

**Centro Estadual de Educação Profissional José Pacífico de
Moura Neto**

Unidade Curricular: Lógica de Programação

Professor: Josenilton de Aragão Lima

APOSTILA DE LÓGICA DE PROGRAMAÇÃO

TERESINA (PI), ABRIL DE 2024

INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

1

Objetivos

Apresentar os conceitos elementares de lógica e sua aplicação no cotidiano. Definir algoritmo. Estabelecer uma relação entre lógica e algoritmos: a lógica de programação. Exemplificar a aplicação dos algoritmos utilizando situações do dia-a-dia. Comparar as principais formas de representação dos algoritmos.

- ▶ Introdução à lógica de programação
- ▶ Algoritmizando a lógica
- ▶ Conceitos e exemplos de algoritmos
- ▶ Noções de fluxos de controle

NOÇÕES DE LÓGICA

O QUE É LÓGICA?

O uso corriqueiro da palavra lógica está normalmente relacionado à coerência e à racionalidade. Frequentemente se associa lógica apenas à matemática, não se percebendo sua aplicabilidade e sua relação com as demais ciências.

Podemos relacionar a lógica com a ‘correção do pensamento’, pois uma de suas preocupações é determinar quais operações são válidas e quais não são, fazendo análises das formas e leis do pensamento. Como filosofia, ela procura saber por que pensamos assim não de outro jeito. Com arte ou técnica, ela nos ensina a usar corretamente as leis do pensamento.

Poderíamos dizer também que a lógica é a ‘arte de bem pensar’, que é a ‘ciência das formas do pensamento’. Visto que a forma mais complexa do pensamento é o raciocínio, a lógica estuda a ‘correção do raciocínio’. Podemos ainda dizer que a lógica tem em vista a ‘ordem da razão’. Isso dá a entender que a nossa razão pode funcionar desordenadamente. Por isso, a lógica estuda e ensina a colocar ‘ordem no pensamento’.

Exemplos

- a. Todo mamífero é um animal.
 Todo cavalo é um mamífero.
 Portanto, todo cavalo é um animal.
- b. Kaiton é país do planeta Stix.
 Todos os Xinpins são de Kaiton.
 Logo, todos os Xinpins são Stixianos.

Esses exemplos ilustram silogismos, que no estudo da Lógica Proposicional (ou Cálculo Sentencial) representam um argumento composto de duas premissas e uma conclusão; e está estabelecendo uma relação, que pode ser válida ou não. Esse é um dos objetivos da lógica, o estudo de técnicas de formalização, dedução e análise que permitam verificar a validade de argumentos. No caso dos exemplos, ambos são válidos.

Devemos ressaltar que, apesar da aparente coerência de um encadeamento lógico, ele pode ser válido ou não em sua estrutura. Nesse sentido, a lógica também objetiva a criação de uma representação mais formal, que se contrapõe à linguagem natural, que é suscetível a argumentações informais.

EXISTE LÓGICA NO DIA-A-DIA?

Sempre que pensamos, a lógica ou a ilógica necessariamente nos acompanham. Quando falamos ou escrevemos, estamos expressando nosso pensamento, logo, precisamos usar a lógica nessas atividades. Podemos perceber a importância da lógica em nossa vida, não só na teoria, como na prática, já que, quando queremos pensar, falar, escrever ou agir corretamente, precisamos colocar ‘ordem no pensamento’, isto é, utilizar lógica.

Exemplos

- a. A gaveta está fechada.
 A caneta está dentro da gaveta.
 Precisamos primeiro abrir a gaveta para depois pegar a caneta.
- b. Anacleto é mais velho que Felisberto.
 Felisberto é mais velho que Marivaldo.
 Portanto, Anacleto é mais velho que Marivaldo.

MAS E A LÓGICA DE PROGRAMAÇÃO?

Significa o uso correto das leis do pensamento, da ‘ordem da razão’ e de processos de raciocínio e simbolização formais na programação de computadores, objetivando a racionalidade e o desenvolvimento de técnicas que cooperem para a produção de soluções logicamente válidas e coerentes, que resolvam com qualidade os problemas que se deseja programar.

O raciocínio é algo abstrato, intangível. Os seres humanos têm a capacidade de expressá-lo através da palavra falada ou escrita, que por sua vez se baseia em um determinado idioma, que segue uma série de padrões (gramática). Um mesmo raciocínio pode ser expresso em

qualquer um dos inúmeros idiomas existentes, mas continuará representando o mesmo raciocínio, usando apenas outra convenção.

Algo similar ocorre com a Lógica de Programação, que pode ser concebida pela mente treinada e pode ser representada em qualquer uma das inúmeras linguagens de programação existentes. Essas, por sua vez, são muito atreladas a uma grande diversidade de detalhes computacionais, que pouco têm a ver com o raciocínio original. Para escapar dessa torre de Babel e, ao mesmo tempo, representar mais fielmente o raciocínio da Lógica de Programação, utilizamos os Algoritmos.

O QUE É UM ALGORITMO?

O objetivo principal do estudo da Lógica de Programação é a construção de algoritmos coerentes e válidos. Mas o que é um algoritmo?

Um **algoritmo** pode ser definido como uma seqüência de passos que visam a atingir um objetivo bem definido.

Na medida em que precisamos especificar uma seqüência de passos, é necessário utilizar ordem, ou seja, ‘pensar com ordem’, portanto precisamos utilizar lógica.

Apesar do nome pouco usual, algoritmos são comuns em nosso cotidiano, como, por exemplo, uma receita de bolo. Nela está descrita uma série de ingredientes necessários e uma seqüência de diversos passos (ações) que devem ser fielmente cumpridos para que se consiga fazer o alimento desejado, conforme se esperava antes do início das atividades (objetivo bem definido).

Quando elaboramos um algoritmo, devemos especificar ações claras e precisas, que a partir de um estado inicial, após um período de tempo finito, produzem um estado final previsível e bem definido. Isso significa que o algoritmo fixa um padrão de comportamento a ser seguido, uma norma de execução a ser trilhada, com vistas a alcançar, como resultado final, a solução de um problema, garantindo que sempre que executado, sob as mesmas condições, produza o mesmo resultado.

ALGORITMIZANDO A LÓGICA

POR QUE É IMPORTANTE CONSTRUIR UM ALGORITMO?

