



# **LÓGICA DE PROGRAMAÇÃO: PYTHON E ABSTRAÇÕES EM SCRATCH**

Rafael Silva Santos

Rafael Silva Santos

*Lógica de programação: Python e  
abstrações em Scratch*

Instituto Federal do Paraná  
Arapongas, 2024

© 2024 Rafael Silva Santos & Instituto Federal do Paraná  
É vedada a reprodução parcial ou total deste livro sem autorização.

*Este trabalho é dedicado aos meus alunos  
do passado, do presente e do futuro.*

*“Nós só podemos ver um pouco do futuro,  
mas o suficiente para perceber que há muito a fazer.”*  
(Alan Mathison Turing)

# Prefácio

**C**ARO leitor, seja bem-vindo ao fascinante mundo da programação! Este livro foi cuidadosamente elaborado para conduzir você, estudante, por uma jornada empolgante através da linguagem Python.

Nosso principal objetivo é tornar a programação acessível e divertida. Faremos abstrações com a linguagem Scratch, buscando ensinar os conceitos básicos de programação de forma lúdica e visual.

Por meio de exemplos práticos, exercícios envolventes e projetos desafiadores, você desenvolverá uma base sólida em programação. Ao concluir este livro, você estará equipado não apenas com habilidades técnicas, mas também com a confiança e a criatividade necessárias para continuar explorando o vasto mundo da programação.

Este livro se destina a todos que desejam aprender a programar. Se você nunca programou antes, encontrará explicações claras e simples que tornarão o aprendizado prazeroso e eficaz. Se você já tem alguma experiência, os desafios e projetos propostos irão aprofundar seu entendimento e expandir suas habilidades.

## CONVENÇÕES USADAS NESTE LIVRO

Para tornar a leitura mais atrativa e dinâmica, este livro utiliza diferentes formatações visuais para destacar: comandos em CLI, códigos-fonte em Python, atalhos do teclado, itens clicáveis (menus, itens de menu, opções e botões da GUI) e caixas de ênfase coloridas. Exemplos de formatação visual autoexplicativos: `comando em CLI`, *|item clicável em uma tela|*, trecho de código-fonte e <atalho do teclado>. As caixas de ênfase coloridas têm uma cor de acordo com uma finalidade específica:

### Definições

As caixas cinzas apresentam definições importantes.

### Exemplos

As caixas azuis apresentam exemplos.

### Alertas

As caixas vermelhas destacam alertas e avisos cruciais que merecem atenção especial.

### Pontos-chave

As caixas cinza-escuras contêm resumos dos principais tópicos discutidos no capítulo.

## Exercícios

As caixas verdes contêm exercícios para reforçar seu aprendizado.

## COMO ESTE LIVRO ESTÁ ORGANIZADO?

Todos os capítulos contêm exemplos e atividades propostas para fixação e consolidação do conteúdo. Sempre que um novo conceito ou recurso de lógica de programação é introduzido, apresentamos exemplos comentados.

Ao longo do processo de aprendizagem de lógica de programação, é necessário que o estudante possua alguns conhecimentos prévios, como a utilização da linha de comando de um sistema operacional e depuração de códigos-fonte. Parte desses assuntos foi incluída nos apêndices. O livro está organizado em nove capítulos. A seguir, apresentamos uma visão geral dos tópicos abordados em cada capítulo:

- **Capítulo 1:** Introdução ao tema, com a motivação para aprender a programar.
- **Capítulo 2:** Discussão sobre o conceito de algoritmo.
- **Capítulo 3:** Preparação do ambiente de desenvolvimento Python e acesso à plataforma Scratch.
- **Capítulo 4:** Apresentação do primeiro programa e do fluxo de execução sequencial.
- **Capítulo 5:** Discussão sobre variáveis e tipos de dados.
- **Capítulo 6:** Uso de operadores aritméticos e do operador de atribuição.
- **Capítulo 7:** Estratégias de interação com o usuário na CLI.
- **Capítulo 8:** Apresentação de operadores relacionais e operadores lógicos.
- **Capítulo 9:** Estruturas condicionais e controle de fluxo condicional.
- **Capítulo 10:** Estruturas de repetição e controle de laço de repetição.

## MATERIAL DE APOIO

Todos os exemplos do livro estão disponíveis *on-line* em [repositório de exemplos no GitHub](#).

## VAMOS COMEÇAR!

Prepare-se para embarcar em uma jornada emocionante. Você não estará apenas aprendendo a programar, estará explorando novas formas de pensar e resolver problemas. Então, pegue seu computador, abra sua mente e seja bem-vindo ao mundo da programação!

# Lista de Figuras

Figura 1	– Logo da linguagem de programação Scratch. . . . .	14
Figura 2	– Logotipo da linguagem de programação Python. . . . .	15
Figura 3	– Atores de <i>Monty Python's Flying Circus</i> . . . . .	15
Figura 4	– Tela inicial do instalador Python no Microsoft Windows . . . . .	23
Figura 5	– Teste do ambiente Python no Microsoft Windows . . . . .	24
Figura 6	– Tela inicial da IDE IDLE . . . . .	24
Figura 7	– Site oficial do Scratch. . . . .	25
Figura 8	– Tela inicial do Visual Studio Code. . . . .	26
Figura 9	– Aba <i> Extensions </i> do Visual Studio Code com as extensões Python e Pylance instaladas. . . . .	27
Figura 10	– “Hello, World!” em execução no Visual Studio Code . . . . .	30
Figura 11	– “Hello, World!” em execução na plataforma Scratch. . . . .	31
Figura 12	– Exemplo de código-fonte com erro sintático e detecção pelo interpretador. . . . .	33
Figura 13	– Tela do Scratch com a declaração de uma variável <i>a</i> e duas atribuições de valor. . . . .	39
Figura 14	– Estrutura <i>if</i> em Scratch. . . . .	67
Figura 15	– Estrutura <i>if-else</i> em Scratch. . . . .	68
Figura 16	– Estrutura <i>if-elif-else</i> em Scratch. . . . .	69
Figura 17	– Estrutura <i>for</i> em Scratch. . . . .	77
Figura 18	– Exemplo de código-fonte com <i>breakpoints</i> nas linhas 4 e 5 no VS Code. . . . .	87
Figura 19	– Exemplo de código-fonte em depuração com parada em <i>breakpoint</i> no VS Code. . . . .	88
Figura 20	– Ícones de controle de depuração no VS Code. . . . .	89



# Lista de Quadros

1 – Comandos para a instalação do ambiente de desenvolvimento Python em distribuições GNU/Linux por meio dos gerenciadores de pacotes apt e yum . . . . .	25
2 – Principais palavras reservadas na linguagem Python. . . . .	40
3 – Tipos de dados <i>built-in</i> em Python. . . . .	42
4 – Ordem de precedência e associatividade entre as operações aritméticas (da mais alta para a mais baixa). . . . .	49
5 – Funções de conversão de tipos <i>built-in</i> em Python. . . . .	57
6 – Título do quadro. . . . .	60
7 – Tabela-verdade do operador de conjunção (and). . . . .	62
8 – Tabela-verdade do operador de disjunção (or). . . . .	63
9 – Tabela-verdade do operador de negação (not). . . . .	63
10 – Ordem de precedência e associatividade entre operações lógicas (do mais alto para o mais baixo) . . . . .	63
11 – Ordem de precedência entre todos os operadores da linguagem Python. . . . .	64

# Lista de Tabelas

Tabela 1 – Índice TIOBE em junho de 2024 . . . . .	15
--	----

# Lista de abreviaturas e siglas

CMD	<i>Prompt de comando</i>
CLI	<i>Command Line Interface</i>
CLT	Consolidação das Leis do Trabalho
f-strings	<i>Formatted string literals</i>
GNU	GNU's Not Unix!
GUI	<i>Graphical User Interface</i>
IDE	<i>Integrated Development Environment</i>
IDLE	Integrated Development and Learning Environment
IMC	Índice de Massa Corpórea
INSS	Instituto Nacional do Seguro Social
IRPF	Imposto sobre a Renda da Pessoa Física
pip	Package installer for Python
MIT	Massachusetts Institute of Technology
TI	Tecnologia da Informação
VS Code	Visual Studio Code

# Sumário

<b>1</b>	<b>Introdução</b>	<b>12</b>
1.1	Por que aprender a programar?	12
1.2	Linguagem de programação	13
1.3	Como os computadores “entendem” os dados?	13
1.4	Scratch	14
1.5	Python	14
1.6	Resumo do capítulo	16
1.7	Atividades propostas	16
<b>2</b>	<b>Algoritmo</b>	<b>17</b>
2.1	O que é um algoritmo?	17
2.2	Importância de algoritmos na programação	19
2.3	Como construir um algoritmo?	19
2.4	Código-fonte	19
2.5	Resumo do capítulo	20
2.6	Atividades propostas	21
<b>3</b>	<b>Ambiente de desenvolvimento</b>	<b>22</b>
3.1	Preparando o ambiente de desenvolvimento	22
3.2	Instalação do ambiente de desenvolvimento Python	22
3.2.1	Microsoft Windows	23
3.2.2	GNU/Linux	23
3.3	Scratch	25
3.4	Visual Studio Code	26
3.5	Resumo do capítulo	27
3.6	Atividades propostas	28
<b>4</b>	<b>Primeiro programa e o fluxo de execução sequencial</b>	<b>29</b>
4.1	Como executar um programa Python?	29
4.1.1	Execução de programas pelo Visual Studio Code	30
4.1.2	Execução pela CLI	30
4.2	A função <code>print()</code>	31
4.3	Comentários	32
4.3.1	Comentários de uma linha	32
4.3.2	Comentários de múltiplas linhas	32
4.4	Sintaxe	32
4.5	Fluxo de execução sequencial	33
4.6	Exemplos comentados	35
4.7	Resumo do capítulo	36
4.8	Atividades propostas	36
<b>5</b>	<b>Variáveis e tipos de dados</b>	<b>38</b>
5.1	Variáveis	38
5.1.1	Nome de uma Variável	39
5.1.1.1	Regras para nomeação de variáveis	39
5.1.1.2	Palavras Reservadas	39
5.1.1.3	Boas práticas para nomes de variáveis	40
5.1.2	Declaração e atribuição de valores a variáveis	40
5.2	Tipos de Dados	41
5.3	A função <code>type()</code>	42
5.4	Exemplos comentados	42
5.5	Resumo do capítulo	44
5.6	Atividades propostas	44

<b>6</b>	<b>Operadores aritméticos e operador de atribuição</b>	<b>46</b>
6.1	Operações aritméticas	46
6.1.1	Operador de Adição (+)	47
6.1.2	Operador de Subtração (-)	47
6.1.3	Operador de Multiplicação (*)	47
6.1.4	Operador de Divisão (/)	48
6.1.5	Operador de Divisão Inteira (//)	48
6.1.6	Operador de Módulo (%)	48
6.1.7	Operador de Exponenciação (**)	48
6.2	Precedência e associatividade entre operadores aritméticos	49
6.3	Operações de atribuição	49
6.3.1	Operador de atribuição básico	50
6.3.2	Operadores de atribuição combinada	50
6.3.2.1	Operador de Adição e Atribuição (+=)	50
6.3.2.2	Operador de Subtração e Atribuição (-=)	50
6.3.2.3	Operador de Multiplicação e Atribuição (*=)	50
6.3.2.4	Operador de Divisão e Atribuição (/=)	50
6.3.2.5	Operador de Divisão Inteira e Atribuição (//=)	50
6.3.2.6	Operador de Módulo e Atribuição (%=)	51
6.3.2.7	Operador de Exponenciação e Atribuição (**=)	51
6.4	Exemplos comentados	51
6.5	Resumo do capítulo	54
6.6	Atividades propostas	54
<b>7</b>	<b>Interação com o usuário</b>	<b>56</b>
7.1	A função input()	56
7.1.1	Convertendo tipos de dados	56
7.2	f-strings e formatação de valores	57
7.2.1	Formatação de valores int	57
7.2.2	Formatação de valores float	57
7.3	Resumo do capítulo	58
7.4	Atividades propostas	58
<b>8</b>	<b>Operadores relacionais e operadores lógicos</b>	<b>60</b>
8.1	Operadores relacionais	60
8.1.1	Operador igual a	60
8.1.2	Operador diferente de	61
8.1.3	Operador maior que	61
8.1.4	Operador menor que	61
8.1.5	Operador maior ou igual a	61
8.1.6	Operador menor ou igual a	61
8.2	Precedência e associatividade entre operações relacionais	61
8.3	Operadores lógicos	61
8.3.1	Operador de conjunção (and)	62
8.3.2	Operador de disjunção (or)	62
8.3.3	Operador de negação (not)	62
8.3.4	Precedência e associatividade entre operações lógicas	63
8.4	Ordem de precedência entre tipos de operação	63
8.5	Exemplos comentados	64
8.6	Resumo do capítulo	65
8.7	Atividades propostas	65
<b>9</b>	<b>Fluxo de execução condicional</b>	<b>66</b>
9.1	Estruturas condicionais	66
9.1.1	Estrutura if	66
9.1.2	Estrutura if-else	67
9.1.3	Estrutura if-elif-else	68
9.2	Estruturas condicionais aninhadas	69

9.3	Exemplos comentados	70
9.4	Resumo do capítulo	72
9.5	Atividades Propostas	72
<b>10</b>	<b>Fluxo de execução com repetição</b>	<b>76</b>
10.1	Estruturas de Repetição, Laços de Repetição ou <i>Loops</i> ?	76
10.2	Estrutura <i>for</i>	76
10.3	Função <i>range</i>	77
10.4	Estrutura <i>while</i>	78
10.5	Exemplos Comentados	78
10.6	Resumo do capítulo	81
10.7	Atividades propostas	81
<b>A</b>	<b>Interface de Linha de Comando</b>	<b>83</b>
A.1	Utilização da CLI em sistemas operacionais Microsoft Windows	83
A.1.1	Comandos básicos	83
A.1.1.1	Comando <i>cd</i>	83
A.1.1.2	Comando <i>dir</i>	84
A.1.1.3	Comando <i>cls</i>	84
A.1.2	Uso da tecla <i>Tab</i>	84
A.2	Utilização da CLI em sistemas operacionais GNU/Linux	84
A.2.1	Comandos básicos	84
A.2.1.1	Comando <i>cd</i>	84
A.2.1.2	Comando <i>ls</i>	85
A.2.1.3	Comando <i>clear</i>	85
A.2.2	Uso da tecla <i>Tab</i>	85
<b>APÊNDICE B</b>	<b>Depuração</b>	<b>86</b>
B.1	Teste de mesa	86
B.1.1	Como funciona o teste de mesa?	86
B.1.2	Vantagens do teste de mesa	87
B.2	Depuração via software	87
B.2.1	Depuração de Python no VS Code	87
B.2.2	Passo 1: Inserção de <i>breakpoints</i>	87
B.2.3	Passo 2: Iniciar depuração	88
B.2.4	Passo 3: Execução da depuração	88
B.2.5	Executar com depuração ou sem depuração?	89
<b>Referências</b>		<b>90</b>

# Introdução

Neste capítulo, abordamos a importância de aprender a programar e apresentamos as linguagens de programação Python e Scratch.

## 1.1 POR QUE APRENDER A PROGRAMAR?

Antes de começar a programar, é fundamental refletir sobre a pergunta: **“Por que aprender a programar?”**. Se você é um entusiasta ou pretende atuar na área de programação, já conhece a resposta. Contudo, mesmo que esse não seja o seu caso, você pode se beneficiar enormemente com esse conhecimento. Programar é uma habilidade transformadora que promove a criatividade, a resolução de problemas e a inovação. Nos dias de hoje, a programação deixou de ser uma habilidade exclusiva dos especialistas e se tornou essencial no século XXI, conforme apontam diversos pesquisadores e profissionais da Tecnologia da Informação (TI).

Este livro oferece uma abordagem prática e envolvente para aprender os fundamentos da lógica de programação, combinando a simplicidade visual do Scratch com a versatilidade e o poder do Python.

Scratch é uma plataforma de programação visual desenvolvida pelo Massachusetts Institute of Technology (MIT), focada em proporcionar o desenvolvimento de atividades educacionais. Utiliza blocos de código coloridos e encaixáveis, permitindo a criação de programas sem a necessidade de escrever código. Python, por sua vez, é uma poderosa linguagem de programação de propósito geral, amplamente utilizada em diversas áreas, como Análise de Dados, Estatística e Inteligência Artificial.

Ao longo deste livro, você aprenderá a programar e a pensar de forma algorítmica, sendo guiado por conceitos fundamentais como variáveis, estruturas de controle e funções, de maneira gradual e intuitiva. Antes de prosseguir, vamos definir o termo **programação**.

### Definição: Programação.

Programação é o processo de escrever um conjunto de instruções que um computador pode seguir para executar uma tarefa específica. Essas instruções são escritas em uma linguagem de programação, que permite aos desenvolvedores comunicar suas intenções ao computador de maneira clara e precisa. Programar envolve a resolução de problemas, a organização de ideias e a aplicação de lógica para desenvolver soluções eficazes. Os desenvolvedores trabalham em uma variedade de tarefas, desde o desenvolvimento de aplicativos e jogos para dispositivos móveis até a criação de sites e sistemas de gerenciamento de dados.

## 1.2 LINGUAGEM DE PROGRAMAÇÃO

Linguagem de programação é o meio pelo qual o desenvolvedor se comunica com o computador. Em outras palavras, é a ponte que conecta o pensamento humano à lógica computacional, transformando ideias abstratas em programas.

### **Definição:** Linguagem de programação.

Uma linguagem de programação é um conjunto de regras, sintaxes e símbolos que permite aos desenvolvedores escreverem instruções que um computador pode entender e executar. É uma linguagem formal usada para a construção de programas.

As linguagens de programação foram desenvolvidas para facilitar a comunicação entre humanos e máquinas. Assim como diferentes idiomas humanos permitem que as pessoas se comuniquem entre si, diferentes linguagens de programação permitem que desenvolvedores se comuniquem com diferentes tipos de hardware e software.

Ao escrever um programa, você está criando uma série de comandos que instruem o computador sobre o que fazer. Esses comandos são escritos em uma linguagem específica que o computador pode interpretar.

Além de Python e Scratch, existem centenas de linguagens de programação, cada uma com suas características e propósitos, mas todas seguem princípios fundamentais semelhantes. Alguns exemplos de linguagens de programação bem conhecidas são: C, C++, Dart, Java, JavaScript e PHP.

## 1.3 COMO OS COMPUTADORES “ENTENDEM” OS DADOS?

Uma das questões mais intrigantes para quem está começando a aprender sobre computadores e programação é: “Por que dizemos que os computadores entendem os dados em 0s e 1s?” Para responder a essa pergunta, precisamos compreender os fundamentos da lógica binária, da eletrônica digital e como esses conceitos formam a base do funcionamento dos computadores.

Os computadores utilizam o sistema binário, que é um sistema de numeração base-2. Isso significa que ele usa apenas dois dígitos: 0 e 1. Diferentemente do sistema decimal, que usamos no dia a dia e que é base-10 (com dígitos de 0 a 9), o sistema binário é ideal para computadores devido à sua simplicidade e eficiência em termos de hardware.

Para entender por que o sistema binário é utilizado, precisamos explorar a eletrônica digital. Os computadores são constituídos por componentes eletrônicos, como transistores, que podem estar em um de dois estados distintos:

- Ligado (*On*): Representado pelo dígito 1.
- Desligado (*Off*): Representado pelo dígito 0.

Os transistores funcionam como interruptores, controlando o fluxo de corrente elétrica através dos circuitos. Essa capacidade de estar apenas em dois estados distintos torna o sistema binário perfeitamente adequado para representar dados e realizar operações em um computador.

Os dados em um computador são armazenados e processados em forma binária. A menor unidade de dados em um computador é o *bit*, que pode assumir o valor 0 ou 1. Os *bits* são agrupados em grupos de oito para formar *bytes*.

A linguagem de programação facilita a comunicação entre o desenvolvedor e o computador. O computador opera utilizando códigos de máquina que é descrito usando o sistema binário.



## 1.4 SCRATCH

Scratch é uma linguagem de programação visual e uma plataforma on-line desenvolvidas pelo grupo Lifelong Kindergarten do MIT em 2007. A linguagem utiliza uma interface visual simples, baseada no uso de blocos. Segundo a página oficial do projeto, Scratch é uma das maiores comunidades de programação para crianças no mundo.

A linguagem permite que o usuário crie histórias, jogos e animações visuais, além de possibilitar o aprendizado de lógica de programação de forma intuitiva e visual. O logo da linguagem é apresentado na [Figura 1](#).

Figura 1 – Logo da linguagem de programação Scratch.



Fonte: [Fundação Scratch \(2024\)](#).

## 1.5 PYTHON

Python é uma linguagem de programação desenvolvida pelo matemático e programador holandês Guido van Rossum em 1991. Os códigos-fonte<sup>1</sup> escritos na linguagem se destacam pela sintaxe simples e legibilidade, o que a torna uma boa escolha tanto para desenvolvedores experientes quanto para iniciantes. Python é uma das linguagens de programação mais populares e versáteis do mundo. A linguagem é multiplataforma, interpretada, de alto nível e de propósito geral. Sua capacidade de ser utilizada em diversas áreas, desde desenvolvimento web até Ciência de Dados e Inteligência Artificial, faz dela uma das linguagens mais usadas globalmente. As principais características da linguagem são:

**Multiplataforma** Um programa em Python pode ser executado em diferentes ambientes, incluindo a maioria dos sistemas operacionais das famílias Microsoft Windows, GNU/Linux e macOS.

**Interpretada** Python é uma linguagem interpretada, o que significa que o código é executado linha por linha pelo interpretador.

**Alto nível** Uma linguagem de alto nível abstrai fortemente os detalhes do controle do computador, tornando a tarefa de programar mais simplificada em comparação com linguagens de médio e baixo nível.

**Propósito geral** A linguagem Python pode ser usada em diversas áreas, como Ciência de Dados, Inteligência Artificial e desenvolvimento web.

Outras características de Python incluem: programação funcional, uso de indentação<sup>2</sup> para definição de blocos, orientação a objetos e vasta disponibilidade de bibliotecas e módulos.

A [Figura 2](#) apresenta o logotipo da linguagem Python. Observando atentamente, é possível identificar que ele é formado por duas serpentes, uma azul e uma amarela. É interessante notar que, apesar da referência, o nome da linguagem não é uma alusão às serpentes do gênero píton.

<sup>1</sup> Código-fonte é um conjunto de instruções escritas em uma linguagem de programação que um desenvolvedor cria para desenvolver um software. O [Capítulo 2](#) apresenta o conceito em maiores detalhes.

<sup>2</sup> Indentação é um neologismo derivado do inglês *indentation*. O termo é muito utilizado em programação e significa utilizar recuos no início das linhas de código-fonte para facilitar a leitura e a compreensão.

O nome da linguagem é, na verdade, uma homenagem ao programa humorístico britânico *Monty Python's Flying Circus*, que era um dos favoritos de Guido van Rossum. Uma foto dos atores do seriado é apresentada na [Figura 3](#).

Figura 2 – Logotipo da linguagem de programação Python.



Fonte: [Python.org \(2024\)](#).

Figura 3 – Atores de *Monty Python's Flying Circus*.



Fonte: [History of the BBC \(1969\)](#).

