

UM GUIA INFINITY PARA INICIANTES CONCEITOS GERAIS DE LÓGICA DA PROGRAMAÇÃO E PYTHON

IN INFINITY SCHOOL

Infinity School

Conceitos gerais de lógica da programação e Python
um guia Infinity para iniciantes

Reality Editora
2024

Sumário

Apresentação

03

Introdução

04

Capítulo 1

Bem-vindo ao mundo da programação

06

Capítulo 2

Próximos passos na terra da programação

20

Capítulo 3

Seus programas ainda mais espertos

36

Capítulo final

Até logo, programador(a)

54

Apresentação

Bem-vindo ao emocionante universo da programação! Seja você um curioso que deseja desvendar os segredos por trás dos aplicativos que usa todos os dias, seja um aspirante a desenvolvedor em busca de uma car-

reira repleta de oportunidades, ***Conceitos gerais de lógica da programação e Python: um guia Infinity para iniciantes*** é o seu ponto de partida.

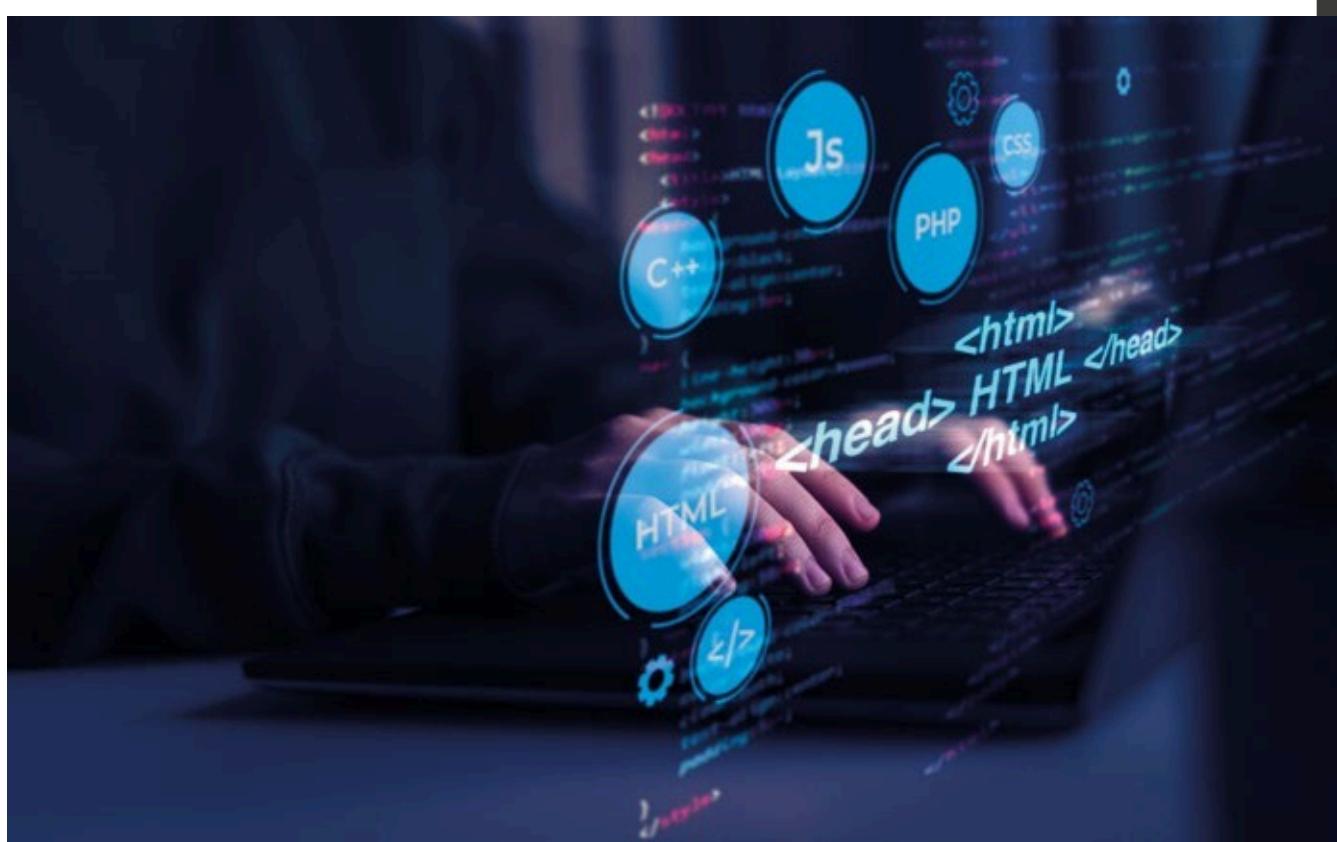
Não importa de onde você venha ou qual seja o seu nível de experiência,

este livro está aqui para guiá-lo nesse emocionante percurso, apresentando a você conceitos fundamentais para a programação. Neste guia

para iniciantes, você será convidado a mergulhar em um mundo de possibilidades, em que você aprenderá as bases da lógica de programação de forma simples e acessível, usando a linguagem Python como sua aliada.

***Prepare-se para uma jornada
de aprendizado que nunca termina,
em que sua criatividade é o único limite.***

I



Introdução

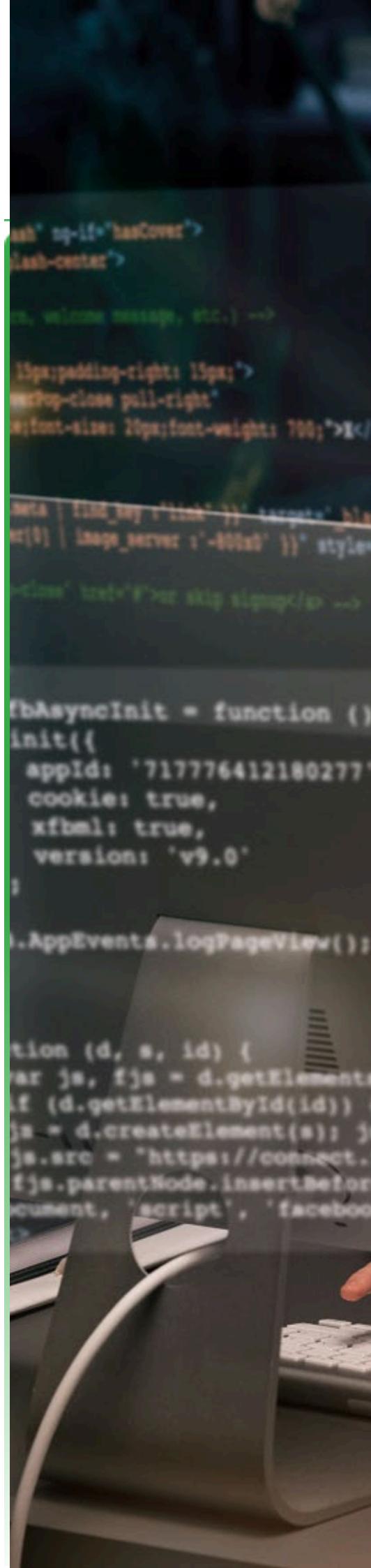
A programação é como a linguagem secreta do século XXI, permitindo que computadores e dispositivos eletrônicos realizem tarefas complexas para tornar nossas vidas mais fáceis. Você já deve ter interagido com a programação sem perceber. Por exemplo, quando você usa um aplicativo de entrega de comida para pedir seu prato favorito ou assiste a um vídeo no YouTube, a magia por trás dessas ações é o resultado de códigos escritos por programadores. A programação é uma ferramenta poderosa que ajuda a solucionar problemas de formas incríveis.

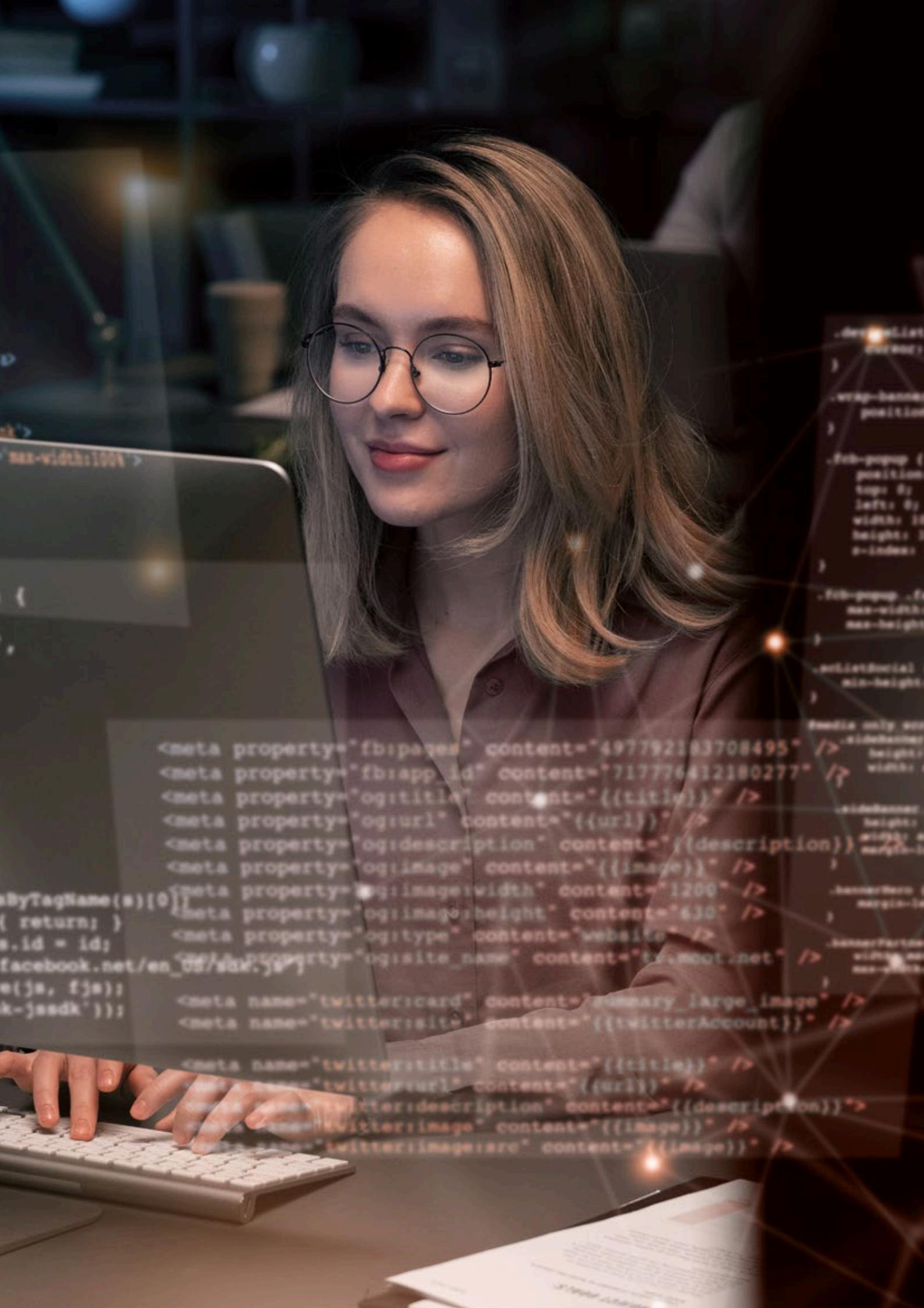
Imagine que você está planejando uma viagem de carro e quer saber a melhor rota para chegar ao seu destino. Isso é um problema de otimização, e programadores criaram **algoritmos** que usam dados de tráfego em tempo real para calcular a rota mais eficiente, economizando tempo e combustível. Ou considere a medicina: os médicos usam algoritmos de aprendizado de máquina para diagnosticar doenças com maior precisão, baseando-se em dados de pacientes de todo o mundo. A programação também é fundamental na pesquisa espacial, ajudando a controlar satélites e explorar planetas distantes. Hoje em dia, a

programação está em toda parte e desempenha um papel vital em quase todos os setores. Da inteligência artificial que impulsiona assistentes virtuais, como a Siri e a Alexa, à automação de processos em fábricas, a capacidade de escrever código abre portas para soluções

inovadoras. Neste livro, você aprenderá as habilidades essenciais da lógica de programação usando a linguagem Python, preparando-o para participar desse emocionante mundo de resolução de problemas. Com paciência e

prática, você se tornará um solucionador de problemas digital, capaz de criar aplicativos, automatizar tarefas e contribuir para o nosso mundo cada vez mais digital. Portanto, prepare-se para uma jornada empolgante e comece a desvendar os segredos por trás da programação!





```
<meta property="fb:pages" content="497792183708495" />
<meta property="fb:app_id" content="717776412180277" />
<meta property="og:title" content="{{title}}"/>
<meta property="og:url" content="{{url}}"/>
<meta property="og:description" content="{{description}}"/>
<meta property="og:image" content="{{image}}"/>
<meta property="og:image:width" content="1200"/>
<meta property="og:image:height" content="630"/>
<meta property="og:type" content="website"/>
<meta property="og:site_name" content="t3moot.net"/>
<meta name="twitter:card" content="summary_large_image"/>
<meta name="twitter:site" content="{{twitterAccount}}"/>
<meta name="twitter:title" content="{{title}}"/>
<meta name="twitter:url" content="{{url}}"/>
<meta name="twitter:description" content="{{description}}"/>
<meta name="twitter:image" content="{{image}}"/>
<meta name="twitter:image:src" content="{{image}}"/>
```

Capítulo 1

Bem-vindo ao mundo da programação

1.1 Compreendendo os algoritmos

Algoritmos são conjuntos de instruções sequenciais e bem definidas que descrevem um **processo** ou uma série de **ações** a serem executadas para realizar uma tarefa ou **resolver um problema específico**. Eles são uma parte fundamental da ciência da computação e da programação. É a especificação, o passo a passo de uma sequência ordenada, finita e inequívoca de instruções que devem ser seguidas na resolução de um determinado problema.

Observe que, apesar do nome complicado, os algoritmos estão mais presentes do que tudo em nosso cotidiano. Assim, quando vamos atravessar a rua, por exemplo, nosso cérebro automaticamente executa um algoritmo: *há carros vindo pela via? Se sim, devo esperar que os carros passem para, posteriormente, atravessar a rua. Se não, apenas atravesso a rua.*



Você se lembra da velha charada: *por que a galinha cruzou a rua?* Imagino que, se você conhece essa, também deva conhecer a resposta: *para chegar ao outro lado*. Ou seja, chegar ao outro lado da rua era o **objetivo** da galinha. Por essa razão, o algoritmo para cruzar a rua teve um fim, um objetivo que pode ser alcançado em função do passo a passo executado.

