



**INSTITUTO
FEDERAL**
Paraná

LÓGICA DE PROGRAMAÇÃO COM PYTHON E SCRATCH

Rafael Silva Santos

Rafael Silva Santos

*Lógica de programação com Python e
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

CARO leitor, seja bem-vindo ao fascinante mundo da programação! Este livro foi cuidadosamente elaborado para conduzir você, estudante, por uma jornada empolgante através das linguagens Python e Scratch.

Nosso principal objetivo é tornar a programação acessível e divertida. Faremos sempre uma abstração entre Scratch e Python, 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|*, 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.

Resumos

As caixas cinza-escuras contêm resumos dos principais pontos.

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. Parte desses assuntos foi incluída nos apêndices.

O livro está organizado da seguinte forma...

VAMOS COMEÇAR!

Prepare-se para embarcar em uma jornada emocionante. Com Python e Scratch, 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	– Logo da linguagem de programação Python.	15
Figura 3	– Atores de Monty Python's Fly Circus.	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 em um GNU/Linux	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.	30
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 a e duas atribuições de valor.	38
Figura 14	– Estrutura <code>if</code> em Scratch.	57
Figura 15	– Estrutura <code>if-else</code> em Scratch.	58
Figura 16	– Estrutura <code>if-elif-else</code> em Scratch.	59

Lista de Quadros

1 – Instalação do ambiente de desenvolvimento Python em distribuições GNU/Linux com gerenciadores de pacotes apt ou yum	25
2 – Principais palavras reservadas na linguagem Python.	38
3 – Tipos de dados <i>built-in</i> em Python.	40
4 – Ordem de precedência e associatividade entre as operações aritméticas (do mais alta para o mais baixa).	42
5 – Tabela-verdade do operador de conjunção (and).	53
6 – Tabela-verdade do operador de disjunção (or).	53
7 – Tabela-verdade do operador not (“não”).	54
8 – Ordem de precedência e associatividade entre operações lógicas (do mais alta para o mais baixa)	54
9 – Ordem de precedência entre todos os operadores da linguagem Python.	54

Lista de Tabelas

Tabela 1 – Índice TIOBE em junho de 2024	16
----------------------------------------------------	----

Lista de abreviaturas e siglas

CMD	<i>Prompt de comando</i>
CLI	<i>Command Line Interface</i>
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
pip	Package installer for Python
MIT	Massachusetts Institute of Technology
TI	Tecnologia da Informação
VS Code	Visual Studio Code

Sumário

1	Introdução	12
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	15
1.7	Atividades propostas	16
2	Algoritmo	17
2.1	O que é um algoritmo?	17
2.2	Importância de algoritmos na programação	18
2.3	Como construir um algoritmo?	19
2.4	Código-fonte	19
2.5	Resumo do capítulo	20
2.6	Atividades propostas	20
3	Ambiente de Desenvolvimento	22
3.1	Instalação do Ambiente de Desenvolvimento Python	22
3.1.1	Microsoft Windows	23
3.1.2	GNU/Linux	23
3.2	Scratch	25
3.3	Visual Studio Code	25
3.4	Resumo do capítulo	26
3.5	Atividades propostas	27
4	Primeiro programa	29
4.1	Como executar um programa Python?	29
4.1.1	Execução de programas pelo Visual Studio Code	29
4.1.2	Execução pela CLI	31
4.2	A função <code>print()</code>	31
4.3	Comentários	31
4.3.1	Comentários de uma linha	31
4.3.2	Comentários de múltiplas linhas	32
4.4	Sintaxe	32
4.5	Execução sequencial	32
4.6	Exemplos comentados	34
4.7	Atividades propostas	35
5	Variáveis, tipos de dados e operadores aritméticos	37
5.1	Variáveis	37
5.1.1	Declaração e atribuição de valores a variáveis	39
5.2	Tipos de Dados	39
5.3	Operações aritméticas	40
5.3.1	Operador de adição	40
5.3.2	Operador de subtração	40
5.3.3	Operador de multiplicação	40
5.3.4	Operador de divisão	41
5.3.5	Operador de divisão inteira	41
5.3.6	Operador de módulo ou resto da divisão	41
5.3.7	Operador de exponenciação	41
5.4	Precedência e associatividade entre operações aritméticas	41
5.5	Operações de atribuição	42

5.5.1	Operador de atribuição básico	42
5.5.2	Operador de atribuição combinado	43
5.5.2.1	Operador de adição e atribuição	43
5.5.2.2	Operador de subtração e atribuição	43
5.5.2.3	Operador de multiplicação e atribuição	43
5.5.2.4	Operador de divisão e atribuição	43
5.5.2.5	Operador de divisão inteira e atribuição	43
5.5.2.6	Operador de módulo e atribuição	43
5.5.2.7	Operador de exponenciação e atribuição	43
5.6	Exemplos comentados	44
5.6.1	Calculadora da área do retângulo	44
5.6.2	Conversor de distância	44
5.6.3	Calculadora de Índice de Massa Corpórea (IMC)	44
5.6.4	Teste de mesa de código-fonte	44
5.6.5	Conversor de tempo	46
5.7	Atividades propostas	46
6	Interatividade com o usuário	48
6.1	A função <code>input()</code>	48
6.1.1	Convertendo tipos de dados	48
6.2	A função <code>type()</code>	49
6.3	<i>f-strings</i> e formatação de valores	49
6.3.1	Formatação de valores <i>int</i>	49
6.3.2	Formatação de valores <i>float</i>	50
6.4	Atividades propostas	50
7	Operadores relacionais e operadores lógicos	51
7.1	Operadores relacionais	51
7.1.1	Operador “igual a” (<code>=</code>)	51
7.1.2	Operador “diferente de” (<code>!=</code>)	51
7.1.3	Operador “maior que” (<code>></code>)	52
7.1.4	Operador “menor que” (<code><</code>)	52
7.1.5	Operador “maior ou igual a” (<code>>=</code>)	52
7.1.6	Operador “menor ou igual a” (<code><=</code>)	52
7.2	Precedência e associatividade entre operações relacionais	52
7.3	Operadores lógicos	52
7.3.1	Operador de conjunção (<code>and</code>)	53
7.3.2	Operador de disjunção (<code>or</code>)	53
7.3.3	Operador de negação (<code>not</code>)	53
7.3.4	Precedência e associatividade entre operações lógicas	54
7.4	Ordem de precedência entre tipos de operação	54
7.5	Atividades propostas	55
8	Fluxo Condicional	56
8.1	Estruturas Condicionais	56
8.1.1	Comando <code>if</code>	56
8.1.2	Comandos <code>if-else</code>	57
8.1.3	Comandos <code>if-elif-else</code>	58
8.2	Exemplos Comentados	59
8.3	Atividades Propostas	60
9	Depuração	62
9.1	Teste de mesa	62
9.1.1	Como funciona o teste de mesa?	62
9.1.2	Vantagens do teste de mesa	63
A	Command Line Interface	64
A.1	Utilização da CLI em sistemas operacionais Microsoft Windows	64
A.1.1	Comandos básicos	64

	A.1.1.1	Comando <code>cd</code>	64
	A.1.1.2	Comando <code>dir</code>	65
	A.1.1.3	Comando <code>cls</code>	65
A.1.2		Uso da tecla <i>Tab</i>	65
A.2		Utilização da CLI em sistemas operacionais GNU/Linux	65
	A.2.1	Comandos básicos	65
	A.2.1.1	Comando <code>cd</code>	65
	A.2.1.2	Comando <code>ls</code>	66
	A.2.1.3	Comando <code>clear</code>	66
A.2.2		Uso da tecla <i>Tab</i>	66
Referências			67

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 MIT (Massachusetts Institute of Technology) e 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, é importante definirmos 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 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 computadores entendem os dados com 0s e 1s?”. Para responder a essa pergunta, precisamos entender 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, o qual 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 usado, 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. O *bit* é a unidade que representa os estados, assumindo o valor 0 ou o valor 1. Um *byte* é equivalente a oito *bits*.

Os dados em um computador são armazenados e processados em forma binária. Por exemplo, um *bit* (a menor unidade de dados em um computador) pode ser 0 ou 1. Vários bits podem ser agrupados para formar *bytes* (8 *bits*), palavras (16, 32 ou 64 bits, dependendo da arquitetura do computador) e assim por diante.

A linguagem de programação facilita a comunicação entre o desenvolvedor e o computador. O computador opera utilizando códigos de máquina (sistema binário). Os dados em um computador são armazenados e processados em forma binária. Por exemplo, um *bit* (a menor unidade de dados em um computador) pode ser 0 ou 1. Vários *bits* podem ser agrupados para formar

bytes (8 *bits*), palavras (16, 32 ou 64 *bits*, dependendo da arquitetura do computador) e assim por diante.

1.4 SCRATCH

Scratch é uma linguagem de programação visual desenvolvida pelo grupo Lifelong Kindergarten da universidade americana MIT (Massachusetts Institute of Technology) em 2007. A linguagem utiliza uma interface visual simples por meio do 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 desenvolvedor crie histórias, jogos e animações visuais. Além disso, possibilita o aprendizado de lógica de programação de forma visual.