Python é amplamente reconhecida como uma das linguagens de programação mais usadas no mundo. No momento da escrita deste livro, a linguagem estava em primeiro lugar no índice TIOBE<sup>3</sup>. A [Tabela 1](#) apresenta as 10 linguagens mais populares segundo esse índice.

Tabela 1 – Índice TIOBE em junho de 2024

Posição	Linguagem de programação	Frequência
#1	Python	16,12%
#2	C++	10,34%
#3	C	9,48%
#4	Java	8,59%
#5	C#	6,72%
#6	JavaScript	3,79%
#7	Go	2,19%
#8	Visual Basic	2,08%
#9	Fortran	2,05%
#10	SQL	2,04%

Fonte: Adaptado de [TIOBE \(2024\)](#).

<sup>3</sup> Índice de popularidade das linguagens de programação produzido pela empresa TIOBE Software com base na frequência de pesquisa nos principais motores de busca da Web.

## 1.6 RESUMO DO CAPÍTULO

### Pontos-chave

**Linguagem de programação** Linguagem escrita e formal que apresenta um conjunto de regras e instruções usadas para o desenvolvimento de softwares.

**Programação** Processo de desenvolvimento de um software.

**Python** Linguagem de programação textual multiparadigma, multiplataforma, de propósito geral e de alto nível.

**Scratch** Linguagem de programação visual voltada a fins educacionais.

## 1.7 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Explique com suas palavras o que é uma linguagem de programação. Cite ao menos três exemplos de linguagens de programação.

**Questão 2** Qual é a definição de programação? Em sua opinião, por que é importante aprender a programar no século XXI?

**Questão 3** Por que dizemos que os computadores entendem “Os e Is”? Qual é a importância de uma linguagem de programação nesse contexto?

### Exercícios complementares

**Questão 1** Quais são as cinco linguagens de programação mais populares atualmente? Consulte a *web* para verificar o índice TIOBE.

**Questão 2** Compare e contraste as linguagens de programação Python e Scratch, destacando suas principais características e propósitos. Explique com suas palavras.

# Algoritmo

Lara era uma estudante entusiasmada, mas completamente nova no mundo da programação. Sentada em frente ao computador, ela sentia um misto de empolgação e nervosismo. Ela já tinha ouvido falar sobre algoritmos, mas não sabia ao certo o que eles eram.

“Por onde eu começo?”, perguntou Lara.

## 2.1 O QUE É UM ALGORITMO?

Antes de entender os detalhes de como os algoritmos funcionam, é importante compreender o que são.

O dicionário Michaelis apresenta algumas definições de algoritmo:

1 MATEMÁTICA OBSOLETO Sistema de notação aritmética com algarismos arábicos.

2 MATEMÁTICA Processo de cálculo que, por meio de uma sequência finita de regras, raciocínios e operações aplicadas a um número finito de dados, leva à resolução de grupos análogos de problemas.

3 MATEMÁTICA Operação ou processo de cálculo; sequência de etapas articuladas que produz a solução de um problema; procedimento sequenciado que leva ao cumprimento de uma tarefa.

4 LÓGICA Conjunto das regras de operação (conjunto de raciocínios) cuja aplicação permite resolver um problema enunciado por meio de um número finito de operações; pode ser traduzido em um programa executado por um computador, detectável nos mecanismos gramaticais de uma língua ou no sistema de procedimentos racionais finito utilizado em outras ciências, para a resolução de problemas semelhantes.

5 INFORMÁTICA Conjunto de regras, operações e procedimentos, definidos e ordenados, usados na solução de um problema ou de uma classe de problemas, em um número finito de etapas ([ALGORITMO](#), 2024).

Todas essas definições remetem à “resolução de problemas”. Simplificando, *um algoritmo é um conjunto de instruções precisas e ordenadas que levam à resolução de um problema ou à execução de uma tarefa.*

### Definição: Algoritmo.

Um algoritmo é um conjunto de instruções precisas e ordenadas que levam à resolução de um problema ou à execução de uma tarefa.

Imagine que Lara está fazendo uma receita de bolo. Cada etapa que ela segue, desde a mistura dos ingredientes até a colocação no forno, é um algoritmo. Se Lara seguir esses passos corretamente, ela possivelmente fará um bolo delicioso. Da mesma forma, algoritmos na programação são usados para resolver problemas e executar tarefas.

O ser humano executa algoritmos o tempo todo no cotidiano. Construir e executar um algoritmo não depende do uso de um computador. Imagine agora que Lara está dirigindo e o pneu do seu veículo furou. Pare e reflita:

- Qual é o problema de Lara?
- Quais passos ela deve seguir para solucionar o problema?

**Exemplo: Algoritmo para trocar o pneu furado de um automóvel.**

Desenvolver um algoritmo para trocar o pneu furado de um automóvel.

**Solução**

1. Separar o estepe, o macaco e as demais ferramentas necessárias.
2. Posicionar o macaco embaixo do veículo e próximo ao pneu furado.
3. Suspender o veículo.
4. Retirar o pneu furado.
5. Instalar o estepe.
6. Baixar o veículo.
7. Guardar o pneu furado, o macaco e as demais ferramentas usadas.

Lara está passando por mais um contratempo. Descobriu que uma lâmpada da sua residência queimou. Como seria um algoritmo para trocar uma lâmpada queimada?

**Exemplo: Algoritmo para trocar uma lâmpada queimada.**

Desenvolver um algoritmo para trocar uma lâmpada queimada.

**Solução**

1. Separar uma lâmpada nova.
2. Pegar uma escada, cadeira ou banco se não conseguir alcançar a lâmpada queimada.
3. Retirar a lâmpada queimada.
4. Instalar a lâmpada nova.
5. Guardar a escada, cadeira ou banco, se utilizados.

As estratégias para trocar o pneu furado de um automóvel e para trocar uma lâmpada queimada têm muito em comum. Ambas foram construídas sob uma perspectiva algorítmica.

Você pode ter pensado em soluções diferentes das apresentadas nos exemplos anteriores. Note que isso não significa que a solução pensada está incorreta. É possível encontrar diferentes soluções válidas para um mesmo problema. Em outras palavras, diferentes algoritmos podem ser usados para alcançar o mesmo objetivo.

## 2.2 IMPORTÂNCIA DE ALGORITMOS NA PROGRAMAÇÃO

Os algoritmos são a base de toda a programação. Eles representam a forma como os desenvolvedores instruem os computadores sobre o que fazer. Sem algoritmos, os computadores seriam incapazes de executar tarefas ou resolver problemas de maneira eficiente.

Os algoritmos são essenciais para a resolução de problemas computacionais e são utilizados em todas as áreas da Ciência da Computação, desde o desenvolvimento de software até a Análise de Dados e a Inteligência Artificial.

Compreender como os algoritmos funcionam é fundamental para se tornar um programador habilidoso. Ao entender os princípios por trás dos algoritmos, é possível escrever códigos mais eficientes, resolver problemas complexos e criar uma variedade de softwares.

## 2.3 COMO CONSTRUIR UM ALGORITMO?

Criar um algoritmo pode parecer assustador no início, mas é um processo bastante simples. Abaixo estão algumas etapas básicas que você pode seguir:

**Entenda o problema.** Antes de começar a escrever um algoritmo, é fundamental entender completamente o problema que você está tentando resolver. Isso inclui identificar as entradas, saídas e quaisquer restrições ou condições especiais.

**Divida o problema em etapas menores.** Uma vez que você compreendeu o problema, divida-o em etapas menores e mais gerenciáveis. Isso tornará o processo de resolução do problema mais fácil e organizado.

**Escreva os passos de cada etapa.** Agora é o momento de escrever os passos específicos que você precisa seguir para resolver o problema. Certifique-se de que cada passo seja claro, conciso e fácil de entender.

**Atenção:** Dois ou mais algoritmos para o mesmo problema.

A maioria dos problemas ou tarefas admite mais de um algoritmo como solução. Portanto, é perfeitamente possível construir diferentes algoritmos que produzam o mesmo resultado.

## 2.4 CÓDIGO-FONTE

Existem diferentes formas de expressar algoritmos. A forma apresentada na [seção 2.2](#) é conhecida como linguagem natural. Em programação, algoritmos são implementados em *código-fonte*.

Os algoritmos são essenciais para a programação e servem como base para toda a tecnologia moderna. Eles permitem que os computadores realizem tarefas complexas e resolvam problemas de maneira eficiente. Compreender os princípios básicos dos algoritmos e como implementá-los em código-fonte o preparará para se tornar um programador habilidoso e criar soluções inovadoras.

**Definição:** Código-fonte.

Código-fonte é um conjunto de instruções escritas em uma linguagem de programação específica que formam um software. O código-fonte é um texto ou esquema visual legível por seres humanos que define o comportamento e a lógica de um programa. Quando um programador cria um código-fonte, ele define como o computador deve se comportar para executar o programa. Fazendo uma analogia com a construção civil, o código-fonte

pode ser comparado à planta de um edifício, enquanto o programa em si é o edifício construído.

É importante entender que código-fonte e algoritmo são conceitos distintos. Enquanto o código-fonte é a implementação específica de um algoritmo em uma linguagem de programação, o algoritmo é uma abstração mais geral e independente da linguagem.

Vamos considerar o problema de calcular o dobro de um número real. De forma matemática, podemos pensar no algoritmo como uma função  $f(x) = 2x$ , onde  $x$  é um número real e  $f(x)$  é o dobro desse número. O algoritmo escrito em Python é apresentado na [Programa 1](#). O mesmo algoritmo em Scratch é mostrado na [Programa 2](#). Não se preocupe em entender os comandos e instruções agora.

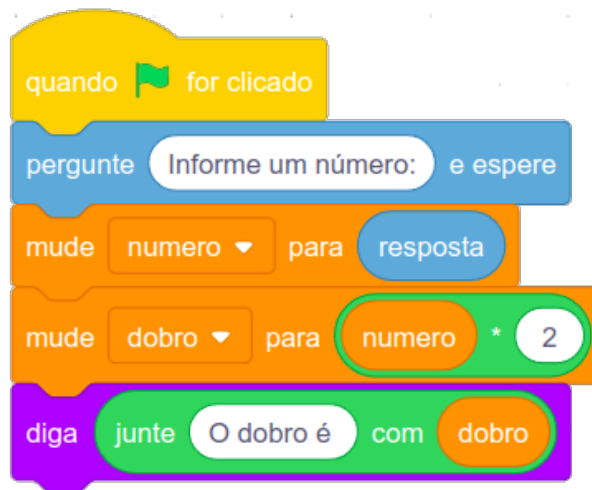
---

**Programa 1** Dobro de um número em Python.

```
1 num = float(input("Informe um número: "))
2 dobro = num * 2
3 print(dobro)
```

---

---

**Programa 2** Dobro de um número em Scratch.

## 2.5 RESUMO DO CAPÍTULO

### Resumo

**Algoritmo** Sequência ordenada de passos para resolver um problema.

**Código-fonte** Um conjunto de instruções escritas por programadores em uma linguagem de programação. Essas instruções são interpretadas ou compiladas para criar um programa executável, que realiza tarefas específicas no computador.

## 2.6 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Qual das seguintes afirmações melhor descreve um algoritmo?

- (a) Um algoritmo é um software que executa uma única tarefa específica.
- (b) Um algoritmo é uma sequência de instruções ordenadas que resolvem um problema ou realizam uma tarefa.
- (c) Um algoritmo é um dispositivo físico usado para armazenar e processar dados.
- (d) Um algoritmo é uma linguagem de programação popular.

**Questão 2** Por que os algoritmos são importantes na programação?

**Questão 3** Escreva um algoritmo para escovar os dentes.

### Exercícios complementares

**Questão 1** Escreva um algoritmo para fazer um suco de limão.

**Questão 2** Escreva um algoritmo para fazer café filtrado.

**Questão 3** Escreva um algoritmo para lavar louças.



# Ambiente de desenvolvimento

## 3.1 PREPARANDO O AMBIENTE DE DESENVOLVIMENTO

O mundo da programação é vasto e complexo, repleto de termos técnicos e conceitos que podem parecer intimidantes para iniciantes. No entanto, entender o básico é essencial para qualquer programador. Uma das primeiras tarefas é preparar o ambiente de programação.

Um ambiente de desenvolvimento é um conjunto de ferramentas que permite ao desenvolvedor criar softwares que serão executados pelos computadores. Quando um conjunto abrangente de ferramentas e um editor de código-fonte são fornecidos simultaneamente, esse ambiente é denominado IDE (do inglês *Integrated Development Environment*).

### Definição: IDE.

IDE é um conjunto de ferramentas e recursos que auxiliam os programadores na criação de software. Um ambiente de desenvolvimento integrado geralmente inclui tudo o que é necessário para escrever, compilar, depurar e testar código de maneira eficiente e organizada. Uma IDE típica inclui os seguintes componentes:

**Editor de código-fonte** Ferramenta onde os desenvolvedores escrevem e editam seu código-fonte. Oferece recursos como destaque de sintaxe, autocompletar, formatação automática e realce de erros de sintaxe, o que ajuda a melhorar a legibilidade e a organização do código.

**Compilador ou interpretador** Plataforma que "converte" o código-fonte em código de máquina, permitindo que o computador execute as instruções.

**Depurador** Ferramenta essencial para encontrar e corrigir erros no código-fonte. Um depurador permite a inspeção do código instrução por instrução, facilitando a identificação e correção de problemas.

## 3.2 INSTALAÇÃO DO AMBIENTE DE DESENVOLVIMENTO PYTHON

Python é compatível com vários sistemas operacionais modernos, especialmente Microsoft Windows, GNU/Linux e macOS. Antes de instalar, é importante identificar o sistema operacional do computador. A maioria das distribuições GNU/Linux já vem com Python instalado por padrão. No entanto, isso não é comum em sistemas operacionais Microsoft Windows e macOS.

O ambiente de desenvolvimento Python pode ser obtido no site oficial [python.org](https://python.org). O site oferece também documentação, notícias, comunidades e outros recursos úteis para desenvolvedores.

A instalação típica do ambiente de desenvolvimento inclui: interpretador, depurador, bibliotecas padrão, instalador de pacotes e IDE.

### 3.2.1 Microsoft Windows

Para instalar Python no Microsoft Windows, siga os passos abaixo:

1. Acesse a página de *download* no site [python.org](https://python.org) e baixe o instalador Python compatível com o seu computador. No momento da escrita deste livro, a versão mais recente é a 3.12.4. Sempre que possível, utilize a versão mais recente, a menos que haja problemas de compatibilidade com o seu sistema ou projetos.
2. Localize e execute o arquivo baixado para iniciar a instalação. Na primeira tela, como mostrado na [Figura 4](#), marque a opção `Add python.exe to PATH` e clique em `Install Now`.

Figura 4 – Tela inicial do instalador Python no Microsoft Windows



Fonte: Elaborada pelo autor.

3. Teste se o ambiente está funcionando abrindo o *prompt* de comando do Windows e digitando: `py`. O comando deve executar o interpretador da linguagem. O resultado deve ser semelhante ao apresentado na [Figura 5](#). Para sair do interpretador Python, digite o comando: `exit()`.

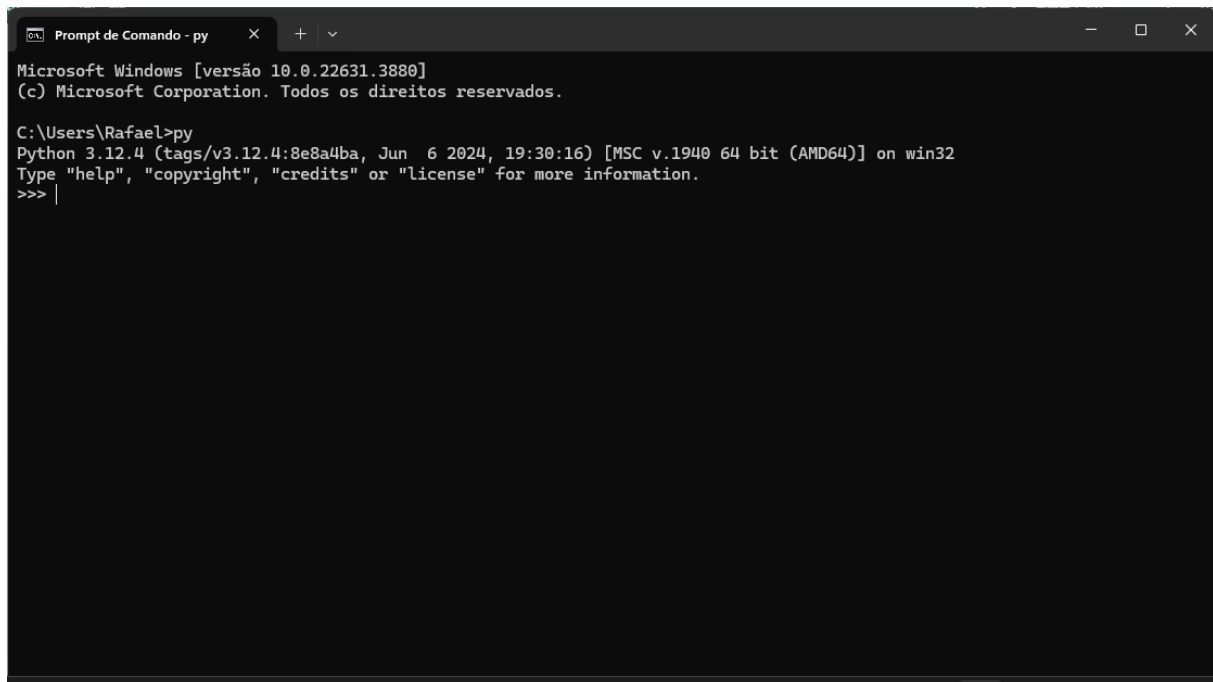
A configuração padrão da instalação do Python inclui o gerenciador de pacotes pip e também a IDE IDLE (do inglês *Integrated Development and Learning Environment*). A tela inicial da IDE é apresentada na [Figura 6](#).

Sugerimos a utilização do Visual Studio Code (VS Code) durante seus estudos. Entretanto, é perfeitamente possível utilizar a IDE IDLE ou qualquer outro editor de código-fonte compatível com a linguagem.

### 3.2.2 GNU/Linux

A maioria das distribuições GNU/Linux modernas já vem com o ambiente de desenvolvimento Python instalado por padrão. Caso não seja o caso, você pode instalar o Python diretamente a partir do gerenciador de pacotes do sistema operacional.

Figura 5 – Teste do ambiente Python no Microsoft Windows

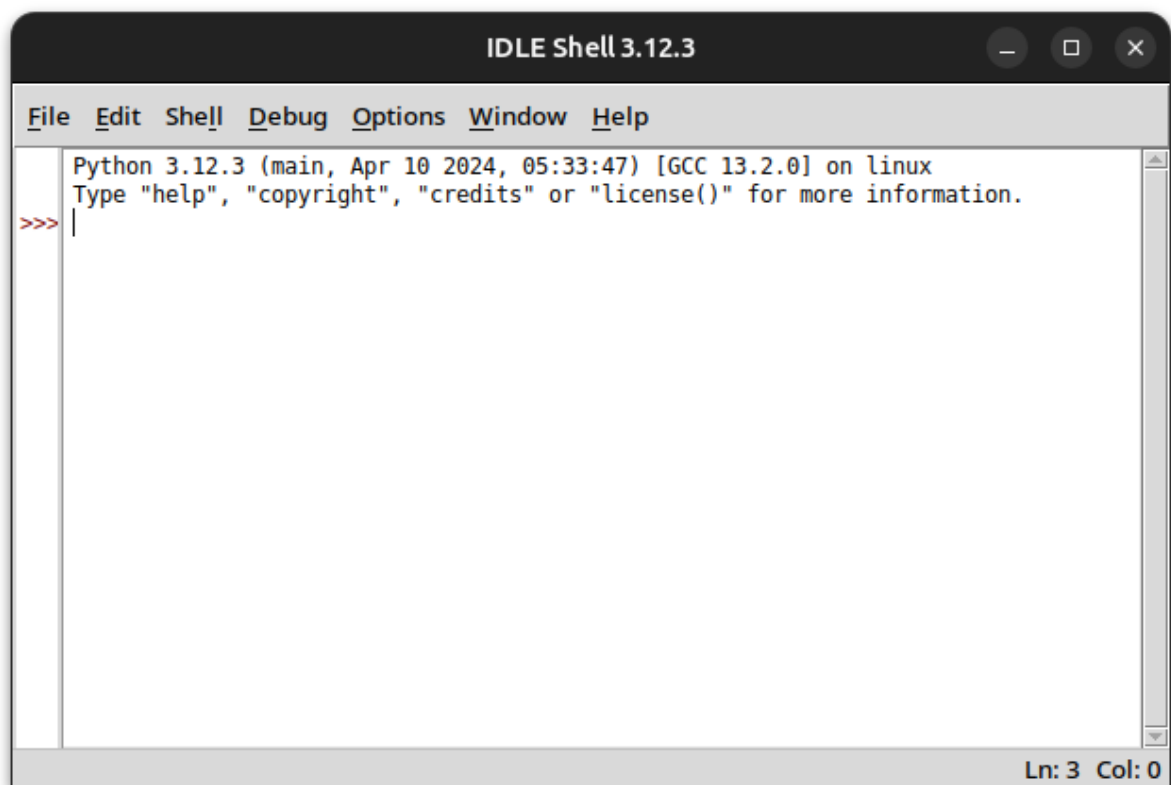


```
Prompt de Comando - py
Microsoft Windows [versão 10.0.22631.3880]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Rafael>py
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Fonte: Elaborada pelo autor.

Figura 6 – Tela inicial da IDE IDLE



