Programação de Computadores I – BCC701

Conteúdo Teórico em Linguagem Python

2020/1

Elaborado pela Comissão de Unificação da Disciplina BCC701,
com a colaboração de professores e estagiários docentes
http://www.decom.ufop.br/bcc701/

DECOM – Departamento de Computação
ICEB – Instituto de Ciências Exatas e Biológicas
UFOP – Universidade Federal de Ouro Preto

Sumário

1 Introdução **4**

1.1 Como funciona um computador? . . . 4
1.2 Programas de Computadores . . . 4
1.2.1 Linguagem Python . . . 5
1.3 Programação Estruturada . . . 5
1.3.1 Sequência . . . 5
1.3.2 Decisão . . . 6
1.3.3 Iteração . . . 6
1.4 Algoritmos e Fluxogramas . . . 7
1.4.1 Exemplo 1 . . . 7
1.5 Exercícios Propostos . . . 9

2 Variáveis e Expressões **11**

2.1 Para que serve uma variável? . . . 11
2.2 Regras para o nome de uma variável . . . 12
2.3 Atribuição de Valores para uma Variável . . . 12
2.4 Tipos de Dados . . . 15
2.4.1 Numéricos . . . 15
2.4.2 String . . . 16
2.5 Operadores aritméticos . . . 16
2.5.1 Precedência e associatividade de operadores . . . 19
2.6 Funções elementares . . . 19
2.6.1 Resumo das funções elementares . . . 21
2.7 Constantes . . . 21
2.8 Exercícios Propostos . . . 23

3 Entrada e Saída **25**

3.1 Saída: função print . . . 26
3.1.1 Strings de formatação . . . 26
3.1.2 Especificando a formatação . . . 27
3.1.3 Inserção de caracteres especiais em uma string . . . 29
3.2 Entrada: função input . . . 30
3.2.1 Conversão de tipos . . . 31
3.2.2 Conversão para inteiro: função int . . . 31
3.2.3 Convertão para real: função float . . . 31
3.2.4 Convertão para string: função str . . . 31
3.2.5 Cuidados especiais com conversão de tipos . . . 32
3.2.6 Conversão de dados fornecidos pelo usuário . . . 32
3.3 Entrada e Saída: Exemplos Práticos . . . 32
3.4 Exercícios Propostos . . . 37

4 Estrutura de Decisão **41**
4.1 O comando if . . . 42
4.2 Operadores Relacionais . . . 45
4.3 A cláusula else . . . 45
4.4 Operadores Lógicos . . . 49
4.5 Estruturas de decisão aninhadas . . . 50
4.6 Exercícios Propostos . . . 54

5 Estrutura de Repetição **58**
5.1 Repetição controlada logicamente: comando while . . . 59
5.2 Repetição controlada por contador: comando for . . . 64
5.2.1 A função range . . . 65
5.3 Estruturas de repetição aninhadas . . . 69
5.4 Exercícios Propostos . . . 75

6 Funções **80**
6.1 Definição de Funções . . . 81
6.2 Chamada de Funções . . . 82
6.3 Escopo de variáveis . . . 84
6.4 Retornando vários valores . . . 85
6.5 Exercícios Propostos . . . 86

7 Estruturas Homogêneas **90**
7.1 Vetores . . . 91
7.1.1 Sintaxe básica . . . 91
7.1.2 Operações com vetores . . . 92
7.1.3 Exercícios propostos – Vetores . . . 95
7.2 Matrizes . . . 97
7.2.1 Sintaxe básica . . . 98
7.2.2 Operações com matrizes . . . 98
7.2.3 Exercícios propostos – Matrizes . . . 100
7.3 Construindo uma biblioteca de funções úteis . . . 103
7.3.1 Criando um vetor com valor padrão . . . 103
7.3.2 Criando uma matriz com valor padrão . . . 103
7.3.3 Criando um vetor com valores de uma string . . . 103
7.3.4 Criando uma matriz com valores de uma string . . . 104
7.3.5 Imprimir um vetor na tela . . . 105
7.3.6 Imprimir uma matriz na tela . . . 105
7.3.7 Criando e usando a biblioteca . . . 106
7.4 Exercícios propostos . . . 108

8 Estruturas Heterogêneas **111**
8.1 Definição . . . 111

8.2 Sintaxe básica 112
8.3 Vetores e matrizes de registros 114
8.4 Registros como argumento ou retorno de função 115
8.5 Exercícios propostos 116

9 Soluções dos Exercícios Propostos **119**
9.1 Exercícios do Módulo 1: Introdução 119
9.2 Exercícios do Módulo 2: Variáveis e Expressões 122
9.3 Exercícios do Módulo 3: Entrada e Saída 126
9.4 Exercícios do Módulo 4: Estrutura de Decisão 129
9.5 Exercícios do Módulo 5: Estrutura de Repetição 135
9.6 Exercícios do Módulo 6: Função 142
9.7 Exercícios do Módulo 7: Estruturas Homogêneas 145
9.8 Exercícios do Módulo 8: Estruturas Heterogêneas 156

MÓDULO 1

Introdução

O presente material foi elaborado pela Comissão de Unificação da disciplina de Programação de Computadores I (BCC701) com a colaboração de professores e estagiários docentes do DECOM durante o semestre letivo 2019-02. Nele procuramos apresentar todos os conceitos e recursos mínimos a serem abordados no conteúdo programático da disciplina. Professores podem abordar os assuntos à sua maneira, expandindo determinado conteúdo quando considerar necessário, assim como alterando a ordem de apresentação dos mesmos.

Neste módulo introduzimos alguns conceitos importantes sobro o funcionamento dos computadores, programação de computadores, programação estruturada e algoritmos.

1.1 Como funciona um computador?

Para compreender melhor o conteúdo da disciplina, é preciso entender o que são os **programas de computadores** e o que significa **programação estruturada**.

Um computador funciona baseado em 3 principais elementos: **Processador**, **Memória** e **Armazenamento**. Assista a este vídeo para compreender melhor este assunto: https://youtu.be/MpKbTNonIwc.

1.2 Programas de Computadores

Um programa de computador é uma sequência de instruções que um computador segue para realizar uma tarefa. Diversas tarefas do nosso dia-a-dia seguem esse tipo de lógica: receitas culinárias; instruções de um manual; execução de uma peça de música; etc.

De maneira similar a um músico que executa uma peça a partir de uma partitura, um computador precisa de um conjunto de instruções para executar alguma tarefa. Essas instruções estão contidas em um arquivo denominado **código fonte**. Assista a este vídeo para compreender melhor este assunto: https://youtu.be/hD42nKsCimQ.

Existem diversas linguagens nas quais podemos desenvolver nosso código fonte. Entre elas estão: C; C++; Java; Javascript; Ruby; Matlab/Octave; **Python**; etc.

Nessa disciplina utilizaremos a linguagem de programação **Python** devido à simplicidade e extensa documentação na internet.

1.2.1 Linguagem Python

A linguagem **Python**, utilizada nessa disciplina, é uma das linguagens mais populares e utilizadas do mundo. Além disso, ela é simples de entender e mais ainda de usar. Abaixo, vamos ensinar um jeito fácil de começar a usar a linguagem.

Python é uma linguagem muito ampla e flexível, utilizaremos neste curto um conjunto limitado de seus recursos, com o intuito de aplicar e fixar o conhecimento de estruturas e conceitos de programação de computadores em nível introdutório, compreendendo o conteúdo programático da disciplina.

Escrever um código em **Python** é semelhante a escrever qualquer lista ou sequência de instruções. Contudo, um computador não consegue executar qualquer tipo de instrução. As instruções aceitas são lógicas, aritméticas ou de controle. A seguir vemos um exemplo possível de programa em **Python**, feito para elevar um número ao quadrado:

```
1 # Entrada
2 x = float(input('Digite um número para elevar ao quadrado: '))
3 # Processamento
4 y = x * x
5 # Saída
6 print(f'O resultado do processamento é {y}')
```

No código acima, as linhas indicadas por # são comentários. Isso indica ao computador que tudo que estiver escrito nessas linhas, a partir de #, não deve ser interpretado como instruções a serem executadas. As demais linhas são maneiras de dar instruções ao computador usando a linguagem **Python**. Durante o decorrer da disciplina, aprenderemos a função de cada uma delas.

É importante **praticar** e **aprimorar** os seus conhecimentos na ferramenta de programação escolhida. Utilizaremos o **Thonny**, uma plataforma simples de aprendizado. Está disponível para Mac, Linux e Windows, e é super simples de instalar. Basta baixar no link https://thonny.org/ e executar. No linux, é só seguir a instrução para instalação via linha de comando do terminal.

1.3 Programação Estruturada

Conhecendo o conceito de programas de computador, agora é importante saber também o que significa **Programação Estruturada**. Conforme dito, um programa de computador executa uma sequência de instruções para realizar uma tarefa. Na programação estruturada, instruções podem ser classificadas em três tipos de estrutura: **Sequência**; **Decisão**; **Iteração**.

A seguir apresentamos estas três estruturas em fluxogramas. Os fluxogramas são uma forma de representar um programa de forma mais "alto nível" para entendermos melhor a lógica dos programas implementados. Não se preocupe se você ainda não entender tão bem essas três estruturas. Elas serão trabalhadas novamente e com mais detalhes ao longo da disciplina.

1.3.1 Sequência

Como o nome sugere, essa estrutura é composta por uma **sequência** enfileirada de instruções que são executadas uma após a outra, como demonstrado na Figura 1. Com esse tipo de estrutura, podemos representar a visão geral do problema que desejamos resolver.

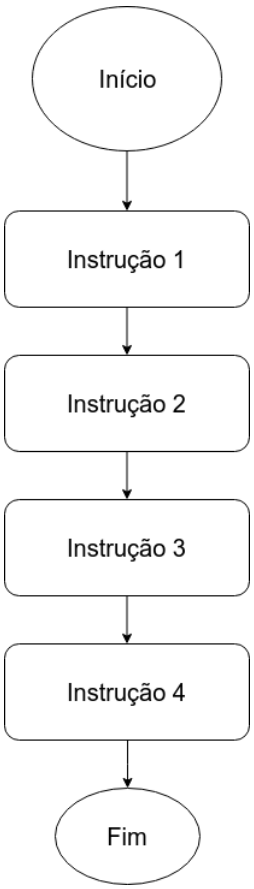

Figura 1: Estrutura do tipo **sequência**.

1.3.2 Decisão

Estruturas de **decisão** são utilizadas para realizar um controle e efetuar ações de acordo com uma determinada condição, como demonstrado na Figura 2.

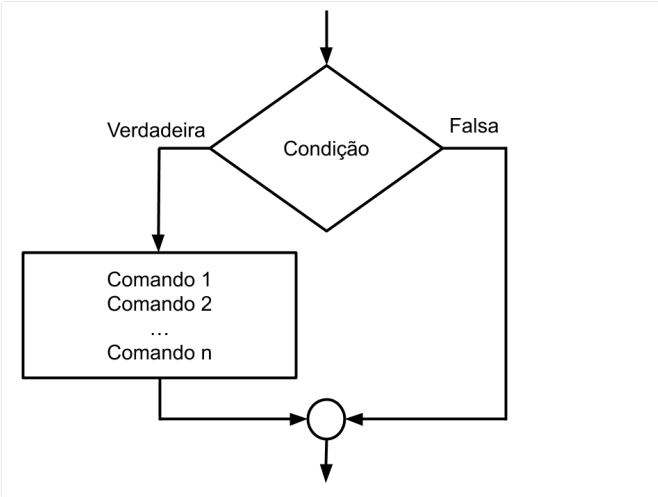

Figura 2: Estrutura do tipo **Decisão**.

1.3.3 Iteração

Estruturas de **iteração** (**repetição** ou **laço**) são utilizadas para realizar uma tarefa repetitiva de forma que alguma condição seja cumprida, como demonstrado na Figura 3.

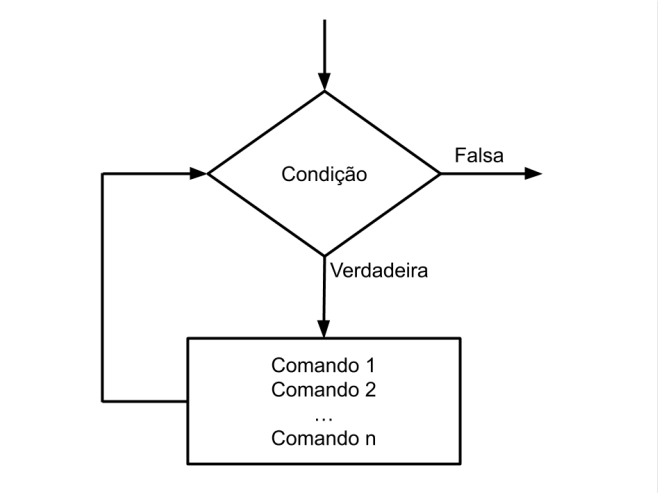

Figura 3: Estrutura do tipo **Iteração**.

1.4 Algoritmos e Fluxogramas

Para criar um programa de computador, precisamos de uma sequência de instruções claras e sem ambiguidade, usando os conceitos apresentados na Seção 1.3 – Programação Estruturada. O conjunto dessas instruções e como elas são estruturadas é o que chamamos de **Algoritmo**. Utilizamos algoritmos para resolver os mais diversos problemas do dia-a-dia, da nossa vida acadêmica ou profissional.

Para organização, dividiremos os algoritmos em **3 etapas**:

1. Entrada
2. Processamento
3. Saída

Para representar um algoritmo, uma ferramenta interessante são os **fluxogramas**, da mesma maneira que representamos anteriormente as estruturas de *sequência*, *decisão* e *iteração*. Eles representam cada um dos passos realizados por um algoritmo em caixas. O caminho tomado pelo algoritmo é representado por setas interligando as caixas. Esse tipo de representação permite criar visualizações amigáveis e fáceis de entender. Vejamos um exemplo a seguir.

1.4.1 Exemplo 1

Suponha que na aula de física você aprendeu a converter uma temperatura de **Celsius** para **Fahrenheit**. A fórmula usada é:

$$T_F = (T_C . \frac{9}{5}) + 32$$

Em uma sequência sistemática de passos, o raciocínio pode ser o seguinte:

```
1. Entrada:
    1.1. Ler um valor de temperatura em Celsius;
2. Processamento
    2.1. Multiplicar o valor por 9 e dividir por 5;
    2.2. Somar 32 ao valor encontrado;
3. Saída:
    3.1. Imprimir o valor da saída.
```

Vamos agora organizar esse programa em um fluxograma, para visualizar os passos que o programa vai tomar. Observe na Figura 4, que o fluxograma apenas organiza os passos antes representados por uma lista textual.

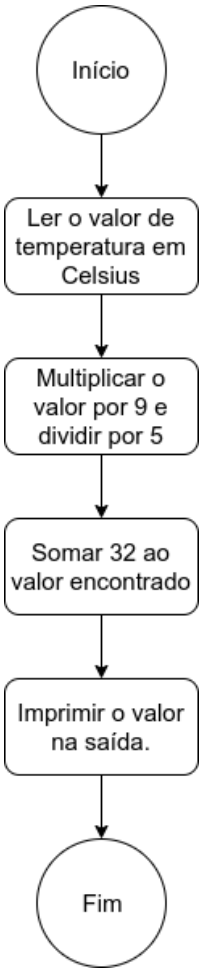

Figura 4: Fluxograma para o Exemplo 1.

O fluxograma mostra a sequência de passos a ser tomada para atingir o objetivo final. Abaixo, podemos ver a aplicação e o funcionamento disso em um programa em **Python**. Tente utilizar o **Thonny** para criar este programa, salvando-o em um arquivo e em seguida executando o código para ver o que acontece.

Solução em Python para o exemplo 1:

```
1 # Entrada
2 C = float(input('Digite a temperatura em Celsius: '))
3 # Processamento
4 F = C * 9 / 5
5 F = F + 32
6 #Saída
7 print(f'A temperatura em Fahrenheit é {F:.1f}')
```

Observe que para este exemplo foram apresentadas 3 representações para se resolver algoritmicamente o problema. Primeiro de forma textual, depois em um fluxograma e, por fim, em um programa em **Python**. Todas estas representações estão corretas e resolvem o problema, porém, quando utilizamos uma linguagem de programação é possível executar o programa e ver o resultado final. Nesta disciplina o foco principal será o programa em linguagem **Python**, mas você pode gerar representações

intermediárias, ou em mais “alto nível”, para entender melhor como resolver o problema e somente depois implementar seu programa na linguagem **Python**.

A seguir alguns exercícios propostos, para serem resolvidos através de fluxogramas.

1.5 Exercícios Propostos

Exercício 1

Dada a equação que mostra a conversão de Fahrenheit para Celsius:

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

Crie um fluxograma que indique as etapas de um algoritmo que representa essa transformação. Veja a sequência na lista abaixo:

1. Ler um valor em Fahrenheit;
2. Subtrair 32 no valor;
3. Multiplicar por 5 e dividir por 9;
4. Imprimir o valor de saída;

Exercício 2

Faça a lista de instruções e o fluxograma para calcular a associação de dois resistores em série. As entradas são os valores dos resistores em ohms.

Fórmula:

$$R_e = R_1 + R_2$$

Exercício 3

Faça a lista de instruções e o fluxograma para calcular a associação de dois resistores em paralelo. As entradas são os valores dos resistores em ohms.

Fórmula:

$$R_e = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

Exercício 4

Apresente o fluxograma para calcular a aceleração de um corpo sendo puxado por uma mola, considerando como entrada a constante elástica da mola (k), a massa do corpo (m) e a extensão a partir da posição inicial (x).

Fórmulas:

$$F = k \cdot x$$

$$a = \frac{F}{m}$$

Exercício 5

Dado um objeto em queda livre, calcular a velocidade em um instante de tempo e a distância total percorrida desde o início da queda, tendo como a entrada do usuário o instante de tempo t.

Fórmulas:

$$v = g \cdot t$$

$$s = \frac{g \cdot t^2}{2}$$

$$g = 9,807$$

MÓDULO 2

Variáveis e Expressões

Em programação de computadores, uma **variável** é um contêiner que serve para armazenar dados na memória do computador. O dado que é armazenado pode ser referenciado usando o nome da variável. O programador pode escolher qualquer nome para uma variável, exceto as palavras-chave do **Python** (palavras reservadas para finalidades específicas), e é uma boa prática escolher nomes significativos que reflitam o conteúdo da variável.

Variáveis não são os próprios valores, você pode pensar nelas como pequenas caixas nas quais você pode guardar coisas, conforme ilustrado na Figura 5.

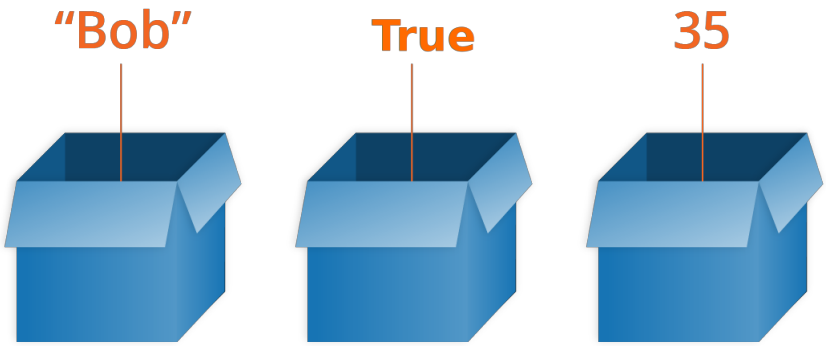

Figura 5: Representação de três variáveis.

2.1 Para que serve uma variável?

Variáveis são criadas para reservar um espaço na memória do computador para armazenar valores. Quando você cria uma variável, você está reservando um espaço na memória. Dependendo do tipo de variável que você criar, você pode armazenar valores inteiros, caracteres, valores decimais, etc.

2.2 Regras para o nome de uma variável

Para dar nomes às variáveis em **Python**, algumas regras devem ser observadas:

- Não podem conter espaços;
- Não podem iniciar com números;
- Além de caracteres alfanuméricos, pode conter somente o caracter: "_";
- Apesar de permitido em **Python**, caracteres acentuados em nomes de variáveis devem ser evitados, pois a maioria das linguagens de programação não permite acentos em nomes de variáveis.
- **Python** diferencia letras maiúsculas e minúsculas, ou seja: `Nome` $\neq$ `nome` $\neq$ `NOME`;
- Além disso, é recomendado que as variáveis tenham nomes significativos. A escolha de nomes significativos para as variáveis ajuda o programador a entender o que o programa faz e a prevenir erros.

Alguns exemplos de nomes **válidos** em **Python**:

- var
- nome1
- total_de_alunos
- José (**dica**: *apesar de válidos em Python, evite nomes de variáveis com acentos!*)

Alguns exemplos de nomes **inválidos** em **Python**:

- 1Aluno (o primeiro caractere é um algarismo)
- total de alunos (tem espaços)
- #funcionarios (o uso do caracter inválido "#")
- %valor (o uso do caracter inválido "%")

2.3 Atribuição de Valores para uma Variável

A Atribuição de Valores é a passagem de informação a determinada variável. Toda variável, por sua definição, pode receber valores ou então, pode ter seu valor alterado.

A linguagem **Python** tem definido que o sinal que conhecemos pelo nome de igual ($=$) será o **operador de atribuição**, conforme ilustrado na Figura 6.

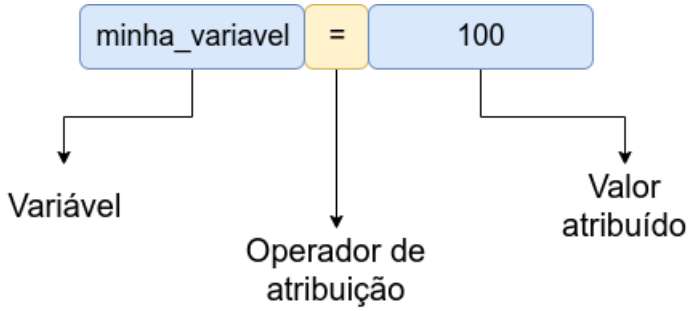

Figura 6: Atribuição de valor para uma variável.

A parte ao lado esquerdo do operador de atribuição sempre receberá o valor definido no lado direito do operador. Sendo assim, a única estrutura de linguagem aceita do lado esquerdo do operador é um nome de variável válido. Por outro lado, qualquer expressão válida, que resulte em um valor da linguagem de programação, poderá ser usado no lado direito do operador de atribuição.

Diferente de outras linguagens de programação, em **Python** não é necessário especificar o tipo de dado na declaração de uma variável, pois a linguagem infere o tipo de uma variável no momento da primeira atribuição.

Um valor a ser atribudo à variável pode ser, por exemplo:

- Um valor inteiro (int): 2, 4, 10, etc.;
- Um valor real (float): 2.7698, 0.0023, etc.;
- Um valor textual (string): `"Um texto qualquer"`;
- Um valor lógico (boolean): **True** ou **False**;
- Outros valores disponibilizados pela linguagem;

A seguir alguns exemplos de uso de variáveis em programas. Recomendamos que você crie os programas de exemplo no **Thonny** e execute alterando os valores para observar os diferentes resultados.

Exemplo 1:

Criar um programa contendo uma variável para guardar a informação da idade de uma pessoa, em seguida apresentar no monitor do computador o valor que está guardado na variável.

Solução:

```
1 idade = 19
2 print(idade)
```

Na primeira linha, o operando à esquerda do sinal de igual é o nome da variável (idade), enquanto o operando à direita do sinal de igual é o valor que será guardado na variável (19).

Na segunda linha, temos o comando **print** que apresenta na tela do computador o valor que está armazenado na variável idade. Este comando é utilizado nos exemplos a seguir de forma semelhante, apenas para imprimir o valor das variáveis, ele será detalhado no próximo módulo.

Exemplo 2:

Criar um programa contendo uma variável para armazenar o nome do cliente de uma loja que, em seguida, apresenta o valor desta variável na tela do computador.

Solução:

```
1 cliente = "João da Silva"
2 print(cliente)
```

Exemplo 3:

Criar um programa que armazene o valor 6 em uma variável e em seguida imprime esse valor na tela do computador.

Solução:

```
1 resultado = 6
2 print(resultado)
```

Exemplo 4:

Criar um programa que armazene o valor 27 em uma variável e em seguida divida esse valor por 3 e armazene em outra variável e por fim, imprima o resultado obtido na tela do computador.

Solução:

```
1 a = 27
2 x = a/3
3 print(x)
```

Exemplo 5:

Criar um programa que armazene o texto '*Um texto qualquer.*' em uma variável e em seguida imprima esse texto na tela do computador

Solução:

```
1 texto = 'Um texto qualquer.'
2 print(texto)
```

Exemplo 6 (Contraexemplo):

Suponha que você deseja resolver o seguinte problema matemático: **Calcule o valor de x a partir do valor de y, dado que o dobro de x equivale ao triplo de y.**

Você naturalmente formulará a seguinte expressão para resolver o problema: $2 * x = 3 * y$. Mas, caso você queira resolver o problema para $y = 10$ em um programa de computador, você não pode criar o programa usando diretamente esta expressão, veja:

Solução INCORRETA:

```
1 y = 10
2 2 * x = 3 * y
3 print(x)
```

A solução anterior está incorreta porque você está cometendo um **erro de sintaxe**, pois a atribuição é definida como:

$$nome_variavel = valor$$

e não temos no lado esquerdo do símbolo $=$ o nome de uma variável, temos a expressão $2 * x$.

Mesmo na Matemática, para resolver o problema seria recomendado realizar operações para adequar a equação para o cálculo de x, obtendo a expressão: $x = (3 * y)/2$.

E, a partir daí, já podemos representar o cálculo em um programa que respeite a sintaxe da atribuição de um valor para a variável x:

Solução CORRETA:

```
1 y = 10
2 x = (3 * y) / 2
3 print(x)
```

Obs.: Sempre existem diferentes formas de se implementar um programa, este é apenas um exemplo de solução correta.

2.4 Tipos de Dados

Quando você cria uma variável, os dados armazenados nela podem ter muitos tipos diferentes. Por exemplo, você pode armazenar a velocidade de um carro com um valor numérico e, em seguida, armazenar o nome do carro, usando um valor alfanumérico. O **Python** possui vários tipos de dados diferentes que você pode usar para armazenar tipos diferentes de valores, cada um com seu próprio conjunto de operações exclusivas que você pode usar para manipular e trabalhar com as variáveis.

O **Python** possui os variados tipos de variáveis, por exemplo: *Numéricos*, *String*, *Lógico*, *Listas*, *Tuplas*, *Dicionários*. Inicialmente, apresentaremos apenas os dois primeiros tipos.

Se você não está seguro de qual é o tipo de uma variável específica em seu programa, você pode usar a função **type** que retorna o tipo de uma variável.

Exemplo 7:

```
1 n = 14
2 print(type(n))
```

Na primeira linha, a variável **n** recebe o valor 14. É importante observar que, no momento em que fazemos uma atribuição de um valor para uma variável, também estamos definindo o tipo de dado da variável. Nesse caso, a variável **n** está sendo definida como **int**, pois ele recebeu um valor inteiro.

Na segunda linha, o comando **print** apresenta na tela do computador o tipo de dado da variável **n**. Para realizarmos a identificação do tipo, utilizamos a função **type**. Aqui temos como parâmetro da função **print** a função **type**, nesse caso, a função mais interna é executada primeiro, e o seu resultado será o parâmetro para a função mais externa. A função **type**, por sua vez, recebe como parâmetro a variável **n**. Desta forma, na execução do programa, para gerar o resultado esperado do comando da linha 2, primeiro é obtido o resultado de **type(n)**, que corresponde ao tipo do valor atribuído à variável, e em seguida este resultado é repassado para a função **print**, que realizará sua exibição na tela do computador.

2.4.1 Numéricos

Os tipos de dados **numéricos** são criados quando você cria uma variável usando um valor numérico reconhecido pela linguagem **Python**.

Exemplo 8:

Criar um programa capaz de armazenar o preço de um produto e em seguida, o programa deve apresentar o valor dessa variável na tela do computador.

Solução:

```
1 preco_produto = 4
2 print('O valor armazenado na variável é:')
3 print(preco_produto)
4 print('A variável é do tipo:')
5 print(type(preco_produto))
```

Aqui, atribuímos à variável **preco_produto** um valor numérico 4. Em seguida, imprimimos o tipo da variável. O **Python** aloca automaticamente a memória como um valor numérico.

O **Python** suporta 4 tipos de valores numéricos:

1. **int**: esse é o tipo de dado mais usado para manipular números inteiros. Seu valor máximo é 2.147.483.647 e seu valor mínimo é - 2.147.483.648.

2. **long**: esse é o tipo de dado mais usado para manipular números inteiros com valor muito alto.

3. **float**: esse tipo é usado para manipular números reais.

4. **complex**: esse tipo é usado para manipular números complexos.

2.4.2 String

Uma **string** é identificada como um conjunto de caracteres que deve estar entre aspas. O **Python** permite que os valores **string** sejam declarados com aspas simples (') ou duplas (").

Exemplo 9:

Criar um programa capaz de armazenar o nome do cliente e em seguida, o programa deve apresentar o valor dessa variável na tela do computador.

Solução:

```
1 nome_cliente = "João da Silva"
2 print('O valor armazenado na variável é:')
3 print(nome_cliente)
4 print('A variável é do tipo:')
5 print(type(nome_cliente))
```

Na linha 1 do código acima, atribuímos o conjunto de caracteres `"João da Silva"` para a variável **nome_cliente**. Utilizamos o comando **print** para exibir o conteúdo da variável na linha 3 e o tipo da variável na linha 5.

Exemplo 10:

Criar um programa capaz de guardar na memória dois valores e em seguida exibir o conteúdo armazenado em cada variável.

Solução:

```
1 a = 8
2 b = 4.5
3 print('Valores atribuídos: ')
4 print('a =')
5 print(a)
6 print('b =')
7 print(b)
```

2.5 Operadores aritméticos

Operadores aritméticos são símbolos especiais que representam cálculos matemáticos como adição, multiplicação e divisão. Os operadores aritméticos mais usados em **Python** são listados na Tabela 1. O seu funcionamento é como conhecemos na Matemática.

Exemplo 11:

Criar um programa capaz de **somar** dois números e em seguida apresentar o resultado na tela do computador.

Operador	Operação
$+$	Adição
$-$	Subtração
$*$	Multiplicação
$/$	Divisão
$**$	Potência

Tabela 1: Operadores aritméticos.

Solução:

```
1 a = 3
2 b = 6
3 c = a + b
4 print(c)
```

Obs.: Perceba que no exemplo acima foi criada uma variável (**c**) adicional para armazenar o resultado da soma. Ela é dispensável, pois poderia ser feito **print(a + b)**, assim, primeiro o resultado da expressão é calculado para depois o **print** fazer a exibição do resultado. A criação de variáveis adicionais é recomendado para facilitar a organização e entendimento do código, não é essencial neste caso, mas poderá ser em várias outras situações daqui para frente.

Exemplo 12:

Criar um programa para calcular a **diferença** entre dois números e em seguida, o programa deve apresentar o resultado na tela do computador.

Solução:

```
1 a = 3
2 b = 6
3 c = a - b
4 print(c)
```

Exemplo 13:

Criar um programa para calcular a **multiplicação** entre dois números e em seguida, o programa deve apresentar o resultado na tela do computador.

Solução:

```
1 a = 3
2 b = 6
3 c = a * b
4 print(c)
```

Exemplo 14:

Criar um programa para calcular a **divisão** entre dois números e em seguida, o programa deve apresentar o resultado na tela do computador.

Solução:

```
1 a = 6
2 b = 3
3 c = a / b
4 print(c)
```

Exemplo 15:

Criar um programa para calcular 9 elevado ao cubo e em seguida o programa deve apresentar o resultado na tela do computador.

Solução:

```
1 a = 9
2 b = 3
3 c = a ** b
4 print(c)
```

Exemplo 16:

Criar um programa para calcular a soma do primeiro número com o dobro do segundo número e ao final apresentar o resultado na tela.

Solução CORRETA:

```
1 a = 3
2 b = 6
3 c = a + b * 2
4 print(c)
```

Deve-se tomar cuidado, no entanto, ao agrupar expressões com mais de um operador. O resultado será diferente para se você calcular o dobro da soma dos dois números, que estará incorreto em relação ao que o exercício solicita, conforme solução **incorreta** apresentada a seguir.

Solução INCORRETA:

```
1 a = 3
2 b = 6
3 c = (a + b) * 2
4 print(c)
```

Exemplo 17:

Criar um programa capaz de inicializar duas variáveis atribuindo numéricos e em seguida somar essas duas variáveis e por fim exibir o resultado da soma no monitor do computador.

Solução :

```
1 a = 7
2 b = 11
3 c = a + b
4 print('Soma =')
5 print(c)
```

2.5.1 Precedência e associatividade de operadores

A **precedência** de operadores indica qual operador deverá ser executado primeiro:

- Assim, na expressão aritmética $2 + 36$, a sub-expressão 36 é executada primeiro;
- Portanto, tem-se como resultado para a expressão o valor 20.

Exemplo 18:

Criar um programa para calcular e imprimir o resultado da seguinte expressão matemática: $2+3*6$.

Solução :

```
1 resultado = 2 + 3 * 6
2 print(resultado)
```

A **associatividade** define a regra usada quando os operadores possuem a mesma precedência:

- Define-se que a ordem de avaliação será da esquerda para a direita ou o contrário, da direita para a esquerda;
- Para a expressão $A - B + C + D$, o termo $A - B$ é avaliado primeiro;
- Para a expressão $A**B**C**D$, o termo $C**D$ é avaliado primeiro.

Exemplo 19:

Criar um programa para calcular e imprimir o resultado das seguintes operações matemáticas: $5 - 4 + 2$ e $5**4**2$.

Solução :

```
1 e = 5 - 4 + 2
2 print(e)
3 e = 5 ** 4 ** 2
4 print(e)
```

Qual a ordem de execução das operações? Utilize a depuração do **Thonny** para observar como o Python executa estas expressões.

2.6 Funções elementares

A linguagem **Python** possui diferentes módulos (ou bibliotecas) que fornecem diversas funções úteis para o programador. Um desses módulos é o **math** que fornece várias funções matemáticas prontas que são úteis para realizar cálculos matemáticos.

Para usar qualquer módulo em **Python** é necessário importá-lo antes. Por exemplo, para usar o módulo **math** temos que acrescentar o seguinte comando no início do programa:

```
1 import math
```

Após o comando de importação do módulo, é possível usar as funções definidas por ele. A importação é necessária apenas uma vez para cada programa.

A seguir exploraremos várias funções elementares, apenas uma não pertence ao módulo **math**, por isso iniciaremos por ela. A Tabela 2 sumariza todas estas funções.

Exemplo 20: função abs(x):

Função para calcular o valor absoluto do argumento x, ou seja, a distância (positiva) entre x e zero.

```
1 resultado = abs(-12)
2 print(resultado)
```

Na primeira linha, a variável **resultado** recebe o valor absoluto de -12, que é calculada a partir da função **abs**. Na segunda linha, temos o comando **print** que apresenta na tela do computador o valor que está armazenado na variável **resultado**.

Observe que neste exemplo não foi necessário importar o módulo **math**, pois ela pertence ao módulo padrão do **Python**, assim como função **print** usada em todos os exemplos.

Outra observação importante é em relação ao "**argumento x**". Uma função que necessita receber algum valor (um ou mais) para realizar sua tarefa, recebe estes valores a partir de **argumentos**, aqui o nome **x** é usado apenas de forma ilustrativa, tanto que ele não apareceu no programa de exemplo. Ao utilizar a função **abs**, que recebe apenas um argumento, nós passamos o valor que será utilizado pela função, neste caso o valor -12. Qualquer valor numérico ou expressão que resulte em valor numérico poderá ser utilizado, por exemplo, o resultado de **abs(1 - 13)** será o mesmo de **abs(-12)**, já que $1 - 13 = -12$.

Agora exploraremos algumas funções do módulo **math**.

Exemplo 21: função sqrt(x):

É uma função para calcular a raiz quadrada de do argumento x.

```
1 import math
2 resultado = math.sqrt(9)
3 print(resultado)
```

A primeira linha importa o módulo **math**, permitindo assim o de todas as suas funções dentro do programa. Na segunda linha, a variável **resultado** recebe a raiz quadrada de 64, que é calculada a partir da função **sqrt**. Na terceira linha, temos o comando **print** que apresenta na tela do computador o valor que está armazenado na variável **resultado**.

Exemplo 22: função cos(x):

Função para calcular o cosseno do argumento x, em radianos.

```
1 import math
2 resultado = math.cos(30)
3 print(resultado)
```

A primeira linha importa o módulo **math**. Na segunda linha, a variável **resultado** recebe o cosseno de 30 radianos, que é calculada a partir da função **cos**. Na terceira linha, temos o comando **print** que apresenta na tela do computador o valor que está armazenado na variável **resultado**.

Exemplo 23: função tan(x):

Função para calcular a tangente do argumento x, em radianos.

```
1 import math
2 resultado = math.tan(3.1417)
3 print(resultado)
```

A primeira linha importa o módulo **math**. Na segunda linha, a variável resultado recebe a tangente de 3.1417, que é calculada a partir da função **tan**.

Na terceira linha, temos o comando **print** que apresenta na tela do computador o valor que está armazenado na variável resultado.