Dessa maneira, sempre que construímos um plano mental para realizar uma determinada tarefa, aplicamos conceitos algorítmicos, diária e constantemente, sempre que temos que realizar um conjunto de etapas até atingir

o objetivo desejado, seja para atravessar a rua, seja para fazer uma compra no mercadinho da esquina. O processo de tomada de decisões e de escolhas para alcançar um determinado objetivo são algoritmos.

Veja alguns exemplos de algoritmos na vida real:

Receita culinária | Instruções de montagem | Navegação | Buscas na internet

Uma receita de bolo, de arroz de forno, macarrão etc.

Manual de instruções, guia de uso de produto etc.

GPS, mapas etc.

Buscas no Google por imagens, textos, produtos etc.

A partir do infográfico anterior, podemos citar inúmeros exemplos, pois a nossa rotina é um algoritmo. Para fazermos um bolo de caneca, por exemplo, precisamos seguir um algoritmo, que neste caso é a **receita**. A receita servirá para orientar o processo de produção do bolo de caneca.

Em outras palavras, o *algoritmo é o nosso programa, e a finalidade do programa é fazer o bolo*. Ao final da sua execução, teremos um bolo de caneca para saborear, caso o algoritmo tenha sido executado corretamente. *Mas e se o houver erros durante a execução do algoritmo?* Ou ainda: *o que acontece caso esqueçamos de alguma etapa ou troquemos um ingrediente por outros? O resultado será o mesmo? Como ficaria o nosso bolo de caneca?* Essas e outras perguntas são essenciais no pro-

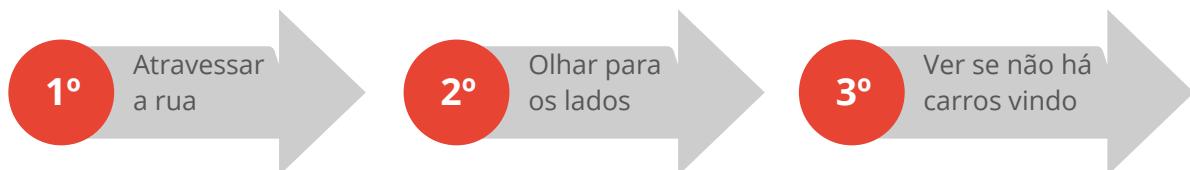
cesso de construção de algoritmos, pois, **mas**. Portanto, quando criamos um algoritmo, ao desenvolver um programa computacional, apontamos uma das várias formas de resolver precisarmos deixar todas as instruções para o problema ou criar a finalidade para a execução do programa bem definidas. Caso gramática. Assim, sempre teremos alternativas contrário, podemos ter resultados possíveis para a resolução do problema: algumas soluções mais adequadas, outras menos; algumas mais eficientes, outras nem tanto...

Em síntese, um algoritmo é uma série de passos lógicos e finitos que permitem resolver ou solucionar um problema. Aprender a criar

um algoritmo é um passo muito importante para o programador, pois estamos falando da base de conhecimento de uma linguagem de programação.



Para tanto, existem muitas maneiras de visualizar e resolver o mesmo problema, ou seja, **vários algoritmos diferentes podem ser criados para solucionar o mesmo problema**.



O exemplo acima é um algoritmo correto para atravessar a rua? Certamente não! Por essa razão, é muito importante que a sequência de instruções esteja bem definida para que o algoritmo funcione, ou seja, para que se chegue ao objetivo. Assim, para que um computador realize ou execute uma tarefa, são necessárias instruções bem detalhadas, em uma linguagem que a máquina consiga compreender: uma **linguagem de programação**.

Na seção a seguir, aprenderemos a pensar esses algoritmos a partir de alguns exemplos do dia a dia. Você também encontrará exercícios práticos e comentados para praticar a modelagem de um algoritmo, ou seja, como organizar os passos de uma tarefa para que as instruções fiquem claras, bem definidas e objetivas, garantindo, assim, a funcionalidade do algoritmo.



1.2 Aprendendo a modelar um algoritmo

Antes de seguirmos, precisamos ter em mente que um bom algoritmo se estrutura em cinco pilares fundamentais: **sequencialidade, definição clara, finitude, eficiência e objetividade**. Veja o esquema a seguir e entenda melhor cada um desses elementos.



RECEITA DE BOLO

Passo 1: preaqueça o forno a uma temperatura específica.

Passo 2: misture a farinha, o açúcar, os ovos, o leite e os outros ingredientes em uma tigela.

Passo 3: despeje a mistura em uma forma untada.

Passo 4: asse no forno preaquecido por um tempo determinado.

Passo 5: verifique a cozedura com um palito. Se sair limpo, o bolo está pronto.



INSTRUÇÕES DE GPS

Passo 1: determine a localização atual do veículo.

Passo 2: defina o destino desejado.

Passo 3: calcule a rota mais curta com base no tráfego em tempo real.

Passo 4: instrua o motorista a virar à esquerda, à direita ou seguir em frente em cada interseção até chegar ao destino.



MÁQUINA DE CAFÉ

Passo 1: insira uma cápsula de café na máquina.

Passo 2: encha o reservatório de água até a marca desejada.

Passo 3: pressione o botão de ligar.

Passo 4: a máquina moerá o café, aquecerá a água e preparará a bebida.

Passo 5: sirva o café quando estiver pronto.

Observe que todos os exemplos anteriores possuem uma *sequência definida (sequentialidade)* com *instruções claras (bem definidas)*, são *finitos*, quer dizer, têm um fim (*finitude*), são *eficientes* e, finalmente, conduzem a um *objetivo para a resolução de um problema (objetividade)*. Temos, portanto, um algoritmo completo que dá conta de resolver um problema. Evidentemente, esse algoritmo não é único, tampouco perfeito, mas é uma solução encontrada para dado problema.

Agora que você já conhece algumas coisas sobre os algoritmos, na próxima seção deste livro você irá mais além: aprenderá como utilizar seus algoritmos para acessar e manipular variáveis na memória do computador.

1.3 Variáveis e a memória do computador

Imagine que você esteja em uma festa de aniversário. Ao final dessa festa, você decide pegar um copo descartável para guardar alguns salgadinhos, docinhos e um grande pedaço de bolo para salvar o café da manhã do dia seguinte. Assim, você utiliza vários copos para guardar cada um desses itens para que, no dia seguinte, possa comê-los e saboreá-los durante o café.



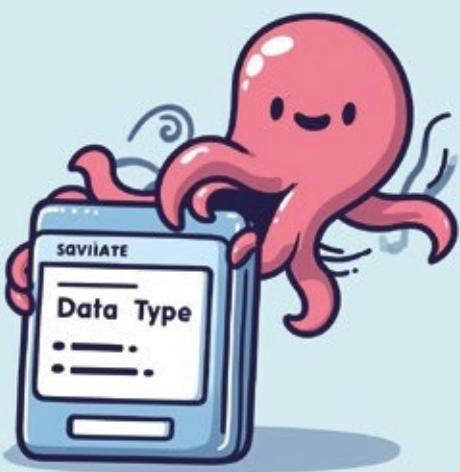
No exemplo anterior, o copo descartável foi utilizado **armazenar** alimentos: docinhos, salgadinhos, um pedaço de bolo... Para além disso, podemos utilizar nosso copo para colocar outras coisas dentro: refrigerante, água, açúcar, farinha etc. Nossa copo é bem versátil e pode guardar coisas de vários tipos sólidos, gasosos e muitas outras.

Mas você deve estar se perguntando: *qual a relação disso com programação?* Mais ainda, *o que um copo tem a ver com variáveis?* Bom,

uma **variável** é como um copo descartável, pois ela pode guardar várias coisas. No entanto, diferentemente do copo, ao

invés

de guardar salgadinhos, docinhos e tantas



outras coisas, a variável pode guardar diferentes **tipos de dados** (tipos de dados são como os salgadinhos no copo, mas com algumas diferenças que você aprenderá logo mais).

Na imagem ao lado, aparecem alguns tipos de dados armazenados em diferentes variáveis. Essas variáveis, por sua vez, encontram-se localizadas na **memória do computador**. Assim, guardamos os dados na memória do computador para que possamos utilizá-los depois.

Dessa forma, ao colocarmos uma variável na memória do computador, podemos utilizar um dado de diversas formas: realizar operações matemáticas, realizar operações relacionais, guardar o nome de um usuário, guardar um conjunto de informações e outros tipos de dados.

A memória do computador é como a nossa **memória** mesmo: tem a capacidade de guardar dados (ou informações) por um determinado tempo. A partir de então, se precisarmos desse dado, basta acessarmos a memória, seja do computador, seja a nossa memória.

Em resumo, as variáveis e a memória do computador estão totalmente ligadas à programação.



O conhecimento sobre como as variáveis são armazenadas e manipuladas na memória é essencial para o desenvolvimento de software eficiente e a resolução de problemas de forma eficaz. Programadores precisam equilibrar a alocação de recursos e o desempenho ao lidar com variáveis, criando **algoritmos** que funcionem de maneira otimizada e atendam às necessidades das aplicações.

1.4 Variáveis e tipos de dados: acessando a memória do computador

Agora que você já sabe sobre as variáveis, é importante aprender um pouco sobre os **tipos de dados** que essas variáveis podem armazenar (guardar). Isso porque o computador é uma máquina que trabalha com dados. Por isso, é importante aprender sobre seus tipos e suas características.



Conforme ilustra a imagem ao lado, um computador **recebe, processa e devolve dados** de alguma maneira. Por essa razão, criamos variáveis na memória do computador para armazenar esses dados para que eles sejam processados e devolvidos ao usuário final. Em outras palavras, esses dados são utilizados para **resolver problemas diversos**: cálculos matemáticos, contagens, comparações etc. Uma vez que o problema foi resolvido (**processado**), podemos fazer alguma coisa com esse **produto final**. Como dito



anteriormente, os dados são fundamentais para que o computador resolva um problema. Por essa razão, assim como os problemas são diversos, os tipos de dados também são. Assim, em Python, temos **dados do tipo texto** (palavras, letras etc.), **dados do tipo numérico** (números inteiros e números decimais) e **dados do tipo lógico** (verdadeiro ou falso):

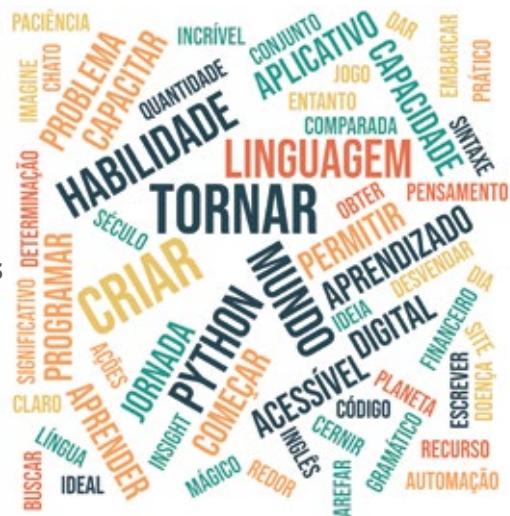
Tipos de dados primitivos

Decimais

Textos (strings)	Inteiros (int)	Decimais (float)	Lógicos (bool)
---------------------	-------------------	---------------------	-------------------

A partir da imagem anterior, nota-se que os dados do tipo texto são comumente chamados de strings. Um dado **string** é colocado dentro de **aspas**, como forma de mostrar ao computador que estamos lidando com um **texto**. Por essa razão, tudo o que estiver entre aspas será interpretado como texto.

Os **dados do tipo numérico**, que podem ser **inteiros** ou **reais**, são representados por números diversos. Assim, quando informamos ao computador os números 1, 2, 3, por exemplo, estamos lidando com dados do tipo inteiro (**int**). Nessa mesma direção, ao declararmos variáveis que guardem números como 3.14, 4.2 ou 15.3, temos definidos dados do tipo real (**float**):



Exemplos de números do tipo inteiro (int)

11 7 14 22 15 76 0 1

Exemplos de números do tipo real/decimal (float)

0.01 0.47 1.2 3.14 8.76 9.11

Ainda sobre os números reais (**float**), em programação utilizamos o ponto (.) ao invés da vírgula (,) para indicar as casas decimais.

Finalmente, os **dados do tipo lógico**, também conhecidos como booleans (**bool**), são utilizados para informar dois estados: verdadeiro (**True**) ou falso (**False**). Dessa forma, é possível utilizá-los para indicar se um sistema está funcionando, se a luz do quarto está acesa, se o veículo tem gasolina, se o usuário está logado no sistema etc. Apenas com as palavras reservadas **True** e **False**, podemos indicar uma série de situações ao computador:

TRUE | FALSE

Bom, você já aprendeu algumas coisas até aqui, não é mesmo? Mas ainda falta uma coisa: transformar todos esses conhecimentos em código Python! Para isso, você encontrará a seguir uma série de atividades comentadas para desenvolver. Utilizaremos o **Google Colab** para fazer os códigos, e você encontrará nos anexos deste livro o tutorial para utilização da ferramenta.

1.5 Seus primeiros códigos: tutorial

Para começar, criaremos um notebook no Google Colab para exibir um nome. Para isso, utilizaremos um comando que exibe mensagens de texto, números e muitas outras coisas na tela: o **print()**. Dentro dos parênteses do comando, colocaremos um nome da seguinte maneira: “**Kauê**”. Ficará assim:

```
[ ] print("Kauê")
```

Note que é importante colocar o nome informado dentro das aspas, pois estamos trabalhando com um dado do tipo texto. Ou seja, sempre que quisermos exibir textos a partir do comando **print()**, precisamos colocar o texto dentro das aspas. Caso contrário, teremos um erro, e a mensagem não será exibida.

Ao passar o mouse por cima do bloco de código que acabamos de criar, aparecerá um botão de *play* para que executemos o código Python:



```
print("Kauê")
```

play



```
print("Kauê")
```

Kauê

Em seu primeiro código Python, você exibiu um nome por meio do comando **print()**. Assim, sempre que você ler em algum lugar ou ouvir alguém dizer *faça um programa que exiba na tela ou faça um programa para printar uma mensagem*, basta você lembrar do comando **print()**.

