MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE ABERTA DO BRASIL

GRADUAÇÃO EM
TECNOLOGIA EDUCACIONAL
LICENCIATURA

Introdução a Algoritmos

Raoni Florentino da Silva Teixeira

2019

Secretaria de Tecnologia Educacional
Universidade Federal de Mato Grosso

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE ABERTA DO BRASIL

GRADUAÇÃO EM
TECNOLOGIA EDUCACIONAL
LICENCIATURA

Introdução a Algoritmos

Raoni Florentino da Silva Teixeira

2019

Secretaria de Tecnologia Educacional
Universidade Federal de Mato Grosso

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE ABERTA DO BRASIL

Introdução a Algoritmos

Raoni Florentino da Silva Teixeira

2019

Secretaria de Tecnologia Educacional
Universidade Federal de Mato Grosso

Ministro da Educação
Ricardo Vélez Rodríguez

Presidente da CAPES
Anderson Ribeiro Correia

Diretor Nacional da UAB
Carlos Cezar Mordenel Lenuzza

Reitora UFMT
Myrian Thereza de Moura Serra

Vice-Reitor
Evandro Aparecido Soares da Silva

Pró-reitor Administrativo
Bruno Cesar Souza Moraes

Pró-reitora de Planejamento
Tereza Mertens Aguiar Veloso

Pró-reitor de Cultura, Extensão e Vivência
Fernando Tadeu de Miranda Borges

Pró-reitora de Ensino e Graduação
Lisiane Pereira de Jesus

Pró-reitora de Pesquisa
Patrícia Silva Osório

Secretário de Tecnologia Educacional
Alexandre Martins dos Anjos

Coordenador da UAB/UFMT
Alexandre Martins dos Anjos

Coord. do Curso de Licenciatura em Tecnologia Educacional
Silas Borges Monteiro

SUMÁRIO

UNIDADE 1

BIBLIOTECA DE ÍCONES

Reflexão – Sinaliza que uma atividade reflexiva será desenvolvida. Para isso, sugerimos que leia a questão feita e anote o que você pensa a respeito da abordagem, antes de qualquer assimilação de novos conhecimentos. Você pode convidar seus colegas para debates, questionar a equipe de tutoria e docentes (usando a ferramenta *mensagem* ou *fórum*). No final do processo, faça uma síntese das ideias resultantes das novas abordagens que você assimilou e/ou construiu, de forma a se preparar para responder perguntas ou questionamentos sobre o assunto refletido.

Pesquisa e Exercícios – Indica uma atividade de pesquisa ou exercício propriamente dito, elaborada com a finalidade de conferir a sua compreensão sobre um determinado contexto informativo.

Saiba mais – Sugere o desenvolvimento de estudo complementar. No ambiente virtual do curso, na área de "Saiba Mais", é possível localizar materiais auxiliares, como textos e vídeos, que têm por premissa apoiar o seu processo de compreensão dos conteúdos estudados, auxiliando-o na construção da aprendizagem.

Atividades – Aponta que provavelmente você terá uma chamada no seu Ambiente Virtual de Aprendizagem para desenvolver e postar resultados de seu processo de estudo, utilizando recursos do ambiente virtual.

Vamos aos estudos?

UNIDADE 1
Algoritmos e Resolução de Problemas

Após a leitura deste capítulo, você será capaz de:

- Entender o que é um algoritmo e quais são suas características.
- Perceber a relação entre lógica, algoritmos e resolução de problemas.
- Identificar as tarefas que devem ser executadas para projetar seu próprio algoritmo.

O tópico central deste curso é a **resolução de problemas** usando um computador. Trata-se de uma tarefa importante, pois os computadores estão em toda parte, e muitas tarefas de nossa vida cotidiana são ou serão realizadas por eles.

Quais tarefas da sua vida cotidiana são realizadas por algum tipo de computador?

É curioso notar o quanto somos dependentes de uma tecnologia, e normalmente não percebemos a intensidade com que isso acontece. Avanços no processamento e na aquisição de dados (câmeras e outros sensores) permitiram que os computadores fossem aplicados em uma grande quantidade de tarefas. Este é o caso, por exemplo, das operações realizadas pelas lavadoras de roupas mais modernas. Também é o caso do mecanismo empregado no câmbio automático de um carro, que controla a troca de marchas de modo quase imperceptível para o motorista. Mais recentemente, apareceram ainda carros que estacionam e, até mesmo, dirigem de maneira automática. Todas essas tarefas são realizadas por computadores e, como você entenderá no fim desta aula, também por um algoritmo. Tal como a eletricidade, o computador é um exemplo de tecnologia que passou a compor o nosso entorno sem que percebêssemos toda a complexidade envolvida.

Algoritmo é o nome dado à sequência de ações indicando exatamente o que o computador deve fazer para realizar uma tarefa ou resolver um problema.

Muḥammad ibn Mūsā al-Khwārizmī (780-850)

O termo algoritmo tem sua origem com o matemático do século IX **Muḥammad ibn Mūsā al-Khwārizmī**, cujo sobrenome foi latinizado para **Algoritmi**. O conceito por trás do nome existe há séculos. Matemáticos gregos, por exemplo, já usavam para descrever como encontrar números primos ou o maior divisor comum entre dois números inteiros (KNUTH, 1968).

Pesquise sobre o algoritmo **Crivo de Eratóstenes**, utilizado para encontrar números primos na Grécia antiga.

CARACTERÍSTICAS DE UM ALGORITMO
• As ações ou operações executadas devem ser simples. Ações complexas devem ser subdivididas em ações menores.
• Cada ação ou operação deve ser entendida precisamente. Não há espaço para ambiguidades, dúvidas ou mal-entendidos.
• A sequência de ações deve ser finita, ou seja, o algoritmo deve, obrigatoriamente, terminar.

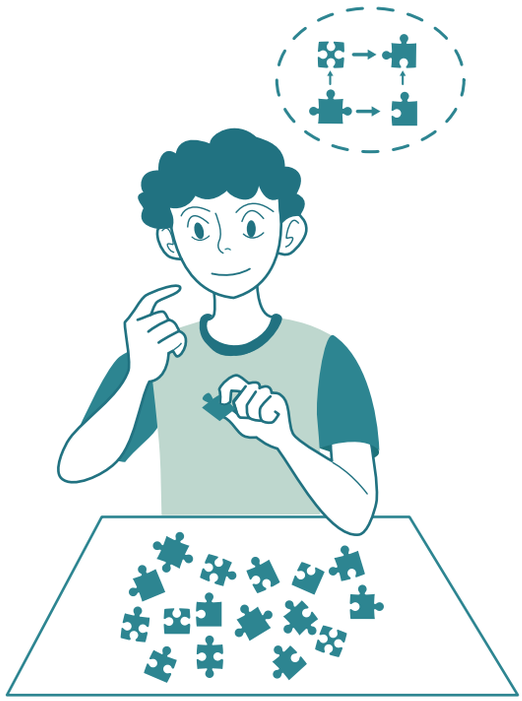
Fonte: (CORMEN, 2014, p. 7)

O objetivo deste capítulo é promover uma primeira aproximação dos cursistas aos conceitos relacionados com a resolução de problemas, ao projeto e à construção de algoritmos e programas de computador.

Ressaltamos, porém, que as noções aqui apresentadas constituem apenas os princípios básicos. Logo, convidamos todos a participar das atividades do SAIBA MAIS no Guia de Estudos, e sugerimos a leitura dos livros indicados na seção Referências para aprofundamento e detalhes não abordados.

2 Como projetar um algoritmo

Escrever um algoritmo é um processo muito parecido com montar um quebra-cabeça. Cada peça do projeto em questão é uma ação a ser executada. Em geral, as possíveis ações são conhecidas, e o desafio é descobrir uma maneira adequada de combinar as peças.

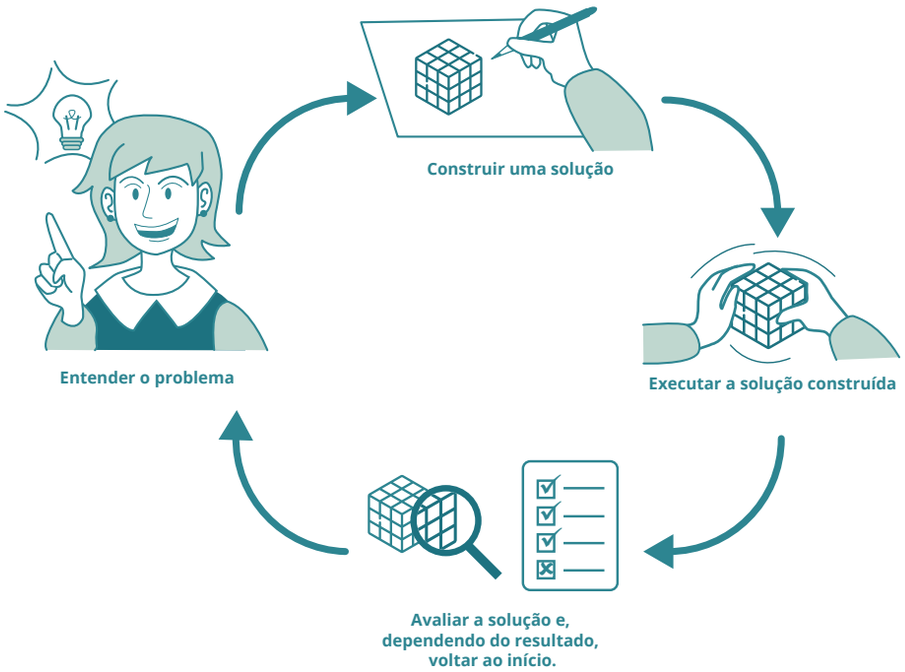


Mas afinal, como encontramos a solução para um problema? Como se dá esse complexo processo? Somos capazes de descrever o que pensamos e/ou fazemos quando encontramos a solução para um problema?