Um algoritmo tem por objetivo representar mais fielmente o raciocínio envolvido na Lógica de Programação e, dessa forma, permite-nos abstrair de uma série de detalhes computacionais, que podem ser acrescentados mais tarde. Assim, podemos focalizar nossa atenção naquilo que é importante: a lógica da construção de algoritmos.

Outra importância da construção dos algoritmos é que uma vez concebida uma solução algorítmica para um problema, esta pode ser traduzida para qualquer linguagem de programação e ser agregada das funcionalidades disponíveis nos diversos ambientes; costumamos denominar esse processo de codificação.

VAMOS A UM EXEMPLO?

Podemos escrever um primeiro algoritmo de exemplo, utilizando português coloquial, que descreva o comportamento na resolução de um determinada atividade, como, por

exemplo, a troca de uma lâmpada. Apesar de aparentemente óbvia demais, muitas vezes realizamos esse tipo de atividade inconscientemente, sem percebermos seus pequenos detalhes, que são as ações que nos levam a alcançar o objetivo proposto.

Vejamos esse primeiro algoritmo, descrito passo a passo:

ALGORITMO 1.1 Troca de lâmpada

- pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;
 - subir na escada;
 - retirar a lâmpada velha;
 - colocar a lâmpada nova.
-

Involuntariamente, já seguimos uma determinada **seqüência** de ações que, representadas nesse algoritmo, fazem com que ele seja seguido naturalmente por qualquer pessoa, estabelecendo um padrão de comportamento, pois qualquer pessoa agiria da mesma maneira.

A **seqüenciação** é uma convenção com o objetivo de reger o fluxo de execução do algoritmo, determinando qual a primeira ação a ser executada e qual ação vem a seguir. Nesse caso, a seqüência é linear, de cima para baixo, assim como é a seqüência pela qual lemos um texto, de cima para baixo e da esquerda para a direita.

Reexaminando o algoritmo anterior, notamos que ele tem um objetivo bem definido: trocar uma lâmpada. Porém, e se a lâmpada não estivesse queimada? A execução das ações conduziria a uma troca, independentemente de a lâmpada estar ou não queimada, porque não foi prevista essa possibilidade em sua construção.

Para solucionar essa necessidade, podemos efetuar um teste, a fim de verificar se a lâmpada está ou não queimada. Uma solução para esse novo algoritmo seria:

ALGORITMO 1.2 Troca de lâmpada com teste

- pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;
 - acionar o interruptor;
 - se a lâmpada não acender, então
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar a lâmpada nova.
-

Agora estamos ligando algumas ações à condição lâmpada não acender, ou seja, se essa condição for verdadeira (lâmpada queimada) efetuaremos a troca da lâmpada, seguindo as próximas ações:

- subir na escada;
- retirar a lâmpada queimada;
- colocar a lâmpada nova.

Se a condição lâmpada não acender for falsa (a lâmpada está funcionando), as ações relativas à troca da lâmpada não serão executadas, e a lâmpada (que está em bom estado) não será trocada.

O que ocorreu nesse algoritmo foi a inclusão de um teste **seletivo**, através de uma condição que determina qual ou quais ações serão executadas (note que anteriormente, no **Algoritmo 1.1**, todas as ações eram executadas), dependendo da inspeção da condição resultar em verdadeiro ou falso.

Esse algoritmo está correto, uma vez que atinge seu objetivo, porém, pode ser melhorado, uma vez que buscamos uma escada e uma lâmpada sem saber se serão necessárias. Mudemos então o teste condicional se a lâmpada não acender para o início da sequência de ações:

ALGORITMO 1.3 Troca de lâmpada com teste no início

- acionar o interruptor;
 - se a lâmpada não acender, então
 - pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;
 - acionar o interruptor;
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar a lâmpada nova.
-

Observe que, agora, a ação acionar o interruptor é a primeira do algoritmo e a condição lâmpada não acender já é avaliada. Nesse caso, pegar uma escada até colocar a lâmpada nova dependem de a lâmpada estar efetivamente queimada. Há muitas formas de resolver um problema, afinal cada pessoa pensa e age de maneira diferente, cada indivíduo tem uma heurística própria. Isso significa que, para esse mesmo problema de trocar lâmpadas, poderíamos ter diversas soluções diferentes e corretas (se atingissem o resultado desejado de efetuar a troca), portanto, o bom senso e a prática de lógica de programação é que indicarão qual a solução mais adequada, que com menos esforço e maior objetividade produzirá o resultado almejado.

A solução apresentada no **Algoritmo 1.3** é aparentemente adequada, porém não prevê a possibilidade de a lâmpada nova não funcionar e, portanto, não atingir o objetivo nessa situação específica. Podemos fazer um refinamento, uma melhoria no algoritmo, de tal modo que se troque a lâmpada diversas vezes, se necessário, até que funcione. Uma solução seria:

ALGORITMO 1.4 Troca de lâmpada com teste e repetição indefinida

- acionar o interruptor;
- se a lâmpada não acender, então
 - pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;

(Continua)

- acionar o interruptor;
- subir na escada;
- retirar a lâmpada queimada;
- colocar a lâmpada nova;
- se a lâmpada não acender, então
 - retirar a lâmpada queimada;
 - colocar outra lâmpada nova;
 - se a lâmpada não acender, então
 - retirar a lâmpada queimada;
 - colocar outra lâmpada nova;
 - se a lâmpada não acender, então
 - retirar a lâmpada queimada;
 - colocar outra lâmpada nova;
-
-
-

Até quando?

Notamos que o **Algoritmo 1.4** não está terminado, falta especificar até quando será feito o teste da lâmpada. As ações cessarão quando conseguirmos colocar uma lâmpada que acenda; caso contrário, ficaremos testando indefinidamente (note que o interruptor continua acionado!). Essa solução está mais próxima do objetivo, pois garante que a lâmpada acenda novamente, ou melhor, que seja trocada com êxito, porém, temos o problema de não saber o número exato de testes das lâmpadas.

Observemos que o teste da lâmpada nova é efetuado por um mesmo conjunto de ações:

- se a lâmpada não acender, então
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova.