Mas e se quiséssemos exibir uma mensagem na tela? É muito simples: basta substituir o nome que está dentro as aspas pela mensagem que você deseja exibir.



```
print("Olá, Mundo! Meu nome é Kauê.")
```

Olá, Mundo! Meu nome é Kauê.

Nós ainda podemos utilizar o mesmo comando para exibir números inteiros (1, 2, 3), números decimais/reais (1.57, 3.14, 7.71) ou até mesmo um valor lógico (**True** ou **False**). Veja os exemplos a seguir:

Exibindo um valor numérico do tipo inteiro

A screenshot of a code editor interface. On the left is a play button icon. To its right is a dark grey bar containing the green text `print(2023)`. Below this is a light grey bar containing the white text `2023`.

Exibindo um valor numérico do tipo float (real/decimal)

A screenshot of a code editor interface. On the left is a play button icon. To its right is a dark grey bar containing the green text `print(3.14)`. Below this is a light grey bar containing the white text `3.14`.

Exibindo um valor do tipo booleano (lógico)

A screenshot of a code editor interface. On the left is a play button icon. To its right is a dark grey bar containing the blue text `print(True)`. Below this is a light grey bar containing the white text `True`.

Observe que em nenhum dos exemplos anteriores foi necessário usar as aspas, conforme utilizado no exemplo do nome ou da frase, pois não é necessário utilizar as aspas para exibir valores numéricos e valores lógicos – apenas para nossos textos. No caso dos valores lógicos, por exemplo, o computador já sabe que `True` e `False` são valores padrões, ou seja, eles são entendidos sem a necessidade de informar as aspas. No caso dos valores numéricos, por sua vez, podemos combiná-los uns aos outros para resolver operações matemáticas de adição, subtração, multiplicação e divisão:

Operação matemática de adição

A screenshot of a code editor interface. On the left is a play button icon. To its right is a dark grey bar containing the green text `print(12+2)`. Below this is a light grey bar containing the white text `14`.

Operação matemática de subtração

A screenshot of a code editor interface. On the left is a play button icon. To its right is a dark grey bar containing the green text `print(8-6)`. Below this is a light grey bar containing the white text `2`.

Operação matemática de multiplicação



```
print(2*2)
```

4

Operação matemática de divisão



```
print(12/3)
```

4

Note que, para realizarmos as quatro operações matemáticas fundamentais, utilizamos os mesmos sinais da matemática, conforme a tabela a seguir:

Tabela de operadores matemáticos em Python

Sinal	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Além disso, nós também podemos exibir um número que foi guardado em uma variável. Afinal, utilizamos as variáveis para guardar valores, não é mesmo?



```
numero = 10
print(numero)
```

10

No código acima, definimos uma variável “**numero**” (sem acento). Depois, utilizamos o sinal de igualdade (=) para atribuir a essa variável um valor numérico – **10**, no caso. Na sequência, ao invés de colocarmos o número 10 dentro do comando **print()**, colocamos a própria variável “**numero**”. Finalmente, ao clicarmos no botão *play*, é exibido o número guardado na variável. Assim sendo, não é necessário utilizar aspas quando formos exibir valores que foram guardados (atribuídos em uma variável).

Veja outros exemplos com variáveis e a exibição de valores com o comando **print()**:

Declarando uma variável para armazenar um valor do tipo texto



```
nome = "Kauê"  
print(nome)
```

Kauê

Declarando uma variável para armazenar um valor do tipo inteiro



```
idade = 12  
print(idade)
```

12

Declarando uma variável para armazenar um valor do tipo float (real/decimal)



```
altura = 1.87  
print(altura)
```

1.87

Declarando uma variável para armazenar um valor do tipo booleano (lógico)



```
solteiro = True  
print(solteiro)
```

True

Agora que você já sabe como utilizar o comando **print()** e as variáveis na prática, é sua vez de brilhar! A seguir, você encontrará uma lista com alguns exercícios. Tente desenvolvê-los com tranquilidade. Ao final do livro, nos apêndices, você encontrará as respostas para os exercícios. Vamos lá?

Mãos ao código | Hora de praticar

1 • Exibindo seu nome na tela:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para armazenar seu **nome**. Na sequência, utilize o comando **print()** para exibir essa informação.

2 • Exibindo uma frase na tela:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para armazenar **uma frase** e outra para armazenar um **autor**. Na primeira variável, atribua a seguinte frase: “*A melhor maneira de prever o futuro é criá-lo*”. Na variável **autor**, por sua vez, atribua o seguinte valor: “Peter Drucker”. Na sequência, utilize dois comandos **print()** para exibir cada um desses valores guardados nas variáveis.

Dica: após fazer o primeiro **print()**, salte uma linha para fazer o segundo.

3 • Resolvendo operações matemáticas:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para armazenar um **número 1** e outra para armazenar um **número 2**. Atribua às variáveis valores da sua preferência. Na sequência, utilize dois comandos **print()** para exibir a soma e a subtração desses números.

Dica: após fazer o primeiro **print()**, salte uma linha para fazer o segundo.

4 • Exibindo valores no mesmo print():

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para armazenar um **número 1** e outra para armazenar um **número 2**. Atribua às variáveis valores da sua preferência. Na sequência, utilize um comando **print()** para exibir os dois valores de uma vez, um ao lado do outro.

Dica: após informar o nome da primeira variável, coloque uma vírgula e escreva o nome da segunda variável dentro dos parênteses do comando **print()**.

5 • Exibindo texto e número no mesmo print():

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para armazenar um **nome** e outra para armazenar uma **idade**. Atribua às variáveis valores da sua preferência. Na sequência, utilize um comando **print()** para exibir os dois valores de uma vez, um ao lado do outro.

Dica: após informar o nome da primeira variável, coloque uma vírgula e escreva o nome da segunda variável dentro dos parênteses do comando **print()**.

Infinity Quiz | Teste seus conhecimentos

1 • Qual dos seguintes tipos de dados é usado para representar números inteiros em Python?

- a) String
- b) Float
- c) Int
- d) Boolean

2 • Qual é o tipo de dado apropriado para armazenar números reais/decimais em Python?

- a) String
- b) Float
- c) Int
- d) Boolean

3 • Qual é o tipo de dado apropriado para armazenar valores lógicos em Python?

- a) String
- b) Float
- c) Int
- d) Boolean

4 • Para declarar uma variável de string em Python, você pode usar aspas simples (' ') ou aspas duplas (" "). Qual é a diferença?

- a) Não há diferença, você pode usar qualquer uma.
- b) Aspas simples são usadas para strings curtas, e aspas duplas para strings longas.
- c) Aspas simples não podem ser usadas para strings.
- d) As aspas simples não podem ser usadas em Python.

5 • Qual é a representação correta de um valor booleano verdadeiro em Python?

- a) true
- b) 1
- c) TRUE
- d) True

Respostas Infinity Quiz:

1 – C; 2 – B; 3 – D; 4 – A; 5 – D.

Resumindo tudo até aqui...

- Um algoritmo é um conjunto de instruções finitas que visam algum objetivo.
- Para que os **algoritmos** funcionem bem, podemos criar variáveis para guardar dados diversos.
- Os dados podem ser do tipo texto (strings), do tipo numérico – inteiro (int) ou real/decimal (float) – ou lógicos (**True ou False**).
- Em Python, temos um comando para exibir mensagens na tela: o **print()**.
- Podemos fazer operações matemáticas ou exibir números de tipos variados dentro dos parênteses do comando **print()**.

Capítulo 2

Próximos passos na terra da programação

2.1 A estrutura *if...else*: verificação condicional simples

Imagine um usuário que acabou de criar uma conta no Instagram. Conforme o tema dessa rede social, é necessário definir um **nome de usuário** e uma **senha**, que sempre serão solicitados quando um novo login for feito. Assim, se o usuário informar corretamente o usuário e a senha, será redirecionado à sua página inicial. Caso contrário, o acesso não acontecerá.

Perceba que, no exemplo anterior, é necessário que seja feita uma verificação para que o login seja realizado. Ou seja, caso o usuário e/ou a senha informada não estejam corretos de acordo com as credenciais cadastradas, o sistema não pode permitir que o usuário realize o login.



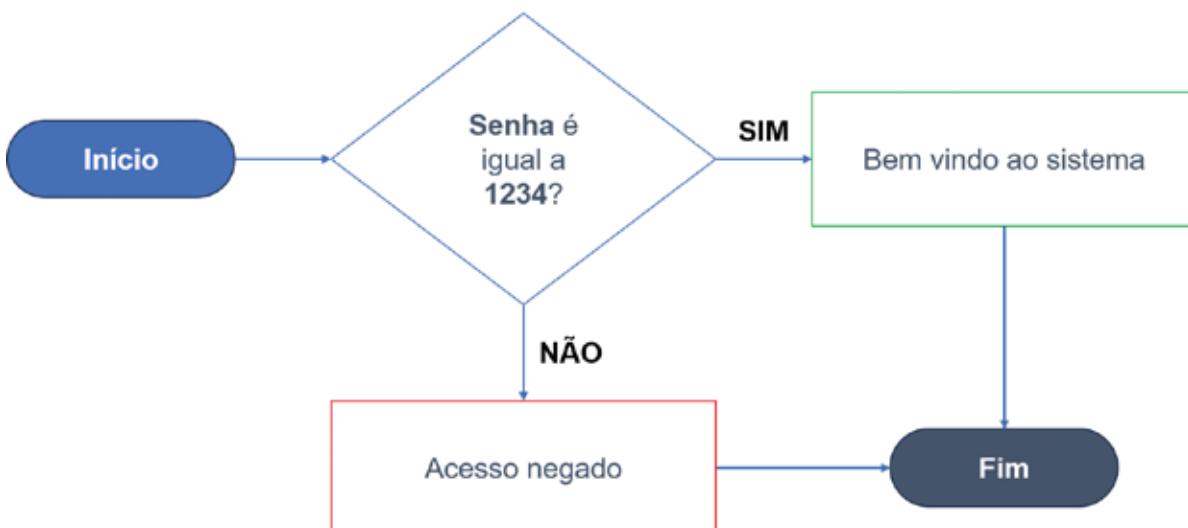
Por essa razão, para que nossos **algoritmos** fiquem ainda melhores e mais potentes, utilizamos a **estrutura condicional** para fazer verificações. Assim, ao invés de todo o código ser executado deliberadamente, o computador fará perguntas a si mesmo: *o usuário e a senha estão corretos? Se sim, posso exibir a página inicial; se não, devo exibir uma mensagem de erro.*

Verificação se idade é maior-igual a 18



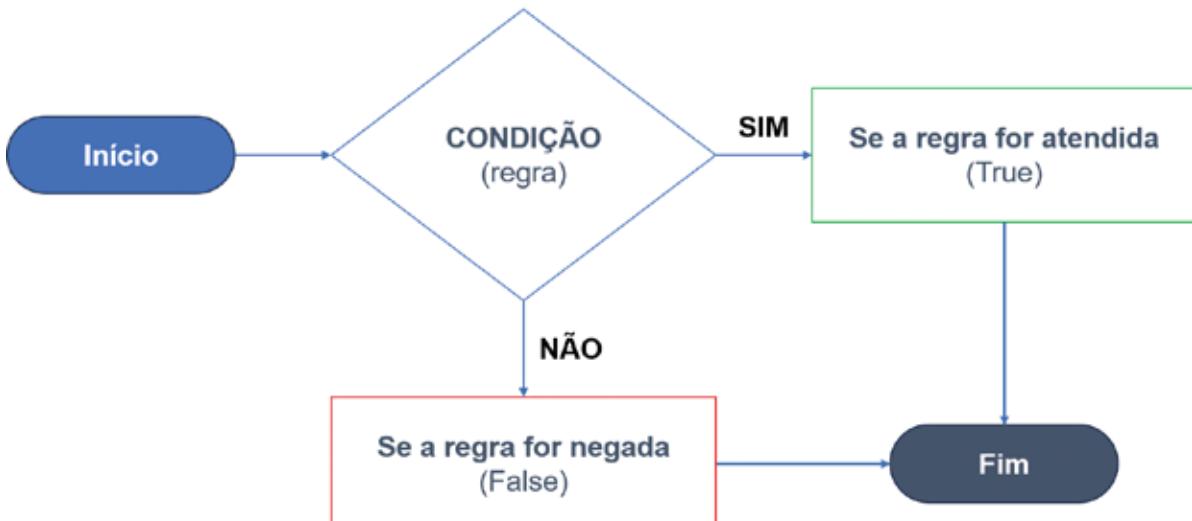
Em Python, assim como em outras linguagens de programação, utilizamos as palavras **if...else** para fazer verificações, sendo **if** o mesmo que **se**, em português; e **else** equivalente a **se não**. Com apenas essas palavras, podemos fazer a verificação de usuário e senha do exemplo anterior, bem como muitas outras verificações. Veja outro exemplo a seguir:

Verificação se a senha é igual a 1234



No exemplo anterior, utilizamos da estrutura **if...else** para fazer verificações simples. Assim como na vida real, realizamos certas ações apenas se algo for verdadeiro ou se alguma coisa acontecer. Em outras palavras, a estrutura **if...else** verifica uma **regra** e executa um bloco de código se essa regra for **verdadeira**. Mas e se a regra não for verdadeira? Isso mesmo, se a regra não é verdadeira, ela é **falsa**, logo temos uma exceção à regra:

Esquema if...else



Agora que você já conhece a estrutura condicional, que tal colocar em prática o que você aprendeu até aqui? Para isso, abra seu Google Colab e tente desenvolver os exercícios a seguir. Lembre-se de criar uma célula para cada exercício e executá-las para testar seus códigos. Ah! Para te dar uma forcinha, a seguir você encontrará alguns exemplos práticos comentados.

2.2 Exemplos práticos comentados

No exemplo a seguir, faremos a verificação de uma idade. Caso o valor atribuído à variável `idade` seja maior-igual a 18, será exibida a mensagem: "Você é maior de idade". Caso contrário: "Você não é maior de idade". Veja o exemplo a seguir:

▶

```

idade = 17

if idade >= 18:
    print("Você é maior de idade")
else:
    print("Você não é maior de idade")
  
```

Você não é maior de idade

Note que, além de utilizarmos a **estrutura condicional**, foi necessário usar um **operador relacional** para verificar a relação existente entre o valor que a variável `idade` está guardando e o número 18. Assim, utilizamos o sinal `>` em referência a **maior**, enquanto o sinal `=` faz referência à igualdade. Ou seja, para que a regra testada seja verdadeira, é necessário que o valor seja maior do que 18, mas também pode ser 18: 18, 19, 20 ... 40, 50, 60 e assim por diante.