Figura 1 – Logo da linguagem de programação Scratch.



Fonte: [Fundação Scratch](#) (2024).

1.5 PYTHON

Python é uma linguagem de programação desenvolvida pelo matemático Guido van Rossum em 1991. Os códigos-fonte¹ 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 de programação é multiplataforma, interpretada, de alto nível e de propósito geral. A capacidade de Python de ser usada em diversas áreas, desde desenvolvimento web até Ciência de Dados e Inteligência Artificial, faz dela uma das linguagens mais utilizadas no mundo. As principais características da linguagem são:

Multiplataforma Um programa em Python pode ser executado em diferentes ambientes, em especial, na 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 é uma linguagem que abstrai fortemente detalhes do controle do computador. Tal característica torna a tarefa de programar mais simplificada do que comparada às linguagens de médio e baixo nível.

Propósito geral A linguagem Python pode ser usada para diversas áreas, como, por exemplo, Ciência de Dados, Inteligência Artificial e desenvolvimento Web.

Outras características de Python incluem: programação funcional, uso de indentação² por blocos, orientação a objetos e vasta disponibilidade de bibliotecas e módulos.

¹ 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.

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

A Figura 2 apresenta o logotipo da linguagem Python. Ao observar atentamente, é possível identificar que ele é formado duas serpentes, uma azul e uma amarela. Apesar disso, é interessante saber que o nome da linguagem não é uma referência às serpentes do gênero píton. O nome da linguagem é, na verdade, uma homenagem ao programa humorístico britânico Monty Python's Fly Circus, o qual era um dos favoritos de Guido van Rossum. Uma foto dos atores do seriado é apresentada na Figura 3.

Figura 2 – Logo da linguagem de programação Python.



Fonte: [Python.org](https://python.org) (2024).

Figura 3 – Atores de Monty Python's Fly Circus.



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

Python é amplamente apontada com uma das linguagens de programação mais usadas no mundo. No momento da escrita deste livro, a linguagem estava em primeiro lugar de popularidade no índice TIOBE³. A Tabela 1 apresenta as 10 linguagens mais populares segundo esse índice.

1.6 RESUMO DO CAPÍTULO

Resumo

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

Programação Processo de criação 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.

³ 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.

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 \(2024\)](#).

1.7 ATIVIDADES PROPOSTAS

Exercícios de fixação

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

Questão 2 Faça uma breve comparação entre Python e Scratch.

Questão 3 Quais são as cinco linguagens de programação mais populares atualmente? Procure na *web* pelo índice TIOBE.

Exercícios de complementares

Questão 1 Por que dizemos que computadores entendem “Os e Is”? Qual a importância de uma linguagem de programação nesse contexto?

Algoritmo

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

“Por onde eu começo?”, perguntou Lara.

2.1 O QUE É UM ALGORITMO?

Antes de entender detalhes de como algoritmo funcionam, é importante entender 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 resolução de problemas semelhantes.

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

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

Definição: Algoritmo.

Um algoritmo é um conjunto de instruções precisas e ordenadas que levam à resolução de um problema ou a 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 seguir esses passos corretamente, Lara possivelmente fará um bolo delicioso. Da mesma forma, os algoritmos na programação são usados para resolver problemas e executar tarefas.

O ser humano executa algoritmos o tempo todo em seu cotidiano. Construir e executar um algoritmo independe 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 o solucionar o problema?

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

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 anda mesmo azarada. Descobriu que uma lâmpada da sua residência queimou. Como seria um algoritmo para trocar uma lâmpada queimada?

Exemplo: Algoritmo para trocar uma lâmpada queimada.

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

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

Você pode ter pensado em soluções diferentes das apresentados nos exemplos anteriores. Veja que isso não quer dizer que a solução pensada é incorreta. É possível encontrar diferentes soluções válidas para um mesmo problema. Em outras palavras, diferentes algoritmos podem ser concluídos para atingir um mesmo objetivo.

2.2 IMPORTÂNCIA DE ALGORITMOS NA PROGRAMAÇÃO

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

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

Entender como os algoritmos funcionam é essencial para se tornar um programador habilidoso. Ao compreender os princípios por trás dos algoritmos, é possível escrever códigos eficientes, resolver problemas complexos e criar softwares diversos.

2.3 COMO CONSTRUIR UM ALGORITMO?

Criar um algoritmo pode parecer assustador no início, mas, na verdade, é bastante simples. Algumas etapas básicas que você pode seguir:

Entenda o problema. Antes de começar a escrever um algoritmo, você precisa entender completamente o problema que está tentando resolver. Isso envolve 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 hora 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.

Alerta: 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

Há diferentes formas de expressar algoritmos. A forma apresentada na [seção 2.2](#) é chamada de linguagem natural. Algoritmos são escritos em *códigos-fonte* no contexto de programação.

Os algoritmos são essenciais para a programação e são a base de toda a tecnologia moderna. Eles permitem que os computadores executem tarefas complexas e resolvam problemas de maneira eficiente. Ao entender os princípios básicos dos algoritmos e como criá-los, você estará bem equipado para se tornar um programador habilidoso e criar soluções inovadoras.

Definição: Código-fonte.

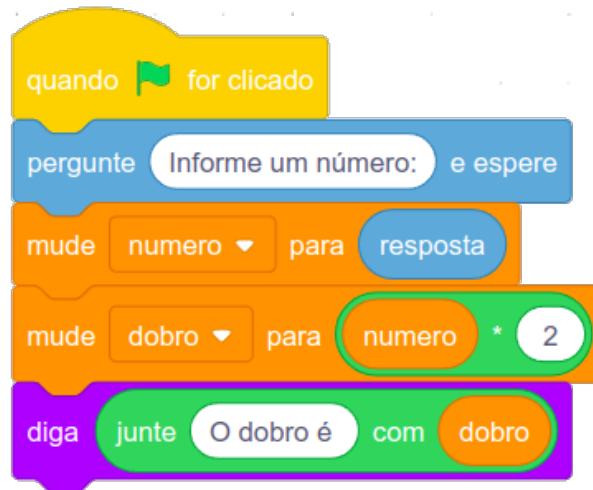
Código-fonte é um conjunto de instruções escritas em uma linguagem de programação específica que formam um software. O código-fonte é um texto ou esquema visual legível por seres humanos que define o comportamento e a lógica de um programa. Quando um programador cria um código-fonte, ele define como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

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 pensar no problema de calcular o dobro de um número real. De forma matemática, podemos pensar no algoritmo por meio de 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 no [Programa 1](#). O mesmo algoritmo em Scratch é apresentado no [Programa 2](#). Não se preocupe em entender os comandos e as instruções agora.

Programa 1 Dobro de um número em Python.

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

Programa 2 Dobro de um número em Scratch.

Fonte: Elaborado pelos autores.

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

O mundo da programação é vasto e complexo, repleto de termos técnicos e conceitos que podem parecer intimidador 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 permitem ao desenvolvedor criar softwares executados pelos computadores. Quando um conjunto abrangente de ferramentas e um editor de código-fonte são fornecidos ao mesmo tempo, 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. O ambiente fornece geralmente 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 na qual 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/Interpretador Plataformas que "convertem" o código-fonte em código de máquina para que o computador seja capaz de executar as instruções.

Depurador Ferramenta essencial para encontrar e corrigir erros em um código-fonte. Um depurador permite que o código-fonte seja inspecionado instrução a instrução.

3.1 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 possui 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. O site contém também documentação, notícias, comunidades e outros recursos úteis para desenvolvedores.

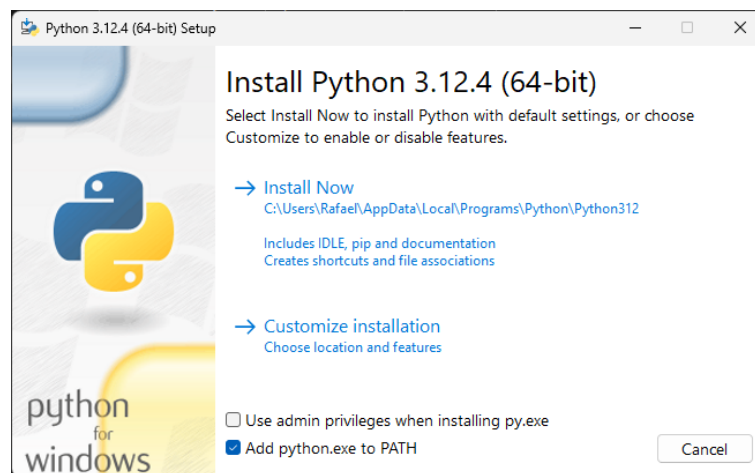
A instalação típica do ambiente de desenvolvimento inclui: interpretador, depurador, bibliotecas padrão, instalador de pacotes e IDE.

3.1.1 Microsoft Windows

A instalação no Microsoft Windows deve ser realizada da seguinte forma. Em python.org, acesse a página de *download* e baixe o instalador Python compatível com o computador utilizado. No momento da escrita deste livro, a versão mais recente é a 3.12.4. Não havendo problemas de compatibilidade com o computador e com os projetos, procure sempre utilizar a versão mais nova.