```
IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
Python 3.12.3 (main, Apr 10 2024, 05:33:47) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
Ln: 3 Col: 0
```

Fonte: Elaborada pelo autor.

O [Quadro 1](#) apresenta os comandos para instalação utilizando os gerenciadores de pacotes apt (Debian, Ubuntu e derivados) e yum (RedHat, Fedora, CentOS e derivados).

Quadro 1 – Comandos para a instalação do ambiente de desenvolvimento Python em distribuições GNU/Linux por meio dos gerenciadores de pacotes apt e yum

Gerenciador de pacotes	Comando
apt	<code>sudo apt-get install python3</code>
yum	<code>sudo yum install python3</code>

Fonte: Elaborado pelo autor.

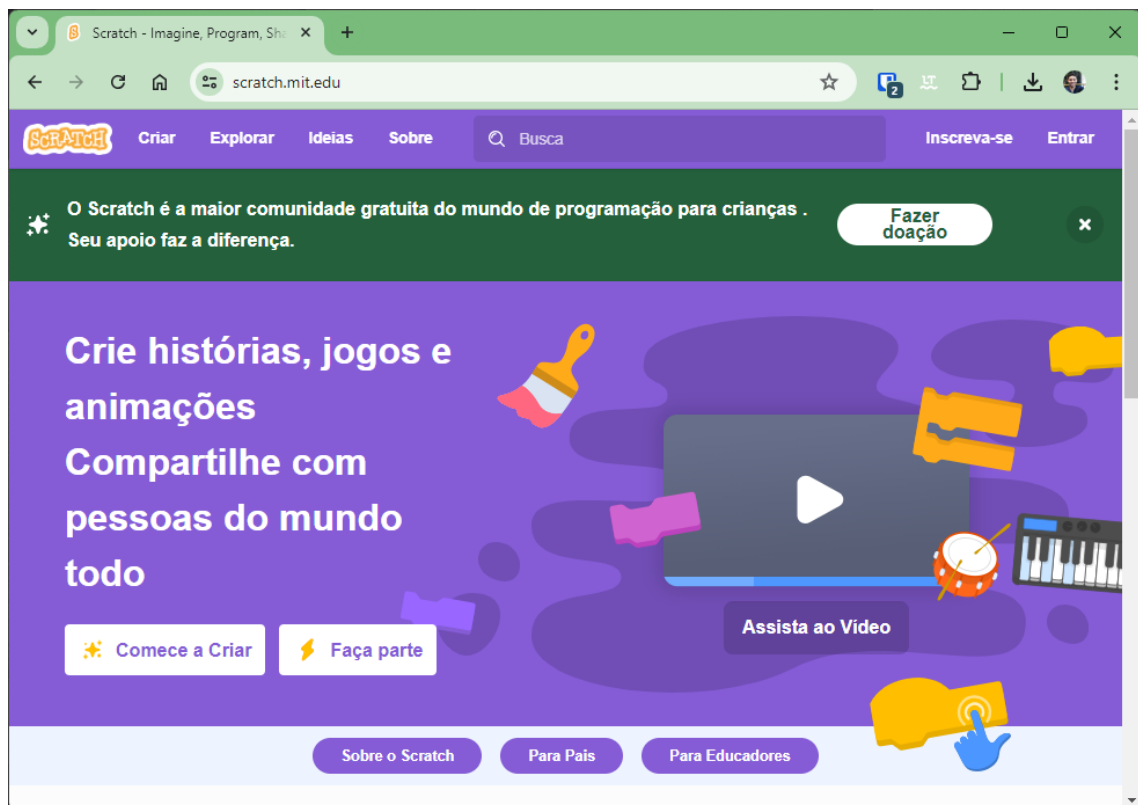
O ambiente de desenvolvimento Python é geralmente acompanhado do pip. Caso a distribuição não tenha o pacote instalado, você pode instalá-lo com o comando correspondente ao seu gerenciador de pacotes, substituindo `python3` por `python3-pip`.

No GNU/Linux, o interpretador Python é executado pelo comando `python3`. O teste de execução produz um resultado semelhante ao obtido em um sistema operacional Microsoft Windows (veja a Figura 5).

### 3.3 SCRATCH

O ambiente desenvolvimento Scratch está disponível no site oficial [scratch.mit.edu](https://scratch.mit.edu), veja a Figura 7. O ambiente pode ser utilizado diretamente por qualquer navegador compatível e é necessário criar uma conta de usuário. Alternativamente, é possível instalar o ambiente de desenvolvimento para execução direta no computador local. O instalador pode ser obtido em [scratch.mit.edu/download](https://scratch.mit.edu/download).

Figura 7 – Site oficial do Scratch.



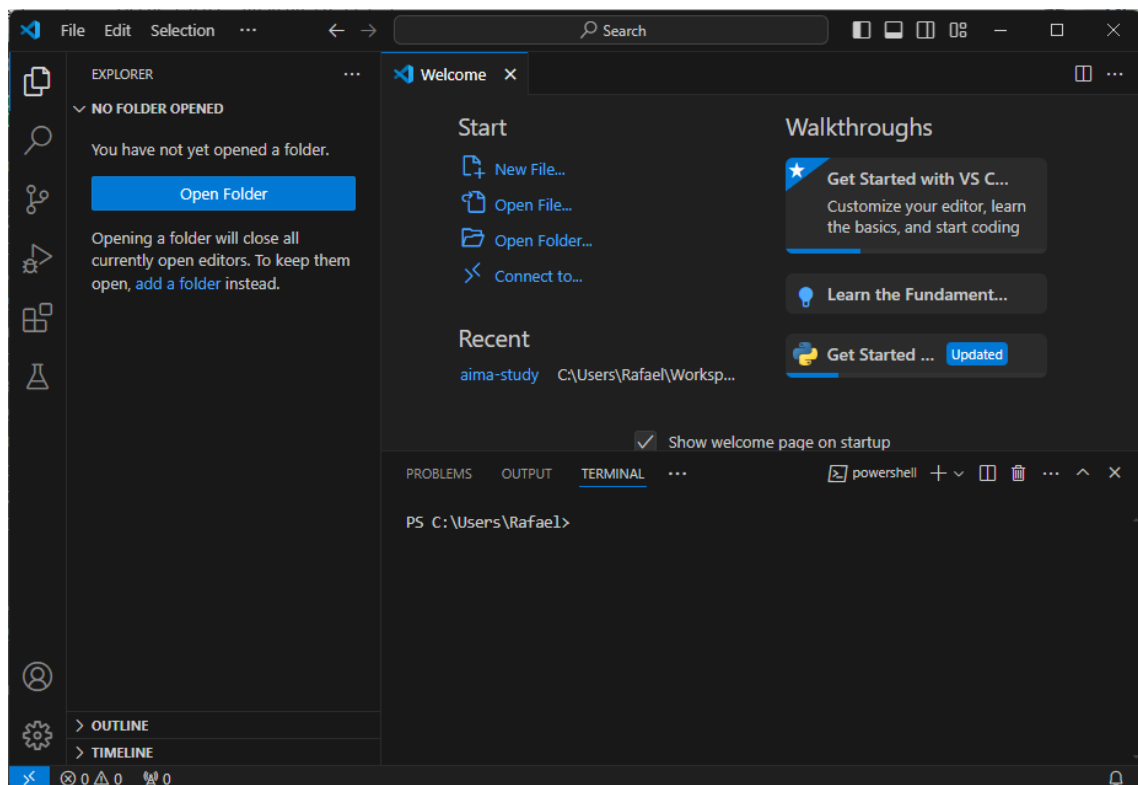
Fonte: Elaborada pelo autor.

## 3.4 VISUAL STUDIO CODE

Visual Studio Code, frequentemente abreviado como VS Code, é um editor de código-fonte desenvolvido pela Microsoft. É gratuito, de código aberto e multiplataforma, disponível para Microsoft Windows, macOS e GNU/Linux. O software pode ser baixado em [code.visualstudio.com](https://code.visualstudio.com).

O VS Code suporta uma ampla variedade de linguagens de programação e é reconhecido por sua robustez, leveza e vasta gama de recursos e ferramentas, proporcionando uma experiência de desenvolvimento eficiente. A Figura 8 apresenta a tela inicial do VS Code. É possível instalar uma tradução para deixar a interface do software em português do Brasil.

Figura 8 – Tela inicial do Visual Studio Code.

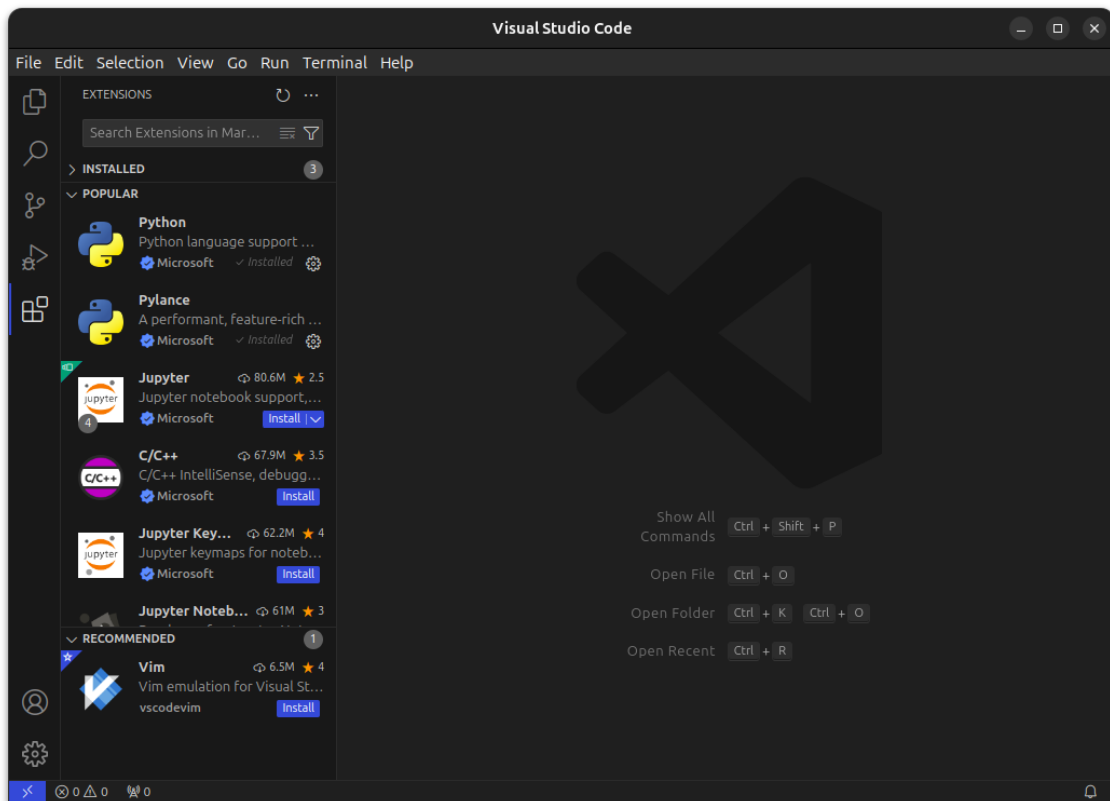


Fonte: Elaborada pelos autores.

Para aprimorar o suporte do VS Code à linguagem Python, é recomendado instalar as extensões *Python* e *Pylance*. Essas extensões proporcionam funcionalidades avançadas, como IntelliSense e análise de código, facilitando o desenvolvimento em Python. A Figura 9 mostra a aba *Extensions* do VS Code com as extensões *Python* e *Pylance* instaladas.

A instalação de extensões no VS Code é simplificada por meio da aba *Extensions*. Para instalar uma extensão, siga os passos abaixo:

1. Abra o VS Code e clique no ícone *Extensions* na barra lateral esquerda.
2. Na barra de pesquisa, digite o nome da extensão desejada, como “Python” ou “Pylance”.
3. Selecione a extensão na lista de resultados e clique na opção *Install*.
4. Após a instalação, a extensão estará pronta para uso, oferecendo suporte adicional e funcionalidades aprimoradas para o desenvolvimento em Python.

Figura 9 – Aba |*Extensions*| do Visual Studio Code com as extensões Python e Pylance instaladas.

Fonte: Elaborada pelos autores.

### 3.5 RESUMO DO CAPÍTULO

#### Pontos-chave

**Ambiente de desenvolvimento Python** Inclui o interpretador Python, a IDE IDLE, e o gerenciador de pacotes pip. O Python pode ser instalado em diversos sistemas operacionais, como Microsoft Windows, GNU/Linux e macOS, e pode ser testado por meio do *prompt* de comando ou terminal.

**Editor de código-fonte** Ferramenta que permite escrever, editar e gerenciar o código-fonte de softwares. Diferencia-se de editores de texto tradicionais por oferecer recursos avançados, como destaque de sintaxe, autocompletar e ferramentas de depuração, tornando a programação mais dinâmica e eficiente.

**IDE** Sigla para *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado, em português). É um software que oferece um conjunto de ferramentas e recursos para facilitar o desenvolvimento de software. Uma IDE típica inclui um editor de código-fonte, compilador ou interpretador e um depurador.

**Visual Studio Code** Editor de código-fonte desenvolvido pela Microsoft, gratuito e de código aberto. Suporta múltiplas linguagens de programação e é aprimorado com extensões como *Python* e *Pylance*, que oferecem funcionalidades avançadas para o desenvolvimento em Python.

## 3.6 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** A instalação do ambiente de desenvolvimento Python independe do sistema operacional utilizado no computador?

- (a) Certo.
- (b) Errado.

**Questão 2** Visual Studio Code é um sofisticado editor de código-fonte utilizado exclusivamente para Python?

- (a) Certo.
- (b) Errado.

**Questão 3** Não é possível executar programas em Scratch localmente?

- (a) Certo.
- (b) Errado.

### Exercícios complementares

**Questão 1** Explique com suas palavras o que é uma IDE (do inglês *Integrated Development Environment*).

**Questão 2** Pesquise na web por editores de código-fonte que suportem a linguagem Python. Cite ao menos três.

**Questão 3** Há alguma plataforma on-line que permita programar em Python diretamente pelo navegador? Pesquise na web e forneça exemplos.

# Primeiro programa e o fluxo de execução sequencial

A expressão “Hello, World!” é um jargão comum entre desenvolvedores ao aprender uma nova linguagem de programação. Em português, seria “Oi, Mundo!”. O objetivo é criar um programa extremamente simples que apenas exiba essa mensagem na linha de comando (CLI, do inglês *Command Line Interface*). Isso ajuda iniciantes a se familiarizarem com a sintaxe básica da linguagem e a configurarem seu ambiente de desenvolvimento.

O código do “Hello, World!” em Python é apresentado no [Programa 3](#), e o mesmo algoritmo em Scratch é apresentado no [Programa 4](#).

---

**Programa 3** “Hello, World!” em Python.

---

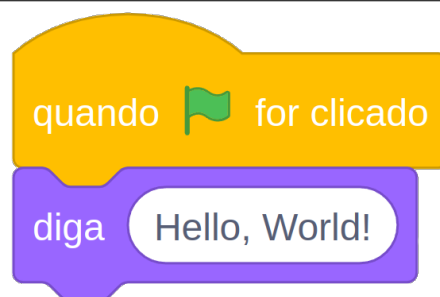
```
1 print("Hello, World!")
```

---

---

**Programa 4** “Hello, World!” em Scratch.

---



## 4.1 COMO EXECUTAR UM PROGRAMA PYTHON?

Existem várias maneiras de executar um programa Python. Na maioria delas, é necessário criar um arquivo de texto com a extensão \*.py contendo o código-fonte. O arquivo deve ser executado pelo interpretador.

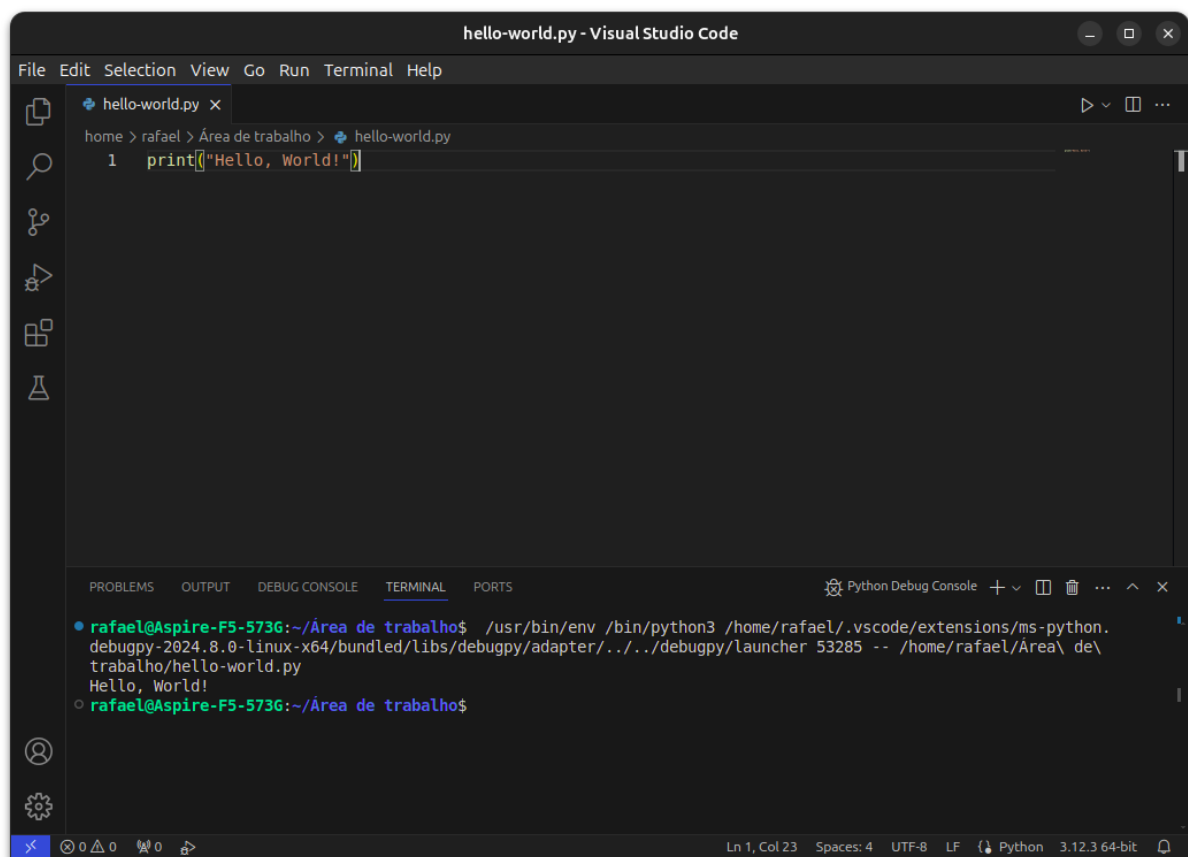


### 4.1.1 Execução de programas pelo Visual Studio Code

Depois de configurar o Visual Studio Code (VS Code) e selecionar o interpretador Python, siga estes passos para criar e executar seu programa Python:

1. **Criar um novo arquivo.** Vá ao menu `|File|` e selecione `|New File|` ou pressione `<Ctrl+N>`. Digite seu código no novo arquivo, por exemplo, copie o código-fonte do [Programa 3](#).
2. **Salvar o arquivo.** Para salvar, selecione `|Save As|` no menu `|File|` ou pressione `<Ctrl+Shift+S>`. Nomeie o arquivo com a extensão `*.py`, por exemplo, `hello-world.py`.
3. **Executar o programa.** Escolha `|Run Without Debugging|` no menu `|Run|` ou pressione `<Ctrl+F5>`. A execução será exibida em uma aba de CLI, conforme mostrado na [Figura 10](#).

Figura 10 – “Hello, World!” em execução no Visual Studio Code



Fonte: Elaborada pelo autor.

### 4.1.2 Execução pela CLI

Para executar um programa Python via CLI (Command Line Interface), siga estas etapas:

1. **Abrir o terminal ou *prompt* de comando.** No Microsoft Windows, abra o *prompt* de comando; no GNU/Linux ou macOS, abra o terminal.
2. **Navegar até o diretório do arquivo:** Use comandos de CLI para mudar para o diretório onde o arquivo do código-fonte está salvo.
3. **Executar o arquivo.**

- No *prompt* de comando do Microsoft Windows, digite:

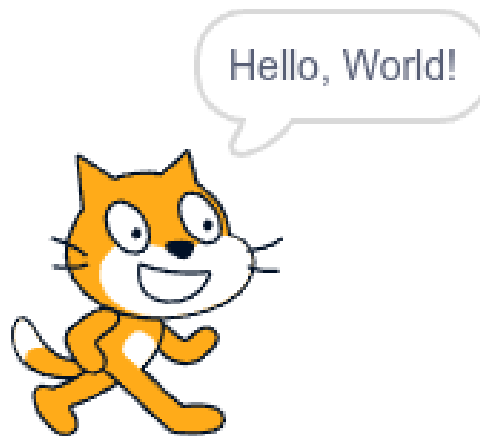
```
py hello-world.py
```

- No terminal do GNU/Linux, digite:

```
python3 hello-world.py
```

O resultado da execução do código em Python será a mensagem “Hello, World!” exibida no terminal ou *prompt* de comando. O resultado da execução do [Programa 4](#) na plataforma Scratch é apresentado na [Figura 11](#).

Figura 11 – “Hello, World!” em execução na plataforma Scratch.



Fonte: Elaborada pelos autores.

## 4.2 A FUNÇÃO PRINT()

A função `print()` é uma função embutida (*built-in*) em Python que exibe informações na tela. Ela imprime uma ou mais expressões, convertendo-as para texto (do tipo *string*) e enviando a mensagem resultante para a tela da CLI (Command Line Interface). A função `print()` é equivalente ao bloco “diga” da linguagem Scratch (veja o [Programa 4](#)) e sua execução é comparável ao balão de diálogo mostrado na [Figura 11](#).

A sintaxe básica da função `print()` requer que as *strings* estejam entre aspas simples (') ou aspas duplas(" "). No [Programa 3](#), o texto está entre aspas duplas.

A função `print()` pode receber mais de um argumento. Os argumentos passados em uma função são separados por vírgulas. Cada argumento é convertido para *string* e concatenado com um espaço entre eles. Veja o exemplo no [Programa 5](#). A concatenação das *strings* "Hello," e "World!" resulta no mesmo resultado produzido pelo [Programa 3](#).

---

**Programa 5** “Hello, World!” com dois argumentos em Python.

---

```
1 print("Hello, ", "World!")
```

---

Por padrão, a função `print()` utiliza um espaço como separador entre os argumentos e pula uma linha ao final do texto. Esses comportamentos podem ser alterados com os parâmetros nomeados `sep` e `end`, que definem, respectivamente, o caractere separador e o caractere final. A documentação da função está disponível em [documentação da função print](#).

## 4.3 COMENTÁRIOS

Comentários são trechos de texto no código-fonte que são ignorados pelo interpretador. Eles são usados para explicar o que o código-fonte faz, melhorando a legibilidade. Python suporta comentários de uma linha e de múltiplas linhas.

### 4.3.1 Comentários de uma linha

Comentários de uma linha começam com o caractere `#` e vão até o final da linha. Tudo após o `#` é ignorado pelo interpretador. Veja um exemplo no [Programa 6](#). Na linha 1, o comentário é ignorado pelo interpretador. A linha 2, por estar em branco, também é ignorada. Já na linha 3, o texto após o `#` é desconsiderado.

**Programa 6** “Hello, World!” com comentário de linha única em Python

```
1 # Comentário de única linha e linha em branco
2
3 print("Hello, World!") # Função que imprime mensagem de saída
```

### 4.3.2 Comentários de múltiplas linhas

Python não possui uma sintaxe específica para comentários de múltiplas linhas, mas *strings* de múltiplas linhas (três aspas simples ou duplas) são frequentemente usadas para esse propósito. Essas *strings* são ignoradas pelo interpretador se não estiverem atribuídas a uma variável. Veja o exemplo de comentário de múltiplas linhas no [Programa 7](#).

**Programa 7** “Hello, World!” com comentário de múltiplas linhas em Python.