Além da relação **maior-igual**, também podemos verificar se um valor é **menor-igual** do que outro, se é apenas maior ou apenas menor do que outro, se é di-

ferente ou se é igual. A seguir, veja a tabela de operadores relacionais e aprenda um pouco mais sobre essas relações:

Tabela de operadores relacionais em Python

Sinal	Operação
>	Maior
< ==	Menor
!= >=	Igual
<=	Diferente
	Maior-igual
	Menor-igual

Entre os **operadores relacionais**, é importante termos atenção especial ao operador de igualdade (==) e ao operador de diferença (!=).

No caso do operador de igualdade, note que utilizamos dois iguais para representá-lo, pois em programação, quando utilizamos apenas um igual, estamos atribuindo um valor a uma variável, não é mesmo? Assim, é importante que sejam utilizados dois iguais para que o computador entenda que estamos realizando uma operação de comparação, e não de atribuição:



```
palavra = "amor"

if palavra == "roma":
    print("As palavras são iguais")
else:
    print("As palavras são diferentes")
```

As palavras são diferentes

Quanto ao operador de diferença, por sua vez, utilizamos o ponto de exclamação (!) combinado ao igual (=) para realizar a representação de diferença. Assim, com esse sinal, podemos verificar se um número *x* é diferente de um número *y*, se uma palavra é diferente da outra etc.



```
palavra = "amor"

if palavra != "roma":
    print("As palavras são diferentes")
else:
    print("As palavras são iguais")
```

As palavras são diferentes

Veja outros exemplos:

Verificando um número da sorte



```
numero = 33

if numero == 33:
    print("O número está certo!!!")
else:
    print("Não foi dessa vez...")
```

O número está certo!!!

Verificando a temperatura



```
temperatura = 21.3

if temperatura <= 20.1:
    print("Hoje fará frio...")
else:
    print("A temperatura está OK")
```

A temperatura está OK

Verificando a idade para votação



```
idade = 72

if idade >= 70:
    print("Você não precisa votar")
else:
    print("Você precisa votar")
```

Você não precisa votar

Ufa, aprendemos muitas coisas sobre **estruturas condicionais**, não é mesmo? Agora, para colocar tudo em prática, você encontrará mais alguns exercícios para treinar. Tente desenvolvê-los com tranquilidade. Ao final do livro, nos apêndices, você encontrará as respostas para os exercícios. Vamos lá?

Mãos ao código | Hora de praticar

6 • Verificando se um número é maior do que cinco:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável para guardar um número inteiro. Na sequência, verifique se o número guardado na variável é maior do que 5. Caso verdadeiro, exiba a mensagem: "O número é maior do que 5". Se não, informe: "O número não é maior do que 5".

7 • Verificando qual o maior de dois números:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare duas variáveis e atribua a cada uma delas valores inteiros quaisquer. Na sequência, verifique se o número guardado na primeira variável é **maior** do que o número guardado na segunda variável. Caso verdadeiro, exiba a mensagem: "O primeiro número é **maior** do que o segundo". Se não, informe: "O primeiro número é **menor** do que o segundo".

8 • Verificando qual o menor de dois números:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare duas variáveis e atribua a cada uma delas valores inteiros quaisquer. Na sequência, verifique se o número guardado na primeira variável é **menor** do que o número guardado na segunda variável. Caso verdadeiro, exiba a mensagem: "O primeiro número é **menor** do que o segundo". Se não, informe: "O primeiro número é **maior** do que o segundo".

9 • Verificando senha de acesso ao sistema:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável e atribua a ela a senha: "kaue1234". Na sequência, verifique se a senha guardada na variável é igual (==) a "admin1234". Caso verdadeiro, exiba a mensagem: "Olá, administrador. Bem-vindo ao sistema". Se não, informe: "Você não é um usuário administrador".

10 • Verificando se um número é 3, 6 ou 9:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável e atribua a ela um número qualquer, de 1 a 9. Na sequência, verifique se o número guardado na variável é 3, 6 ou 9. Caso verdadeiro, exiba a mensagem: "O número é múltiplo de 3". Se não, o programa deve exibir a mensagem: "O número não é múltiplo de 3".

2.3 A estrutura if...elif...else: quando há mais de uma regra

Norberto é dono de um mercadinho muito famoso em sua cidade. Como não há concorrência, Norberto vende para praticamente toda a cidade.

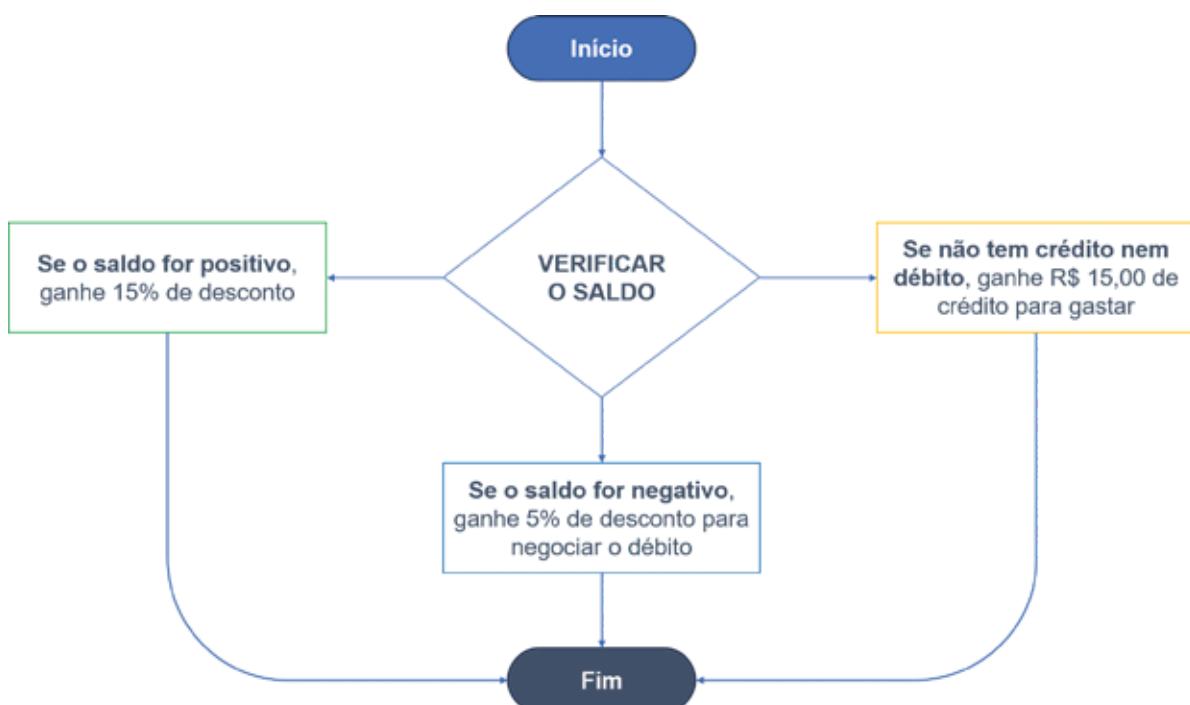
Recentemente, ao fazer o balanço das contas, Norberto identificou três perfis de clientes do seu mercadinho: i) aqueles que têm crédito no comércio, ou seja,

saldo positivo; ii) aqueles que têm débito, saldo negativo; e, finalmente, iii) aqueles que não têm créditos nem débitos.

Pensando nisso, Norberto decidiu realizar uma série de ações para que aqueles que têm crédito ganhem 15% de desconto em todas as compras acima de R\$ 50,00. Aqueles que têm débitos, por sua vez, ganharão 5% de desconto para negociar os débitos até o quinto dia útil do mês seguinte. Finalmente, os que não têm débitos nem créditos ganharão um saldo de R\$ 15,00 para gastar no mercadinho, válidos até o final de semana.

Bom, o exemplo de Norberto ilustra uma situação muito comum em programação: a necessidade de fazermos condicionais ainda mais complexas, com novas regras a serem avaliadas. Ou seja, no exemplo anterior, apenas uma condição simples (**if...else**) não atenderia às ações de Norberto, pois, além de avaliar os clientes com saldo positivo e aqueles com saldo negativo, será necessário identificar aqueles com o saldo zerado, sem débitos e créditos.

Em Python, podemos dar maior robustez às nossas condicionais incluindo a instrução **elif**, que é uma mistura do **if** com o **else**. Basicamente, o **if** é a primeira regra que será verificada. No entanto, caso a regra do **if** não seja verdadeira, antes de cairmos na exceção, ou seja, no bloco **else**, o computador testará outra regra: o **elif**. Veja como ficaria o caso de Norberto:



Conforme o exemplo anterior, o **if** verificou se o cliente possui saldo positivo no mercadinho, enquanto o **elif** verificou se o cliente possui saldo negativo. Bom, se o cliente não possuir saldo positivo nem saldo negativo, o saldo só pode ser zero. Ou seja, **exceção!** E foi exatamente isso que definimos no código anterior. Fizemos as condições possíveis para o caso de Norberto. Veja o exemplo em código Python:

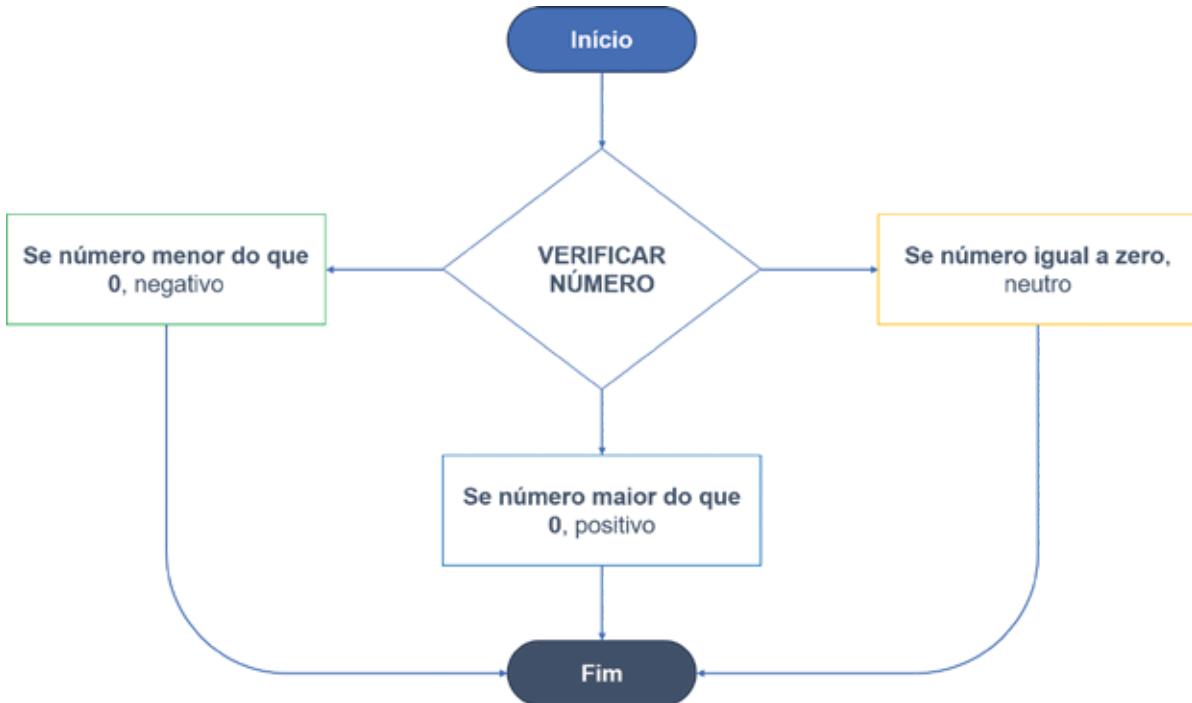
```
saldo = -17

if saldo > 0:
    print("Saldo positivo = ")
    print("Você ganhou 15% de desconto")
elif saldo < 0:
    print("Saldo negativo = ")
    print("Você ganhou 5% de desconto")
else:
    print("Seu saldo está zerado!")
    print("Você ganhou R$ 15,00 de crédito")

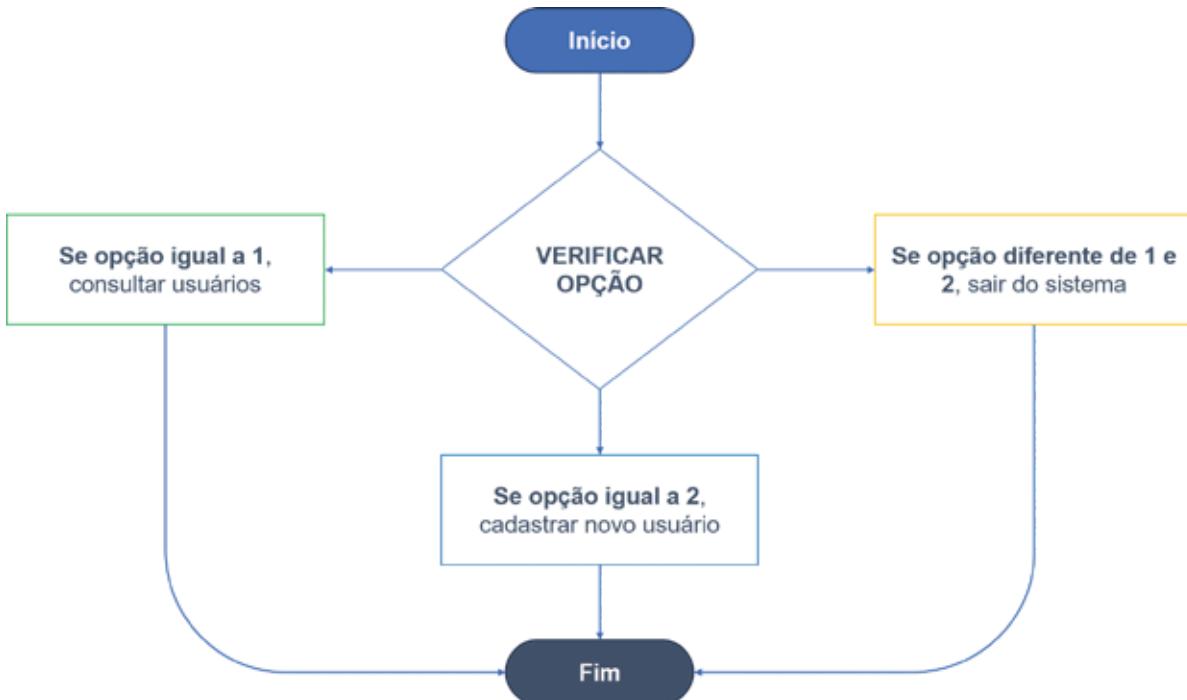
Saldo negativo =
Você ganhou 5% de desconto
```

Em outras palavras, sempre que tivermos mais do que uma regra para verificar, podemos utilizar o **elif**, seja um, sejam vários, dependendo única e exclusivamente do problema que precisamos resolver:

Fluxograma número menor que zero, maior que zero e neutro



Fluxograma menu de opções



É importante você observar que, assim como a condição **if**, o bloco **elif** também verifica uma regra. Assim, é possível fazer vários blocos de **elif** para verificar quantas regras forem necessárias até que se chegue em uma exceção (**else**). Veja o código a seguir:

▶

```

opcao = 1

if opcao == 1:
    print("Consultar usuários do sistema")
elif opcao == 2:
    print("Cadastrar usuário no sistema")
elif opcao == 3:
    print("Atualizar cadastro de usuário")
elif opcao == 4:
    print("Deletar usuário do sistema")
else:
    print("Sair do sistema")
  
```

Agora, para não perdermos a prática, vamos fazer mais alguns exercícios. Lembre-se das instruções anteriores para realização dos exercícios. Ao final do livro, nos apêndices, você encontrará as respostas.