Exemplo 24: função sin(x):

Função para calcular o seno de do argumento x.

```
1 import math
2 resultado = math.sin(3.1417)
3 print(resultado)
```

Exemplo 25: função log(x):

Calcula o **logaritmo natural** do argumento x.

```
1 import math
2 resultado = math.log(10)
3 print(resultado)
```

Exemplo 26: função log10(x):

Calcula o logaritmo de x na base 10.

```
1 import math
2 resultado = math.log10(1000)
3 print(resultado)
```

Exemplo 27: função log2(x):

Calcula o logaritmo de x na base 2.

```
1 import math
2 resultado = math.log2(256)
3 print(resultado)
```

Exemplo 28: função log(x, b):

Calcula o logaritmo de x na base b.

```
1 import math
2 resultado = math.log(81, 3)
3 print(resultado)
```

Calcular o logaritmo na base 2 ou base 10 usando **log2(x)** e **log10(x)**, resulta em valores mais precisos do que **log(x, 2)** e **log(x, 10)**, já que são funções específicas para estas duas bases e utilizam algoritmos mais precisos para realizar os cálculos necessários.

2.6.1 Resumo das funções elementares

A Tabela 2 apresenta um resumo das funções apresentadas na seção anterior. Lembre-se de que todas são fornecidas pelo módulo **math**, à exceção apenas da função **abs**.

2.7 Constantes

A linguagem **Python** possui algumas constantes, como por exemplo: **pi**, **número de Euler** e **Tau**. Constantes úteis em diversos cálculos matemáticos são fornecidas no módulo **math**. A Tabela 3 apresenta um resumo de algumas das constantes fornecidas.

Função	Nome	Exemplo	Resultado
Valor absoluto	**abs**	abs(-2)	2
Raiz quadrada	math.**sqrt**	sqrt(9)	3
Cosseno	math.**cos**	cos(30)	0.1542514
Tangente	math.**tan**	tan(7.3456)	1.7945721
Seno	math.**sin**	sin(pi)	1.2246e-16
Parte inteira	math.**trunc**	trunc(5.2)	5
Menor inteiro maior que	math.**ceil**	ceil(5.2)	6
Maior inteiro menor que	math.**floor**	floor(5.2)	5
Potência	math.**pow**	pow(2,3)	8
Logaritmo natural	math.**log**	log(148.41315910257657)	5
Logaritmo base 10	math.**log10**	log10(1000)	3
Logaritmo base 2	math.**log2**	log2(256)	8
Logaritmo qualquer base	math.**log**	log(81,3)	4

Tabela 2: Funções elementares.

Constante	Nome	Valor
Pi (π)	math.**pi**	3.141592..., até a precisão disponível
Número neperiano (e)	math.**e**	2.718281..., até a precisão disponível
Tau (τ)	math.**tau**	6.283185..., até a precisão disponível
Infinito (∞)	math.**inf**	Uma representação do infinito positivo

Tabela 3: Algumas constantes do módulo **math**.

A utilização das constantes é semelhante ao das funções elementares apresentadas anteriormente, à exceção de que para constantes **não é necessário fazer a passagem de argumentos**, conforme exemplos a seguir.

Exemplo 29: constante pi (π):

A constante matemática π é utilizada em diversas situações. Uma delas é para se calcular o comprimento de uma circunferência, através da equação:

$$C = 2 * \pi * r,$$

onde C é o comprimento e r o raio da circunferência.

O programa a seguir pode ser usado para calcular o comprimento de uma circunferência de raio igual a 100 metros.

```
1 import math
2 r = 100
3 C = 2 * math.pi * r
4 print(C)
```

Observe que, para usar a constante π nós importamos o módulo **math** e utilizamos **math.pi**, que representa o valor numérico de π com a maior precisão possível na linguagem **Python**.

Exemplo 30: número de Euler (ou neperiano) (e):

O número neperiano também é muito útil em diversas situações, como em logaritmos naturais por exemplo. Uma outra utilização é para o cálculo de juros compostos, onde o *montante* após um tempo

t, M_t, a uma taxa de juros composto r é definido como:

$$M_t = M_i * e^{r*t},$$

onde M_i é o *montante* inicial.

O programa a seguir pode ser utilizado para calcular o valor a ser pago considerando um montante inicial de R$ 1.000,00, com juros composto a uma taxa mensal de 1,5%, durante 12 meses.

```
1 import math
2 Mi = 1000
3 r = 0.015
4 t = 12
5 Mt = Mi * math.e ** (r*t)
6 print(Mt)
```

2.8 Exercícios Propostos

Exercício 1

Qual é o valor impresso ao final da seguinte sequência de comandos?

```
1 dia = "quinta-feira"
2 dia = 32.5
3 dia = 19
4 print(dia)
```

Exercício 2

O seguinte nome é válido para uma variável em **Python**? Justifique sua resposta.

```
1 Uma_boa_nota_é_A+ = 9
```

Exercício 3

Um prêmio em dinheiro será dividido entre três ganhadores de um concurso.

- O primeiro ganhador receberá 46% do valor total do prêmio
- O segundo ganhador receberá 32% do valor total do prêmio
- O terceiro receberá o restante

Escreva um programa que defina uma variável com o valor **780000** para o prêmio e, em seguida, calcule e imprima a quantia que deve ser distribuída para cada um dos ganhadores.

Exemplos de execução:

Exemplo 1:

```
O primeiro ganhador deve receber R$:
358800.0
O segundo ganhador deve receber R$:
249600.0
O terceiro ganhador deve receber R$:
171600.0
```

Exercício 4

Faça um programa que defina a largura e a altura de uma parede em 2.8 e 5.2 metros em duas variáveis. Posteriormente, calcule a sua área e a quantidade de tinta necessária para pintá-la, sabendo que cada litro de tinta pinta uma área de 2 metros quadrados.

Exemplos de execução:

Exemplo 1:

```
Área da parede em metros quadrados:
14.559999999999999
Litros de tinta necessários para pintar essa parede:
7.279999999999999
```

Exercício 5

Escreva um programa que defina, em variáveis, a quantidade de Km percorridos por um carro alugado (com valor 879) e a quantidade de dias pelos quais ele foi alugado (com valor 6). Calcule o preço a pagar, sabendo que o carro custa R$ 60,00 por dia e R$ 0,15 por Km rodado.

Exemplos de execução:

Exemplo 1:

```
O cliente deve pagar R$:
491.85
```

Exercício 6

Faça um programa que defina uma variável com área de um quadrado no valor de 256 e calcule o valor de cada lado do quadrado.

Exemplos de execução:

Exemplo 1:

```
Cada lado tem comprimento igual à:
16.0
```

Exercício 7

Faça um programa que defina uma variável com o raio de uma circunferência no valor de 1.2 e calcule sua área. Utilize a fórmula:

$$\text{area} = \pi * (\text{raio}^2)$$

Utilize a constante π definida na linguagem.

Ao final, o seu programa deve apresentar o valor da área da circunferência calculada.

Exemplos de execução:

Exemplo 1:

```
A área dessa circunferência é:
4.523893421169302
```

MÓDULO

3

Entrada e Saída

Normalmente, em um programa de computador, é necessário interagir com o usuário. Em algum momento é necessária a inserção de dados, comumente definidos como **entradas** do usuário, que posteriormente serão manipulados por um conjunto de operações definidas no programa. Depois, com os dados obtidos através das operações executadas, o resultado poderá ser apresentado como um retorno ou uma **saída** para o usuário do programa. O fluxograma da Figura 7 ilustra o cenário descrito com as principais etapas de um programa que fornece instruções de entrada e saída.

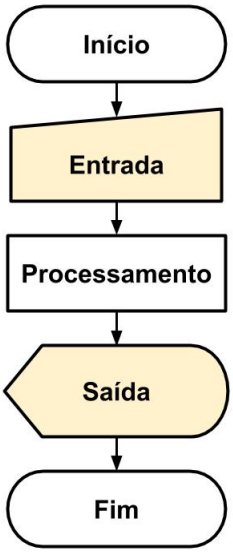

Figura 7: Fluxograma de exemplo para entrada e saída.

Na linguagem de programação **Python**, são fornecidos mecanismos para a **entrada** e **saída** através do *terminal*, a função **input()** realiza a entrada e a função **print()** realiza a saída, e são apresentadas a seguir. Por uma questão didática, começaremos pela **saída** e em seguida a **entrada**.

3.1 Saída: função print

A **saída** de dados em **Python**, como descrito anteriormente, através de sua interface padrão, é feita por meio da função **print**.

Abaixo descrevemos a sintaxe da função **print**:

`print(`*mensagem*`)`

onde:

- **print**: nome da função responsável por exibir mensagens como saída para o usuário.
- *mensagem*: argumento de entrada para a função. É uma **string** (**valor textual**) que será exibida como mensagem no terminal. Normalmente é uma **string** gerada a partir de informações de variados tipos, conforme veremos a seguir.

O exemplo seguinte apresenta o uso da função **print** gerando como saída para o usuário a frase `"Olá Mundo!"`.

```
1 frase = 'Olá Mundo!'
2 print(frase)
```

3.1.1 Strings de formatação

Em muitos cenários, um conjunto heterogêneo de dados poderá ser apresentado como saída ao usuário. Neste sentido, a linguagem **Python** oferece vários recursos para a formatação e posteriormente a apresentação destes dados. Um destes recursos consiste no uso de uma forma especial de **literais string** onde é possível intercalar valores de expressões no meio do texto da **string**.

Uma **string de formatação** (ou simplesmente **f-string**) é escrita colocando-se a letra **f** imediatamente antes do delimitador que começa a **string** (aspas ou apóstrofo). Por exemplo:

`f'Olá, mundo!'`

Em uma **f-string**, os caracteres **abre chave** (**{**) e **fecha chave** (**}**) são especiais, e são usados para intercalar valores de expressões na **string**. Basta escrever a expressão desejada entre chaves dentro da **f-string**, e o **Python** se encarregará de avaliar a expressão e inserir o seu valor no meio dela, onde o programador colocou as chaves. Por exemplo:

`f'A soma de 2 e 3 é {2+3}, certo?'`

Nesta **f-string** a expressão entre chaves (**2+3**) é avaliada e o seu resultado (**5**) é inserido na **string**, substituindo a expressão e as chaves. O texto gerado é:

`'A soma de 2 e 3 é 5, certo?'`

Podemos usar strings de formatação como argumentos para a função **print** para produzirmos saídas de dados formatadas em nossos programas.

Exemplo 1:

Faça um programa que pergunta ao usuário o seu nome, e exibe uma saudação de boas-vindas, incluindo o nome na saudação.

```
1 nome = input('Qual é o seu nome? ')
2 print(f'Bom dia {nome}, seja bem vindo(a)!')
```

Exemplo 2:

Escreva um programa que exibe a média aritmética de dois números disponíveis em duas variáveis.

```
1 x = 5.6
2 y = 8.8
3 media = (x + y) / 2
4 print(f'A média aritmética de {x} e {y} é {media}')
```

3.1.2 Especificando a formatação

O **Python** permite a indicação de como os valores das expressões devem ser formatados ao serem inseridos em uma **f-string**. Para tanto basta colocar, imediatamente após a expressão, antes de fechar as chaves, o caractere dois pontos ("**:**") seguido da indicação de formatação desejada.

Uma **especificação de formatação simplificada** (*existem opções avançadas*) consiste em indicar:

1. (*opcional*) a **largura** do campo (quantidade mínima de caracteres a ser usada);
2. (*opcional*) o caractere "**.**" seguido da **precisão** para números reais (quantidade de casas decimais);
3. o **tipo** do dado:
   - "**s**" para **string** (sequência de caracteres);
   - "**d**" para **decimal** (valor numérico inteiro em notação decimal);
   - "**f**" para **float** (valor numérico real).

Exemplo 3:

Para demonstrar o uso do operador de formatação observe o bloco de códigos abaixo.

```
1 cidade     =  'Ouro Preto'
2 celsius    =  32
3 fahrenheit =  89.60
4 print(f'A temperatura em {cidade}:')
5 print(f'{celsius} Celsius')
6 print(f'{fahrenheit:.2f} Fahrenheit')
```

Neste código foram definidas e inicializadas três variáveis, nas linhas 1 a 3, que armazenam valores de tipos distintos de dados. Na linha 4, a função **print**, como visto anteriormente, tem como parâmetro uma **f-string** correspondente à mensagem a ser exibida. Observe a ocorrência da variável **cidade** entre chaves – o seu valor será inserido na mensagem. Na linha 5, a função **print** também é usado com uma **f-string** na qual é inserido o valor da variável **celsius**. Já na linha 6, o valor da variável **fahrenheit** é inserido em uma **f-string** usando duas casas decimais.

Exemplo 4:

O programa a seguir ilustra as indicações de tipo e tamanho do campo na formatação do valor inserido em uma **f-string**.

```
1  x = 1
2  y = 10
3  print('Sem formatação:')
4  print(f'x = {x}')
5  print(f'y = {y}')
6  print('Formatação para número inteiro:')
7  print(f'x = {x:d}')
8  print(f'y = {y:d}')
9  print('Formatação para número inteiro com largura mínima igual a 10:')
10 print(f'x = {x:10d}')
11 print(f'y = {y:10d}')
```

Neste programa estão declaradas duas variáveis **x** e **y**, inicializadas com valores do tipo inteiro. Observe que a especificação de formatação "**:10d**" reserva um espaço mínimo de dez caracteres para a inserção do valor da expressão. O espaço não utilizado é preenchido com espaço em branco à esquerda do número.

Exemplo 5:

O código a seguir mostra como o número de casas decimais pode ser ajustado na formatação de números reais.

```
1  x = 1.1111111111
2  y = 1.88888888
3  z = 2.0
4  print('Sem formatação:')
5  print(f'x = {x}')
6  print(f'y = {y}')
7  print(f'z = {z}')
8  print('Formatação para número real:')
9  print(f'x = {x:f}')
10 print(f'y = {y:f}')
11 print(f'z = {z:f}')
12 print('Formatação para número real com largura mínima igual a 10:')
13 print(f'x = {x:10f}')
14 print(f'y = {y:10f}')
15 print(f'z = {z:10f}')
16 print('Formatação para número real com largura mínima igual a 10 e
       precisão igual a 3:')
17 print(f'x = {x:10.3f}')
18 print(f'y = {y:10.3f}')
19 print(f'z = {z:10.3f}')
```

Este programa faz uso de três variáveis **x**, **y** e **z** para ilustrar a formatação de números reais. A especificação de formatação "**:10.3f**" reserva um espaço mínimo de largura **dez** para inserir um número real, que será formatado com **três** casas decimais.

Exemplo 6:

Para ilustrar a formatação **strings** usamos o seguinte programa.

```
1  oi = 'Olá Mundo'
2  tchau = 'Adeus mundo cruel'
3  print('Sem formatação:')
4  print(f'{oi}!!!')
5  print(f'{tchau}!!!')
6  print('Formatação para string:')
7  print(f'{oi:s}!!!')
8  print(f'{tchau:s}!!!')
9  print('Formatação para string com largura mínima igual a 25:')
10 print(f'{oi:25s}!!!')
11 print(f'{tchau:25s}!!!')
```

A especificação de formatação "**:25s**" usada no exemplo insere a **string** em um espaço de largura mínima de **25** caracteres.

3.1.3 Inserção de caracteres especiais em uma string

Em algumas situações desejamos incluir **caracteres especiais**, geralmente que não tem uma representação gráfica, em um literal **string**. **Python** permite o uso de **sequências de escape** dentro de uma **string** para a inclusão destes caracteres.

Uma sequência de escape é formada pelo caractere "\" seguido de um código que identifica o caractere especial a ser incluído na **string**.

Algumas sequências de escape são listadas a seguir:

- \n: Representa o caractere de nova linha. Quando digitado no programa faz com que o cursar avance para a próxima linha naquela posição.
- \t: Representa o caractere de tabulação horizontal. Quando digitado no programa faz com que o cursor avance para a próxima parada de tabulação. Normalmente as paradas de tabulação ocorrem a cada oito posições.
- \": Representa o caractere aspas. Deve ser usado para inserir aspas em um literal **string** delimitado por aspas.
- \': Representa o caractere apóstrofo. Deve ser usado para inserir apóstrofo em um literal **string** delimitado por apóstrofos.
- \\: Representa o caractere barra inclinada para a esquerda. Deve ser usado para inserir a barra, pois a barra por si só já é especial, uma vez que é usada para começar uma sequência de escape.
- Se um literal **string** é prefixado com as letras "r" ou "R" este mecanismo de sequências de escape é desativado, e todos os caracteres são inseridos literalmente na **string**.
- Em um literal **f-string** os caracteres "{" e "}" são especiais e indicam a inserção do valor de uma expressão. Para inserirmos estes caracteres na **string** precisamos escrevê-los duas vezes em sequência na **string**: "{{" e "}}".

Exemplo 7:

O programa abaixo exibe várias mensagens com ocorrências de sequências de escape em literais string.

```
1 print('Ouro\tPreto\nMinas\tGerais')
2 print('\n')
3 print('Uma aspa \" no meio da string')
4 print('Um apóstrofo \' no meio da string')
5 print('Uma barra \\ no meio da string')
6 print(r'Aqui o caracter \ não é especial')
7 print(R'Nesta string \n não é nova linha')
8 print(f'Chaves {{ aqui }} em uma string de formatação')
```

3.2 Entrada: função input

Abaixo a sintaxe da função **input**:

input(*mensagem*)

onde:

- **input**: nome da função responsável pela entrada de dados informados pelo usuário;
- *mensagem*: argumento de entrada para a função. É uma **string** (**valor textual**) que será exibida como mensagem no terminal para orientar o usuário.

Abaixo, como exemplo, a função **input** imprime a mensagem `"Informe um nome: "` e aguarda a entrada de dados pelo usuário. Após a inserção dos dados é necessário um clique na tecla **ENTER** para indicar o término da entrada e o retorno para o programa.

```
1 input('Informe um nome: ')
```

É comum no desenvolvimento, que os dados fornecidos sejam armazenados em variáveis para posteriormente serem manipulados em algum ponto do programa. Em **Python** as entradas feitas através da função **input** podem ser atribuídas (usando o operador de atribuição "=") diretamente a uma variável.

Abaixo a sintaxe da função **input** em conjunto com a atribuição de dados a uma variável:

variável = **input**(*mensagem*)

onde:

- *variável*: nome da variável que armazenará o valor da entrada;
- =: operador de atribuição;
- **input**: nome da função responsável pela entrada de dados informados pelo usuário;
- *mensagem*: argumento de entrada para a função. É uma **string** (**valor textual**) que será exibida como mensagem no terminal para orientar o usuário.

Abaixo, um exemplo com a atribuição, primeiro a função **input** exibirá a mensagem na tela, aguardará até que o usuário digite algo e conclua com a tecla **ENTER**, e depois, o valor informado será atribuído à variável como um dado do tipo **string** (**str**), ou seja, uma sequência de caracteres digitada pelo usuário através do terminal.

```
1 nome = input('Informe um nome: ')
```

3.2.1 Conversão de tipos

O resultado da função **input** será sempre um valor **string**, independente do conteúdo informado pelo usuário. Ou seja, para a realização de cálculos matemáticos por exemplo, será necessário converter o valor **string** para inteiro (**int**) ou real (**float**), conforme os exemplos a seguir.

3.2.2 Conversão para inteiro: função int

Converte um valor para o tipo numérico **inteiro**:

```
1 x = 10.7
2 y = '10'
3 print(f'Valor de x original (real): {x}')
4 print(f'Valor de x como um inteiro: {int(x)}')
5 print(f'Valor de y original (string): {y}')
6 print(f'Valor de y como um inteiro: {int(y)}')
```

3.2.3 Convertão para real: função float

Converte um valor para o tipo numérico **real**:

```
1 x = 10
2 y = '10.2'
3 print(f'Valor de x original (inteiro): {x}')
4 print(f'Valor de x como um real: {float(x)}')
5 print(f'Valor de y original (string): {y}')
6 print(f'Valor de y como um real: {float(y)}')
```

3.2.4 Convertão para string: função str

Converte um valor para o tipo **string**:

```
1 var_inteira = 10
2 var_string = str(var_inteira)
3 soma = var_inteira + 5
4 concatenacao = var_string + '5'
5 print(soma)
6 print(concatenacao)
```

Como resultado da execução do programa anterior teríamos como valor para a variável **soma** o número inteiro 15, que será impresso no primeiro **print**, como valor para a variável **var_string** o valor string `"10"` e como valor para a variável **concatenação** o valor string `"105"`, que será impresso pelo segundo **print**.

Observe também que aqui o operador "+" funciona de maneira distinta para tipos numéricos e tipos string. Para valores numéricos ele realiza a **soma matemática** e para valores string ele realiza a **concatenação textual**. O **Python** não permite usar este operador para um operando numérico e outro operando string, se você tentar fazer isso ocorrerá um erro no programa.

3.2.5 Cuidados especiais com conversão de tipos

Existe um problema com a conversão de tipos envolvendo **strings**. Se o texto não puder ser representado pelo tipo convertido, ocorrerá um erro de execução. Isto vale para qualquer tipo de conversão. Veja os exemplos a seguir.

```
1 y = '10.2'
2 print(f'Valor de y como um inteiro: {int(y)}')
```

```
1 y = 'ops'
2 print(f'Valor de y como um inteiro: {int(y)}')
```

```
1 y = 'ops'
2 print(f'Valor de y como um real: {float(y)}')
```

3.2.6 Conversão de dados fornecidos pelo usuário

Dados fornecidos pelo usuário são normalmente armazenados em uma variável, assim, basta converter o valor da variável fornecida:

```
1 peso = input('Forneça o peso total (número real): ')
2 peso = float(peso)
3 unid = input('Forneça a quantidade de unidades (número inteiro): ')
4 unid = int(unid)
5 print(f'O peso de cada unidade é {peso/unid:.2f}')
```

Observe que os valores esperados pelo programa anterior são: **número real** para o peso e **número inteiro** para a quantidade de unidades. Caso o usuário forneça valores que não possam ser convertidos, ocorrerá erro de execução, pois as conversões feitas levam em consideração os tipos esperados.

Outra observação interessante é que foram usadas as próprias variáveis atribuídas pelos **inputs** para receberem os resultados das conversões. Isto é possível porque o **Python** é uma linguagem **dinamicamente tipada**, o que significa que as variáveis podem mudar de tipo durante a execução do programa.

Porém, sempre que precisarmos fazer a conversão desta forma, podemos fazê-la em uma única linha de instrução, colocando o **input** dentro da função de conversão utilizada:

```
1 peso = float(input('Forneça o peso total (real): '))
2 unid = int(input('Forneça a quantidade de unidades (inteiro): ' ))
3 print(f'O peso de cada unidade é {peso/unid:.2f}')
```

O resultado final é exatamente o mesmo, mas obtemos um programa mais compacto e ele realiza menos operações (**economizamos** duas atribuições).

3.3 Entrada e Saída: Exemplos Práticos

Exemplo 8:

Implemente um programa que leia dois valores inteiros e calcule sua soma, armazenando o resultado em uma variável. A seguir imprima o resultado da soma.

Solução:

Para a resolver este problema, devemos começar pensando em um **encadeamento de passos**, ou seja, um **algoritmo**, a ser executado:

1. Entrada dos dados: duas variáveis com valor inteiro;
2. Cálculo da soma dos valores das duas entradas;
3. Saída do resultado: valor da soma.

O fluxograma da Figura 8 apresenta estas etapas de uma forma gráfica e um pouco mais detalhada. Com isso, fica mais fácil converter as etapas em código para a construção da solução que é apresentada de forma **iterativa** a seguir.

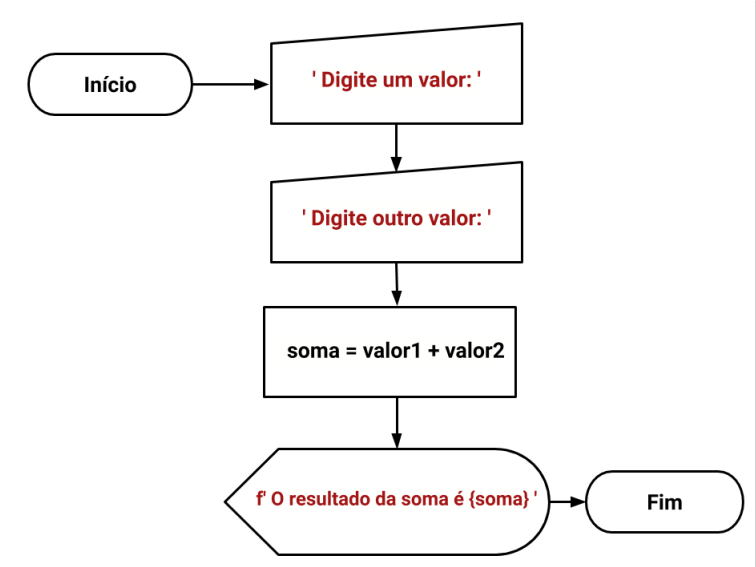

Figura 8: Fluxograma de solução para o exemplo 8.

Como primeiro passo definimos as entradas dos dados. No exemplo são solicitadas as leituras de dois valores inteiros, assim declaramos duas variáveis **valor1** e **valor2**. E através da função **input** associada à função **int**, definimos as entradas:

```
1 # Entrada
2 valor1 = int(input('Digite um valor: '))
3 valor2 = int(input('Digite outro valor: '))
```

Na etapa seguinte, declaramos a variável **soma** que será responsável por receber e armazenar o valor correspondente ao somatório dos valores contidos nas variáveis **valor1** e **valor2**:

```
1 # Entrada
2 valor1 = int(input('Digite um valor: '))
3 valor2 = int(input('Digite outro valor: '))
4 # Processamento
5 soma = valor1 + valor2
```

No final, como saída para o usuário, utilizaremos a função **print** e como argumento da função: a **f-string** *f'O resultado da soma é {soma}'* onde o valor da variável **soma** será inserido no texto na respectiva posição.

```
# Entrada
valor1 = int(input('Digite um valor: '))
valor2 = int(input('Digite outro valor: '))
# Processamento
soma = valor1 + valor2
# Saída
print(f'O resultado da soma é {soma}')
```

Exemplo 9:

Modifique o programa anterior, onde os dois valores passam a ser reais e o resultado da soma o numerador de uma divisão. O denominador será um novo valor real lido. O programa imprime apenas o resultado final da divisão, que também deverá estar armazenado em uma variável. A saída do resultado deve ser arredondada para duas casas decimais de precisão.

Solução:

Novamente, vamos começar pensando em um **algoritmo**:

1. Entrada dos dados: três variáveis com valor real;
2. Cálculo da soma dos dois primeiros valores;
3. Cálculo do resultado final: soma dividida pelo terceiro valor;
4. Saída do resultado final: valor da divisão.

Agora vamos observar o fluxograma da Figura 9 e convertê-lo em código de forma **iterativa** a seguir.

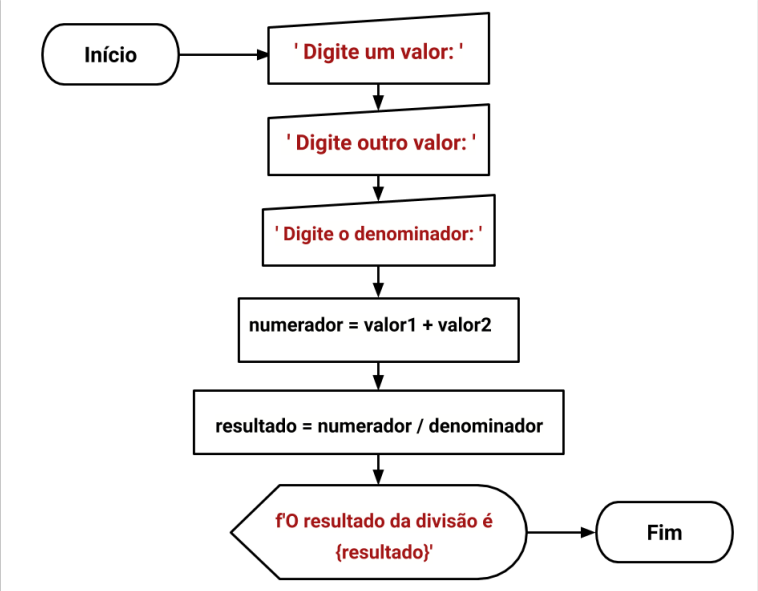

Figura 9: Fluxograma de solução para o exemplo 9.

Na resolução deste exemplo, declaramos três variáveis **valor1**, **valor2** e **denominador**, que serão informadas pelo usuário. As variáveis **valor1** e **valor2** armazenarão os valores que serão somados. E a variável **denominador** armazenará o valor de denominador utilizado posteriormente na operação de divisão.

```
# Entrada
valor1 = float(input('Digite um valor: '))
valor2 = float(input('Digite outro valor: '))
denominador = float(input('Digite o denominador: '))
```

Na etapa seguinte definimos a variável **numerador**, sendo esta inicializada com a soma dos valores contidos nas variáveis **valor1** e **valor2**. E a variável **resultado**, que é inicializada com a divisão do valor armazenado na variável **numerador** pela variável **denominador**.

```
# Entrada
valor1 = float(input('Digite um valor: '))
valor2 = float(input('Digite outro valor: '))
denominador = float(input('Digite o denominador: '))
# Processamento
numerador = valor1 + valor2
resultado = numerador / denominador
```

No final, como solução para o exemplo, definimos uma saída através da função **print**, que recebe a **f-string** *f'O resultado da divisão é {resultado:.2f}'* como argumento. Observe que, conforme solicitado no enunciado, formatamos a saída com duas casas decimais de precisão.

```
# Entrada
valor1 = float(input('Digite um valor: '))
valor2 = float(input('Digite outro valor: '))
denominador = float(input('Digite o denominador: '))
# Processamento
numerador = valor1 + valor2
resultado = numerador / denominador
# Saída
print(f'O resultado da divisão é {resultado:.2f}')
```

Exemplo 10:

Implemente um programa que imprima a hipotenusa (h) de um triângulo retângulo de acordo com a leitura de seus catetos (a e b). A saída deve ser arredondada para três casas decimais.

Observação: $h = \sqrt{a^2 + b^2}$.

Solução:

Novamente, o algoritmo segue uma estrutura semelhante: 1. Entradas; 2. Processamento; 3. Saída; que será diretamente representado pelo fluxograma da Figura 10.

Inicialmente declaramos as variáveis **a** e **b**, responsáveis por armazenar os valores correspondentes aos catetos, e serão inicializadas pelo usuário através da função **input** e seus valores convertidos pela função **float**.

```
# Entrada
a = float(input('Digite o cateto a: '))
b = float(input('Digite o cateto b: '))
```

Na sequência, declaramos a variável **h** que armazenará o valor do cálculo da hipotenusa. Note que para calcular o valor da raiz utilizamos a função **sqrt**, disponível no módulo **math** através da declaração da diretiva **import** no início do programa.

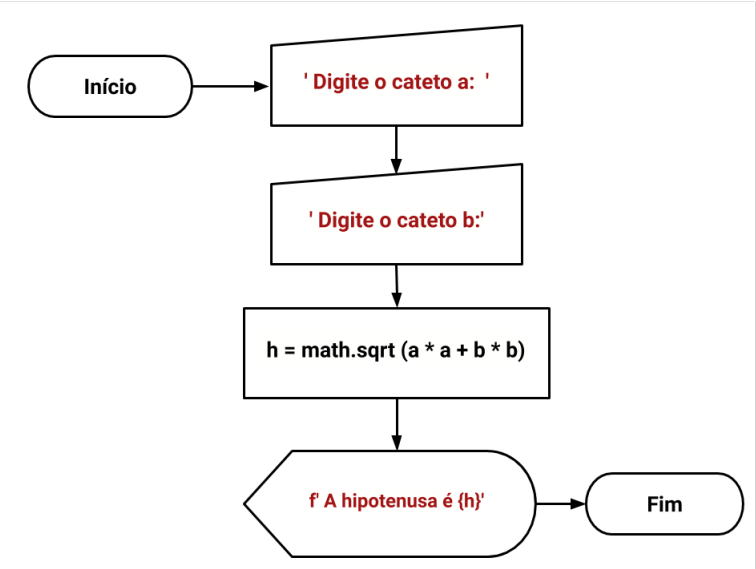

Figura 10: Fluxograma de solução para o exemplo 10.

```
1 import math
2 # Entrada
3 a = float(input('Digite o cateto a: '))
4 b = float(input('Digite o cateto b: '))
5 # Processamento
6 h = math.sqrt(a*a + b*b)
```

Ao final, como saída do programa definiremos como parâmetros para a função **print** a **f-string** *f'A hipotenusa é {h:.2f}'*, onde o valor da hipotenusa armazenado na variável **h** é inserido com três casas decimais.

```
1 import math
2 # Entrada
3 a = float(input('Digite o cateto a: '))
4 b = float(input('Digite o cateto b: '))
5 # Processamento
6 h = math.sqrt(a*a + b*b)
7 # Saída
8 print(f'A hipotenusa é {h:.3f}')
```

Exemplo 11:

Implemente um programa que leia do teclado um valor de temperatura em Celsius (C), calcule e imprima essa temperatura em Fahrenheit (F) e em Kelvin (K), com precisão de uma casa decimal.

Equações de conversão:

$$F = C * 1.8 + 32$$

$$K = C + 273.15$$

Solução:

O **algoritmo** tem as mesmas características dos exemplos anteriores, e seu fluxograma é apresentado na Figura 11.

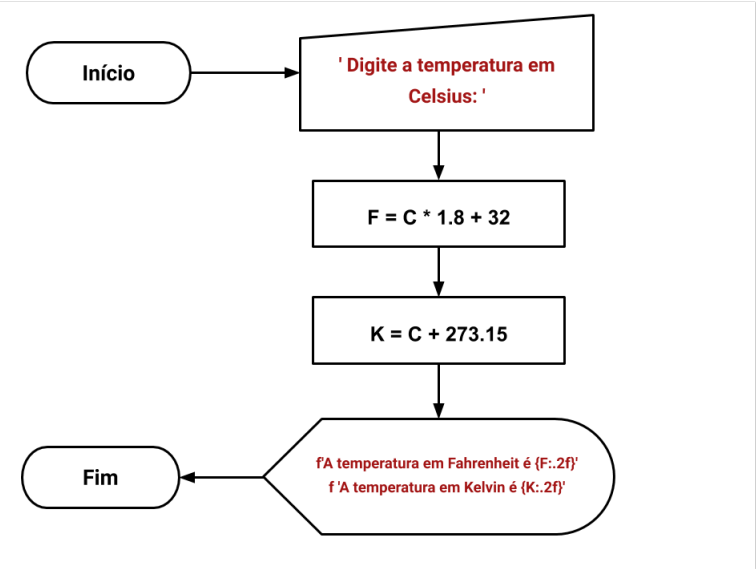

Figura 11: Fluxograma de solução para o exemplo 11.

O programa construído é apresentado a seguir:

```
1 # Entrada
2 C = float(input(' Digite a temperatura em Celsius: '))
3 # Processamento
4 F = C * 1.8 + 32
5 K = C + 273.15
6 # Saída
7 print(f'A temperatura em Fahrenheit é {F:.1f}')
8 print(f'A temperatura em Kelvin é {K:.1f}')
```

3.4 Exercícios Propostos

Exercício 1

Elabore um programa que leia o valor do raio de um círculo qualquer (número inteiro) e calcule sua respectiva área. Ao final imprima como saída o valor da área do círculo (com duas casas decimais).

Fórmula:

$$rea = \pi.r^2$$

onde:

- π : constante **Pi**;
- r: raio do círculo.

Exemplos de execução:

Exemplo 1:

```
Digite o valor do raio: 5
A área do círculo é: 78.54
```

Exemplo 2:

```
Digite o valor do raio: 9
A área do círculo é: 254.47
```

Exemplo 3:

```
Digite o valor do raio: 15
A área do círculo é: 706.86
```

Exercício 2

Implemente um programa que leia o valor do raio **r** de uma esfera (número real) e calcule o volume da esfera. No final imprima como saída o valor calculado (com duas casas decimais).

Fórmula:

$$Volume = \frac{4\pi r^3}{3}$$

onde:

- π : constante **Pi**;
- r: raio do círculo.

Exemplos de execução:

Exemplo 1:

```
Digite o valor do raio: 5
O volume da esfera é: 523.60
```

Exemplo 2:

```
Digite o valor do raio: 10.5
O volume da esfera é: 4849.05
```

Exemplo 3:

```
Digite o valor do raio: 5.75
O volume da esfera é: 796.33
```

Exercício 3

Implemente um programa que tem como dados de entrada o comprimento, a largura e a altura (números inteiros), calcule e imprima o volume de um paralelepípedo (com duas casas decimais), utilizando a seguinte fórmula:

$$Volume = comprimento * largura * altura$$

Exemplos de execução:

Exemplo 1:

```
Digite o valor do comprimento: 50
Digite o valor da largura: 20
Digite o valor da altura: 15
O volume do paralelepípedo é: 15000.00
```

Exemplo 2:

```
Digite o valor do comprimento: 100
Digite o valor da largura: 50
Digite o valor da altura: 25
O volume do paralelepípedo é: 125000.00
```

Exemplo 3:

```
Digite o valor do comprimento: 75
Digite o valor da largura:  50
Digite o valor da altura:  30
O volume do paralelepípedo é: 112500.00
```

Exercício 4

Elabore um programa que receba como entradas um determinado valor em Real (R$) como saldo de um usuário e um valor para a cotação do Dólar (US$) (números reais). No final apresente como saída o valor convertido em Dólar (com duas casas decimais).