```
1 """
2 Comentário de múltiplas linhas.
3 Geralmente usado para documentação ou
4 para explicar seções maiores do código-fonte
5 """
6 print("Hello, World!")
```

## 4.4 SINTAXE

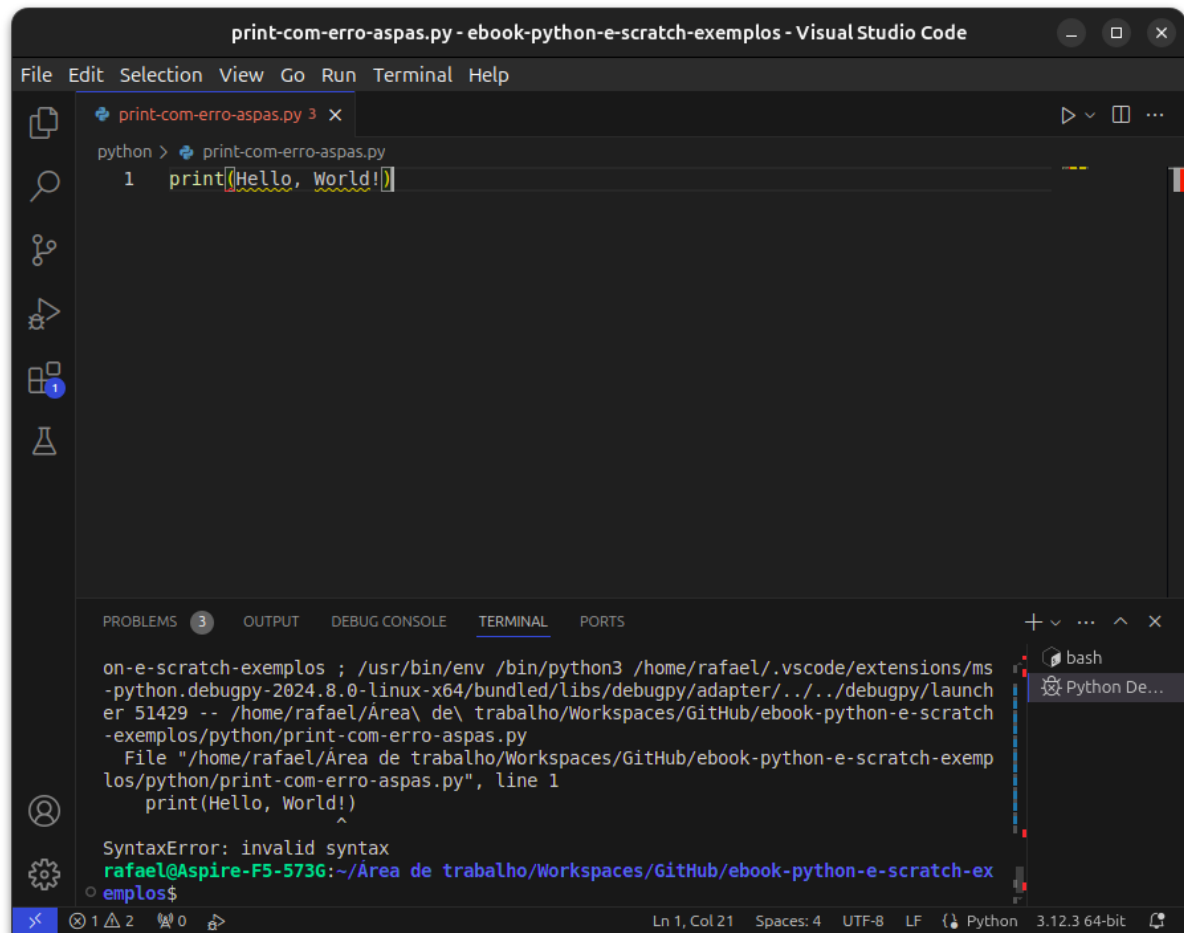
A sintaxe é um conjunto de regras que define a estrutura e a organização correta do código-fonte em uma linguagem de programação. O interpretador de Python analisa o código-fonte para verificar se ele está de acordo com essas regras. Caso alguma regra de sintaxe não seja atendida, um erro será gerado durante a execução do código.

VS Code oferece uma funcionalidade adicional ao detectar alguns erros de sintaxe antes da execução do código. Isso permite aos desenvolvedores identificar e corrigir problemas de sintaxe antes que o código seja executado, economizando tempo e evitando frustrações.

Para garantir que o código esteja correto e funcional, é crucial prestar atenção às mensagens de erro fornecidas pelo interpretador e pelo editor. Erros de sintaxe podem incluir problemas como falta de parênteses, aspas não fechadas ou indentação incorreta. Seguir as regras de sintaxe da linguagem Python ajuda a evitar esses erros.

Um exemplo de código-fonte com erro sintático e a mensagem de erro informada pelo compilador é apresentado na [Figura 12](#). Este exemplo ilustra como um erro de sintaxe pode ser identificado e corrigido.

Figura 12 – Exemplo de código-fonte com erro sintático e detecção pelo interpretador.



Fonte: Elaborada pelos autores.

#### Atenção: Erro semântico.

Embora um código-fonte possa executar sem apresentar erros de sintaxe, isso não garante que ele esteja correto do ponto de vista lógico. Erros semânticos, que são falhas na lógica do algoritmo, não são detectados pelo interpretador e podem ser difíceis de identificar.

## 4.5 FLUXO DE EXECUÇÃO SEQUENCIAL

O Capítulo 2 define algoritmo como uma sequência finita de passos para resolver problemas e realizar tarefas. Ao construir soluções algorítmicas, estabelecemos uma ordem específica na qual esses passos devem ser executados. Esse conceito é conhecido como **fluxo de execução**.

A forma mais simples de organizar esses passos é através do **fluxo de execução sequencial**, onde cada passo é executado na ordem em que aparece no algoritmo. Em Python, isso significa que cada linha de código é executada uma após a outra, seguindo a sequência em que estão dispostas. O interpretador executa a primeira linha, depois a segunda, e assim sucessivamente. Da mesma forma, no Scratch, os blocos de código são executados de maneira sequencial. Veja o exemplo a seguir para compreender como o fluxo de execução sequencial funciona.

**Exemplo:** Lista dos cinco países com maior extensão territorial.

Implementar um programa que imprima uma lista dos cinco países com maior extensão territorial.

**Solução** Os países com maior extensão territorial são, em ordem: Rússia, Canadá, China, Estados Unidos e Brasil. Implemente um programa que imprima essa lista de países, de forma que seja apresentado um país por linha.

Uma solução para esse exemplo em Python é apresentada no [Programa 8](#). O interpretador Python executa cada linha de código em sequência, começando pela impressão de “Rússia” até “Brasil”. Quando o código é digitado em um arquivo \*.py no VS Code, o editor exibe números de linha de 1 a 5. A execução segue exatamente essa sequência.

**Programa 8** Lista dos cinco países com maior extensão territorial em Python.

```
1 print("Rússia")
2 print("Canadá")
3 print("China")
4 print("Estados Unidos")
5 print("Brasil")
```

A implementação do mesmo algoritmo em Scratch é apresentada no [Programa 9](#). Observe que o fluxo de execução sequencial segue o encaixe dos blocos de cima para baixo, garantindo que cada instrução seja executada na ordem apresentada. O bloco diga \_ por \_ segundos é utilizado para simular a exibição sequencial dos balões de diálogo, imitando o comportamento de nova linha na CLI.

**Programa 9** Lista dos cinco países com maior extensão territorial em Scratch.

## 4.6 EXEMPLOS COMENTADOS

### Exemplo: Exibir uma mensagem de uma linha.

Implementar um programa que exiba a mensagem “Bem-vindo ao mundo da programação!”.

#### Solução:

```
1 print("Bem-vindo ao mundo da programação!")
```

Para imprimir uma mensagem na tela, usamos a função `print()`. Esta função é fundamental para apresentar informações ao usuário. No exemplo, a função `print()` recebe uma *string* como argumento. A mensagem contida nesta *string* é exibida na CLI quando o programa é executado. O código apresentado faz exatamente isso: utiliza a função `print()` para mostrar a mensagem “Bem-vindo ao mundo da programação!” ao usuário.

### Exemplo: Exibir uma mensagem de múltiplas linhas.

Implementar um programa que exiba o seguinte texto, garantindo que a saída reproduza o texto a seguir, seguindo a formatação e o número de linhas.

Cidades mais populosas do Brasil:

- São Paulo
- Rio de Janeiro
- Brasília

#### Solução 1:

```
1 print("Cidades mais populosas do Brasil:")
2 print("- São Paulo")
3 print("- Rio de Janeiro")
4 print("- Brasília")
```

Nesta solução, cada linha do texto é impressa usando um comando `print()` separado. A função `print()` adiciona automaticamente uma quebra de linha após cada chamada, garantindo que o texto seja exibido conforme solicitado, com cada linha começando em uma nova linha na saída da CLI.

#### Solução 2:

```
1 print("Cidades mais populosas do Brasil:\n- São Paulo\n- Rio de\n      Janeiro\n- Brasília")
```

Nesta solução, o caractere especial “\n” é utilizado para inserir quebras de linha dentro da *string*. Cada “\n” faz com que o texto após ele seja exibido em uma nova linha na saída da CLI. Ao passar a *string* completa para a função `print()`, o texto é exibido exatamente com a formatação e o número de linhas desejados.

#### Solução 3:

```
1 print("""Cidades mais populosas do Brasil:
2 - São Paulo
3 - Rio de Janeiro
4 - Brasília""")
```

Para imprimir um texto com múltiplas linhas e formatação específica, a função `print()` pode ser utilizada com quebras de linha incorporadas na *string*. Nesta solução, a função

`print()` é usada para exibir cada linha do texto como é apresentado. As quebras de linha são feitas automaticamente quando a mensagem contém quebras de linha no texto entre aspas triplas, como mostrado no código. As aspas triplas permitem que o texto seja escrito em múltiplas linhas, preservando a formatação original, incluindo quebras de linha e indentação.

## 4.7 RESUMO DO CAPÍTULO

### Pontos-chave

**Fluxo de execução sequencial** Refere-se à execução de instruções em uma ordem específica, uma após a outra, como estão dispostas no código. Em Python e em Scratch, o fluxo de execução sequencial garante que cada linha ou bloco de código seja processado na sequência correta.

**Função `print`** Utilizada para exibir informações na tela. A função `print()` é fundamental para a saída de texto e pode receber múltiplos argumentos, que serão impressos separados por espaços. A função pode ser usada com *strings* literais e caracteres especiais como “\n” para quebras de linha, ou com aspas triplas para textos multilinha.

**Sintaxe** Define as regras de estrutura do código que devem ser seguidas para que o programa seja compreendido e executado corretamente. Erros de sintaxe são detectados pelo interpretador durante a execução ou por ferramentas de desenvolvimento, como editores de código. É essencial seguir as regras de sintaxe para evitar interrupções na execução do programa.

**Semântica** Refere-se ao significado do código e à lógica do algoritmo. Erros semânticos ocorrem quando o código está sintaticamente correto, mas a lógica do programa leva a resultados inesperados. Depuração e testes são necessários para identificar e corrigir erros semânticos.

## 4.8 ATIVIDADES PROPOSTAS

**Exercícios de fixação**

**Questão 1** Implementar um programa que imprima o texto “Olá, mundo!”.

**Questão 2** Implementar um programa que imprima o seu nome completo.

**Questão 3** Qual a saída resultante da execução da linha de código a seguir?

```
print("Hello") #World!
```

**Questão 4** Implementar um programa que imprima o texto a seguir, reproduzindo a formatação:

Lista de compras:

- Arroz
- Feijão
- Óleo

**Questão 5** Qual a diferença entre erro sintático e erro lógico no contexto de programação?

**Exercícios complementares**

**Questão 1** O que significa dizer que o fluxo de execução de um código-fonte é sequencial? Explique com suas palavras.

**Questão 2** O código-fonte a seguir está correto sintaticamente? Caso contrário, explique o que está errado.

```
print(Alan Turing)
```

**Questão 3** O código-fonte a seguir tem como resultado a saída:

```
print("1", "2", "3", "4")
```

- (a) “1234”
- (b) “1 2 3 4”
- (c) Erro de sintaxe.



# Variáveis e tipos de dados

Neste capítulo, exploraremos como os dados são armazenados e representados em um algoritmo.

## 5.1 VARIÁVEIS

Até agora, nossos programas se restringiram à implementação de algoritmos básicos de impressão. No entanto, para criar algoritmos mais complexos, é crucial entender como trabalhar com dados. Mas por que isso é importante? Computadores são projetados para armazenar e processar dados, e entender como manipular esses dados por meio de algoritmos é fundamental. Em algoritmos, os dados são representados por **variáveis** e **literais**.

**Definição:** Variável e literal.

**Variável** Um espaço de armazenamento identificado por um nome simbólico, que pode conter diferentes valores durante a execução de um programa. As variáveis são locais reservados na memória do computador para armazenar dados que podem ser modificados e acessados em várias partes do código.

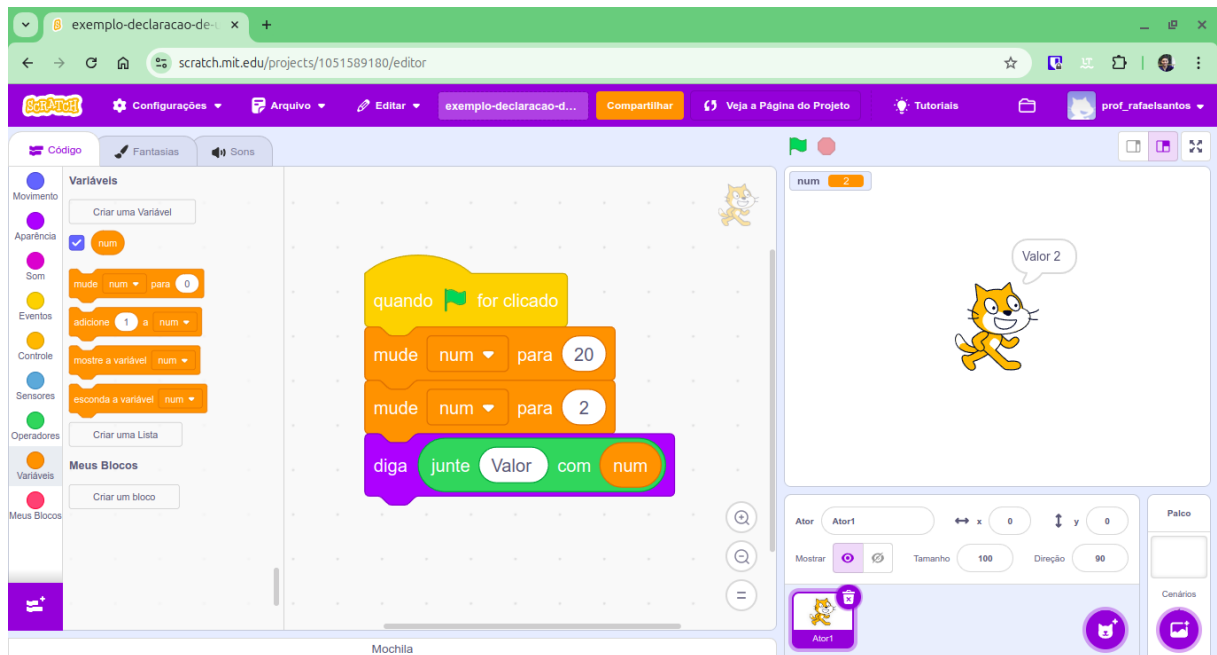
**Literal** Um valor diretamente representado no código-fonte de um programa. Literais geralmente são números, *strings* ou valores booleanos, explicitamente escritos e reconhecidos como constantes. Eles representam valores fixos que não mudam durante a execução do programa.

Variáveis podem ser comparadas às incógnitas em equações matemáticas e literais a constantes. No entanto, ao contrário de uma incógnita, que representa um valor a ser descoberto em uma equação, uma variável pode ter seu valor alterado ao longo da execução do programa. A flexibilidade das variáveis é crucial na programação, permitindo a alteração e o gerenciamento dinâmico dos dados durante a execução do algoritmo.

Você não precisa declarar o tipo de uma variável explicitamente em Python. A linguagem é de tipagem dinâmica, isso significa que o interpretador deduz o tipo com base no valor atribuído.

Uma analogia útil para entender uma variável é pensar nela como uma “caixa”, um “espaço de armazenamento” que guarda um valor ao longo da execução do programa. A [Figura 13](#) ilustra um código-fonte em Scratch com uma variável `num` e duas atribuições de valor. Veja que analogia é visualmente ilustrada.

Figura 13 – Tela do Scratch com a declaração de uma variável a e duas atribuições de valor.



Fonte: Elaborada pelo autor.

### 5.1.1 Nome de uma Variável

Um programa pode ter múltiplas variáveis, cada uma identificada por um nome exclusivo escolhido pelo programador. É essencial seguir certas regras ao nomear variáveis para garantir que o código seja legível e funcione corretamente. Vamos explorar essas regras e boas práticas.

#### 5.1.1.1 Regras para nomeação de variáveis

- **Início do nome:** O nome de uma variável deve começar com uma letra (a-z, A-Z) ou um subtraço (\_).
- **Caracteres válidos:** Após o primeiro caractere, o nome pode incluir letras, números (0-9) e subtraços.
- **Restrições:** O nome de uma variável não pode começar com um número e deve evitar o uso de palavras reservadas da linguagem de programação.

Alguns exemplos de nomes válidos e inválidos:

- **Válidos:** idade, \_total, valor1, nome\_usuario
- **Inválidos:** 1valor (não é permitido iniciar o nome com número), total! (exclamação é caractere inválido), class (porque é uma palavra reservada)

#### 5.1.1.2 Palavras Reservadas

Palavras reservadas são termos que têm um significado especial na linguagem e definem a sua sintaxe e gramática. Por isso, não podem ser usadas como nomes de variáveis.

Quadro 2 – Principais palavras reservadas na linguagem Python.

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

Fonte: Elaborado pelos autores.

### 5.1.1.3 Boas práticas para nomes de variáveis

#### Alerta: Boas práticas para nomes de variáveis.

**Case-sensitive** Python diferencia letras maiúsculas de minúsculas. Por exemplo, `Numero` é uma variável diferente de `numero`. Tenha cuidado com a consistência no uso de maiúsculas e minúsculas para evitar erros inesperados.

**Letras do Inglês** Embora a linguagem Python suporte caracteres do idioma português, é uma boa prática utilizar apenas letras do alfabeto inglês para nomear variáveis. Isso melhora a legibilidade e a portabilidade do código.

**Nomes descritivos** Uma boa prática é utilizar nomes que remetam diretamente ao que a variável armazena. Nomes genéricos prejudicam a legibilidade do código-fonte.

### 5.1.2 Declaração e atribuição de valores a variáveis

A criação de uma variável é chamada de declaração. Em Python, a declaração de variáveis é direta e não requer a especificação do tipo da variável. Isso é feito através do uso do **operador de atribuição** (`=`). O [Capítulo 6](#) apresenta em maiores detalhes o funcionamento desse operador.

Veja o exemplo do [Programa 10](#). A variável `num` é declarada na linha 1, pois é a primeira vez que ela aparece no código. Após a declaração, o valor 20 é atribuído à variável. Na linha 2, o valor da variável é atualizado para 2 utilizando o operador de atribuição. Finalmente, o comando `print()` é utilizado para exibir o valor armazenado na variável, que é 2.

#### Programa 10 Declaração de uma variável e impressão do valor armazenado.

```

1 num = 20      # Declaração da variável num e atribuição do valor 20
2 num = 2       # Atribuição do valor 2 à variável num
3 print("Valor: ", num) # Impressão de num

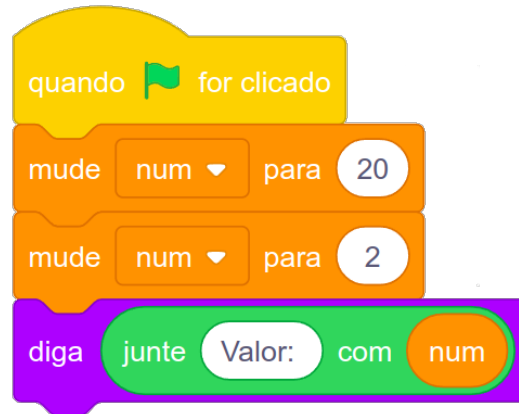
```

Uma variável é criada na primeira atribuição de valor. Em atribuições subsequentes, o valor armazenado é atualizado. É possível usar quantas variáveis forem necessárias, e o valor de uma variável só é alterado após uma operação de atribuição.

O código-fonte em Scratch equivalente ao código-fonte do [Programa 10](#) é apresentado no [Programa 11](#). O bloco `mude _ para _` é equivalente ao operador de atribuição.

O exemplo do [Programa 12](#) ilustra a declaração de duas variáveis, `num1` e `num2`, que armazenam respectivamente os valores 4 e 7. Além disso, o código imprime esses valores usando a função `print()`. Observe que é possível combinar a impressão de *strings* com os valores das variáveis em um único comando. O código-fonte equivalente em Scratch é apresentado no [Programa 13](#).

**Programa 11** Declaração de uma variável e impressão do valor armazenado em Scratch.

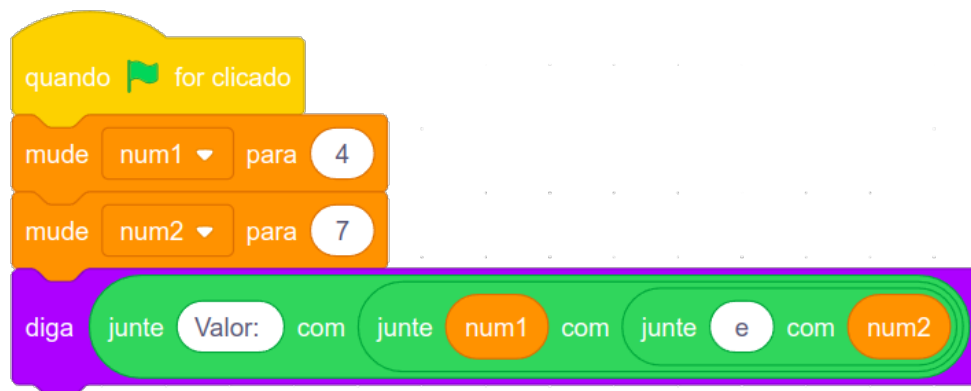


**Programa 12** Declaração de duas variáveis e impressão dos valores armazenados.