Localize e execute o arquivo baixado para iniciar a instalação. Na primeira tela, veja a [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.

Teste se o ambiente está funcionando, abrindo o *prompt* de comando do Microsoft Windows e digitando: `py`. O comando executa o interpretador da linguagem. O resultado deve ser semelhante ao apresentado na [Figura 5](#). Para sair do interpretador do 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 durante seus estudos. Entretanto, destacamos que é perfeitamente possível utilizar a IDE IDLE ou qualquer outro editor de código-fonte compatível com a linguagem.

3.1.2 GNU/Linux

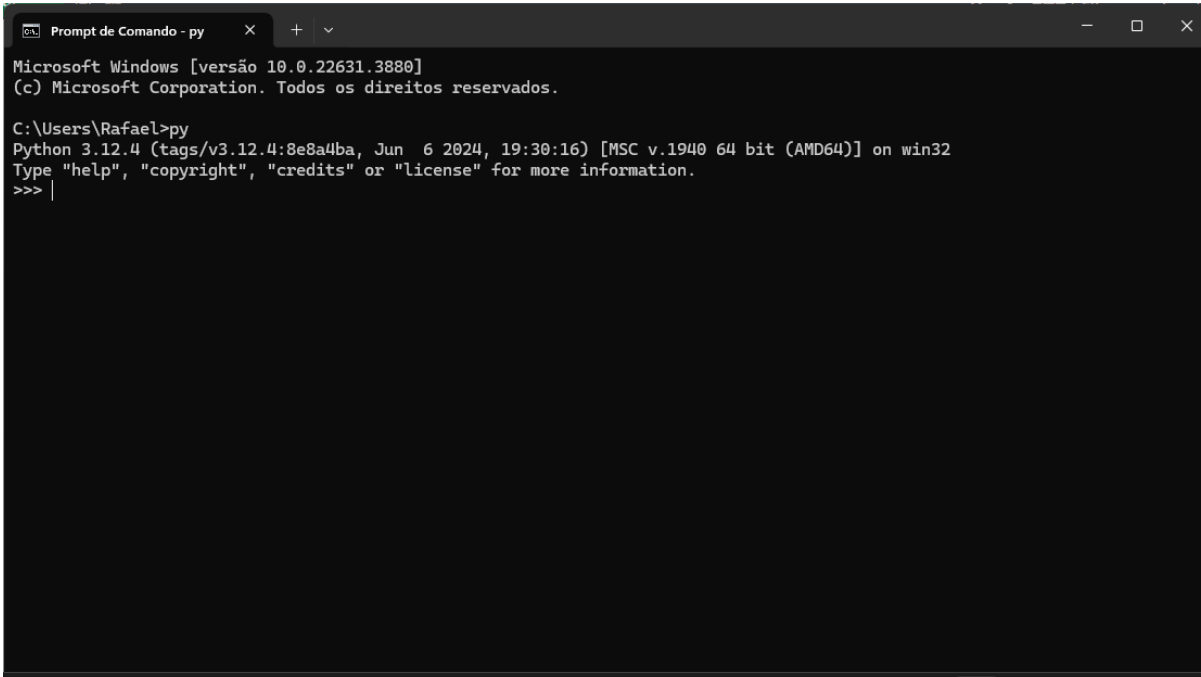
A maioria das distribuições GNU/Linux modernas vem com o ambiente de desenvolvimento Python instalado por padrão. Caso não seja o caso, é possível instalar diretamente a partir do gerenciador de pacotes do sistema operacional.

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).

O ambiente de desenvolvimento Python é geralmente acompanhado do pip. Caso a distribuição não tenha o pacote instalado, repita o processo descrito em [Quadro 1](#), substituindo `python3` por `python3-pip`.

O interpretador Python é executado por meio do comando `python3` em ambientes GNU/Linux. O teste de execução obtém resultado similar ao obtido em um sistema operacional Microsoft Windows (veja a [Figura 5](#)).

Figura 5 – Teste do ambiente Python no Microsoft Windows

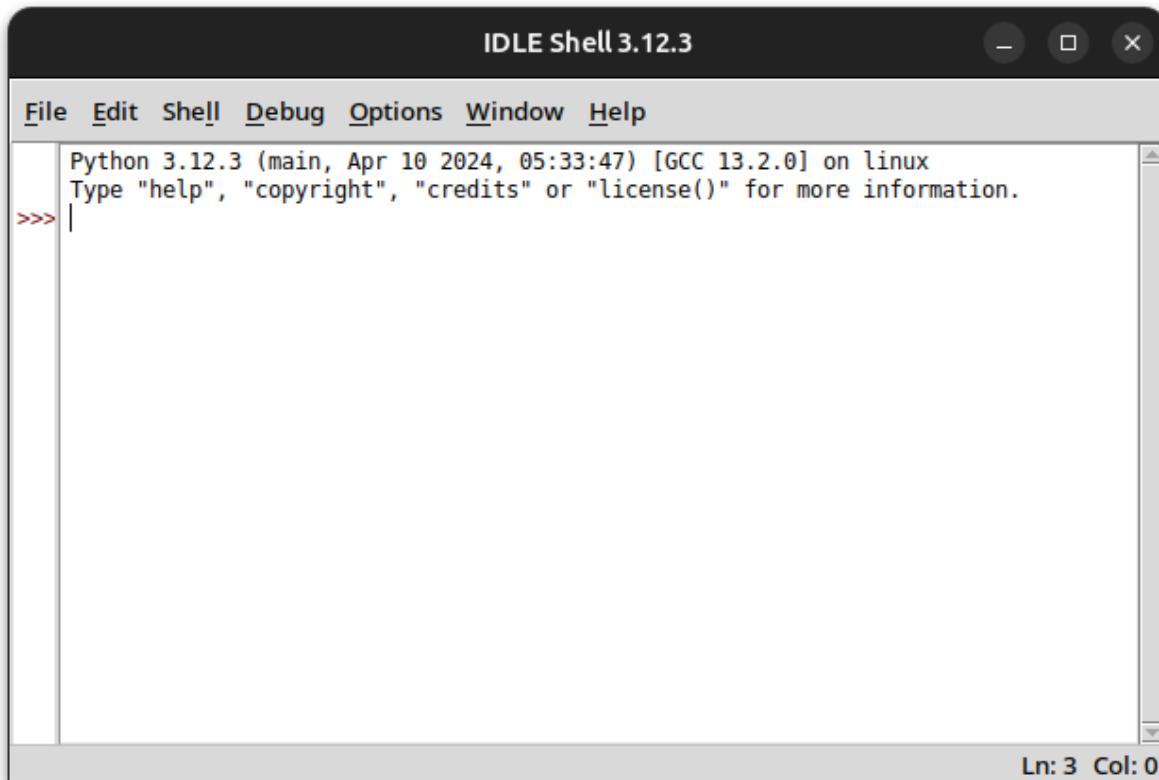


```
Prompt de Comando - py
Microsoft Windows [versão 10.0.22631.3880]
(c) Microsoft Corporation. Todos os direitos reservados.

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 em um GNU/Linux



```
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.

Quadro 1 – Instalação do ambiente de desenvolvimento Python em distribuições GNU/Linux com gerenciadores de pacotes apt ou yum

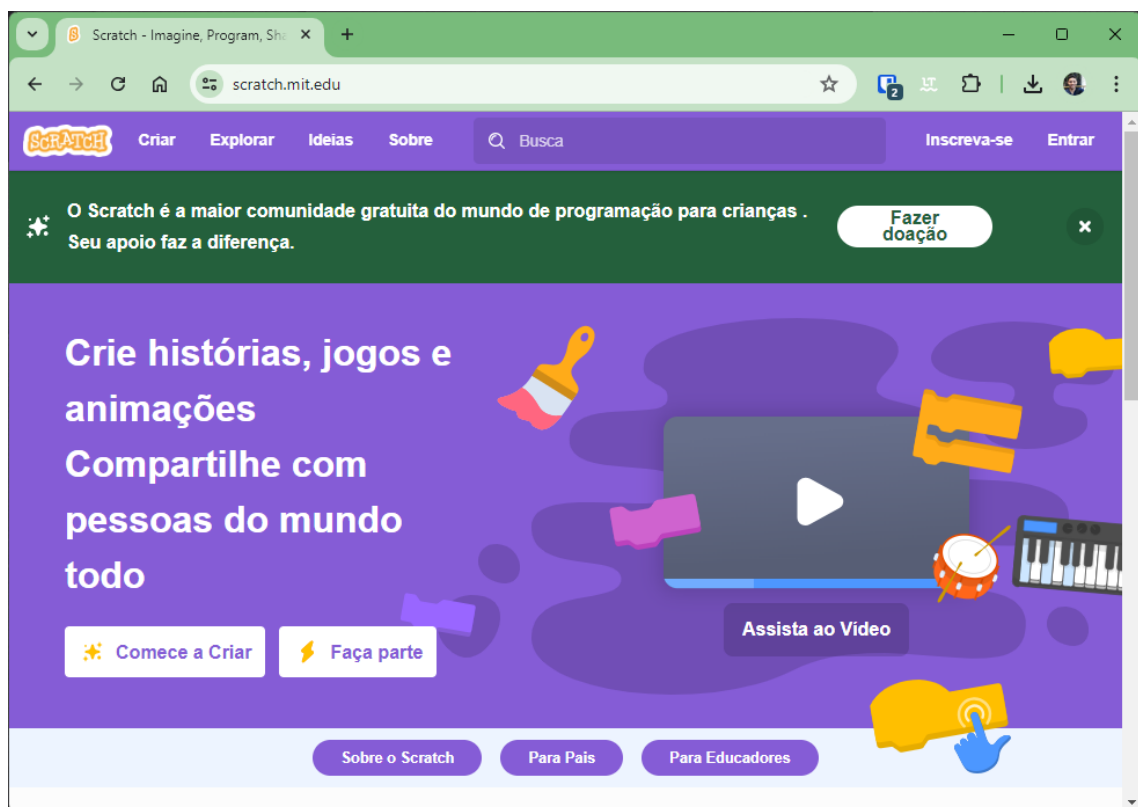
Gerenciador de Pacote	Comando
apt	# apt-get install python3
yum	# yum install python3