Exemplos de execução:

Exemplo 1:

```
Digite o valor em real: 500
Digite o valor de cotação do dólar: 4.15
O valor convertido é: 120.48
```

Exemplo 2:

```
Digite o valor em real: 1000
Digite o valor de cotação do dólar: 4.18
O valor convertido é: 239.23
```

Exemplo 3:

```
Digite o valor em real: 999
Digite o valor de cotação do dólar: 4.11
O valor convertido é: 243.07
```

Exercício 5

Implemente um programa que receba como entrada o valor de um veículo (número real) e calcule e imprima o valor do IPVA (Imposto sobre a Propriedade de Veículos Automotores) considerando a alíquota de 4.0% (com duas casas decimais).

Exemplos de execução:

Exemplo 1:

```
Digite o valor do veículo: 100000
O valor do IPVA é : 4000.00
```

Exemplo 2:

```
Digite o valor do veículo: 55000
O valor do IPVA é : 2200.00
```

Exemplo 3:

```
Digite o valor do veículo: 35000
O valor do IPVA é : 1400.00
```

4

MÓDULO

Estrutura de Decisão

Normalmente, no cotidiano da vida das pessoas é necessário que algumas condições sejam consideradas para que uma determinada tarefa seja executada.

Em uma tarefa, por exemplo, que envolve o consumo de um determinado medicamento, poderão estar atreladas certas condições como o horário, a quantidade de dosagem, o intervalo de consumo entre outros, que deverão ser estritamente seguidos. Neste sentido, para que seja efetuada esta tarefa, as condições estabelecidas deverão ser avaliadas para direcionar sua execução.

Este cenário também ocorre frequentemente no desenvolvimento de programas, onde determinadas instruções na resolução de um problema serão executadas apenas em decorrência de alguma condição.

No desenvolvimento de programas uma **estrutura de decisão** possibilita, por meio da validação de uma **condição**, a execução de um **bloco de comandos**. Caso a condição seja válida o **bloco de comandos** será executado, caso contrário, será ignorado.

O teste de uma condição retornará sempre um valor do tipo lógico (o tipo de dados **bool** em **Python**), que podem assumir os valores: verdade (**True**) ou falso (**False**). Assim, para uma **condição válida**, ou seja, **verdadeira**, o valor de retorno será **True** e para uma **condição falsa** o valor retornado será **False**. A Tabela 4 apresenta exemplos de condições avaliadas e o respectivo valor de retorno.

Condição	Valor
1 >0	**True**
1 != 0	**True**
1 <0	**False**
1 == 0	**False**

Tabela 4: Exemplos de condições do tipo lógico.

Os fluxogramas das Figuras 12 e 13 ilustram o fluxo de execução de uma **Estrutura de Decisão Simples** e de uma **Estrutura de Decisão Composta**, respectivamente. Observe que na Figura 12, o bloco de comandos de 1 a n será executado somente se a condição declarada for logicamente **verdadeira**. Na Figura 13 são definidos dois blocos de comandos, um bloco que será executado quando a condição for **verdadeira** e outro bloco alternativo a ser executado quando a condição for **falsa**.

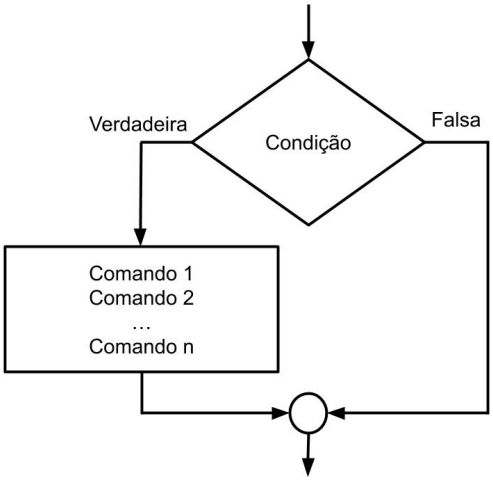

Figura 12: Estrutura de Decisão simples.

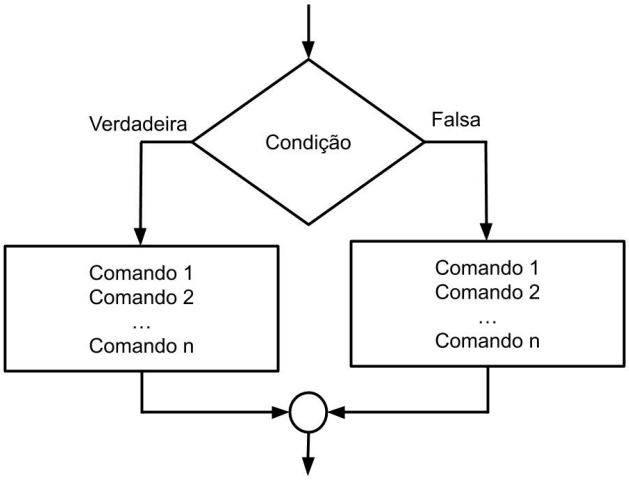

Figura 13: Estrutura de Decisão composta.

4.1 O comando if

Na linguagem de programação **Python** uma estrutura de decisão simples é definida pelo comando **if** e sua sintaxe é dada abaixo:

```
if condição:
    comando 1
    comando 2
    ...
    comando n
continuação
```

Deste modo, o conjunto de comandos de **1** a **n** simbolizam um sub-bloco de instruções, que em **Python** é feito com a **indentação** do código, que normalmente adotam-se espaçamentos ou tabulações. Estas instruções ficam definidas dentro do escopo do comando **if** e serão executadas se o teste da *condição* for **verdadeiro**. Caso a condição seja **falsa**, então o sub-bloco de instruções é ignorado e o programa seguirá o seu fluxo normal de execução passando para a próxima instrução declarada, no caso a partir do comando *continuação*.

Exemplo 1:

Escreva um programa que leia dois valores inteiros distintos e posteriormente imprima, como saída, o maior valor informado.

Solução:

O fluxograma da Figura 14 apresenta a sequência de tarefas a serem seguidas para resolver o exemplo 1. Nem sempre é fácil realizar a implementação direta do programa na linguagem de programação, estabelecer primeiro uma ideia do algoritmo passa a ser uma boa alternativa para facilitar a construção final do programa.

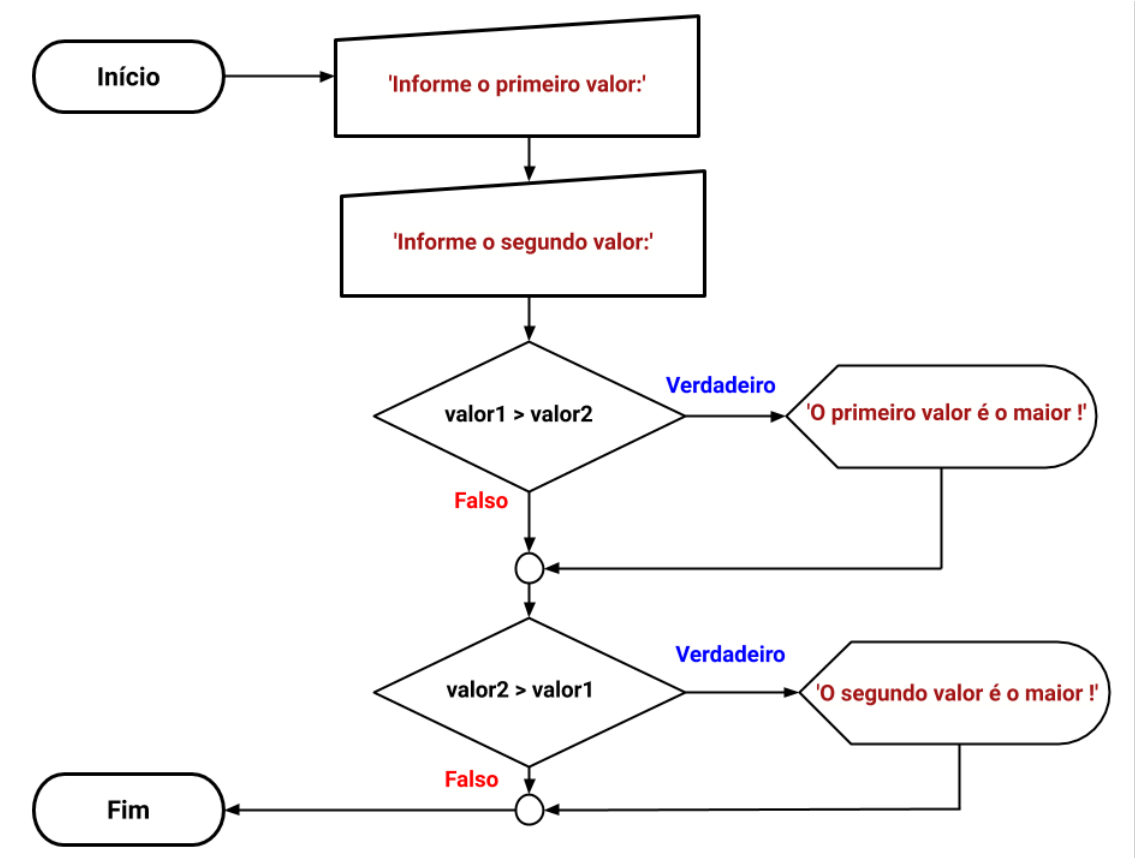

Figura 14: Fluxograma de exemplo para solução do exemplo 1.

O algoritmo também pode ser representado em um nível mais alto, de forma textual:

1. Faça a entrada de dois valores
2. Se o primeiro for maior que o segundo
   (a) Imprima que o primeiro é maior
3. Se o segundo for maior que o primeiro
   (a) Imprima que o segundo é maior

Agora descreveremos como chegar ao programa seguindo uma implementação incremental. Inicialmente definiremos as operações de entradas de dados. No enunciado do exercício é solicitada a leitura ou entrada de dois valores inteiros e distintos. Assim duas operações de entrada deverão ser declaradas e posteriormente os valores armazenados em variáveis. Consideramos que o usuário digitará valores distintos.

As variáveis, como apresenta o bloco de códigos abaixo, são denominadas **valor1** e **valor2** e seus valores são inicializados por meio das instrução de entrada de dados **input** e em seguida convertidos por meio do comando **int**. O programa parcialmente produzido fica assim:

```
1 # Entrada
2 valor1 = int(input('Informe o primeiro valor: '))
3 valor2 = int(input('Informe o segundo valor: '))
```

Em seguida, após a definição das varáveis de entrada e seus valores, a verificação do maior valor é feita por meio da declaração de uma condição simples. Assim, a primeira estrutura de decisão deverá verificar se a condição definida (**valor1 > valor2**) é **verdadeira**. Caso seja **verdade**, uma instrução de saída exibirá uma mensagem como retorno.

O bloco de códigos seguinte apresenta a declaração, através do comando **if**, da condição a ser verificada (**valor1 > valor2**) e da saída definida pela função **print** se a condição for **verdadeira**. Observe que o comando `print('O primeiro valor é o maior')` está afastado (**indentado**) do início da linha, ou seja, em relação à coluna do comando **if**. Isto é obrigatório no **Python** e identifica o bloco a ser executado caso a condição seja **verdadeira** e o seu final.

```
1 # Entrada
2 valor1 = int(input('Informe o primeiro valor: '))
3 valor2 = int(input('Informe o segundo valor: '))
4 # Processamento e Saída
5 if valor1 > valor2:
6     print('O primeiro valor é o maior !')
```

Para finalizar o exercício, como o primeiro comando **if** faz apenas a verificação de uma condição, ou seja, quando o **valor1** for maior que o **valor2**. Uma segunda estrutura condicional **if**, como declarado no bloco de códigos seguinte, é inserido para fazer a verificação inversa, ou seja, quando **valor2 > valor1**.

```
1 # Entrada
2 valor1 = int(input('Informe o primeiro valor: '))
3 valor2 = int(input('Informe o segundo valor: '))
4 # Processamento e Saída
5 if valor1 > valor2:
6     print('O primeiro valor é o maior !')
7 if valor2 > valor1:
8     print('O segundo valor é o maior !')
```

Observe que este programa realiza exatamente a mesma sequência de tarefas descrita no fluxograma e no algoritmo apresentados anteriormente, porém, respeitando todas as regras e utilizando todos os recursos da linguagem necessários para executar o que é preciso e na ordem correta de execução.

4.2 Operadores Relacionais

Normalmente em uma estrutura de decisão a definição de uma condição envolve uma relação lógica entre dois elementos. Esta relação lógica é determinada por meio de **operadores relacionais**. A Tabela 5 apresenta uma lista dos operadores relacionais suportadas em **Python**:

Operador	Operação
==	Igualdade
!=	Desigualdade
>	Maior que
>=	Maior ou igual
<	Menor que
<=	Menor ou igual

Tabela 5: Operadores relacionais.

Assim, conforme descrito anteriormente, o teste de uma condição tem como resultado um valor lógico. Ou seja, se a relação for **verdadeira** o valor de retorno será **True**, caso contrário a relação sendo **falsa** o retorno será o valor **False**. Os valores **True** e **False** são utilizados pelo **Python** para representar os valores **verdadeiro** e **falso**, respectivamente.

4.3 A cláusula else

Uma forma de definir um fluxo alternativo de execução ao bloco de instruções definidas pelo comando **if** é através do emprego da cláusula **else**. Assim, caso a condição definida no comando **if** for falsa o fluxo da execução do programa será redirecionado para o bloco de instruções alternativas declaradas na cláusula **else**.

```
if <condição> :
    comando 1
    comando 2
    ...
    comando n
else:
    comando n+1
    comando n+2
    ...
    comando n+k
```

O *exemplo 1* resolvido anteriormente poderia ser implementado usando a cláusula **else**:

```
1 # Entrada
2 valor1 = int(input('Informe o primeiro valor: '))
3 valor2 = int(input('Informe o segundo valor: '))
4 # Processamento e Saída
5 if valor1 > valor2:
6     print('O primeiro valor é o maior !')
7 else: # implica valor2 > valor1, pois são condições mutuamente
    excludentes
8     print('O segundo valor é o maior !')
```

Observe que agora temos um único comando **if**, usando a cláusula **else**. Porém, o comportamento do programa será exatamente o mesmo do anterior. Sendo assim, do ponto de vista lógico, os dois estão corretos.

Exemplo 2:

Elabore um programa que leia a nota de um aluno em uma determinada disciplina (número real) e exiba uma mensagem indicando sua aprovação, se a nota for maior ou igual a 70. Caso contrário, o programa deverá exibir uma mensagem indicando a reprovação do aluno.

Solução:

Na primeira etapa definiremos as entradas solicitadas no exercício. No enunciado pede-se apenas a leitura da nota do aluno. Assim, declararemos a variável **nota** que será informada pelo usuário. Em seguida, dependendo do valor da nota será necessário tomar uma decisão para imprimir se o aluno foi aprovado ou reprovado. O fluxograma da Figura 15 apresenta a sequência das operações a serem definidas via código para a construção da solução do exemplo 2.

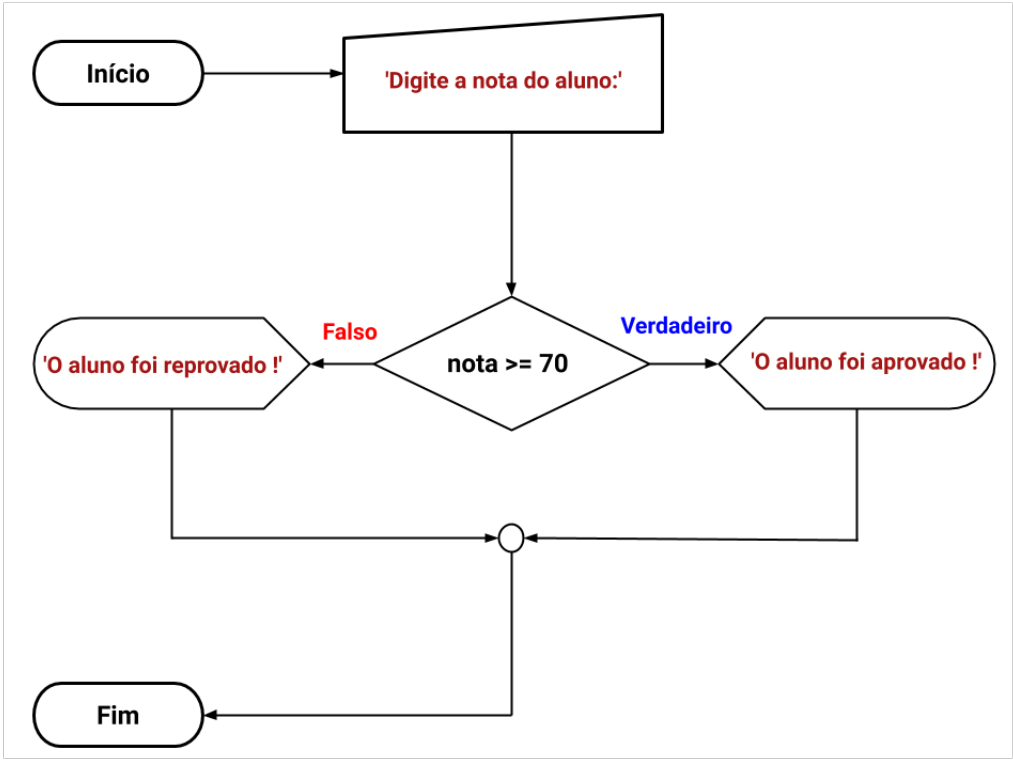

Figura 15: Fluxograma de exemplo para solução do exemplo 2.

O programa completo fica assim:

```
1 # Entrada
2 nota = float(input('Digite a nota do aluno: '))
3 # Processamento e Saída
4 if nota >= 70:
5     print('O aluno foi aprovado !')
6 else:
7     print('O aluno foi reprovado !')
```

Exemplo 3:

Elabore um programa que calcule as raízes de uma equação do 2^o grau qualquer através da **fórmula de Bhaskara**:

- O nome da fórmula foi dado em homenagem ao matemático Bhaskara Akaria;
- Tem como finalidade encontrar as duas raízes de uma equação de 2ž grau;
- Formato: $ax^2 + bx + c = 0$, sendo $a \neq 0$;
- Se $a = 0$, informar que não é uma equação do 2^o grau;
- Caso contrário, calcular Δ e as raízes da equação;
- $\Delta = b^2 - 4ac$;
- $x1 = \frac{-b+\sqrt{\Delta}}{2a}$; e $x2 = \frac{-b-\sqrt{\Delta}}{2a}$.

Solução:

Inicialmente, definiremos as entradas a serem fornecidas e observamos a condição $a == 0$, para que uma equação seja considerada do segundo grau. Neste sentindo, definiremos uma estrutura de decisão para efetuar o teste desta condição como declarado no bloco de códigos abaixo.

```
print('Cálculo das raízes de uma equação do 2o grau')
a = float(input('Digite o coeficiente A: '))
b = float(input('Digite o coeficiente B: '))
c = float(input('Digite o coeficiente C: '))
if a == 0:
    print('Erro: Coeficiente a == 0 !')
```

Na próxima etapa, com as entradas informadas e a condição imposta, um fluxo alternativo (**else**) pode ser definido para prosseguir com a declaração das demais operações necessárias para o cálculo solicitado. No bloco de códigos abaixo, definimos uma instrução para o cálculo do valor de **Delta**. Observe que para o cálculo da raiz de delta utilizamos a função **sqrt**, definida na biblioteca **math** importada no início do programa.

```
import math
print('Cálculo das raízes de uma equação do 2o grau')
a = float(input('Digite o coeficiente A: '))
b = float(input('Digite o coeficiente B: '))
c = float(input('Digite o coeficiente C: '))
if a == 0:
    print('Erro: Coeficiente a == 0 !')
else:   # a é diferente de zero
    delta = b*b - 4*a*c
    print(f"Delta = {delta}")
    x1 = (-b + math.sqrt(delta)) / (2*a)
    x2 = (-b - math.sqrt(delta)) / (2*a)
    print(f'x1 = {x1:.2f}')
    print(f'x2 = {x2:.2f}')
```

Todavia, com a instrução para o cálculo do valor de delta definido, uma nova condição deve ser imposta para o cálculo das raízes da equação, $\Delta > 0$. Assim, após a definição desta condição, como apresenta o bloco de códigos seguinte, as instruções para o cálculo das raízes da equação podem ser declaradas e os resultados exibidos.

O fluxograma da figura 16 apresenta a sequência das operações que representam a solução para o exemplo 3, apresentado no código completo a seguir.

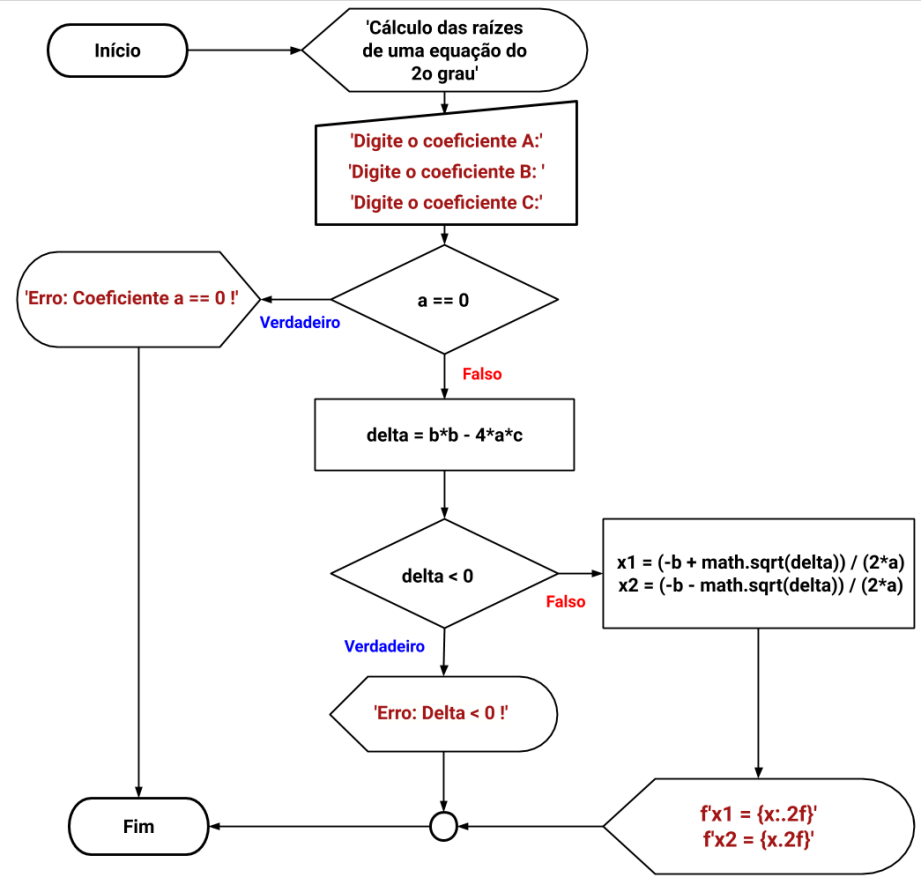

Figura 16: Fluxograma de exemplo para solução do exemplo 3.

```
1  import math
2  print('Cálculo das raízes de uma equação do 2o grau')
3  a = float(input('Digite o coeficiente A: '))
4  b = float(input('Digite o coeficiente B: '))
5  c = float(input('Digite o coeficiente C: '))
6  if a == 0:
7      print('Erro: Coeficiente a == 0 !')
8  else:    # a é diferente de zero
9      delta = b*b - 4*a*c
10     if delta < 0:
11         print('Erro: Delta < 0 !')
12     else: # delta é maior ou igual a zero
13         x1 = (-b + math.sqrt(delta)) / (2*a)
14         x2 = (-b - math.sqrt(delta)) / (2*a)
15         print(f'x1 = {x1:.2f}')
16         print(f'x2 = {x2:.2f}')
```

4.4 Operadores Lógicos

Em uma Estrutura de Decisão os **operadores lógicos** podem ser utilizados para combinar duas ou mais condições. Em **Python** são definidos basicamente três operadores lógicos como apresenta a Tabela 6.

Operador	Operação	Descrição
not	não	Nega (inverte) um valor lógico, ou seja, **not True** resulta em **False**, e **not False** resulta em **True**.
and	e	Retorna **True** caso **TODAS** as condições ligadas sejam **True**.
or	ou	Retorna **True** caso **PELO MENOS UMA** das condições ligadas seja **True**.

Tabela 6: Operadores lógicos.

Exemplo 4:

Vamos retornar ao *exemplo 2*, que avaliava a aprovação de um aluno descrito como: *Elabore um programa que leia a nota de um aluno em uma determinada disciplina (número real) e exiba uma mensagem indicando sua aprovação, se a nota for maior ou igual a 70. Caso contrário, o programa deverá exibir uma mensagem indicando a reprovação.*

Agora vamos inserir um novo critério, o de **frequência**. Para o aluno ser aprovado ele precisa ter também frequência maior ou igual a 75%.

Solução:

O programa poderia ser reformulado para avaliar o resultado do aluno, da seguinte forma:

```
1 # Entrada
2 nota = float(input('Digite a nota do aluno: '))
3 frequencia = float(input('Digite a frequência do aluno: '))
4 # Processamento e Saída
5 if nota >= 70 and frequencia >= 75:
6     print('O aluno foi aprovado !')
7 else:
8     print('O aluno foi reprovado !')
```

Outra forma **correta** para fazer isso seria:

```
1 # Entrada
2 nota = float(input('Digite a nota do aluno: '))
3 frequencia = float(input('Digite a frequência do aluno: '))
4 # Processamento e Saída
5 if nota < 70 or frequencia < 75:
6     print('O aluno foi reprovado !')
7 else:
8     print('O aluno foi aprovado !')
```

A seguir, mais uma solução **correta**:

```
1 # Entrada
2 nota = float(input('Digite a nota do aluno: '))
3 frequencia = float(input('Digite a frequência do aluno: '))
4 # Processamento e Saída
5 if not (nota < 70 or frequencia < 75):
6     print('O aluno foi aprovado !')
7 else:
8     print('O aluno foi reprovado !')
```

Todas as soluções anteriores são **corretas**, pois as expressões lógicas avaliam corretamente a condição imposta para **aprovação** ou **reprovação**, imprimindo a mensagem correspondente ao **resultado correto**.

A solução a seguir, por outro lado, está **incorreta**:

```
1 # Entrada
2 nota = float(input('Digite a nota do aluno: '))
3 frequencia = float(input('Digite a frequência do aluno: '))
4 # Processamento e Saída
5 if not (nota < 70 and frequencia < 75):
6     print('O aluno foi aprovado !')
7 else:
8     print('O aluno foi reprovado !')
```

Isto acontece porque existem situações em que a resposta não estará correta. Um exemplo disso é quando o aluno for reprovado apenas pela nota, ou pela frequência, pois o resultado impresso será que o aluno foi aprovado. Outro exemplo é quando o aluno for aprovado por ambos os critérios, situação em que ele será considerado reprovado. Teste os programas anteriores para estas variadas situações e observe estas considerações.

Um recurso interessante quando se fala em operadores lógicos é a **tabela verdade**, ela mostra todos os resultados possíveis para estas operações, facilitando o seu entendimento. Veja a Tabela 7.

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	False	False	False	True
False	True	False	True	True

Tabela 7: Tabela verdade para operadores lógicos.

4.5 Estruturas de decisão aninhadas

Uma outra possibilidade de construção por meio da estrutura de decisão **if**, quando é necessária a verificação de múltiplos testes condicionais, é através do aninhamento destas condições, ou seja, a definição vários testes de forma sequencial.

Em **Python** a cláusula **elif** possibilita a declaração de uma condição e um sub-bloco de instruções alternativos ao definido na instrução **if**. Assim, várias condições e sub-blocos de instruções podem ser inseridos e na medida em que estas condições são verificadas o término culminará com a verificação de

todas as condições ou com a validação de uma condição e a execução do bloco de instruções associado ou até o bloco definido com o comando **else**, se este for declarado. Abaixo descrevemos a sintaxe de uma estrutura aninhada definida pelo comando **elif**.

```
if <condição 1> :
    comando a.1
    comando a.2
    ...
    comando a.n
elif <condição 2> :
    comando b.1
    comando b.2
    ...
    comando b.o
    ...
elif <condição k> :
    comando c.1
    comando c.2
    ...
    comando c.p
else:
    comando d.1
    comando d.2
    ...
    comando d.q
```

Exemplo 5:

Dados três valores inteiros distintos fornecidos pelo usuário escreva um programa para imprimi-los em ordem crescente.

Solução:

O enunciado é curto, mas a solução não é tão trivial. Por isso, vamos elaborar nossa solução aos poucos, de forma **iterativa**, e prestando atenção aos detalhes de cada parte da solução.

Primeiramente, é importante não nos esquecermos de um detalhe importante descrito no enunciado: "(...) valores inteiros **distintos** (...)". Ou seja, os números precisam ser diferentes entre si. Então, antes de tentar fazer uma ordenação, precisamos tomar uma **decisão**, se os números são **válidos ou não**. Caso sejam **válidos** (**todos diferentes entre si**), devemos avaliar a ordem dos números, mas, caso sejam **inválidos**, demos informar esta situação ao usuário. Então, em uma primeira versão de nosso algoritmo, colocaremos a validação da entrada, ficando assim:

1. Faça a entrada de três valores numéricos inteiros
2. Se os valores forem **Inválidos**:
    (a) Informe ao usuário que os valores são **inválidos**
3. Caso contrário:
    (a) Avalie a ordem dos números e gere a saída adequada

Para um número ser considerado inválido, devemos recorrer aos operadores relacionais e lógicos. Como eles precisam ser **todos diferentes** para serem **válidos**, a lógica para que sejam **inválidos** é a inversa, que **pelo menos um par seja igual**. Para isto, podemos avaliar: $N1 == N2$ **or** $N1 == N3$ **or** $N2 == N3$. Nosso algoritmo já tem uma estrutura inicial definida que nos permite iniciar o programa, ficando desta maneira:

```
1 N1 = int(input('Informe o primeiro valor: '))
2 N2 = int(input('Informe o segundo valor: ' ))
3 N3 = int(input('Informe o terceiro valor: '))
4 if N1 == N2 or N1 == N3 or N2 == N3:
5     # Informe ao usuário que os valores são inválidos
6 else:
7     # Avalie a ordem dos números e gere a saída adequada
```

Primeiro passo dado, já conseguimos estruturar o programa de forma a realizar a entrada e avaliar se os dados fornecidos são **válidos** ou **inválidos**, determinando um fluxo de execução que permite realizar as tarefas necessárias após a avaliação feita. Vamos nos preocupar agora com estas tarefas específicas.

A primeira, "*informar ao usuário que os valores são inválidos*" é uma tarefa simples, resolvida apenas com a função **print**. Então, avançamos mais um pouco:

```
1 N1 = int(input('Informe o primeiro valor: '))
2 N2 = int(input('Informe o segundo valor: ' ))
3 N3 = int(input('Informe o terceiro valor: '))
4 if N1 == N2 or N1 == N3 or N2 == N3:
5     print('Erro: Os valores informados devem ser distintos.')
6 else:
7     # Avalie a ordem dos números e gere a saída adequada
```

A segunda tarefa, "*avaliar a ordem dos números e gerar a saída adequada*", é um pouco mais complexa, e é através das **estruturas de decisão aninhadas** que nós a resolveremos. Se tivermos três números, por exemplo **N1**, **N2** e **N3**, poderemos listar inúmeras possibilidades de ordenação crescente para eles. Caso já soubéssemos quais são os números, a resposta seria muito fácil. Porém, eles serão fornecidos pelo usuário, então precisamos de uma solução algorítmica que avalie todas as possibilidades.

O importante aqui é começar a definir uma estratégia para isso. Como temos apenas três números, é fácil listar as possibilidades, por exemplo, a ordem que primeiro nos vem à cabeça poderia ser **N1**, **N2** e **N3**, para isso, teríamos que aplicar duas avaliações relacionais, e ambas resultarem em verdadeiro: $N1 < N2$ **e** $N2 < N3$. Observe que não é necessário comparar **N1** com **N3**, pois se as duas expressões forem verdadeiras, está garantido que $N1 < N3$. Portanto, caso $N1 < N2$ **and** $N2 < N3$, a ordem **N1**, **N2** e **N3** está **correta**, e podemos gerar o resultado. Caso isso não seja verdade, precisamos seguir avaliando as outras possibilidades. Sendo assim, uma outra possibilidade seria a ordem **N1**, **N3** e **N2**, que, de forma similar, poderia ser verificada por $N1 < N3$ **and** $N3 < N2$. E assim, por diante.

Com isto em mente, podemos elaborar um fluxograma que resolve o problema, conforme apresentado na Figura 17. E, em seguida, implementar o programa completo.

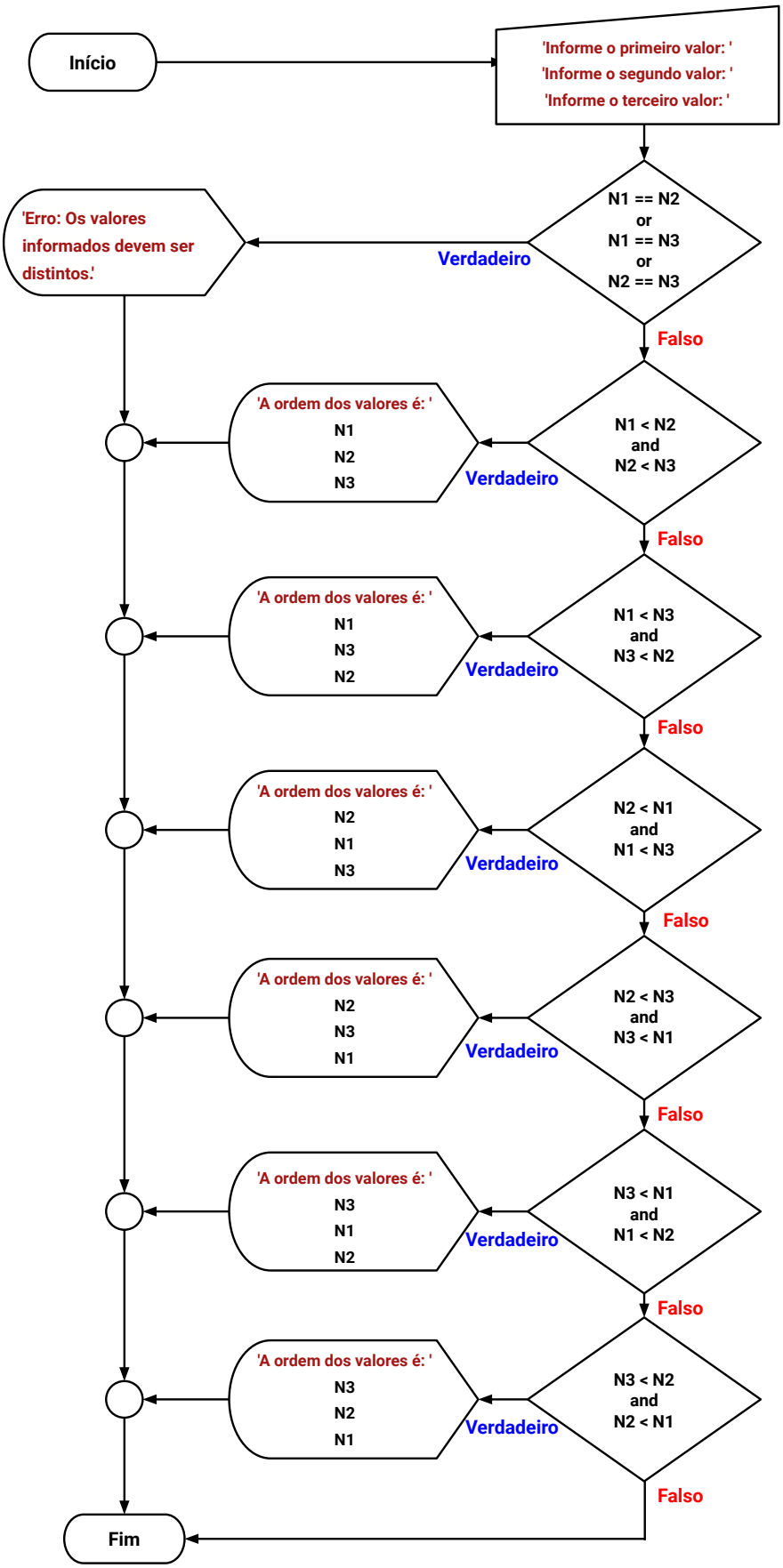

Figura 17: Fluxograma de exemplo para solução do exemplo 5.

```
N1 = int(input('Informe o primeiro valor: '))
N2 = int(input('Informe o segundo valor: ' ))
N3 = int(input('Informe o terceiro valor: '))
if N1 == N2 or N1 == N3 or N2 == N3:
    print('Erro: Os valores informados devem ser distintos.')
else:
    if N1 < N2 and N2 < N3:
        print(f'A ordem dos valores é: {N1}, {N2}, {N3}')
    elif N1 < N3 and N3 < N2:
        print(f'A ordem dos valores é: {N1}, {N3}, {N2}')
    elif N2 < N1 and N1 < N3:
        print(f'A ordem dos valores é: {N2}, {N1}, {N3}')
    elif N2 < N3 and N3 < N1:
        print(f'A ordem dos valores é: {N2}, {N3}, {N1}')
    elif N3 < N1 and N1 < N2:
        print(f'A ordem dos valores é: {N3}, {N1}, {N2}')
    elif N3 < N2 and N2 < N1:
        print(f'A ordem dos valores é: {N3}, {N2}, {N1}')
```

Observe que no programa anterior foi necessário utilizar dois comandos **if**. Uma decisão para a validação e outra para as diferentes possibilidades de ordenação dos números. Existem outras formas de estruturar o programa, com apenas uma decisão ou com várias outras decisões. Porém, recomendamos utilizar a estrutura do programa apresentado, pois ele separa bem as duas tarefas principais do problema proposto: (1) validar os números; e (2) determinar a ordem dos números.