Portanto, em vez de reescrevermos várias vezes esse conjunto de ações, podemos alterar o fluxo seqüencial de execução de forma que, após executada a ação colocar outra lâmpada nova, voltemos a executar o teste se a lâmpada não acender, fazendo com que essas ações sejam executadas o número de vezes necessário sem termos de reescrevê-las.

Precisamos, então, expressar essa repetição da ação sem repetir o texto que representa a ação, assim como determinar um limite para tal repetição, com o objetivo de garantir uma **condição de parada**, ou seja, que seja cessada a atividade de testar a lâmpada nova quando ela já estiver acesa. Uma solução seria:

- enquanto a lâmpada não acender, faça
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova.

A condição lâmpada não acender permaneceu e estabelecemos um fluxo repetitivo que será finalizado assim que a condição de parada for falsa, ou seja, assim que a lâmpada acender. Percebemos que o número de repetições é **indefinido**, porém é **finito**, e que depende

apenas da condição estabelecida, o que leva a repetir as ações até alcançar o objetivo: trocar a lâmpada queimada por uma que funcione. O novo algoritmo ficaria:

ALGORITMO 1.5 Troca de lâmpada com teste e condição de parada

- acionar o interruptor;
 - se a lâmpada não acender, então
 - pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;
 - acionar o interruptor;
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova;
 - enquanto a lâmpada não acender, faça
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova;
-

Até agora estamos efetuando a troca de uma única lâmpada, na verdade estamos testando um soquete (acionado por um interruptor), trocando tantas lâmpadas quantas forem necessárias para assegurar que o conjunto funcione. O que faríamos se tivéssemos mais soquetes a testar, por exemplo, dez soquetes?

A solução aparentemente mais óbvia seria repetir o algoritmo de uma única lâmpada para os dez soquetes existentes, ficando algo como:

ALGORITMO 1.6 Troca de lâmpada com teste para 10 soquetes

- acionar o interruptor do **primeiro** soquete;
- se a lâmpada não acender, então
 - pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;
 - acionar o interruptor;
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova;
 - enquanto a lâmpada não acender, faça
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova;
- acionar o interruptor do **segundo** soquete;
- se a lâmpada não acender, então
 - pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - .
 - .
 - .
- acionar o interruptor do **terceiro** soquete;
- se a lâmpada não acender, então

(Continua)

- -
 -
 - acionar o interruptor do **quarto** soquete;
 -
 -
 -
 - acionar o interruptor do **décimo** soquete;
 -
 -
 -
-

Observamos que o **Algoritmo 1.6** é apenas um conjunto de dez repetições do **Algoritmo 1.5**, uma vez para cada soquete, havendo a repetição de um mesmo conjunto de ações por um número definido de vezes: dez. Como o conjunto de ações que foram repetidas é exatamente igual, poderíamos alterar o fluxo seqüencial de execução de modo a fazer com que ele voltasse a executar o conjunto de ações relativas a um único soquete (**Algoritmo 1.5**) tantas vezes quantas fossem desejadas. Uma solução para dez soquetes seria:

ALGORITMO 1.7 Troca de lâmpada com teste para 10 soquetes com repetição

- ir até o interruptor do **primeiro** soquete;
 - enquanto a quantidade de soquetes testados for menor que dez, faça
 - acionar o interruptor;
 - se a lâmpada não acender, então
 - pegar uma escada;
 - posicionar a escada embaixo da lâmpada;
 - buscar uma lâmpada nova;
 - acionar o interruptor;
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova;
 - enquanto a lâmpada não acender, faça
 - retirar a lâmpada queimada;
 - colocar uma lâmpada nova;
 - ir até o interruptor do **próximo** soquete;
-

Quando a condição quantidade de soquetes testados for menor que dez for verdadeira, as ações responsáveis pela troca ou não de um único soquete serão executadas. Caso a condição de parada seja falsa, ou seja, todos os dez soquetes já tiverem sido testados, nada mais será executado.

Todo o exemplo foi desenvolvido a partir do problema de descrevermos os passos necessários para efetuar a troca de uma lâmpada, ou seja, construir um **algoritmo** para esse fim.

Inicialmente, tínhamos um pequeno conjunto de ações que deveriam ser executadas, todas passo a passo, uma após a outra, compondo uma ordem seqüencial de execução, a estrutura **seqüencial**.

Notamos que nem sempre todas as ações previstas deveriam ser executadas. Tal circunstância sugeriu que um determinado conjunto de ações fosse evitado, selecionando conforme o resultado de uma determinada condição. Construímos, assim, uma **estrutura seletiva** através de um **teste condicional** que permitia ou não que o fluxo de execução passasse por um determinado conjunto de ações.

Quando deparamos com a inviabilidade da aplicação da estrutura de seleção para a verificação do êxito na troca da lâmpada, precisamos repetir um mesmo trecho do algoritmo, o que foi realizado alterando-se o fluxo de execução de modo que ele passasse pelo mesmo trecho diversas vezes, enquanto a condição não fosse satisfeita; agimos de forma semelhante na situação de trocar dez lâmpadas, construindo uma **estrutura de repetição**.

Devemos ressaltar que qualquer pessoa, fundamentada na própria experiência, seria capaz de resolver o problema na prática, envolvendo inclusive circunstâncias inusitadas que pudessem surgir. Contudo, um programa de computador tradicional não tem conhecimento prévio nem adquire experiências, o que implica que devemos determinar em detalhes todas as ações que ele deve executar, prevendo todos os obstáculos e a forma de transpô-los, isto é, descrever uma seqüência finita de passos que garantam a solução do problema. Tal atividade é realizada pelos **programadores**, que podemos chamar de **construtores de algoritmos**.

DE QUE MANEIRA REPRESENTAREMOS O ALGORITMO?

Convém enfatizar mais uma vez que um algoritmo é uma linha de raciocínio, que pode ser descrito de diversas maneiras, de forma gráfica ou textual.

Os algoritmos representados até o momento estavam em forma textual, usando português coloquial.

As formas gráficas são mais puras por serem mais fiéis ao raciocínio original, substituindo um grande número de palavras por convenções de desenhos. Para fins de ilustração mostraremos como ficaria o **Algoritmo 1.7** representado graficamente em um fluxograma tradicional (**Algoritmo 1.8**) e em um Chapin (**Algoritmo 1.9**).