Fonte: Elaborado pelo autor.

3.2 SCRATCH

O ambiente desenvolvimento Scratch está disponível no site oficial scratch.mit.edu, veja a [Figura 7](#). O ambiente pode ser utilizado diretamente por qualquer navegador web 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.

Figura 7 – Site oficial do Scratch.



Fonte: Elaborada pelo autor.

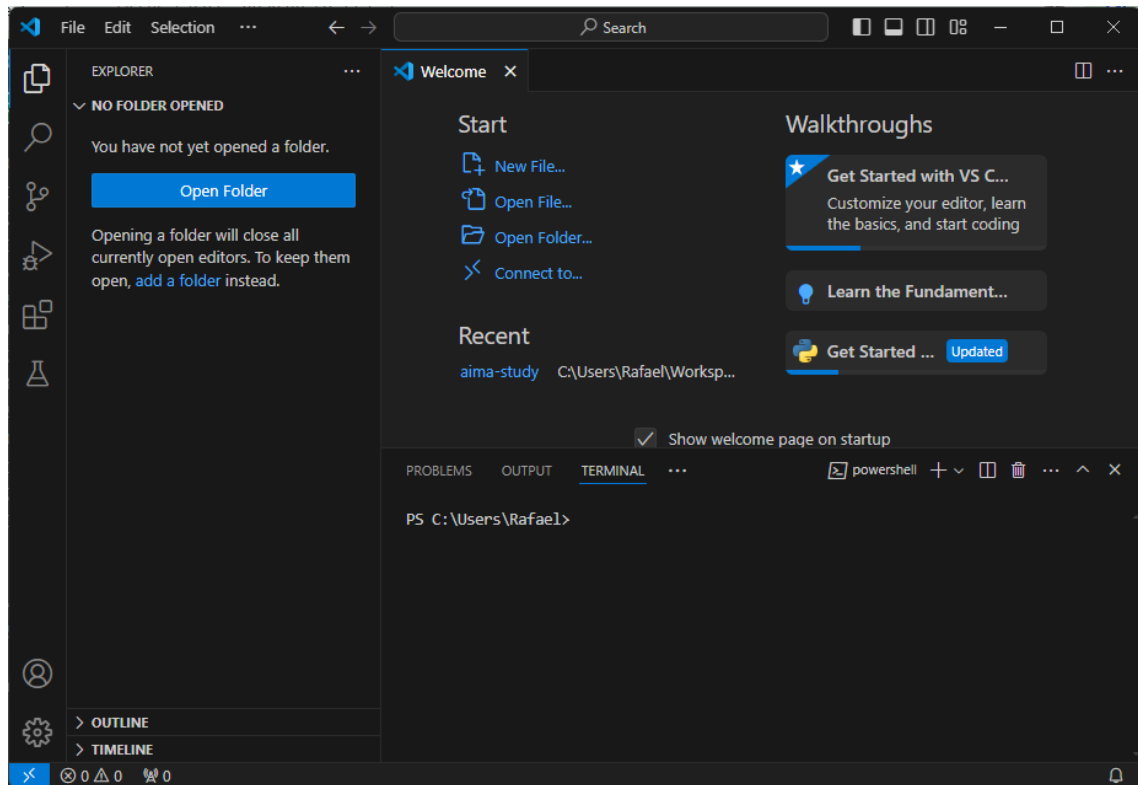
3.3 VISUAL STUDIO CODE

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 está disponível em code.visualstudio.com.

O VS Code suporta uma grande variedade de linguagens de programação. 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 pelo autor.

O suporte do VS Code à linguagem Python é aprimorado com a instalação das extensões Python e Pylance instaladas. Faça a instalação conforme apresentado na [Figura 9](#).

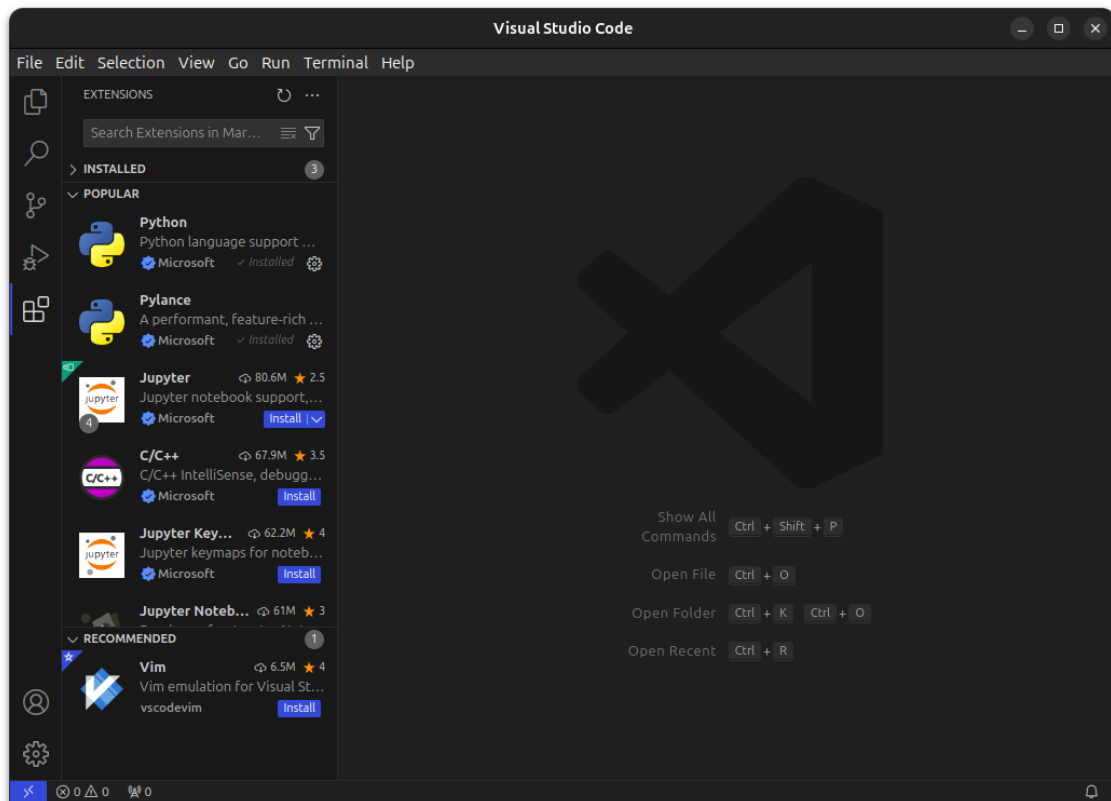
A instalação de extensões do VS Code é simplificada por meio da aba `|Extensions|`. Após encontrar uma extensão, basta clicar na opção `|Install|`.

3.4 RESUMO DO CAPÍTULO

Resumo

IDE Sigla para Integrated Development Environment (Ambiente de Desenvolvimento Integrado, em português). É um software que fornece um conjunto de ferramentas e recursos para facilitar o desenvolvimento de software.

Editor de código-fonte Ferramenta que permite escrever, editar e gerenciar o código-fonte de softwares. Diferenciam-se de editores de texto tradicionais por apresentar uma ampla variedade de recursos para tornar programação mais dinâmica e intuitiva.

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

Fonte: Elaborada pelo autor.

3.5 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.

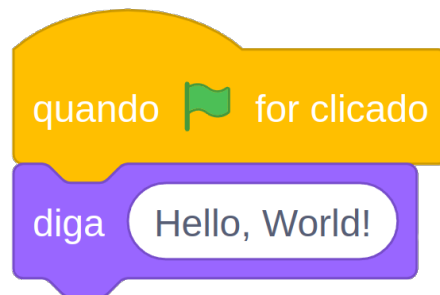
Primeiro programa

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 exemplo em Python está no [Programa 3](#), e o mesmo algoritmo em Scratch está 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.