4.6 Exercícios Propostos

Exercício 1

Elabore um programa que leia as quatro notas de um aluno, números reais entre 0 a 100, e calcule a média das notas. E no final, como saída, imprima a média (com uma casa decimal) e uma mensagem de aprovação se a média for maior ou igual a 70 ou reprovação caso a média for inferior.

Exemplos de execução:

Exemplo 1:

```
Digite a primeira nota: 90
Digite a segunda nota: 85
Digite a terceira nota: 95
Digite a quarta nota: 100
A média é 92.5. O aluno foi aprovado.
```

Exemplo 2:

```
Digite a primeira nota: 50
Digite a segunda nota: 65
Digite a terceira nota: 70
Digite a quarta nota: 45
A média é 57.5. O aluno foi reprovado.
```

Exemplo 3:

```
Digite a primeira nota: 100
Digite a segunda nota: 95
Digite a terceira nota: 101
Digite a quarta nota: 90
Erro: os valores das notas devem estar entre 0 a 100.
```

Exercício 2

Elabore um programa que leia três valores inteiros x, y e z e verifique se estes valores podem constituir os comprimentos de um triângulo e se caso positivo informe se este é um triângulo equilátero, isósceles ou escaleno.

Observações:

- O triângulo equilátero possui os três lados com medidas iguais;
- O triângulo isósceles possui dois lados com medidas iguais;
- O triângulo escaleno possui os três lados com medidas diferentes.
- Condição de desigualdade triangular: $(x < y + z),\ (y < x + z)\ e\ (z < x + y)$

Exemplos de execução:

Exemplo 1:

```
Digite o valor de x: 10
Digite o valor de y: 10
Digite o valor de z: 10
Triângulo Equilátero
```

Exemplo 2:

```
Digite o valor de x: 50
Digite o valor de y: 25
Digite o valor de z: 50
Triângulo Isósceles
```

Exemplo 3:

```
Digite o valor de x: 100
Digite o valor de y: 200
Digite o valor de z: 150
Triângulo Escaleno
```

Exemplo 4:

```
Digite o valor de x: 200
Digite o valor de y: 0
Digite o valor de z: 400
Erro: os valores informados não constituem um triângulo.
```

Exercício 3

Implemente um programa que leia o valor de um veículo (número real) e sua respectiva classe (número inteiro), como descreve a Tabela abaixo, para o cálculo do IPVA (Imposto sobre a Propriedade de Veículos Automotores). O cálculo deve considerar a alíquota da respectiva classe do veículo. No final imprima o resultado do cálculo como saída.

Classe	Descrição	Alíquota
1	Automóveis, veículos de uso misto e utilitários, caminhonetes cabine estendida e dupla.	4,0%
2	Caminhonetes de carga (pick-ups) e furgão.	3,0%
3	Automóveis, veículos de uso misto e utilitários com autorização para transporte público (ex: táxi, escolar) comprovada mediante registro no órgão de trânsito na categoria aluguel.	2,0%
4	Motocicletas e similares.	2,0%
5	Veículos de locadoras (pessoa jurídica).	1,0%
6	Ônibus, micro ônibus, caminhão, caminhão trator.	1,0%

Exemplos de execução:

Exemplo 1:

```
Digite o valor do veículo: 120000
Digite a classe do veículo: 1
O valor do IPVA é : 4800.00
```

Exemplo 2:

```
Digite o valor do veículo: 250000
Digite a classe do veículo: 6
O valor do IPVA é : 2500.00
```

Exemplo 3:

```
Digite o valor do veículo: 8000
Digite a classe do veículo: 4
O valor do IPVA é : 160.00
```

Exercício 4

Elabore um programa que leia dois valores reais x e y e que pergunte qual operação deve ser realizada. As operações que poderão ser calculadas são a soma $(+)$, subtração$(-)$, multiplicação$(*)$, divisão$(/)$ e potenciação $(\wedge)$. **Observação**: para efetuar a divisão, y deverá ser diferente de zero.

Exemplos de execução:

Exemplo 1:

```
Digite o primeiro valor: 25
Digite o segundo valor: 25
Informe a operação: +
25.00 + 25.00 = 50.00
```

Exemplo 2:

```
Digite o primeiro valor: 5
Digite o segundo valor: 5
Informe a operação: ^
5.00 ^ 5.00 = 3125.00
```

Exemplo 3:

```
Digite o primeiro valor: 81
Digite o segundo valor: 9
Informe a operação: /
81.00 / 9.00 = 9.00
```

Exemplo 4:

```
Digite o primeiro valor: 1000
Digite o segundo valor: 0
Informe a operação: /
Erro: Divisão por zero !
```

Exercício 5

Um vendedor recebe um determinado valor de comissão sobre o total de suas vendas efetuadas. Em vendas superiores a 5000,00 o valor a receber corresponde a 25%, os valores na faixa de 5000,00 a 2500,00 corresponde a 20% e para valores inferiores a 2500,00 o valor corresponde a 15%. Implemente um programa que leia o valor total das vendas (número real) e calcule a quantia que o vendedor irá receber.

Exemplos de execução:

Exemplo 1:

```
Digite o valor total das vendas: 7000
O valor de comissão é: 1750.00
```

Exemplo 2:

```
Digite o valor total das vendas: 3000
O valor de comissão é: 600.00
```

Exemplo 3:

```
Digite o valor total das vendas: 500
O valor de comissão é: 75.00
```

MÓDULO 5

Estrutura de Repetição

Em muitos cenários no desenvolvimento de programas para facilitar a resolução de determinados problemas será necessário definir um subconjunto de comandos que deverão ser executados de forma consecutiva em um número finito de vezes. Assim, as principais linguagens de programação proporcionam as estruturas de **repetição** (ou **iteração** ou **laço**) que definem blocos de instruções a serem executadas de forma repetida.

Existem basicamente duas formas de definir uma estrutura de repetição:

- **Controlada Logicamente**: define-se explicitamente uma condição através de uma expressão lógica, enquanto a condição se mantiver verdadeira durante a execução, as instruções definidas no bloco da estrutura de repetição serão executadas;
- **Controlada por Contador**: define-se implicitamente uma condição por meio de um contador, existindo uma sequência de contagem predefinida: **início**, **passo** e **fim**. A cada iteração o valor do contador é alterado pelo valor do **passo**. Assim, o bloco de instruções declaradas na estrutura de repetição será executado enquanto o contador não atingir ou ultrapassar o valor **final** da sequência.

No fluxograma da Figura 18 é apresentado um esquema do fluxo de execução para um estrutura de repetição **controlada logicamente**, onde o conjunto de comandos (*comando* 1, *comando* 2, ..., *comando* n) serão executados enquanto uma condição definida se mantiver verdadeira.

O fluxograma da Figura 19 apresenta um esquema de execução para um estrutura de repetição **controlada por contador**, onde o conjunto de comandos (*comando* 1, *comando* 2, ..., *comando* n) serão executados enquanto o contator não atingir o valor final ou superior. A definição dos valores de início, passo e fim determinará o número de iterações a serem executadas.

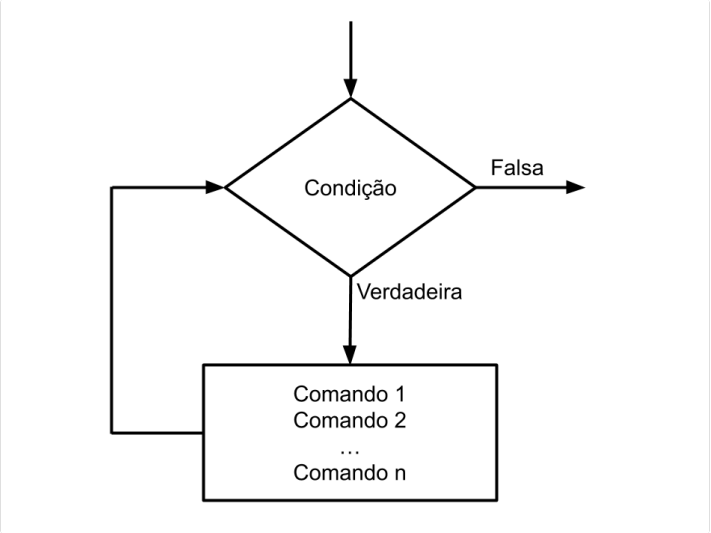

Figura 18: Estrutura de Repetição controlada logicamente.

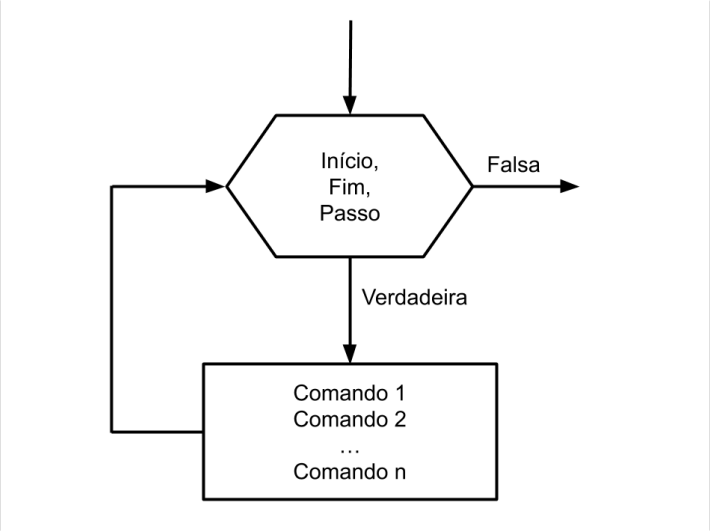

Figura 19: Estrutura de Repetição controlada por contador.

5.1 Repetição controlada logicamente: comando while

O comando **while** em **Python** define uma estrutura de repetição utilizada para a execução de um subconjunto de instruções declarados em seu escopo de forma iterativa enquanto uma condição lógica pré-determinada se mantiver verdadeira. O comando **while** define uma estrutura de repetição do tipo **controlada logicamente**.

Como mostrado abaixo, a sintaxe do comando **while** envolve a definição de uma condição que enquanto verdadeira determinará a repetição e execução do bloco de comandos (`comando 1, comando 2, ..., comando n`). Em **Python**, como visto em módulos anteriores, a definição de um bloco de comandos é feito através de indentação, onde normalmente se utilizam espaçamentos ou tabulações.

```
while <condição>:
    comando 1
    comando 2
    ...
    comando n
```

Exemplo 1:

Implemente um programa que leia do teclado um valor numérico N do tipo inteiro positivo e no final imprima todos os valores numéricos que compreendem o intervalo de 0 a N.

Solução:

Para desenvolver uma solução neste exemplo através de uma estrutura de repetição, primeiro consideraremos a definição das seguintes etapas a serem realizadas (um algoritmo em alto nível):

1. Definir a entrada do valor de **N**;
2. Declarar a variável **item**, que será responsável por a cada iteração assumir um valor da sequência que será gerada através da estrutura de repetição e inicializá-la com o primeiro valor da sequência, ou seja, zero;
3. Definir a estrutura de repetição através do comando **while** e sua condição de repetição, ou seja, enquanto o valor da variável for `item <= N`;
4. No escopo do comando **while** definir as seguintes instruções:
   (a) Imprimir o valor da variável **item**;
   (b) Atualizar o valor da variável, com a fórmula: `item = item + 1`, ou seja, a cada iteração a variável **item** terá seu valor incrementado com um, gerando assim uma sequência de valores;
   (c) Verificar a condição de parada. Se a condição `item <= N` continuar verdadeira, execute novamente as instruções definidas no bloco do comando **while**, caso contrário, se a condição for falsa, saia da estrutura de repetição.

No fluxograma da Figura 20 é apresentado o encadeamento das operações relacionadas às etapas definidas anteriormente para uma proposta de solução via código para o Exemplo 1 através do uso de uma estrutura de repetição controlada logicamente.

O bloco de códigos abaixo apresenta uma solução desenvolvida. Primeiro, é efetuada a entrada e atribuição para a variável **N**, e na sequência, a declaração da variável **item** que terá seu valor alternado na estrutura de repetição, entre o intervalo de inicialização, ou seja, o valor zero, até o valor de **N**. Depois temos a declaração da estrutura de repetição **while** com a condição de sua execução (`item <= N`).

No bloco interno a estrutura de repetição temos o comando de saída que retornará o valor assumido pela variável **item**. E a instrução responsável por atualizar o valor da variável `item`. Este processo de imprimir e atualizar o valor de **item** se repetirá enquanto a condição declarada na estrutura de repetição for válida. E assim, serão impressos o valores da sequência: $0, 1, 2, ..., N$.

```
1 # Entrada
2 N = int(input('Digite o valor de N: '))
3 # Processamento e Saída
4 item = 0
5 while item <= N :
6     print(item)
7     item = item + 1
```

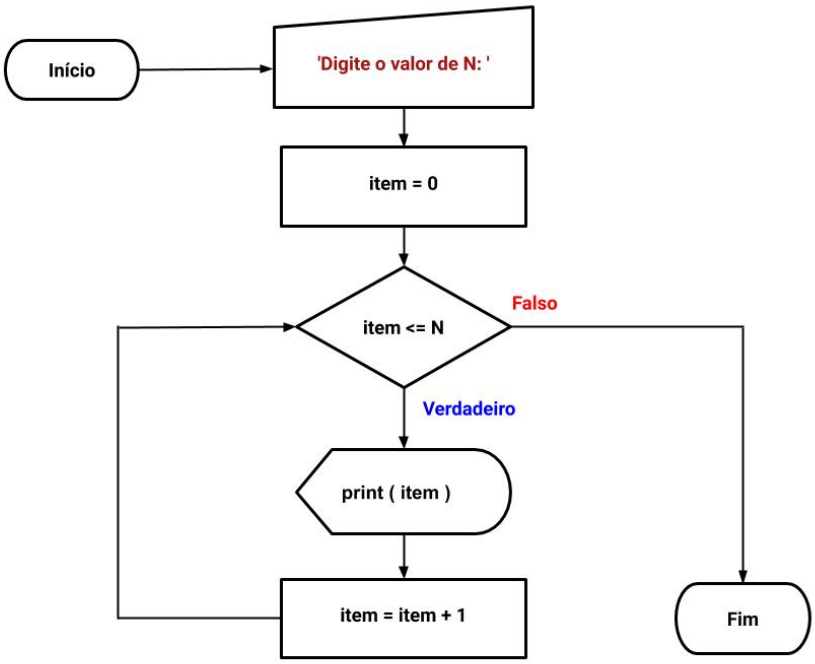

Figura 20: Fluxograma de execução para o Exemplo 1.

Exemplo 2:

Implemente um programa que leia um valor numérico N e calcule o somatório $S = 1 + 2 + 3, ..., N$. No final imprima o resultado do somatório.

Solução:

Neste exemplo, para o desenvolvimento de uma solução, também definiremos um algoritmo que descreve a sequência de etapas a serem seguidas para a elaboração da estrutura de repetição, que será responsável por gerar os valores e efetuar o cálculo descrito. Assim as etapas consistem em:

1. Definir a entrada para o valor **N**;
2. Definir a variável **item**, que será responsável por assumir a cada iteração um valor da sequência $(1, 2, 3, ..., N)$ gerada pela estrutura de repetição. A variável **item** terá seu valor inicial definido com o primeiro valor da sequência 1;
3. Definir uma variável responsável por armazenar o valor resultante da soma dos itens a cada iteração, denominamos como **somatorio** e a inicializamos com o valor $zero$;
4. Declarar a estrutura de repetição e sua condição de repetição. A variável **item**, como visto anteriormente, será responsável por a cada iteração assumir um valor da sequência, assim a repetição será realizada enquanto o valor da variável **item** for menor ou igual ao valor de **N**;
5. No bloco do comando **while** definir as seguintes instruções:
   (a) Atualizar o valor da variável **somatorio** com seu valor incrementado ao valor da variável **item**;
   (b) Atualizar o valor da variável **item** com incremento de 1 (um), gerando o próximo item da sequência (`item = item + 1`);
   (c) Verificar a condição de parada. Se a condição definida for válida as operações declaradas na estrutura de repetição irão ser novamente executadas, senão a repetição será interrompida;

6. Imprimir o resultado do somatório.

O fluxograma da Figura 21 apresenta a definição de uma sequência de operações a serem realizadas em uma proposta de solução para o Exemplo 2, através do emprego de uma estrutura de repetição controlada logicamente.

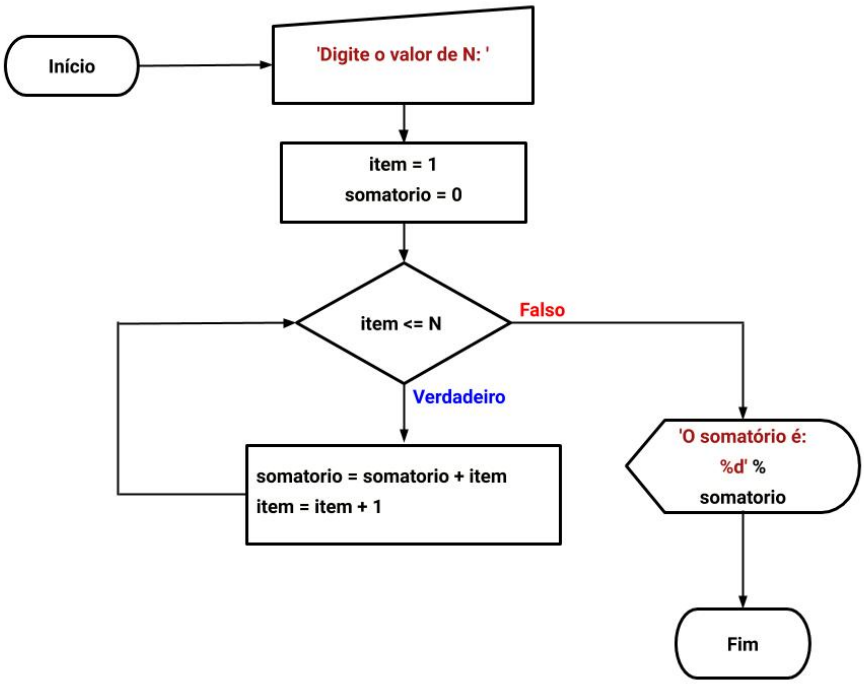

Figura 21: Fluxograma de execução para o Exemplo 2.

O bloco de códigos abaixo apresenta um solução para o exemplo, similar ao Exemplo 1, através do uso de uma estrutura de repetição. Inicialmente, realizamos a leitura para o valor da variável **N**. Posteriormente declaramos e inicializamos a variável **item** que terá seu valor alternado com um valor da sequência a cada repetição executada na estrutura e a variável **somatorio** responsável realizar e armazenar o resultado da soma do valores gerados. Em seguida, definimos a estrutura de repetição com a condição: $item <= N$. Assim, na medida em que os valores da variável **item** vão sendo alternados é imposta a condição de que o valor seja menor ou igual ao valor de **N** e quando: $item > N$, a estrutura de repetição será interrompida. Por fim, é realizada a impressão do resultado obtido com o cálculo efetuado.

```
1  # Entrada
2  N = int(input('Digite o valor de N: '))
3  # Processamento
4  contador = 1
5  somatorio = 0
6  while contador <= N:
7      somatorio = somatorio + contador
8      contador = contador + 1
9  # Saída
10 print(f'O somatório é: {somatorio}')
```

Exemplo 3:

Elabore um programa que leia uma quantidade indeterminada de valores numéricos e no final imprima o maior valor informado. O programa deverá finalizar a leitura quando o valor 0 for digitado.

Solução:

Neste exemplo para a elaboração de uma solução iremos considerar a sequência de etapas definidas abaixo. No fluxograma da Figura 22 é apresentado o fluxo das principais operações para uma proposta de solução do Exemplo 3 através de uma estrutura de repetição controlada logicamente:

1. Declarar a variável que armazenará os valores informados na estrutura de repetição, denominamos como **valor** e definimos a primeira entrada;
2. Declarar uma variável que armazenará o maior valor informado, denominamos como **maior_valor**. E inicializamos **maior_valor** com o valor da variável **valor**;
3. Definir uma estrutura de repetição, com a condição do valor informado ser diferente de $zero$ (`valor != 0`). E no bloco da estrutura de repetição:
   (a) Definir uma nova entrada de um valor para a variável **valor**;
   (b) Definir uma estrutura de decisão responsável por verificar se o novo **valor** é maior que o valor armazenado na variável **maior_valor**. E no bloco da estrutura de decisão se a condição for verdadeira:
      i. Atualizar o valor da variável **maior_valor** para o valor da variável **valor**;
   (c) Retornar ao teste da condição da estrutura de repetição. Se for verdadeira executar novamente as instruções definidas no bloco, se for falsa saia da estrutura de repetição;
4. Imprimir o maior valor informado, armazenado em **maior_valor**.

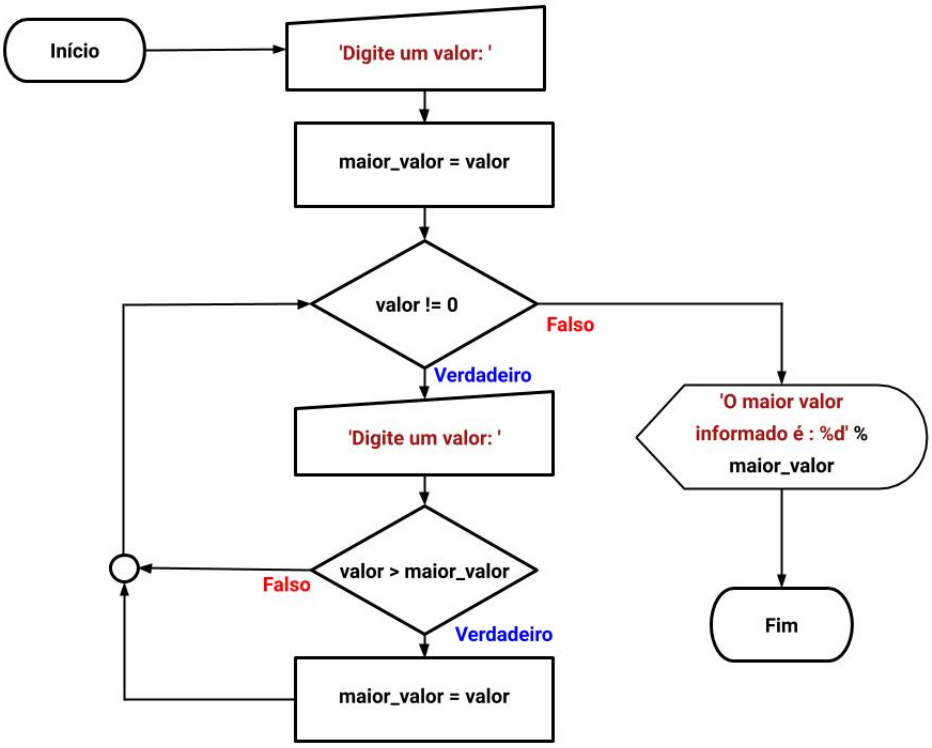

Figura 22: Fluxograma de execução para o Exemplo 3.

No programa a seguir apresentamos uma solução para o exemplo. Primeiro, é efetuada a leitura do primeiro valor informado armazenado na variável **valor**. Depois, é declarada a variável **maior_valor** que será responsável por armazenar o maior valor informado durante as iterações, esta variável é inicializada com o valor da variável **valor**. Em seguida, temos a definição da estrutura de repetição feita com o comando **while** com a condição para repetição sendo `valor != 0`. No sub-bloco de instruções do comando **while** temos a entrada de um novo valor. Em seguida, a declaração de uma estrutura de decisão com a condição `valor > maior_valor`, sendo a condição verdadeira uma instrução responsável por atualizar o valor da variável **maior_valor** pelo novo valor informado, e caso seja falsa a execução retornará na primeira instrução definida na estrutura de repetição solicitando assim a leitura de novo valor. Estas instruções se repetirão enquanto o valor $zero$ não for informando. Quando o valor $zero$ for informado a estrutura de repetição será finalizada e a instrução de impressão será executada imprimindo a frase com o maior valor informado.

```
1 # Entrada e Processamento
2 valor = float(input('Digite um valor: '))
3 maior_valor = valor
4 while valor != 0:
5     valor = float(input('Digite um valor: '))
6     if valor > maior_valor:
7         maior_valor = valor
8 # Saída
9 print(f'O maior valor informado é: {maior_valor}')
```

5.2 Repetição controlada por contador: comando for

O comando **for** pode ser utilizado como uma estrutura de repetição do tipo controlada por contador para a execução de um sub-bloco de instruções de forma iterativa. É comumente utilizado quando o número de iterações a serem realizadas é conhecido.

Desta maneira, a declaração do comando **for** será expressa com a definição de uma sequência com os valores de **início**, **fim** e o **passo** que servirão como base no processo das repetições a serem realizadas. Em **Python**, como veremos na próxima seção, uma sequência numérica que representa o contador pode ser definida na declaração do comando **for** através da função auxiliar **range**. Existem variadas formas de utilização do **for** em linguagem **Python**, mas aqui utilizaremos apenas aquela que melhor se ajusta ao conceito de repetição controlada por contador, foco do nosso conteúdo programático.

A sintaxe do comando **for** em **Python** é descrita abaixo:

```
for <variável> in <sequência>:
    comando 1
    comando 2
    ...
    comando n
```

A sintaxe apresenta os seguintes itens em destaque:

- **for**: comando que define a estrutura de repetição;
- **<variável>**: variável que assume a cada iteração um valor definido pela **<sequência>**;
- **<sequência>**: uma sequência enumerável de valores, em **Python** esta sequência pode ser definida de variadas formas, aqui usaremos sempre a função **range**, de forma a definir os parâmetros

do contador:

  - **início**: valor inicial da sequência;
  - **fim**: valor limite do número de itens da sequência;
  - **passo**: valor de passo ou de incremento sobre os itens da sequência;

- **comando 1, comando 2, ..., comando n**: conjunto de comandos declarados no bloco do comando for a serem executados de forma iterativa.

5.2.1 A função range

A função **range** em **Python** é normalmente utilizada em conjunto com o comando de repetição **for** para representar uma repetição controlada por contador. É responsável por gerar uma sequência de itens numéricos com base nos argumentos definidos. Na sequência, descreveremos as notações comuns de definição dos parâmetros na função **range** que poderão se ajustadas para gerar as sequências numéricas desejadas como apresentam os exemplos abaixo. Assim:

- range($vi,\ n,\ passo$): gera uma sequência de números inteiros, sendo **vi** o primeiro item da sequência (**início**); os itens seguintes são definidos de acordo com o valor de **passo**; e o item $n-1$ como valor limite (**final**);
- range($vi,\ n$): gera uma sequência de números inteiros, sendo **vi** o primeiro item da sequência (**início**); os itens seguintes são definidos de acordo com o valor de **passo** padrão, `passo = 1`; e o último item $n-1$ como valor limite (**final**);
- range(n): gera uma sequência de n números inteiros, sendo por padrão $zero$ o primeiro item da sequência (**início**); os itens seguintes são definidos de acordo com o valor de **passo** padrão, `passo = 1`; e o último item $n-1$ como valor limite (**final**).

```
1 for i in range(0, 10, 1):
2     print(i)
```

```
1 for i in range(1, 10, 1):
2     print(i)
```

```
1 for i in range(0, 10, 2):
2     print(i)
```

```
1 for i in range(0, 10):
2     print(i)
```

```
1 for i in range(1, 10):
2     print(i)
```

```
1 for i in range(10):
2     print(i)
```

Crie programas no **Thonny** com os códigos anteriores e execute para ver o resultado gerado por cada um deles.

Exemplo 4:

Elabore um programa que determine o Fatorial de um valor inteiro positivo fornecido como entrada. E no final imprima o resultado do cálculo.

Observações:

- $0! = 1$;
- $n! = n * (n-1) * (n-2) * (n-3) * ... * 3 * 2 * 1$

Solução:

No fluxograma da Figura 23 estão definidas uma sequência lógica de etapas das operações para uma proposta de solução para o Exemplo 4 através de um estrutura de repetição controlada por contador.

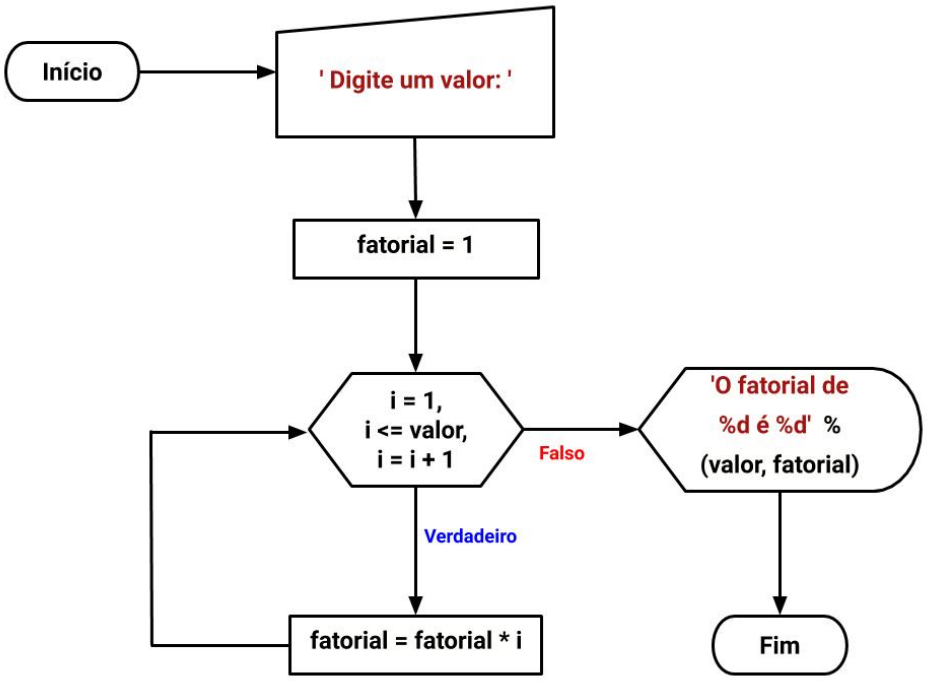

Figura 23: Fluxograma de execução para o Exemplo 4.

No bloco de instruções abaixo uma proposta de solução é definida para o exemplo. Primeiro, é efetuada a entrada armazenada na variável **valor** para o cálculo do fatorial. Depois declaramos uma variável denominada **fatorial** responsável por armazenar o valor do fatorial. Em seguida, a estrutura de repetição com os itens definidos para a sequência de iteração variando entre 1 e $valor$, com o valor de incremento igual a 1. A única instrução do bloco de comandos de repetição é responsável por efetuar a multiplicação do valor contido na variável **fatorial** com o valor da variável **i** que armazena a cada iteração um valor definido na sequência cujos valores variam no intervalo de 1 a $valor$ durante as iterações da estrutura de repetição. O resultado a cada repetição é armazenado na própria variável **fatorial**. Chamamos este tipo de cálculo de **cumulativo**, ou seja, a cada iteração da estrutura de repetição o valor é atualizado acumulativamente. O valor final é obtido com o término da execução da estrutura de repetição, e impresso através da função **print**.

```
1 # Entrada
2 valor = int(input('Digite um valor: '))
3 # Processamento
4 fatorial = 1
5 for i in range(1, valor + 1, 1):
6     fatorial = fatorial * i
7 # Saída
8 print(f'O fatorial de {valor} é {fatorial}')
```

Exemplo 5:

Implemente um programa que leia como entrada um valor N e calcule e imprima como saída o valor da série:

$$S = \sum_{i=1}^{N} \frac{2i-1}{i}$$

Solução:

No fluxograma da Figura 24 estão definidas uma sequência lógica de etapas das operações para uma proposta de solução para o Exemplo 5 através da aplicação de uma estrutura de repetição controlada por contador.

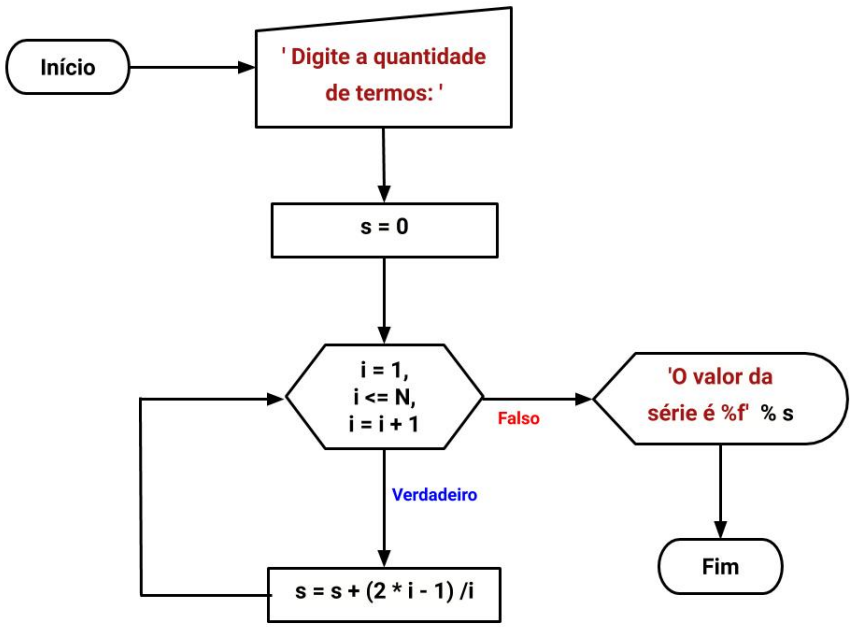

Figura 24: Fluxograma de execução para o Exemplo 5.

No bloco de instruções abaixo uma proposta de solução é definida para o exemplo. Primeiro é definida a instrução de entrada para a variável **N**. Em seguida, a declaração da variável **s** responsável por armazenar o valor do somatório descrito pela fórmula, a variável é previamente inicializada com o valor $zero$. Depois é declarada a estrutura de repetição usando o comando **for** sobre a sequência de valores definida por: $(1,\ N+1,\ 1)$, que retornam a cada iteração os itens do intervalo de 1 a N (inclusive). No bloco de comandos de repetição, temos a instrução responsável por efetuar o cálculo do somatório, onde o valor de **i** é obtido através da variação do itens da sequência ao longo das repetições, atualizando a variável **s** a cada iteração da repetição. No final, o valor do somatório é impresso com a função **print**.

```
# Entrada
N = int(input('Digite a quantidade de termos: '))
# Processamento
s = 0
for i in range(1, N + 1, 1):
    s = s + (2 * i - 1) /i
# Saída
print(f'O valor da série é {s}')
```

Exemplo 6:

Implemente um programa que gere e imprima os N termos da série de Fibonacci. O valor de N deve ser fornecido como entrada.

A série de Fibonacci é uma sequência de números inteiros cujos valores iniciais são 0 e 1, e cada termo subsequente corresponde à soma dos dois anteriores. Assim, um novo termo pode ser determinado por meio da seguinte fórmula:

$$F_n = F_{n-1} + F_{n-2}$$

E os valores iniciais são:

- $F_1 = 0$;
- $F_2 = 1$.

Solução:

O fluxograma da Figura 25 apresenta uma sequência de etapas das operações para a definição de uma proposta de solução para o Exemplo 6 por meio de uma estrutura de repetição controlada por contador.

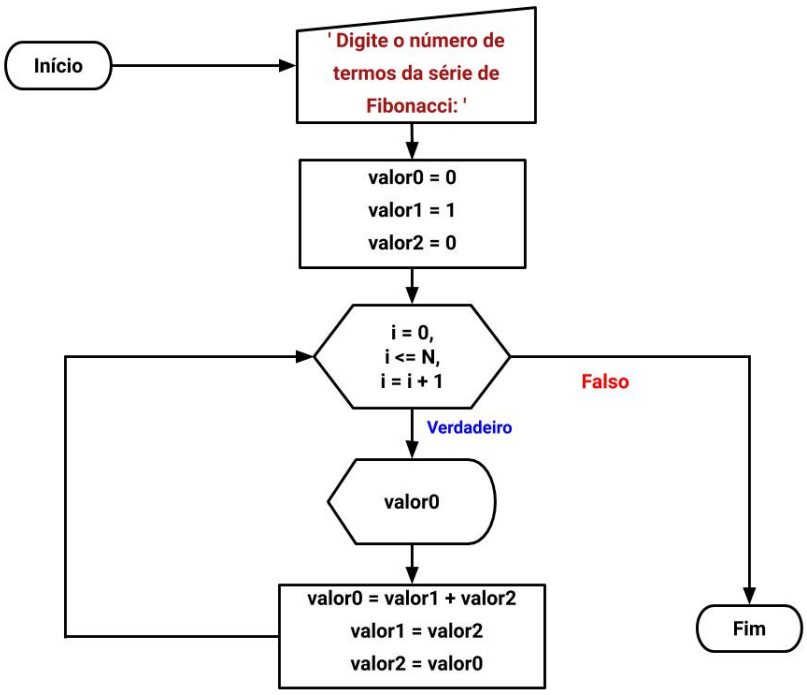

Figura 25: Fluxograma de execução para o Exemplo 6.