George Pólya (1887-1985)

Em várias de suas obras, o matemático e educador húngaro **George Pólya** (1887, 1985) tentou responder essas e outras perguntas relacionadas com resolução de problemas. Polya (1945) identificou e catalogou vários métodos sistemáticos de solução de problemas e descoberta de conhecimento em matemática, utilizados tanto por alunos, quanto por professores e pesquisadores.

A estratégia apresentada a seguir é uma adaptação do trabalho de Polya (1945), especificamente para o caso do projeto de algoritmos. Trata-se de uma estratégia iterativa, em que a solução vai sendo construída aos pouquinhos. A ideia é que, a cada ciclo, melhoremos nosso entendimento sobre o problema e nos aproximemos da solução desejada.



ESTRATÉGIA PARA PROJETO DE ALGORITMO
ENTENDIMENTO DO PROBLEMA
• Leia atentamente o enunciado, esclarecendo possíveis dúvidas de vocabulário e identificando os dados do problema e o resultado esperado. • Identifique quais ações podem ser executadas pelo algoritmo. • Tente responder as perguntas: Você já viu este problema antes? Você conhece um problema relacionado?
CONSTRUÇÃO DO ALGORITMO
• Responda a questão: Você consegue transformar este problema em algum outro que já foi resolvido? • Avalie se o problema ainda parece complicado e, caso necessário, o transforme-o em um problema auxiliar, eliminando alguma das variáveis envolvidas. Considere os casos mais simples primeiro. • Brinque com a solução e tente entender o que acontece quando grupos de ações são combinadas. • Procure encontrar padrões e descreva o algoritmo.

ESTRATÉGIA PARA PROJETO DE ALGORITMO
EXECUÇÃO DO ALGORITMO
• Antecipe o resultado da solução do problema. • Simule a execução do algoritmo e guarde a solução encontrada.
AVALIAÇÃO DA RESPOSTA
• Avalie a solução. A cada iteração você aprende mais sobre o problema. • Se estiver resolvendo um problema auxiliar, aumente a complexidade do problema e refaça o processo. • Inicie o processo novamente, se o resultado esperado não for alcançado.

Fonte: Adaptado de (POLYA, 1945 p. XIV-XV)

Em linhas gerais, a estratégia consiste em quatro etapas. Antes de qualquer coisa, devemos entender o problema, pois dificilmente vamos resolver o que não entendemos. Depois, tentamos encontrar conexões entre os dados do problema e a solução esperada. Neste ponto, podemos considerar problemas auxiliares que são mais simples que o problema original. O resultado desta etapa é um algoritmo que será executado passo a passo na etapa seguinte. Na quarta e última etapa, avaliamos a resposta encontrada. Se o resultado esperado não for alcançado, o processo é reiniciado.

É importante **fazer anotações** em todas as etapas da estratégia. O processo requer curiosidade, imaginação e trabalho duro. Como no começo tudo parece complicado, o ideal é adotar um procedimento semelhante ao do aprendiz de cozinheiro, que começa seguindo religiosamente o que está escrito no livro de receitas, sem entender bem o que está fazendo; e, depois de alguma prática, consulta as receitas apenas por alto e a passa a se virar por conta própria, inventando e descobrindo coisas, métodos e ingredientes.

2.1 Torre de Hanoi

Para ilustrar a aplicação da estratégia de solução proposta por Polya (1945), tomemos com exemplo o quebra-cabeça clássico chamado **Torre de Hanoi**, proposto pelo matemático francês **Édouard Lucas** (1842,1891) em 1883.

Édouard Lucas
(1842-1891)

O quebra-cabeça consiste em uma base contendo três pinos. No primeiro pino, há 3 (três) discos dispostos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O desafio é movimentar todos os discos para o último pino, usando o pino do meio para movimentos auxiliares. Apenas o disco do topo pode ser movimentado. Uma restrição importante deste problema é que um disco nunca pode ficar em cima de outro com diâmetro menor ao dele (KNUTH et al. 1988).

Como esse é o nosso primeiro problema, infelizmente, nunca vimos um problema parecido antes. O quadro a seguir apresenta algumas propriedades do problema que podem ser capturadas a partir de uma leitura atenta do enunciado aplicando a estratégia proposta no trabalho do Polya (1945).

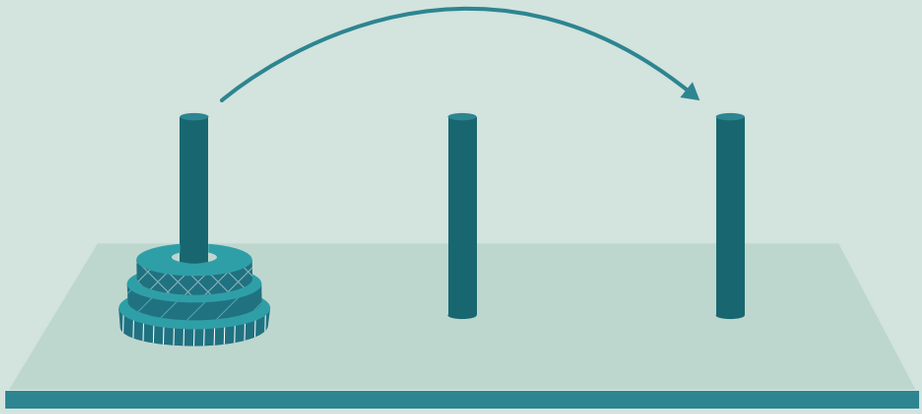
ITERAÇÃO 1
ANOTAÇÕES DA FASE DE ENTENDIMENTO DO PROBLEMA
Após a leitura atenta do enunciado percebemos que: • Devemos movimentar os três discos do primeiro para o último pino. • Só podemos movimentar um disco por vez. A única ação executada, portanto, é movimentar o disco do topo do pino x para o pino y. • Um disco nunca pode ficar em cima de outro com diâmetro menor ao dele.

ANOTAÇÕES DA FASE DE CONSTRUÇÃO DO ALGORITMO

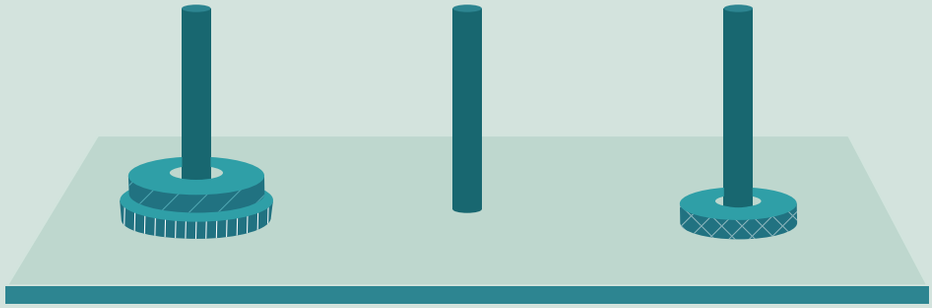
- Mesmo após a fase de entendimento, o problema ainda parece complicado e não fica evidente como movimentar os 3 (três) discos seguindo as regras apresentadas.
- Vamos ter de simplificar o problema. Suponha agora que devemos mover apenas um disco. Neste caso, a solução é relativamente fácil. Apenas uma ação deve ser executada:

Movimentar o disco do pino 1 para o pino 3.

Esta é a configuração original da torre.



Após a execução da ação, temos o seguinte resultado.

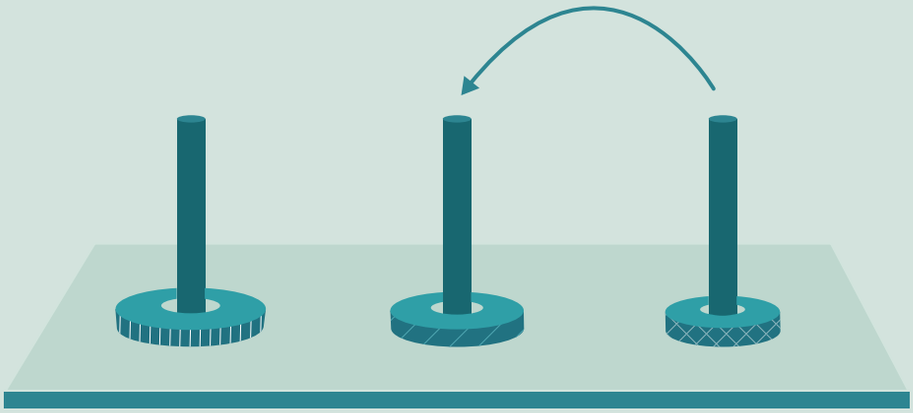


O que acontece se você movimentar o disco que está no topo do primeiro pino para o pino intermediário? E se, na sequência, você também movimentar o menor disco (que está no terceiro pino) para o pino intermediário?

A ação

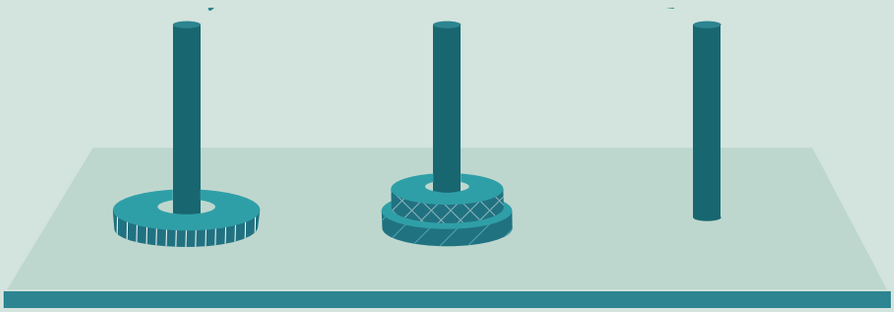
Movimentar o disco do pino 1 para o pino 2.

tem o seguinte resultado



A Figura a seguir ilustra o resultado da ação:

Movimentar o disco do pino 3 para o pino 2.