Cada uma dessas técnicas tem suas vantagens e desvantagens particulares. Porém, podemos perceber que ambas permitem um nível grande de clareza quanto ao fluxo de execução. Contudo, deve ser menos fácil entender essas representações do que a do **Algoritmo 1.7** em sua forma textual. Isso ocorre porque é necessário conhecer as convenções gráficas de cada uma dessas técnicas, que apesar de simples não são naturais, pois estamos mais condicionados a nos expressar por palavras.

Outra desvantagem é que sempre se mostra mais trabalhoso fazer um desenho do que escrever um texto, mesmo considerando o auxílio de réguas e moldes. A problemática é ainda maior quando é necessário fazer alguma alteração ou correção no desenho. Esses fatores podem desencorajar o uso de representações gráficas e, algumas vezes, erroneamente, a própria construção de algoritmos.

Assim, justificamos a opção pelos métodos textuais, que, apesar de menos puros, são mais naturais e fáceis de usar.

1

O QUE É PROGRAMAÇÃO?

1.1 PROGRAMAÇÃO E O RACIOCÍNIO LÓGICO

Quando nos deparamos com o desafio de programar, imaginamos que essa é uma atividade complexa e nunca, antes, executada por nós. No entanto, para começar a falar em programação, podemos mencionar uma atividade que executamos todos os dias, a todo momento em nossas vidas, que é o raciocínio lógico.

O raciocínio lógico é executado constantemente em nossas vidas, pois em simples ações, como tarefas do dia a dia, utilizamos o raciocínio. Um exemplo prático disso seria o raciocínio que você desenvolve todos os dias para realizar a tarefa de ir à escola; ou, ainda, o que você desenvolve para realizar a soma total de suas compras no supermercado, e/ou do valor máximo que você tem disponível para gastar nessa compra.

Portanto, o raciocínio lógico está na tarefa de fazer um bolo, como misturar os ingredientes, definir em quanto tempo se deve colocá-lo no fogo, e assim por diante.

Então, com o objetivo de executar uma tarefa específica, você pensa em pequenos passos sequenciados que o levarão ao objetivo final. É assim que você trabalha mentalmente na resolução de pequenos problemas do dia a dia. É dessa forma também que você trabalha mentalmente para solucionar um problema de matemática, por exemplo.

Como fazemos isso? Imagine o seguinte problema de matemática:

Utilizando a propriedade distributiva, efetue a operação sem o auxílio da calculadora: $(+5) \times (-3-4)$.

Lembrando que a propriedade distributiva é utilizada quando um número está multiplicando uma adição ou subtração, basta multiplicar em separado cada termo, e somar ou subtrair o resultado.

Então, tente descrever o seu raciocínio em pequenos passos na hora de realizar essa tarefa.

Vamos lá! Esse foi um raciocínio lógico utilizado para a execução do problema:

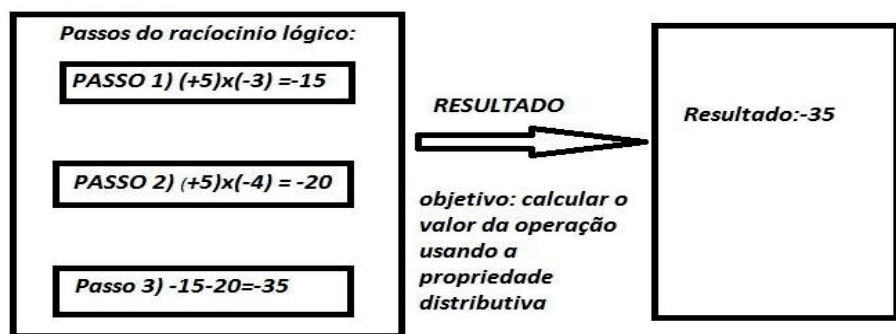
Passo 1) $(+5) \times (-3) = -15$

Passo 2) $(+5) \times (-4) = -20$

Passo 3) $-15 - 20 = -35$

Acima, observamos que para solucionar o problema, devemos decompô-lo em três passos diferentes. Na Figura 1, abaixo, você vê um esquema com a sequência de passos do raciocínio lógico, o objetivo desse raciocínio e o resultado esperado.

Figura 1 – Sequência de passos de um raciocínio lógico



Fonte: elaborada pela autora (2021).

Na Figura 1, você consegue visualizar o raciocínio lógico decomposto em passos, com o propósito de calcular o valor da operação $(+5) \times (-3-4)$ usando a propriedade distributiva. Verifique que após todos os passos serem executados, obtém-se o resultado do raciocínio. No exemplo dado, o valor -35.

Assim como o problema de matemática apresentado acima, que foi decomposto em três pequenos passos para ser solucionado, você está o tempo todo resolvendo tarefas no seu dia a dia, pensando sobre a melhor maneira de solucioná-las e, possivelmente, decompondo essas tarefas em passos. Isso é raciocínio lógico. E você utiliza-o o tempo todo.

Então, agora que falamos um pouco sobre raciocínio lógico, podemos concluir que a atividade de programação é uma atividade essencialmente de raciocínio lógico. Olha que legal, você já está acostumado a usar o raciocínio lógico automaticamente em quase todas as tarefas do seu dia a dia!

1.2 A LINGUAGEM DE PROGRAMAÇÃO DE PYTHON

Imagine agora que você pedirá para o computador executar a mesma sequência de passos realizada no exemplo da Figura 1. Então, como explicar isso para o computador? Agora, o problema é de comunicação. Você consegue traduzir para o seu cérebro o problema e efetuar essa tarefa sozinho, mas como explicar isso para que o computador execute essa tarefa? Para realizar essa comunicação, você precisa de uma **LINGUAGEM DE PROGRAMAÇÃO**, que determinará a forma como você deve descrever os passos que o computador executará, bem como o resultado que será apresentado.

Existem muitas linguagens de programação no mercado, cada uma com suas características específicas e regras para determinar essa comunicação. É como definir se você falará português, inglês ou espanhol – depende muito de seu objetivo, da mesma maneira, para cada uma das linguagens, existem formas diferentes de definir os passos a serem executados.