No bloco de instruções seguinte encontra-se uma proposta de solução para o Exemplo 6. Primeiro, é efetuada a entrada e inicialização da variável **N**. Em seguida são declaradas e inicializadas as variáveis **valor0**, **valor1** e **valor2**. Na estrutura de repetição, a variável **valor0** será responsável por armazenar o valor corrente da série que, será impresso com a função **print**, a cada iteração. As variáveis **valor1** e **valor2** armazenam os itens anteriores da sequência utilizadas para o cálculo do novo valor corrente, e também são atualizadas a cada iteração de forma a permitir a sequência correta de cálculo dos valores.

```
1  # Entrada
2  N = int(input('Digite o número de termos da série de Fibonacci: '))
3  # Processamento e Saída
4  valor0 = 0
5  valor1 = 1
6  valor2 = 0
7  for i in range(0, N, 1):
8      print(valor0)
9      valor0 = valor1 + valor2
10     valor1 = valor2
11     valor2 = valor0
```

5.3 Estruturas de repetição aninhadas

O **aninhamento** de estruturas de repetição consiste em declarar uma estrutura de repetição internamente ao bloco de comandos de uma outra estrutura de repetição definida externamente. O aninhamento das estruturas de repetição podem compreender, dependendo do problema, vários níveis, apesar de não ser uma prática eficiente.

Em uma estrutura de repetição aninhada a execução tem início na estrutura declarada mais **externamente**. A cada iteração das estruturas mais externa implicará em uma execução completa das estruturas definidas mais internamente.

Abaixo a sintaxe de duas estruturas de repetição aninhadas. A primeira contém duas estruturas declaradas através do comando **while**. E na segunda duas estruturas declaradas com o comando **for**:

```
while <condição 1>:
    comando a.1
    comando a.2
    while <condição 2>:
        comando b.1
        comando b.2
        ...
        comando b.n
    ...
    comando a.o
```

```
for <variável 1> in range(<intervalo 1>) :
    comando a.1
    comando a.2
    for <variável 2> in range(<intervalo 2>) :
        comando b.1
        comando b.2
        ...
        comando b.n
    ...
    comando a.o
```

Apesar dos exemplos de sintaxe apresentados anteriormente, a definição de uma estrutura de repetição aninhada não se restringe somente ao uso de duas declarações com comandos **for** ou comandos **while**, podendo existir construções com o uso de um comando **for** e **while** ou vice-versa, como definimos nas seguintes sintaxes:

```
for < variável > in range(< intervalo >) :
    comando a.1
    comando a.2
    while <condição>:
        comando b.1
        comando b.2
        ...
        comando b.n
    ...
    comando a.o
```

```
while <condição>:
    comando a.1
    comando a.2
    for < variável > in range(< intervalo >) :
        comando b.1
        comando b.2
        ...
        comando b.n
    ...
    comando a.o
```

Para finalizar, uma estrutura de repetição aninhada não se limita a apenas duas declarações, ou seja, pode haver qualquer número de repetições aninhadas, caso isso seja necessário para realizar corretamente uma tarefa.

Exemplo 7:

Implemente um programa que exiba como saída a tabuada de multiplicação de 1 a 9, conforme o exemplo de saída:

```
1 x 1 = 1
1 x 2 = 1
...
1 x 8 = 8
1 x 9 = 8
2 x 1 = 2
2 x 2 = 4
...
9 x 8 = 72
9 x 9 = 81
```

Solução:

O fluxograma da Figura 26 apresenta o encadeamento de duas estruturas de repetição para uma solução do Exemplo 7. A primeira estrutura de repetição define uma sequência de valores para a variável **i** no intervalo $[1,\ 9]$. E a segunda estrutura define uma sequência de valores para a variável **j** no intervalo $[1,\ 9]$, onde para cada valor assumido pela variável **i** uma sequência no intervalo $[1,\ 9]$ para a variável **j** é gerada.

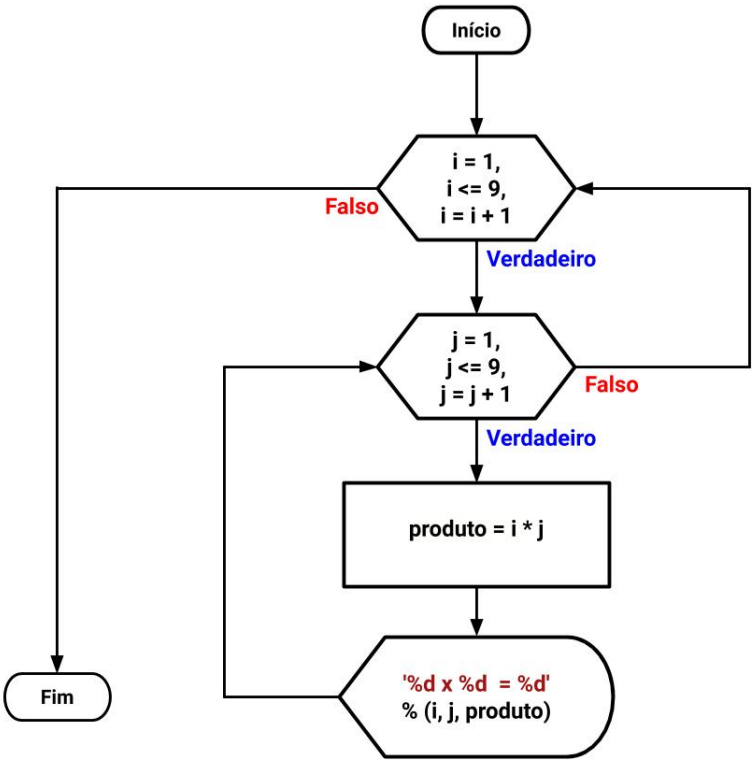

Figura 26: Fluxograma de execução para o Exemplo 7.

A seguir, o código em Python para a solução do exemplo.

```
1 # Processamento e Saída
2 for i in range(1, 10):
3     for j in range(1, 10):
4         produto = i * j
5         print(f'{i} x {j}  = {produto}')
```

Exemplo 8:

Implemente um programa que leia como entrada um número N que determina a quantidade de linhas a serem impressas na tela, contendo uma **string** com asteriscos, conforme o exemplo para $N = 5$:

```
*
**
***
****
*****
```

Solução:

O fluxograma da Figura 27 apresenta a sequência das operações contendo duas estruturas de repetição encadeadas para uma proposta de solução para o Exemplo 8.

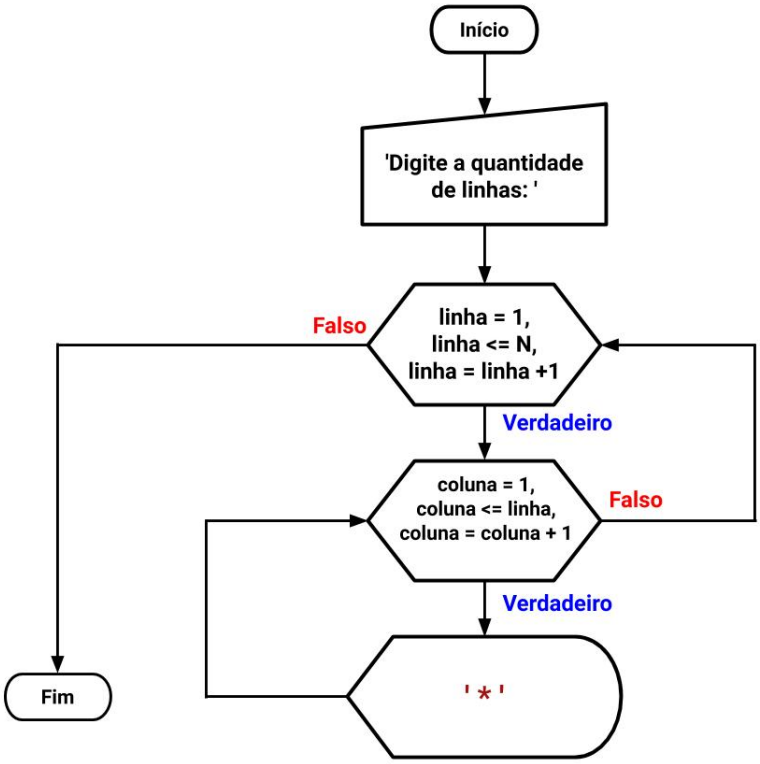

Figura 27: Fluxograma de execução para o Exemplo 8.

A seguir, o código em Python para a solução do exemplo.

```
1 # Entrada
2 N = int(input('Digite a quantidade de linhas: '))
3 # Processamento e Saída
4 for linha in range(1, N + 1):
5     for coluna in range(1, linha + 1):
6         print('*', end = '')
7     print('')
```

Neste programa tivemos a necessidade de usar um recurso especial do **Python** na função **print**. Por padrão, esta função sempre encerra sua tarefa com uma **quebra de linha**. Como não queremos que isso ocorra todas as vezes que imprimimos um caractere "*", usamos um segundo argumento par a função: **end=''**, indicando que queremos modificar o caractere final do **print** para um espaço em branco, este caractere substituirá a **quebra de linha** que não desejamos imprimir. A quebra de linha deve ser inserida apenas após a impressão do último "*" da linha corrente, isso será feito apenas com a impressão de um texto vazio, através da chamada de **print('')** após o **for** interno, e dentro do **for** externo.

Exemplo 9:

Implemente um programa que receba o valor de N e em seguida leia N sequências de valores numéricos não nulos ($valor\ !=\ 0$). Cada sequência lida é encerrada pela leitura de um valor nulo. Para cada sequência de números lidos, o programa deve calcular o somatório dos valores pares e imprimir o resultado deste somatório na tela. Caso o valor numérico não seja inteiro, deve-se arrendondá-lo para determinar se ele é par ou impar.

Solução:

No fluxograma da Figura 28 são destacadas uma sequência com as principais operações em uma proposta de solução para o Exemplo 9. No Fluxograma temos a definição de duas estruturas de repetição aninhadas. A primeira, declarada externamente, controlada por contador e a segunda, declarada internamente no bloco da primeira estrutura, controlada por uma condição lógica.

O bloco de códigos seguinte mostra uma solução desenvolvida para o Exemplo 9. Primeiro, é realizada a entrada para a variável **N** que define o número de sequências numéricas a serem geradas. Em seguida, através do comando **for** gera, a cada iteração, a quantidade de sequências numéricas a serem lidas. Logo depois, temos a inicialização da variável soma responsável por efetuar a soma do valores pares de cada sequência informada e a leitura do primeiro valor a ser informado de cada sequência. Depois de ler o primeiro valor, temos a declaração do comando **while** que fará a leitura da sequência de valores que serão informados enquanto a condição definida ($valor!\ =\ 0$) for satisfeita. Observe que se o primeiro valor for $zero$, o bloco de comandos do **while** nem chegará a ser executado a esta sequência. No bloco de comandos do **while** a primeira tarefa é arredondar o valor contido na variável **valor**, em seguida, o comando **if** verifica através da condição, se o valor informado é par, quando verdadeiro deve-se somar o valor arredondado à variável **soma**, de forma acumulativa para se obter o somatório de todos os valores pares ao final da execução. Antes de finalizar o bloco de comandos do **while** é solicitado novamente uma leitura para a variável **valor**, que será avaliado novamente na condição deste comando e processado se ele for diferente de $zero$, através da execução de mais uma iteração. Esta lógica na repetição **while** é do tipo: *ler*, *validar* e *processar*, nesta ordem. A validação é feita na própria condição do **while**, por isso precisamos ler o primeiro valor antes do **while**, nos comandos do bloco de repetição, primeiro temos que processar o valor lido, e por último, encerrando o bloco, ler o próximo valor. Concluída a excução do **while** o resultado do somatório dos valores pares da sequência definida será impresso através do comando **print** e o trabalho continua para a próxima sequência pelo comando **for**, que continuará executando até que a última sequência tenha sido definida e processada.

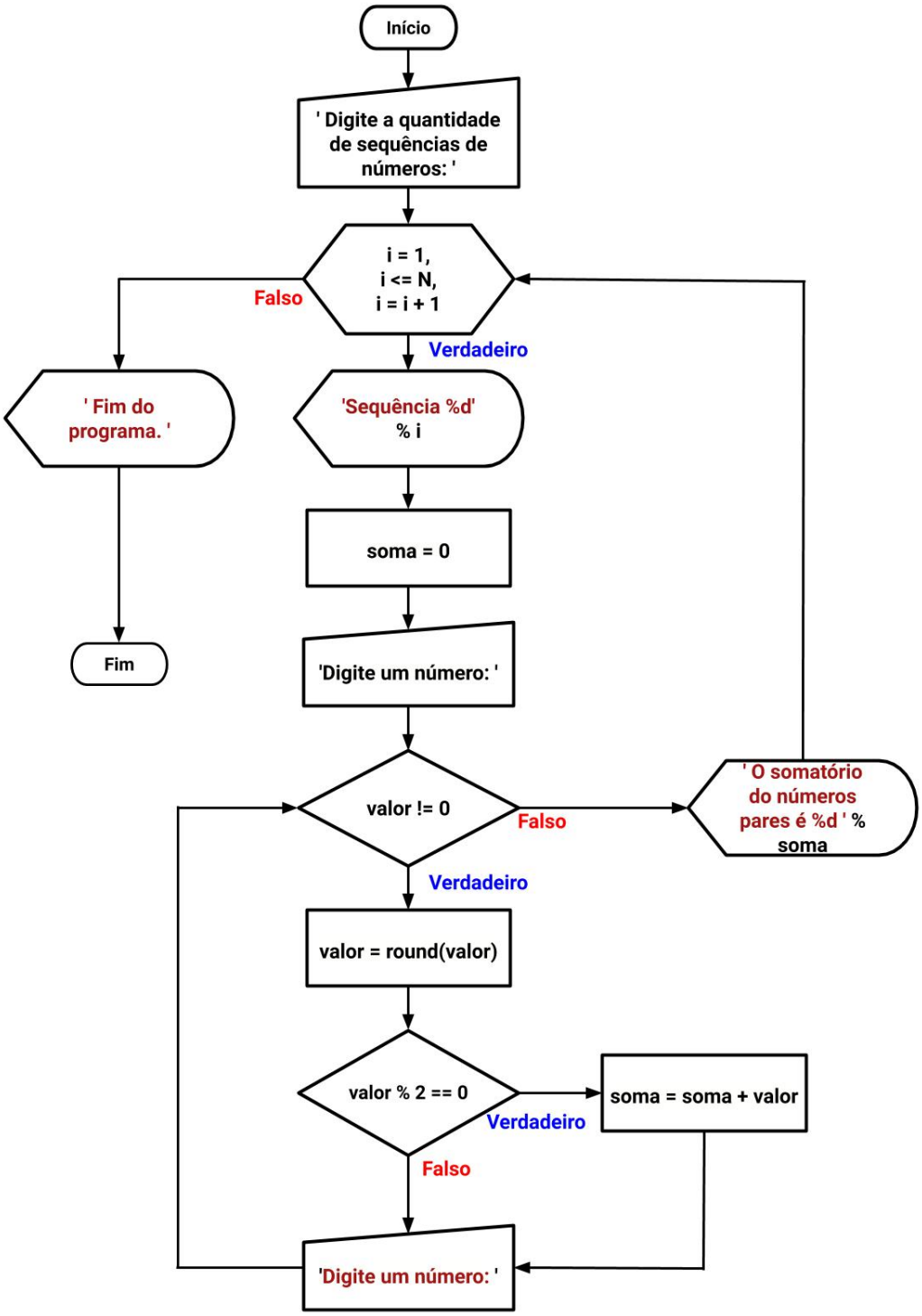

Figura 28: Fluxograma de execução para o Exemplo 9.

```
1  N = int(input('Digite a quantidade de sequências de números: '))
2  for i in range (1, N + 1):
3      print(f'Sequência {i}')
4      soma = 0
5      valor = int(input('Digite um número: '))
6      while valor != 0:
7          if valor % 2 == 0:
8              soma = soma + valor
9          valor = int(input ('Digite um número: '))
10     print(f'O somatório do números pares é {soma}')
```

5.4 Exercícios Propostos

Exercício 1

Escreva um programa que leia um valor inteiro positivo N e calcule o somatório dos valores ímpares e o somatório dos valores pares presentes no intervalo entre 1 a N. No final imprima o resultado dos somatórios.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de N: 10
A soma dos valores pares é: 30
A soma dos valores impares é: 25
```

Exemplo 2:

```
Digite o valor de N: 99
A soma dos valores pares é: 2450
A soma dos valores impares é: 2500
```

Exemplo 3:

```
Digite o valor de N: 5000
A soma dos valores pares é: 6252500
A soma dos valores impares é: 6250000
```

Exercício 2

Implemente um programa que efetue a leitura de uma quantidade indeterminada de valores inteiros positivos. Quando um valor inválido (menor ou igual a zero) for informado o processo de leitura deverá ser finalizado. No final apresente como saída o menor valor, o maior valor, a média aritmética dos valores informados. O processamento só será possível se pelo menos 2 valores válidos forem fornecidos.

Exemplos de execução:

Exemplo 1:

```
Digite um valor: 10
Digite um valor: 20
Digite um valor: 30
Digite um valor: 40
Digite um valor: 50
Digite um valor: -1
O menor valor informado é: 10
O maior valor informado é: 50
A média dos valores é: 30.00
```

Exemplo 2:

```
Digite um valor: 55
Digite um valor: 25
Digite um valor: 5
Digite um valor: 75
Digite um valor: 95
Digite um valor: 85
Digite um valor: 45
Digite um valor: -1
O menor valor informado é: 5
O maior valor informado é: 95
A média dos valores é: 55.00
```

Exemplo 3:

```
Digite um valor: 101
Digite um valor: 555
Digite um valor: 780
Digite um valor: 896
Digite um valor: 155
Digite um valor: -1
O menor valor informado é: 101
O maior valor informado é: 896
A média dos valores é: 497.40
```

Exemplo 4:

```
Digite um valor: -1
Quantidade de valores insuficiente
```

Exemplo 5:

```
Digite um valor: 10
Digite um valor: -1
Quantidade de valores insuficiente
```

Exemplo 6:

```
Digite um valor: 10
Digite um valor: 20
Digite um valor: -1
O menor valor informado é: 10
O maior valor informado é: 20
A média dos valores é: 15.00
```

Exercício 3

Implemente um programa que leia dois valores inteiros positivos X e Y. O programa deve calcular a potência X^Y e imprimir o resultado como saída.

Exemplos de execução:

Exemplo 1:

```
Digite o valor da base x: 2
Digite do expoente y: 5
2 ^ 5 = 32
```

Exemplo 2:

```
Digite o valor da base x: 10
Digite do expoente y: 0
10 ^ 0 = 1
```

Exemplo 3:

```
Digite o valor da base x: 9
Digite do expoente y: 9
9 ^ 9 = 387420489
```

Exercício 4

Implemente um programa que leia dois números inteiros positivos x e y quaisquer e calcule o Máximo Divisor Comum (MDC). E no final imprima o resultado.

Observação: Para saber se um número é divisível por outro, normalmente verificamos se o resto da divisão é $zero$. Em Python, isso pode ser feito usando o operador %, que retorna o resto da divisão. Por exemplo, $9 \% 3$ resulta em $zero$, $5 \% 2$ resulta em 1, $24 \% 5$ resulta em 4. Assim, para saber se **A** é divisível por **B**, use a expressão lógica $A \% B == 0$.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de x: 120
Digite o valor de y: 90
O MDC é: 30
```

Exemplo 2:

```
Digite o valor de x: 175
Digite o valor de y: 150
O MDC é: 25
```

Exemplo 3:

```
Digite o valor de x: 10892
Digite o valor de y: 10780
O MDC é: 28
```

Exercício 5

Elabore um programa que leia um valor inteiro positivo N e no final imprima todos os valores primos inferiores a N.

Observação: Os números primos são números naturais que possuem apenas dois divisores: 1 e ele mesmo. Por definição, o número 1 não é considerado primo.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de N: 10
2
3
5
7
```

Exemplo 2:

```
Digite o valor de N: 50
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
```

Exemplo 3:

```
Digite o valor de N: 100
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
```

MÓDULO

6 Funções

Nos módulos anteriores, aprendemos que o fluxo de um programa de computador é usualmente composto por três principais etapas: **Entrada**, **processamento** e **saída**. Aprendemos ainda a usar operações e funções definidos na própria linguagem **Python** para resolver nossos problemas. Relembre algumas funções definidas dentro do próprio **Python** que você já utilizou:

- Entrada e saída: **input** e **print**;
- Conversão de tipo: **int**, **float** e **str**;
- Cálculos matemáticos: **abs**, math.**sqrt**, math.**cos**, math.**ceil** e math.**pow**

Além das funções definidas dentro do próprio escopo da linguagem e seus módulos, podemos também definir **nossas próprias funções**. Isso permite reaproveitar, sempre que necessário, o mesmo **bloco de operações**.

As principais vantagens no uso de funções são:

- Códigos mais legíveis, especialmente para programas grandes;
- Manutenções e modificações no código são mais fáceis, com menor chance de prejudicar o que já foi feito;
- Possibilidade de reaproveitamento da mesma estrutura em diversas seções do código-fonte.

As funções são blocos que tomam **argumentos de entrada** (*input*), realizam operações e retornam um **valor de saída** (*output*), conforme ilustrado na Figura 29. As entradas e saídas podem ser variáveis de qualquer tipo.

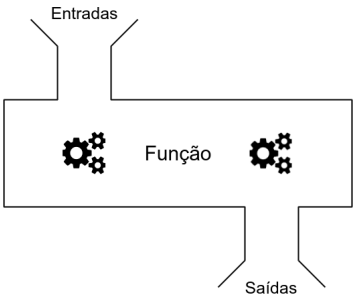

Figura 29: Representação ilustrativa de uma função.

6.1 Definição de Funções

Para que o **Python** reconheça um bloco como uma função, ele precisa estar organizado e indentado. O comando para definir funções é **def**. Abaixo apresentamos a sintaxe utilizada para esse fim:

```
def <nome da função> (<argumentos>):
    comando 1
    comando 2
    ...
        return <resultado>
    ...
    comando n
    return <resultado>
```

A palavra **def** determina que será definida uma função, que será composta de:

- **<nome da função>**: assim como as variáveis criadas pelo programador, as funções precisam ter um nome. As regras definidas são as mesmas das variáveis;
- **<argumentos>**: os argumentos determinam nomes de variáveis internas da função, automaticamente criadas quando a função for chamada, e responsáveis por receber os valores que devem ser fornecidos como entrada para a função. Uma função pode não receber argumento algum, ou um número arbitrário de argumentos. Isto ficará mais claro ao longo dos exemplos e exercícios apresentados a seguir;
- **corpo da função**: a corpo da função determina a sequência de comandos que realiza a tarefa necessária e gera os resultados esperados com a execução da função. Ele será composto de **comandos**, que realizam a tarefa, e cláusulas **return**, que encerram a execução da função e retornam os resultados de saída gerados. Estes resultados serão repassados para o *chamador* da função. Se a função não precisa retornar nada, a cláusula **return** não precisa ser usada, a menos que se deseje explicitamente definir o término da sua execução em algum momento. Quando esta cláusula não é usada, o término da execução ocorre da mesma forma que em um programa, ou seja, quando seu último comando for executado, e nada será retornado;
- A utilização dos parênteses, caracteres “(” e “)”, delimitando os argumentos, e do caractere “:” após o fechamento dos parênteses é obrigatória, assim como a indentação do código que representa o corpo da função.

6.2 Chamada de Funções

Para chamar uma função no **Python**, basta escrever o nome dela dentro do código, com os argumentos entre parêntesis. Isso você já faz há muito tempo, quando usa as funções definidas internamente no **Python**, exemplificadas no início deste módulo (como **print**, **int** e **abs**, por exemplo). Observe o programa abaixo:

```
1  # Definindo de funções (isto é só um comentário, não é obrigatório)
2  def my_function(x):
3      y = 2*x + 3
4      return y
5
6  # Programa principal (outro comentário, não obrigatório)
7  y1 = my_function(1)
8  y2 = my_function(2)
9  y3 = my_function(3)
10 print(f'y1 = {y1}, y2 = {y2}, y3 = {y3}!')
```

Neste programa temos a **definição da função**, entre as linhas 2 e 4, e o que denominamos **programa principal**, entre as linhas 7 e 9. As demais linhas são comentários ou estão em branco e são ignoradas na execução do programa, utilizados aqui apenas para organizar e facilitar o entendimento do programa. Ao iniciar a execução, primeiramente a definição da função será **interpretada**, todas as informações relacionadas à função (nome, argumentos e corpo) serão registradas na memória e deixadas lá para quando a função for chamada. Até agora ele não executou a função, apenas registrou para executar futuramente. A execução do programa de fato começa na atribuição da variável **y1**, linha 7, neste momento há uma chamada à função **my_function**. Devido à esta chamada, a função será de fato executada. Isso é feito com um **desvio de fluxo** de execução do programa, da linha 7 para a linha 2, onde será transferido o valor 1, passado como argumento de entrada para a função, para a variável **x** da função, cujo nome foi determinado na definição da função. Feita a transferência dos argumentos de entrada, inicia-se a execução do corpo da função. Neste caso, a linha 3 será executada, criando uma outra variável dentro da função, chamada de **y** e assumirá o valor $2 * 1 + 3 = 5$, já que $x = 1$. Concluída a atribuição, o próximo comando a ser executado encontra-se na linha 4, cujo resultado é o retorno do valor da variável **y** para quem fez a chamada da função. Neste momento, um novo **desvio de fluxo** é realizado, concluindo a execução da função e retornando para a linha 7, de onde a chamada ocorreu. Neste desvio, o valor de **y**, que é 5, assumirá o lugar da função, gerando a atribuição $y1 = 5$, cujo resultado é a criação da variável **y1**, com o valor 5. Esta ação conclui a execução da linha 7, o que faz com que seja realizada a execução da linha 8. De forma similar, tudo que aconteceu durante a execução da linha 7, acontecerá de novo agora, já que as duas linhas são muito parecidas. Porém, agora trata-se de outra chamada à função, com o argumento de entrada no valor 2, sendo assim, novamente dentro da função, será definido um novo valor para a variável **y**, $y = 2 * 2 + 3 = 7$, que define um novo valor de retorno, a ser atribuído a **y2**, ou seja, $y2 = 7$. Na linha 9, tudo acontece novamente, mas o resultado final agora será $y3 = 9$. O programa se encerrará após a execução da função **print**, linha 10, tendo executado a função em 3 ocasiões distintas, com valores de entrada e saída diferentes, atribuindo os valores 5, 7 e 9, às variáveis do programa principal **y1**, **y2** e **y3**, respectivamente, imprimindo a mensagem `"y1 = 5, y2 = 7, y3 = 9!"` no terminal.

Exemplo 1:

Crie um programa que realize a soma ou a subtração de dois números reais fornecidos pelo usuário, imprimindo em segui o resultado destas operações. Porém, deseja-se que as duas operações sejam realizadas por duas funções, uma para cada operação. As duas funções receberão como entrada os dois números e retornará como resultado a operação aplicada aos dois números.

Solução:

Não há limitação lógica do número de funções definidas em um programa, nem tampouco do número de argumentos passados a uma função. A única restrição é que as regras da linguagem sejam respeitadas. Sendo assim, podemos implementar o seguinte programa, que resolve o problema respeitando as suas restrições:

```
1  # Declaração de funções
2  def soma(num1, num2):
3      resultado = num1 + num2
4      return resultado
5
6  def subtracao(num1, num2):
7      return num1 - num2
8
9  # Programa principal
10 num1 = float(input('Digite o primeiro valor: '))
11 num2 = float(input('Digite o segundo valor: '))
12 s = soma(num1, num2)
13 print(f'A soma é {s}')
14 print(f'A subtração é {subtracao(num1, num2)}'}
```

Aproveitamos este exemplo para explicar vários conceitos relacionados às funções. Primeiro, em linhas gerais, definimos duas funções que têm como objetivo retornar uma soma e uma subtração, respectivamente. Em um programa principal, fizemos a leitura dos números, as chamadas das funções e a impressão dos resultados. Adotamos estratégias diferentes em relação às duas funções. Veja que, para a soma, utilizamos uma variável para armazenar o resultado da operação que, em seguida, é retornado como resultado da execução da função. Para a subtração, nós não utilizamos uma variável, simplesmente retornamos como resultado da função o resultado direto da expressão. Observe que, assim como para os programas, existem variadas formas de se implementar as funções, o importante é que elas cumpram a sua tarefa corretamente. Para isso, você pode usar os recursos disponíveis pela linguagem **Python** para fazer o que precisa ser feito pela função, isso inclui variáveis, operadores matemáticos, decisões, repetições, e até mesmo chamadas de funções.

Outro ponto importante explorado no exemplo é com relação às diferentes possibilidades de chamadas das funções. Observe que, mais uma vez, foi criada uma variável para a obtenção da soma. Na linha 12 a variável **s** será criada com o valor retornado pela chamada da função **soma** para depois ser impressa na tela. A variável não pode ter o mesmo nome da função, senão teremos um conflito que poderá causar um mal funcionamento do programa. Tome sempre muito cuidado com os nomes de variáveis e funções. Por outro lado, para a subtração não foi criada uma variável. Já que o objetivo é apenas imprimir o resultado na tela, a chamada da função **subtracao** foi feito dentro do **print**, para que o resultado retornado seja diretamente usado para compor a f-string utilizada para determinar a mensagem.

6.3 Escopo de variáveis

As variáveis criadas no programa principal e dentro das funções possuem o que chamamos de **escopo** diferentes. O **escopo** de uma variável é onde ela **existe**, ou seja, onde ela pode ser reconhecida e utilizada. Os argumentos de uma função e as variáveis criadas dentro dela estão **exclusivamente** dentro do escopo da função, ou seja, só existem e podem ser referenciadas dentro da função, elas não existem fora dela. Por isso, denominamos este escopo como **local**, pois as variáveis só existem localmente na função onde foram declaradas. As variáveis definidas dentro do programa principal possuem escopo **global**, ou seja, existem em todo o programa, incluindo as funções definidas dentro dele. Obviamente, uma variável só passa a existir após ter algum valor atribuído a ela.

Uma questão muito importante, que temos como uma **boa prática de programação**, é evitar fazer uso de variáveis globais dentro de funções. Embora acessíveis, recomenda-se que valores necessários para a execução de uma função sejam passados como argumentos de entrada. Já para os valores definidos dentro da função, que sejam tratados como valores de retorno da função. Ou seja, evite acessar ou definir valores de variáveis globais dentro de uma função.

Para finalizar, é preciso ressaltar que variáveis de escopo diferentes podem possuir o mesmo nome, sem embaralhar os seus respectivos valores. Variáveis globais **coexistem** com variáveis locais de mesmo nome. Porém, dentro de uma função prevalecem as variáveis **locais**, ou seja, as variáveis globais de mesmo nome não são acessíveis.

Exemplo 2:

Apresentamos uma segunda versão de programa para resolver o Exemplo 1 para observarmos algumas questões relacionadas ao escopo de variáveis.

```
1  # Declaração de funções
2  def soma(num1, num2):
3      resultado = num1 + num2
4      return resultado
5
6  def subtracao(num1, num2):
7      resultado = num1 - num2
8      return resultado
9
10 # Programa principal
11 num1 = float(input('Digite o primeiro valor: '))
12 num2 = float(input('Digite o segundo valor: '))
13 resultado = soma(num1, num2)
14 print(f'A soma é {resultado}')
15 resultado = subtracao(num1, num2)
16 print(f'A subtração é {resultado}'}
```

Agora temos inúmeras variáveis de mesmo nome, mas escopos diferentes. Temos 3 variáveis que se chamam **num1**, no programa principal e nas duas funções. Isso significa que em alguns momentos do fluxo de execução do programa nós teremos duas variáveis **num1**. Isso porque as variáveis locais deixam de existir quando as funções que as declararam terminam sua executar (lembrando que elas só serão criadas quando a função for executada. Como em nosso programa as funções **soma** e **subtracao** não podem estar executando ao mesmo tempo, suas variáveis não poderão coexistir durante nenhum fluxo de execução do programa. Porém, temos a variável global **num1**, que poderá então coexistir com uma das variáveis de mesmo nome das duas funções. O mesmo acontece com a variável **resultado**.

Apenas para ilustrar o que acontece com os escopos das variáveis e como os valores são manipulados, execute o código do programa a seguir para ver o resultado. Antes, tente definir o resultado que você esperaria ver apenas interpretando o código.