Mãos ao código | Hora de praticar

11 • Determinando a faixa etária:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável **idade** e atribua a ela uma idade qualquer. Na sequência, com base na idade armazenada na variável, utilize o comando **print()** para informar a faixa etária:

"Criança" (0 a 12 anos)	"Adolescente" (13 a 19 anos)	"Adulto" (20 a 64 anos)	"Idoso" (65 anos ou mais)
----------------------------	---------------------------------	----------------------------	------------------------------

12 • Classificação de nota escolar:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável **nota** e atribua a ela uma nota qualquer. Na sequência, com base na nota armazenada na variável, utilize o comando **print()** para informar a situação acadêmica:

"Reprovado" (nota de 0 a 4)	"Recuperação" (nota de 5 a 6)	"Aprovado" (nota de 7 a 10)
--------------------------------	----------------------------------	--------------------------------

13 • Calculadora de IMC:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável **peso** e atribua a ela o valor **103.3**. Crie também uma variável **altura** e atribua a ela o valor **1.85**. Crie ainda uma variável **IMC** para armazenar a seguinte operação: **peso / (altura * altura)**. Na sequência, com base no resultado da operação atribuída à variável **IMC**, utilize o comando **print()** para informar a categoria do IMC:

"Abaixo do peso" (abaixo de 18,5)	"Peso normal" (a partir de 18,5)	"Sobrepeso" (a partir de 25,0)	"Obesidade" (a partir de 30,0)
--------------------------------------	-------------------------------------	-----------------------------------	-----------------------------------

14 • Decisão de compra:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável **preco**, que armazenará o preço de um produto (atribua um preço qualquer). Na sequência, utilize o comando **print()** para definir se o item está caro, barato ou com um preço razoável. Considere as informações a seguir para fazer as definições:

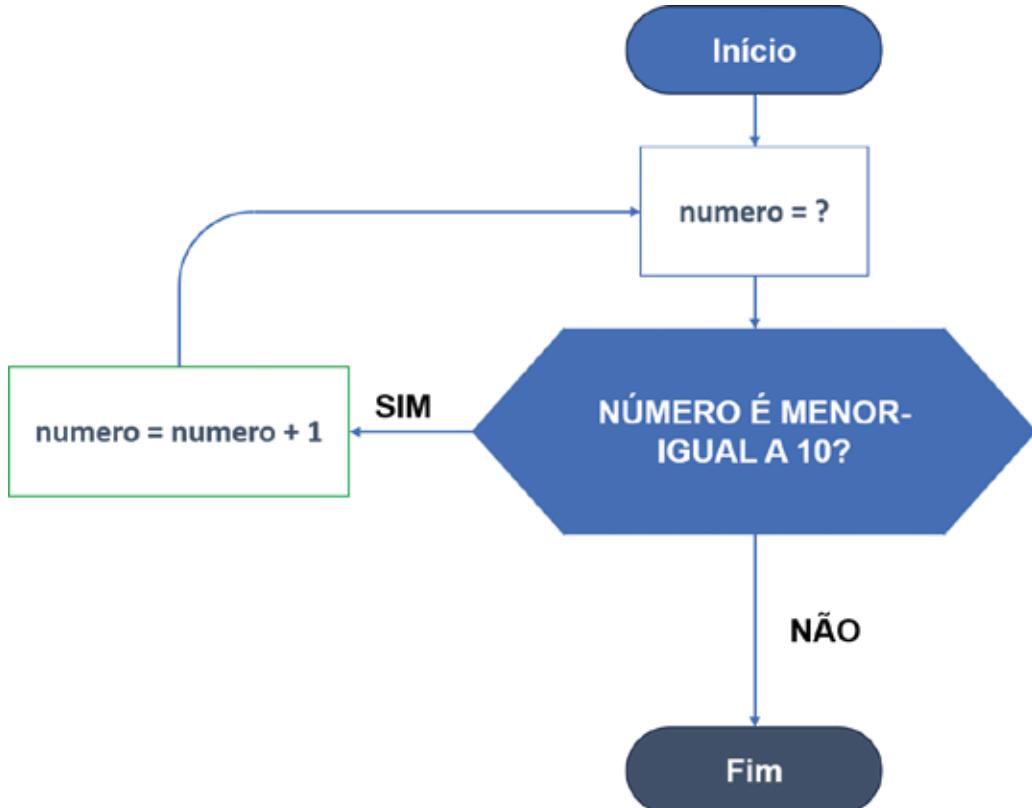
"Barato" (abaixo de R\$ 5,00)	"Razoável" (até R\$ 10,00)	"Sobrepeso" (a partir de 25,0)	"Caro" (acima de R\$ 10,00)
----------------------------------	-------------------------------	-----------------------------------	--------------------------------

15 • Vogal ou número?

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, declare uma variável **caractere** e atribua a ela uma vogal ou um número. Na sequência, utilize a função **print()** para exibir se o valor guardado na variável é uma vogal ou um número.

2.4 Repetindo instruções: a estrutura While

Nas seções anteriores, você aprendeu uma estrutura importante em programação: a estrutura condicional. Com essa estrutura, é possível, por exemplo, realizar verificações mais robustas em nossos códigos, fazendo com que o nosso algoritmo execute tarefas baseado em uma ou outra condição. No entanto, para além dessa estrutura, há outras tão importantes quanto, como a **estrutura de repetição While** (enquanto):



A **estrutura de repetição While** é bem parecida com a condicional **`if...else`**. Porém, o **While** permite que você execute um bloco de código repetidamente, **enquanto uma condição específica for verdadeira**. Dessa forma, assim como no **`if...else`**, o **While** precisa de uma condição, que é o que fará a repetição. Assim, se a condição for *enquanto um valor for menor do que 5*, a repetição acontecerá até que esse valor seja **maior ou igual** a 5:

The screenshot shows a code editor with a dark theme. On the left, there is a play button icon. The main area contains the following Python code:

```

numero = 1
while numero <= 5:
    print(numero)
    numero = numero + 1
  
```

Below the code, the output is displayed in a light gray box:

```

1
2
3
4
5
  
```

No exemplo anterior, foi necessário, além da condição, **um controle para a repetição**. Esse controle é o que fará a repetição parar em algum momento (no caso, quando o valor for maior ou igual a 5). Por essa razão, um **While** precisa de uma **condição** e de um **controle** para que funcione corretamente, evitando loops infinitos, por exemplo. No código anterior, o que controlará o **While** é a operação de incremento: **numero = numero + 1**.

Mas o que aconteceria caso não colocássemos essa instrução? Bom, nesse caso, teríamos um loop infinito. Veja o exemplo a seguir:

```
numero = 1  
  
while numero <= 5:  
    print(numero)
```

No código acima, temos uma condição bem definida (**enquanto número for menor-igual a cinco**), mas não temos um controle, quer dizer, não temos nada que faça a condição While ser interrompida. Por essa razão, ao executarmos o código, teremos uma **repetição sem fim** (loop infinito) de vários números 1, afinal não há nada controlando o **While**. Se quiséssemos resolver o problema desse exemplo, bastaria incluir a instrução **numero = numero + 1** ou adotar uma outra estratégia. Veja o exemplo a seguir:

```
numero = 1  
controle = True  
  
while controle:  
    print(numero)  
    numero = numero + 1  
  
    if numero > 5:  
        controle = False
```

```
1  
2  
3  
4  
5
```

Para resolver o problema anterior, criamos uma variável **controle** e atribuímos a ela um valor **True**. Assim, quando o valor da variável **numero** for maior do que 5, a variável **controle** terá seu valor alterado para **False**, e o loop será encerrado. Temos, portanto, uma **cláusula de controle**: um **if** que faz a verificação e, se verdadeira a condição, interrompe o funcionamento do **While**. Note que a saída foi a mesma do exemplo do primeiro loop, mas dessa vez o passo a passo, o **algoritmo** para resolver, foi diferente. Ou seja, **algoritmos** diferentes podem levar ao mesmo resultado!

Veja outros exemplos com a **estrutura de repetição While**:

Fazendo um contador decrescente com While



```
numero = 5
controle = True

while controle:
    print(numero)
    numero = numero - 1

    if numero == 0:
        controle = False

5
4
3
2
1
```

No exemplo anterior, temos a variável **numero** com o valor 5 atribuído. Agora, ao invés de fazer um **incremento**, ou seja, somar 1 à variável, fizemos um **decremento**, subtraímos 1. Assim, quando o valor na variável **numero** for igual a 0, a variável de controle terá seu valor alterado para **False**, interrompendo a repetição e evitando que o loop fique infinito.

Repetindo uma palavra



```
numero = 5
frase = "Eu sou programador!"

while numero > 0:
    print(frase)
    numero = numero - 1

Eu sou programador!
```

Nesse exemplo, utilizamos o laço de repetição para repetir uma frase 5 vezes, sendo possível exibir letras, palavras e outras informações, para muito além dos números. Ou seja, pode-se repetir qualquer coisa, desde que haja uma **condição bem definida** e um **controle**.

Por fim, um último exemplo, utilizando o **While** para somar uma sequência de número:



```
numero = 1
soma = 0

while numero <= 10:
    soma = soma + numero
    numero = numero + 1
else:
    print(soma)
```

55

No código acima, declaramos a variável **soma** para receber a soma dos números até 10. Assim, enquanto a variável **numero** tiver um valor menor igual a 10, a repetição fará a soma (**soma = soma + numero**) e fará o incremento da variável **numero** para que ela se aproxime cada vez mais de 10 e, em dado momento, interrompa a repetição. Ao final, quando não for mais possível repetir, exibimos o valor da variável **soma**. Para isso, utilizamos o **else** (exceção). Ou seja, quando a condição de repetição não for mais atendida, faremos outra coisa, conforme definido dentro do bloco **else**.

Ufa! Chegamos ao final de mais um tópico importante: a **estrutura de repetição While**. Você está cada vez mais perto de dar os próximos passos na jornada como programador(a). Para isso, vamos colocar em prática o que foi aprendido até aqui. Fique tranquilo, nenhum desafio é impossível para você, *superprogramador(a)*! E então, vamos lá?

Mãos ao código | Hora de praticar

16 • Contagem regressiva de 10 a 1:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, faça um algoritmo que utilize um loop **While** para exibir uma contagem regressiva de 10 a 1.

17 • Contando de 2 em 2 até 20:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, faça um algoritmo que utilize um loop **While** para contar de 2 em 2 até 20.

Dica: experimente escrever a seguinte instrução nos parênteses do comando **print()**: *numero * 2*.

18 • Soma dos números de 1 a 100:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, faça um algoritmo que utilize um loop **While** para calcular a soma de todos os números de 1 a 100. Finalmente, exiba a soma dos números.

19 • Contando de 3 em 3 até 30:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, faça um algoritmo que utilize um loop **While** para contar de 3 em 3 até 30.

Dica: experimente escrever a seguinte instrução nos parênteses do comando `print(): numero * 3.`

20 • Exibindo apenas os números pares:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, faça um algoritmo que utilize um loop **While** para exibir apenas os números pares até 10. Para isso, dentro do bloco do **While**, crie uma condição:



```
if numero % 2 == 0:  
    print(numero)
```

Não se esqueça de incrementar a variável **numero** fora do bloco do **if**.

Infinity Quiz | Teste seus conhecimentos

1 • Qual é a estrutura condicional básica em Python usada para tomar decisões?

- a) for
- b) if
- c) switch
- d) when

2 • Em Python, sempre que uma condição for verdadeira será executado:

- a) o código dentro do bloco **else**.
- b) o código dentro do bloco **if**.
- c) o código dentro do bloco **else if**.
- d) nada será executado.

3 • Qual é a finalidade da estrutura de repetição While em Python?

- a) Repetir um bloco de código enquanto uma condição for verdadeira.
- b) Repetir um bloco de código um número específico de vezes menos 3.
- c) Executar um bloco de código apenas uma vez.
- d) Parar a execução do programa imediatamente.

4 • O que acontece se a condição em uma estrutura While for sempre verdadeira, sem uma maneira de se tornar falsa?

- a) O programa entra em um loop infinito.
- b) O programa encerra imediatamente.
- c) A estrutura **While** não é suportada em Python.
- d) O programa gera um erro.

5 • A instrução elif serve para:

- a) Definir uma exceção para a regra verificada no **if**.
- b) Definir uma condição falsa para não ter que usar o **if**.
- c) Definir outra regra para a condição.
- d) Não existe a instrução **elif** em Python.

Respostas Infinity Quiz:

1 – B; 2 – B; 3 – A; 4 – A; 5 – C.

Resumindo tudo até aqui...