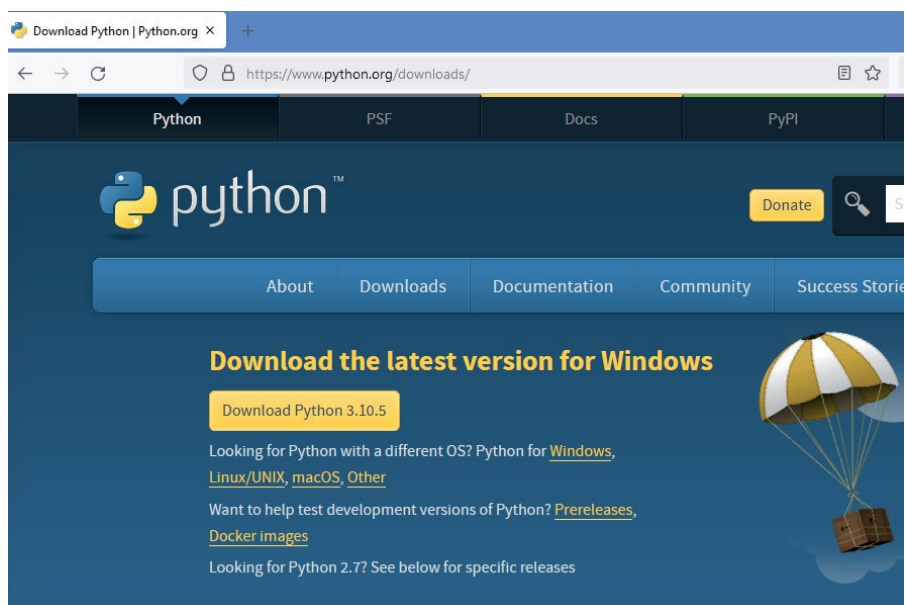
Essa “forma” de definir os passos nós chamamos também de **SINTAXE**, que é a maneira como se escreve. Por exemplo: em português, você diz *Bom dia*; em inglês, *Good morning*, e

assim por diante. Dessa forma, como a língua que você fala possui uma sintaxe específica, a linguagem de computador que você usa também possui sua própria sintaxe.

Logo, para que você faça um **PROGRAMA DE COMPUTADOR**, você precisará usar uma **LINGUAGEM DE PROGRAMAÇÃO**, e para usar essa linguagem de programação, será necessário conhecer a sintaxe dessa linguagem.

A linguagem que utilizaremos neste livro é a linguagem de programação Python versão 3.7. Você pode baixar gratuitamente a linguagem Python no endereço <https://www.python.org/downloads/>. No momento de criação desse material, a última versão disponível no site era a 3.9.5, como visto na Figura 2. Você pode baixar esta versão sem problemas. Qualquer versão acima da 3 será compatível com as atividades realizadas neste livro.

Figura 2 – Download da linguagem Python no site oficial

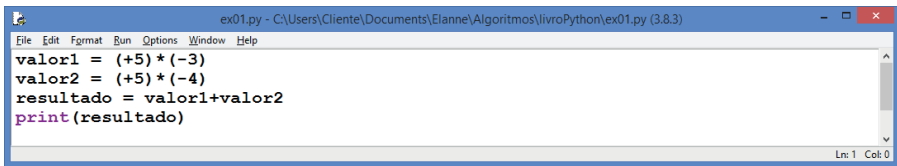


Fonte: (PYTHON, 2021).

Então, agora sabendo de que precisamos de uma **LINGUAGEM DE PROGRAMAÇÃO**, que usaremos a linguagem **PYTHON** e que o Python possui uma **SINTAXE** específica, como fazer um programa em Python que peça para o computador executar a operação $(+5) \times (-3 - 4)$, usando a propriedade distributiva?

A Figura 3 ilustra os três passos para realizar a operação, traduzidos para a sintaxe Python.

Figura 3 – Programa escrito em Python com o objetivo de calcular a operação $(+5) \times (-3 - 4)$ usando a propriedade distributiva



Fonte: elaborada pela autora (2021).

Você não conseguiu entender totalmente os passos realizados na Figura 3, acima? Tudo bem! Provavelmente, você não está entendendo porque ainda não conhece a **SINTAXE** da linguagem Python. Tentaremos agora traduzir cada um dos passos realizados no programa:

Passo 1) $valor1 = (+5) \times (-3)$

O cálculo será realizado e o resultado (o valor -15) será guardado dentro de “valor1”.

Considere “valor1” como uma caixa onde você pode armazenar valores. Na linguagem, chamamos esse recurso de **VARIÁVEL**. Uma variável pode receber diferentes valores durante a execução de um programa.

Passo 2) $valor2 = (+5) \times (-4)$

O cálculo será realizado e o resultado (o valor -20) será guardado dentro da **VARIÁVEL** “valor2”.

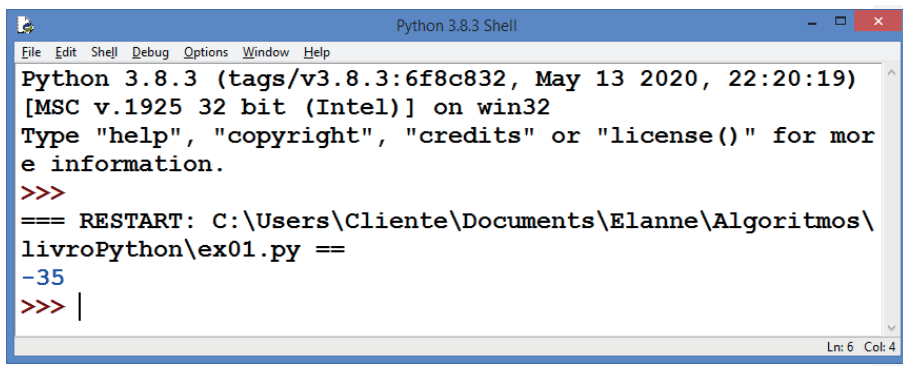
Passo 3) $resultado = valor1 + valor2$

A soma dos valores guardados em “valor1” e “valor2” é realizada, e o resultado é guardado na **VARIÁVEL** “resultado”.