```

1 num1 = 4 # Declaração da variável num1 e atribuição do valor 4
2 num2 = 7 # Declaração da variável num2 e atribuição do valor 7
3 print("Valores:", num1, "e", num2) # Impressão de num1 e num2
  
```

**Programa 13** Declaração de duas variáveis e impressão dos valores armazenado em Scratch.



**Alerta:** Operador de atribuição e operador de igualdade.

Um erro comum entre iniciantes é confundir o operador de atribuição com o operador de igualdade. Embora o símbolo seja o mesmo, o operador de atribuição (`=`) é usado para modificar o valor de uma variável, enquanto o operador de igualdade (`==`) é utilizado para comparar valores.

## 5.2 TIPOS DE DADOS

Dados representados por variáveis e literais podem ser de diferentes tipos em um algoritmo. A maioria das linguagens de programação oferece suporte a uma ampla variedade de tipos, que podem ser primitivos ou compostos. Tipos primitivos são os mais simples e são suportados de forma padrão pela linguagem. Em Python, esses tipos são chamados de *built-in*. Tipos compostos, por outro lado, são estruturados a partir dos tipos primitivos e permitem a criação de estruturas de dados mais complexas e organizadas.

Os principais tipos *built-in* na linguagem Python estão listados no [Quadro 3](#). Aqui estão algumas particularidades:

- O tipo *string* é denominado **str**. Literais do tipo *string* devem sempre ser delimitados por aspas.
- Os literais do tipo *bool* são sempre **True** ou **False**.
- No tipo *complex*, a parte imaginária é representada por **j**, diferentemente do que é comum na Matemática, onde utiliza-se **i**.

Quadro 3 – Tipos de dados *built-in* em Python.

Nome	Tipo	Descrição	Exemplo
<code>bool</code>	Booleano	Assume apenas dois valores: verdadeiro ( <b>True</b> ) e falso ( <b>False</b> ).	<code>a = True</code>
<code>complex</code>	Complexo	Representa números complexos, com a parte imaginária denotada por “j”.	<code>a = 2 + 3j</code>
<code>float</code>	Decimal	Representa números decimais, usando ponto (“.”) como separador decimal.	<code>a = 2.5</code>
<code>int</code>	Inteiro	Representa números inteiros.	<code>a = 2</code>
<code>str</code>	<i>String</i>	Cadeia de caracteres. Os valores devem ser sempre delimitados por aspas duplas ou simples.	<code>a = "Turing"</code>
<code>None</code>	Nulo	Representa a ausência de valor.	<code>a = None</code>

Fonte: Elaborado pelos autores.

**Alerta:** Separador decimal em valores *float*.

Em programação, o separador decimal para números é o ponto (“.”). Isto é diferente da convenção utilizada em português, onde se usa a vírgula (“,”).

### 5.3 A FUNÇÃO `TYPE()`

A tipagem de dados em Python é dinâmica. Isso significa que o tipo de dado é determinado automaticamente com base no valor atribuído à variável ou no literal utilizado.

A função `type()` em Python é usada para verificar o tipo de dados de uma variável ou expressão em um determinado momento. Ela retorna o tipo do objeto passado como argumento. Isso é útil para depuração, validação de entrada e compreensão do comportamento do seu código. A documentação da função está disponível em [documentação da função `type`](#).

O [Programa 14](#) apresenta um exemplo de uso da função `type()`. O tipo de dado é dinamicamente determinado de acordo com o tipo do literal atribuído à variável.

### 5.4 EXEMPLOS COMENTADOS

**Exemplo:** Valor e tipo de dado ao longo da execução do código-fonte.

Qual é a saída da função `print()` nas linhas 2, 3, 5 e 6?

```

1 num = 25
2 print("Valor:", num)
3 print("Tipo:", type(num))
4 num = 25.0

```

**Programa 14** Uso da função `type()`.

```
1 a = 10
2 print(type(a)) #class 'int'>
3
4 b = 3.14
5 print(type(b)) #class 'float'>
6
7 c = "Alan Turing"
8 print(type(c)) #class 'str'>
9
10 d = True
11 print(type(d)) #class 'bool'>
```

```
5 print("Valor:", num)
6 print("Tipo:", type(num))
```

**Solução:**

- **Linha 2:** Valor: 25
- **Linha 3:** Tipo: <class 'int'>
- **Linha 5:** Valor: 25.0
- **Linha 6:** Tipo: <class 'float'>

Na primeira impressão, a variável `num` contém o valor 25 do tipo `int`. Após a segunda atribuição, a variável `num` contém o valor 25.0 do tipo `float`. O tipo é atualizado dinamicamente quando um outro valor é atribuído à variável.

**Exemplo:** Case-sensitive.

Qual é a saída da função `print()` na linha 3?

```
1 nome = "Alan Turing"
2 Nome = "Guido van Rossum"
3 print(nome)
```

**Solução:**

- **Linha 3:** Alan Turing

A variável `nome` foi atribuída ao valor `"Alan Turing"`, e é essa variável que é impressa na linha 3. A variável `Nome` é diferente de `nome` devido à distinção entre maiúsculas e minúsculas (*case-sensitive*) em Python, portanto, não afeta a saída da função `print()` na linha 3.

**Exemplo:** Nome de variável inválido.

Qual é a saída da função `print()` na linha 2?

```
1 valor-2 = 2
2 print(valor-2)
```

**Solução:**

- Saída: `SyntaxError`

A linha `valor-2 = 2` contém um erro de sintaxe. Em Python, o nome de uma variável não pode conter caracteres especiais como o hífen ( `-` ). Portanto, o código resultará em um erro de sintaxe. Para corrigir o erro, você deve usar um nome de variável válido, como `valor2` ou `valor_2`.

## 5.5 RESUMO DO CAPÍTULO

**Pontos-chave**

**Atribuição** Atribuir um valor a uma variável é feito usando o operador de atribuição ( `=` ). A declaração da variável ocorre na primeira atribuição e pode ser modificada posteriormente.

**Função `type`** A função `type()` retorna o tipo do objeto passado como argumento, facilitando a verificação do tipo de dados durante a execução do programa.

**Tipo `bool`** Tipo de dado que representa valores booleanos, podendo ser `True` ou `False`.

**Tipo `float`** Tipo de dado que representa números decimais, usando o ponto ( `.` ) como separador decimal.

**Tipo `int`** Tipo de dado que representa números inteiros, sem casas decimais.

**Tipo `str`** Tipo de dado que representa cadeias de caracteres, delimitadas por aspas duplas ou simples. O tipo `str` é o tipo `string` de Python.

**Variável** Um espaço de armazenamento identificado por um nome simbólico, que pode conter diferentes valores durante a execução de um programa. O valor da variável pode ser alterado ao longo do tempo.

## 5.6 ATIVIDADES PROPOSTAS

**Exercícios de fixação**

**Questão 1** Qual é a saída da função `print()` nas linhas 2, 3, 5 e 6?

```
flag = True
print("Valor:", flag)
print("Tipo:", type(flag))
flag = "Hello, World!"
print("Valor:", flag)
print("Tipo:", type(flag))
```

**Questão 2** Classifique os nomes das variáveis como válidos ou inválidos.

(I) `1num`

(II) `_num`

(III) num\_

(IV) def

(V) nu-m

**Questão 3** Qual é o valor e o tipo da variável Num na linha 3?

```
Num = 4.5  
num = 4  
print(Num)
```

### Exercícios complementares

**Questão 1** O que significa dizer que a linguagem Python é *case-sensitive*?

**Questão 2** Complete as lacunas da afirmação a seguir: “\_\_\_ é um tipo de dado que representa cadeias de caracteres. Valores literais desse tipo devem sempre ser expressos entre \_\_\_.”

**Questão 3** O que são palavras reservadas em uma linguagem de programação?



# Operadores aritméticos e operador de atribuição

Neste capítulo, construiremos programas capazes de armazenar e realizar operações sobre dados. Vamos explorar os operadores aritméticos e operador de atribuição disponíveis em Python, que são fundamentais para executar cálculos matemáticos básicos.

## 6.1 OPERAÇÕES ARITMÉTICAS

Os operadores aritméticos em Python são utilizados para realizar operações matemáticas essenciais. O [Programa 15](#) ilustra o uso de todos os operadores aritméticos em um exemplo prático. Observe atentamente os valores de entrada e o resultado de cada operação para entender como cada operador se comporta.

---

**Programa 15** Exemplo com todos os operadores aritméticos aplicados a tipos *built-in*.

---

```
1 a = 10
2 b = 3
3
4 soma = a + b           # 13
5 subtracao = a - b      # 7
6 multiplicacao = a * b   # 30
7 divisao = a / b         # 3.3333...
8 divisao_inteira = a // b # 3
9 modulo = a % b          # 1
10 exponenciacao = a ** b # 1000
11
12 print("Soma:", soma)
13 print("Subtração:", subtracao)
14 print("Multiplicação:", multiplicacao)
15 print("Divisão:", divisao)
16 print("Divisão inteira:", divisao_inteira)
17 print("Resto da divisão:", modulo)
18 print("Exponenciação:", exponenciacao)
```

---

O [Programa 16](#) é a implementação em Scratch equivalente ao [Programa 15](#).

A seguir, apresentamos cada operador com uma explicação detalhada e exemplos de uso.

**Programa 16** Exemplo com todos os operadores aritméticos em Scratch.**6.1.1** Operador de Adição ( + )

O operador de adição é utilizado para somar dois valores.

```
resultado = 5 + 3 # O resultado é 8
```

Neste exemplo, o operador + soma os valores 5 e 3, resultando em 8.

**6.1.2** Operador de Subtração ( - )

O operador de subtração é utilizado para subtrair um valor de outro.

```
resultado = 5 - 3 # O resultado é 2
```

Neste exemplo, o operador - subtrai o valor 3 de 5, resultando em 2.

**6.1.3** Operador de Multiplicação ( \* )

O operador de multiplicação é utilizado para multiplicar dois valores.

```
resultado = 5 * 3 # O resultado é 15
```

Neste exemplo, o operador `*` multiplica os valores 5 e 3, resultando em 15.

#### 6.1.4 Operador de Divisão (`/`)

O operador de divisão é utilizado para dividir um valor por outro. O resultado é um número de ponto flutuante.

```
resultado = 5 / 2 # O resultado é 2.5
```

Neste exemplo, o operador `/` divide o valor 5 por 2, resultando em 2.5.

#### 6.1.5 Operador de Divisão Inteira (`//`)

O operador de divisão inteira realiza uma divisão e retorna o quociente sem a parte decimal.

```
resultado = 5 // 2 # O resultado é 2
```

Neste exemplo, o operador `//` divide o valor 5 por 2 e retorna apenas a parte inteira do resultado, que é 2.

#### 6.1.6 Operador de Módulo (`%`)

O operador de módulo, também conhecido como resto da divisão, retorna o resto da divisão inteira de dois valores.

```
resultado = 5 % 2 # O resultado é 1
```

Neste exemplo, o operador `%` calcula o resto da divisão de 5 por 2, que é 1.

#### 6.1.7 Operador de Exponenciação (`**`)

O operador de exponenciação, também conhecido como operador de potenciação, é utilizado para elevar um valor a uma determinada potência.

```
resultado = 5 ** 2 # O resultado é 25
```

Neste exemplo, o operador `**` eleva o valor 5 à potência de 2, resultando em 25.

Além de ser usado para exponenciação, o operador `**` também pode ser utilizado para realizar operações de radiciação. A radiciação é a operação inversa da exponenciação. Por exemplo, a raiz quadrada de 9 pode ser expressa como  $\sqrt{9} = 9^{\frac{1}{2}} = 3$ . De maneira semelhante, a raiz cúbica de 27 pode ser expressa como  $\sqrt[3]{27} = 27^{\frac{1}{3}} = 3$ . O [Programa 17](#) apresenta a implementação dessas operações utilizando o operador de exponenciação.

---

**Programa 17** Radiciação utilizando o operador de exponenciação em Python.

---

```
1 num1 = 9
2 num2 = 27
3 raiz_quadrada = num1 ** (1/2) # Raiz quadrada
4 raiz_cubica = num2 ** (1/3) # Raiz cúbica
5 print("Raiz quadrada:", raiz_quadrada)
6 print("Raiz cúbica:", raiz_cubica)
```

---

## 6.2 PRECEDÊNCIA E ASSOCIATIVIDADE ENTRE OPERADORES ARITMÉTICOS

A precedência dos operadores determina a ordem na qual as operações são realizadas. Em Python, a precedência das operações aritméticas segue a mesma lógica usada na Matemática, conforme ilustrado no [Quadro 4](#).

Observe que a associatividade entra em jogo quando há operadores do mesmo nível. Com exceção da exponenciação, os operadores do mesmo nível são avaliados na ordem em que aparecem na linha de código, ou seja, da esquerda para a direita. Por exemplo, em uma expressão que contém múltiplas operações de multiplicação e divisão, elas serão executadas da esquerda para a direita, na ordem em que aparecem.

É importante notar que os parênteses não são um operador em si, mas sim um recurso utilizado para modificar a ordem de precedência das operações. Qualquer expressão entre parênteses tem a maior precedência sobre os operadores fora dos parênteses. Portanto, ao usar parênteses, você pode forçar a avaliação de uma parte da expressão antes de aplicar outros operadores.

Quadro 4 – Ordem de precedência e associatividade entre as operações aritméticas (da mais alta para a mais baixa).

Operador	Operação	Associatividade
( )	Parênteses	Não se aplica. Dentro para fora.
**	Exponenciação ou Potenciação	Direita para a esquerda.
*	Multiplicação	Esquerda para direita.
/	Divisão	Esquerda para direita.
//	Divisão Inteira	Esquerda para direita.
%	Módulo ou Resto da Divisão	Esquerda para direita.
+	Adição	Esquerda para direita.
-	Subtração	Esquerda para direita.

Fonte: Elaborado pelos autores.

Caso seja necessário, você pode alterar a precedência dos operadores utilizando parênteses. Operações dentro de parênteses têm a mais alta precedência e são avaliadas primeiro. Veja a diferença produzida no exemplo abaixo:

```
resultado = 5 + 3 * 2      # O resultado é 11
resultado = (5 + 3) * 2    # O resultado é 16
```

## 6.3 OPERAÇÕES DE ATRIBUIÇÃO

O operador de atribuição em Python (=) é utilizado para atribuir um valor a uma variável. Além disso, ele pode ser combinado com operadores aritméticos para realizar operações e, ao mesmo tempo, atualizar o valor da variável.

### Atenção: Precedência do operador de atribuição.

O operador de atribuição tem a menor ordem de precedência em relação a todos os outros operadores disponíveis na linguagem Python. Isso significa que a atribuição é sempre a última operação a ser executada em uma linha de código.

### 6.3.1 Operador de atribuição básico

O operador de atribuição básico atribui o valor do lado direito à variável do lado esquerdo. Veja o exemplo a seguir:

```
x = 5  # Atribui o valor 5 à variável x
```

Neste exemplo, o valor 5 é atribuído à variável x.

### 6.3.2 Operadores de atribuição combinada

Os operadores de atribuição combinada realizam uma operação aritmética e atribuem o resultado à mesma variável em uma única etapa. A seguir, estão os principais operadores de atribuição combinada:

#### 6.3.2.1 Operador de Adição e Atribuição (+=)

```
x = 5  
x += 3  # Equivalente a x = x + 3. O valor de x se torna 8.
```

Neste exemplo, o operador += adiciona 3 ao valor atual de x e armazena o resultado na variável x.

#### 6.3.2.2 Operador de Subtração e Atribuição (-=)

```
x = 5  
x -= 3  # Equivalente a x = x - 3. O valor de x se torna 2.
```

Neste exemplo, o operador -= subtrai 3 do valor atual de x e armazena o resultado na variável x.

#### 6.3.2.3 Operador de Multiplicação e Atribuição (\*=)

```
x = 5  
x *= 3  # Equivalente a x = x * 3. O valor de x se torna 15.
```

Neste exemplo, o operador \*= multiplica o valor atual de x por 3 e armazena o resultado na variável x.

#### 6.3.2.4 Operador de Divisão e Atribuição (/=)

```
x = 5  
x /= 2  # Equivalente a x = x / 2. O valor de x se torna 2.5.
```

Neste exemplo, o operador /= divide o valor atual de x por 2 e armazena o resultado na variável x.

#### 6.3.2.5 Operador de Divisão Inteira e Atribuição (//=)

```
x = 5  
x //= 2  # Equivalente a x = x // 2. O valor de x se torna 2.
```

Neste exemplo, o operador //= realiza a divisão inteira do valor atual de x por 2 e armazena o resultado na variável x.

### 6.3.2.6 Operador de Módulo e Atribuição (%=)

```
x = 5
x %= 2 # Equivalente a x = x % 2. O valor de x se torna 1.
```

Neste exemplo, o operador %= calcula o resto da divisão do valor atual de x por 2 e armazena o resultado na variável x.

### 6.3.2.7 Operador de Exponenciação e Atribuição (\*\*=)

```
x = 5
x **= 2 # Equivalente a x = x ** 2. O valor de x se torna 25.
```

Neste exemplo, o operador \*\*= eleva o valor atual de x à potência de 2 e armazena o resultado na variável x.

## 6.4 EXEMPLOS COMENTADOS

### Exemplo: Calculadora da área do retângulo.

Implementar um programa que calcule a área do retângulo, dados os valores base = 4 e altura = 5. Fórmula matemática:

$$\text{área} = \text{base} \times \text{altura}$$

#### Solução:

```
1 base = 4
2 altura = 5
3 area = base * altura
4 print("Área do retângulo:", area)
```

O código calcula a área de um retângulo utilizando os valores fornecidos para a base e a altura. Primeiro, as variáveis base e altura são definidas com os valores 4 e 5, respectivamente. Em seguida, a área do retângulo é calculada multiplicando-se a base pela altura e o resultado é armazenado na variável area. Finalmente, o resultado é exibido na tela usando a função print(). Assim, o programa mostra a mensagem "Área do retângulo: 20".

### Exemplo: Conversor de distância de metros para quilômetros.

Implementar um programa que converta uma distância em metros para quilômetros. Considere o valor de 1516 metros. Fórmula matemática:

$$1 \text{ km} = 1000 \text{ m}$$

#### Solução:

```
1 metros = 1516
2 quilometros = metros / 1000
3 print("O valor em quilômetros é", quilometros)
```

O código realiza a conversão de distância de metros para quilômetros. Primeiro, a variável metros é definida com o valor 1516, que é a distância a ser convertida. Em seguida, a conversão é realizada dividindo o valor em metros por 1000, pois 1 quilômetro equivale a 1000 metros. O resultado da divisão é armazenado na variável quilometros. Finalmente, o programa exibe o resultado na tela com a mensagem "O valor em quilômetros é

1.516". Lembre-se que ponto é o separador decimal em Python.

### Exemplo: Calculadora de IMC.

Implementar um programa que calcule o IMC (Índice de Massa Corpórea) de uma pessoa, dados os valores peso = 80 kg e altura = 1,75 m. Fórmula matemática:

$$\text{IMC} = \frac{\text{peso}}{\text{altura} \times \text{altura}}$$

### Solução:

```
1 peso = 80
2 altura = 1.75
3 imc = peso / (altura * altura)
4 print("IMC:", imc)
```

O programa calcula o Índice de Massa Corpórea (IMC) usando a fórmula  $\text{IMC} = \frac{\text{peso}}{\text{altura} \times \text{altura}}$ . Primeiro, definimos as variáveis `peso` com o valor 80 e `altura` com o valor 1.75. Em seguida, calculamos `imc` dividindo o peso pelo quadrado da altura e, por fim, imprimimos o resultado.

### Exemplo: Teste de mesa para determinar o valor de num1 e num2.

Quais são os valores das variáveis ao final da execução do código? Encontre os valores por meio de um teste de mesa.

```
num1 = (8 + 2) * (5 - 3) / 2
num2 = num1 % 7
```

**Solução:** Para executar um teste de mesa<sup>1</sup>, vamos analisar expressão matemática:

$$\text{num1} \leftarrow (8 + 2) \times (5 - 3) \div 2$$

Começamos calculando `num1`. Primeiro, resolvemos as expressões dentro dos parênteses:

$$\text{num1} \leftarrow (10) \times (2) \div 2$$

Substituímos os resultados na expressão principal:

$$\text{num1} \leftarrow 10 \times 2 \div 2$$

Realizamos a multiplicação e a divisão:

$$\text{num1} \leftarrow 20 \div 2$$

$$\text{num1} \leftarrow 10$$

Portanto, `num1` é 10. A partir daqui, podemos calcular `num2`. Usamos o valor de `num1` calculado anteriormente:

$$\text{num2} \leftarrow \text{resto de } \text{num1} \div 7$$

$$\text{num2} \leftarrow \text{resto de } 10 \div 7$$

Realizamos a operação de resto da divisão:

$$\text{um2} \leftarrow 3$$

Portanto, `num2` é 3.

**Exemplo:** Teste de mesa para determinar o valor de `resultado`.

Qual é o valor da variável `resultado` ao final da execução do código? Encontre os valores por meio de um teste de mesa.

```
a = 4
b = 2
c = 5
resultado = (a + b) * c - (a / b) + (c % b) ** 2
```

**Solução:** Para executar um teste de mesa<sup>1</sup>, vamos analisar expressão matemática:

$$resultado \leftarrow (a + b) \times c - (a \div b) + (\text{resto de } c \div b)^2$$

Substituímos os valores das variáveis  $a = 4$ ,  $b = 2$  e  $c = 5$  na expressão.

$$resultado \leftarrow (4 + 2) \times 5 - (4 \div 2) + (\text{resto de } 5 \div 2)^2$$

Primeiro, resolvemos as expressões dentro dos parênteses:

$$resultado \leftarrow (6) \times 5 - (2) + (1)^2$$

$$resultado \leftarrow 6 \times 5 - 2 + 1^2$$

Realizamos as operações seguintes em ordem de precedência:

$$resultado \leftarrow 6 \times 5 - 2 + 1$$

$$resultado \leftarrow 30 - 2 + 1$$

$$resultado \leftarrow 29$$

Portanto, o valor de `resultado` é 29.

**Exemplo:** Conversor de tempo em segundos.

Implementar um programa que, dado um valor de tempo em segundos, faça a conversão para um valor de tempo horas, minutos e segundos. Encontre a solução para 4000 segundos. Lembre-se de que há 60 segundos em um minuto e 60 minutos em uma hora.

**Solução:**

```
1 total_em_segundos = 4000
2 horas = total_em_segundos // 3600
3 resto_da_hora = total_em_segundos % 3600
4 minutos = resto_da_hora // 60
5 segundos = resto_da_hora % 60
6 print(horas, "hora(s)", minutos, "minuto(s)", segundos, "segundo(s).")
```

O programa converte um valor total de segundos em horas, minutos e segundos. Primeiro, calcula o número de horas dividindo o total de segundos por 3600 (número de segundos em uma hora). Em seguida, usa o resto dessa divisão para calcular os minutos, dividindo o restante por 60 (número de segundos em um minuto). Finalmente, o resto dessa divisão representa os segundos restantes. O resultado é então impresso no formato desejado.

<sup>1</sup> Teste de mesa é uma técnica de depurar um código-fonte manualmente. O [Capítulo 7](#) contém uma explicação detalhada.



## 6.5 RESUMO DO CAPÍTULO

### Pontos-chave

**Associatividade de operadores** A associatividade determina a ordem em que operadores de mesmo nível são avaliados. Em Python, com exceção da exponenciação, operadores como adição e subtração são avaliados da esquerda para a direita.

**Atribuição** O operador de atribuição (`=`) é utilizado para definir o valor de uma variável. Ele pode ser combinado com operadores aritméticos para atualizar o valor da variável após uma operação.

**Atribuição combinada** Os operadores de atribuição combinada, como `+=` e `*=`, realizam uma operação aritmética e atribuem o resultado à variável em uma única etapa.

**Operadores aritméticos** Em Python, os operadores aritméticos básicos incluem adição (`+`), subtração (`-`), multiplicação (`*`), divisão (`/`), divisão inteira (`//`), módulo (`%`) e exponenciação (`**`).

**Operador de divisão inteira** O operador de divisão inteira (`//`) retorna a parte inteira da divisão, descartando a parte decimal.

**Operador de módulo ou resto da divisão** O operador de módulo (`%`) retorna o resto da divisão entre dois números.

**Precedência de operadores** A precedência dos operadores determina a ordem em que as operações são realizadas. Operações entre parênteses têm a mais alta precedência, seguidas pela exponenciação, multiplicação, divisão e assim por diante.

## 6.6 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Implementar um programa que calcule a área do triângulo. Fórmula matemática:

$$area = \frac{base \times altura}{2}$$

**Questão 2** Implementar um programa que faça a conversão da temperatura de Celsius para Fahrenheit. Fórmula matemática:

$$^{\circ}F = ^{\circ}C \times 1,8 + 32.$$

**Questão 3** Implementar um programa que, dados três números, calcule a média aritmética entre eles. Para calcular o valor da média aritmética, deve-se somar todos os números e dividir essa soma pela quantidade de elementos.

$$media = \frac{numero_1 + numero_2 + numero_3}{3}$$

**Questão 4** Qual é o valor da variável `resultado` ao final da execução do código? Encontre o valor por meio de um teste de mesa.