Brincando com os movimentos, descobrimos uma maneira de movimentar 2 (dois) discos para o pino intermediário. É importante notar que nenhuma regra do problema foi quebrada por esta solução.

O algoritmo encontrado é composto pelas ações:

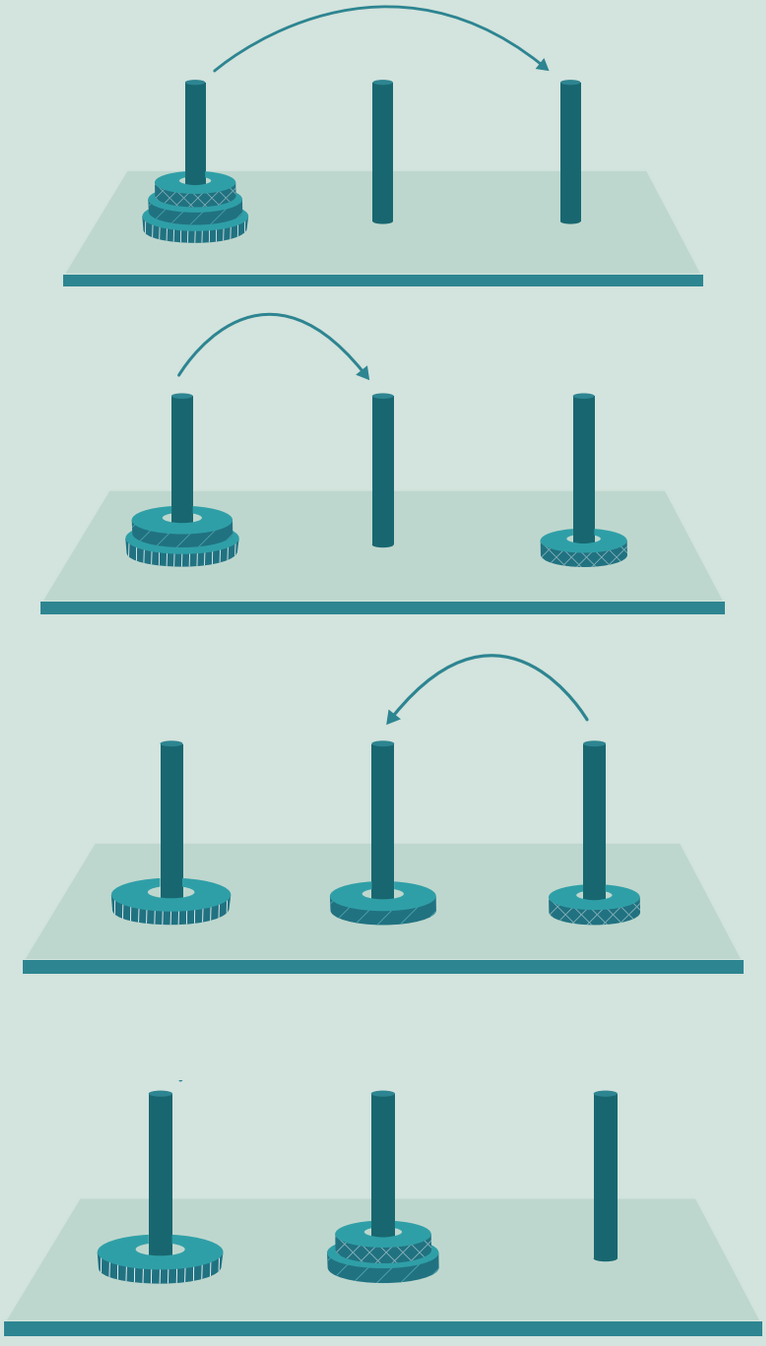
Movimentar o disco do pino 1 para o pino 3.

Movimentar o disco do pino 1 para o pino 2.

Movimentar o disco do pino 3 para o pino 2.

ANOTAÇÕES DA FASE DE EXECUÇÃO DO ALGORITMO

O resultado da execução de cada ação do algoritmo é apresentado novamente a seguir.



ANOTAÇÕES DA FASE DE EXECUÇÃO DO ALGORITMO

Alcançamos a solução do problema? O que falta para resolver o problema original?

O algoritmo encontrado movimenta os 2 (dois) discos menores. Para resolvermos o problema, precisamos movimentar também o disco com maior diâmetro.

Como fazemos movimentar o último disco? Como resolver o problema original, partindo desta solução intermediária?

Resolveremos estas questões na próxima iteração da estratégia.

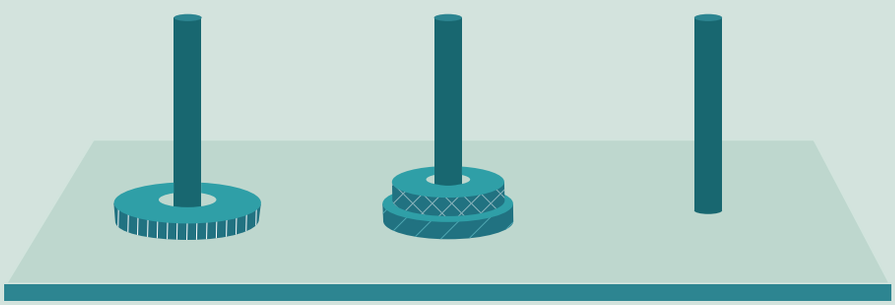
ITERAÇÃO 2

ANOTAÇÕES DA FASE DE ENTENDIMENTO DO PROBLEMA

Após a primeira iteração percebemos que:

É possível movimentar 2 (dois) discos utilizando 3 (três) movimentos.

Após estes três movimentos, chegamos à seguinte configuração de pinos:

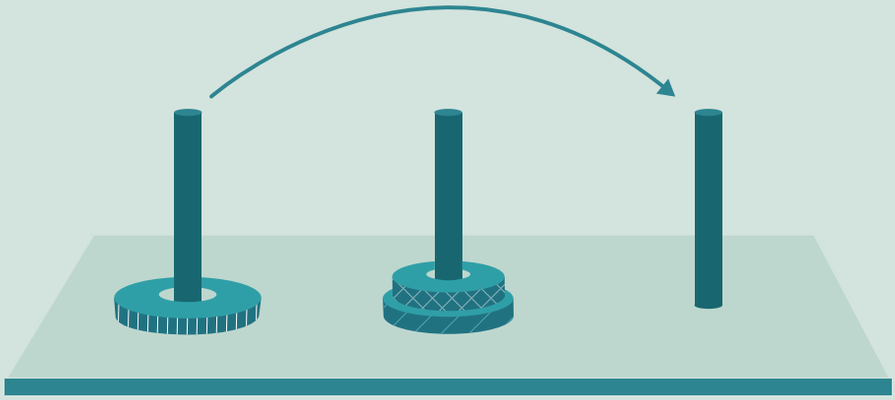


ANOTAÇÕES DA FASE DE CONSTRUÇÃO DO ALGORITMO

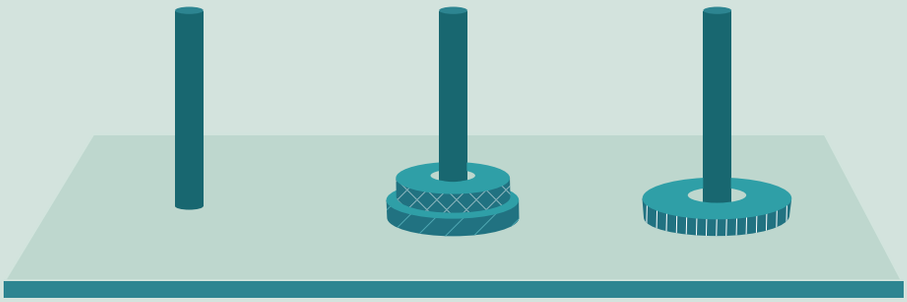
O problema agora é ligeiramente mais fácil que o problema original. Há apenas um disco no primeiro pino, e o terceiro pino está vazio.

Depois de tudo que fizemos, fica até fácil perceber que podemos mover o disco do primeiro para o último pino com uma ação:

Movimentar o disco do pino 1 para o pino 3.



Após a execução da ação, temos o seguinte resultado:.



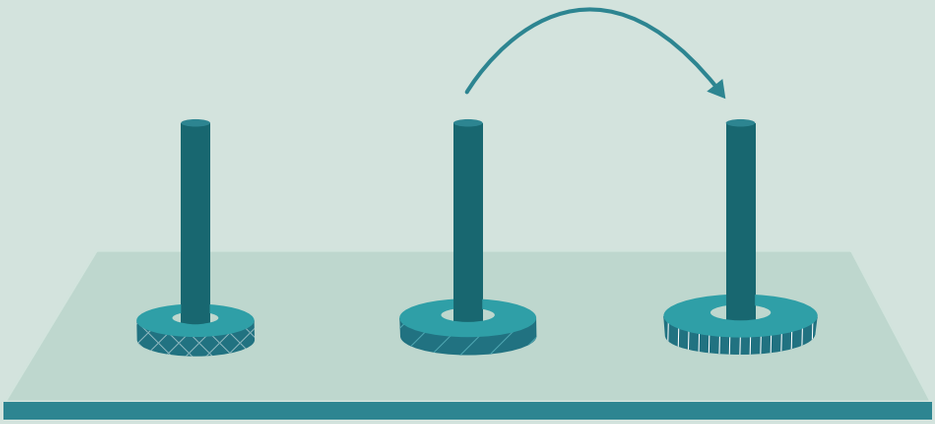
O que falta para resolver o problema? Como fazemos para mover os dois discos do pino do meio para o terceiro pino?

Novamente, vamos brincar com os movimentos.

