e_sorteados(numeros_jogados,
numeros_sorteados)
        print('\n')
        if len(acertos) > 0:
            if len(acertos) == 6:
                print('Parabéns! Você acertou todos os
números!\n')
                print(acertos)
            else:
                print(f'Os números jogados que foram sorteados
são: {acertos}\n')
            else:
                print('Não houveram acertos!\n')
        elif opcao == 0:
            print('\n')
            print('Sistema de jogo encerrado!\n')
            break
        else:
            print('\n')
            print('Opção inválida!\n')
            print('Sistema de jogo encerrado!\n')
            break

executar_mega_sena()

```

O algoritmo apresentado acima utiliza a mesma lógica dos demais ilustrados nessa unidade 5 no que se refere à construção de um menu de opções. Além disso, é importante analisar algumas validações feitas aqui como não permitir que um número que já foi sorteado pelo método *random()* apareça novamente no sorteio. Outro ponto importante é não deixar o usuário escolher números repetidos, que seria o equivalente a marcar apenas 5 números no volante em uma lotérica na vida real. Também não deve ser permitido marcar números imediatamente após o sorteio.

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada: 1
Insira um número: 56
Insira um número: 32
Insira um número: 11
Insira um número: 9
Insira um número: 35
Insira um número: 18
Obrigado por inserir os 6 números!

Os números jogados são: [56, 32, 11, 9, 35, 18]

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada:

Figura 5.7: Menu principal do algoritmo da Mega-Sena mostrando a escolha dos seis números

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada: 2

Os números sorteados são: [38, 34, 42, 55, 57]

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada:

Figura 5.8: Menu principal do algoritmo da Mega-Sena mostrando os números sorteados

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada: 3

Não houveram acertos!

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada:

Figura 5.9: Menu principal do algoritmo da Mega-Sena e a conferência dos resultados

As figuras acima (Figura 5.7, Figura 5.8 e Figura 5.9) mostram o menu do algoritmo e a seleção das opções de inserção de números, sorteio e comparação entre jogado e sorteado. A opção 3 em especial, avalia se existe uma lista com valores de sorteio da Mega-Sena e caso contrário solicita que um sorteio seja realizado. De posse dos valores do sorteio, uma comparação é feita por meio de um laço for que percorre os valores sorteados e verifica se eles existem na lista de valores que foram jogados (selecionados) pelo usuário. Quando não existe nenhum elemento igual entre os dois conjuntos o sistema exibe a mensagem de que não houve acertos (Figura 5.9).

ALGORITMO DA MEGA SENA

- 0 - Encerrar
- 1 - Inserir números no volante
- 2 - Realizar sorteio
- 3 - Conferir acertos

Informe a opção desejada: 0

Sistema de jogo encerrado!

Figura 5.10: Menu principal do algoritmo da Mega-Sena mostrando a escolha de encerrar o sistema

Finalmente, a Figura 5.10 acima mostra a mensagem de encerramento do sistema após escolher a opção 0.

5.3. EXERCÍCIOS

Implemente os exemplos abaixo:

1 - Em matemática, sequências de números são estudadas exaustivamente e duas em especial merecem destaque: A progressão aritmética (PA) e a progressão geométrica (PG). Implemente um algoritmo que possua um menu no qual o usuário escolha com qual dessas duas sequências ele deseja trabalhar. O menu deve possuir uma maneira de sair ao digitar 0. Uma vez escolhida a PA ou a PG o usuário deve ser capaz de digitar quantos números ele quiser para formar uma PA ou uma PG. Finalmente, após a inserção dos números, uma mensagem aparece solicitando ao usuário para digitar uma posição futura para que o sistema calcule qual número irá ocupar tal posição (de acordo com a teoria de PA e PG).

2 - Implemente um algoritmo para cadastrar funcionários de uma empresa fictícia. O algoritmo deve possuir um menu que exhibe as seguintes opções:

Cadastrar funcionário

Exibir os nomes dos funcionários cadastrados

Exibir o funcionário com maior salário

Exibir o funcionário com o menor salário

Sair do sistema

Na parte de cadastro de usuário o sistema deve solicitar o nome e o salário e armazenar em listas (uma para o nome e outra para o salário). Na parte de exibição dos nomes cadastrados, basta imprimir cada elemento da lista de nomes. Para exibir o funcionário com o maior e com o menor salário, basta avaliar a lista de salários, obter o índice do valor (maior ou menor em cada caso) e buscar o nome na outra lista que possui o mesmo índice para finalmente imprimir na tela. O usuário deve digitar 0 para sair do sistema. O sistema deve ser implementado utilizando uma lógica que permita que o usuário cadastre quantos funcionários desejar (*while*), ou seja, o sistema só encerra após a digitação do 0.

3 - Implemente um algoritmo que seja capaz de calcular o volume dos principais sólidos geométricos (cubo, paralelepípedo, cilindro, pirâmide e esfera). Deve ser implementada uma função para o cálculo de volume de cada figura. Além disso, construa um menu para que o usuário possa escolher uma opção de cálculo. Siga o exemplo apresentado na seção 5.2 para o cálculo da área das figuras planas.

FINALIZAR

Após a leitura desse material, espero que o aprendizado dos conceitos relacionados à lógica de programação esteja mais fácil. Essa foi uma introdução ao assunto e já habilita o leitor a implementar seus primeiros exemplos.

As técnicas utilizadas nesse material são básicas e de caráter educativo, visto que o leitor precisa se familiarizar com os conteúdos e amadurecer aos poucos para conseguir compreender tópicos mais complexos. Além disso, existem diversos conceitos relacionados à qualidade de código e boas práticas de programação que o leitor precisa buscar para produzir sistemas de qualidade.

Espero que esse material tenha despertado a curiosidade no leitor para continuar investigando esse mundo maravilhoso dos algoritmos e se aprofundar. A área de tecnologia é gigante e sempre necessita de bons profissionais, o que serve de incentivo para o leitor estar sempre ativo nos estudos. Busque novas tecnologias e aprendizados. Parabênizo o esforço empregado até aqui e convido o leitor a mergulhar de cabeça nos estudos pois vale a pena.

Prof. Dr. Thiago Santana Lemes

Sobre o autor

Thiago Santana Lemes é doutor em Engenharia Elétrica e de Computação pela Universidade Federal de Goiás (UFG), Cientista de dados na Ernst Young (EY). Experiência docente no ensino básico atuando na disciplina de matemática e nos cursos de Engenharia e Sistemas de Informação atuando nas disciplinas de cálculo diferencial, álgebra linear, cálculo numérico, equações diferenciais, probabilidade e estatística, algoritmos dentre outras. Desenvolve pesquisas na área de inteligência artificial com modelos de Machine Learning (implementação de modelos de regressão, classificação e recomendação), Deep Learning (redes convolucionais, redes recorrentes) e modelos de otimização (algoritmos genéticos e modelos lineares).

Referências Bibliográficas

- Farrer, H., Becker, C. G., Faria, E. C., Matos, H. F. de, Santos, M. A. dos, & Maia, M. L. (1999). *Algoritmos Estruturados* (3rd ed.). LTC.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Lambert, K. A. (2022). *Fundamentos de Python* (1st ed.). Cengage Learning.
- Manzano, J. A. N. G., & Oliveira, J. F. de. (2019). *Algoritmos* (29th ed.). Érica.
- Mueller, J. P. (2020). *Começando a Programar em Python Para Leigos* (2nd ed.). Alta Books.