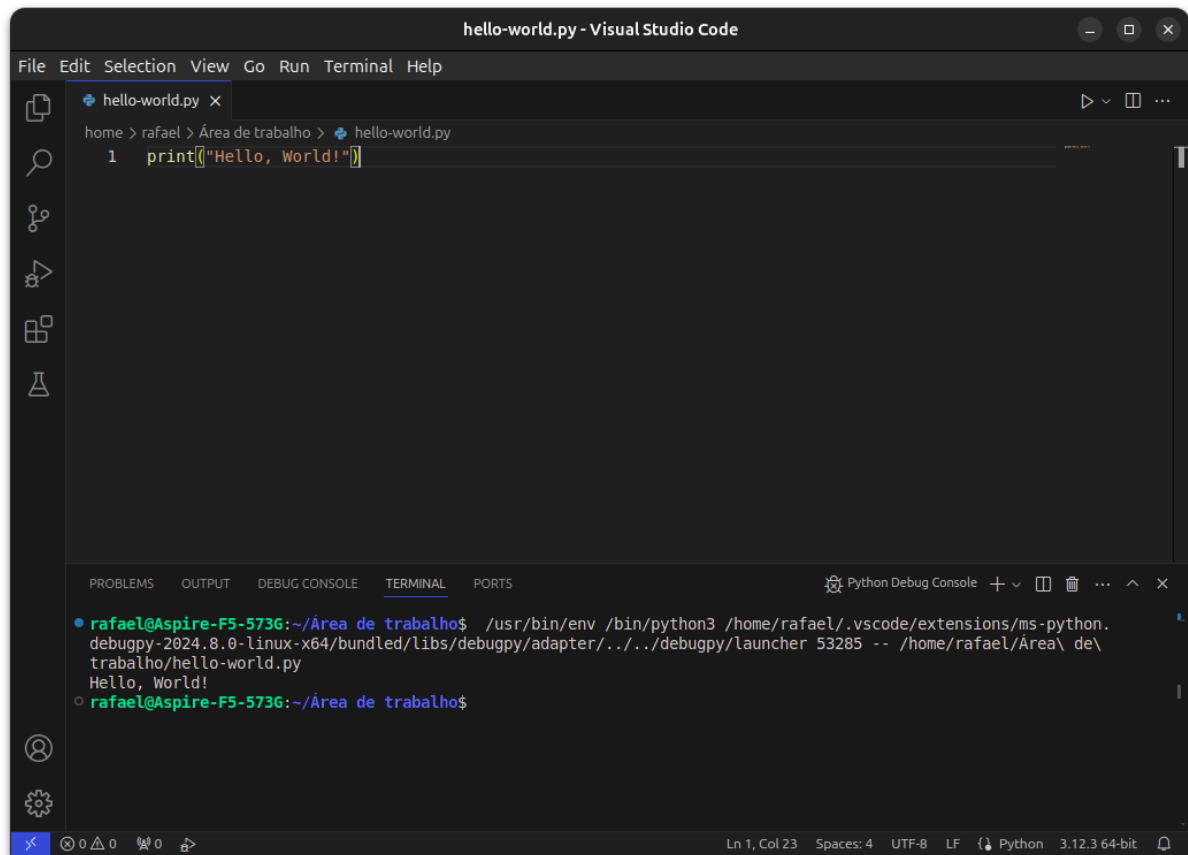
4.1.1 Execução de programas pelo Visual Studio Code

Depois de configurar o VS Code e selecionar o interpretador Python, crie e salve seu código-fonte. Para criar um novo arquivo, vá ao menu `|File|` e selecione `|New File|` ou pressione `<Ctrl+N>`. Digite seu código no novo 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`.

Execute o programa escolhendo `|Run Without Debugging|` no menu `|Run|` ou pressionando `<Ctrl+F5>`. A execução será exibida em uma aba de CLI, conforme mostrado na [Figura 10](#).

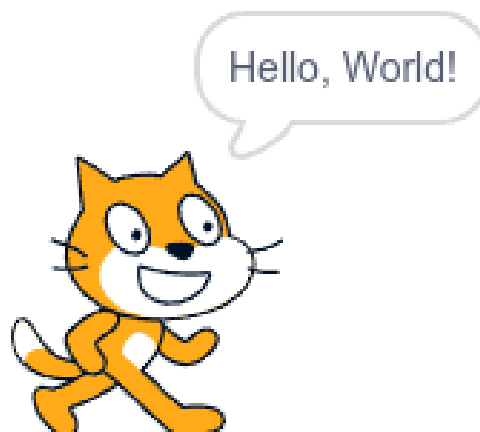
Figura 10 – “Hello, World!” em execução no Visual Studio Code



Fonte: Elaborada pelo autor.

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 pelo autor.

4.1.2 Execução pela CLI

Para executar via CLI, abra o *prompt* de comando no Windows ou o terminal no GNU/Linux e navegue até o diretório onde o arquivo `hello-world.py` está salvo.

No *prompt* de comando do Microsoft Windows, use:

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

No terminal do GNU/Linux, use:

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

4.2 A FUNÇÃO PRINT()

A função `print()` exibe informações na tela. Trata-se de uma função embutida (*built-in*) que imprime uma ou mais expressões, convertendo-as para texto (do tipo *string*) e enviando a mensagem resultante para a tela da CLI. 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 requer que as *strings* estejam entre aspas duplas (") ou aspas simples ('). 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 na saída "Hello, World!".

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

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

Por padrão, a função usa 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.

4.3 COMENTÁRIOS

Comentários em Python são trechos de texto no código-fonte ignorados pelo interpretador. 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

O interpretador analisa sintaticamente o código-fonte ao executá-lo. Caso alguma regra de sintaxe não seja atendida, um erro será informado durante a execução. O VS Code também consegue detectar alguns erros de sintaxe antes da execução do código, oferecendo uma forma de corrigir problemas previamente.

A sintaxe de uma linguagem de programação define como os códigos devem ser estruturados e escritos para que o interpretador ou compilador possa compreendê-los e executá-los corretamente. Erros de sintaxe geralmente ocorrem quando o código não segue as regras estabelecidas pela linguagem, como falta de parênteses, aspas não fechadas ou indentação incorreta.

Para garantir que o código esteja correto, é importante prestar atenção às mensagens de erro fornecidas pelo interpretador e pelo editor, e seguir as regras de sintaxe da linguagem Python. Utilizar ferramentas de desenvolvimento como o VS Code pode facilitar a identificação e correção de erros de sintaxe antes da execução do programa.

Um exemplo de código-fonte com erro sintático e a mensagem de erro informada pelo compilador é apresentado na [Figura 12](#).

Alerta: Erro semântico.

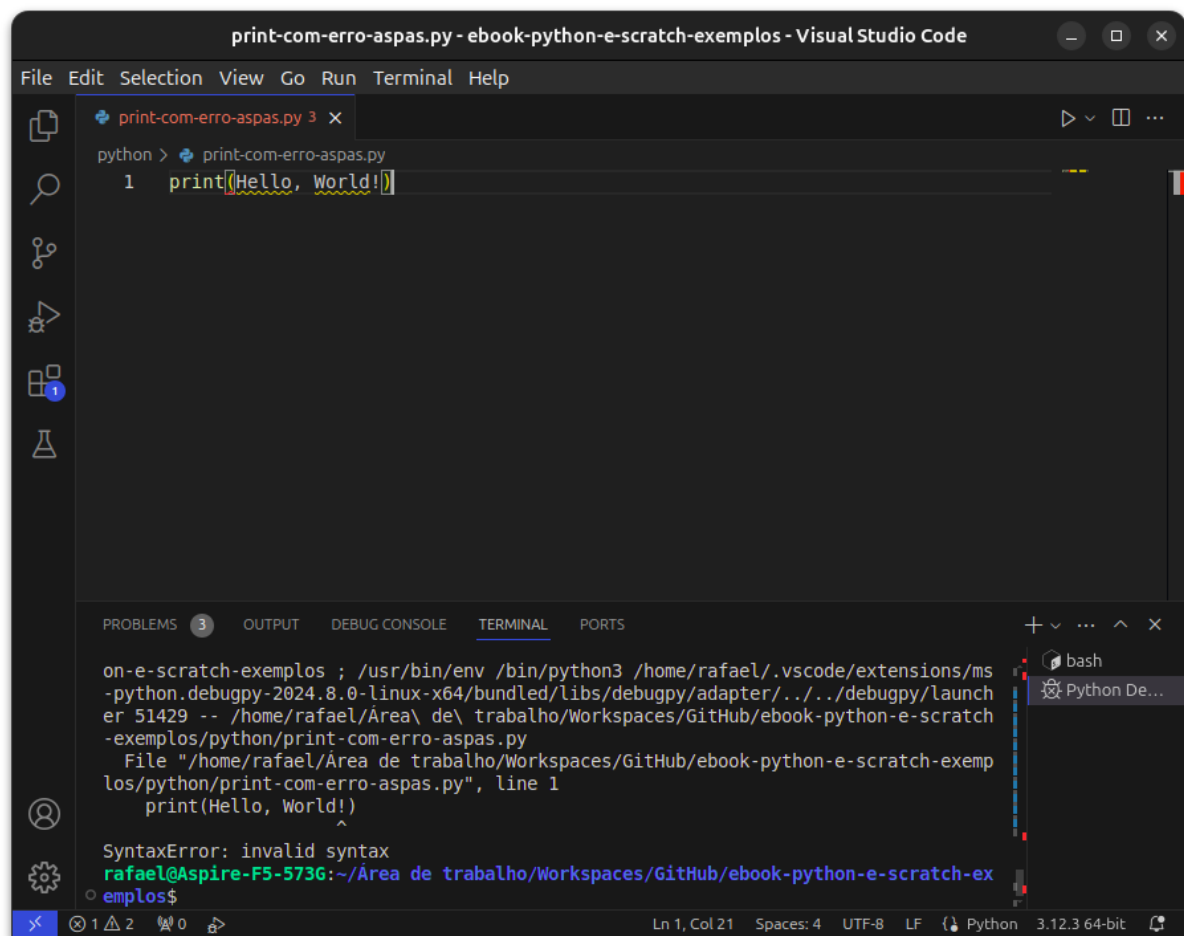
Muitos programadores inexperientes podem achar que, se o código-fonte é executado sem erros, ele está correto. No entanto, isso não é verdade. Erros semânticos, ou erros na lógica do algoritmo, não são detectados pelo interpretador e podem ser muito difíceis de identificar. Para detectar e corrigir esses erros, a depuração é uma ferramenta valiosa.