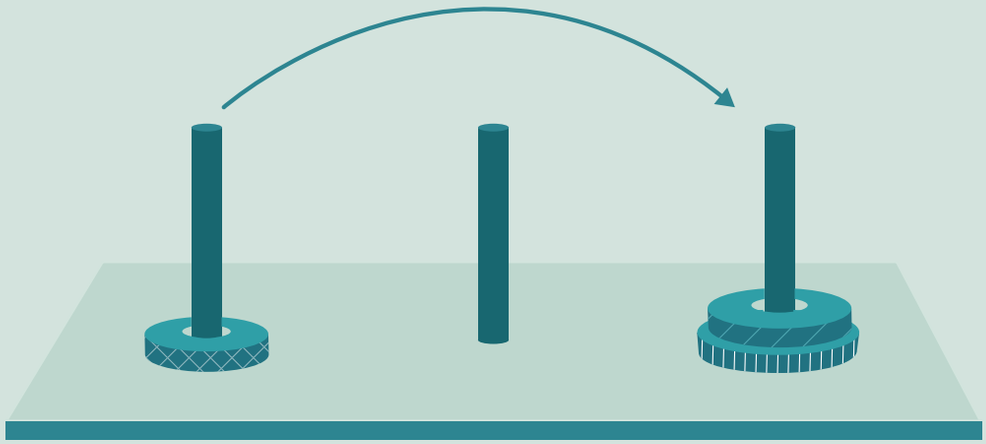
A ação

Movimentar o disco do pino 2 para o pino 1.

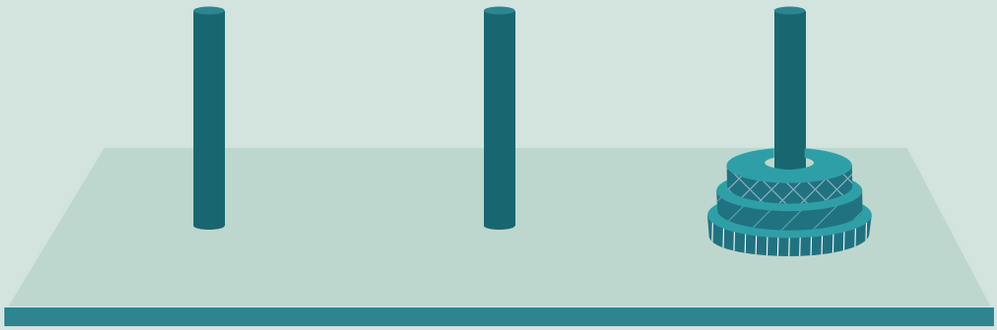
tem o seguinte resultado:.



Agora estamos quase identificando um padrão. O disco do segundo pino pode ser movimentado para o terceiro pino sem ferir nenhuma regra. Após este movimento, temos a seguinte configuração:



Agora falta apenas movimentar o disco menor do primeiro para o terceiro pino. O resultado deste movimento é ilustrado a seguir.



O algoritmo final com todas as operações é dado por:

Movimentar o disco do pino 1 para o pino 3.

Movimentar o disco do pino 1 para o pino 2.

Movimentar o disco do pino 3 para o pino 2.

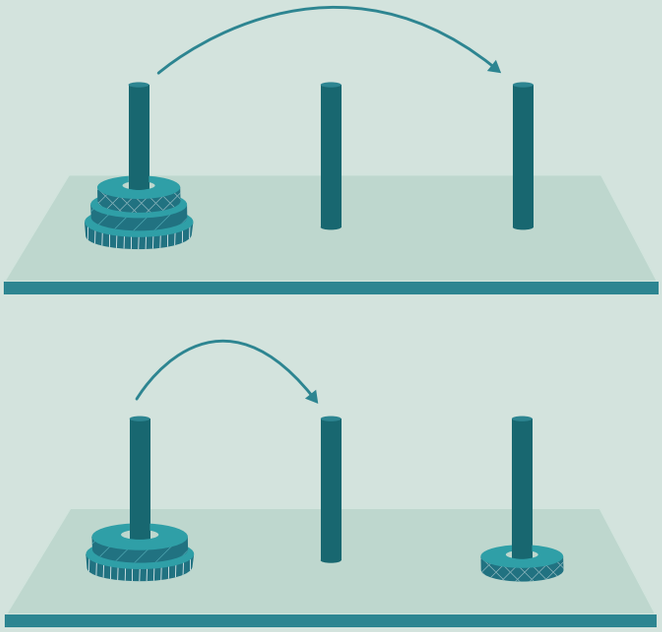
Movimentar o disco do pino 1 para o pino 3.

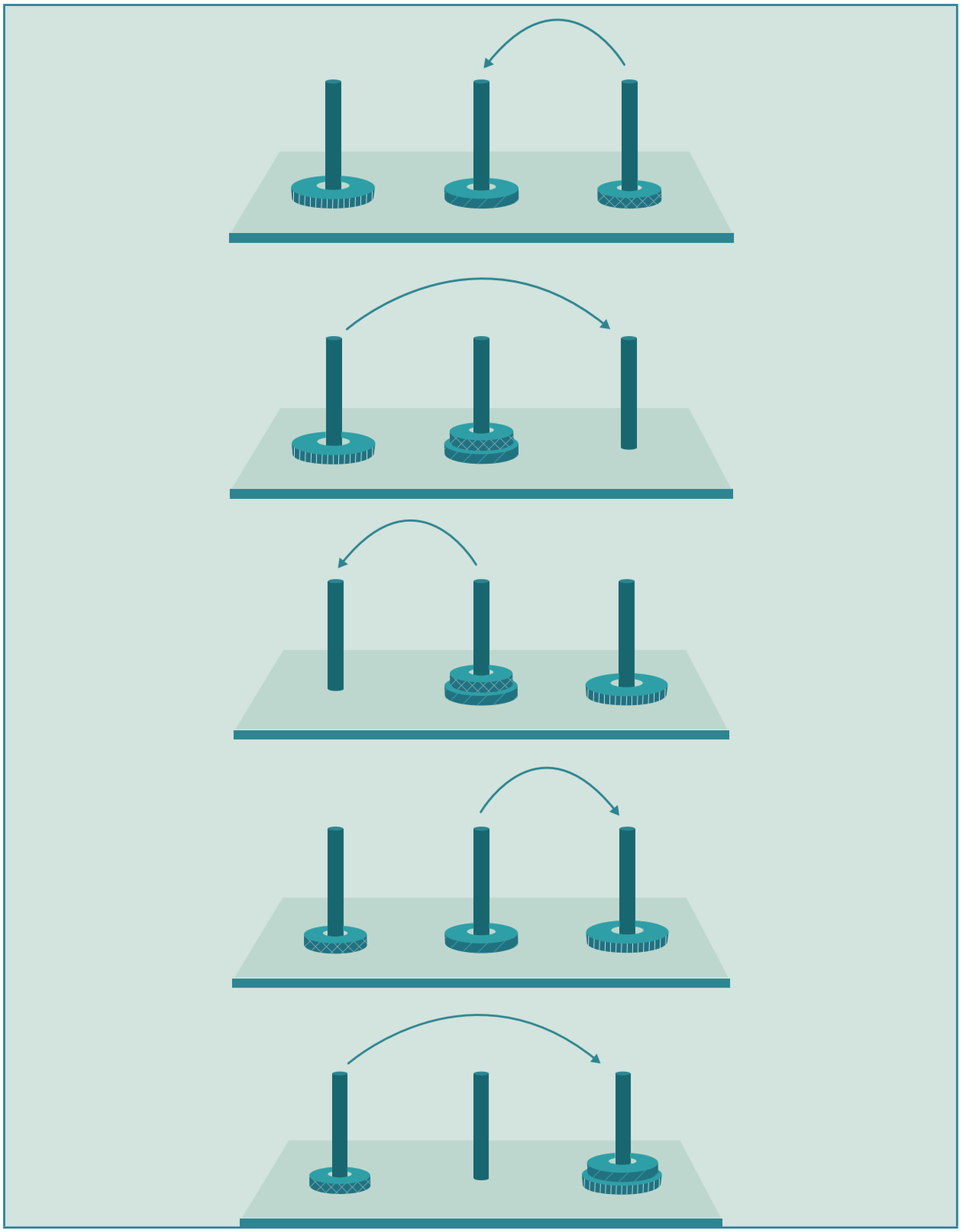
Movimentar o disco do pino 2 para o pino 1.

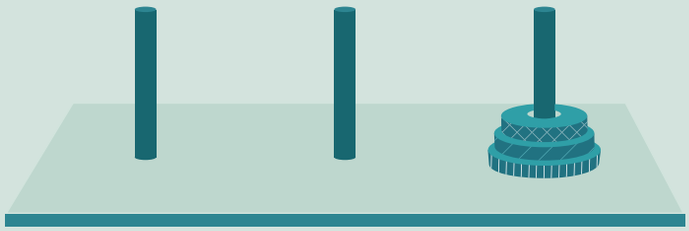
Movimentar o disco do pino 1 para o pino 3.

ANOTAÇÕES DA FASE DE EXECUÇÃO DO ALGORITMO

O resultado da execução de todos os movimentos é apresentado a seguir.







ANOTAÇÕES DA AVALIAÇÃO DA RESPOSTA

Chegamos à solução procurada e não há nada mais a ser feito.

A estratégia foi efetiva e foram necessárias apenas duas rodadas de aplicação e refinamento para encontrar a solução. Dependendo do problema, porém, mais iterações são necessárias. Outras técnicas podem ser utilizadas (em combinação ou não com a estratégia apresentada aqui). No futuro, quando você se deparar com um problema que exige uma técnica de projeto mais elaborada, consulte os trabalhos de Edmonds (2010) e Kleinberg e Tardos (2005).