Passo 4) print(resultado)

Reparou que o passo 4 não existia na primeira solução? Os passos anteriores realizaram as operações e usaram **VARIÁVEIS** (“valor1”, “valor2” e “resultado”) para guardar os resultados. Mas em nenhum momento, esse resultado foi apresentado na tela do computador. A **FUNÇÃO** “print()” é usada, em Python, para imprimir um valor, este valor (chamado de argumento) é definido entre os parênteses que acompanham o nome da função. Logo, no exemplo, quando o programa for executado, print(resultado) mostrará na tela do computador o valor guardado na variável “resultado”, conforme mostra a Figura 4.

Figura 4 – Ao executar o passo “print(resultado)”, o valor da variável “resultado” é apresentado na tela do computador



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19)
[MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
==== RESTART: C:\Users\Cliente\Documents\Elanne\Algoritmos\
livroPython\ex01.py ====
-35
>>> |
```

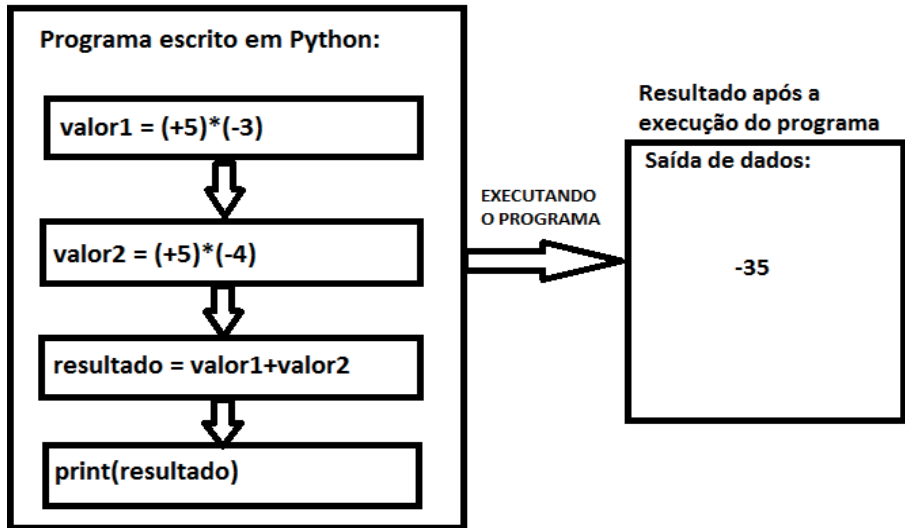
Fonte: elaborada pela autora (2021).

Além da **FUNÇÃO** “*print()*”, existem várias outras funções definidas na linguagem Python. Ainda sobre o exemplo da Figura 4, até agora, vimos que para realizar a operação $(+5) \times (-3 - 4)$ usando a linguagem Python, usamos **VARIÁVEIS** (“valor1”, “valor2” e “resultado”) e uma **FUNÇÃO** (“*print()*”). Além disso, no Python, percorremos quatro passos para alcançar o objetivo desejado.

A esses passos realizados você pode chamar de **COMANDOS**. Os **COMANDOS** são executados, no exemplo dado, linha a linha, um após o outro, ou seja, sequencialmente. Para que os **COMANDOS** sejam executados corretamente, vimos que é preciso conhecer a **SINTAXE** da linguagem Python, isto é, a forma correta de escrever tais comandos.

Na linha de comando “`print(resultado)`”, é impresso o valor da variável “resultado” na tela do computador. Logo, podemos dizer que esse é um **COMANDO DE SAÍDA DE DADOS**. Veja, na Figura 5, o novo esquema (agora usando a linguagem Python), similar ao da Figura 1.

Figura 5 – Sequência de comandos e sua execução em Python



Fonte: elaborada pela autora (2021).

1.3 O QUE NÓS VIMOS NESTE CAPÍTULO?

Conversamos aqui sobre como programar tem muito a ver com uma atividade básica do seu dia a dia: o uso do raciocínio lógico. Falamos que para programar, precisamos usar

o raciocínio lógico na resolução de problemas, assim como de uma linguagem de programação. A linguagem de programação que utilizaremos neste livro é a linguagem Python.

Falamos rapidamente também sobre alguns conceitos usados em programação, tais como sintaxe, variáveis, a função “*print()*” do Python, comandos e comandos de saída de dados. Muita coisa? Não precisa ficar preocupado! Todos esses conceitos serão discutidos de forma mais detalhada nos próximos capítulos. Vamos continuar?

2

A LINGUAGEM PYTHON

2.1 POR QUE USAR A LINGUAGEM PYTHON?

Python é uma linguagem muito fácil de aprender: sua sintaxe é muito simples e permite ao programador concentrar-se mais na solução do problema e menos com regras de sintaxe de linguagem. Esse é um dos principais motivos para Python ser uma linguagem tão adequada ao uso no estudo de algoritmos e linguagem de programação, tendo em vista que estamos começando o desafio de aprender a programar.

Por sinal, Python é um *software* livre, ou seja, pode ser usado gratuitamente. Além disso, pode ser usado em qualquer arquitetura de computadores ou sistema operacional como, por exemplo, Windows, Linux ou Mac.

2.2 CARACTERÍSTICAS DE LINGUAGENS DE PROGRAMAÇÃO

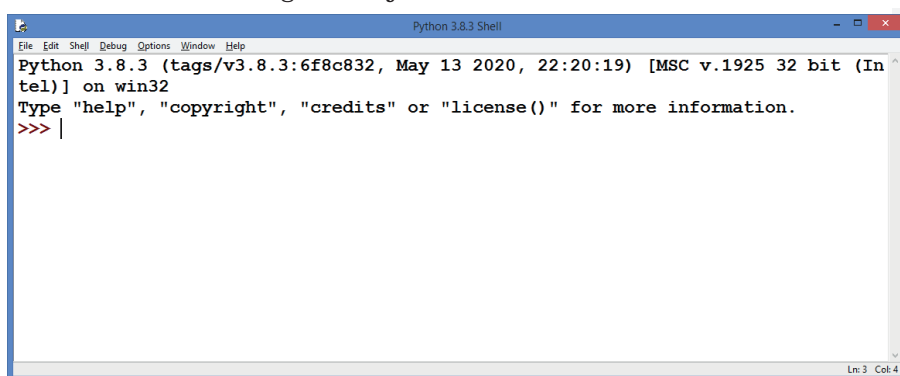
2.2.1 O interpretador Python