4.5 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. Isso 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. Para uma comparação, no Scratch, os blocos de código são executados da mesma maneira sequencial.

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



Fonte: Elaborada pelo autor.

Exemplo: Imprimir uma lista dos cinco países com maior extensão territorial.

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 o exemplo é apresentada no Código-fonte [Programa 8](#). O interpretador Python executa cada linha de código em sequência, começando pela impressão de “Rússia” até “Brasil”. Ao digitar o código em um arquivo *.py no VS Code, o editor mostra números de linhas que vão 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 algoritmo em Scratch é apresentada no [Programa 9](#). Observe que o fluxo de execução sequencial segue o encaixe de cada bloco de cima para baixo, garantindo que cada instrução seja executada exatamente nessa ordem. Além disso, veja que é usado um bloco

`diga` com 2 segundos para que apareça um balão de diálogo em sequência do outro. Isso simula o comportamento de nova linha na CLI.

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



Você pode ter pensado em uma solução diferente que ainda assim é correta. Lembre-se de que existem várias maneiras de resolver o mesmo problema ou realizar a mesma tarefa.

4.6 EXEMPLOS COMENTADOS

Exemplo:

Enunciado:

Implemente 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* (uma sequência de caracteres) como argumento. A mensagem contida nesta *string* é exibida na linha de comando (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:

Enunciado:

Implemente um programa que exiba o seguinte texto, garantindo que a saída reproduza a formatação e o número de linhas do texto apresentado:

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")
```

Neste exemplo, 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")
```

Neste exemplo, 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*. No exemplo a seguir, 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.

Neste 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 ATIVIDADES PROPOSTAS

Exercícios de fixação

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

Questão 2 Implemente um programa que imprima o seu nome completo.

Questão 3 Implemente um programa que imprima o texto a seguir, reproduzindo a formatação:

Lista de compras:

- Arroz
- Feijão
- Óleo

Exercícios Complementares

Questão 1 O código-fonte a seguir está correto sintaticamente? Caso contrário, explique o que está errado.

```
print(Alan Turing)
```

Questão 2 O código-fonte `print("1", "2", "3", "4")` tem como resultado a saída:

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

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

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

Variáveis, tipos de dados e operadores aritméticos

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 usadas para armazenar dados que podem ser modificados e acessados em várias partes do código.

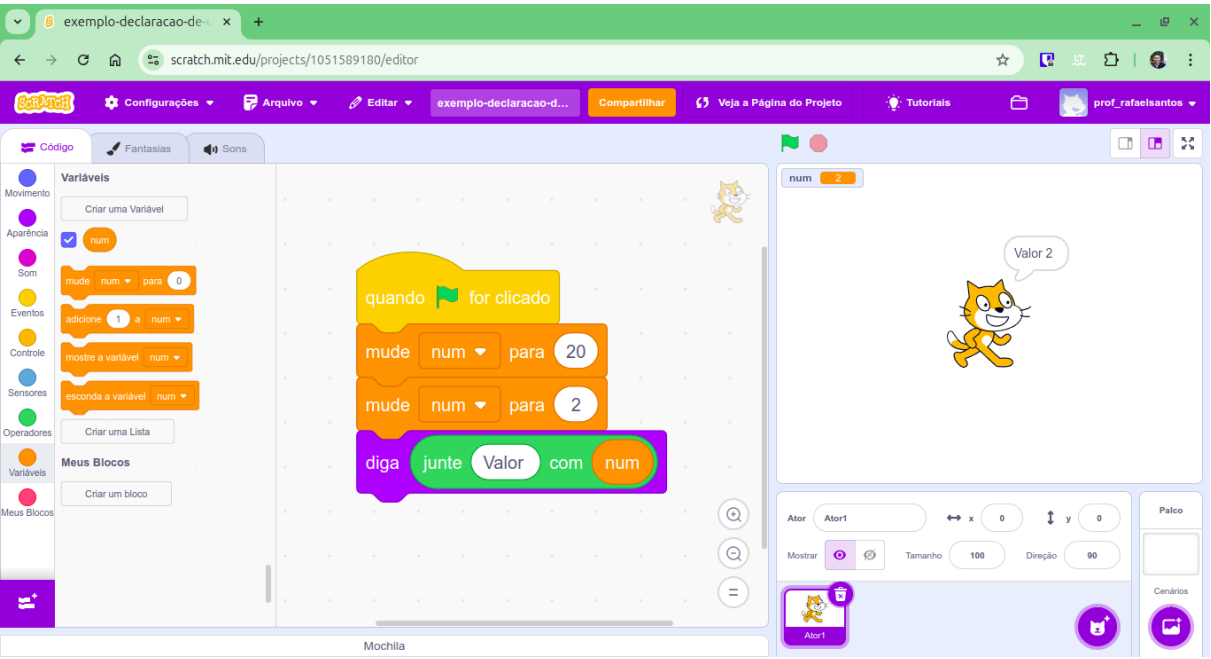
Literal Um valor diretamente representado no código-fonte de um programa. Literais geralmente são números, *strings* ou valores booleanos, explicitamente escritos e reconhecidos como constantes. Eles representam valores fixos que não mudam durante a execução do programa.

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.

Em Python, uma linguagem de tipagem dinâmica, você não precisa declarar o tipo de uma variável explicitamente. 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.

Um programa pode ter múltiplas variáveis, cada uma com um nome exclusivo definido pelo programador. O nome de uma variável deve começar com uma letra ou subtraço (`_`), seguido por letras, números ou subtraços. Não pode começar com números e deve evitar palavras reservadas da linguagem de programação. O [Quadro 2](#) apresenta as principais palavras reservadas em Python.

Definição: Palavra Reservada.

Palavras reservadas são termos que definem a sintaxe e a gramática (estrutura) da linguagem e não podem ser usados 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 pelo autor.

Alerta: Boas Práticas para nomes de variáveis.

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.

Case sensitive. Lembre-se de que Python diferencia letras maiúsculas e minúsculas. Por exemplo, `Numero` é uma variável diferente de `numero`.

5.1.1 Declaração e atribuição de valores a variáveis

Criar uma variável é conhecido como declaração. Em Python, isso é feito de maneira simples, sem a necessidade de especificar o tipo da variável. Você atribui um valor a uma variável usando o **operador de atribuição** (`=`).

Veja o exemplo do [Programa 10](#). A variável `num` é declarada na linha 1, pois é a primeira vez que aparece no código. Após a declaração, o valor 20 é armazenado na variável. Na linha 2, o valor da variável é modificado para 2 com o uso do operador de atribuição. Por fim, 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, e em atribuições subsequentes, o valor armazenado é atualizado. É possível utilizar quantas variáveis forem necessárias. O valor somente é alterado após uma operação de atribuição.

O exemplo do [Programa 11](#) apresenta duas variáveis `num1` e `num2`, que armazenam respectivamente os valores 4 e 7. Além disso, acontece a impressão por meio do comando `print`. Veja que é possível combinar a impressão de strings e das variáveis em um único comando.