Como você faria para movimentar 4 (quatros) discos do primeiro para o terceiro pino? Após quantas iterações você encontrou uma solução para o problema? Há algum padrão que pode ser repetido? Quantos movimentos foram realizados no algoritmo?

Vimos nesta unidade o que é um algoritmo e como ele pode ser construído utilizando uma estratégia reflexiva composta de várias fases. Ilustramos a estratégia aplicando-a no problema chamado de Torre de Hanoi. Como vimos, o principal ponto da estratégia é transformar o problema original em um problema mais simples, eliminando alguma das variáveis envolvidas. Este aspecto será amplamente explorado nas próximas unidades.

Alguns exercícios são indicados no GUIA DE ESTUDOS e sua resolução é fortemente indicada. Na próxima unidade, veremos como ir de algoritmos (estudados aqui) para programas de computador.

Referências

CORMEN, Thomas H. **Desmistificando Algoritmos**. Rio de Janeiro: Elsevier, 2014.

EDMONDS, Jeff. **Como pensar sobre algoritmos**. LTC, 2010.

POLYA, George. **How to solve it**. Princeton, 1945.

KNUTH, Donald E. **The Art of Computer Programming: Volume 1: Fundamental Algorithms**. Addison-Wesley Professional, 1968.

KNUTH, Donald E, PATASHNIK, Oren e GRAHAM, Ronald. **Matemática Concreta: Fundamentos para a Ciência da Computação**. Addison-Wesley, 1988.

KLEINBERG, Jon e TARDOS, Éva. **Algorithm Design**. Pearson, 2005.

UNIDADE 2

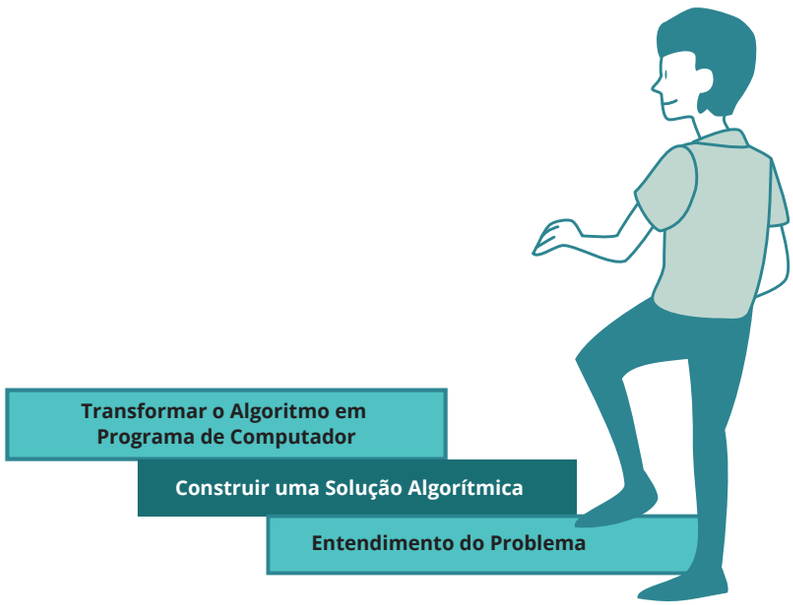
UNIDADE 2
De algoritmos à programas de computador

Após a leitura deste capítulo, você será capaz de:

- Entender o que é computador e quais são as propriedades de um tipo de algoritmo chamado programa de computador.
- Conhecer a linguagem de programação Python e como interagir com o computador.
- Escrever programas de computador simples, com interação com teclado e monitor e envolvendo operações aritméticas.

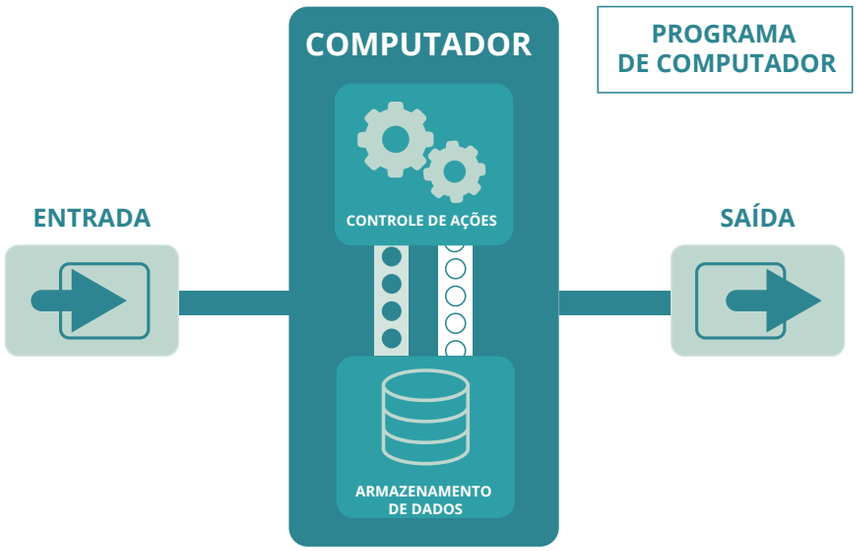
1 Introdução

Neste curso, estamos trilhando o caminho da **resolução de problemas** via computador. Vimos na unidade anterior como o conceito de algoritmo pode nos ajudar a estruturar a solução de um problema. Agora, daremos um passo adiante para entender como transformar estes algoritmos em programas de computador.



Um **programa de computador** é um tipo especial de algoritmo que foi projetado usando apenas um conjunto de ações que podem ser entendidas por um computador.

Afinal, o que é um computador? Como ele pode ser programado?



Um computador é uma máquina de resolução de problemas que manipula dados a partir da lista de instruções indicadas em um programa (ou, se preferir, algoritmo). Este processo funciona assim: o computador recebe um conjunto de dados de entrada, os processa e produz um conjunto de dados de saída. Uma parte do computador é responsável pelo **acesso** e **armazenamento de dados** e outra cuida do **controle das ações** definidas no programa.

Todo programa é escrito em uma **linguagem de programação** que especifica as regras (sintaxe, gramática etc) utilizadas para representação das ideias do algoritmo de uma maneira que o computador as entenda. Há dois tipos de linguagens: de **alto** e de baixo **nível** (às vezes, chamadas de *linguagens de máquina* ou *de montagem*). Resumindo um pouco as coisas, o computador só consegue executar programas escritos em linguagens de baixo nível. Programas escritos em linguagens de alto nível precisam ser processados antes que possam ser executados. Esse processamento extra (uma espécie de tradução) é chamado de compilação ou interpretação.

Pesquise sobre as linguagens de programação de alto nível e baixo nível, as vantagens e desvantagens de cada uma delas e como funciona a compilação e a interpretação.

2 Programas de computador e a linguagem Python

Guido van Rossum
(1956 -)

A linguagem de programação que você vai aprender é o **Python**. Trata-se de uma linguagem de alto nível, criada por **Guido van Rossum** (1956, -) em 1991, cujo nome é uma homenagem ao grupo humorístico britânico **Monty Python**.

De um ponto de visto operacional, uma linguagem de programação pode ser entendida como um conjunto de regras que definem como armazenar, acessar e processar dados. Programar é encontrar uma maneira de combinar essas instruções para resolver um problema. E é exatamente isso que faremos no restante desta unidade

Em Python, você pode escrever um programa inteiro em um arquivo de texto e utilizar o interpretador para traduzir o conteúdo do arquivo para que seja executado pelo computador. Esse arquivo é normalmente chamado de **código fonte**. Por exemplo, nós podemos usar um editor de texto para criar o código fonte chamado “meuprograma.py”. O .py no final do nome do arquivo indica que se trata de um programa escrito em Python.

No ambiente virtual de aprendizagem, há um vídeo explicando exatamente como isso deve ser realizado.

2.1 Comando de saída

A primeira instrução em linguagem Python que estudaremos é o comando **print**. Como o seu nome sugere, este comando imprime (ou melhor, escreve) alguma coisa na tela do computador. Um exemplo de impressão de uma frase é apresentado a seguir.

```
print('Sou aluno da Universidade Aberta do Brasil.')
```

Observe que a frase é indicada entre um bloco de aspas simples. A saída deste comando é:

```
Sou aluno da Universidade Aberta do Brasil.
```

Também é possível escrever números usando o comando print.

```
print(1234)
```

Agora não é preciso usar aspas. A saída é:

```
1234
```

2.2 Variáveis e operações aritméticas

Mas como os dados são manipulados e processados?

Antes de entender como os dados são representados em Python, vamos entender como isso acontece no computador.

Como o computador armazena os dados? Como eles podem ser acessados pelo programa?

Todo computador possui uma **memória** para armazenar dados. O programa de computador tem acesso a pedaços dessa memória que são chamados de **variáveis**. Cada variável possui um nome (por exemplo, **x**) que é associado ao pedaço específico da memória reservado para ela.

Para facilitar o entendimento, podemos enxergar a memória do computador como uma espécie de gaveteiro gigante. Cada gaveta é uma variável. Como em qualquer gaveteiro, é possível manipular cada uma das gavetas da memória individualmente. Dessa forma, podemos guardar um número na quinta gaveta, por exemplo.

Assim, toda vez que uma variável é criada em um programa de computador, uma das gavetas da memória é associada a ela. Suponha que a variável **x** foi criada. Uma gaveta da memória (a quinta gaveta, por exemplo) foi associada a **x**. Toda vez que esta variável for manipulada, o conteúdo dessa gaveta será acessado.