- Uma **estrutura condicional** visa verificar regras que, se atendidas, executarão certos códigos.
 - A estrutura **if...else** verifica se uma regra é verdadeira (**if**). Caso seja, entraremos no bloco **if** e executaremos alguns comandos. Caso contrário, entraremos no bloco **else** e executaremos outros comandos.
 - Para melhorar ainda mais nossos algoritmos, podemos utilizar operadores relacionais para verificar igualdade e diferença entre dados ou a relação de maioria ou menoridade.
- A estrutura de repetição **While** repete um conjunto de instruções enquanto uma regra for verdadeira.
- Quando a regra não for mais verdadeira, o laço de repetição é interrompido.
- Para que um loop **While** funcione corretamente, são necessários uma condição e um controle. Caso contrário, poderemos ter um loop sem fim.

Capítulo 3

Seus programas ainda mais espertos

3.1 Listas em Python

Imagine que você está indo ao supermercado para fazer compras. Você cria uma lista de itens que precisa comprar, como pão, leite, frutas, legumes e produtos de higiene pessoal. Essa lista é sua maneira de se organizar, garantindo que você não se esqueça de nada e economize tempo durante as compras. Na vida real, uma lista ajuda a manter as coisas organizadas e acessíveis, facilitando a gestão de tarefas. Agora, você pode estar se perguntando: *mas o que são listas em Python?*

Bom, se pensarmos “fora do computador”, a resposta para essa pergunta não parece tão difícil, afinal uma lista seria uma relação de nomes, números, informações diversas etc. Assim, é possível que, por exemplo, tenhamos uma lista de pessoas que estão autorizadas a entrar em uma festa, uma lista de alunos de uma classe (a chamada) e até mesmo uma lista de compras, em que usualmente registramos os itens que pretendemos comprar na feira, no sacolão, no supermercado ou em outro lugar para que nos lembremos de tudo.

Agora que pensamos um pouco sobre a lista externamente ao contexto da programação, compreenderemos um pouco melhor a estrutura e o comportamento de uma lista em Python.

Afinal, essa **estrutura de dado** será utilizada para a resolução de muitos problemas, além de ampliar as possibilidades de criação/desenvolvimento durante a realização de uma tarefa que demande conhecimentos em Python.



Uma lista é uma estrutura de dados que pode armazenar vários valores de uma só vez. É como se a lista fosse uma caixa com várias divisórias. Isso quer dizer que você pode guardar em uma lista um valor, dois valores, três valores... cem valores ou quantos forem necessários. Assim, ao invés de criar uma variável para cada número que você deseja guardar, é possível criar uma lista e armazenar todos os valores lá dentro.

Primeiramente, a lista é uma estrutura de dados, ou seja, é um modo como organizamos dados primitivos ou não em uma variável desse tipo (lista). Dados primitivos, por sua vez, são aqueles tipos mais comuns que você viu em sala de aula: inteiro (int), decimal/real (float), texto (str), lógico (verdadeiro ou falso) etc. Ou seja, uma lista consegue armazenar outros tipos de dados, por exemplo, uma lista de números pares:



```
pares = [2, 3, 4, 5, 6]
```

No exemplo acima, temos uma variável de nome **pares** com uma lista de valores inteiros atribuídos a ela. Com a lista, foi possível armazenar cinco valores sem a necessidade de criar cinco variáveis. Ou seja, uma **estrutura de dados** lista pode armazenar vários dados, facilitando a vida do programador. Veja outros exemplos com listas:

Listas de números ímpares



```
ímpares = [1, 3, 5, 7, 9]
```

Listas de vogais



```
vogais = ["a", "e", "i", "o", "u"]
```

Listas de palavras



```
palavras = ["sol", "lua", "dia", "noite"]
```

Listas com valores de vários tipos



```
lista_mista = [1, "a", "olá", False]
```

Conforme os exemplos anteriores, é perfeitamente possível guardar inúmeros tipos de dados em uma lista, inclusive **dados de tipos diferentes**. Com isso, sempre que utilizarmos a função **print()**, por exemplo, teremos exibido o conjunto de todos os valores que a lista está guardando:



```
lista_mista = [1, "a", "olá", False]
```

```
print(lista_mista)
```

```
[1, 'a', 'olá', False]
```

E se quiséssemos acessar apenas um valor da lista? Nesse caso, além de ser possível exibir todos os valores da lista, também podemos exibir um valor informado que se encontra em dada posição da lista. Assim, imagine a seguinte lista: [1, 3, 5, 7, 9]. Caso queira exibir apenas o número 1, basta informar **o índice do valor**, ou seja, **a posição que ele ocupa** na lista:



```
impares = [1, 3, 5, 7, 9]
```

```
print(impares[0])
```

1

Observe que, após informar a variável que guarda nossa lista, foi necessário informar, dentro dos colchetes, a posição (**o índice**) do número que queríamos exibir, no caso, o número 1. Mas por qual razão passamos o número 0 nos colchetes? O número 1 não estaria na posição 1?

Em programação, é comum que iniciemos as contagens de índices de uma lista, por exemplo, do número 0. Assim, o primeiro elemento de uma lista será o elemento número 0, o segundo será o 1, o terceiro será o 2, e assim por diante.

Dessa

forma, uma lista com 10 valores terá o 9 como índice final. Veja mais exemplos a seguir sobre índices de uma lista:



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

```
print(numeros[9])
```

10

Além de acessar os índices da lista, também podemos verificar se um determinado valor faz parte (**está contido**) naquela lista:



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

```
if 12 in numeros:  
    print("O número existe na lista")  
else:  
    print("O número não existe na lista")
```

O número não existe na lista

Observe que utilizamos o **operador relacional in** para verificar se um determinado valor está contido na lista. Assim, caso o valor esteja na lista, ele será exibido. Caso contrário, uma mensagem será exibida: "O número não existe na lista". Assim, podemos fazer **algoritmos** ainda mais complexos, como verificar se uma letra é vogal:



```
vogais = ["a", "e", "i", "o", "u"]
letra = "b"

if letra in vogais:
    print("A letra é uma vogal")
else:
    print("A letra não é uma vogal")
```

A letra não é uma vogal

Na prática, listas em Python são bem simples: basta declararmos a variável e atribuir a ela um conjunto de valores dentro dos colchetes. Caso sejam valores do tipo texto, é necessário informá-los dentro das aspas.

Mas chega de conversa e bora praticar! A seguir, você encontrará uma listinha de exercícios para colocar em prática seus conhecimentos.

Mãos ao código | Hora de praticar

21 • Exibindo a lista:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Crie uma lista de palavras da sua preferência e, depois, utilize o comando **print()** para exibir todos os elementos da lista.

22 • Acessando elementos da lista:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Crie uma lista de números inteiros e acesse o primeiro, o último e um elemento no meio da lista. Depois, exiba esses elementos separadamente com o comando **print()**.

23 • Verificação de elemento na lista:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista de palavras e verifique se uma palavra específica está na lista. Use uma estrutura condicional para exibir uma mensagem informando se a palavra está ou não na lista. Caso a palavra exista na lista, utilize o comando **print()** para exibir a própria palavra. Caso contrário, exiba a mensagem: “A palavra informada não existe”.

24 • Acessando o último elemento da lista:

A partir do notebook criado no Google Colab, adicione um novo bloco de código e crie uma lista com três frutas. Na sequência, utilize o comando `print()` para exibir o elemento que está no índice **-1** da lista. Finalmente, teste com os índices **-2** e **-3**.

25 • Descobrindo o tamanho de uma lista:

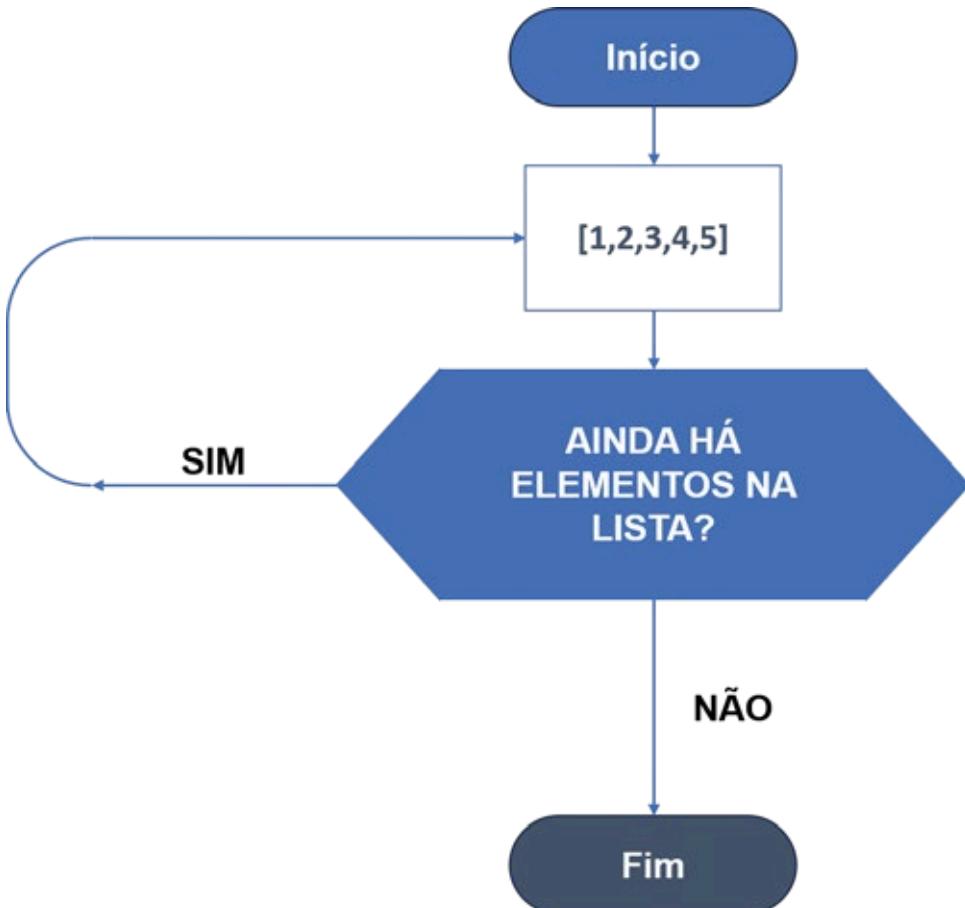
A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista de valores da sua preferência. Na sequência, utilize o comando `len()` para exibir a quantidade de elementos que a lista possui, conforme o exemplo a seguir:



```
print(len(valores))
```

3.2 Estrutura de iteração (laço FOR)

O **laço For** é uma **estrutura de iteração** fundamental na linguagem de programação Python, desempenhando um papel crucial na iteração e repetição de tarefas. Ele oferece uma maneira eficiente de percorrer uma sequência de elementos, como listas, tuplas, strings ou dicionários, e executar um bloco de código para cada elemento presente na sequência.



Um exemplo simples de uso do laço **For** é percorrer uma lista de números e imprimir cada número individualmente:



```
numeros = [1, 2, 3, 4, 5]

for numero in numeros:
    print(numero)

1
2
3
4
5
```

Nesse exemplo, o laço **For** percorre a lista de **numeros** e executa o bloco de código para cada elemento da lista. A variável **numero** assume o valor de cada elemento durante a iteração. Assim, cada vez que o laço faz uma nova repetição, a variável **numero** (singular) recebe o próximo elemento da lista **numeros** (plural). Esse procedimento é repetido até que toda a lista tenha sido percorrida.

Além disso, podemos utilizar a **estrutura de iteração For** para percorrer uma palavra, por exemplo, e exibir apenas as vogais. Veja o exemplo a seguir:



```
vogais = ["a", "e", "i", "o", "u"]
palavra = "abacaxi"

for letra in palavra:
    if letra in vogais:
        print(letra)

a
a
a
i
```

Veja que é perfeitamente possível utilizar uma **condicional** dentro de uma repetição, seja o **While**, seja o **For**. Assim, ao percorrer cada letra da **palavra**, verificamos se a **letra** pertence à lista de vogais. Se sim, exibimos a letra; se não, não exibimos. Poderíamos ainda exibir apenas as consoantes, utilizando a palavra reservada **not** antes da palavra **in**. Veja:



```
vogais = ["a", "e", "i", "o", "u"]
palavra = "abacaxi"

for letra in palavra:
    if letra not in vogais:
        print(letra)
```

b
c
x

O laço **For** é uma ferramenta versátil e poderosa na programação Python, facilitando tarefas que envolvem repetição, iteração e processamento de dados. Desde a manipulação de elementos em listas até a análise de texto e a resolução de problemas complexos, o laço **For** desempenha um papel fundamental na programação Python.

Agora que você já conhece o funcionamento da **estrutura de iteração For**, que tal praticar um pouco e mostrar todo o seu potencial como futuro(a) programador(a)?

Mãos ao código | Hora de praticar

26 • Contando as palavras de uma lista:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista com 7 palavras, sendo uma delas repetida. Exemplo: *amor, banana, amor, solidão, vida, saudade, amor*. Na sequência, utilize o **For** para percorrer a lista e exibir apenas a palavra repetida.

27 • Contagem de elementos:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista que contenham nomes de cores. Use o **For** para percorrer a lista. Toda vez que a cor “vermelho” for identificada na lista, uma variável **soma** deverá ter o seu valor incrementado para contar quantas vezes a cor aparece na lista. Ao final, exiba quantas vezes a cor apareceu.

28 • Multiplicação de elementos:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista de números inteiros e use o **For** para percorrer todos os números da lista e multiplicá-los por 1.75. Utilize o comando **print()** para exibir cada uma das multiplicações.

29 • Exibindo apenas os números negativos:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista de números inteiros (negativos e positivos) e use o **For** para percorrer todos os números da lista e exibir apenas os negativos.

30 • Múltiplos de 3:

A partir do notebook criado no Google Colab, adicione um novo bloco de código. Depois, crie uma lista de números inteiros de 1 até 20. Na sequência, utilize uma **estrutura condicional** para verificar apenas os múltiplos de 3. Finalmente, utilize o comando **print()** para exibir os múltiplos.

Infinity Quiz | Teste seus conhecimentos

1 • O que é uma lista em Python?

- a) Uma função que repete um bloco de código.
- b) Um tipo de dado que armazena uma sequência de valores.
- c) Um operador usado para comparar valores.
- d) Uma variável que armazena apenas números inteiros.

2 • Qual é a finalidade do laço de repetição For em Python?