Programa 11 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, num2) #Impressão de num1 e num2
```

É recomendável usar nomes de variáveis que sejam descritivos e que representem claramente o valor que armazenam. Além disso, adotar o estilo de nomenclatura com letras minúsculas e subtraços (`_`) para separar palavras facilita a leitura e a manutenção do código-fonte.

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 (True) e falso (False).	<code>a = True</code>
<code>complex</code>	Complexo	Representa números complexos, com a parte imaginária denotada por “ <i>j</i> ”.	<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 = "Rafael"</code>
<code>None</code>	Nulo	Representa a ausência de valor.	<code>a = None</code>

Fonte: Elaborado pelo autor.

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.

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 OPERAÇÕES ARITMÉTICAS

Os operadores aritméticos em Python são utilizados para realizar operações matemáticas básicas como adição, subtração, multiplicação, divisão, divisão inteira, módulo (resto da divisão) e exponenciação. A seguir operadores e suas funcionalidades:

5.3.1 Operador de adição

```
resultado = 5 + 3  #0 resultado é 8
```

5.3.2 Operador de subtração

```
resultado = 5 - 3  #0 resultado é 2
```

5.3.3 Operador de multiplicação

```
resultado = 5 * 3  #0 resultado é 15
```

5.3.4 Operador de divisão

```
resultado = 5 / 2 #0 resultado é 2.5
```

5.3.5 Operador de divisão inteira

```
resultado = 5 // 2 #0 resultado é 2
```

5.3.6 Operador de módulo ou resto da divisão

```
resultado = 5 % 2 #0 resultado é 1
```

5.3.7 Operador de exponenciação

```
resultado = 5 ** 2 #0 resultado é 25
```

[Programa 12](#) é um exemplo de uso de todos os operadores aritméticos. Observe atentamente os valores de entrada e o resultado de cada operação.

Programa 12 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 divisao1 = a / b       # 3.3333...
8 divisao2 = 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:", divisao1)
16 print("Divisão inteira:", divisao2)
17 print("Resto da divisão:", modulo)
18 print("Exponenciação:", exponenciacao)
```

5.4 PRECEDÊNCIA E ASSOCIATIVIDADE ENTRE OPERAÇÕES ARITMÉTICAS

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, como 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 devem ser avaliados na ordem em que aparecem na linha de código, isto é, da esquerda para a direita. Por exemplo, em uma expressão

envolvendo 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. Por isso, 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 (do mais alta para o mais baixa).

Operador	Operação	Associatividade
()	Parênteses	Não se aplica. Dentro para fora.
**	Exponenciação	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 pelo autor.

Caso seja necessário, é possível alterar a precedência dos operadores utilizando parênteses. Operações em parênteses têm a mais alta precedência, sendo avaliadas primeiro. Veja a diferença produzida no exemplo:

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

5.5 OPERAÇÕES DE ATRIBUIÇÃO

O operador de atribuição em Python (=) é usado 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.

Alerta: Precedência do operador de atribuição em relação aos demais operadores.

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.

5.5.1 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
```

5.5.2 Operador de atribuição combinado

O operador de atribuição combinado realiza uma operação aritmética e atribui o resultado à mesma variável em uma única etapa. Aqui estão os principais operadores de atribuição combinados:

5.5.2.1 Operador de adição e atribuição

```
x = 5  
x += 3  #É equivalente à expressão x = x + 3. Temos x igual a 8.
```

5.5.2.2 Operador de subtração e atribuição

```
x = 5  
x -= 3  #É equivalente à expressão x = x - 3. Temos x igual a 2.
```

5.5.2.3 Operador de multiplicação e atribuição

```
x = 5  
x *= 3  #É equivalente à expressão x = x * 3. Temos x igual a 15.
```

5.5.2.4 Operador de divisão e atribuição

```
x = 5  
x /= 2  #É equivalente à expressão x = x / 2. Temos x igual a 2,5.
```

5.5.2.5 Operador de divisão inteira e atribuição

```
x = 5  
x //= 2  #É equivalente à expressão x = x // 2. Temos x igual a 2.
```

5.5.2.6 Operador de módulo e atribuição

```
x = 5  
x %= 2  #É equivalente à expressão x = x % 2. Temos x igual a 1.
```

5.5.2.7 Operador de exponenciação e atribuição

```
x = 5  
x **= 2  #É equivalente à expressão x = x ** 2. Temos x igual a 25.
```

5.6 EXEMPLOS COMENTADOS

5.6.1 Calculadora da área do retângulo

Exemplo: Calculadora da área do retângulo.

Enunciado:

Implementar um programa que calcule a área do retângulo, dados os valores de entrada da base e da altura. Fórmula matemática:

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

Solução:

5.6.2 Conversor de distância

Exemplo: Conversor de distância de metros para quilômetros.

Enunciado:

Implementar um programa que, a partir de uma distância em metros, faça a conversão para quilômetros. Fórmula matemática:

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

Solução:

5.6.3 Calculadora de Índice de Massa Corpórea (IMC)

Exemplo: Calculadora de IMC.

Enunciado:

Implementar um programa que calcule o IMC (Índice de Massa Corpórea) de uma pessoa, dados os valores de entrada de peso e altura. Fórmula matemática:

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

Solução:

5.6.4 Teste de mesa de código-fonte

Exemplo: Teste de mesa do código-fonte A.

Enunciado:

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:

A expressão é matematicamente:

$$\text{num1} = (8 + 2) \times (5 - 3) / 2$$

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

$$8 + 2 = 10$$

$$5 - 3 = 2$$

Substituímos os resultados na expressão principal:

$$\text{num1} = 10 \times 2 / 2$$

Realizamos a multiplicação e a divisão:

$$10 \times 2 = 20$$

$$20 \div 2 = 10.0$$

Valor de num1 é 10.0. A partir daqui, podemos calcular num2. A expressão é:

$$\text{num2} = \text{num1} \text{ resto da } \div 7$$

Usamos o valor de num1 calculado anteriormente:

$$\text{num2} = 10.0 \text{ resto da } \div 7$$

Realizamos a operação de módulo:

$$10.0 \div 7 = 1 \text{ com resto } 3.0$$

Realizamos a operação de módulo:

$$10.0 \div 7 = 1 \text{ com resto } 3.0$$

Portanto, num2 é 3.0.

Exemplo: Teste de mesa do código-fonte B.**Enunciado:**

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:

Vamos começar calculando a expressão para `resultado`. A expressão é:

$$\text{resultado} = (a + b) \times c - \left(\frac{a}{b}\right) + (c \text{ resto da } \div b)^2$$

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

resolvemos as expressões dentro dos parênteses:

$$a + b = 4 + 2 = 6$$

$$c \text{ resto da } \div b = 5 \text{ resto da } \div 2 = 1$$

Substituímos os resultados na expressão principal:

$$\text{resultado} = (6 \times 5) - \left(\frac{4}{2}\right) + 1^2$$

Realizamos as operações seguintes:

$$6 \times 5 = 30$$

$$\frac{4}{2} = 2$$

$$1^2 = 1$$

Substituímos os resultados finais na expressão principal:

$$\text{resultado} = 30 - 2 + 1$$

$$30 - 2 = 28$$

$$28 + 1 = 29$$

Portanto, o valor de `resultado` é 29.

5.6.5 Conversor de tempo

Exemplo: Conversor de tempo em segundos.

Enunciado:

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

Solução:

5.7 ATIVIDADES PROPOSTAS

Exercícios de fixação

Questão 1 Implementar um programa que calcule a área do triângulo, a partir dos valores de entrada de base e altura. Fórmula matemática:

$$\text{área} = \frac{\text{base} \times \text{altura}}{2}$$

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

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

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.

$$\text{média} = \frac{\text{num}_1 + \text{num}_2 + \text{num}_3}{3}$$

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

Em construção...

Interatividade com o usuário

Em muitas aplicações, a interatividade com o usuário é fundamental. Receber dados do usuário em tempo real permite que nossos programas sejam dinâmicos e responsivos. Em Python, a função `input()` é a ferramenta que usamos para captar essas entradas. Neste capítulo, vamos explorar como essa função funciona e como podemos utilizá-la de maneira eficaz.

6.1 A FUNÇÃO `INPUT()`

A função `input()` em Python permite que seu programa pause e aguarde que o usuário digite algum texto. Esse texto é então retornado como uma *string*, que pode ser armazenada em uma variável para uso posterior.

Programa 13 apresenta um primeiro exemplo de interatividade com o usuário. Na linha 1, o programa apresenta a mensagem “Nome:” e aguarda que o usuário digite alguma coisa e pressione <Enter>. O texto digitado é capturado e armazenado na variável `nome`. A função `print()` na linha 2 exibe a saída com a *string* armazenada na variável.

Programa 13 Imprimir o nome informado pelo usuário.

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

6.1.1 Convertendo tipos de dados

A função `input()` sempre retorna uma *string*, independentemente do que o usuário digite. Em muitos casos, precisamos converter essa *string* para outro tipo de dado, como um *int* ou um *float*. **Programa 14** apresenta um exemplo dessa situação.

Programa 14 Imprimir a idade e altura informadas pelo usuário.

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

A função `input()` na linha de código 1 exibe a mensagem e captura a entrada do usuário. Antes da atribuição do valor capturado à variável `idade`, acontece uma conversão. Lembre-se que o retorno sempre é do tipo *string*. A função `int()` converte a *string* para *int*. A linha

de código 2 segue o mesmo comportamento, mas a conversão é realizada usando a função `float()` porque o número capturado pode ser do tipo `float`.

É importante entender que, quando temos funções aninhadas, a execução sempre acontece de dentro para fora.

Alerta: Erros de conversão de tipos

O usuário pode muitas vezes digitar de forma incorreta a entrada e a conversão de tipo não ser possível. Quando isso acontece, o interpretador exibe uma mensagem de erro.

6.2 A FUNÇÃO `TYPE()`

A função `type()` em Python é usada para verificar o tipo de dados de uma variável ou expressão. Ela retorna o tipo do objeto passado como argumento. Isso é útil para depuração, validação de entrada e compreensão do comportamento do seu código.

Programa 15 apresenta um exemplo de uso da função `type()`. É importante saber que o tipo `string` em Python é escrito sempre como `str`.

Programa 15 Uso da função `type()`.