Para começarmos a desenvolver programas utilizando a linguagem Python, precisamos instalar o interpretador Python no computador, que é o programa que permite executar todos os comandos da linguagem. Para desenvolver os exemplos apresentados neste livro, você pode instalar a versão 3.8 ou superior. Você pode baixar o interpretador Python no site oficial <https://www.python.org/downloads/>.

Quando você abrir o programa interpretador, verá a janela inicial do IDLE. A sigla IDLE significa *Integrated Development and Learning Environment* (Ambiente Integrado de Desenvolvimento e Aprendizagem). O IDLE é uma interface gráfica utilizada

pelo interpretador como um ambiente de desenvolvimento integrado para a execução de programas e comandos escritos em Python. A aplicação vem junto com a maioria das instalações do Python e está presente na distribuição oficial, desde a versão 2.3. Veja a Figura 6.

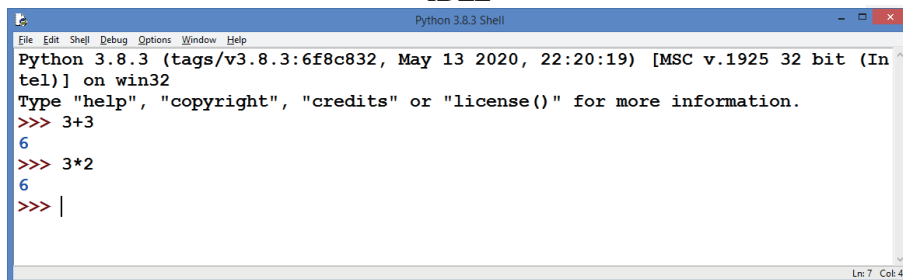
Figura 6 – Janela inicial do IDLE



Fonte: elaborada pela autora (2021).

A janela inicial, na Figura 6, é a janela do interpretador. Observe, ainda, que o cursor está posicionado dentro da janela, e a linha de comando é iniciada pela sequência “>>>” (também chamada de *prompt* de comando). Você pode executar os comandos Python linha a linha na janela do interpretador, conforme mostra a Figura 7.

Figura 7 – Executando comandos na linha de comando do interpretador IDLE

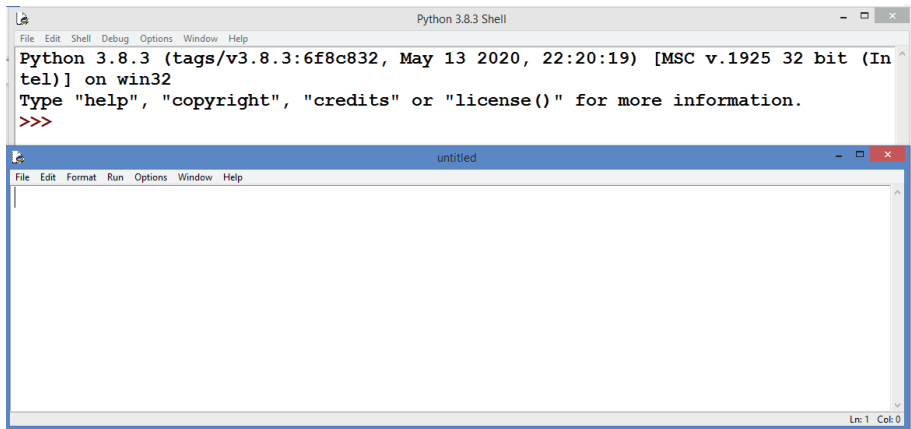


Fonte: elaborada pela autora (2021).

Na Figura 7, o interpretador está sendo usado para realizar as operações matemáticas “3+3” e, logo em seguida, “3*2”.

Em Python, você pode utilizar qualquer editor de textos disponível, mas o IDLE também possui um editor de textos próprio. Com a janela do IDLE aberta, clique no menu FILE, selecione a opção NEW FILE. Uma nova janela será aberta. Nela, você pode escrever seu programa Python. Veja a Figura 8.

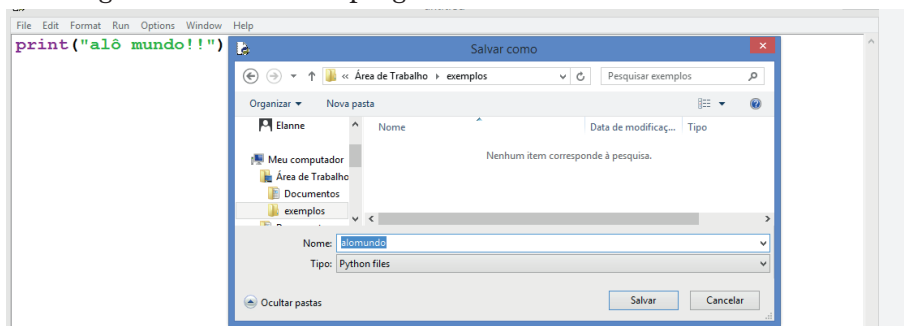
Figura 8 – Usando o editor de textos do IDLE



Fonte: elaborada pela autora (2021).

Na Figura 8, podemos visualizar duas janelas: uma é a do interpretador (a mesma da Figura 6); a outra é a do editor de textos. Para salvar seu programa escrito no editor, você pode clicar no menu FILE, selecionar a opção SAVE ou SAVE AS (salvar e modificar o nome do arquivo). O programa escrito em Python será salvo com a extensão “.py” automaticamente. Veja na Figura 9.

Figura 9 – Salvando o programa como o nome de “alomundo”

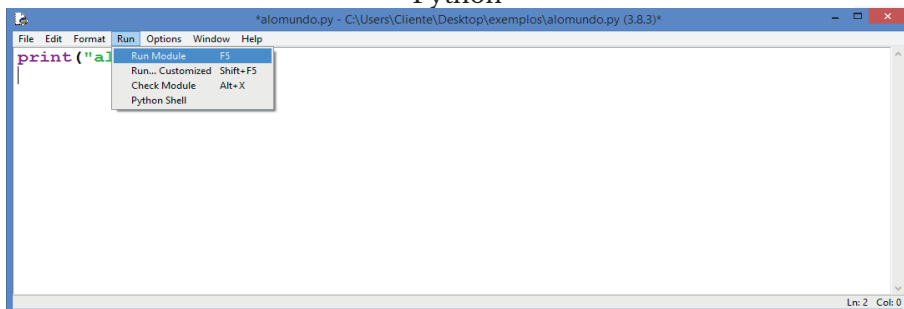


Fonte: elaborada pela autora (2021).

Verifique, na Figura 9, que na parte inferior da janela SALVAR COMO, existe a definição do nome do arquivo e, logo em seguida, o tipo do arquivo. No exemplo dado, o nome do arquivo foi definido como “alomundo”.

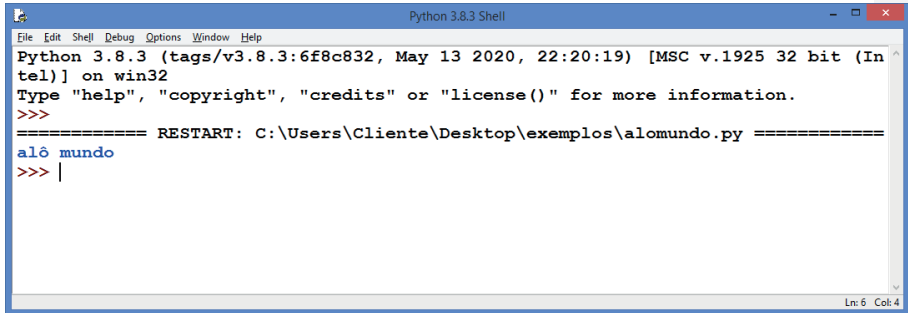
Ainda na Figura 9, observe que o tipo de arquivo é definido automaticamente como um arquivo do tipo Python (na janela descrito como “*Python files*”). Assim, o programa escrito em Python será salvo com a extensão “.py”. No exemplo visto, o nome do arquivo, juntamente com a sua extensão, será “alomundo.py”. Para executar o programa “alomundo.py” no IDLE, você pode clicar em menu RUN, opção RUN MODULE. Veja a Figura 10. A execução do programa será apresentada na janela inicial (janela do interpretador), conforme mostra a Figura 11.

Figura 10– Opção de menu RUN usada para executar um programa Python



Fonte: elaborada pela autora (2021).

Figura 11 – Execução do programa “alomundo.py” na janela do interpretador IDLE

The image shows a screenshot of the Python 3.8.3 Shell window. The title bar reads "Python 3.8.3 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area displays the following content:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Cliente\Desktop\exemplos\alomundo.py =====
alô mundo
>>> |
```