- a) Realizar uma comparação entre dois valores.
- b) Percorrer uma lista ou uma coleção de itens.
- c) Manipular strings em Python.
- d) Criar funções personalizadas.

3 • Como você acessa o último elemento de uma lista em Python?

- a) Usando o índice -1.
- b) Usando o índice 0.
- c) Usando o índice 1.
- d) Usando o índice 2.

4 • Como você acessa o primeiro elemento de uma lista em Python?

- a) Usando o índice -1.
- b) Usando o índice 0.
- c) Usando o índice 1.
- d) Usando o índice 2.

5 • Qual é a saída do seguinte código Python?



```
frutas = ["maçã", "limão", "abacaxi"]
for fruta in frutas:
    print(fruta)
```

- a) Nada será impresso.
- b) "maçã", "limão", "abacaxi", em linhas separadas.
- c) "maçã", "limão", "abacaxi", na mesma linha.
- d) Um erro será gerado.

Respostas Infinity Quiz:

1 – B; 2 – B; 3 – A; 4 – A; 5 – C.

Resumindo tudo até aqui...

- As listas em Python são estruturas que podem guardar (armazenar) dados de diferentes tipos de uma só vez.
- Cada elemento fica em uma posição da lista.
- O primeiro elemento de uma lista está no índice (posição) 0 e, para acessá-lo, deve-se informar o índice dentro dos colchetes.
- O último elemento de uma lista está no índice (posição) -1.
- Podemos acessar a lista em ordem crescente ou em ordem decrescente. Para isso, basta informar o índice correspondente: 0, 1, 2 ou -1, -2, -3.
- A estrutura de iteração **For** percorre listas e outras coleções em Python. Assim, enquanto houver valores em uma lista, o laço será repetido.
 - Diferentemente do **While**, a estrutura **For** não necessita de controle, uma vez que a repetição dependerá da quantidade de itens da lista ou da coleção.

3.3 Funções em Python: o que são?

Uma função é um **bloquinho de código** que pode fazer várias coisas. Para entendê-la, basta pensar que a **função** é uma **ferramenta** que pode ser utilizada para **executar uma tarefa**, como um martelo que pode ser utilizado para fixar um prego. Ou seja, uma função é como um martelo que fica guardado na caixa de ferramentas. Até que alguém precise do martelo, ele fica lá, quietinho. No entanto, caso alguém precise fixar um prego, basta buscar o martelo na caixa de ferramentas e utilizá-lo:



```
def martelo():
    print("Martelando...")
```

Observe que se utiliza a palavra reservada **def** para dizer ao Python que estamos criando uma função. Depois, basta informar o nome da função – em nosso caso, **martelo**. Na sequência, abrimos e fechamos os parênteses. Por fim, colocamos dois pontos para indicar, a partir de agora, tudo o que queremos colocar no corpo da nossa função.



```
def saudacao():
    print("Seja bem-vindo!")
```

Nossa função está pronta e já pode ser utilizada. Mas, primeiro, que tal praticarmos um pouco mais a estrutura para a declaração de uma função?

Praticando funções em Python

31 • Função Parabéns para Você

Crie um novo notebook no Google Colab para praticar funções. Depois, declare uma função **parabens()** que exiba a mensagem: “**Parabéns para você.**”. Finalmente, depois de declarar a função, salte algumas linhas e utilize a função. Para isso, basta escrever o nome da função seguido dos parênteses, conforme os exemplos vistos anteriormente.

32 • Função Seja Bem-Vindo ao Sistema

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função **saudacao()** que exiba a mensagem: “**Seja bem-vindo ao sistema, Kauê.**”. Finalmente, depois de declarar a função, salte algumas linhas e utilize a função. Para isso, basta escrever o nome da função seguido dos parênteses.

33 • Função Preço da Coxinha

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `preco_coxinha()` que exiba a mensagem: “**Coxinha – R\$ 2.00.**”. Finalmente, depois de declarar a função, salte algumas linhas e utilize a função. Para isso, basta escrever o nome da função seguido dos parênteses.

34 • Função Chamar Bruna

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `chamar_bruna()` que exiba a mensagem: “**Bruna... Bruna... Bruna...**”. Finalmente, depois de declarar a função, salte algumas linhas e utilize a função. Para isso, basta escrever o nome da função seguido dos parênteses.

35 • Função Número

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `numero()` que exiba a mensagem: “**12 é o número!**”. Finalmente, depois de declarar a função, salte algumas linhas e utilize a função. Para isso, basta escrever o nome da função seguido dos parênteses.

Resumindo tudo até aqui...

- Uma função é um bloco de código que faz alguma coisa no programa.
- Para declarar uma função, basta utilizar a palavra reservada **def** seguida do nome da função e dos parênteses.
- Para utilizar a função, basta saltar algumas linhas e chamá-la no código principal do seu programa Python.
- Você pode utilizar uma função várias vezes, sem precisar digitar os mesmos códigos definidos no corpo da sua função.
- Uma função pode chamar outras funções para resolver problemas.

3.3.1 Parâmetros das funções No **exercício 32** da atividade anterior, você declarou uma função chamada **saudacao()**, que exibia a mensagem: “**Seja bem-vindo, Kauê.**”. Imagino que sua função tenha ficado assim:



```
def saudacao():
    print("Seja bem-vindo, Kauê.")
```

Além disso, se você quisesse utilizar sua função, bastaria chamá-la nas próximas linhas:



```
def saudacao():
    print("Seja bem-vindo, Kauê.")

saudacao()
```

Seja bem-vindo, Kauê.

O que aconteceu quando você chamou sua função? Certamente, foi exibida a mensagem que você definiu, não é mesmo? No entanto, imagine que, ao invés de dizermos “**Seja bem-vindo, Kauê.**”, quiséssemos dizer “**Seja bem-vindo, Carlos.**” ou “**Seja bem-vindo, Pedro.**”. Como fazer isso e tornar nossa função mais eficiente?

Nesse caso, podemos definir um **parâmetro** para nossa função **saudacao()** e utilizar esse parâmetro para personalizar nossa mensagem. Para isso, vamos utilizar os parênteses da nossa função e dizer que, para ela funcionar corretamente, nós precisaremos colocar uma coisa “dentro” dela – no caso, um **nome** que será utilizado para personalizar a mensagem. Veja o exemplo a seguir:



```
def saudacao(nome):
    print("Seja bem-vindo,", nome)

saudacao("Kaleb")
```

Seja bem-vindo, Kaleb

Note que, dentro dos parênteses da função **saudacao()**, definimos um parâmetro chamado **nome**. Na sequência, fizemos pequenas alterações no **saudacao()**, adicionando uma vírgula para informar o parâmetro **nome** no lugar de **Kauê**. Finalmente, ao chamarmos a função, foi necessário dizer nos parênteses um valor para o **nome**. Agora, nossa função estará devidamente personalizada e, caso queira mudar o nome de exibição da mensagem, basta informar outro valor para a função.



```
def saudacao(nome):  
    print("Seja bem-vindo,", nome)  
  
saudacao("Kainã")
```

Seja bem-vindo, Kainã

Isso é o **parâmetro de uma função**. Basicamente, fizemos o seguinte: dissemos à função que, para ela funcionar, é necessário passar uma informação para ela – no caso, um nome. Dessa maneira, sempre que a função for utilizada, será necessário informar um nome nos parênteses da função para que ela consiga executar os comandos definidos. Mas o que aconteceria se esse nome não fosse informado? Nossa função continuaria funcionando? Tente executar sua função sem informar um valor para ela e comente o que aconteceu.

A seguir, veja alguns exemplos de parâmetros que podemos passar para uma função e tente testá-los, melhorando as funções que você criou antes.

Função que recebe como parâmetro um número inteiro



```
def idade(idade):  
    print("Você tem", idade, "anos.")  
  
idade(12)
```

Você tem 12 anos.

Função que recebe como parâmetro um número float (real/decimal)



```
def altura(altura):  
    print("Você tem", altura, "de altura.")  
  
altura(1.86)
```

Você tem 1.86 de altura.

Função que recebe como parâmetro um valor lógico



```
def brasileiro(logico):  
    print(logico)  
  
brasileiro(True)
```

True

Crie novas funções a partir desses exemplos. Seja criativo e implemente três novas funções, cada uma recebendo um parâmetro: inteiro, real e lógico. Depois, mostre aos seus familiares, às pessoas próximas, aos amigos e/ou colegas o seu progresso em programação!

A seguir, para que tudo fique ainda melhor, vamos praticar um pouco mais!

Praticando funções em Python

36 • Função Parabéns Personalizada

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função **parabens()** com a mensagem: “**Parabéns para você, [NOME]**”. Para isso, defina nos parênteses da função um parâmetro **nome**. Finalmente, teste sua função.

37 • Função Perfil

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função **perfil()** com a mensagem: “**Seu nome é [NOME], e você tem [IDADE] anos.**”. Para isso, defina nos parênteses da função um parâmetro **nome** e **idade**. Finalmente, teste sua função.

38 • Função Tabela de Preços

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função **tabela_precos()** que exiba a mensagem:
“**Coxinha – R\$ [PRECO1]**”
“**Empada – R\$ [PRECO2]**”
Para isso, defina nos parênteses da função um parâmetro **preco1** e **preco2**. Finalmente, teste sua função.
Dica: utilize um **print()** para o preço da coxinha e outro para o preço da empada.

39 • Função Envelhecer

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função **envelhecer()** com a mensagem: “**Você ficou um ano mais velho, agora com [IDADE] anos.**”. Para isso, defina nos parênteses da função um parâmetro **idade**. Finalmente, teste sua função.

40 • Função Configurações

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `configuracoes()` que exiba a mensagem:

“Marca: [MARCA]”

“Modelo: [MODELO]”

“Preço: R\$ [PRECO]”

“Ligado: [STATUS]”

Para isso, defina nos parênteses da função os parâmetros necessários para que a mensagem seja exibida corretamente. Finalmente, teste sua função.

Resumindo tudo até aqui...

- Uma função pode ter parâmetros para que sejam passados valores a ela.
- Os parâmetros da função são como informações de que ela precisa para funcionar.
- Os valores passados como parâmetro podem ser do tipo texto (**string**), do tipo inteiro (**int**), do tipo real (**float**), do tipo lógico (**bool**) ou de outro tipo qualquer que exista na linguagem Python.
- Uma vez definido(s) o(s) parâmetro(s) da função, é necessário informar o valor do parâmetro sempre que formos utilizar a função. Só assim ela funcionará corretamente.
- Uma função pode ter ou não ter parâmetros. Depende do problema que formos resolver com ela.

3.3.2 Condicionais: nossa função ainda mais funcional

Até aqui, temos desenvolvido algumas funções para fazer coisas em nosso programa. Aprendemos ainda como declarar uma função e como utilizá-la, além de conhecer um pouco sobre os parâmetros da função, que nos ajudam a deixá-la ainda melhor, exibindo mensagens personalizadas. No entanto, temos uma pequena situação para resolver. Imagine o exemplo do **exercício 32**, em que você declarou uma função chamada `saudacao()` que exibia na tela (terminal) a mensagem: “**Seja bem-vindo ao sistema, Kauê.**”. Mais à frente, nós melhoramos essa função para exibir a mensagem personalizada de acordo com o nome informado no parâmetro `nome` da nossa função. Acontece que os exemplos utilizados foram nomes

masculinos: Kauê, Kaleb, Kainã... E se quiséssemos mostrar a mensagem “**seja bem-vindo**” para homens e “**seja bem-vinda**” para mulheres? Como implementar uma função que verifique se o nome é masculino ou feminino e, depois, exiba a mensagem adequada para cada caso?

Veja o exemplo a seguir e observe o que foi feito para resolver o problema anteriormente descrito:

```
def saudacao(nome, sexo):
    if sexo == 'm':
        print("Seja bem-vindo,", nome)
    if sexo == 'f':
        print("Seja bem-vinda,", nome)

saudacao("Anhaporã", "f")
```

Seja bem-vinda, Anhaporã

Para resolver o problema, utilizamos o **if** para testar se o valor do parâmetro **sexo** é igual (**==**) a “**m**” ou igual a “**f**”. Para isso, foi preciso definir outro parâmetro do tipo string: **sexo**. Note que, com poucas implementações, nossa função ficou mais inteligente. Agora, ela pode personalizar a mensagem de acordo com o sexo, o que deixa tudo ainda melhor!

Em resumo, é perfeitamente possível usar condicionais dentro do corpo das funções para melhorar nossos resultados. Veja outros exemplos a seguir:

Verificando se o valor *idade* é maior-igual a 18 com *if...else*:

```
def saudacao(nome, idade):
    if idade >= 18:
        print("Você pode entrar,", nome)
    else:
        print("Você não pode entrar,", nome)

saudacao("Kaike", 17)
```

Você não pode entrar, Kaike

Verificando se o valor *habilitado* é igual a *True* com *if...else*:

```
def pode_dirigir(habilitado):
    if habilitado == True:
        print("Você pode dirigir.")
    else:
        print("Você não pode dirigir.")

pode_dirigir(False)
```

Você não pode dirigir.

Verificando as strings `senha` e `senha_acesso` são iguais com `if...else`:



```
def acesso(senha):
    if senha == "1234":
        print("Logado com sucesso.")
    else:
        print("Senha incorreta.")

acesso("1235")
```

Senha incorreta.

Praticando funções em Python

41 • Função Verificador de Vogais

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `verificador_vogal()` que mostre se a letra informada como valor do parâmetro é uma vogal. Para isso, defina nos parênteses da função um parâmetro `letra`. Finalmente, teste sua função.

42 • Função Verificador de Consoantes

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `verificador_consoante()` que mostre se a letra informada como valor do parâmetro é uma consoante. Para isso, defina nos parênteses da função um parâmetro `letra`. Finalmente, teste sua função..

43 • Função Verificador de Negativos

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `verificador_negativo()` que mostre se o número informado como valor do parâmetro é **negativo**, ou seja, **menor** do que zero. Para isso, defina nos parênteses da função um parâmetro `numero`. Finalmente, teste sua função.

44 • Função Verificador de Positivos

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `verificador_positivo()` que mostre se o número informado como valor do parâmetro é **positivo**, ou seja, **maior** do que zero. Para isso, defina nos parênteses da função um parâmetro `numero`. Finalmente, teste sua função.