```
1 x = 10
2 print(type(x))      #class 'int'
3
4 y = 3.14
5 print(type(y))      #class 'float'
6
7 nome = "Alan Turing"
8 print(type(nome))    #class 'str'
```

6.3 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*.

6.3.1 Formatação de valores `int`

Para formatar um `int` sem nenhuma especificação adicional, basta utilizar 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, você pode usar `: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
```

6.3.2 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
```

6.4 ATIVIDADES PROPOSTAS

Exercícios de fixação

Questão 1 Implementar um programa que calcule a área do quadro, a partir do valor de lado informado pelo usuário. Fórmula matemática:

$$\text{área} = \text{lado} \times \text{lado}$$

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

$$^{\circ}\text{K} = ^{\circ}\text{C} \times +273.$$

Questão 3 Implementar um programa que, dados quatro números informados pelo usuário, 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. Fórmula matemática:

$$\text{média} = \frac{\text{num}_1 + \text{num}_2 + \text{num}_3 + \text{num}_4}{4}$$

Exercícios complementares

Em construção...

Operadores relacionais e operadores lógicos

A maioria dos programas que utilizamos no cotidiano implementa algoritmos de decisão. Por exemplo, uma janela pode solicitar ao usuário que confirme ou cancele uma ação. Entender o funcionamento desses recursos em lógica de programação é fundamental. Neste capítulo, apresentamos os operadores relacionais e lógicos, recursos essenciais para a construção de tais algoritmos.

7.1 OPERADORES RELACIONAIS

Os operadores relacionais são usados para comparar dois valores e sempre retornam um resultado booleano (**True** ou **False**). Eles são fundamentais na lógica de programação, pois permitem que os programas tomem decisões com base em condições.

A linguagem Python possui seis operadores relacionais: igual a ($=$), diferente de (\neq), maior que ($>$), maior ou igual a (\geq), menor que ($<$) e menor ou igual a (\leq). A seguir são apresentados exemplos de funcionamento de cada um desses operadores.

7.1.1 Operador “igual a” ($=$)

Verifica se dois valores são iguais.

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

7.1.2 Operador “diferente de” (\neq)

Verifica se dois valores são diferentes.

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

7.1.3 Operador “maior que” (>)

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

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

7.1.4 Operador “menor que” (<)

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

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

7.1.5 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
```

7.1.6 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
```

7.2 PRECEDÊNCIA E ASSOCIATIVIDADE ENTRE OPERAÇÕES RELACIONAIS

Os operadores relacionais tem o mesmo nível de precedência entre si. A associatividade acontece da esquerda para direita.

7.3 OPERADORES LÓGICOS

Os operadores lógicos são usados para combinar condições e fazer avaliações mais complexas. Eles são muito úteis em estruturas de controle como condicionais e laços de repetição. Operandos avaliados por operadores lógicos devem ser do tipo *bool*. Desta forma, os operandos podem ser uma variável ou uma operação relacional.

O entendimento dos operadores lógicos é tipicamente descrito por meio de tabelas verdades. Uma tabela-verdade mostra todas as combinações possíveis de valores de verdade para as variáveis envolvidas 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.

7.3.1 Operador de conjunção (and)

Representado pela palavra reservada **and**. O operador pode ser interpretado como “e”. Retorna **True** se ambas as condições forem verdadeiras, conforme a tabela-verdade do [Quadro 5](#).

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 5 – 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 pelo autor.

7.3.2 Operador de disjunção (or)

Representado pela palavra reservada **or**. O operador pode ser interpretado como “ou”. Retorna **True** se pelo menos uma das condições for verdadeira. A tabela-verdade do operador é apresentada no [Quadro 6](#).

Quadro 6 – 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 pelo autor.

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

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

7.3.3 Operador de negação (not)

Representado pela palavra reservada **not**. O operador pode ser interpretado como “não”. Inverte o valor da condição, conforme descrito na tabela-verdade apresentada no [Quadro 7](#). Note que a operação é unária, isto é, 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 7 – Tabela-verdade do operador `not` (“não”).

Operando	Resultado
True	False
False	True

Fonte: Elaborado pelo autor.

7.3.4 Precedência e associatividade entre operações lógicas

A ordem de precedência e a associatividade entre as operações lógicas é apresentada no [Quadro 8](#). Assim como acontece com os demais tipos de operações, operações lógicas são avaliadas de acordo com uma ordem definida.

Quadro 8 – Ordem de precedência e associatividade entre operações lógicas (do mais alta para o mais baixa)

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

Fonte: Elaborado pelo autor.

7.4 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 entre operações aritméticas, de atribuição, relacionais e o lógicas. [Quadro 9](#) apresenta essa ordem. Atente-se ao fato que dentro do mesmo tipo também há uma ordem de precedência.

Quadro 9 – Ordem de precedência entre todos os operadores da linguagem Python.

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

Fonte: Elaborado pelo autor.

7.5 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 Condicional

Em muitos algoritmos, precisamos tomar decisões para executar tarefas diferentes com base em certas condições. Por exemplo, um programa de e-mail pode liberar o acesso ou bloqueá-lo dependendo das informações de *login* inseridas pelo usuário. Para implementar essas decisões, usamos estruturas de controle chamadas instruções condicionais. Essas estruturas permitem que o programa execute diferentes partes do código dependendo das condições específicas. 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.

8.1 ESTRUTURAS CONDICIONAIS

Estruturas condicionais, também conhecidas como estruturas de seleção, são fundamentais para que um programa escolha entre diferentes blocos de código dependendo das condições avaliadas. Essas condições são avaliadas para determinar se são verdadeiras ou falsas. Uma condição é sempre uma expressão composta por literais booleanos (*bool*), operadores relacionais e/ou operadores lógicos.

Em Python, as principais estruturas condicionais são:

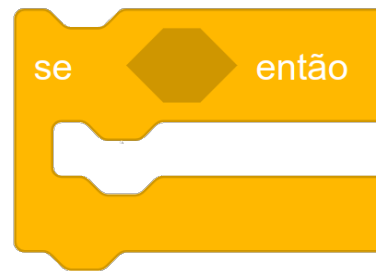
- `if`
- `if-else`
- `if-elif-else`

8.1.1 Comando `if`

O comando `if` faz com que um bloco de código seja executado 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.

Alerta: 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 quer dizer 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.

8.1.2 Comandos `if-else`

Quando você precisa lidar com duas possibilidades, uma verdadeira e outra falsa, utiliza-se os comandos `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 concional em que a condicao é verdadeira
3 else:
4     #Bloco concional em que a condicao é 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 no Scratch é ilustrada na Figura 15.

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



Fonte: Elaborada pelo autor.

8.1.3 Comandos `if-elif-else`

Quando há várias condições para verificar, usamos os comandos `if-elif-else`. Esses comandos são úteis quando você tem várias possibilidades a considerar e quer 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.

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 a 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:

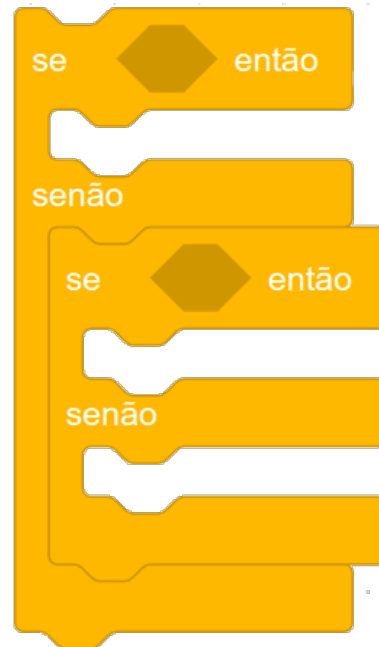
```

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.

8.2 EXEMPLOS COMENTADOS

Exemplo: Maior de idade?**Enunciado:**

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?**Enunciado:**

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?

Enunciado:

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.").

8.3 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 para:

I. $a = 10, b = 5$ e $c = 8$?

II. $a = 3, b = 7$ e $c = 9$?

III. $a = 6, b = 6$ 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, 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.

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.

9.1 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.

9.1.1 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.

9.1.2 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.

Command Line Interface

CLI, ou Interface de Linha de Comando (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.

Alerta: 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 (`/`).

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 16.