```
1  def f1(x):
2      print(x)
3      x = x * 2
4      print(x)
5      return x
6
7  def f2(x):
8      print(x)
9      x = x * 3
10     print(x)
11     return x
12
13 x = 5
14 print(x)
15 y = f1(x)
16 print(x)
17 print(y)
18 y = f2(y)
19 print(x)
20 print(y)
```

6.4 Retornando vários valores

Eventualmente é necessário que uma função retorne mais de um valor como resultado. Para fazer isso em **Python**, basta colocar todos os valores separados por vírgula na cláusula **return** e considerar esta questão na chamada da função. Veja o exemplo a seguir e os comentários que na sequência.

Exemplo 3:

Imagine que você quer fazer um programa que calcula as raízes de uma equação de segundo grau. Sabemos que o formato dessas equações é:

$$ax^2 + bx + c = 0$$

Ainda, sabemos que as raízes são calculadas usando a fórmula de Bháskara:

$$x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

E por último, sabemos que:

$$\Delta = b^2 - 4ac$$

Vamos criar um programa que tem duas funções definidas. Uma para calcular o delta e outra para calcular as raízes. Como são duas raízes, esta função deverá retornar dois valores.

É importante lembrar que precisamos verificar se as raízes reais existem ou não. **Observação:** Para simplificar, utilizaremos o tipo de dado **None** (que em **Python** representa um valor **nulo**) para indicar que não houve nenhum retorno para as raízes.

Vejamos então o programa:

```
1  import math
2
3  def delta(a, b, c):
4      d = (b**2) - (4*a*c)
5      return d
6
7  def raizes(a, b, c):
8      d = delta(a,b,c)
9      if (d < 0):
10         return None, None
11     else:
12         x1 = (-b + (math.sqrt(d)))/(2*a)
13         x2 = (-b - (math.sqrt(d)))/(2*a)
14         return x1, x2
15
16 # Entrada
17 a = int(input('Digite o valor do termo A: '))
18 b = int(input('Digite o valor do termo B: '))
19 c = int(input('Digite o valor do termo C: '))
20 # Processamento
21 r1, r2 = raizes(a,b,c)
22 # Saída
23 if (r1 != None):
24     print(f'Raízes: {r1:.2f} e {r2:.2f}')
25 else:
26     print('Não existem raízes reais')
```

Como a função **raizes** retorna dois valores, um para a primeira raiz e outro para a segunda, observe que a cláusula **return**, desta função, sempre retorna dois valores separados por vírgula. Por consequência, quando esta função é chamada, espera-se também receber dois valores, na atribuição, antes do operador de atribuição nós definimos duas variáveis para receber os valores retornados. Neste caso, a atribuição respeitará a ordem das variáveis e dos valores retornados. Implementando desta maneira, há restrições à chamada da função, recomendamos usar sempre a atribuição para receber os valores retornados diretamente para serem atribuídos a variáveis.

É importante ressaltar também que, no caso de **delta** ser um valor negativo, não existirão raízes reais, então, o resultado será uma mensagem indicando esta situação. Para identificá-la, a função **raizes** retorna dois valores nulos (**None**) e, no programa principal, uma decisão é tomada com base nisso. Se uma das raízes for diferente de **None**, enão imprime-se as raízes, caso contrário, imprime-se a mensagem de erro. Não é necessário testar as duas variáveis, com apenas uma delas já é possível tomar a decisão, neste caso, apenas a variável **r1** foi testada.

6.5 Exercícios Propostos

Exercício 1

Fazer uma função que calcula a distância euclidiana entre dois pontos bidimensionais e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função

implementa. Utilizar x_1, x_2, y_1 e y_2 como argumentos da função.

Fórmulas:

- $P_1 = (x_1, y_1)$
- $P_2 = (x_2, y_2)$
- $D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Exemplos de execução:

Exemplo 1:

```
Digite o valor da coordenada X1: 1
Digite o valor da coordenada Y1: 2
Digite o valor da coordenada X2: 2
Digite o valor da coordenada Y2: 3
A distância entre os pontos é 1.41
```

Exercício 2

Fazer uma função que converte uma entrada de temperatura em Celsius para Fahrenheit e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_F = (T_c.\frac{9}{5}) + 32$

Exemplos de execução:

Exemplo 1:

```
Digite uma temperatura em Celsius: 37
A temperatura em Fahrenheit é 98.6
```

Exercício 3

Fazer uma função que converte uma entrada de temperatura em Fahrenheit para Celsius e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_C = (T_F - 32).\frac{5}{9}$

Exemplos de execução:

Exemplo 1:

```
Digite a temperatura em Fahrenheit: 98.6
A temperatura em Celsius é 37.0
```

Exercício 4

Fazer uma função que calcula a extensão de uma mola vertical com constante elástica k ao pendurar um objeto de massa m. Utilizar k e m como argumentos da função e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmulas:

- $F_e = k.x$
- $F_g = m.g$
- $g = 9,807 m/s^2$

Exemplos de execução:

Exemplo 1:

```
Digite a massa do objeto: 5
Digite a constante da mola: 100
A extensão da mola é 0.49 m
```

Exercício 5

A inclinação de um telhado é dada por um percentual da largura, conforme a Figura 30.

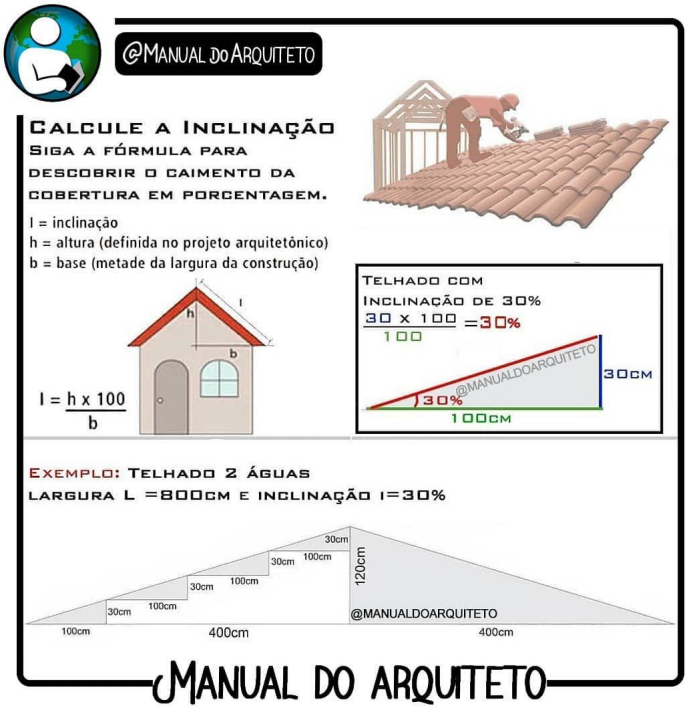

Figura 30: Esquema de inclinação de telhado.

Quando o telhado é de "duas águas", a largura deve ser dividida por dois antes de calcular a altura. Esses dados podem ser usados para calcular o comprimento do telhado, que é a hipotenusa do triângulo retângulo.

Fazer duas funções: uma função que calcula o comprimento c_1 de um telhado de duas águas e uma que calcula o comprimento c_2 de um telhado de uma água, dadas como entradas uma largura l e um percentual p.

Implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada às funções implementas.

Fórmulas (uma água):

- $h = \frac{p.l}{100}$
- $c_1 = \sqrt{l^2 + h^2}$

Fórmulas (duas águas):

- $h = \frac{p.(l/2)}{100}$
- $c_2 = 2.\sqrt{(l/2)^2 + h^2}$

Observação: Os resultados encontrados devem ser iguais (por semelhança de triângulos, é fácil observar essa relação).

Exemplos de execução:

Exemplo 1:

```
Digite a largura do telhado: 10
Digite a inclinação percentual: 30
O comprimento do telhado de uma água é 10.44
O comprimento do telhado de duas águas é 10.44
```

MÓDULO 7

Estruturas Homogêneas

Em módulos anteriores, aprendemos quais tipos de dados podem ser armazenados e como utilizar esses valores em **Python**. Alguns dos tipos de dados que aprendemos a usar são: Inteiros (**int**); Números de ponto flutuante (**float**); texto (**string**); Lógicos (**boolean**).

Aprendemos também a armazenar esses tipos de dados dentro de variáveis, conforme a nossa demanda, como por exemplo:

```
1 # Declarações de Variáveis
2 nome = 'Pedro'
3 idade = 15
4 matriculado = True
```

Contudo, algumas vezes queremos armazenar toda uma coleção de dados na memória durante a execução de um programa. Suponha que em um programa, você queira registrar o valor da conta de luz dos 15 moradores do seu prédio. Somente com as ferramentas ensinadas até o momento, seriam necessárias 15 variáveis diferentes, cada uma delas contendo o valor da conta de luz de cada morador:

```
1  conta_m0 = input('Digite o valor da conta: ')
2  conta_m1 = input('Digite o valor da conta: ')
3  conta_m2 = input('Digite o valor da conta: ')
4  conta_m3 = input('Digite o valor da conta: ')
5  conta_m4 = input('Digite o valor da conta: ')
6  ...
7  conta_m11 = input('Digite o valor da conta: ')
8  conta_m12 = input('Digite o valor da conta: ')
9  conta_m13 = input('Digite o valor da conta: ')
10 conta_m14 = input('Digite o valor da conta: ')
```

Além de ocupar muito espaço e tornar o código difícil de ler e entender, essa solução dificulta a capacidade de realizar operações sobre todos os dados em conjunto. Se quisermos compreender a média do valor das contas de energia, por exemplo, precisaremos escrever manualmente todas as operações.

Para facilitar o agrupamento de dados durante a execução do programa, existem estruturas chamadas **Vetores** e **Matrizes**.

- Os **Vetores** são estruturas de uma dimensão, contendo uma lista de dados de determinado tipo.
- As **Matrizes** são estruturas de duas dimensões, que armazenam uma tabela endereçada contendo os dados de determinado tipo.

Nesse módulo, compreenderemos como criar e utilizar **vetores** e **matrizes**.

7.1 Vetores

Uma das maneiras mais comuns que utilizamos para agrupar dados é na forma de listas. Fazemos listas de compras nos supermercados, listas de afazeres, listas de objetos para colocar na mala. Da mesma maneira, uma das formas mais simples de armazenar dados de um tipo é através de listas, aqui denominadas **Vetores**. Conforme definido anteriormente, **vetor** é um tipo de dado **homogêneo**, portanto, todos os dados armazenados devem ser do mesmo tipo. Na Figura 31 representamos uma lista de nomes de produtos de um supermercado agrupados em um vetor.

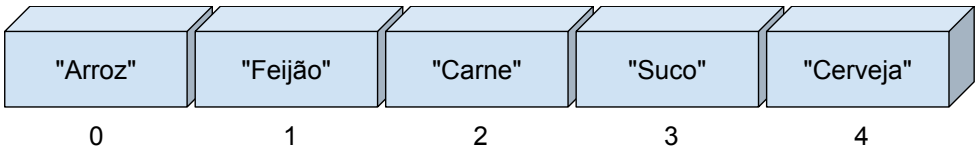

Figura 31: Representação ilustrativa de um vetor de *strings*.

Nessas estruturas de dados, cada "caixa" pode receber um valor, como se fosse uma variável. Assim como as casas de uma rua ou caixas de correio de um prédio, os contêineres de um vetor são **endereçados** através de um número. Em **Python**, o primeiro endereço dentro de um vetor é o $zero$ (0), seguindo sequencialmente com acréscimo de 1 unidade, ou seja, os endereços são valores inteiros a partir de $zero$. Esses endereços são chamados **índices** dos elementos do vetor. Sendo assim, podemos expressar um vetor A com n elementos da seguinte forma:

$$A_n = \begin{bmatrix} a_0 & a_1 & a_2 & .. & a_{n-1} \end{bmatrix}$$

onde: a_i representa o elemento de índice **i**, armazenado no vetor.

7.1.1 Sintaxe básica

Agora que compreendemos o conceito da estrutura de dados **vetor**, precisamos entender como criá-los em **Python**. Em **Python** existem muitas estruturas de dados compostas, buscamos uma estrutura com sintaxe simples para representar o conceito de vetor em nosso curso, a estrutura de dado denominada **lista**. Em uma lista no **Python** os elementos armazenados podem ser de qualquer tipo, permitindo misturar elementos de variados tipos. Porém, como os elementos podem ser de tipos diferentes, esta estrutura do **Python** não é conceitualmente um **vetor**. Sendo assim, nos referiremos às **listas** em **Python** como **vetores** toda vez que seus elementos forem sempre de um mesmo tipo de dados. Em nossos programas, trataremos as variáveis **lista** como **vetores**, garantindo sempre armazenar valores de um único tipo.

Um vetor é declarado de maneira similar a uma variável, porém com colchetes indicando a coleção de dados. Vamos entender isso melhor. Um vetor de inteiros, por exemplo, pode ser declarado já contendo os dados delimitados por colchetes e separados por vírgulas:

```
1 # Vetor de valores numéricos
2 num_vector = [ 2, 4.3, 6, 8.2, 10.8 ]
3
4 # Vetor de booleanos
5 bool_vector = [ True, False, True, True ]
6
7 # Vetor de strings
8 words_vector = [ 'alpha', 'bravo', 'celta' ]
```

Observe que tivemos a preocupação de definir sempre elementos de um mesmo tipo de dados, isso está garantindo que, conceitualmente, todas as variáveis do programa anterior sejam denominadas **vetores** no contexto de nossa disciplina.

Um vetor pode inclusive ser declarado vazio, para ser preenchido posteriormente, conforme o exemplo a seguir. Este recurso será muito utilizado futuramente.

```
1 # Vetor vazio
2 empty_vector = [ ]
```

A seguir, vamos entender melhor como criar, modificar e adicionar dados a um vetor.

7.1.2 Operações com vetores

Na seção anterior, aprendemos a sintaxe básica dos vetores. Agora, aprenderemos a utilizar esses objetos. As principais operações com vetores são:

- Criar um vetor;
- Inserir elementos em um vetor;
- Acessar/modificar elementos de um vetor;
- Remover elementos de um vetor;
- Descobrir a dimensão de um vetor.

Criando um vetor: Conforme demonstramos antes, é possível criar um vetor já com alguns valores alocados ou mesmo um vetor vazio.

```
1 vetor_vazio = [ ]
2 vetor_inteiros = [ 1, 2, 3, 4 ]
```

Inserindo elementos em um vetor: Para inserir elementos em um vetor, utilizamos a função **append**. Veja abaixo como utilizar:

```
1 V = []
2 # Inserindo o número 1 no vetor
3 V.append(1)
```

Acessando/modificando elementos de um vetor: Um elemento de um vetor tem o comportamento semelhante a uma variável simples de qualquer tipo. Para tratar de um elemento individualmente, basta utilizar o índice correspondente ao endereço que se deseja acessar ou modificar. Veja abaixo como funciona:

```
1 V = [ 1, 2, 3, 4, 5 ]
2 print(f'V no endereço 2: {V[2]}')
3 # O programa vai imprimir:
4 # V no endereço 2: 3
5 V[0] = V[0] + 1
6 # O vetor V passa conter: [ 2, 2, 3, 4, 5 ]
```

Removendo elementos de um vetor: Para remover elementos de um vetor, utilizaremos a função pop. Essa função recebe o índice do elemento a ser removido. Além de remover o elemento da posição informada, ela retorna o valor armazenado. Lembrando que o endereçamento de vetores começa no zero.

```
1 V = [ 1, 2, 3, 4, 5 ]
2 x = V.pop(1)
3 # O vetor V passa conter: [ 1, 3, 4, 5 ]
4 print(x) # imprime o valor 2
```

Descobrindo a dimensão de um vetor: Para descobrir a dimensão de um vetor, utilizamos a função **len**. Ela retorna o tamanho do vetor passado como argumento. Veja como utilizar abaixo.

```
1 V = [ 1, 2, 3, 4, 5 ]
2 tamanho = len(V)
3 # tamanho = 5
```

Exemplo 1 (média de N valores):

No começo desse capítulo falamos sobre o problema da média de vários valores. Uma das maneiras de organizar esses dados é dentro de vetores. Vejamos como é possível escrever um programa que calcula a média de um conjunto de valores armazenados em um vetor.

Primeiramente, organizamos nosso problema em seções de entrada, processamento e saída.

1. Entrada:
   (a) Valores para calcular a média;
2. Processamento:
   (a) Somar valores;
   (b) Dividir pelo tamanho do vetor;
3. Saída:
   (a) Valor da média;

A seguir, o programa em **Python**:

```
valores = [ ] # inicialmente vazio (ainda não lemos nenhum número)
run = True # para entrar no laço pelo menos uma vez
#Entrada
while (run):
    continuar = input('Digite o próximo valor (Para terminar digite "N
       " ou "n"): ')
    if (continuar == "N" or continuar == "n"):
        run = False # deve parar de ler valores
    else:
        val = float(inp) # convertendo o valor informado
        valores.append(val) # adicionando o valor no vetor
#Processamento
soma = 0 # valor inicial, ainda não processamos nenhum valor
for i in range(len(valores)):
    soma = soma + valores[i] # somatório cumulativo
media = soma / len(valores)
#Saída
print(f'O valor da média é: {media:%.2f}')
```

Exemplo 2 (elementos de uma função linear):

Como sabemos, uma função linear é dada por uma relação do tipo:

$$f(x) = ax + b$$

Podemos usar dois vetores, um x e um y para representar pares de pontos pertencentes a essa função. Vamos analisar então como fazer um programa que toma os elementos a e b como entrada, além de um número de pares desejado, e gera pares de pontos positivos pertencentes a essa função, imprimindo os valores ao final.

A estrutura do nosso programa é:

1. Entrada:
   (a) Número de pares desejados;
   (b) Valores a e b;
2. Processamento:
   (a) Gerar vetor X;
   (b) Utilizando a, b e X, gerar o vetor Y;
3. Saída:
   (a) Imprimir os pares de pontos usando X e Y.

A seguir, o programa em **Python**:

```
# Entrada
n = int(input('Digite o numero de pares desejado: '))
a = float(input('Digite o valor do termo "a": '))
b = float(input('Digite o valor do termo "b": '))
# Processamento
X = []
Y = []
for i in range(n):
    X.append(i)
    Y.append(a*i+b)
# Saída
print('Pares de pontos:')
for i in range(n):
    print(f'{X[i]} {Y[i]}')
```

7.1.3 Exercícios propostos – Vetores

Exercício 1

Utilizando vetores, imprimir uma sequência de pontos indicando a distância de queda livre de um corpo, dada uma sequência de tempo de $t = 0$ até um valor dado como entrada pelo usuário. Os valores impressos devem ter uma precisão de uma casa decimal.

Fórmulas:

- $g = 9.807 m/s^2$
- $d = \frac{gt^2}{2}$

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
```

Exemplo 2:

```
Digite o tempo final: 6
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
4 78.5
5 122.6
6 176.5
```

Exercício 2

Uma função de segundo grau é dada pela fórmula:

$$f(x) = ax^2 + bx + c$$

Dadas como entradas:

- Valores a, b e c
- Ponto x_0 inicial
- Ponto x_f final

Criar dois vetores com os dados dos valores da função nesses pontos.
Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -3
Digite o valor final de x: 3
x f(x)
-3 13
-2 6
-1 3
0 4
1 9
2 18
3 31
```

Exemplo 2:

```
Digite o valor de "a": 2
Digite o valor de "b": 0
Digite o valor de "c": 0
Digite o valor inicial de x: -5
Digite o valor final de x: 2
x f(x)
-5 50
-4 32
-3 18
-2 8
-1 2
0 0
1 2
2 8
```

Exercício 3

Implemente um programa que realize a entrada de dados de um vetor. Primeiro o usuário fornece a quantidade de elementos do vetor, em seguida, fornece um valor numérico para cada elemento. Após preenchido o vetor, os valores armazenados no mesmo devem ser impressos na tela com uma precisão de uma casa decimal.

Exemplos de execução:

Exemplo 1:

```
Informe a dimensão do vetor: 3
Valor do elemento 0: 3.5
Valor do elemento 1: 7
Valor do elemento 2: 10.65
Elementos do vetor:
3.5
7.0
10.7
```

Exemplo 2:

```
Informe a dimensão do vetor: 5
Valor do elemento 0: 6.92
Valor do elemento 1: 4
Valor do elemento 2: 5.271
Valor do elemento 3: 2.34
Valor do elemento 4: 2.319
Elementos do vetor:
6.9
4.0
5.3
2.3
2.3
```

7.2 Matrizes

Na última seção, aprendemos como criar, manipular e utilizar vetores para os mais diversos fins. Nessa seção, aprenderemos a utilizar **matrizes**. Como uma **estrutura homogênea**, uma matriz é uma estrutura de n linhas por m colunas, onde cada elemento armazena um dado de um tipo. Sendo assim, podemos expressar uma matriz A, com dimensão $n \times m$ da seguinte forma:

$$A_{n\times m} = \begin{bmatrix} a_{0,1} & a_{0,1} & a_{0,2} & .. & a_{0,m-1} \\ a_{1,1} & a_{1,1} & a_{1,2} & .. & a_{1,m-1} \\ a_{2,1} & a_{2,1} & a_{2,2} & .. & a_{2,m-1} \\ : & : & : & .. & : \\ a_{n-1,1} & a_{n-1,1} & a_{n-1,2} & .. & a_{n-1,m-1} \end{bmatrix}$$

Podemos interpretar as matrizes como "vetores de vetores". Cada uma das n linhas da matriz corresponderá a um vetor contendo m elementos. Utilizaremos essa interpretação para ilustrar a maneira de criar, acessar e modificar elementos de uma matriz. Também podemos dizer que um **vetor** possui **uma**

dimensão, enquanto uma **matriz** possui **duas dimensões**. As linguagens de programação permitem criar estruturas com mais do que duas dimensões, porém, neste curso nós exploraremos apenas uma e duas dimensões.

7.2.1 Sintaxe básica

De forma similar aos vetores, existem variadas estruturas de dados compostas capazes de representar **matrizes** em **Python**. Novamente, queremos escolher uma estrutura com sintaxe simples. Sendo assim, utilizaremos a mesma estratégia utilizada nos vetores e representaremos uma matriz em **Python** através do tipo **lista**, garantindo que sempre teremos elementos de um mesmo tipo, a diferença é que agora também garantiremos que ela tenha duas dimensões.

Consideraremos uma matriz com tamanho predefinido antes de preencher seus valores. Assim, a interpretação de uma matriz é bem similar à de uma matriz matemática. Um elemento qualquer de uma matriz, em matemática, é representado por $a_{lin,col}$, onde lin representa a linha e col representa a coluna. De maneira similar, um elemento qualquer de uma matriz é representado por:

```
1 # Representação computacional de um elemento de uma matriz
2 A[lin][col]
```

Conforme dito, interpretaremos cada linha da matriz como um vetor. Isso coincide com a representação computacional. Veja abaixo:

```
1 # Matriz
2 A
3 # Linha de uma matriz (resulta em um vetor)
4 A[lin]
5 # Elemento de uma matriz (resulta o valor de um elemento específico)
6 A[lin][col]
```

A seguir, vamos entender melhor como criar, modificar e adicionar dados a uma matriz.

7.2.2 Operações com matrizes

Na seção anterior, aprendemos a sintaxe básica das matrizes. Agora, aprenderemos a utilizar esses objetos. As principais operações com matrizes são:

- Criar uma matriz;
- Acessar/modificar elementos de uma matriz;
- Descobrir a dimensão de uma matriz.

Criando uma matriz: **Python** é uma linguagem muito flexível. Para simplificar, trataremos a criação de matrizes a partir de duas estratégias: Criando de maneira fixa ou criando a partir de entradas do usuário.

Para criar uma matriz de maneira fixa, existem dois caminhos. Primeiramente, é possível escrever todos os elementos previamente. Veja a seguir:

```
# Criando uma matriz
A = [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
# Matriz resultante:
# 1 2 3
# 4 5 6
# 7 8 9
```

Também é possível criar uma matriz usando uma rotina de código, a partir de um valor fixo ou dado pelo usuário com linhas e colunas. Veja a seguir:

```
# Criando matrizes com rotinas:
linhas = 3
colunas = 3
#Criando matriz de zeros
matriz = [ ] # uma lista vazia para representar a matriz
for lin in range(linhas):
    linha = [ ] # uma lista vazia para representar a linha
    for col in range(colunas):
        linha.append(0) # preenchendo valor da coluna
    matriz.append(linha) # colocando a linha na matriz
```

Na matriz anterior, tanto as dimensões quanto os valores poderiam ser obtidos através de entradas do usuário, com a função **input**. Veja a seguir:

```
# Criando uma matriz com entradas do usuário
linhas = int(input('Quantidade de linhas: '))
colunas = int(input('Quantidade de colunas: '))
matriz = [ ]
for lin in range(linhas):
    linha = [ ]
    for col in range(colunas):
        elemento = float(input(f'Digite o elemento {lin},{col}: '))
        linha.append(elemento)
    matriz.append(linha)
```

Acessando e modificando elementos em uma matriz: Os elementos de uma matriz podem ser acessados e modificados de maneira similar aos dos vetores. Para acessar um elemento, basta utilizar seus índices l e c entre chaves. Veja abaixo:

```
# Acessando o elemento [lin, col] da matriz A para diversos fins
val = A[lin][col]
print(A[lin][col])
result = 5 + A[lin][col]
```

Um elemento qualquer pode ser modificado de maneira idêntica à de uma variável, assim como nos vetores. Veja a seguir:

```
1 # Modificando o valor do elemento [i,j] da matriz A
2 A[i][j] = A[i][j] + 3
3 A[i][j] = A[i][j] * 2
4 A[i][j] = 5
```

Descobrir as dimensões das matrizes Descobrir as dimensões de uma matriz é um pouco diferente do que para os vetores, já que ela tem duas dimensões. O uso da função `len(A)` retorna a quantidade de linhas da matriz A. Para obter a quantidade de colunas, aplicar **len** a uma linha qualquer da própria matriz.

```
1 # Número de linhas
2 len(matriz)
3 #Número de colunas
4 len(matriz[0])
```

Observe que, se a matriz estiver vazia, a linha 0 não existe, portanto, `len(matriz[0])` resultaria em erro. Sendo assim, é prudente realizar este comando apenas quando a quantidade de linhas da matriz for diferente de $zero$, ou seja, `len(matriz) != 0` (em uma decisão).

7.2.3 Exercícios propostos – Matrizes

Exercício 4

Repetir a atividade do Exercício 1, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
```

Exemplo 2:

```
Digite o tempo final: 6
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
[4, 78.456]
[5, 122.5875]
[6, 176.526]
```

Exercício 5

Repetir a atividade do Exercício 2, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -5
Digite o valor final de x: 5

['x', 'f(x)']
[-5, 39]
[-4, 24]
[-3, 13]
[-2, 6]
[-1, 3]
[0, 4]
[1, 9]
[2, 18]
[3, 31]
[4, 48]
[5, 69]
```

Exemplo 2:

```
Digite o valor de "a": 4
Digite o valor de "b": 0
Digite o valor de "c": -4
Digite o valor inicial de x: -3
Digite o valor final de x: 6

['x', 'f(x)']
[-3, 32]
[-2, 12]
[-1, 0]
[0, -4]
[1, 0]
[2, 12]
[3, 32]
[4, 60]
[5, 96]
[6, 140]
```

Exercício 6

Fazer um programa que gere uma matriz quadrada de dimensão n, preenchida em todas as posições por um valor v. Ambos n e v são valores informados pelo usuário, como entradas. Imprimir o conteúdo da matriz ao final do programa.

Exemplos de execução:

Exemplo 1:

```
Digite a dimensão da matriz: 4
Digite o valor para preencher os campos: 2
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
```

Exemplo 2:

```
Digite a dimensão da matriz: 6
Digite o valor para preencher os campos: -3
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
```

Exercício 7

Implemente um programa que realize a entrada de dados de uma matriz. Primeiro o usuário fornece as dimensões da matriz, em seguida, fornece um valor textual para cada elemento. Após preenchida a matriz, o usuário fornece uma linha e coluna, e o programa imprime o valor correspondente a estes índices. Considere que os dados informados são sempre válidos.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de linhas: 2
Informe a quantidade de colunas: 3
Valor do elemento 0, 0: Sardinha
Valor do elemento 0, 1: Salame
Valor do elemento 0, 2: Picanha
Valor do elemento 1, 0: Coca-cola
Valor do elemento 1, 1: Guaraná
Valor do elemento 1, 2: Pipoca
Informe uma linha: 1
Informe uma coluna: 2
O valor armazenado em [1,2] é Pipoca
```

7.3 Construindo uma biblioteca de funções úteis

Podemos observar nos exemplos e exercícios sobre vetores e matrizes, apresentados anteriormente, que muitas tarefas serão sempre iguais. Alguns exemplos: criar um vetor ou matriz com valores padrão ou valores definidos pelo usuário e imprimir os valores de um vetor ou matriz. Podemos criar algumas funções para compor uma biblioteca padrão, e toda vez que precisarmos realizar uma destas tarefas, basta chamar a função. A seguir, apresentamos algumas funções úteis que nos ajudarão a criar programas mais rapidamente. Depois apresentamos exemplos de utilização da biblioteca criada.

7.3.1 Criando um vetor com valor padrão

Para esta função, consideraremos como argumentos de entrada a *quantidade de elementos* e o *valor padrão* a ser atribuído a todos eles. Basicamente, a função precisa criar um vetor vazio e, em seguida, repetir a operação de adicionar um novo elemento com o *valor padrão*, ao final, o vetor criado deve ser retornado. O código é apresentado a seguir.

```
1 def criarVetor(qtdElementos, valorPadrao):
2     vetor = [ ]
3     for i in range(qtdElementos):
4         vetor.append(valorPadrao)
5     return vetor
```

Dada a flexibilidade proporcionada pelo **Python**, não é necessário se preocupar com o tipo do *valor padrão*, basta passar o argumento com o valor desejado, independente de seu tipo.

7.3.2 Criando uma matriz com valor padrão

De forma similar, podemos criar uma matriz com um valor padrão, o código fica um pouco diferente, mas a lógica é praticamente a mesma, veja a seguir.

```
1 def criarMatriz(qtdLinhas, qtdColunas, valorPadrao):
2     matriz = [ ]
3     for lin in range(qtdLinhas):
4         linha = [ ]
5         for col in range(qtdColunas):
6             linha.append(valorPadrao)
7         matriz.append(linha)
8     return matriz
```

7.3.3 Criando um vetor com valores de uma string

Quando fazemos a entrada de dados de um vetor lendo o valor de cada elemento, são feitas várias entradas do usuário. Em termos práticos, é uma tarefa enfadonha para o usuário fazer várias entradas. Podemos oferecer a possibilidade do usuário fornecer todos os valores a partir de uma única entrada, estabelecendo para isso um padrão. Em nosso caso, separaremos os elementos com um caractere específico, a vírgula `","`. Neste caso, todos os elementos serão fornecidos em uma **string**, obtida em um único **input**, que precisará ser tratada para preencher o vetor. Apresentamos o código a seguir e logo depois explicamos mais detalhadamente seu funcionamento.

```
1  def preencherVetor(valores, tipo):
2      vetor = [ ]
3      valores = valores.split(',')
4      for i in range(len(valores)):
5          if tipo == 'int':
6              valor = int(valores[i].strip())
7          elif tipo == 'float':
8              valor = float(valores[i].strip())
9          else:
10             valor = valores[i].strip()
11         vetor.append(valor)
12     return vetor
```

Nossa função criará um novo vetor contendo os elementos passados na **string** de entrada `valores`, convertendo os valores conforme o `tipo`, passado como segundo argumento.

Como todos os valores são passados em uma única **string**, separados pela vírgula, é necessário "dividi-los" de acordo com o caractere `","`. Isto é feito com o uso da função **split**. Porém, o uso desta função é diferente daquilo que estamos acostumados, ela funciona sobre um valor textual, com o uso de um ponto separando o **valor** da **função**, que receberá como argumento o texto utilizado para fazer a separação da **string**: `valores.split(',')`, onde a **string** armazenada na variável `valores` será dividida, tendo como referência a vírgula, resultando em um vetor com os valores divididos. Por exemplo, para a **string** `valores = "1, 2, 6, 5"`, o resultado de `valores.split(',')` será o vetor `[ "1", " 2", " 6", " 5", ]`. Observe que, antes da linha 3, o valor da variável `valores` será uma **string**, e depois da linha 3 ela passará a armazenar um vetor de **strings**, resultado do **split**.

Em seguida, o laço for vai processar cada valor **string** contido no vetor `valores`. De acordo com o `tipo` informado na chamada da função, será feita uma conversão para **int** ou **float**, ou então será mantido o tipo **string**. Agora, usamos a função **strip**, também aplicada a uma **string** de forma similar à **split**, mas o resultado é a elimininação dos espaços à direita e à esquerda da **string**, por exemplo, para o segundo elemento o valor `" 2"` resultará em `"2"`.

Concluindo o laço, o valor processado é inserido no vetor que será retornado ao final.

7.3.4 Criando uma matriz com valores de uma string

De forma similar à função anterior, criaremos outra função, agora para uma matriz. Neste caso, separaremos os elementos de cada coluna com a vírgula (`","`) e cada linha com um ponto-e-vírgula (`";"`). Neste caso, a lógica é muito parecida, mas primeiro deveremos dividir as linhas e, para cada linha, teremos que dividir as colunas. Apresentamos o código a seguir.

```
1  def preencherMatriz(valores, tipo):
2      matriz = [ ]
3      linhas = valores.split(';')
4      for lin in range(len(linhas)):
5          colunas = linhas[lin].split(',')
6          vetor = [ ]
7          for col in range(len(colunas)):
8              if tipo == 'int':
9                  valor = int(colunas[col].strip())
10             elif tipo == 'float':
11                 valor = float(colunas[col].strip())
12             else:
13                 valor = colunas[col].strip()
14             vetor.append(valor)
15         matriz.append(vetor)
16     return matriz
```

7.3.5 Imprimir um vetor na tela

Outra tarefa que pode ser comum ao manipular vetores é imprimir seu conteúdo na tela. Estabeleceremos como padrão de impressão o mesmo usado para a atribuição, com o uso de colchetes para delimitar e vírgula para separar os elementos, por exemplo: `"[ 1, 2, 3 ]"`. Veja o código da função a seguir.

```
1  def imprimeVetor(vetor):
2      print('[', end='')
3      if len(vetor) > 0:
4          print(f' {vetor[0]}', end='')
5          for i in range(1, len(vetor)):
6              print(f', {vetor[i]}', end='')
7      print(' ]', end='')
```

O resultado desta função é apenas a impressão dos valores contidos no vetor na tela, portanto, nenhum valor é retornado por ela, assim, não utilizamos a cláusula `return`. Outra questão importante é que, se o vetor estiver vazio, apenas `"[ ]"` deverá ser impresso na tela, isso é garantido pela decisão da linha 3. Caso o vetor não esteja vazio, o primeiro valor é impresso sem o uso do separador `","` (fora do laço **for**), mas do segundo em diante inserimos o separador (dentro do laço). Todos os **print**'s usados na função utilizam um segundo argumento, `end=''`, indicando que não será inserida uma quebra de linha (padrão para o **print**) após imprimir a mensagem definida pelo primeiro argumento.

7.3.6 Imprimir uma matriz na tela

De forma similar a impressão do vetor, implementaremos duas funções para imprimir o conteúdo de uma matriz na tela. Em uma primeira função, usaremos o mesmo padrão utilizado na atribuição, imprimindo tudo em uma linha, por exemplo: `"[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]"`. Por isso, chamaremos esta função de `imprimeMatrizEmLinha`, e apresentamos o código a seguir.

```
1 def imprimeMatrizEmLinha(matriz):
2     print('[ ', end='')
3     if len(matriz) > 0:
4         imprimeVetor(matriz[0])
5         for lin in range(1, len(matriz)):
6             print(', ', end='')
7             imprimeVetor(matriz[lin])
8     print(' ]', end='')
```

Um ponto bem interessante aqui é que utilizamos a função de impressão de um vetor para imprimir a matriz, uma vez que cada linha da matriz pode ser considerada como um vetor, que segue o mesmo padrão de impressão de vetor. A lógica é a mesma, mas aqui ao invés de imprimir um único elemento por vez, imprimimos uma linha por vez, chamando a função `imprimeVetor`.

Em uma segunda função, faremos a impressão da matriz em formato de tabular, separando as linhas. Esta função se chamará simplesmente `imprimeMatriz`, e o código é apresentado a seguir.

```
1 def imprimeMatriz(matriz):
2     if len(matriz) > 0:
3         imprimeVetor(matriz[0])
4         print('')
5         for lin in range(1, len(matriz)):
6             imprimeVetor(matriz[lin])
7             print('')
```

A diferença para a função anterior é que não delimitamos toda a matriz por `"[ ]"` e a cada linha da matriz nós imprimimos uma quebra de linha na tela com a instrução `print('')`, que não imprime nada na tela, mas que por padrão quebra uma linha após a sua execução, devido ao funcionamento da função **print**.

7.3.7 Criando e usando a biblioteca

Criar a biblioteca consiste em salvar todas as definições de funções em um arquivo ".py", como por exemplo "biblioteca.py". E, para usa-la, você pode importar o arquivo de biblioteca em seu programa, um outro arquivo ".py", como por exemplo "teste.py". No entanto, esta forma simples de utilização da biblioteca exige que os dois arquivos estejam juntos, na mesma pasta do computador. Porém, isso já é suficiente para que possamos resolver os exercícios que serão propostos, por isso, não aprofundaremos este assunto.

No programa a seguir testamos nossas funções, salvas no arquivo "biblioteca.py". Nele, importamos o arquivo da biblioteca, com a instrução `import biblioteca as bib`, definindo um **apelido** para a biblioteca: **bib**. Isto significa que, quando formos usar uma função da biblioteca, podemos usar `"bib."` no lugar de `"biblioteca."`. Por exemplo, para criar o primeiro vetor, contendo 5 elementos com valor 3, usamos `bib.criarVetor(5, 3)` e em seguida, para imprimir seus valores na tela, usamos `bib.imprimeVetor(v)`. Para deixar a saída mais agradável de se ver, utilizamos **print** com mensagens e quebras de linhas estratégicas (`"\n"`). O resultado da execução (*saída*) deste programa você pode ver logo em seguida.

```
1  import biblioteca as bib
2
3  v = bib.criarVetor(5, 3)
4  print('Conteúdo do vetor:')
5  bib.imprimeVetor(v)
6
7  m = bib.criarMatriz(2, 3, 0.5)
8  print('\n\nConteúdo da matriz (em linha):')
9  bib.imprimeMatrizEmLinha(m)
10 print('\n\nConteúdo da matriz (tabular):')
11 bib.imprimeMatriz(m)
12
13 v = bib.preencherVetor('1.0, 1.2, 1.4', 'float')
14 print('\nConteúdo do vetor:')
15 bib.imprimeVetor(v)
16
17 m = bib.preencherMatriz('1.0, 1.2, 1.4 ; 2.0, 2.2, 2.4', 'float')
18 print('\n\nConteúdo da matriz (em linha):')
19 bib.imprimeMatrizEmLinha(m)
20 print('\n\nConteúdo da matriz (tabular):')
21 bib.imprimeMatriz(m)
```

Saída:

```
Conteúdo do vetor:
[ 3, 3, 3, 3, 3 ]

Conteúdo da matriz (em linha):
[ [ 0.5, 0.5, 0.5 ], [ 0.5, 0.5, 0.5 ] ]

Conteúdo da matriz (tabular):
[ 0.5, 0.5, 0.5 ]
[ 0.5, 0.5, 0.5 ]

Conteúdo do vetor:
[ 1.0, 1.2, 1.4 ]

Conteúdo da matriz (em linha):
[ [ 1.0, 1.2, 1.4 ], [ 2.0, 2.2, 2.4 ] ]

Conteúdo da matriz (tabular):
[ 1.0, 1.2, 1.4 ]
[ 2.0, 2.2, 2.4 ]
```

7.4 Exercícios propostos

Resolva os exercícios a seguir fazendo uso da biblioteca de funções úteis.

Exercício 8

Implemente um programa para contabilizar a pontuação em uma Gincana, seguindo o algoritmo:

1. Pergunte a quantidade de equipes.
2. Crie um vetor, cada elemento representa uma equipe e possui pontuação inicial $zero$.
3. Pergunte o índice da equipe que pontuou.
4. Enquanto o índice for válido:
   (a) Pergunte a quantidade de pontos.
   (b) Atualize, de forma cumulativa, a pontuação da equipe no vetor.
   (c) Pergunte o índice da equipe que pontuou.
5. Imprima o conteúdo do vetor.

Tanto os índices quanto a pontuação devem ser valores inteiros.
Exemplos de execução:

Exemplo 1:

```
Quantidade de equipes: 5
Índice da equipe: 2
Quantidade de pontos: 10
Índice da equipe: 1
Quantidade de pontos: 5
Índice da equipe: 0
Quantidade de pontos: 30
Índice da equipe: 2
Quantidade de pontos: 25
Índice da equipe: -1
Pontuação das equipes:
[ 30, 5, 35, 0, 0 ]
```

Exemplo 2:

```
Quantidade de equipes: 6
Índice da equipe: 5
Quantidade de pontos: 4
Índice da equipe: 1
Quantidade de pontos: 3
Índice da equipe: 1
Quantidade de pontos: 4
Índice da equipe: 6
Pontuação das equipes:
[ 0, 7, 0, 0, 0, 4 ]
```

Exercício 9

Implemente um programa para ajudar na leitura de consumo de energia feito pela CEMIG nos imóveis de uma cidade. O registro é feito pelo mapeamento de ruas e imóveis e armazenado em uma matriz, as linhas representam as ruas e as colunas representam os números dos imóveis. Primeiro o programa solicita que seja fornecida como entrada uma **string** que contenha os valores iniciais, preenchendo uma matriz com estes valores. Em seguida, o programa pergunta quantas novas leituras individuais serão fornecidas. Para cada leitura individual, o programa pergunta o índice da rua (linha), o índice do imóvel (coluna), e o valor lido pelo funcionário. O programa deve então fornecer a diferença da última leitura referente ao imóvel do valor lido pelo funcionário, e em seguida atualizar a matriz com o valor lido para o imóvel.