REGRAS PARA NOMES DE VARIÁVEIS

- Deve começar com uma letra ou subscrito.
- Nunca pode começar com um número.
- Pode conter letras maiúsculas, minúsculas, números e subscrito.
- Não se pode utilizar como parte do nome de uma variável os símbolos: **{ (+ - * / \ , ; . ! ?**
- Não se pode utilizar acentos.
- Exemplos de nomes inválidos: **5num**, **?hoje** e **+h**.
- Exemplos de nomes válidos: **var**, **numero**, **media** e **soma**.
- As palavras **and**, **as**, **assert**, **break**, **class**, **continue**, **def**, **del**, **elif**, **else**, **except**, **exec**, **finally**, **for**, **from**, **global**, **if**, **import**, **in**, **is**, **lambda**, **nonlocal**, **not**, **or**, **pass**, **raise**, **return**, **try**, **while**, **with**, **yield**, **True**, **False** e **None** não devem ser escolhidas como nomes de variáveis

Uma dica importante é sempre escolher um nome intuitivo para a variável. Um bom nome descreve o dado armazenado e facilita o entendimento do código.

A linguagem possui instruções que criam e manipulam o conteúdo individual de cada variável. O quadro a seguir mostra as instruções em Python para manipular variáveis, realizar operações aritméticas (soma, multiplicação, divisão, resto da divisão e subtração) e comparar valores.

OPERAÇÕES ARITMÉTICAS E MANIPULAÇÃO DE VARIÁVEIS EM PYTHON		
Tipo de operação	**Operador**	**Exemplo**
Atribuição de valor a variável	=	x = 5
Soma de dois números	+	5 + 1
Subtração de dois números	-	5 - 1
Multiplicação de dois números	*	5 * 1
Divisão de dois números	/	5 / 2
Resto da divisão inteira	%	5 % 2

Para ilustrar como estes comandos podem ser utilizados na prática, vamos escrever um programa que calcula e imprime na tela a média de 5 (cinco) números.

Como você faria para encontrar a média de 5 (cinco) números usando lápis e papel?

Com lápis e papel, somaríamos os números e dividiríamos o resultado por cinco.

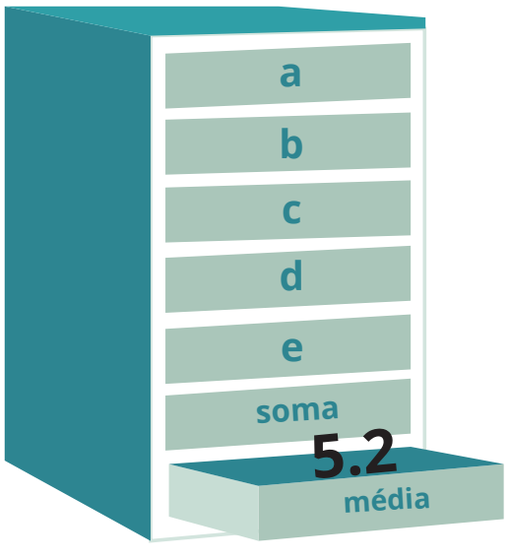
Pensando em um programa de computador, os cinco números são os dados de entrada do problema e precisam ser armazenados em variáveis.

Vamos precisar de quantas variáveis? Cinco?

Na verdade, seria interessante utilizar 7 (sete) variáveis: 5 (cinco) para os dados de entrada, uma para armazenar o resultado da soma e outra para a média. A solução desse problema utiliza as instruções de atribuição de valores, soma e divisão. No fim, também utilizamos o comando print para mostrar o resultado na tela do computador. O código a seguir mostra o programa de computador criado.

```
a = 1
b = 10
c = 5
d = 6
e = 4
soma = a + b + c + d + e
media = soma/5
print("Media dos numeros digitados: ", media)
```

```
Media dos numeros: 5.2
```



2.3 Comando de entrada

Uma limitação da solução anterior é que para alterar os números temos que mudar os valores no próprio código. Agora, vamos estudar o comando **input** e entender como realizar a leitura de valores a partir do teclado.

No exemplo a seguir, lemos um nome digitado do teclado e o armazenamos na variável **nome**.

```
nome = input('Qual o seu nome? ')
print('Olá ' + nome)
```

A saída do programa é apresentada a seguir.

```
Qual o seu nome?
```

Note que a frase informada ao comando **input** foi impressa na tela. Agora, o interpretador Python entrou em um modo que permite que o usuário digite um texto no teclado. Após o usuário digitar o que deseja e pressionar a tecla Enter, os dados digitados serão armazenados na variável nome.

Ao fim do processamento do programa, a seguinte saída será apresentada na tela.

```
Qual o seu nome? Guido
Olá Guido
```

O programa a seguir mostra como o código do problema da média dos 5 (cinco) números pode ser adaptado para ler os valores do teclado.

```
a = int(input("Informe um número: "))
b = int(input("Informe um número: "))
c = int(input("Informe um número: "))
d = int(input("Informe um número: "))
e = int(input("Informe um número: "))
soma = a + b + c + d + e
media = soma/5
print("Média dos números digitados: ", media)
```

Neste código, o comando **int** foi utilizado para transformar o texto digitado pelo usuário em um número inteiro.

```
Informe um número: 1
Informe um número: 1
Informe um número: 1
Informe um número: 2
Informe um número: 1
Média dos números digitados:  1.2
```

Nosso último exemplo desta unidade é um programa que calcula a área de uma circunferência.

Como você faria para calcular a área de um círculo usando lápis e papel?

A área A de uma circunferência de raio r é dada por:

$$A = \pi * r^2$$

Para transformar essa equação em um programa de computador, vamos precisar de 2 (duas) variáveis: uma para o raio e outra para área. O valor da variável raio deve ser lido do teclado e o valor da área deve ser calculado utilizando a fórmula. O código a seguir implementa esta ideia.

```
raio = float(input("Informe o raio do círculo: "))

area = 3.14 * raio * raio

print(Área: ', area)
```

O comando **float** foi utilizado para transformar o texto digitado pelo usuário no teclado em um número real (de ponto flutuante).

```
Informe o raio do círculo: 2

Área:  12.5
```

UNIDADE 3

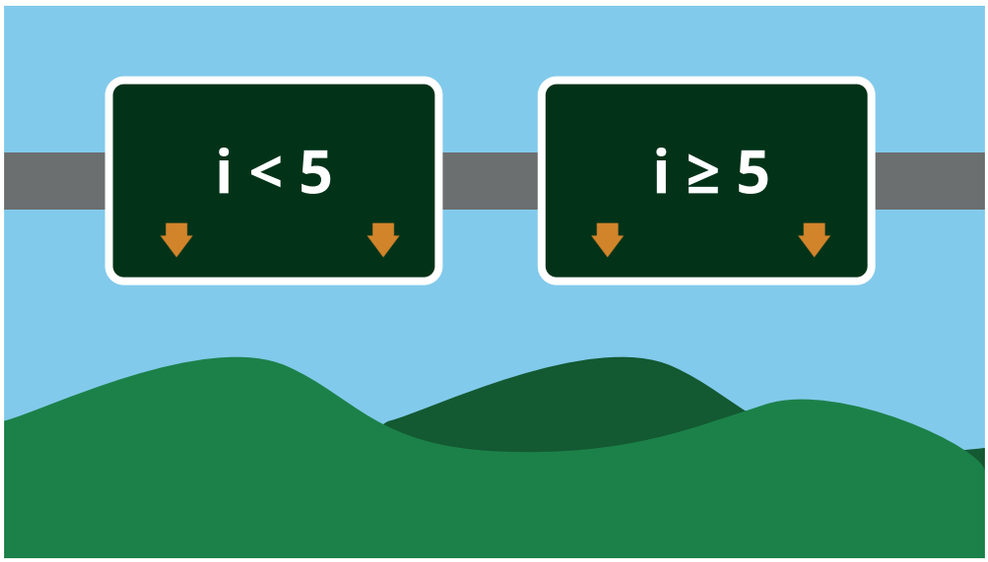
UNIDADE 3
Estruturas de decisão

Após a leitura deste capítulo, você será capaz de:

- Entender como um programa de computador pode tomar decisões e escolher entre executar ou não um conjunto de comandos.
- Aprender a escrever expressões lógicas utilizando os operadores e, ou e não.
- Visualizar os valores de uma expressão lógica utilizando uma Tabela Verdade.
- Conhecer o comandos if e else e escrever programas em Python que realizam decisões.

1 Introdução

Na Unidade II, vimos como escrever programas em Python que podem executar uma sequência de comandos de forma linear (executando sempre um comando após o outro). Embora esse conhecimento seja suficiente para resolver muitos problemas, há uma limitação importante: não podemos escrever programas que tomam decisões e escolhem executar ou não um conjunto de comandos.

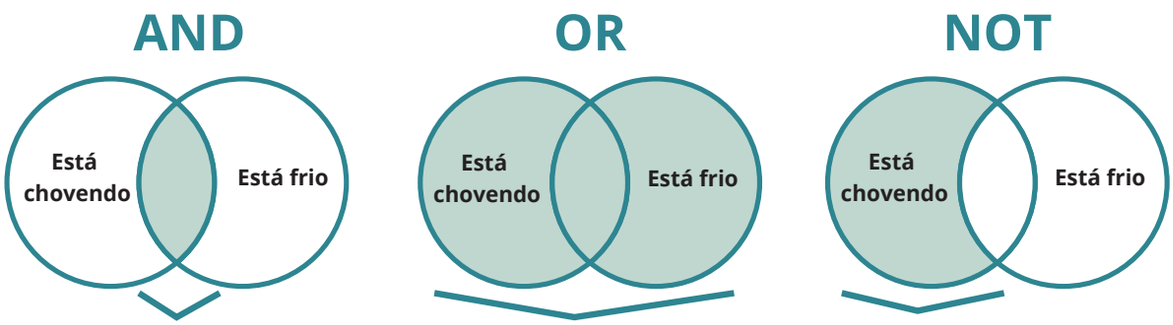


O mecanismo da linguagem de programação que permite decidir se um determinado bloco de comandos deve ou não ser executado é chamado de **comando condicional**. A decisão é realizada a partir do resultado de uma expressão em **álgebra booleana**.