```
a = 1
b = 2
c = 6
resultado = ((a * b) + (c / (b + 1))) - ((a ** 2) % c)
```

**Questão 5** Implementar um programa que, dado um valor de entrada em segundos, faça a conversão para um tempo em dias, horas, minutos e segundos. Lembre-se de que há 60 segundos em um minuto, 60 minutos em uma hora e 24 horas em um dia.

### Exercícios complementares

**Questão 1** O que acontece ao utilizar operadores aritméticos em valores *strings*?

**Questão 2** Qual o valor da variável após execução do código a seguir?

```
num = 2 + 3 * 5
```

**Questão 3** Qual o valor da variável após execução do código a seguir?

```
num = 2 ** (1 + 5 % 2)
num *= 5
```

# Interação com o usuário

A interação com o usuário é essencial em muitas aplicações, pois permite que os programas sejam dinâmicos e ajustem-se às entradas em tempo real. Em Python, utilizamos a função `input()` para capturar essas entradas. Neste capítulo, exploraremos como essa função funciona e como utilizá-la de forma eficaz. Também discutiremos como formatar valores *string*.

## 7.1 A FUNÇÃO INPUT()

A função `input()` em Python faz com que o programa pause e aguarde que o usuário digite um texto. Esse texto é retornado como uma *string*, que pode ser armazenada em uma variável para uso posterior. Mais detalhes estão disponíveis na [documentação da função input](#).

O [Programa 18](#) ilustra um exemplo básico de interação com o usuário. O programa exibe a mensagem “Nome:” e aguarda que o usuário digite uma entrada e pressione <Enter>. O texto digitado é armazenado na variável `nome`, e a função `print()` exibe essa informação na tela.

---

**Programa 18** Imprimir o nome informado pelo usuário.

```
1 nome = input("Informe seu nome:")
2 print("Nome:", nome)
```

---

### 7.1.1 Convertendo tipos de dados

A função `input()` sempre retorna a entrada do usuário como uma *string*, independentemente do que foi digitado. Muitas vezes, é necessário converter essa *string* para outro tipo de dado, como *int* ou *float*. O [Programa 19](#) mostra um exemplo de como realizar essas conversões.

---

**Programa 19** Imprimir idade e altura informados pelo usuário.

```
1 idade = int(input("Informe sua idade:"))
2 altura = float(input("Informe sua altura:"))
3 print("Idade:", idade, "Altura:", altura)
```

---

No exemplo, a função `input()` captura a entrada do usuário. Antes de atribuir o valor à variável `idade`, usamos a função `int()` para converter a *string* em um número inteiro. Para valores decimais, usamos a função `float()` para converter a *string* em um número decimal.

Todos os tipos *built-in* em Python possuem funções de conversão. O [Quadro 5](#) lista essas funções e fornece *links* para suas respectivas documentações.

Quadro 5 – Funções de conversão de tipos *built-in* em Python.

Tipo	Função de conversão	Documentação
bool	bool()	<a href="#">Documentação da função bool()</a>
complex	complex()	<a href="#">Documentação da função complex()</a>
float	float()	<a href="#">Documentação da função float()</a>
int	int()	<a href="#">Documentação da função int()</a>
str	str()	<a href="#">Documentação da função str()</a>

Fonte: Elaborado pelos autores.

### Atenção: Erros de conversão de tipos.

O usuário pode inserir dados incorretos ou o desenvolvedor pode tentar converter tipos incompatíveis. Nestes casos, o interpretador Python exibirá uma mensagem de erro. É essencial validar e tratar essas entradas para evitar falhas no programa.

## 7.2 F-STRINGS E FORMATAÇÃO DE VALORES

A formatação de números (*int* e *float*) pode ser feita de maneira elegante utilizando *f-strings* (“*formatted string literals*”). Introduzidas no Python 3.6, *f-strings* proporcionam uma maneira clara e concisa de incorporar expressões dentro de *strings*.

### 7.2.1 Formatação de valores *int*

Para formatar um *int* sem especificações adicionais, utilize as chaves dentro da *f-string*:

```
valor = 42
mensagem = f"O valor é {valor}"           # O valor é 42
```

Para adicionar zeros à esquerda até alcançar um comprimento específico, use `:0Nd`, onde *N* é o comprimento desejado:

```
valor = 42
mensagem = f"O valor é {valor:05d}"       # O valor é 00042
```

Para incluir separadores de milhares, use `: ,:`

```
valor = 1000000
mensagem = f"O valor é {valor: ,}"        # O valor é 1,000,000
```

### 7.2.2 Formatação de valores *float*

Para formatar um *float* sem especificar o número de casas decimais:

```
valor = 3.14159
mensagem = f"O valor é {valor}"           # O valor é 3.14159
```

Para limitar o número de casas decimais, utilize `:.Nf`, onde *N* é o número de casas decimais desejadas:

```
valor = 3.14159
mensagem = f"O valor é {valor:.2f}"       # O valor é 3.14
```

Você pode combinar o separador de milhares com a formatação de casas decimais:

```
valor = 1234567.8910
mensagem = f"O valor é {valor:,.2f}"      # O valor é 1,234,567.89
```

## 7.3 RESUMO DO CAPÍTULO

### Pontos-chave

**Conversão de tipo** A função `input()` retorna sempre uma *string*. Para converter essa *string* em outros tipos de dados, como *int* ou *float*, utilizam-se funções de conversão específicas, como `int()` e `float()`. É essencial validar e tratar possíveis erros de conversão para garantir que o programa funcione corretamente.

**f-strings** Introduzidas no Python 3.6, as *f-strings* permitem a formatação elegante de *strings*. Elas facilitam a inclusão de expressões dentro de *strings* e suportam especificações como largura, preenchimento, separadores de milhares e controle do número de casas decimais.

**Função `input`** A função `input()` pausa o programa e aguarda a entrada do usuário, retornando essa entrada como uma *string*. É uma ferramenta essencial para interatividade com o usuário, permitindo capturar e usar dados fornecidos em tempo real.

Aqui está a seção revisada, mantendo o formato e ajustando as descrições para que fiquem claras e concisas:

“latex

## 7.4 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Implementar um programa que, dado um valor de lado informado pelo usuário, calcule a área do quadrado e formate a saída para que o valor seja apresentado com duas casas decimais. Utilize a fórmula matemática:

$$area = lado \times lado$$

**Questão 2** Implementar um programa que, dados os valores  $a$ ,  $b$  e  $c$  como coeficientes da Equação do Segundo Grau, calcule os valores de  $x_1$  e  $x_2$ . Utilize a fórmula de Bháskara:

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

**Questão 3** Implementar um programa que, dados dois valores de entrada informados pelo usuário, troque os valores entre as variáveis iniciais.

### Exercícios complementares

**Questão 1** Implemente um programa que, dada uma quantidade de litros de abastecimento e o preço em reais do combustível informados pelo usuário, calcule e apresente o valor total do abastecimento. Formate a saída para que seja apresentada

com duas casas decimais.

**Questão 2** Um programa faz a conversão de uma *string* capturada pela função `input()` para o tipo inteiro. O que acontece se o usuário digitar uma entrada que não possa ser convertida?

**Questão 3** É possível converter valores numéricos para o tipo *string* usando a função `str()`? Explique como isso pode ser feito.

# Operadores relacionais e operadores lógicos

A maioria dos programas que usamos no dia a dia inclui algoritmos de decisão. Por exemplo, uma janela pode pedir ao usuário que confirme ou cancele uma ação. Compreender como funcionam esses mecanismos na lógica de programação é essencial. Neste capítulo, exploraremos os operadores relacionais e lógicos, que são fundamentais para construir tais algoritmos.

## 8.1 OPERADORES RELACIONAIS

Os operadores relacionais são usados para comparar dois valores e sempre retornam um resultado booleano (**True** ou **False**). Eles são essenciais na lógica de programação, pois permitem que os programas tomem decisões com base em condições. Python possui os operadores relacionados apresentados no [Quadro 6](#).

Quadro 6 – Título do quadro.

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

Fonte: Elaborado pelos autores.

### 8.1.1 Operador igual a

Verifica se dois valores são iguais.

```
x = 10
y = 20
resultado = x == y      # False
```

### 8.1.2 Operador diferente de

Verifica se dois valores são diferentes.

```
x = 10
y = 20
resultado = x != y      # True
```

### 8.1.3 Operador maior que

Verifica se o valor à esquerda é maior que o valor à direita.

```
x = 10
y = 5
resultado = x > y       # True
```

### 8.1.4 Operador menor que

Verifica se o valor à esquerda é menor que o valor à direita.

```
x = 10
y = 15
resultado = x < y       # True
```

### 8.1.5 Operador maior ou igual a

Verifica se o valor à esquerda é maior ou igual ao valor à direita.

```
x = 10
y = 10
resultado = x >= y      # True
```

### 8.1.6 Operador menor ou igual a

Verifica se o valor à esquerda é menor ou igual ao valor à direita.

```
x = 10
y = 20
resultado = x <= y      # True
```

## 8.2 PRECEDÊNCIA E ASSOCIATIVIDADE ENTRE OPERAÇÕES RELACIONAIS

Os operadores relacionais têm o mesmo nível de precedência entre si. A associatividade dos operadores relacionais é da esquerda para a direita.

## 8.3 OPERADORES LÓGICOS

Os operadores lógicos são usados para combinar condições e fazer avaliações mais complexas. Eles são essenciais em estruturas de controle, como condicionais e laços de repetição, e permitem a criação de expressões lógicas compostas. Em Python, os operandos avaliados por operadores lógicos devem ser do tipo *bool*, que pode ser uma variável ou o resultado de uma operação relacional.



A avaliação de uma operação lógica pode ser realizada por meio de tabelas-verdade. Uma tabela-verdade mostra todas as combinações possíveis de valores para os operandos envolvidos em uma proposição lógica. Para cada combinação, a tabela exibe o resultado da expressão lógica composta por essas variáveis.

**Definição:** Tabela-verdade.

Uma tabela-verdade é uma ferramenta usada na lógica para ilustrar todos os possíveis valores de verdade (ou falsidade) das proposições lógicas e suas combinações. Ela é especialmente útil para analisar e entender como operadores lógicos funcionam em lógica proposicional.

### 8.3.1 Operador de conjunção (and)

O operador de conjunção é representado pela palavra reservada **and**. Pode ser interpretado como "e". Este operador retorna **True** somente se ambas as condições forem verdadeiras. A tabela-verdade para o operador **and** é apresentada no [Quadro 7](#).

Veja um exemplo do operador **and** em que os operandos são expressões relacionais:

```
x = 10
y = 20
resultado = x > 5 and y < 30      #True
```

Quadro 7 – Tabela-verdade do operador de conjunção (and).

Operando 1	Operando 2	Resultado
True	True	True
True	False	False
False	True	False
False	False	False

Fonte: Elaborado pelos autores.

### 8.3.2 Operador de disjunção (or)

O operador de disjunção é representado pela palavra reservada **or**. Pode ser interpretado como "ou". Este operador retorna **True** se pelo menos uma das condições for verdadeira. A tabela-verdade para o operador **or** é apresentada no [Quadro 8](#).

Veja um exemplo do operador **or** em que os operandos são expressões relacionais:

```
x = 10
y = 30
resultado = x > 5 or y < 30      #True
```

### 8.3.3 Operador de negação (not)

O operador de negação é representado pela palavra reservada **not**. Pode ser interpretado como "não". Este operador inverte o valor da condição, conforme descrito na tabela-verdade apresentada no [Quadro 9](#). Note que a operação é unária, ou seja, aplicável somente a um operando.

Veja um exemplo do operador **not** em que o operando é uma expressão relacional:

```
x = 10
resultado = not x > 5            #False
```

Quadro 8 – Tabela-verdade do operador de disjunção (**or**).

Operando 1	Operando 2	Resultado
True	True	True
True	False	True
False	True	True
False	False	False

Fonte: Elaborado pelos autores.

Quadro 9 – Tabela-verdade do operador de negação (**not**).

Operando	Resultado
True	False
False	True

Fonte: Elaborado pelos autores.

### 8.3.4 Precedência e associatividade entre operações lógicas

A ordem de precedência e a associatividade entre as operações lógicas são apresentadas no [Quadro 10](#). Assim como acontece com as operações aritméticas, as operações lógicas são avaliadas de acordo com uma ordem definida.

Quadro 10 – Ordem de precedência e associatividade entre operações lógicas (do mais alto para o mais baixo)

Operador	Operação	Associatividade
( )	Parênteses	Não se aplica. Dentro para fora.
not	Negação	Não se aplica. Dentro para fora.
and	Conjunção	Esquerda para direita.
or	Disjunção	Esquerda para direita.

Fonte: Elaborado pelo autor.

## 8.4 ORDEM DE PRECEDÊNCIA ENTRE TIPOS DE OPERAÇÃO

A ordem de precedência entre os tipos de operadores da linguagem de programação é importante para determinar a ordem em que as operações são realizadas em uma expressão. Uma linha de código pode conter uma combinação de operações aritméticas, de atribuição, relacionais e/ou lógicas.

Agora que já apresentamos todos os operadores da linguagem Python, é possível observar a ordem de precedência de todas as operações. O [Quadro 11](#) apresenta essa ordem. Note que dentro do mesmo tipo também há uma ordem de precedência.

Quadro 11 – Ordem de precedência entre todos os operadores da linguagem Python.

Tipos de operação	Operações em ordem de precedência
Parênteses	( )
Operadores aritméticos	** * / // % + -
Operadores relacionais	< <= > >= != ==
Operadores lógicos	not and or
Operadores de atribuição	=

Fonte: Elaborado pelo autor.

## 8.5 EXEMPLOS COMENTADOS

### Exemplo: Teste de mesa para determinar o valor de `resultado`.

Qual é o valor da variável `resultado` ao final da execução do código? Encontre os valores por meio de um teste de mesa.

```
a = 3
b = 2
c = 4
resultado = (a + b) * c < a / b and b ** c >= b - a
```

#### Solução:

Para executar um teste de mesa, vamos analisar a expressão matemática:

$$resultado \leftarrow (a + b) \times c < a \div b \text{ and } b^c \geq b - a$$

Substituímos os valores das variáveis  $a = 3$ ,  $b = 2$  e  $c = 4$  na expressão.

$$resultado \leftarrow (3 + 2) \times 4 < 3 \div 2 \text{ and } 2^4 \geq 2 - 3$$

Primeiro, resolvemos as expressões dentro dos parênteses:

$$resultado \leftarrow (5) \times 4 < 3 \div 2 \text{ and } 2^4 \geq 2 - 3$$

$$resultado \leftarrow 5 \times 4 < 3 \div 2 \text{ and } 2^4 \geq 2 - 3$$

Realizamos as operações seguintes em ordem de precedência:

$$resultado \leftarrow 5 \times 4 < 3 \div 2 \text{ and } 16 \geq 2 - 3$$

$$resultado \leftarrow 20 < 1,5 \text{ and } 16 \geq 2 - 3$$

$$resultado \leftarrow 20 < 1,5 \text{ and } 16 \geq -1$$

$$resultado \leftarrow \text{False and True}$$

$$resultado \leftarrow \text{False}$$

Portanto, o valor de `resultado` é **False**.

## 8.6 RESUMO DO CAPÍTULO

### Pontos-chave

**Operadores lógicos** Operadores utilizados para combinar condições e realizar avaliações mais complexas. Incluem **and**, **or**, e **not**, com tabelas-verdade que ilustram seus comportamentos.

**Operadores relacionais** Permitem comparar valores e retornam resultados booleanos (**True** ou **False**). Incluem **=**, **!=**, **>**, **<**, **>=**, e **<=**.

**Tipo booleanos** Representa valores de verdadeiro (**True**) e falso (**False**), utilizados em operações lógicas e condicionais para controlar o fluxo do programa. Em Python, são identificados pela palavra-reservada **bool**.

## 8.7 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Qual é o valor da variável `resultado` ao final da execução do código? Encontre o valor por meio de um teste de mesa.

```
x = 7
y = 3
z = 2
resultado = (x % y != 1) or (z ** 2 < x and y != z)
```

**Questão 2** Qual é o valor da variável `resultado` ao final da execução do código? Encontre o valor por meio de um teste de mesa.

```
p = 8
q = 4
r = 2
resultado = (p - q * r <= q) and (not (r ** 2 > p))
```

**Questão 3** Qual é o valor da variável `resultado` ao final da execução do código? Encontre o valor por meio de um teste de mesa.

```
a = True
b = False
c = True
d = False
resultado = (a and b) or (c and d)
```

### Exercícios complementares

**Questão 1** Operadores relacionais e operadores lógicos são importantes para construção de algoritmos? Explique com suas palavras e dê um exemplo.

**Questão 2** Qual é o comportamento de operadores relacionais com valores do tipo *string*? Explique com suas palavras.

**Questão 3** Quais são as diferenças entre as tabelas-verdade dos operadores de conjunção (**and**) e disjunção (**or**)? Explique com suas palavras.

# Fluxo de execução condicional

Em muitos algoritmos, é necessário tomar decisões para executar tarefas diferentes com base em condições específicas. Por exemplo, um programa de e-mail pode permitir ou bloquear o acesso dependendo das informações de *login* inseridas pelo usuário. Para implementar essas decisões, utilizamos estruturas de controle chamadas instruções condicionais. Essas estruturas permitem que o programa execute diferentes partes do código dependendo das condições avaliadas. Em Python, uma das principais estruturas para isso é a instrução `if`.

O fluxo condicional permite que o programa decida quais blocos de código devem ser executados com base na avaliação de condições. Sem essas estruturas, o código seria executado de forma linear e sem variações, sempre na mesma sequência.

## 9.1 ESTRUTURAS CONDICIONAIS

Estruturas condicionais, também conhecidas como estruturas de seleção, são fundamentais para que um programa escolha entre diferentes blocos de código com base nas condições avaliadas. Essas condições são expressões compostas por literais booleanos (*bool*), operadores relacionais e/ou operadores lógicos.

Em Python, as principais estruturas condicionais são:

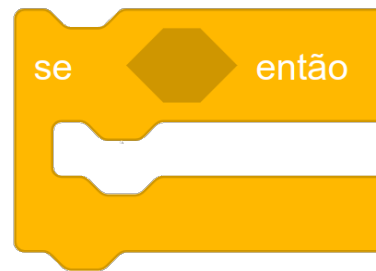
- `if`
- `if-else`
- `if-elif-else`

### 9.1.1 Estrutura `if`

A estrutura `if` executa um bloco de código somente se uma condição for verdadeira. Em outras palavras, é como dizer “se a condição for verdadeira, faça isso”. A sintaxe básica é a seguinte:

```
1 if condicao:  
2     # Bloco condicional em que condicao é verdadeira  
3     # Restante do código a ser executado
```

O bloco de código na linha 2 será executado somente se a `condicao` for verdadeira. Caso contrário, esse bloco será ignorado. A linha 3, por outro lado, será executada independentemente da avaliação da condição.

Figura 14 – Estrutura `if` em Scratch.

Fonte: Elaborada pelo autor.

A Figura 14 ilustra o bloco “se...então” em Scratch, que é equivalente ao comando `if` em Python. Tanto em Scratch quanto em Python, é possível definir blocos de código que só serão executados se a condição avaliada for verdadeira. Em Scratch, isso é feito com a estrutura do bloco “se então”, que delimita o código a ser executado dentro dele. Em Python, a execução condicional é controlada pela indentação: o código dentro do comando `if` deve ser indentado para a direita, em relação ao próprio comando `if`.

A indentação é crucial porque define quais linhas de código fazem parte do bloco condicional. Isso garante que somente o código desejado seja executado quando a condição for verdadeira.

**Definição:** Indentação.

A indentação é usada para definir blocos de código. Diferentemente de outras linguagens que usam chaves (`{}`) ou palavras-chave para delimitar blocos, Python utiliza a indentação para essa finalidade. Cada bloco de código é indentado com um número consistente de espaços ou tabulações. A indentação correta é crucial porque Python não possui um delimitador explícito para blocos de código. Se a indentação estiver incorreta, o Python levantará um erro de sintaxe. Em comparação com Scratch, tudo que faz parte do bloco de código deve ser encaixado na parte interna do bloco “se...então”.

**Atenção:** Como indentar um bloco de código corretamente?

Por padrão, são usados quatro espaços para a indentação, mas você pode escolher usar tabulações (tecla `<Tab>`) se preferir, desde que seja consistente em todo o código. Isso significa que você não deve usar espaços em uma linha e tabulação em outra linha no mesmo código-fonte. VS Code converte automaticamente as tabulações em quatro espaços, mas esse recurso não é padrão entre editores de código-fonte.

### 9.1.2 Estrutura `if-else`

Quando você precisa lidar com duas possibilidades, uma verdadeira e outra falsa, utiliza-se a estrutura `if-else`. Isso pode ser entendido como uma estrutura de decisão onde se diz “se a condição for verdadeira, faça isso; senão, faça aquilo”. A sintaxe é:

```
1 if condicao:  
2     # Bloco de código executado se a condição for verdadeira  
3 else:  
4     # Bloco de código executado se a condição for falsa  
5     # Restante do código a ser executado
```

O bloco de código na linha 2 será executado se a `condicao` for verdadeira. Se a condição for falsa, então o bloco de código na linha 4 será executado. A linha 5, que está fora dos blocos condicionais, será executada independentemente da condição avaliada.

Assim como no comando `if`, a indentação é usada para definir quais linhas pertencem ao bloco condicional. As linhas 2 e 4 devem ser corretamente indentadas para dentro dos comandos `if` e `else`, respectivamente. A estrutura visual dessa lógica é bloco “se...então..senão” no Scratch que é ilustrado na Figura 15.

Figura 15 – Estrutura `if-else` em Scratch.



Fonte: Elaborada pelo autor.

#### Atenção: Comando `else` isolado.

O comando `else` deve sempre vir após um comando `if` ou `elif`. Não deve ser usado de forma isolada. Lembre-se de que, se necessário, você pode modificar a condição avaliada no `if` ou no `elif`.

### 9.1.3 Estrutura `if-elif-else`

Quando há várias condições a verificar, usamos os comandos `if-elif-else`. Esses comandos são úteis quando você tem várias possibilidades a considerar e deseja tomar decisões diferentes com base na condição que for verdadeira. A estrutura pode ser compreendida como: “se a condição 1 for verdadeira, faça isso; se a condição 2 for verdadeira, faça aquilo; se nenhuma das condições for verdadeira, faça algo diferente”. A sintaxe é:

```

1 if condicao_1:
2     # Bloco de código executado se a condição 1 for verdadeira
3 elif condicao_2:
4     # Bloco de código executado se a condição 1 for falsa e a condição 2 for
      verdadeira
5 else:
6     # Bloco de código executado se nenhuma das condições for verdadeira
7     # Restante do código a ser executado

```

A linha 2 será executada se a `condicao_1` for verdadeira. Se a `condicao_1` não for verdadeira, mas a `condicao_2` for, então a linha 4 será executada. Se nenhuma das condições for verdadeira, a linha 5 será executada.

O comando `elif` é uma forma elegante usada em Python para evitar um grande número de indentação ao usar `else` e `if` aninhados.

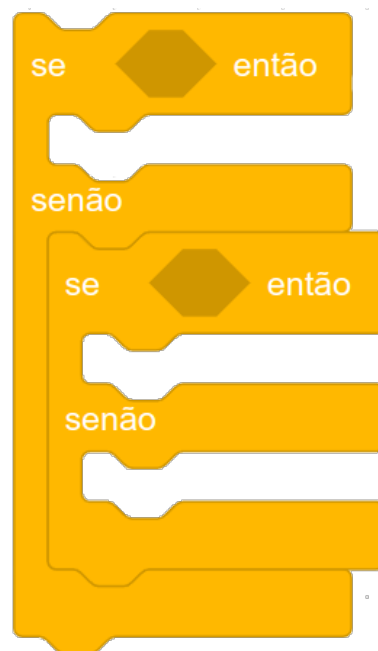
Esses comandos permitem que o programa tome decisões mais complexas e execute diferentes blocos de código com base nas condições definidas. A estrutura condicional com `elif` é uma forma elegante de evitar o aninhamento excessivo de `if`, tornando o código mais limpo e legível.

A lógica em Scratch, equivalente ao comando `if-elif-else` em Python, é mostrada na Figura 16. O comando `elif` é particularmente útil para evitar a necessidade de aninhar

comandos `if` dentro de outros comandos `if`. A estrutura condicional aninhada em Python, equivalente à lógica em Scratch, seria:

```
1 if condicao_1:
2     # Bloco de código executado se a condição 1 for verdadeira
3 else:
4     if condicao_2:
5         # Bloco de código executado se a condição 1 for falsa e a condição 2 for
           verdadeira
6 else:
7     # Bloco de código executado se a condição 1 e a condição 2 forem falsas
8 # Restante do código a ser executado
```

Figura 16 – Estrutura `if-elif-else` em Scratch.



Fonte: Elaborada pelo autor.

#### Atenção: Estrutura `elif` isolada.

É importante entender que é necessário ter um comando `if` para usar o comando `elif`. O comando `elif` pode ou não estar associado a um `else` final.

## 9.2 ESTRUTURAS CONDICIONAIS ANINHADAS

Estruturas condicionais aninhadas são aquelas em que uma condição é testada dentro de outra condição. Esse tipo de estrutura é útil quando é necessário avaliar múltiplas condições em níveis diferentes, permitindo uma tomada de decisão mais refinada. Em Python, isso é feito colocando um comando `if` dentro de outro comando `if` (ou `else`, `elif`), criando uma hierarquia de decisões.

Considere o seguinte trecho de código para verificação de usuário em um programa:

```
1 usuario = input("Digite seu nome de usuário: ")
2 senha = input("Digite sua senha: ")
3
4 if usuario == "turing":
```



```
5     if senha == "12345":
6         print("Bem-vindo, turing!")
7     else:
8         print("Senha incorreta.")
9 else:
10    print("Nome de usuário não encontrado.")
```

Neste exemplo, são realizadas duas verificações sequenciais para autenticar um usuário:

1. **Verificação do nome de usuário:** Primeiro, o código verifica se o nome de usuário inserido é “turing”. Se for verdadeiro, o código prossegue para a próxima verificação.
2. **Verificação da senha:** Se o nome de usuário estiver correto, então verifica se a senha é “12345”. Se a senha estiver correta, o usuário é saudado. Caso contrário, é exibida uma mensagem de senha incorreta.
3. **Tratamento de erros:** Se o nome de usuário não for “turing”, o código exibe uma mensagem informando que o nome de usuário não foi encontrado.

Este código lida com dois cenários de falha:

- O nome de usuário não ser encontrado.
- A senha ser incorreta.

Ao usar estruturas condicionais aninhadas, o código permite uma resposta apropriada para cada situação específica de erro, tornando a autenticação mais robusta e informativa.

## 9.3 EXEMPLOS COMENTADOS

### Exemplo: Maior de idade?

Implementar um programa que, dada uma idade informada pelo usuário, apresenta a mensagem “Você é maior de idade.” caso a informação seja verdadeira.

#### Solução:

```
1 idade = int(input("Informe sua idade: "))
2 if idade >= 18 :
3     print("Você é maior de idade.")
```

O programa solicita que o usuário insira sua idade (linha 1). O comando `if` (linha 2) verifica se a idade é maior ou igual a 18. Se essa condição for verdadeira, a mensagem da linha 3 (“Você é maior de idade.”) é exibida. Caso contrário, nada acontece.

### Exemplo: Maior ou menor de idade?

Implementar um programa que, dada uma idade informada pelo usuário, apresenta a mensagem “Você é maior de idade.” caso a informação seja verdadeira ou apresenta a mensagem “Você é menor de idade.” caso a informação seja falsa.

#### Solução:

```
1 idade = int(input("Informe sua idade: "))
2 if idade >= 18 :
3     print("Você é maior de idade.")
4 else:
```

```
5 print("Você é menor de idade.")
```

O programa solicita a idade do usuário (linha 1) e usa uma estrutura condicional `if-else` (linha 2) para determinar se a pessoa é maior ou menor de idade. Se a idade for 18 ou mais, o programa exibe a mensagem da linha 3 ("Você é maior de idade."). Caso contrário, ele exibe a mensagem da linha 4 ("Você é menor de idade.").

### Exemplo: Criança, adolescente ou adulto?

Implementar um programa que, dada uma idade informada pelo usuário, apresenta a mensagem "Você é criança." caso a idade seja menor do que 12 anos, a mensagem "Você é adolescente." caso a idade seja até 18 anos, e "Você é adulto." caso não se enquadre nas duas primeiras categorias.

#### Solução:

```
1 idade = int(input("Informe sua idade: "))
2 if idade < 12 :
3     print("Você é criança.")
4 elif idade <= 18:
5     print("Você é adolescente.")
6 else:
7     print("Você é adulto.")
```

O programa solicita a idade do usuário e usa uma estrutura condicional `if-elif-else` para determinar a faixa etária. Se a idade for menor que 12 anos, exibe a mensagem da linha 3 ("Você é criança.") Se a idade for entre 12 e 18 anos, exibe a mensagem da linha 5 ("Você é adolescente."). Caso contrário, exibe a mensagem da linha 7 ("Você é adulto.").

### Exemplo: Login em serviço de e-mail

Implementar um programa que, dado um endereço de e-mail e uma senha informados pelo usuário, simule o funcionamento de um serviço de e-mail. O programa deve verificar inicialmente se o e-mail informado existe. Após a confirmação, deve solicitar a senha. Se a senha estiver correta, o programa deve apresentar uma mensagem de saudação. Caso contrário, o programa deve informar o usuário sobre o erro nos dados de entrada.

#### Solução:

```
1 usuario = input("Informe seu e-mail: ")
2 if usuario == "turing@email.com":
3     senha = input("Informe sua senha: ")
4     if senha == "12345":
5         print("Acesso liberado!")
6     else:
7         print("Senha incorreta!")
8 else:
9     print("Usuário não encontrado.")
```

No código acima, utilizamos estruturas condicionais para implementar a lógica de verificação de um login:

- A primeira estrutura condicional `if usuario == "turing@email.com"` verifica se o e-mail informado pelo usuário é o esperado.
- Se o e-mail for correto, o programa solicita a senha e utiliza uma segunda estrutura condicional `if senha == "12345"` para verificar se a senha fornecida está

correta.

- A segunda estrutura condicional está aninhada dentro da primeira. Isso significa que a verificação da senha só é realizada se a verificação do e-mail for bem-sucedida.
- Se a senha estiver correta, o programa imprime "Acesso liberado!". Caso contrário, imprime "Senha incorreta!".
- Se o e-mail fornecido não corresponder ao esperado, o programa imprime "Usuário não encontrado!".

A estrutura condicional aninhada permite que o programa execute uma verificação adicional somente se a primeira verificação for bem-sucedida, garantindo assim uma validação em etapas.

## 9.4 RESUMO DO CAPÍTULO

### Pontos-chave

**Estrutura `if`** Utilizada para executar um bloco de código somente se uma condição for verdadeira. A sintaxe básica envolve o uso de uma expressão condicional seguida por um bloco de código indentado.

**Estrutura `else`** Empregada em conjunto com a estrutura `if` para fornecer um bloco de código alternativo que será executado se a condição do `if` não for verdadeira. Não pode ser utilizado de forma isolada e deve sempre seguir um comando `if` ou `elif`.

**Estrutura `elif`** Permite a verificação de múltiplas condições. Usado após uma estrutura `if` para avaliar condições adicionais. Se a condição do `if` não for verdadeira, o `elif` permite checar outras condições, e o bloco de código associado ao primeiro `elif` verdadeiro será executado. Pode ser seguido por um `else` opcional.

**Estrutura condicional aninhada** Refere-se ao uso de um comando `if` dentro de outro comando `if` (ou `else`, `elif`), permitindo a avaliação de condições adicionais em níveis diferentes e possibilitando a tomada de decisões mais complexas.

## 9.5 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Implemente um programa que, dado um número inteiro informado pelo usuário, determine se o número é par ou ímpar.

**Questão 2** Implemente um programa que, dada a idade informada pelo usuário, determine se ele pode ou não votar nas eleições. No sistema eleitoral brasileiro, pessoas com 16 anos já têm o direito de votar.

**Questão 3** Analise o trecho de código a seguir:

```
if a > b:
    if b > c:
        print("Bloco 1")
    else:
```

```

    print("Bloco 2")
else:
    if c > a:
        print("Bloco 3")
    else:
        print("Bloco 4")

```

Qual bloco será impresso os valores a seguir?

- a = 10, b = 5 e c = 8.
- a = 3, b = 7 e c = 9.
- a = 6, b = 6 e c = 4.

**Questão 4** Implemente um programa que, dados dois números informados pelo usuário, apresente esses números em ordem crescente.

**Questão 5** Implemente um programa que, dado um ano informado pelo usuário, determine se o ano é bissexto. Um ano é bissexto se:

- for múltiplo de 400 ou
- for múltiplo de 4 e não múltiplo de 100.

### Exercícios complementares

**Questão 1** Implemente um programa que, dado um número inteiro informado pelo usuário, determine se o número é múltiplo de sete.

**Questão 2** Implemente um programa que, dada a altura em metros e o peso em quilogramas informados pelo usuário e determine a classificação de IMC (Índice de Massa Corpórea) do usuário. Fórmula matemática:

$$\text{IMC} = \frac{\text{peso}}{\text{altura} \times \text{altura}}$$

Quadro de classificação do IMC:

IMC	Classificação
Abaixo de 18,5	Abaixo do peso
Entre 18,5 e 25	Normal
Acima de 25 e até 30	Acima do peso
Acima de 30	Obesidade

**Questão 3** Implemente um programa que, dadas três palavras informadas pelo usuário, apresente essas palavras em ordem alfabética.

**Questão 4** Implemente um programa que simule o sistema de caixa de uma loja. O programa deve calcular o valor a ser pago pelo cliente com base no valor informado pelo operador de caixa. Além disso, o operador de caixa informará se o pagamento será à vista ou a prazo e a quantidade de parcelas na última opção. No caso de pagamento à vista, deve-se aplicar um desconto de 10%. Para pagamentos a prazo, que são sem juros, o programa deve apresentar a quantidade de parcelas e o valor de cada parcela.

**Desafio 1** Implemente um programa que simule um sistema bancário básico. O programa deve permitir que o cliente acesse o sistema informando o número da conta e a senha. Após a autenticação bem-sucedida, o cliente deve ter acesso às seguintes opções:

1. Mostrar saldo atual;
2. Fazer um saque;
3. Fazer um depósito;
4. Sair.

O sistema deve iniciar com um saldo de R\$ 500,00. A senha para acesso é 54321 e o número da conta é 858585 - 8.

Ao escolher as opções de saque ou depósito, o programa deve solicitar o valor da operação e informar o saldo atualizado após a transação.

Além disso, o programa deve apresentar mensagens de erro claras ao usuário, tais como:

- Mensagem de erro caso o número da conta ou a senha estejam incorretos;
- Mensagem de erro em caso de saldo insuficiente para a realização de um saque;
- Mensagem de erro para qualquer opção inválida no menu.

**Desafio 2** Implemente um programa que calcule o salário líquido de um trabalhador a partir do salário bruto informado pelo usuário. Considere que o trabalhador é celetista, ou seja, está contratado com base na Consolidação das Leis do Trabalho (CLT). O cálculo deve considerar: a contribuição previdenciária para o INSS (Instituto Nacional do Seguro Social) e o IRPF (Imposto sobre a Renda das Pessoas Físicas). As tabelas atualizadas no momento da escrita deste livro são as seguintes:

Salário bruto	Alíquota do INSS
Até R\$ 1.412,00	7,5%
De R\$ 1.412,00 até R\$ 2.666,68	9%
De R\$ 2.666,69 até R\$ 4.000,03	12%
De R\$ 4.000,04 até R\$ 7.786,02	14%

Salário tributável	Alíquota do IRPF	Parcela a Deduzir
Até R\$ 2.259,20	Isento	R\$ 0,00
De R\$ 2.259,21 até R\$ 2.826,65	7,5%	R\$ 169,44
De R\$ 2.826,66 até R\$ 3.751,05	15,0%	R\$ 381,44
De R\$ 3.751,06 até R\$ 4.664,68	22,5%	R\$ 662,77
Acima de R\$ 4.664,68	27,5%	R\$ 896,00

Algumas dicas para o cálculo:

1. A contribuição previdenciária é definida por faixas. Por exemplo, se o salário bruto for de R\$ 4.000,00, a contribuição total é calculada como aproximadamente R\$ 378,82 (7,5% de R\$ 1.412,00 + 9% de R\$ 1.254,68 + 12% de R\$ 1.333,31).
2. A contribuição previdenciária tem um limite. Esse teto é de aproximadamente R\$ 908,86 (7,5% de R\$ 1.412,00 + 9% de R\$ 1.254,68 + 12% de R\$ 1.333,34 + 14% de R\$ 378,82). Veja que salários superiores R\$ 7.786,02 pagam o valor do teto.

3. O salário tributável é o salário restante após descontar a contribuição previdenciária.
4. O cálculo do IRPF é feito multiplicando o salário tributável pela alíquota correspondente e subtraindo a parcela a deduzir. Por exemplo, com um salário bruto de R\$ 4.000,00, o salário tributável após a contribuição do INSS é de R\$ 3.621,18. O imposto devido será de R\$ 161,73, resultando em um salário líquido de R\$ 3.459,45.
5. O IRPF não tem teto.

# Fluxo de execução com repetição

Muitas situações do cotidiano exigem que realizemos tarefas repetitivas. Tomemos como exemplo a tarefa de lavar a louça do almoço: a ação de lavar um prato é repetida até que não reste nenhum prato sujo.

No contexto da programação, o conceito de repetição é uma parte fundamental de muitos programas. Por exemplo, o sistema de um caixa de supermercado suporta a repetição da leitura de códigos de barras até que todos os itens de uma compra sejam processados.

As linguagens de programação modernas possuem estruturas de repetição que permitem a implementação de algoritmos que necessitam desse tipo de abordagem. Em Python, essas estruturas são representadas pelos comandos **for** e **while**. No Scratch, as estruturas equivalentes são os blocos “repita...vezes” e “repita até que...”.

## 10.1 ESTRUTURAS DE REPETIÇÃO, LAÇOS DE REPETIÇÃO OU *LOOPS*?

Os três termos são usados de forma intercambiável na programação para se referir ao mesmo conceito: a execução repetida de um bloco de código até que uma condição seja satisfeita. *Loops* é o termo em inglês, enquanto “estruturas de repetição” e “laços de repetição” são os termos mais comuns em português. A escolha do termo pode depender do contexto ou da preferência do programador, mas todos se referem ao mesmo mecanismo fundamental.

É importante entender que o uso das estruturas *for* ou *while* cria o fluxo de execução repetida.

## 10.2 ESTRUTURA FOR

A estrutura **for** em Python é utilizada para iterar sobre uma sequência de valores ou uma estrutura de dados, como uma lista, tupla ou *string*. Ela é geralmente usada quando o número de iterações é conhecido. A estrutura equivalente ao **for** em Scratch é o bloco “repita...vezes”, apresentado na [Figura 17](#).

### Definição: Iteração.

Iteração é o processo de repetir uma série de instruções até que uma condição específica seja alcançada. Em programação, refere-se ao ato de percorrer elementos de uma estrutura de dados ou repetir um bloco de código um número definido de vezes.

Figura 17 – Estrutura for em Scratch.



Fonte: Elaborada pelos autores.

A sintaxe do **for** compreende o comando **in**, uma variável que assume o valor de cada elemento de uma sequência e a própria sequência. Veja a seguir a estrutura típica:

```
for i in sequencia:
    # Bloco de repetição
    # A variável i assume o valor de um elemento da sequência a cada iteração
```

A estrutura de repetição faz com que o fluxo de execução fique em *loop* até que toda a sequência seja esgotada. A cada iteração, a variável *i* assume um valor da sequência. Assim como as estruturas condicionais, as estruturas de repetição delimitam um bloco de código.

**Definição:** Variável de controle do laço **for**.

A variável de controle em um **for** é a variável que assume cada valor da sequência durante as iterações. Ela é usada para acessar e manipular os elementos da sequência dentro do bloco de repetição. É comum, na maioria das linguagens de programação, que essa variável seja nomeada como *i*.

Uma forma comum de gerar uma sequência em Python é por meio da função **range()**. Por exemplo, para gerar uma sequência de 0 a 9, você pode usar o seguinte código:

```
for i in range(10):
    # Bloco de repetição
    # O valor da variável i é atualizado a cada iteração
```

Na primeira iteração, *i* é igual a 0; na segunda, *i* é igual a 1; e assim por diante, até que, na décima iteração, *i* seja igual a 9.

### 10.3 FUNÇÃO RANGE

A função **range()** gera uma sequência de números com base nos argumentos fornecidos. A função pode ser usada de forma flexível para gerar sequências conforme a necessidade do problema. É possível usar a função de três formas diferentes. A documentação da função está disponível em [documentação da função range](#).

Quando a função **range()** é utilizada com um único parâmetro, que é o parâmetro de parada (*stop*), ela gera uma sequência que começa em 0 e vai até *stop* - 1. Por exemplo:

```
range(5)      # 0, 1, 2, 3, 4
```

Neste caso, a sequência gerada será: 0, 1, 2, 3 e 4. Da mesma forma, ao utilizar a instrução **range(7)**, a sequência gerada será: 0, 1, 2, 3, 4, 5 e 6.

Quando a função **range()** é utilizada com dois parâmetros, que são o parâmetro de início (*start*) e o parâmetro de parada (*stop*), ela gera uma sequência que começa em *start* e vai até *stop* - 1. Por exemplo:



```
range(15, 19)    # 15, 16, 17, 18
```

Neste caso, a sequência gerada será: 15, 16, 17 e 18. Da mesma forma, ao utilizar a instrução `range(1007, 1010)`, a sequência gerada será: 1007, 1008 e 1009.

Quando a função `range()` é utilizada com três parâmetros, que são o parâmetro de início (*start*), o parâmetro de parada (*stop*) e o parâmetro de passo (*step*), ela gera uma sequência que começa em *start* e vai até *stop*, onde cada elemento está a um intervalo de *step* um do outro. É importante saber que o parâmetro de parada não é incluído na sequência gerada. Por exemplo:

```
range(0, 50, 5)    # 0, 5, 10, 20, 25, 30, 35, 40, 45
```

Neste caso, a sequência gerada será: 0, 5, 10, 15, 20, 25, 30, 35, 40 e 45. Da mesma forma, ao utilizar a instrução `range(1000, 1020, 3)`, a sequência gerada será: 1000, 1003, 1006, 1009, 1012, 1015 e 1018.

O parâmetro de passo também pode assumir valores negativos. A instrução `range(5, -1, -1)` retorna a sequência: 5, 4, 3, 2, 1.

## 10.4 ESTRUTURA WHILE

A estrutura `while` em Python permite que um bloco de código seja repetido enquanto uma condição for verdadeira. Essa estrutura é útil quando não se sabe, de antemão, quantas vezes o *loop* precisará ser executado.

Em construção...

## 10.5 EXEMPLOS COMENTADOS

### Exemplo: Sequência dos $n$ primeiros números naturais a partir de 0.

Implemente um programa que, dado um número natural  $n$  informado pelo usuário, imprima a sequência dos  $n$  primeiros números naturais a partir de 0. Por exemplo, o programa deve imprimir “0”, “1” e “2” para  $n = 3$ .

#### Solução:

```
1 n = int(input("Informe a quantidade de números da sequência: "))
2 for i in range(n):
3     print(i)
```

O programa começa pedindo ao usuário que informe a quantidade de números naturais que deseja na sequência, armazenando este valor na variável  $n$ . Em seguida, a estrutura de repetição `for` é utilizada para iterar a partir de 0 até  $n - 1$ . A sequência é gerada pela função `range(n)`, que cria uma sequência de números de 0 até  $n - 1$ . A cada iteração, a variável de controle  $i$  assume um valor da sequência e o imprime utilizando a função `print()`. É importante notar que o laço `for` executa  $n$  iterações, correspondendo ao número de elementos desejado na sequência.

### Exemplo: Contagem regressiva de 10 até 0.

Implemente um programa que imprima uma contagem regressiva de 10 até 0. Ao final, o programa deve imprimir a mensagem “Valendo!”.

#### Solução:

```
1 for i in range(10, -1, -1):
2     print(i)
3 print("Valendo!")
```

Neste exemplo, a função `range(10, -1, -1)` gera uma sequência que começa em 10 e termina em 0, decrementando de 1 em 1. A função `range()` é utilizada com três parâmetros: o início da sequência (10), o fim (não incluído -1) e o passo (-1). O laço `for` itera sobre essa sequência, imprimindo cada número. Após a contagem regressiva, o programa exibe a mensagem “Valendo!” para sinalizar o fim da contagem.

### Exemplo: Soma dos $n$ primeiros números naturais a partir de 0.

Implemente um programa que, dado um número natural  $n$  informado pelo usuário, some a sequência dos  $n$  primeiros números naturais a partir de 0. Por exemplo, o programa deve imprimir “6” ( $0 + 1 + 2 + 3$ ) para  $n = 4$ .

#### Solução:

```
1 n = int(input("Informe a quantidade de números da sequência: "))
2 soma = 0 # O valor 0 é elemento neutro da soma
3 for i in range(n):
4     soma += i # Equivalente a soma = soma + i
5 print("A soma dos", n, "primeiros números naturais é", soma)
```

O programa pede ao usuário para informar um número natural  $n$ , que define quantos números naturais, começando de 0, serão somados. A variável `soma` é inicializada com 0, pois 0 é o elemento neutro da soma.

A estrutura de repetição `for` é utilizada para iterar sobre a sequência de números de 0 até  $n - 1$ . Em cada iteração do laço:

- O valor atual da variável `i` (o número atual na sequência) é adicionado à variável `soma`.
- A cada passo, `soma` é atualizada com a soma parcial dos números até aquele ponto.
- Por exemplo, se  $n$  for 4, o laço executa 4 vezes, e `soma` é atualizada da seguinte forma:
  - Iteração 1:  $soma = 0$  (inicial) +  $0 = 0$
  - Iteração 2:  $soma = 0$  (atual) +  $1 = 1$
  - Iteração 3:  $soma = 1$  (atual) +  $2 = 3$
  - Iteração 4:  $soma = 3$  (atual) +  $3 = 6$

Ao final do laço, a variável `soma` contém o valor total da soma dos  $n$  primeiros números naturais. O resultado é impresso na tela.

### Exemplo: Produto dos $n$ primeiros números naturais a partir de 1.

Implemente um programa que, dado um número natural  $n$  informado pelo usuário, multiplique a sequência dos  $n$  primeiros números naturais a partir de 1. Por exemplo, o programa deve imprimir “24” ( $1 \times 2 \times 3 \times 4$ ) para  $n = 4$ .

**Solução:**

```

1 n = int(input("Informe a quantidade de números da sequência: "))
2 produto = 1          # 0 valor 1 é elemento neutro da multiplicação
3 for i in range(1, n + 1):
4     produto *= i      # Equivalente a produto = produto * i
5 print("O produto dos", n, "primeiros números naturais é", produto)

```

O programa solicita ao usuário um número natural  $n$  e inicializa a variável `produto` com 1, que é o elemento neutro da multiplicação. Em seguida, o laço `for` é utilizado para iterar sobre a sequência de números de 1 até  $n$ , inclusivamente. A cada iteração do laço:

- O valor atual da variável `i` é multiplicado pela variável `produto`.
- A cada passo, `produto` é atualizado com o produto parcial dos números até aquele ponto.
- Por exemplo, se  $n$  for 4, o laço executa 4 vezes, e `produto` é atualizado da seguinte forma:
  - Iteração 1: `produto` = 1 (inicial)  $\times$  1 = 1
  - Iteração 2: `produto` = 1 (atual)  $\times$  2 = 2
  - Iteração 3: `produto` = 2 (atual)  $\times$  3 = 6
  - Iteração 4: `produto` = 6 (atual)  $\times$  4 = 24

Ao final do laço, a variável `produto` contém o valor total do produto dos  $n$  primeiros números naturais. O resultado é impresso na tela.

**Exemplo:** Teste de mesa para determinar o valor das variáveis após o laço `for`.

Quais são os valores das variáveis ao final da execução do código? Encontre os valores por meio de um teste de mesa.