Tanto os índices quanto as leituras devem ser valores inteiros. Não é necessário validar dados de entrada, considere que serão sempre válidos.

Exemplo de execução:

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 6
 - Diferença: 2
Leitura 2:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 10
 - Diferença: 4
Leitura 3:
 - Índice da rua: 2
 - Índice do imóvel: 2
 - Valor lido: 5
 - Diferença: 3
Leitura 4:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 5
 - Diferença: 0
Leitura 5:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 7
 - Diferença: 2
```

Exercício 10

Implemente um programa similar ao anterior, Exercício 9. Mas, agora a entrada de cada leitura deve ser feita por um único **input**, armazenando os valores em um vetor, onde, índice 0 representa a rua, índice 1 representa o imóvel e índice 2 representa o valor lido. O restante do programa funciona da mesma maneira, gerando o mesmo resultado, conforme o exemplo de execução a seguir.

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1: 0, 2, 6
 - Diferença: 2
Leitura 2: 0, 2, 10
 - Diferença: 4
Leitura 3: 2, 2, 5
 - Diferença: 3
Leitura 4: 0, 0, 5
 - Diferença: 0
Leitura 5: 0, 0, 7
 - Diferença: 2
```

MÓDULO

8

Estruturas Heterogêneas

No módulo anterior trabalhamos com vetores e matrizes, que se caracterizam pela possibilidade de armazenar diversos valores de um mesmo tipo de dados (estrutura de dados *homogênea*). Quando uma determinada estrutura de dados pode ser composta por diversos valores, sendo estes de tipos de dados diferentes, temos uma **estrutura de dados heterogênea**.

Por exemplo, para representar o cadastro de empregados, poderia ser implementada uma estrutura de dados composta pelos seguintes campos: *nome*, *idade*, *endereço*, *telefone*, *salário* e *cargo*. Nesse caso os campos *idade*, *telefone* e *salário* podem ser do tipo **numérico** enquanto que os demais campos podem ser do tipo **textual**.

8.1 Definição

Basicamente, uma **estrutura de dados heterogênea** pode ser definida como uma coleção de uma ou mais **variáveis relacionadas** (**campos**), onde cada variável pode ser de um tipo distinto. Esta estrutura heterogênea também é conhecida como **registro**. Então, um **registro** contém **campos**, estes, por sua vez, possuem valores associados, conforme a representação do exemplo dado anteriormente:

$$empregado \begin{cases} nome & = & < valor\ textual > \\ idade & = & < valor\ numérico > \\ endereco & = & < valor\ textual > \\ telefone & = & < valor\ numérico > \\ salario & = & < valor\ numérico > \\ cargo & = & < valor\ textual > \end{cases}$$

8.2 Sintaxe básica

Assim como ocorreu com vetores e matrizes, escolheremos uma estrutura com sintaxe simples, mas que permita representar o conceito de estruturas heterogêneas através de **registros** na linguagem **Python** na forma como foi definida conceitualmente na seção anterior. Esta estrutura em **Python** se chama **dicionário**, e permite manipular pares *chave-valor*, conforme o exemplo a seguir.

Exemplo 1:

O código a seguir cria uma variável chamada `empregado` que será utilizada para armazenar os dados de um empregado com os **campos** listados anteriormente. Inicialmente, todos os campos do tipo textual foram definidos com valor de uma **string** vazia '' e os campos numéricos com o valor $zero$. Porém, no momento de criação os valores podem ser definidos conforme sua necessidade. Ou seja, se você já possui nome, idade, etc, pode fornecê-los no momento da criação da variável.

```
1 empregado = {
2     "nome": '',
3     "idade": 0,
4     "endereco": '',
5     "telefone": 0,
6     "salario": 0,
7     "cargo": ''
8 }
```

Pela representação anterior, o registro `empregado` é:

$$empregado \begin{cases} nome & = & '' \\ idade & = & 0 \\ endereco & = & '' \\ telefone & = & 0 \\ salario & = & 0 \\ cargo & = & '' \end{cases}$$

É importante destacar que, após a criação de um registro, `empregado` é uma variável que pode ser manipulada em qualquer parte do programa. Para fazer o acesso ao conteúdo de um campo do registro, deve-se usar o nome do registro seguido por colchetes "[]" e dentro dos colchetes o nome do campo. Perceba que há uma grande semelhança com vetores e matrizes, porém, ao invés de índices, temos a **chave**, que no caso corresponde ao nome do campo em questão. Em **Python**, a chave pode ser representada por outros tipos de dados, não apenas **string**, como faremos uso aqui. No exemplo a seguir mostramos como modificar o registro criado anteriormente, para definir todos os campos de forma individual.

```
1 empregado["nome"] = 'Joaquim'
2 empregado["idade"] = 25
3 empregado["endereco"] = 'Rua A'
4 empregado["telefone"] = 53535353
5 empregado["salario"] = 1212.12
6 empregado["cargo"] = 'Vendedor'
```

Python diferencia letras maiúsculas de minúsculas, se você tentar acessar `empregado["Nome"]` ou `empregado["NOME"]`, por exemplo, ocorrerá um erro, pois estas **chaves** não existem no registro.

Após as alterações feitas anteriormente, o registro pode ser representado desta maneira:

$$empregado \begin{cases} \textit{nome} & = & `Joaquim' \\ \textit{idade} & = & 25 \\ \textit{endereco} & = & `RuaA' \\ \textit{telefone} & = & 53535353 \\ \textit{salario} & = & 1212.12 \\ \textit{cargo} & = & `Vendedor' \end{cases}$$

Exemplo 2:

Neste exemplo criamos um registro para representar um livro definindo os dados na atribuição. Depois fazemos acesso de **leitura**, para imprimir seus valores na tela. Como o acesso ao registro estará dentro do **print**, para diferenciar as aspas da mensagem das aspas da chave do registro, usamos aspas simples para delimitar a mensagem e as aspas duplas para delimitar a chave do registro.

```
1  livro = {
2      "titulo": 'Algoritmos e lógica de programação',
3      "autor": 'Marco Antonio Furlan de Souza',
4      "ano": 2005,
5      "editora": 'Cenage Learning',
6  }
7
8  print('Dados do livro: ')
9  print(f'- Título: {livro["titulo"]}')
10 print(f'- Autor: {livro["autor"]}')
11 print(f'- Ano: {livro["ano"]}')
12 print(f'- Editora: {livro["editora"]}')
```

Exemplo 3:

Agora nós apresentamos um programa que modifica o registro após a sua criação, mas agora inserindo um novo campo. Uma vez que o dicionário existe, podemos inserir novos campos simplesmente com uma atribuição usando a **chave** do novo campo.

```
1  livro = {
2      "titulo": 'Algoritmos e lógica de programação',
3      "autor": 'Marco Antonio Furlan de Souza',
4      "ano": 2005,
5      "editora": 'Cenage Learning',
6  }
7
8  livro["isbn"] = '978-8575227183'
9
10 print('Dados do livro: ')
11 print(f'- Título: {livro["titulo"]}')
12 print(f'- Autor: {livro["autor"]}')
13 print(f'- Ano: {livro["ano"]}')
14 print(f'- Editora: {livro["editora"]}')
15 print(f'- ISBN: {livro["isbn"]}')
```

8.3 Vetores e matrizes de registros

Em programação de computadores, podemos criar uma estrutura de dados que combina registros e vetores ou registros e matrizes a fim de resolver problemas mais complexos. Essa situação acontece, quando temos que armazenar, em nosso programa, vários registros que possuem a mesma estrutura. Por exemplo, armazenar dados de diversos alunos. Nesse caso, os dados de cada aluno será um registro, assim, temos vários registros diferentes com a mesma estrutura.

Exemplo 4:

Neste exemplo criaremos um programa que armazena o nome, a nota e a frequência dos alunos de uma turma em um único vetor através do registro aluno. As informações são fornecidas pelo usuário. Em seguida, ele imprime os alunos aprovados, que possuem nota maior ou igual a 6 e frequência maior ou igual a 75.

```
1  N = int(input('Quantidade de alunos: '))
2  alunos = [ ]
3  for a in range(N):
4      aluno = { }
5      print(f'Dados do aluno {a+1}:')
6      aluno["nome"] = input('- Nome: ')
7      aluno["nota"] = float(input('- Nota: '))
8      aluno["freq"] = int(input('- Frequência: '))
9      alunos.append(aluno)
10 print('\nAlunos aprovados:')
11 for a in range(N):
12     if alunos[a]['nota'] >= 6 and alunos[a]['freq'] >= 75:
13         print(f'- {alunos[a]["nome"]}')
```

Observe que inicialmente criamos um vetor vazio chamado `alunos`. Depois temos um laço **for** para fazer a entrada de dados de cada aluno, inserindo-os individualmente no vetor. Para cada aluno, a primeira ação é criar um registro vazio, sem nenhum campo, de forma similar à criação do vetor vazio, a diferença é o uso das chaves: `aluno = { }`. Estabelecemos aqui um padrão, o vetor, que representa vários alunos, está no plural `alunos`, e o registro que representa um único aluno, está no singular `aluno`. em seguida fazemos três entradas de dados, uma para cada campo, criando novos campos para o registro vazio criado no início do laço. Por fim, o registro `aluno` é adicionado ao vetor `alunos`. Com o término da execução do laço temos todos os dados de alunos armazenados no vetor, passamos então a imprimir os alunos aprovados. Um novo laço é iniciado para avaliar todos os alunos, imprimindo apenas os alunos aprovados. Para isso utilizamos uma decisão, verificando se o aluno `a` (variável contadora do laço, que determina os índices válidos do vetor), possui nota maior do que 6 e frequência maior do que 75. Observe que são duas condições ligadas pelo operador lógico `and` e que, para obter a nota do aluno precisamos acessar o índice do vetor e o nome do campo (`alunos[a]["nota"]`) e para obter a frequência do aluno precisamos acessar o índice do vetor e o nome do campo (`alunos[a]["freq"]`). Construindo a expressão lógica do **if** com estes componentes e as operações relacionais necessárias para atender à condição de aprovação. Caso a condição resulte em **verdadeiro**, deve-se imprimir o nome do aluno, caso contrário, não tem nada a ser feito. De forma similar, para obter o nome do aluno precisamos acessar o índice do vetor e o nome do campo (`alunos[a]["nome"]`).

A seguir um exemplo de execução do programa:

```
Quantidade de alunos: 3
Dados do aluno 1:
- Nome: João
- Nota: 6.5
- Frequência: 65
Dados do aluno 2:
- Nome: Maria
- Nota: 10
- Frequência: 95
Dados do aluno 3:
- Nome: José
- Nota: 3.8
- Frequência: 100

Alunos aprovados:
- Maria
```

O uso de registro em matrizes acontece de forma bastante similar, a diferença é que para matriz precisamos acessar dois índices antes de acessar o nome do campo. Como exemplo, imagine que você agora esteja representando o desempenho dos alunos em variadas disciplinas, para cada disciplina uma nota e uma frequência. Você pode representar os alunos nas linhas e as disciplinas nas colunas. Para obter a nota do aluno de índice de linha `lin` para a disciplina de índice de coluna `col`, você poderia acessar: `alunos[lin][col]["nota"]`. O termo `alunos[lin][col]` representa o acesso a um elemento específico da matriz, e o termo `["nota"]` representa o acesso ao campo nota deste elemento da matriz.

8.4 Registros como argumento ou retorno de função

Como os registros são armazenados em variáveis, assim como vetores e matrizes, eles podem ser utilizados como argumentos ou valor de retorno de uma função.

Exemplo 5:

Neste exemplo criamos uma função que recebe um registro que representa um **retângulo** com os campos **base** e **altura** e retorna um registro de medidas contendo o **perímetro** e a **área** do retângulo.

```
1  def calcRetangulo(retangulo):
2      resultado = { }
3      resultado["perímetro"] = 2 * (retangulo["base"] + retangulo["
           altura"])
4      resultado["área"] = retangulo["base"] * retangulo["altura"]
5      return resultado
6
7  ret = { "base": 10, "altura": 10 }
8  med = calcRetangulo(ret)
9  print(f'O perímetro do retângulo é {med["perímetro"]}')
10 print(f'A área do retângulo é {med["área"]}')
```

Observe que, como a chave do registro é do tipo textual, pode-se utilizar caracteres acentuados sem nenhum problema. Porém, é aconselhável que se estabeleça padrões nas definições de nomes de campos, uma boa prática é optar por não usar caracteres acentuados ou especiais. **Mais importante aqui é observar que os nomes dos campos usados dentro e fora da função devem ser exatamente iguais, diferentemente dos nomes das variáveis**.

Veja o resultado da execução do programa anterior:

```
O perímetro do retângulo é 40
A área do retângulo é 100
```

8.5 Exercícios propostos

Exercício 1

Implemente uma versão do Exemplo 4 para armazenar os dados de alunos em diferentes disciplinas. Sendo assim, deve-se utilizar uma matriz para armazenar os dados, em cada elemento da matriz será armazenada a nota e a frequência. As linhas representam alunos e as colunas representam disciplinas, que serão referenciadas pelos seus respectivos índices. Primeiramente o programa pergunta a quantidade de alunos e disciplinas, em seguida pergunta nota e frequência para cada elemento da matriz. Depois disso, para cada aluno, o programa deve identificar a disciplina com a maior nota, imprimindo a nota com precisão de uma casa decimal e a frequência nesta disciplina. Notas são valores reais e frequência inteiros.

Exemplos de execução:

Exemplo 1:

```
Quantidade de alunos: 2
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 4
  - Frequência: 4
 - Dados da disciplina 2:
  - Nota: 3
  - Frequência: 3

Dados do aluno 2:
 - Dados da disciplina 1:
  - Nota: 2
  - Frequência: 2
 - Dados da disciplina 2:
  - Nota: 1
  - Frequência: 1

Maiores notas e frequências:
 - Aluno 1: nota = 4.0; frequência = 4
 - Aluno 2: nota = 2.0; frequência = 2
```

Exemplo 2:

```
Quantidade de alunos: 3
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 1
  - Frequência: 10
 - Dados da disciplina 2:
  - Nota: 2
  - Frequência: 20

Dados do aluno 2:
 - Dados da disciplina 1:
  - Nota: 3
  - Frequência: 30
 - Dados da disciplina 2:
  - Nota: 4
  - Frequência: 40

Dados do aluno 3:
 - Dados da disciplina 1:
  - Nota: 5
  - Frequência: 50
 - Dados da disciplina 2:
  - Nota: 6
  - Frequência: 60

Maiores notas:
 - Aluno 1: nota = 2.0; frequência = 20
 - Aluno 2: nota = 4.0; frequência = 40
 - Aluno 3: nota = 6.0; frequência = 60
```

Exercício 2

Implemente um programa que receba informações sobre retas, armazenando-as em um vetor. Primeiramente pergunta-se quantas retas serão informadas, e em seguida pede-se as coordenadas dos dois pontos (**A** e **B**) que definem cada reta. Uma reta deve ser representada por um registro que contenha dois campos, representando os pontos. Cada ponto é um registro que possui as coordenadas **X** e **Y** como campos. Depois que o vetor estiver totalmente preenchido, você deverá calcular e imprimir o Comprimento de cada reta armazenada no vetor. O Comprimento é obtido através de uma função, definida em seu programa, que recebe a reta como argumento e retorna o comprimento: $C = \sqrt{(B_X - A_X)^2 + (B_Y - A_Y)^2}$. As coordenadas **X** e **Y** são números reais, e a saída das medidas deve possuir precisão de 2 casas decimais. **Dicas:** (1) para obter o ponto **B** de uma **reta**, você deve usar: `reta["B"]`; (2) para obter a coordenada **X** do ponto **B** desta mesma **reta**, você deve usar: `reta["B"]["X"]`, uma vez que temos aqui um registro "dentro" de outro registro.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de retas: 1

Reta 1:
 - Coordenada X de A: 1
 - Coordenada Y de A: 4
 - Coordenada X de B: 1
 - Coordenada Y de B: 1

Medidas das retas:
 - Reta 1: 3.00
```

Exemplo 2:

```
Informe a quantidade de retas: 2

Reta 1:
 - Coordenada X de A: 3.5
 - Coordenada Y de A: 2.1
 - Coordenada X de B: 6
 - Coordenada Y de B: 8.6

Reta 2:
 - Coordenada X de A: -3
 - Coordenada Y de A: 0.9
 - Coordenada X de B: 5.8
 - Coordenada Y de B: -9.1

Medidas das retas:
 - Reta 1: 6.96
 - Reta 2: 13.32
```

MÓDULO

Soluções dos Exercícios Propostos

9.1 Exercícios do Módulo 1: Introdução

Exercício 1

Dada a equação que mostra a conversão de Fahrenheit para Celsius:

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

Crie um fluxograma que indique as etapas de um algoritmo que representa essa transformação. Veja a sequência na lista abaixo:

1. Ler um valor em Fahrenheit;
2. Subtrair 32 no valor;
3. Multiplicar por 5 e dividir por 9;
4. Imprimir o valor de saída;

Solução em fluxograma: Figura 32:

Exercício 2

Faça a lista de instruções e o fluxograma para calcular a associação de dois resistores em série. As entradas são os valores dos resistores em ohms.

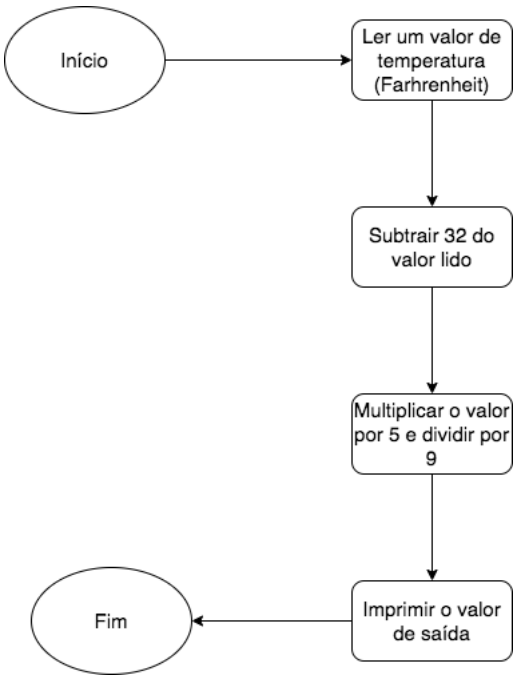

Figura 32: Fluxograma de solução do exercício 1.

Fórmula:

$$R_e = R_1 + R_2$$

Solução em fluxograma: Figura 33:

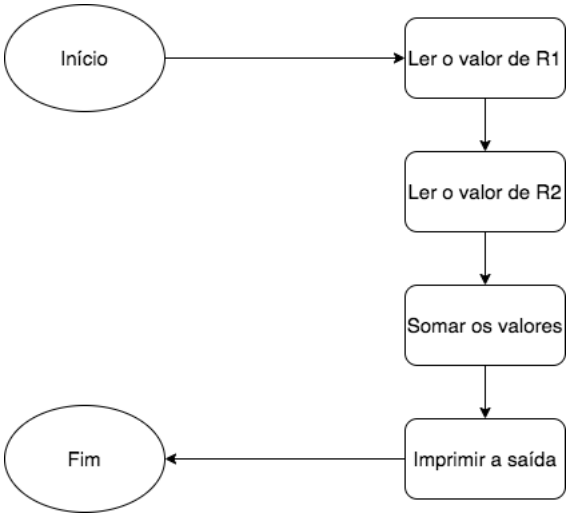

Figura 33: Fluxograma de solução do exercício 2.

Exercício 3

Faça a lista de instruções e o fluxograma para calcular a associação de dois resistores em paralelo. As entradas são os valores dos resistores em ohms.

Fórmula:

$$R_e = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

Solução em fluxograma: Figura 34:

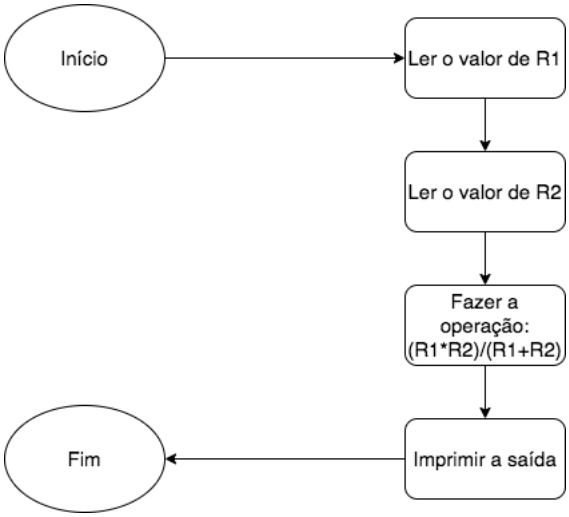

Figura 34: Fluxograma de solução do exercício 3.

Exercício 4

Apresente o fluxograma para calcular a aceleração de um corpo sendo puxado por uma mola, considerando como entrada a constante elástica da mola (k), a massa do corpo (m) e a extensão a partir da posição inicial (x).

Fórmulas:

$$F = k \cdot x$$

$$a = \frac{F}{m}$$

Solução em fluxograma: Figura 35:

Exercício 5

Dado um objeto em queda livre, calcular a velocidade em um instante de tempo e a distância total percorrida desde o início da queda, tendo como a entrada do usuário o instante de tempo t.

Fórmulas:

$$v = g \cdot t$$

$$s = \frac{g \cdot t^2}{2}$$

$$g = 9,807$$

Solução em fluxograma: Figura 36:

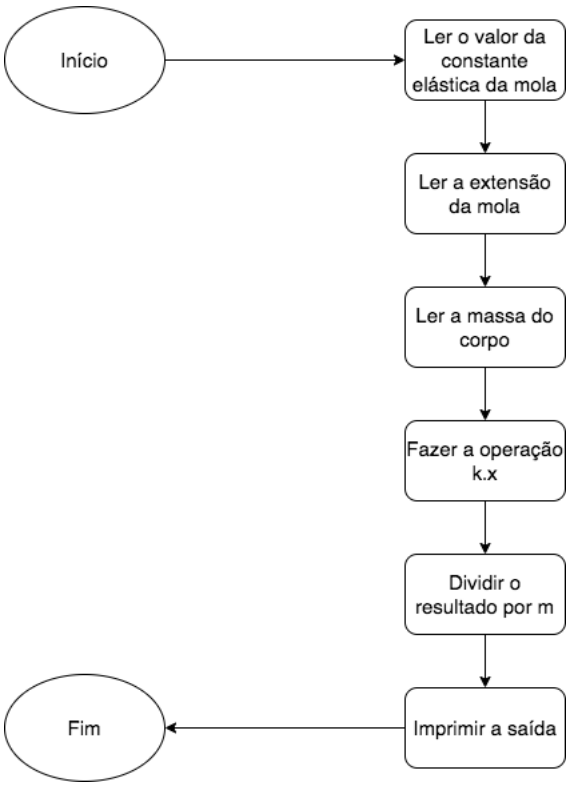

Figura 35: Fluxograma de solução do exercício 4.

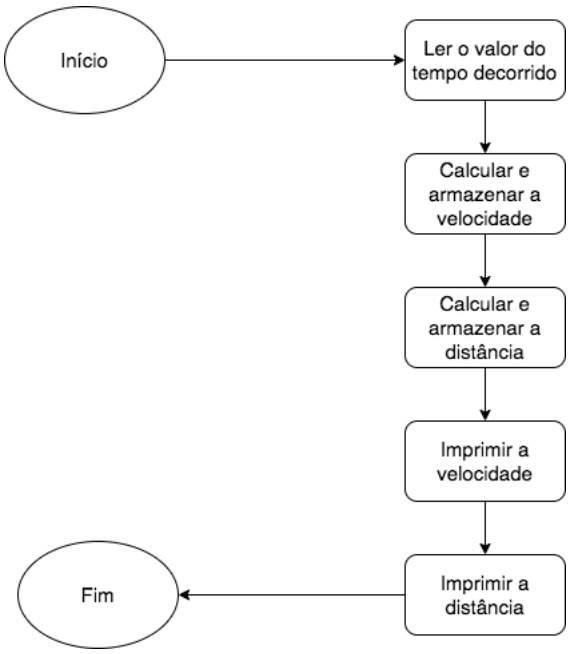

Figura 36: Fluxograma de solução do exercício 5.

9.2 Exercícios do Módulo 2: Variáveis e Expressões

Exercício 1

Qual é o valor impresso ao final da seguinte sequência de comandos?

```
1 dia = "quinta-feira"
2 dia = 32.5
3 dia = 19
4 print(dia)
```

Resposta: 19, pois este corresponde ao último valor atribuído à variável durante a execução do programa até o comando **print**.

Exercício 2

O seguinte nome é válido para uma variável em **Python**? Justifique sua resposta.

```
1 Uma_boa_nota_é_A+ = 9
```

Resposta: Não, pois a linguagem não permite o uso do símbolo "+" na declaração do nome da variável.

Exercício 3

Um prêmio em dinheiro será dividido entre três ganhadores de um concurso.

- O primeiro ganhador receberá 46% do valor total do prêmio
- O segundo ganhador receberá 32% do valor total do prêmio
- O terceiro receberá o restante

Escreva um programa que defina uma variável com o valor **780000** para o prêmio e, em seguida, calcule e imprima a quantia que deve ser distribuída para cada um dos ganhadores.

Exemplos de execução:

Exemplo 1:

```
O primeiro ganhador deve receber R$:
358800.0
O segundo ganhador deve receber R$:
249600.0
O terceiro ganhador deve receber R$:
171600.0
```

Solução:

```
1  premio = 780000
2  primeiro_ganhador = premio * 46 / 100
3  segundo_ganhador = premio * 32 / 100
4  terceiro_ganhador = premio - primeiro_ganhador - segundo_ganhador
5  print('O primeiro ganhador deve receber R$')
6  print(primeiro_ganhador)
7  print('O segundo ganhador deve receber R$')
8  print(segundo_ganhador)
9  print('O terceiro ganhador deve receber R$')
10 print(terceiro_ganhador)
```

Exercício 4

Faça um programa que defina a largura e a altura de uma parede em 2.8 e 5.2 metros em duas variáveis. Posteriormente, calcule a sua área e a quantidade de tinta necessária para pintá-la, sabendo que cada litro de tinta pinta uma área de 2 metros quadrados.

Exemplos de execução:

Exemplo 1:

```
Área da parede em metros quadrados:
14.559999999999999
Litros de tinta necessários para pintar essa parede:
7.279999999999999
```

Solução:

```
1 altura = 2.8
2 largura = 5.2
3 area = altura * largura
4 quantidade_tinta = area / 2
5 print('Área da parede em metros quadrados')
6 print(area)
7 print('Litros de tinta necessários para pintar essa parede')
8 print(quantidade_tinta)
```

Exercício 5

Escreva um programa que defina, em variáveis, a quantidade de Km percorridos por um carro alugado (com valor 879) e a quantidade de dias pelos quais ele foi alugado (com valor 6). Calcule o preço a pagar, sabendo que o carro custa R$ 60,00 por dia e R$ 0,15 por Km rodado.

Exemplos de execução:

Exemplo 1:

```
O cliente deve pagar R$:
491.85
```

Solução:

```
km_percorridos = 879
numero_diarias = 6
custo_km_percorridos = km_percorridos * 0.15
custo_diarias = numero_diarias * 60
custo_total = custo_km_percorridos + custo_diarias
print('O cliente deve pagar R$')
print(custo_total)
```

Exercício 6

Faça um programa que defina uma variável com área de um quadrado no valor de 256 e calcule o valor de cada lado do quadrado.

Exemplos de execução:

Exemplo 1:

```
Cada lado tem comprimento igual à:
16.0
```

Solução:

```
import math
area_quadrado = 256
comprimento_quadrado = math.sqrt(area_quadrado)
print('Cada lado tem comprimento igual à:')
print(comprimento_quadrado)
```

Exercício 7

Faça um programa que defina uma variável com o raio de uma circunferência no valor de 1.2 e calcule sua área. Utilize a fórmula:

$$\text{area} = \pi * (\text{raio}^2)$$

Utilize a constante π definida na linguagem.

Ao final, o seu programa deve apresentar o valor da área da circunferência calculada.

Exemplos de execução:

Exemplo 1:

```
A área dessa circunferência é:
4.523893421169302
```

Solução:

```
import math
raio_circunferencia = 1.2
area_circunferencia = math.pi * raio_circunferencia ** 2
print('A área dessa circunferência é:')
print(area_circunferencia)
```

9.3 Exercícios do Módulo 3: Entrada e Saída

Exercício 1

Elabore um programa que leia o valor do raio de um círculo qualquer (número inteiro) e calcule sua respectiva área. Ao final imprima como saída o valor da área do círculo (com duas casas decimais).

Fórmula:

$$rea = \pi.r^2$$

onde:

- π : constante **Pi**;
- r: raio do círculo.

Exemplos de execução:

Exemplo 1:

```
Digite o valor do raio: 5
A área do círculo é: 78.54
```

Exemplo 2:

```
Digite o valor do raio: 9
A área do círculo é: 254.47
```

Exemplo 3:

```
Digite o valor do raio: 15
A área do círculo é: 706.86
```

Solução:

```
1 import math
2 # Entrada
3 raio = int(input('Digite o valor do raio: '))
4 # Processamento
5 area = math.pi * raio * raio
6 # Saída
7 print(f'A área do círculo é: {area:.2f}')
```

Exercício 2

Implemente um programa que leia o valor do raio **r** de uma esfera (número real) e calcule o volume da esfera. No final imprima como saída o valor calculado (com duas casas decimais).

Fórmula:

$$Volume = \frac{4\pi r^3}{3}$$

onde:

- π : constante **Pi**;
- r: raio do círculo.

Exemplos de execução:

Exemplo 1:

```
Digite o valor do raio: 5
O volume da esfera é: 523.60
```

Exemplo 2:

```
Digite o valor do raio: 10.5
O volume da esfera é: 4849.05
```

Exemplo 3:

```
Digite o valor do raio: 5.75
O volume da esfera é: 796.33
```

Solução:

```
1 import math
2 # Entrada
3 raio = float(input('Digite o valor do raio: '))
4 # Processamento
5 volume = (4 * math.pi * raio ** 3) / 3
6 # Saída
7 print(f'O volume da esfera é: {volume:.2f}')
```

Exercício 3

Implemente um programa que tem como dados de entrada o comprimento, a largura e a altura (números inteiros), calcule e imprima o volume de um paralelepípedo (com duas casas decimais), utilizando a seguinte fórmula:

$$Volume = comprimento * largura * altura$$

Exemplos de execução:

Exemplo 1:

```
Digite o valor do comprimento: 50
Digite o valor da largura: 20
Digite o valor da altura: 15
O volume do paralelepípedo é: 15000.00
```

Exemplo 2:

```
Digite o valor do comprimento: 100
Digite o valor da largura: 50
Digite o valor da altura: 25
O volume do paralelepípedo é: 125000.00
```

Exemplo 3:

```
Digite o valor do comprimento: 75
Digite o valor da largura:  50
Digite o valor da altura:  30
O volume do paralelepípedo é: 112500.00
```

Solução:

```
# Entrada
comprimento = int(input('Digite o valor do comprimento: '))
largura = int(input('Digite o valor da largura: '))
altura = int(input('Digite o valor da altura: '))
# Processamento
volume = comprimento * largura * altura
# Saída
print(f'O volume do paralelepípedo é: {volume:.2f}')
```

Exercício 4

Elabore um programa que receba como entradas um determinado valor em Real (R$) como saldo de um usuário e um valor para a cotação do Dólar (US$) (números reais). No final apresente como saída o valor convertido em Dólar (com duas casas decimais).

Exemplos de execução:

Exemplo 1:

```
Digite o valor em real: 500
Digite o valor de cotação do dólar: 4.15
O valor convertido é: 120.48
```

Exemplo 2:

```
Digite o valor em real: 1000
Digite o valor de cotação do dólar: 4.18
O valor convertido é: 239.23
```

Exemplo 3:

```
Digite o valor em real: 999
Digite o valor de cotação do dólar: 4.11
O valor convertido é: 243.07
```

Solução:

```
# Entrada
valor = float(input('Digite o valor em real: '))
cotacao = float(input('Digite o valor de cotação do dólar: '))
# Processamento
dolar = valor/cotacao
# Saída
print(f'O valor convertido é: {dolar:.2f}')
```

Exercício 5

Implemente um programa que receba como entrada o valor de um veículo (número real) e calcule e imprima o valor do IPVA (Imposto sobre a Propriedade de Veículos Automotores) considerando a alíquota de 4.0% (com duas casas decimais).

Exemplos de execução:

Exemplo 1:

```
Digite o valor do veículo: 100000
O valor do IPVA é : 4000.00
```

Exemplo 2:

```
Digite o valor do veículo: 55000
O valor do IPVA é : 2200.00
```

Exemplo 3:

```
Digite o valor do veículo: 35000
O valor do IPVA é : 1400.00
```

Solução:

```
# Entrada
valor = float(input('Digite o valor do veículo: '))
taxa = 0.04
# Processamento
ipva = valor * taxa
# Saída
print(f'O valor do IPVA é: {ipva:.2f}')
```

9.4 Exercícios do Módulo 4: Estrutura de Decisão

Exercício 1

Elabore um programa que leia as quatro notas de um aluno, números reais entre 0 a 100, e calcule a média das notas. E no final, como saída, imprima a média (com uma casa decimal) e uma mensagem de aprovação se a média for maior ou igual a 70 ou reprovação caso a média for inferior.

Exemplos de execução:

Exemplo 1:

```
Digite a primeira nota: 90
Digite a segunda nota: 85
Digite a terceira nota: 95
Digite a quarta nota: 100
A média é 92.5. O aluno foi aprovado.
```

Exemplo 2:

```
Digite a primeira nota: 50
Digite a segunda nota: 65
Digite a terceira nota: 70
Digite a quarta nota: 45
A média é 57.5. O aluno foi reprovado.
```

Exemplo 3:

```
Digite a primeira nota: 100
Digite a segunda nota: 95
Digite a terceira nota: 101
Digite a quarta nota: 90
Erro: os valores das notas devem estar entre 0 a 100.
```

Solução:

```
1  # Entrada
2  nota1 = float(input('Digite a primeira nota: '))
3  nota2 = float(input('Digite a segunda nota: '))
4  nota3 = float(input('Digite a terceira nota: '))
5  nota4 = float(input('Digite a quarta nota: '))
6  # Processamento e Saída
7  if (nota1 >= 0 and nota1 <= 100) and (nota2 >= 0 and nota2 <= 100) and
       (nota3 >= 0 and nota3 <= 100) and (nota4 >= 0 and nota4 <= 100):
8    media = (nota1 + nota2 + nota3 + nota4) / 4
9    if media >= 70:
10     print(f'A média é {media:.1f}. O aluno foi aprovado.')
11   else:
12     print(f'A média é {media:.1f}. O aluno foi reprovado.')
13 else:
14   print('Erro: os valores das notas devem estar entre 0 a 100.')
```

Exercício 2

Elabore um programa que leia três valores inteiros x, y e z e verifique se estes valores podem constituir os comprimentos de um triângulo e se caso positivo informe se este é um triângulo equilátero, isósceles ou escaleno.

Observações:

- O triângulo equilátero possui os três lados com medidas iguais;

- O triângulo isósceles possui dois lados com medidas iguais;
- O triângulo escaleno possui os três lados com medidas diferentes.
- Condição de desigualdade triangular: $(x < y + z),\ (y < x + z)\ e\ (z < x + y)$

Exemplos de execução:

Exemplo 1:

```
Digite o valor de x: 10
Digite o valor de y: 10
Digite o valor de z: 10
Triângulo Equilátero
```

Exemplo 2:

```
Digite o valor de x: 50
Digite o valor de y: 25
Digite o valor de z: 50
Triângulo Isósceles
```

Exemplo 3:

```
Digite o valor de x: 100
Digite o valor de y: 200
Digite o valor de z: 150
Triângulo Escaleno
```

Exemplo 4:

```
Digite o valor de x: 200
Digite o valor de y: 0
Digite o valor de z: 400
Erro: os valores informados não constituem um triângulo.
```

Solução:

```
# Entrada
x = int(input('Digite o valor de x: '))
y = int(input('Digite o valor de y: '))
z = int(input('Digite o valor de z: '))
# Processamento e Saída
if (x < y + z) and (y < x + z) and (z < x + y):
  if (x == y) and (y == z):
    print('Triângulo Equilátero')
  elif (x == y) or (y == z) or (x == z):
    print('Triângulo Isósceles')
  else:
     print('Triângulo Escaleno')
else:
  print('Erro: os valores informados não constituem um triângulo.')
```

Exercício 3

Implemente um programa que leia o valor de um veículo (número real) e sua respectiva classe (número inteiro), como descreve a Tabela abaixo, para o cálculo do IPVA (Imposto sobre a Propriedade de Veículos Automotores). O cálculo deve considerar a alíquota da respectiva classe do veículo. No final imprima o resultado do cálculo como saída.

Classe	Descrição	Alíquota
1	Automóveis, veículos de uso misto e utilitários, caminhonetes cabine estendida e dupla.	4,0%
2	Caminhonetes de carga (pick-ups) e furgão.	3,0%
3	Automóveis, veículos de uso misto e utilitários com autorização para transporte público (ex: táxi, escolar) comprovada mediante registro no órgão de trânsito na categoria aluguel.	2,0%
4	Motocicletas e similares.	2,0%
5	Veículos de locadoras (pessoa jurídica).	1,0%
6	Ônibus, micro ônibus, caminhão, caminhão trator.	1,0%

Exemplos de execução:

Exemplo 1:

```
Digite o valor do veículo: 120000
Digite a classe do veículo: 1
O valor do IPVA é : 4800.00
```

Exemplo 2:

```
Digite o valor do veículo: 250000
Digite a classe do veículo: 6
O valor do IPVA é : 2500.00
```

Exemplo 3:

```
Digite o valor do veículo: 8000
Digite a classe do veículo: 4
O valor do IPVA é : 160.00
```

Solução:

```
# Entrada
valor = float(input('Digite o valor do veículo: '))
classe = int(input('Digite a classe do veículo: '))
taxa = 0
# Processamento
if classe == 1:
  taxa = 0.04
elif classe == 2:
  taxa = 0.03
elif classe == 3 or classe == 4:
  taxa = 0.02
elif classe == 5 or classe == 6:
  taxa = 0.01
ipva = valor * taxa
# Saída
print(f'O valor do IPVA é : {ipva:.2f}')
```

Exercício 4

Elabore um programa que leia dois valores reais x e y e que pergunte qual operação deve ser realizada. As operações que poderão ser calculadas são a soma $(+)$, subtração$(-)$, multiplicação$(*)$, divisão$(/)$ e potenciação $(\wedge)$. **Observação**: para efetuar a divisão, y deverá ser diferente de zero.

Exemplos de execução:

Exemplo 1:

```
Digite o primeiro valor: 25
Digite o segundo valor: 25
Informe a operação: +
25.00 + 25.00 = 50.00
```

Exemplo 2:

```
Digite o primeiro valor: 5
Digite o segundo valor: 5
Informe a operação: ^
5.00 ^ 5.00 = 3125.00
```

Exemplo 3:

```
Digite o primeiro valor: 81
Digite o segundo valor: 9
Informe a operação: /
81.00 / 9.00 = 9.00
```

Exemplo 4:

```
Digite o primeiro valor: 1000
Digite o segundo valor: 0
Informe a operação: /
Erro: Divisão por zero !
```

Solução:

```
# Entrada
valor1 = float(input('Digite o primeiro valor:'))
valor2 = float(input('Digite o segundo valor:'))
operacao = input('Informe a operação:')
# Processamento e Saída
if operacao == '+':
  soma = valor1 + valor2
  print(f'{valor1:.2f} + {valor2:.2f} = {soma:.2f}')
elif operacao == '-':
  subtracao = valor1 - valor2
  print(f'{valor1:.2f} - {valor2:.2f} = {subtracao:.2f}')
elif operacao == '*':
  mult = valor1 * valor2
  print(f'{valor1:.2f} * {valor2:.2f} = {mult:.2f}')
elif operacao == '/':
  if valor2 != 0:
    divisao = valor1 / valor2
    print(f'{valor1:.2f} / {valor2:.2f} = {divisao:.2f}')
  else:
    print(f'Erro: Divisão por zero ! ')
elif operacao == '^':
  pot = valor1 ** valor2
  print(f'{valor1:.2f} ^ {valor2:.2f} = {pot:.2f}')
```

Exercício 5

Um vendedor recebe um determinado valor de comissão sobre o total de suas vendas efetuadas. Em vendas superiores a 5000,00 o valor a receber corresponde a 25%, os valores na faixa de 5000,00 a 2500,00 corresponde a 20% e para valores inferiores a 2500,00 o valor corresponde a 15%. Implemente um programa que leia o valor total das vendas (número real) e calcule a quantia que o vendedor irá receber.

Exemplos de execução:

Exemplo 1:

```
Digite o valor total das vendas: 7000
O valor de comissão é: 1750.00
```

Exemplo 2:

```
Digite o valor total das vendas: 3000
O valor de comissão é: 600.00
```

Exemplo 3:

```
Digite o valor total das vendas: 500
O valor de comissão é: 75.00
```

Solução:

```
# Entrada
valor = float(input('Digite o valor total das vendas: '))
# Processamento
comissao = 0
if valor > 5000:
  comissao = valor * 0.25
if valor >= 2500 and valor <= 5000:
  comissao = valor * 0.20
if valor < 2500:
  comissao = valor * 0.15
# Saída
print(f'O valor de comissão é: {comissao:.2f} ')
```

9.5 Exercícios do Módulo 5: Estrutura de Repetição

Exercício 1

Elabore um programa que leia as quatro notas de um aluno, números reais entre 0 a 100, e calcule a média das notas. E no final, como saída, imprima a média (com uma casa decimal) e uma mensagem de aprovação se a média for maior ou igual a 70 ou reprovação caso a média for inferior.

Exemplos de execução:

Exemplo 1:

```
Digite a primeira nota: 90
Digite a segunda nota: 85
Digite a terceira nota: 95
Digite a quarta nota: 100
A média é 92.5. O aluno foi aprovado.
```

Exemplo 2:

```
Digite a primeira nota: 50
Digite a segunda nota: 65
```

```
Digite a terceira nota: 70
Digite a quarta nota: 45
A média é 57.5. O aluno foi reprovado.
```

Exemplo 3:

```
Digite a primeira nota: 100
Digite a segunda nota: 95
Digite a terceira nota: 101
Digite a quarta nota: 90
Erro: os valores das notas devem estar entre 0 a 100.
```

Solução:

```
# Entrada
nota1 = float(input('Digite a primeira nota: '))
nota2 = float(input('Digite a segunda nota: '))
nota3 = float(input('Digite a terceira nota: '))
nota4 = float(input('Digite a quarta nota: '))
# Processamento e Saída
if (nota1 >= 0 and nota1 <= 100) and (nota2 >= 0 and nota2 <= 100) and
     (nota3 >= 0 and nota3 <= 100) and (nota4 >= 0 and nota4 <= 100):
  media = (nota1 + nota2 + nota3 + nota4) / 4
  if media >= 70:
    print(f'A média é {media:.1f}. O aluno foi aprovado.')
  else:
    print(f'A média é {media:.1f}. O aluno foi reprovado.')
else:
  print('Erro: os valores das notas devem estar entre 0 a 100.')
```

Exercício 2

Elabore um programa que leia três valores inteiros x, y e z e verifique se estes valores podem constituir os comprimentos de um triângulo e se caso positivo informe se este é um triângulo equilátero, isósceles ou escaleno.

Observações:

- O triângulo equilátero possui os três lados com medidas iguais;
- O triângulo isósceles possui dois lados com medidas iguais;
- O triângulo escaleno possui os três lados com medidas diferentes.
- Condição de desigualdade triangular: $(x < y + z)$, $(y < x + z)$ e $(z < x + y)$

Exemplos de execução:

Exemplo 1:

```
Digite o valor de x: 10
Digite o valor de y: 10
```

```
Digite o valor de z: 10
Triângulo Equilátero
```

Exemplo 2:

```
Digite o valor de x: 50
Digite o valor de y: 25
Digite o valor de z: 50
Triângulo Isósceles
```

Exemplo 3:

```
Digite o valor de x: 100
Digite o valor de y: 200
Digite o valor de z: 150
Triângulo Escaleno
```

Exemplo 4:

```
Digite o valor de x: 200
Digite o valor de y: 0
Digite o valor de z: 400
Erro: os valores informados não constituem um triângulo.
```

Solução:

```
# Entrada
x = int(input('Digite o valor de x: '))
y = int(input('Digite o valor de y: '))
z = int(input('Digite o valor de z: '))
# Processamento e Saída
if (x < y + z) and (y < x + z) and (z < x + y):
  if (x == y) and (y == z):
    print('Triângulo Equilátero')
  elif (x == y) or (y == z) or (x == z):
    print('Triângulo Isósceles')
  else:
     print('Triângulo Escaleno')
else:
  print('Erro: os valores informados não constituem um triângulo.')
```

Exercício 3

Implemente um programa que leia o valor de um veículo (número real) e sua respectiva classe (número inteiro), como descreve a Tabela abaixo, para o cálculo do IPVA (Imposto sobre a Propriedade de Veículos Automotores). O cálculo deve considerar a alíquota da respectiva classe do veículo. No final imprima o resultado do cálculo como saída.

Classe	Descrição	Alíquota
1	Automóveis, veículos de uso misto e utilitários, caminhonetes cabine estendida e dupla.	4,0%
2	Caminhonetes de carga (pick-ups) e furgão.	3,0%
3	Automóveis, veículos de uso misto e utilitários com autorização para transporte público (ex: táxi, escolar) comprovada mediante registro no órgão de trânsito na categoria aluguel.	2,0%
4	Motocicletas e similares.	2,0%
5	Veículos de locadoras (pessoa jurídica).	1,0%
6	Ônibus, micro ônibus, caminhão, caminhão trator.	1,0%

Exemplos de execução:

Exemplo 1:

```
Digite o valor do veículo: 120000
Digite a classe do veículo: 1
O valor do IPVA é : 4800.00
```

Exemplo 2:

```
Digite o valor do veículo: 250000
Digite a classe do veículo: 6
O valor do IPVA é : 2500.00
```

Exemplo 3:

```
Digite o valor do veículo: 8000
Digite a classe do veículo: 4
O valor do IPVA é : 160.00
```

Solução:

```
# Entrada
valor = float(input('Digite o valor do veículo: '))
classe = int(input('Digite a classe do veículo: '))
taxa = 0
# Processamento
if classe == 1:
  taxa = 0.04
elif classe == 2:
  taxa = 0.03
elif classe == 3 or classe == 4:
  taxa = 0.02
elif classe == 5 or classe == 6:
  taxa = 0.01
ipva = valor * taxa
# Saída
print(f'O valor do IPVA é : {ipva:.2f}')
```

Exercício 4

Elabore um programa que leia dois valores reais x e y e que pergunte qual operação deve ser realizada. As operações que poderão ser calculadas são a soma $(+)$, subtração$(-)$, multiplicação$(*)$, divisão$(/)$ e potenciação $(\wedge)$. **Observação**: para efetuar a divisão, y deverá ser diferente de zero.

Exemplos de execução:

Exemplo 1:

```
Digite o primeiro valor: 25
Digite o segundo valor: 25
Informe a operação: +
25.00 + 25.00 = 50.00
```

Exemplo 2:

```
Digite o primeiro valor: 5
Digite o segundo valor: 5
Informe a operação: ^
5.00 ^ 5.00 = 3125.00
```

Exemplo 3:

```
Digite o primeiro valor: 81
Digite o segundo valor: 9
Informe a operação: /
81.00 / 9.00 = 9.00
```

Exemplo 4:

```
Digite o primeiro valor: 1000
Digite o segundo valor: 0
Informe a operação: /
Erro: Divisão por zero !
```

Solução:

```
# Entrada
valor1 = float(input('Digite o primeiro valor:'))
valor2 = float(input('Digite o segundo valor:'))
operacao = input('Informe a operação:')
# Processamento e Saída
if operacao == '+':
  soma = valor1 + valor2
  print(f'{valor1:.2f} + {valor2:.2f} = {soma:.2f}')
elif operacao == '-':
  subtracao = valor1 - valor2
  print(f'{valor1:.2f} - {valor2:.2f} = {subtracao:.2f}')
elif operacao == '*':
  mult = valor1 * valor2
  print(f'{valor1:.2f} * {valor2:.2f} = {mult:.2f}')
elif operacao == '/':
  if valor2 != 0:
    divisao = valor1 / valor2
    print(f'{valor1:.2f} / {valor2:.2f} = {divisao:.2f}')
  else:
    print(f'Erro: Divisão por zero ! ')
elif operacao == '^':
  pot = valor1 ** valor2
  print(f'{valor1:.2f} ^ {valor2:.2f} = {pot:.2f}')
```

Exercício 5

Um vendedor recebe um determinado valor de comissão sobre o total de suas vendas efetuadas. Em vendas superiores a 5000,00 o valor a receber corresponde a 25%, os valores na faixa de 5000,00 a 2500,00 corresponde a 20% e para valores inferiores a 2500,00 o valor corresponde a 15%. Implemente um programa que leia o valor total das vendas (número real) e calcule a quantia que o vendedor irá receber.

Exemplos de execução:

Exemplo 1:

```
Digite o valor total das vendas: 7000
O valor de comissão é: 1750.00
```

Exemplo 2:

```
Digite o valor total das vendas: 3000
O valor de comissão é: 600.00
```

Exemplo 3:

```
Digite o valor total das vendas: 500
O valor de comissão é: 75.00
```

Solução:

```
1  # Entrada
2  valor = float(input('Digite o valor total das vendas: '))
3  # Processamento
4  comissao = 0
5  if valor > 5000:
6    comissao = valor * 0.25
7  if valor >= 2500 and valor <= 5000:
8    comissao = valor * 0.20
9  if valor < 2500:
10   comissao = valor * 0.15
11 # Saída
12 print(f'O valor de comissão é: {comissao:.2f} ')
```

9.6 Exercícios do Módulo 6: Função

Exercício 1

Fazer uma função que calcula a distância euclidiana entre dois pontos bidimensionais e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa. Utilizar x_1, x_2, y_1 e y_2 como argumentos da função.

Fórmulas:

- $P_1 = (x_1, y_1)$
- $P_2 = (x_2, y_2)$
- $D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Exemplos de execução:

Exemplo 1:

```
Digite o valor da coordenada X1: 1
Digite o valor da coordenada Y1: 2
Digite o valor da coordenada X2: 2
Digite o valor da coordenada Y2: 3
A distância entre os pontos é 1.41
```

Solução:

```
1 def euclidian_dist(x1,x2,y1,y2):
2     return ((x1-x2)**2 + (y1-y2)**2)**(1/2)
3
4 X1 = float(input("Digite o valor da coordenada X1: "))
5 Y1 = float(input("Digite o valor da coordenada Y1: "))
6 X2 = float(input("Digite o valor da coordenada X2: "))
7 Y2 = float(input("Digite o valor da coordenada Y2: "))
8 dist = euclidian_dist(X1,X2,Y1,Y2)
9 print(f'A distância entre os pontos é {dist:.2f}')
```

Exercício 2

Fazer uma função que converte uma entrada de temperatura em Celsius para Fahrenheit e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_F = (T_c.\frac{9}{5}) + 32$

Exemplos de execução:

Exemplo 1:

```
Digite uma temperatura em Celsius: 37
A temperatura em Fahrenheit é 98.6
```

Solução:

```
1 def fahrenheit(celsius):
2     return (celsius*9/5) + 32
3
4 T_c = float(input('Digite uma temperatura em Celsius: '))
5 T_f = fahrenheit(T_c)
6 print(f'A temperatura em fahrenheit é {T_f:.1f}')
```

Exercício 3

Fazer uma função que converte uma entrada de temperatura em Fahrenheit para Celsius e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmula:

- $T_C = (T_F - 32).\frac{5}{9}$

Exemplos de execução:

Exemplo 1:

```
Digite a temperatura em Fahrenheit: 98.6
A temperatura em Celsius é 37.0
```

Solução:

```
1 def celsius(fahrenheit):
2     return (fahrenheit-32)*5/9
3
4 T_f = float(input('Digite uma temperatura em Celsius: '))
5 T_c = celsius(T_f)
6 print(f'A temperatura em celsius é {T_c:.1f')
```

Exercício 4

Fazer uma função que calcula a extensão de uma mola vertical com constante elástica k ao pendurar um objeto de massa m. Utilizar k e m como argumentos da função e implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada à função implementa.

Fórmulas:

- $F_e = k.x$
- $F_g = m.g$
- $g = 9,807 m/s^2$

Exemplos de execução:

Exemplo 1:

```
Digite a massa do objeto: 5
Digite a constante da mola: 100
A extensão da mola é 0.49 m
```

Solução:

```
def extensao(m,k):
    g = 9.807
    return m*g/k

massa = float(input('Digite a massa do objeto: '))
const = float(input('Digite a constante da mola: '))
x = extensao(massa, const)
print(f'A extensão da mola é {x:.2f} m')
```

Exercício 5

A inclinação de um telhado é dada por um percentual da largura, conforme a Figura 30.

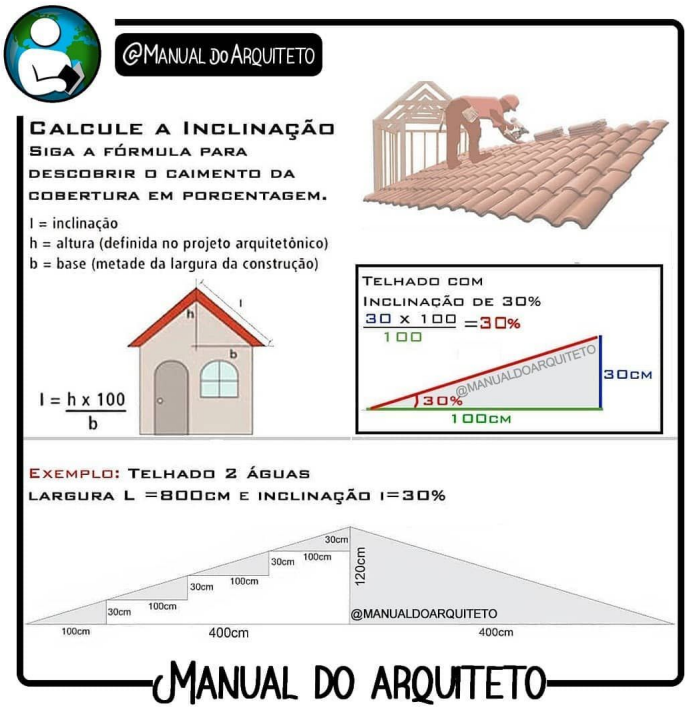

Figura 37: Esquema de inclinação de telhado.

Quando o telhado é de "duas águas", a largura deve ser dividida por dois antes de calcular a altura. Esses dados podem ser usados para calcular o comprimento do telhado, que é a hipotenusa do triângulo retângulo.

Fazer duas funções: uma função que calcula o comprimento c_1 de um telhado de duas águas e uma que calcula o comprimento c_2 de um telhado de uma água, dadas como entradas uma largura l e um percentual p.

Implementar um programa principal que faça a entrada e saída, gerando o resultado a partir de chamada às funções implementas.

Fórmulas (uma água):

- $h = \frac{p.l}{100}$
- $c_1 = \sqrt{l^2 + h^2}$

Fórmulas (duas águas):

- $h = \frac{p.(l/2)}{100}$
- $c_2 = 2.\sqrt{(l/2)^2 + h^2}$

Observação: Os resultados encontrados devem ser iguais (por semelhança de triângulos, é fácil observar essa relação).

Exemplos de execução:

Exemplo 1:

```
Digite a largura do telhado: 10
Digite a inclinação percentual: 30
O comprimento do telhado de uma água é 10.44
O comprimento do telhado de duas águas é 10.44
```

Solução:

```
1  def comprimento_1(l,p):
2      h = l*p/100
3      c = (l**2 + h**2)**(1/2)
4      return c
5
6  def comprimento_2(l,p):
7      h = (l/2)*p/100
8      c = ((l/2)**2 + h**2)**(1/2)
9      return 2*c
10
11 largura = float(input('Digite a largura do telhado: '))
12 percentual = float(input('Digite a inclinação percentual: '))
13 c1 = comprimento_1(largura, percentual)
14 c2 = comprimento_2(largura, percentual)
15 print(f'O comprimento do telhado de uma água é {c1:.2f}')
16 print(f'O comprimento do telhado de duas águas é {c2:.2f}')
```

9.7 Exercícios do Módulo 7: Estruturas Homogêneas

Exercício 1

Utilizando vetores, imprimir uma sequência de pontos indicando a distância de queda livre de um corpo, dada uma sequência de tempo de $t = 0$ até um valor dado como entrada pelo usuário. Os valores impressos devem ter uma precisão de uma casa decimal.

Fórmulas:

- $g = 9.807m/s^2$
- $d = \frac{gt^2}{2}$

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
```

Exemplo 2:

```
Digite o tempo final: 6
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
4 78.5
5 122.6
6 176.5
```

Solução:

```
1  ft = int(input('Digite o tempo final: '))
2  g = 9.807
3  T = []
4  D = []
5  for i in range(ft+1):
6      T.append(i)
7      D.append(g*i*i/2)
8  print('t D(t)')
9  for i in range(ft+1):
10     print(f'{T[i]} {D[i]:.1f}')
```

Exercício 2

Uma função de segundo grau é dada pela fórmula:

$$f(x) = ax^2 + bx + c$$

Dadas como entradas:

- Valores a, b e c
- Ponto x_0 inicial
- Ponto x_f final

Criar dois vetores com os dados dos valores da função nesses pontos.
Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -3
Digite o valor final de x: 3
x f(x)
-3 13
-2 6
-1 3
0 4
1 9
2 18
3 31
```

Exemplo 2:

```
Digite o valor de "a": 2
Digite o valor de "b": 0
Digite o valor de "c": 0
Digite o valor inicial de x: -5
Digite o valor final de x: 2
x f(x)
-5 50
-4 32
-3 18
-2 8
-1 2
0 0
1 2
2 8
```

Solução:

```
a = int(input('Digite o valor de "a": '))
b = int(input('Digite o valor de "b": '))
c = int(input('Digite o valor de "c": '))
x0 = int(input('Digite o valor inicial de x: '))
x1 = int(input('Digite o valor final de x: '))
X = []
Y = []
for i in range(x0, x1+1):
    X.append(i)
    Y.append(a*i*i + b*i + c)
print('x f(x)')
for i in range(len(Y)):
    print(f'{X[i]} {Y[i]}')
```

Exercício 3

Implemente um programa que realize a entrada de dados de um vetor. Primeiro o usuário fornece a quantidade de elementos do vetor, em seguida, fornece um valor numérico para cada elemento. Após preenchido o vetor, os valores armazenados no mesmo devem ser impressos na tela com uma precisão de uma casa decimal.

Exemplos de execução:

Exemplo 1:

```
Informe a dimensão do vetor: 3
Valor do elemento 0: 3.5
Valor do elemento 1: 7
Valor do elemento 2: 10.65
Elementos do vetor:
3.5
7.0
10.7
```

Exemplo 2:

```
Informe a dimensão do vetor: 5
Valor do elemento 0: 6.92
Valor do elemento 1: 4
Valor do elemento 2: 5.271
Valor do elemento 3: 2.34
Valor do elemento 4: 2.319
Elementos do vetor:
6.9
4.0
5.3
2.3
2.3
```

Solução:

```
N = int(input('Informe a dimensão do vetor: '))
vetor = [ ]
for i in range(N):
    vetor.append(float(input(f'Valor do elemento {i}: ')))
print('Elementos do vetor: ')
for i in range(N):
    print(f'{vetor[i]:.1f}')
```

Exercício 4

Repetir a atividade do Exercício 1, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
```

Exemplo 2:

```
Digite o tempo final: 6
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
[4, 78.456]
[5, 122.5875]
[6, 176.526]
```

Solução:

```
ft = int(input('Digite o tempo final: '))
g = 9.807
data = []
for i in range(ft+1):
    linha = []
    linha.append(i)
    linha.append(g*i*i/2)
    data.append(linha)
print('[\'t\', \'D(t)\']')
for i in range(ft+1):
    print(f'[{data[i][0]}, {data[i][1]}]')
```

Exercício 5

Repetir a atividade do Exercício 2, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -5
Digite o valor final de x: 5

['x', 'f(x)']
[-5, 39]
[-4, 24]
[-3, 13]
[-2, 6]
[-1, 3]
[0, 4]
[1, 9]
[2, 18]
[3, 31]
[4, 48]
[5, 69]
```

Exemplo 2:

```
Digite o valor de "a": 4
Digite o valor de "b": 0
Digite o valor de "c": -4
Digite o valor inicial de x: -3
Digite o valor final de x: 6

['x', 'f(x)']
[-3, 32]
[-2, 12]
[-1, 0]
[0, -4]
[1, 0]
[2, 12]
[3, 32]
[4, 60]
[5, 96]
[6, 140]
```

Solução:

```
#Entrada
a = int(input('Digite o valor de "a": '))
b = int(input('Digite o valor de "b": '))
c = int(input('Digite o valor de "c": '))
x0 = int(input('Digite o valor inicial de x: '))
x1 = int(input('Digite o valor final de x: '))
data = []
for i in range(x0, x1+1):
    linha = []
    linha.append(i)
    linha.append(a*i*i + b*i + c)
    data.append(linha)
print('[\'x\', \'f(x)\']')
for i in range(len(data)):
    print(f'[{data[i][0]}, {data[i][1]}]')
```

Exercício 6

Fazer um programa que gere uma matriz quadrada de dimensão n, preenchida em todas as posições por um valor v. Ambos n e v são valores informados pelo usuário, como entradas. Imprimir o conteúdo da matriz ao final do programa.

Exemplos de execução:

Exemplo 1:

```
Digite a dimensão da matriz: 4
Digite o valor para preencher os campos: 2
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
```

Exemplo 2:

```
Digite a dimensão da matriz: 6
Digite o valor para preencher os campos: -3
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
```

Solução:

```
n = int(input('Digite a dimensão da matriz: '))
v = int(input('Digite o valor para preencher os campos: '))
matriz = []
for i in range(n):
    linha = []
    for j in range(n):
        linha.append(v)
    matriz.append(linha)
for i in range(n):
    print(f'[{matriz[i][0]}', end='')
    for j in range(1, n):
        print(f', {matriz[i][j]}', end='')
    print(']')
```

Exercício 7

Implemente um programa que realize a entrada de dados de uma matriz. Primeiro o usuário fornece as dimensões da matriz, em seguida, fornece um valor textual para cada elemento. Após preenchida a matriz, o usuário fornece uma linha e coluna, e o programa imprime o valor correspondente a estes índices. Considere que os dados informados são sempre válidos.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de linhas: 2
Informe a quantidade de colunas: 3
Valor do elemento 0, 0: Sardinha
Valor do elemento 0, 1: Salame
Valor do elemento 0, 2: Picanha
Valor do elemento 1, 0: Coca-cola
Valor do elemento 1, 1: Guaraná
Valor do elemento 1, 2: Pipoca
Informe uma linha: 1
Informe uma coluna: 2
O valor armazenado em [1,2] é Pipoca
```

Solução:

```
qtdLin = int(input('Informa a quantidade de linhas: '))
qtdCol = int(input('Informa a quantidade de colunas: '))
matriz = [ ]
for lin in range(qtdLin):
    linha = [ ]
    for col in range(qtdCol):
        valor = input(f'Valor do elemento {lin}, {col}: ')
        linha.append(valor)
    matriz.append(linha)
```

```
10 lin = int(input('Informe uma linha: '))
11 col = int(input('Informe uma coluna: '))
12 print(f'O valor armazenado em [{lin},{col}] é {matriz[lin][col]}')
```

Exercício 8

Implemente um programa para contabilizar a pontuação em uma Gincana, seguindo o algoritmo:

1. Pergunte a quantidade de equipes.
2. Crie um vetor, cada elemento representa uma equipe e possui pontuação inicial $zero$.
3. Pergunte o índice da equipe que pontuou.
4. Enquanto o índice for válido:
   (a) Pergunte a quantidade de pontos.
   (b) Atualize, de forma cumulativa, a pontuação da equipe no vetor.
   (c) Pergunte o índice da equipe que pontuou.
5. Imprima o conteúdo do vetor.

Tanto os índices quanto a pontuação devem ser valores inteiros.
Exemplos de execução:

Exemplo 1:

```
Quantidade de equipes: 5
Índice da equipe: 2
Quantidade de pontos: 10
Índice da equipe: 1
Quantidade de pontos: 5
Índice da equipe: 0
Quantidade de pontos: 30
Índice da equipe: 2
Quantidade de pontos: 25
Índice da equipe: -1
Pontuação das equipes:
[ 30, 5, 35, 0, 0 ]
```

Exemplo 2:

```
Quantidade de equipes: 6
Índice da equipe: 5
Quantidade de pontos: 4
Índice da equipe: 1
Quantidade de pontos: 3
Índice da equipe: 1
Quantidade de pontos: 4
Índice da equipe: 6
Pontuação das equipes:
[ 0, 7, 0, 0, 0, 4 ]
```

Solução:

```
1  import biblioteca as bib
2
3  N = int(input('Quantidade de equipes: '))
4  pontuacao = bib.criarVetor(N, 0)
5  equipe = int(input('Índice da equipe: '))
6  while equipe >= 0 and equipe < N:
7      pontos = int(input('Quantidade de pontos: '))
8      pontuacao[equipe] = pontuacao[equipe] + pontos
9      equipe = int(input('Índice da equipe: '))
10 print('Pontuação das equipes:')
11 bib.imprimeVetor(pontuacao)
```

Exercício 9

Implemente um programa para ajudar na leitura de consumo de energia feito pela CEMIG nos imóveis de uma cidade. O registro é feito pelo mapeamento de ruas e imóveis e armazenado em uma matriz, as linhas representam as ruas e as colunas representam os números dos imóveis. Primeiro o programa solicita que seja fornecida como entrada uma **string** que contenha os valores iniciais, preenchendo uma matriz com estes valores. Em seguida, o programa pergunta quantas novas leituras individuais serão fornecidas. Para cada leitura individual, o programa pergunta o índice da rua (linha), o índice do imóvel (coluna), e o valor lido pelo funcionário. O programa deve então fornecer a diferença da última leitura referente ao imóvel do valor lido pelo funcionário, e em seguida atualizar a matriz com o valor lido para o imóvel.

Tanto os índices quanto as leituras devem ser valores inteiros. Não é necessário validar dados de entrada, considere que serão sempre válidos.

Exemplo de execução:

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 6
 - Diferença: 2
Leitura 2:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 10
 - Diferença: 4
Leitura 3:
 - Índice da rua: 2
 - Índice do imóvel: 2
 - Valor lido: 5
 - Diferença: 3
```

```
Leitura 4:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 5
 - Diferença: 0
Leitura 5:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 7
 - Diferença: 2
```

Solução:

```
1  import biblioteca as bib
2
3  valores = input('Valores iniciais: ')
4  matriz = bib.preencherMatriz(valores, 'int')
5  N = int(input('Quantidade de novas leituras: '))
6  for i in range(N):
7      print(f'Leitura {i+1}:')
8      lin = int(input(' - Índice da rua: '))
9      col = int(input(' - Índice do imóvel: '))
10     valor = int(input(' - Valor lido: '))
11     diferenca = valor - matriz[lin][col]
12     matriz[lin][col] = valor
13     print(f' - Diferença: {diferenca}')
```

Exercício 10

Implemente um programa similar ao anterior, Exercício 9. Mas, agora a entrada de cada leitura deve ser feita por um único **input**, armazenando os valores em um vetor, onde, índice 0 representa a rua, índice 1 representa o imóvel e índice 2 representa o valor lido. O restante do programa funciona da mesma maneira, gerando o mesmo resultado, conforme o exemplo de execução a seguir.

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1: 0, 2, 6
 - Diferença: 2
Leitura 2: 0, 2, 10
 - Diferença: 4
Leitura 3: 2, 2, 5
 - Diferença: 3
Leitura 4: 0, 0, 5
 - Diferença: 0
Leitura 5: 0, 0, 7
 - Diferença: 2
```

Solução:

```
import biblioteca as bib

valores = input('Valores iniciais: ')
matriz = bib.preencherMatriz(valores, 'int')
N = int(input('Quantidade de novas leituras: '))
for i in range(N):
    leitura = input(f'Leitura {i+1}: ')
    vetor = bib.preencherVetor(leitura, 'int')
    lin = vetor[0]
    col = vetor[1]
    diferenca = vetor[2] - matriz[lin][col]
    matriz[lin][col] = vetor[2]
    print(f' - Diferença: {diferenca}')
```

9.8 Exercícios do Módulo 8: Estruturas Heterogêneas

Exercício 1

Implemente uma versão do Exemplo 4 para armazenar os dados de alunos em diferentes disciplinas. Sendo assim, deve-se utilizar uma matriz para armazenar os dados, em cada elemento da matriz será armazenada a nota e a frequência. As linhas representam alunos e as colunas representam disciplinas, que serão referenciadas pelos seus respectivos índices. Primeiramente o programa pergunta a quantidade de alunos e disciplinas, em seguida pergunta nota e frequência para cada elemento da matriz. Depois disso, para cada aluno, o programa deve identificar a disciplina com a maior nota, imprimindo a nota com precisão de uma casa decimal e a frequência nesta disciplina. Notas são valores reais e frequência inteiros.

Exemplos de execução:

Exemplo 1:

```
Quantidade de alunos: 2
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 4
  - Frequência: 4
 - Dados da disciplina 2:
  - Nota: 3
  - Frequência: 3

Dados do aluno 2:
 - Dados da disciplina 1:
  - Nota: 2
  - Frequência: 2
 - Dados da disciplina 2:
  - Nota: 1
```

```
   - Frequência: 1

  Maiores notas e frequências:
   - Aluno 1: nota = 4.0; frequência = 4
   - Aluno 2: nota = 2.0; frequência = 2
```

Exemplo 2:

```
  Quantidade de alunos: 3
  Quantidade de disciplinas: 2

  Dados do aluno 1:
   - Dados da disciplina 1:
    - Nota: 1
    - Frequência: 10
   - Dados da disciplina 2:
    - Nota: 2
    - Frequência: 20

  Dados do aluno 2:
   - Dados da disciplina 1:
    - Nota: 3
    - Frequência: 30
   - Dados da disciplina 2:
    - Nota: 4
    - Frequência: 40

  Dados do aluno 3:
   - Dados da disciplina 1:
    - Nota: 5
    - Frequência: 50
   - Dados da disciplina 2:
    - Nota: 6
    - Frequência: 60

  Maiores notas:
   - Aluno 1: nota = 2.0; frequência = 20
   - Aluno 2: nota = 4.0; frequência = 40
   - Aluno 3: nota = 6.0; frequência = 60
```

Solução:

```
1 nA = int(input('Quantidade de alunos: '))
2 nD = int(input('Quantidade de disciplinas: '))
3 alunos = [ ]
4 for a in range(nA):
5     linha = [ ]
6     print(f'\nDados do aluno {a+1}:')
```

```
7       for d in range(nD):
8           aluno = { }
9           print(f' - Dados da disciplina {d+1}:')
10          aluno["nota"] = float(input('  - Nota: '))
11          aluno["freq"] = int(input('  - Frequência: '))
12          linha.append(aluno)
13      alunos.append(linha)
14  print('\nMaiores notas:')
15  for a in range(nA):
16      maior = 0
17      for d in range(nD):
18          if alunos[a][d]["nota"] > alunos[a][maior]["nota"]:
19              maior = d
20      print(f' - Aluno {a+1}: nota = {alunos[a][maior]["nota"]:.1f};
            frequência = {alunos[a][maior]["freq"]}')
```

Exercício 2

Implemente um programa que receba informações sobre retas, armazenando-as em um vetor. Primeiramente pergunta-se quantas retas serão informadas, e em seguida pede-se as coordenadas dos dois pontos (**A** e **B**) que definem cada reta. Uma reta deve ser representada por um registro que contenha dois campos, representando os pontos. Cada ponto é um registro que possui as coordenadas **X** e **Y** como campos. Depois que o vetor estiver totalmente preenchido, você deverá calcular e imprimir o Comprimento de cada reta armazenada no vetor. O Comprimento é obtido através de uma função, definida em seu programa, que recebe a reta como argumento e retorna o comprimento: $C = \sqrt{(B_X - A_X)^2 + (B_Y - A_Y)^2}$. As coordenadas **X** e **Y** são números reais, e a saída das medidas deve possuir precisão de 2 casas decimais. **Dicas:** (1) para obter o ponto **B** de uma **reta**, você deve usar: `reta["B"]`; (2) para obter a coordenada **X** do ponto **B** desta mesma **reta**, você deve usar: `reta["B"]["X"]`, uma vez que temos aqui um registro "dentro" de outro registro.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de retas: 1

Reta 1:
 - Coordenada X de A: 1
 - Coordenada Y de A: 4
 - Coordenada X de B: 1
 - Coordenada Y de B: 1

Medidas das retas:
 - Reta 1: 3.00
```

Exemplo 2:

```
Informe a quantidade de retas: 2

Reta 1:
 - Coordenada X de A: 3.5
 - Coordenada Y de A: 2.1
 - Coordenada X de B: 6
 - Coordenada Y de B: 8.6

Reta 2:
 - Coordenada X de A: -3
 - Coordenada Y de A: 0.9
 - Coordenada X de B: 5.8
 - Coordenada Y de B: -9.1

Medidas das retas:
 - Reta 1: 6.96
 - Reta 2: 13.32
```

Solução:

```
1  import math
2
3  def calcReta(reta):
4      d1 = reta["B"]["X"] - reta["A"]["X"]
5      d2 = reta["B"]["Y"] - reta["A"]["Y"]
6      C = math.sqrt(d1 ** 2 + d2 ** 2)
7      return C
8
9  N = int(input('Informe a quantidade de retas: '))
10 retas = [ ]
11 for i in range(N):
12     print(f'\nReta {i+1}:')
13     A = { }
14     A["X"] = float(input(' - Coordenada X de A: '))
15     A["Y"] = float(input(' - Coordenada Y de A: '))
16     B = { }
17     B["X"] = float(input(' - Coordenada X de B: '))
18     B["Y"] = float(input(' - Coordenada Y de B: '))
19     reta = { "A": A, "B": B }
20     retas.append(reta)
21 print('\nMedidas das retas:')
22 for i in range(N):
23     print(f' - Reta {i+1}: {calcReta(retas[i]):.2f}')
```