2 Lógica booleana e Expressões relacionais

A álgebra booleana é o ramo da álgebra que considera variáveis que assumem os valores lógicos **verdadeiro** e **falso**. Ao contrário da álgebra elementar, em que os valores das variáveis são números, e as operações primárias são adição e multiplicação, as operações principais da álgebra booleana são a **conjunção e**, a **disjunção ou** e a **negação não**. A origem do nome é uma homenagem ao matemático inglês **George Boole** (1815, 1864).

Em Python, os valores lógicos verdadeiro e falso são denotados pelas palavras **True** e **False**. Os operadores lógicos de conjunção, disjunção e negação são denotados pelas palavras **and**, **or** e **not**, respectivamente.



Usando AND, a expressão será verdadeira apenas se as duas expressões são verdadeiras

Usando OR, a expressão será verdadeira se uma das expressões for verdadeira

Usando NOT, a expressão será verdadeira apenas quando a expressão negada não for verdadeira

Uma maneira de visualizar e entender uma expressão lógica é construir sua **Tabela Verdade**. A Tabela Verdade é uma de tabela que contém uma coluna para cada variável envolvida e uma linha para cada combinação possível de valores (GERSTING, 2016). A seguir, apresentamos as Tabelas Verdade dos operadores and, or e not.

TABELA VERDADE DO OPERADOR AND		
A	**B**	**A and B**
True	True	**True**
True	False	False
False	True	False
False	False	False

TABELA VERDADE DO OPERADOR OR		
A	**B**	**A or B**
True	True	**True**
True	False	**True**
False	True	**True**
False	False	False

TABELA VERDADE DO OPERADOR NOT	
A	**not A**
True	False
False	**True**

Observe que a expressão A **and** B é verdadeira apenas se A e B forem verdadeiras. A expressão A **or** B é verdadeira sempre que uma delas é verdadeira, e a expressão **not** A é verdadeira apenas se A é falsa.

Existe alguma relação entre estes operadores e a linguagem que falamos no cotidiano? O que queremos dizer quando falamos, por exemplo, que vamos ficar em casa se estiver frio e estiver chovendo? E quando dizemos que vamos ficar em casa se estiver frio ou estiver chovendo?

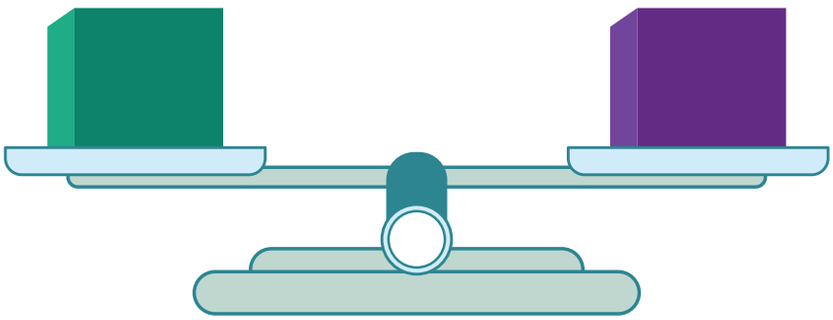
Em Python, podemos escrever o código a seguir para verificar o resultado destas operações.

```
A = True
B = False
print(A and B)
```

Como esperado, a saída deste código é False.

```
False
```

Em programação, também podemos chegar a um valor booleano (True ou False) fazendo uma comparação. As expressões que realizam uma comparação e devolvem um valor booleano são chamadas de **expressões relacionais**.



Uma expressão relacional pode ser enxergada como uma pesagem em uma balança de pratos em que verificamos se duas coisas são iguais, diferentes ou se uma coisa é menor, maior, menor ou igual ou maior ou igual à outra.

OPERADORES RELACIONAIS		
Operador	**Exemplo**	**Descrição**
==	A == B	O resultado da expressão é verdade apenas se A é igual à B.
!=	A != B	O resultado da expressão é verdade apenas se A é diferente de B.
<	A < B	O resultado da expressão é verdade apenas se A é menor que B.
>	A > B	O resultado da expressão é verdade apenas se A é maior que B.
<=	A <= B	O resultado da expressão é verdade apenas se A é menor ou igual à B.
>=	A >= B	O resultado da expressão é verdade apenas se A é maior ou igual à B.

O resultado das expressões relacionais também pode ser verificado no computador. O seguinte código mostra um exemplo para o operador < (menor).

```
A = 1
B = 5
print(A < B)
```

Como esperado, a saída deste código é True.

```
True
```

3 Comandos if e if-else

O **if** é o principal comando condicional da linguagem Python. Sua sintaxe é apresentada a seguir

```
...
if A:
        print('A é verdadeira')
```

Neste exemplo, o comando

```
print('A é verdadeira')
```

será executado apenas se o valor de A for **True**. Dessa maneira, podemos escolher quais comandos serão utilizados.

Podemos utilizar este comando, por exemplo, para verificar se um número digitado pelo usuário é par.

```
n = int(input('Digite um número: '))
if n % 2 == 0:
        print('Você digitou um número par')
```

A tabulação antes da instrução

```
print('Você digitou um número par')
```

serve para indicar que o comando faz parte do bloco de código interno do **if**.

Tal como fizemos na unidade anterior, usamos o comando input para fazer a leitura de um número inteiro do teclado. Para verificar se um número é par, olhamos para o resto da divisão inteira de **n** por 2. Apenas os números pares tem resto igual a zero. Por fim, usamos o operador relacional **==** para verificar se o resto vale zero.

Quando este programa é executado, a mensagem 'Você digitou um número par' será impressa apenas se o número digitado pelo usuário for par. Se o número digitado for ímpar, nada será impresso.

Quando temos que executar um comando, e também quando a condição é falsa, podemos utilizar o comando **if-else**. Os comandos que fazem parte do bloco de código **else** serão executados se a expressão for falsa. O código abaixo mostra um exemplo.

```
n = int(input('Digite um número: '))
if n % 2 == 0:
    print('Você digitou um número par')
else:
    print('Você digitou um número ímpar')
```

Note agora que, se o número digitado for ímpar, a frase 'Você digitou um número ímpar' será impressa na tela. Tal como antes, a mensagem 'Você digitou um número par' será impressa apenas se o número digitado pelo usuário for par.

Para ilustrar como esses comandos podem ser utilizados na prática, vamos apresentar um exemplo de programa que lê e ordena (de maneira crescente) 3 (três) números inteiros.

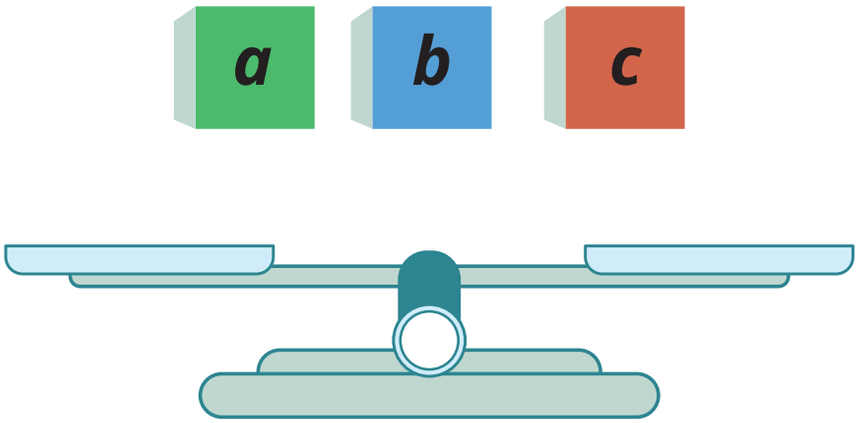
Vamos precisar de quantas variáveis? Três? Seis?

Vamos precisar de, pelo menos, 3 (três) variáveis: uma para número lido do teclado. Usando o que aprendemos na Unidade II, vamos ler os três números inteiros com o comando input.

```
a = int(input('Informe um número: '))
b = int(input('Informe um número: '))
c = int(input('Informe um número: '))
```

Pronto! Resolvemos uma pequena parte do problema. A ideia é ir resolvendo o problema aos poucos. Se você prestar um pouco de atenção, perceberá que esta é a mesma estratégia apresentada na Unidade I.

Mas como ordenar os três números?



Suponha que estamos trabalhando com um problema um pouco diferente. Imagine que temos três caixas com pesos desconhecidos e queremos ordená-los do menor para o maior usando uma balança de pratos. Com um pouco de atenção, é possível perceber que, na essência, este é o mesmo problema que estamos resolvendo.

Seguindo a estratégia da Unidade I precisamos:

1. Entender o problema ;

2. Construir uma solução ;

3. Executar a solução construída;

4. Avaliar a solução e, dependendo do resultado, voltar ao início.

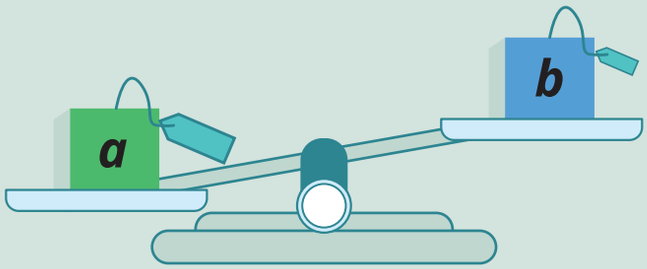
ANOTAÇÕES DA FASE DE ENTENDIMENTO DO PROBLEMA

Após um tempo pensando, percebemos que:

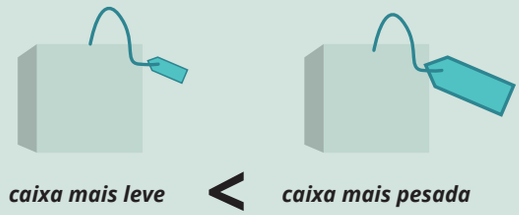
- Devemos ordenar as três caixas em ordem crescente.
- É possível comparar duas caixas com uma pesagem na balança.
- A solução deve funcionar para todos os possíveis pesos de caixas.