```

a = 3
resultado = 0

for i in range(5):
    if i >= a:
        resultado *= i
    else:
        resultado += i

```

**Solução:**

Para encontrar os valores das variáveis após a execução do código, realizamos um teste de mesa. Vamos analisar o código linha por linha e acompanhar as alterações nas variáveis `a`, `resultado` e `i` durante cada iteração do laço `for`.

**Início:**  $a \leftarrow 3$ ,  $resultado \leftarrow 0$

**Iteração 1:**  $i \leftarrow 0$

- Como  $i$  (0) é menor que  $a$  (3), executa `resultado += i`.
- $resultado \leftarrow 0 + 0 = 0$

**Iteração 2:**  $i \leftarrow 1$

- Como  $i$  (1) é menor que  $a$  (3), executa `resultado += i`.
- `resultado`  $\leftarrow 0 + 1 = 1$

**Iteração 3:**  $i \leftarrow 2$

- Como  $i$  (2) é menor que  $a$  (3), executa `resultado += i`.
- `resultado`  $\leftarrow 1 + 2 = 3$

**Iteração 4:**  $i \leftarrow 3$

- Como  $i$  (3) é igual a  $a$  (3), executa `resultado *= i`.
- `resultado`  $\leftarrow 3 \times 3 = 9$

**Iteração 5:**  $i \leftarrow 4$

- Como  $i$  (4) é maior que  $a$  (3), executa `resultado *= i`.
- `resultado`  $\leftarrow 9 \times 4 = 36$

**Após o laço:** O valor final de `resultado` é 36. A variável  $a$  mantém seu valor original de 3, pois não é modificada dentro do laço. A variável  $i$  assume o valor final de 4 após a última iteração do laço `for`.

## 10.6 RESUMO DO CAPÍTULO

### Pontos-chave

**Estrutura `for`** Utilizada para iterar sobre uma sequência de valores ou uma estrutura de dados, como listas, tuplas ou strings.

**Função `range`** Gera uma sequência numérica com base nos parâmetros fornecidos.

**Iteração** O processo de repetir uma série de instruções, tipicamente percorrendo elementos de uma estrutura de dados ou repetindo um bloco de código um número definido de vezes.

**Loop, laço de repetição ou estrutura de repetição** Termos usados para descrever a execução repetida de um bloco de código até que uma condição seja satisfeita.

**Variável de controle** A variável que assume os valores da sequência durante as iterações de um `for`. Ela é usada para acessar e manipular os elementos da sequência dentro do bloco de repetição. Geralmente é nomeada como  $i$ .

## 10.7 ATIVIDADES PROPOSTAS

### Exercícios de fixação

**Questão 1** Implemente um programa que imprima todos os números ímpares entre 1 e 50.

**Questão 2** Implemente um programa que imprima a tabuada de um número informado pelo usuário. O programa deve exibir os resultados das multiplicações desse número por valores de 0 até 10.

**Questão 3** Implemente um programa que calcule a soma de uma sequência de números informados pelo usuário. O programa deve solicitar, inicialmente, a quantidade de números a serem somados. Em seguida, o usuário deve fornecer os números propriamente ditos, um a um. O programa deverá, então, calcular e exibir a soma total desses números.

**Questão 4** Implemente um programa que encontre o menor valor de uma sequência de números informados pelo usuário. O programa deve solicitar, inicialmente, a quantidade de números a serem fornecidos. Em seguida, o usuário deve informar os números um a um. O programa deve encontrar e exibir o menor valor da sequência.

**Questão 5** Implemente um programa que, dado um número inteiro informado pelo usuário, calcule o fatorial desse número. O sinal de exclamação (“!”) é o símbolo matemático que representa o operador fatorial. O fatorial de um número ( $n!$ ) é o produto de todos os inteiros positivos menores ou iguais a ele. Por exemplo,  $4! = 4 \times 3 \times 2 \times 1$ . Por definição,  $0! = 1$ .

### Exercícios complementares

**Questão 1** Implemente um programa que calcule a potência de um número utilizando uma estrutura de repetição. O programa deve solicitar ao usuário que informe a base e o expoente, ambos considerados como inteiros.

**Questão 2** Implemente um programa que calcule e exiba a média de uma sequência de números informados pelo usuário. O programa deve solicitar, inicialmente, a quantidade de números a serem informados. Em seguida, o usuário deve fornecer os números um a um. O programa deverá calcular e exibir a média aritmética desses números.

**Questão 3** Implemente um programa que calcule o  $n$ -ésimo termo da sequência de Fibonacci, onde o valor de  $n$  é informado pelo usuário. A sequência de Fibonacci é uma sequência numérica em que cada número é a soma dos dois números anteriores considerando  $n \geq 3$ . Por definição, Fibonacci de 1 = 1 e Fibonacci de 2 = 1. Por exemplo, a sequência de Fibonacci para  $n = 5$  é 1, 1, 2, 3, 5.

# Interface de Linha de Comando

Interface de Linha de Comando ou CLI (do inglês *Command Line Interface*), é um tipo de interface de usuário que permite aos usuários interagirem com o sistema operacional ou programas de software via comandos digitados em um terminal ou console.

## A.1 UTILIZAÇÃO DA CLI EM SISTEMAS OPERACIONAIS MICROSOFT WINDOWS

*Prompt* de comando é a principal ferramenta para acessar a CLI do Microsoft Windows. É possível abrir o *prompt* de comando de duas formas:

1. pressione <Windows+R>, digite `cmd` e pressione <Enter>; ou
2. clique no botão *|Iniciar|*, digite “cmd” ou “prompt de Comando” e selecione o aplicativo nos resultados da busca.

Quando o Prompt de Comando é aberto, você verá uma janela com um cursor piscando, aguardando a entrada de comandos. A linha de comando típica começa com o caminho do diretório atual, seguido pelo cursor piscando.

### A.1.1 Comandos básicos

Alguns dos comandos básicos que podem ser usados no *prompt* de comando: `cd`, `dir` e `cls`.

#### A.1.1.1 Comando cd

O comando muda o diretório atual. Alguns exemplos:

- `cd C:\Users\Turing` Muda para o diretório especificado.
- `cd ..` Volta ao diretório pai.
- `cd .` Permanece no diretório atual.

### A.1.1.2 Comando `dir`

O comando muda lista o conteúdo de um diretório. Alguns exemplos:

- `dir` Lista o conteúdo do diretório atual.
- `dir C:\Users\Turing` Lista o conteúdo do diretório especificado.

### A.1.1.3 Comando `cls`

Limpa a tela do *prompt* de comando. Alguns exemplos:

- `cls` Limpa o texto da tela do *prompt* de comando.

## A.1.2 Uso da tecla *Tab*

A tecla <Tab> é usada para autocompletar nomes de arquivos e diretórios. Quando você começa a digitar o nome de um arquivo ou diretório e pressiona <Tab>, o *prompt* de comando tenta completar automaticamente o nome com base no que foi digitado. Se houver várias correspondências possíveis, pressionar <Tab> repetidamente vai iterando por todas as opções disponíveis. Isso facilita a navegação e a entrada de comandos, evitando erro de digitação e economizando tempo. Por exemplo, se você digitar `cd Doc` e pressionar <Tab>, o *prompt* de comando pode completar o caminho para `cd Documentos`, se esse for o diretório correspondente. <Tab> ainda facilita a digitação de nomes de arquivos ou diretórios que contêm espaços.

## A.2 UTILIZAÇÃO DA CLI EM SISTEMAS OPERACIONAIS GNU/LINUX

Terminal é a ferramenta para acessar a CLI em sistemas operacionais GNU/Linux. A forma de abrir o terminal depende da distribuição e do ambiente de trabalho instalado. Nas distribuições Ubuntu em instalação padrão, é possível abrir o *terminal* de comando de duas formas:

1. pressione <Ctrl+Alt+T> ou
2. clique no botão `|Mostrar aplicativos|` ou similar, digite “terminal” e selecione o aplicativo nos resultados da busca.

Quando o terminal é aberto, você verá uma janela com um cursor piscando, aguardando a entrada de comandos. A linha de comando típica começa com o caminho do diretório atual, que é o diretório de arquivos do usuário atualmente logado, seguido pelo cursor piscando.

### A.2.1 Comandos básicos

Alguns dos comandos básicos que podem ser usados no terminal: `cd`, `ls` e `clear`.

#### A.2.1.1 Comando `cd`

O comando tem o mesmo funcionamento do que em um ambiente Microsoft Windows e, desta forma, muda o diretório atual ao ser executado. Alguns exemplos:

- `cd /home/Turing` Muda para o diretório especificado.
- `cd ..` Volta ao diretório pai.
- `cd .` Permanece no diretório atual.

### A.2.1.2 Comando `ls`

O comando é equivalente ao comando `dir` utilizado em um ambiente Microsoft Windows. O comando muda lista o conteúdo de um diretório. Alguns exemplos:

- `ls` Lista o conteúdo do diretório atual.
- `ls /home/Turing` Lista o conteúdo do diretório especificado.

### A.2.1.3 Comando `clear`

Limpa a tela do terminal. Alguns exemplos:

- `clear` Limpa o texto da tela do terminal.

## A.2.2 Uso da tecla *Tab*

A tecla <Tab>, assim como acontece em um ambiente Microsoft Windows, é usada para autocompletar nomes de arquivos e diretórios. Quando você começa a digitar o nome de um arquivo ou diretório e pressiona <Tab>, o comando tenta completar automaticamente o nome com base no que foi digitado. Se houver várias correspondências possíveis, as opções são listadas no terminal. <Tab> ainda facilita a digitação de nomes de arquivos ou diretórios que contêm espaços. No caso do terminal, é necessário utilizar aspas caso opte por não utilizar a tecla.

#### **Atenção:** Barra ou barra invertida.

Observe que os sistemas operacionais Microsoft Windows utilizam a barra invertida (\) na estrutura de arquivos e diretórios. Por outro lado, sistemas operacionais GNU/Linux utilizam a barra tradicional (/).



# Depuração

Depuração é o processo de identificar, analisar e corrigir erros ou bugs. O objetivo da depuração é garantir que o programa funcione corretamente e conforme o esperado, permitindo que o programador encontre e resolva problemas que possam causar falhas, comportamentos inesperados ou resultados incorretos.

Neste capítulo, apresentamos algumas técnicas de depuração.

## B.1 TESTE DE MESA

O teste de mesa é uma técnica usada na programação e no desenvolvimento de algoritmos para analisar e verificar o comportamento de um algoritmo ou programa. É uma forma manual e teórica de depuração onde o desenvolvedor simula a execução de um algoritmo em papel, usando um conjunto específico de dados de entrada, para verificar se ele está funcionando corretamente.

### B.1.1 Como funciona o teste de mesa?

1. Escolha do algoritmo e dados de entrada.
  - Defina o algoritmo ou programa que deseja testar e escolha um conjunto de dados de entrada para a simulação.
2. Criação de uma tabela.
  - Monte uma tabela ou uma série de tabelas que representem as variáveis do algoritmo e suas mudanças ao longo da execução. Cada linha da tabela geralmente representa um passo do algoritmo.
3. Simulação manual.
  - Execute o algoritmo manualmente com os dados de entrada e registre as alterações nas variáveis em cada passo. Preencha a tabela com os valores das variáveis em cada estágio da execução.
4. Verificação dos resultados.
  - Compare os resultados finais obtidos com o esperado. Verifique se o algoritmo produz o resultado correto para os dados de entrada fornecidos.

### B.1.2 Vantagens do teste de mesa

O teste de mesa oferece várias vantagens. Em primeiro lugar, é uma técnica eficaz para identificar erros lógicos no algoritmo, por permitir que o desenvolvedor acompanhe o comportamento do algoritmo passo a passo. Além disso, ajuda na compreensão do funcionamento interno e da lógica do algoritmo, facilitando o aprendizado e a análise detalhada. Outra vantagem importante é a sua simplicidade: o teste de mesa não requer ferramentas especiais ou software adicional, bastando papel e caneta para realizar a simulação e registrar as mudanças nas variáveis durante a execução do algoritmo.

**Alerta:** A importância do teste de mesa para aprender lógica de programação.

Programar é uma tarefa desafiadora para iniciantes. Teste de mesa é um recurso extremamente útil que auxilia no aprendizado e na compreensão de lógica de programação. Recomendamos que você pratique teste de mesa sempre que tiver um contato com um novo algoritmo.

## B.2 DEPURAÇÃO VIA SOFTWARE

A depuração via software é um processo essencial para identificar, analisar e corrigir erros ou comportamentos inesperados em um programa. Ela permite que o desenvolvedor monitore a execução do código e verifique o estado das variáveis, facilitando a correção de falhas e a melhoria da qualidade do software.

### B.2.1 Depuração de Python no VS Code

Para realizar a depuração de código Python no VS Code, é necessário ter a extensão Python instalada.

#### B.2.2 Passo 1: Inserção de *breakpoints*

A preparação para a depuração começa com a inserção de *breakpoints* no código-fonte. Isso é feito clicando na margem esquerda ao lado dos números de linha. As linhas com *breakpoints* possuem uma marcação específica. A [Figura 18](#) mostra um exemplo de código-fonte com *breakpoints* nas linhas 4 e 5. Para remover um *breakpoint*, basta clicar novamente na marcação.

Figura 18 – Exemplo de código-fonte com *breakpoints* nas linhas 4 e 5 no VS Code.

```
1  usuario = input("Digite seu nome de usuário: ")
2  senha = input("Digite sua senha: ")
3
4  if usuario == "turing":
5      if senha == "12345":
6          print("Bem-vindo, turing!")
7      else:
8          print("Senha incorreta.")
9  else:
10     print("Nome de usuário não encontrado.")
```

Fonte: Elaborada pelos autores.

Os *breakpoints* são pontos de parada na execução do programa durante a depuração. Quando o interpretador atinge um desses pontos, é possível analisar os valores das variáveis em tempo real.

**Definição: Breakpoint.**

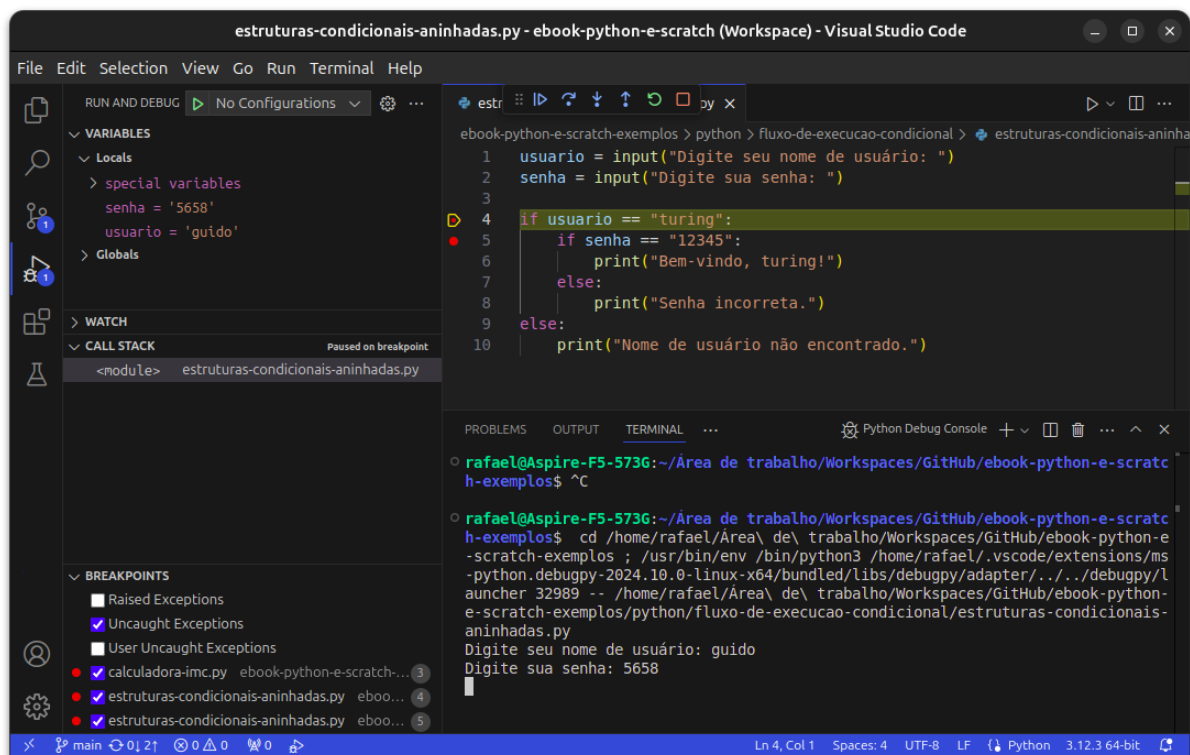
Um *breakpoint* é um ponto específico no código onde a execução do programa é interrompida para permitir que o desenvolvedor examine o estado do programa, incluindo variáveis e o fluxo de execução.

### B.2.3 Passo 2: Iniciar depuração

Para iniciar a execução em modo de depuração no VS Code, selecione a opção `|Start Debugging|` no menu `|Run|` ou use o atalho `<F5>`.

Um exemplo de programa em depuração é apresentado na [Figura 19](#). Observe que, na aba à esquerda, é possível visualizar o valor das variáveis. Se essa informação não estiver visível, clique na opção `|Run and Debug|`.

Figura 19 – Exemplo de código-fonte em depuração com parada em *breakpoint* no VS Code.



Fonte: Elaborada pelos autores.

### B.2.4 Passo 3: Execução da depuração

Os depuradores oferecem diversos recursos para controlar a execução de um programa em depuração. No VS Code, os principais controles disponíveis são: `|Continue|`, `|Step Over|`, `|Step Into|`, `|Step Out|`, `|Restart|` e `|Stop|`. Os ícones desses controles estão mostrados na [Figura 20](#).

- **|Continue|** (`<F5>`): Este botão retoma a execução do programa até que o próximo *breakpoint* seja atingido ou o programa termine. É útil para continuar a execução após uma pausa em um *breakpoint*.
- **|Step Over|** (`<F10>`): Avança a execução do programa para a próxima linha de código, mas sem entrar em funções chamadas nessa linha. Se a linha atual contém uma chamada de função, o depurador executará a função inteira e parará na próxima linha após a chamada.

Figura 20 – Ícones de controle de depuração no VS Code.



Fonte: Elaborada pelos autores.

- **|Step Into| (<F11>):** Avança a execução para a próxima linha de código e, se essa linha contém uma chamada de função, o depurador entrará na função e pausará na primeira linha da função chamada. Isso é útil para investigar o comportamento dentro de funções.
- **|Step Out| (<Shift+F11>):** Se você está dentro de uma função e deseja sair dela, esse comando executará o restante da função e retornará ao ponto onde a função foi chamada. Isso permite que você continue a depuração do ponto onde a função foi chamada sem entrar em detalhes adicionais da função.
- **|Restart| (<Ctrl+Shift+F5>):** Reinicia a sessão de depuração, reiniciando a execução do programa desde o início. É útil para testar alterações no código desde o início do processo de depuração.
- **|Stop| (<Shift+F5>):** Encerra a sessão de depuração atual e interrompe a execução do programa. Use este comando para parar completamente a depuração e retornar ao estado normal de edição.

### B.2.5 Executar com depuração ou sem depuração?

Quando um programa está em fase de produção, isto é, pronto para ser disponibilizado aos usuários finais ou já disponibilizado, ele não deve ser executado em modo de depuração. Executar um programa com o depurador ativado pode introduzir uma série de problemas, incluindo:

- **Desempenho Reduzido:** A execução em modo de depuração é geralmente mais lenta em comparação com a execução normal. Isso ocorre porque o depurador precisa monitorar e registrar informações adicionais, o que pode impactar significativamente o desempenho do programa.
- **Sobrecarregar o Ambiente de Produção:** O modo de depuração pode gerar uma quantidade significativa de logs e dados de diagnóstico, o que pode sobrecarregar o ambiente de produção e afetar a experiência do usuário.
- **Segurança e Privacidade:** Informações sensíveis e detalhes internos do programa podem ser expostos durante a depuração. Executar o programa em modo de depuração em um ambiente de produção pode aumentar o risco de exposição não intencional de dados críticos.

Portanto, o modo de depuração deve ser usado apenas durante as fases de desenvolvimento e teste. Durante essas fases, o depurador ajuda os desenvolvedores a identificar e corrigir erros, testar diferentes cenários e entender o comportamento do programa. Após a conclusão do desenvolvimento e a realização dos testes, o programa deve ser executado em modo normal, sem o depurador ativo, para garantir a melhor performance e segurança para os usuários finais.

# Referências

ALGORITMO. In: DICIONÁRIO Michaelis On-line. Editora Melhoramento, 2024. Disponível em: <<https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/algoritmo/>>. Citado na página 17.

Fundação Scratch. *Scratch*. 2024. Disponível em: <<https://scratch.mit.edu/>>. Citado na página 14.

History of the BBC. *Monty Python's Flying Circus*. 1969. Disponível em: <<https://www.bbc.com/historyofthebbc/anniversaries/october/monty-pythons-flying-circus>>. Citado na página 15.

Python.org. *Python*. 2024. Disponível em: <<https://www.python.org/>>. Citado na página 15.

TIOBE. *TIOBE Index for June 2024*. 2024. Disponível em: <<https://www.tiobe.com/tiobe-index/>>. Citado na página 15.