The status bar at the bottom right indicates "Ln: 6 Col: 4".

Fonte: elaborada pela autora (2021).

2.2.2 Python é uma linguagem interpretada

Quando a sintaxe da linguagem é facilmente entendida por nós, dizemos que ela é de alto nível. Isso posto, a linguagem de programação simplifica o nosso trabalho, possibilitando que nós, usuários programadores, possamos escrever um código-fonte próximo da nossa própria linguagem e/ou entendimento.

Ocorre que o computador não consegue entender diretamente o código-fonte escrito na linguagem de alto nível, daí porque é necessária a tradução desse código para que o computador possa executá-lo. Nesse processo, o código deve ser transformado em um código mais próximo da linguagem de máquina (imagine que o computador só entende sequências de “0” e “1”, chamadas de código binário), ou seja, um código que realmente o computador seja capaz de entender e executar. Para realizar essa tradução, as linguagens de programação usam diferentes estratégias.

O programa que você escreve (linguagem de alto nível), segundo a sintaxe da linguagem, será lido e executado pelo computador. Para que esse programa seja compreendido pela máquina, antes, ele deve ser traduzido para uma forma que o computador consiga entender.

Você pode imaginar esse processo como algo semelhante ao que acontece na vida real: imagine que você precisa se comunicar com um nativo americano que acabou de chegar à sua escola, mas você não entende uma única palavra em inglês e ele, por sua vez, não entende nada em português. Você pedirá ajuda a um amigo que domina o inglês e é capaz de traduzir sua mensagem. Assim, seu amigo será seu tradutor, e o colega americano conseguirá entender sua mensagem depois de traduzida. Veja a Figura 12.

Figura 12 – Exemplo de um processo de tradução na vida real



Fonte: elaborada pela autora (2021).

Cada linguagem de programação usa um processo de tradução. De acordo com o processo de tradução que uma linguagem usa, ela pode ser classificada como compilada ou interpretada.

Em uma linguagem compilada, o texto que você escreve (chamado também de código-fonte do programa) é lido por um tradutor, nomeado de compilador, ou seja, um programa que lê o código-fonte e, a partir dele, cria outro arquivo, um arquivo binário (igualmente chamado de arquivo executável). Esse arquivo é a tradução do que foi escrito no código-fonte, e uma vez realizada a tradução para o arquivo executável, ele pode ser executado diretamente pelo seu computador. Veja a Figura 13.

Figura 13 – Processo de compilação



Fonte: elaborada pela autora (2021).

No processo de compilação da Figura 13, para criar o arquivo com o código executável, o programa compilador executa os seguintes passos:

Cria um código-objeto: converte o código-fonte em linguagem de máquina (linguagem entendida pelo *hardware* do computador). Esse código só é criado quando não há erros no código-fonte. A extensão do arquivo que contém o código-fonte é .OBJ.

Logo em seguida, um ligador ou *linkeditor* “junta” o código-fonte com outros arquivos necessários presentes na linguagem (chamados de “bibliotecas”). As extensões dos arquivos de bibliotecas, geralmente, são .DLL ou .LIB. Dessa união, é criado o arquivo executável. O código executável pode ser, então, executado pelo sistema operacional do computador.

Contudo, em uma linguagem interpretada, não é criado um arquivo executável. Nesse caso, o arquivo de código-fonte é executado por meio do uso de outro programa, chamado de *interpretador*. Assim, toda vez que o programa for usado, o interpretador entrará em ação e o traduzirá diretamente, durante a sua execução. Veja a Figura 14.

Figura 14 – Processo de interpretação



Fonte: elaborada pela autora (2021).

O *interpretador* lê o programa escrito em código-fonte (linguagem de alto nível) e executa-o. Assim, o *interpretador* processa o programa um pouco de cada vez, alternadamente, lendo as linhas de comando e realizando as operações, linha a linha.