45 • Função Verificador de Zeros

Aproveite o notebook anterior e crie um novo bloco de código. Depois, declare uma função `verificador_zero()` que mostre se o número informado como valor do parâmetro é **igual a zero**. Para isso, defina nos parênteses da função um parâmetro `numero`. Finalmente, teste sua função.

Resumindo tudo até aqui...

- Podemos utilizar condicionais no corpo da nossa função para personalizar os resultados que serão exibidos na tela.
- Para fazer uma condição para a função, basta utilizar as instruções `if... elif...else` e fazer verificações a partir dos valores passados aos parâmetros da função.
- É possível utilizar quantas estruturas condicionais forem necessárias para melhorar sua função.

Capítulo final

Até logo, programador(a)

Neste livro, você deu o primeiro passo para desvendar o fascinante mundo da programação. Aprendeu os conceitos fundamentais, descobriu como escrever seu primeiro código e explorou as diversas linguagens e ferramentas disponíveis. Parabéns por chegar até aqui!

A programação é uma habilidade poderosa que se tornou essencial em nosso mundo, que é cada vez mais digital. Ela não apenas nos permite criar aplicativos, sites e software, mas também tem aplicações em praticamente todos os setores, desde a medicina e a ciência até a arte e o entretenimento. As oportunidades com a programação são verdadeiramente infinitas, e você pode usá-la para solucionar problemas do dia a dia, desenvolver aplicações para web e celulares, criar complexos e trabalhar com inteligência artificial etc. Então, o que vem a seguir? A

jornada de aprendizado em programação é contínua e repleta de descobertas. À medida que avança, lembre-se de que é normal enfrentar desafios e obstáculos. A perseverança e a curiosidade são suas melhores amigas. Continue estudando, pratique regularmente e nunca pare de explorar.

Apêndice A

Tutorial de uso do Google Colab para iniciantes em Python

O Google Colab (Colaboratory) é uma plataforma gratuita baseada na nuvem que permite a criação e execução de Notebooks Jupyter diretamente no seu navegador. É uma ferramenta poderosa para aprender, escrever código em Python e realizar tarefas de análise de dados e aprendizado de máquina. Neste tutorial, você aprenderá como começar a usar o Google Colab, criar seu primeiro notebook e executar código Python.

Passo 1: acessando o Google Colab

Abra o seu navegador e accese o Google Colab em colab.research.google.com

Você precisará fazer login com sua conta do Google. Se você não tiver uma conta, crie uma gratuitamente.

Passo 2: criando um novo notebook

Após fazer login, você será direcionado para a página inicial do Google Colab.

Clique no botão “Novo Notebook” para criar um novo notebook em branco.

Passo 3: conhecendo o ambiente do notebook

Barra de ferramentas: contém botões para executar células, adicionar novas células, salvar o notebook e muito mais.

Células: existem dois tipos principais de células: células de texto (Markdown) e células de código (Python).

Barra lateral à direita: oferece guias para explorar arquivos, comandos, configurações e outros recursos.

Passo 4: trabalhando com células

Células de texto (Markdown): use essas células para adicionar texto explicativo, documentação e formatação. Você pode formatar o texto usando Markdown, uma linguagem de marcação simples. Células de código (Python): utilize essas células para escrever e executar código Python. Insira seu código na célula e clique no botão “Executar” ou pressione Shift + Enter para executá-lo.

Passo 5: executando código Python

Para executar código Python em uma célula de código, escreva o código na célula e clique no botão “Executar” ou pressione Shift + Enter.

O resultado da execução será exibido abaixo da célula.

Passo 6: salvando e carregando notebooks

Para salvar seu notebook, clique em “Arquivo” e escolha “Salvar” ou use o atalho Ctrl + S.

Você pode carregar notebooks existentes clicando em “Arquivo” e selecionando “Carregar notebook”. Isso permite que você acesse e continue trabalhando em projetos anteriores.

Passo 7: compartilhando notebooks

Você pode compartilhar seus notebooks com outras pessoas. Clique em “Compartilhar” no canto superior direito e defina as permissões de compartilhamento.

Passo 8: encerrando uma sessão

Quando você fecha o navegador ou a guia, o Colab encerrará a sessão e qualquer progresso não salvo será perdido. Certifique-se de salvar seu trabalho antes de sair.

Apêndice B / Gabarito dos exercícios

1. Exibindo seu nome na tela:

```
nome = "Kauê"
print(nome)
```

Kauê

2. Exibindo uma frase na tela:

```
frase = "A melhor maneira de prever o futuro é
criá-lo"
autor = "Peter Drucker"

print(frase)
print(autor)
```

A melhor maneira de prever o futuro é criá-lo
Peter Drucker

3. Resolvendo operações matemáticas:

```
numero1 = 10
numero2 = 4

print(numero1 + numero2)
print(numero1 - numero2)
```

14
6

4. Exibindo valores no mesmo print():

```
numero1 = 10
numero2 = 4

print(numero1, numero2)
```

10 4

5. Exibindo texto e número no mesmo print():

```
nome = "Kauê"
idade = 12

print(nome, idade)
```

Kauê 12

6. Verificando se um número é maior do que cinco:

```
numero = 7

if numero > 5:
    print("O número é maior do que 5")
else:
    print("O número não é maior do que 5")
```

O número é maior do que 5

7. Verificando qual o maior de dois números:

```
numero1 = 3
numero2 = 10

if numero1 > numero2:
    print("O primeiro número é maior do que o
segundo")
else:
    print("O primeiro número é menor do que o
segundo")
```

O primeiro número é menor do que o segundo

8. Verificando qual o menor de dois números:

```
numero1 = 10
numero2 = 3

if numero1 < numero2:
    print("O primeiro número é menor do que o
segundo")
else:
    print("O primeiro número é maior do que o
segundo")
```

O primeiro número é maior do que o segundo

9. Verificando senha de acesso ao sistema:

```
senha = "kaue1234"

if senha == "admin1234":
    print("Olá, administrador. Bem-vindo ao
sistema")
else:
    print("Você não é um usuário administrador")
```

Você não é um usuário administrador

10. Verificando se um número é 3, 6 ou 9:

```
numero = 7

if numero == 3:
    print("O número é múltiplo de 3")
elif numero == 6:
    print("O número é múltiplo de 3")
elif numero == 9:
    print("O número é múltiplo de 3")
else:
    print("O número não é múltiplo de 3")
```

O número não é múltiplo de 3

11. Determinando a faixa etária:

```
idade = 34

if idade < 13:
    print("Criança")
elif idade < 20:
    print("Adolescente")
elif idade < 65:
    print("Adulto")
else:
    print("Idoso")
```

Adulto

12. Classificação de nota escolar:

```
nota = 9.6

if nota <= 4:
    print("Reprovado")
elif nota <= 6:
    print("Recuperação")
else:
    print("Aprovado")
```

Aprovado

13. Calculadora de IMC:

```
peso = 103.3
altura = 1.85
IMC = peso / (altura * altura)

if IMC < 18.5:
    print("Abaixo do peso")
elif IMC < 25.0:
    print("Peso normal")
elif IMC < 34.9:
    print("Sobrepeso")
else:
    print("Obesidade")
```

Sobrepeso



14. Decisão de compra:

```
preco = 2.75

if preco < 5.0:
    print("Barato")
elif preco <= 10.0:
    print("Razoável")
else:
    print("Caro")
```

Barato

15. Vogal ou número?

```
caractere = "4"

if caractere == "a":
    print("É vogal")
elif caractere == "e":
    print("É vogal")
elif caractere == "i":
    print("É vogal")
elif caractere == "o":
    print("É vogal")
elif caractere == "u":
    print("É vogal")
else:
    print("É número")
```

É número

16. Contagem regressiva de 10 a 1:

```
numero = 10

while numero > 0:
    print(numero)
    numero = numero - 1
```

10
9
8
7
6
5
4
3
2
1



17. Contando de 2 em 2 até 20:

```
numero = 1

while numero <= 10:
    print(numero*2)
    numero = numero + 1
```

2
4
6
8
10
12
14
16
18
20

18. Soma dos números de 1 a 100:

```
soma = 0
numero = 1

while numero <= 100:
    soma = soma + numero
    numero = numero + 1
else:
    print(soma)
```

5050

19. Contando de 3 em 3 até 30:

```
numero = 1

while numero <= 10:
    print(numero*3)
    numero = numero + 1
```

3
6
9
12
15
18
21
24
27
30

20. Exibindo apenas os números pares:

```
numero = 1

while numero <= 10:
    if numero % 2 == 0:
        print(numero)

    numero = numero + 1
```

2
4
6
8
10

21. Exibindo a lista:

```
numeros = [1, 2, 3, 4, 5]

print(numeros)
```

[1, 2, 3, 4, 5]

22. Acessando elementos da lista:

```
numeros = [1, 2, 3, 4, 5]  
  
print(numeros[0])  
print(numeros[2])  
print(numeros[-1])  
  
1  
3  
5
```

23. Verificação de elemento na lista:

```
palavras = ["dia", "tarde", "noite"]  
  
if "sol" in palavras:  
    print("sol")  
else:  
    print("A palavra informada não existe")  
  
A palavra informada não existe
```

24. Acessando o último elemento da lista:

```
frutas = ["pêra", "maçã", "uva"]  
  
print(frutas[-1])  
print(frutas[-2])  
print(frutas[-3])  
  
uva  
maçã  
pêra
```

25. Descobrindo o tamanho de uma lista:

```
valores = [1, 2, 3, 4, 5]  
  
print(len(valores))  
  
5
```

26. Contando as palavras de uma lista:

```
palavras = [  
    "amor",  
    "banana",  
    "solidao",  
    "amor",  
    "vida",  
    "saudade",  
    "amor"  
]  
  
for palavra in palavras:  
    if palavra == "amor":  
        print(palavra)  
  
amor  
amor  
amor
```

27. Contagem de elementos:

```
palavras = [
    "azul",
    "amarelo",
    "vermelho",
    "branco",
    "vermelho",
    "verde",
    "vermelho"
]

quantidade = 0

for palavra in palavras:
    if palavra == "vermelho":
        quantidade = quantidade + 1

print("A cor vermelho apareceu", quantidade,
"vezes")
```

A cor vermelho apareceu 3 vezes

28. Multiplicação de elementos:

```
numeros = [1, 2, 3, 4, 5]

for numero in numeros:
    print(numero * 1.75)
```

1.75
3.5
5.25
7.0
8.75

29. Exibindo apenas os números negativos:

```
numeros = [-2, -1, 0, 1, 2]

for numero in numeros:
    if numero < 0:
        print(numero)
```

-2
-1

30. Múltiplos de 3:

```
numeros = [
    1, 2, 3, 4, 5, 6,
    7, 8, 9, 10, 11,
    12, 13, 14, 15, 16,
    17, 18, 19, 20
]

for numero in numeros:
    if numero % 3 == 0:
        print(numero)
```

3
6
9
12
15
18

31. Função Parabéns para Você:



```
def parabens():
    print("Parabéns para você.")

parabens()

Parabéns para você.
```

32. Função Seja Bem-Vindo ao Sistema:



```
def saudacao():
    print("Seja bem-vindo ao sistema, Kauê.")

saudacao()

Seja bem-vindo ao sistema, Kauê.
```

33. Função Preço da Coxinha:



```
def preco_coxinha():
    print("Coxinha - R$ 2.00.")

preco_coxinha()

Coxinha - R$ 2.00.
```

34. Função Chamar Bruna:



```
def chamar_bruna():
    print("Bruna... Bruna... Bruna...")

chamar_bruna()

Bruna... Bruna... Bruna...
```

35. Função Número:



```
def numero():
    print("12 é o número!")

numero()

12 é o número!
```

36. Função Parabéns Personalizada:



```
def parabens(nome):
    print("Parabéns para você,", nome)

parabens("Caio")

Parabéns para você, Caio
```

37. Função Perfil:



```
def perfil(nome, idade):
    print("Seu nome é", nome, "e você tem",
          idade, "anos")

perfil("Kainã", 26)

Seu nome é Kainã e você tem 26 anos
```

38. Função Tabela de Preços:



```
def tabela_precos(preco1, preco2):
    print("Coxinha - R$", preco1)
    print("Empada - R$", preco2)

tabela_precos(3.15, 4.40)

Coxinha - R$ 3.15
Empada - R$ 4.40
```

39. Função Envelhecer:



```
def envelhecer(idade):
    print("Você ficou um ano mais velho, agora
com", idade, "anos.")

envelhecer(12)
```

Você ficou um ano mais velho, agora com 12 anos.

40. Função Configurações:



```
def configuracoes(marca, modelo, preco,
status):
    print("Marca:", marca)
    print("Modelo:", modelo)
    print("Preço: R$", preco)
    print("Ligado:", status)

configuracoes("Xiaomi", "MI70", 3.000,
"ligado")
```

Marca: Xiaomi
Modelo: MI70
Preço: R\$ 3.000
Ligado: Ligado

41. Função Verificador de Vogais:



```
def verificador_vogal(letra):
    vogais = ["a", "e", "i", "o", "u"]

    if letra in vogais:
        print("É vogal")

verificador_vogal("a")
```

É vogal.

42. Função Verificador de Consoantes:



```
def verificador_consoante(letra):
    vogais = ["a", "e", "i", "o", "u"]

    if letra not in vogais:
        print("É consoante")

verificador_consoante("b")
```

É consoante

43. Função Verificador de Negativos:



```
def verificador_negativo(numero):
    if numero < 0:
        print("O número é negativo")

verificador_negativo(-1)
```

O número é negativo

44. Função Verificador de Positivos:



```
def verificador_positivo(numero):
    if numero > 0:
        print("O número é positivo")

verificador_positivo(3)
```

O número é positivo

45. Função Verificador de Zeros:



```
def verificador_zero(numero):
    if numero == 0:
        print("O número é zero")

verificador_zero(0)
```

O número é zero

Dados Internacionais de Catalogação na Publicação (CIP)
Angelica Ilacqua CRB-8/7057

Infinity School
Aprendendo lógica de programação com Python : um guia Infinity para iniciantes / Infinity School. – [S. I.] : Reality Editora, 2024.

64 p.

ISBN: 978-65-980058-5-6

1. Python (Linguagem de programação de computador)
2. Lógica de programação I. Título

23-6933

CDD 005.13

Índices para catálogo sistemático:
1. Python (Linguagem de programação de computador)



www.infinityschool.com.br



9 786598 005856

REALITY EDITORA