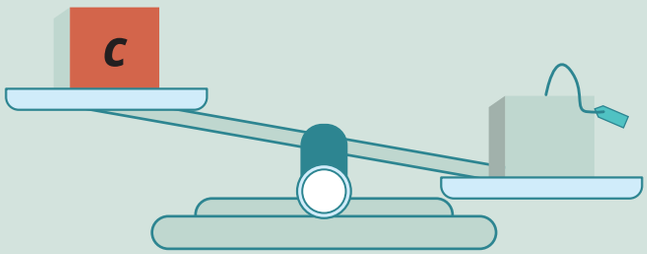
ANOTAÇÕES DA FASE DE CONSTRUÇÃO DO ALGORITMO

Infelizmente, não fica evidente, logo no começo, como ordenar as 3 (três) caixas. Simplificando o problema, suponha que devemos ordenar apenas 2 (duas) caixas.

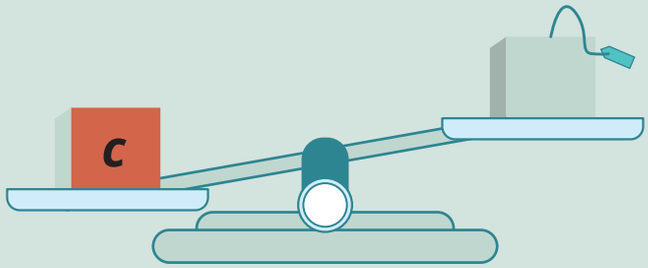


Quando comparamos duas caixas **a** e **b**, ganhamos uma informação sobre sua ordem relativa. Esta ordem não muda. Podemos guardar o resultado da comparação colocando uma etiqueta na caixa mais leve e outra na mais pesada. O resultado parcial é:

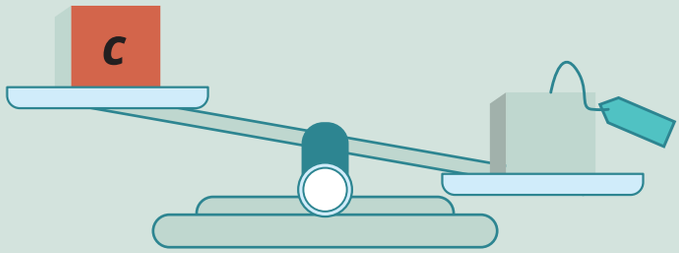




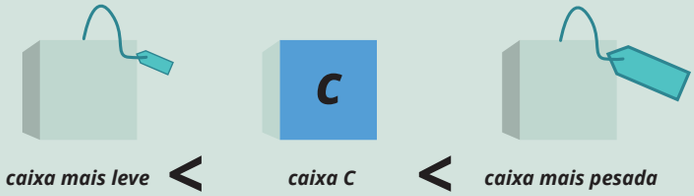
Brincando com as pesagens, podemos notar que se a caixa **c** é mais leve que a caixa, sabemos que a caixa **c** é a mais leve de todas. A ordem total é:



Caso contrário (a caixa **c** é mais pesada que a caixa mais leve), sabemos certamente que a caixa com o rótulo 'mais leve' é a menos pesada de todas. Não sabemos nada sobre a relação de ordem que existe entre a caixa **c** e a caixa mais pesada, e vamos precisar realizar mais uma pesagem.



Se a caixa **c** é mais leve que a caixa que contém o rótulo mais pesada, então a caixa **c** é a segunda mais leve. A ordem total é:



Se a caixa c é mais pesada que a caixa que tem o rótulo mais pesada, então ela é a mais pesada de todas. A ordem total é:

ANOTAÇÕES DA FASE DE EXECUÇÃO DO ALGORITMO
O algoritmo faz três pesagens e consegue ordenar as três caixas independentemente do seu peso. O uso das etiquetas reduziu o número de pesagens.
ANOTAÇÕES DA AVALIAÇÃO DA RESPOSTA

Alcançamos a solução do problema?

Sim, conseguimos!

O que falta para resolver o problema original?

Para resolver o problema original, podemos notar que:

- É possível comparar dois números usando o operador **<** e a instrução **if**.
- É possível utilizar variáveis no lugar dos rótulos das caixas.

A solução completa é apresentada a seguir.

```
a = int(input('Informe um número: '))
b = int(input('Informe um número: '))
c = int(input('Informe um número: '))
if a < b:
   mais_leve = a
   mais_pesada = b
else:
   mais_leve = b
   mais_pesada = a
if c < mais_leve:
   print(c, mais_leve, mais_pesada)
else:
   if(c < mais_pesada):
      print(mais_leve, c, mais_pesada)
   else:
      print(mais_leve, mais_pesada, c)
```

Referências

Judith L. GERSTING. **Fundamentos Matemáticos para a Ciência da Computação**. LTC, 2016.

UNIDADE 4

UNIDADE 4
Estruturas de repetição

Após a leitura deste capítulo, você será capaz de:

- Identificar a estrutura repetitiva de um problema.
- Conhecer o comando while e escrever programas em Python que realizam repetições.
- Aprender a testar e simular programas com comandos de repetição.

1 Introdução

Sem dúvida alguma, a capacidade de repetir operações foi decisiva para consolidar os computadores como ferramentas de resolução de problemas. Uma vantagem clara é que na repetição realizada por um computador todas as operações são executadas do mesmo jeito.

Nesta unidade, estudaremos um comando que pode ser usado para executar uma ou mais instruções repetidamente, até um número desejado de vezes. As repetições são organizadas em ciclos. No início de cada ciclo, uma expressão lógica é testada para determinar se a repetição deve prosseguir ou não.

Para entender como isso acontece, vamos considerar um exemplo. Suponha que temos que escrever um programa que mostra os números 1, 2, 3 e 4 na tela do computador.

Usando o que aprendemos na Unidades II e III, chegamos à solução a seguir.

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

Imagine agora que temos que imprimir todos os números entre 1 e 100.

Agora, usando apenas o que sabemos, vamos precisar escrever 100 linhas de código! Não será difícil cometer erros de digitação, esquecendo de imprimir um número, por exemplo.

```
print(1)
print(2)
print(3)
print(4)
...
print(97)
print(98)
print(99)
print(100)
```

Afinal, como podemos superar a limitação dessa solução?

Para isso, vamos precisar usar um novo comando.

2 Comando while

O comando **while** é utilizado para executar um bloco de comandos *enquanto* uma condição é satisfeita. Importante: a repetição deixa de ser executada quando a condição é falsa. A estrutura do comando é apresentada a seguir.

```
while condicao:
    comandos
```

Três perguntas principais nos ajudam a utilizar este comando:

- O que deve ser repetido?
- Quantas vezes devem se repetir?
- Qual condição pode ser utilizada para representar essa repetição?

Agora que já conhecemos o comando while, podemos voltar ao problema da impressão de todos os números entre 1 e 100.

Com um pouco de atenção, é possível perceber que a instrução

```
print( )
```

é repetida 100 vezes.

O que muda de uma linha para outra é apenas o valor entre parênteses. Este valor varia de 1 a 100.

IMPORTANTE

Identificar o que se repete e o que muda é a tarefa chave nesse processo.

Uma estratégia muito útil é escrever um código com todos os ciclos de repetição. Isso deixa claro quais serão as operações serão repetidas.

O código a seguir mostra os 100 ciclos de repetição usados para imprimir os 100 números. Em geral, podemos usar uma variável para representar o conteúdo que não é fixo. Uma ideia é iniciar uma variável i valendo 1 e incrementar i a cada ciclo da repetição.

```
1.      i = 1
2.      print(i)
3.      i = i + 1
4.      print(i)
5.      i = i + 1
6.      print(i)
7.      i = i + 1
8.      print(i)
9.      i = i + 1
10.     ...
11.     print(i)
12.     i = i + 1
```

Com esse código, podemos responder às questões colocadas acima.

O que deve ser repetido?

Resposta: Impressão dos números 1, 2, 3, ... 100. Neste caso, isso é feito repetindo os comandos:

```
print(i)
i = i + 1
```

Quantas vezes devem se repetir?

Resposta: 100 vezes.

Qual condição pode ser utilizada para representar essa repetição?

Resposta: Podemos utilizar uma variável para contar quantas repetições já foram executadas. No código, a própria variável *i* já faz isso. Dessa forma, a condição pode ser definida usando: *i* ≤ 100. Lembre-se: a impressão será interrompida quando *i* valer 101.

Com essas informações, podemos escrever o código com while a seguir:

```
1.      i = 1
2.      while i <= 100:
3.          print(i)
4.          i = i + 1
```

Tal como queríamos, as linhas 3 e 4 do código serão executadas 100 vezes. Na 101ª vez, a condição deixa de ser verdadeira, pois a variável i vale 101 (não é mais "menor ou igual a 100") e a repetição é interrompida.

2.1 Encontrando o maior número digitado

Vamos resolver outro problema para ilustrar a aplicação desse comando na prática. O objetivo agora é escrever um programa que lê um número inteiro **n** e, em seguida, lê **n** valores inteiros e mostra o maior deles na tela.

Após a leitura de **n** igual a 4, por exemplo, seu programa deve ler 4 (quatro) números inteiros. Estes quatro números poderiam ser, digamos, 6, 4, 1 e 7. Seu programa deveria imprimir, neste caso, o número 7 (o maior número digitado pelo usuário).

Para resolver este problema, vamos utilizar uma estratégia muito interessante. A ideia é dividir o problema em partes bem pequenas, resolver cada uma delas separadamente e, depois, juntar tudo. Cada parte bem pequena do problema precisa ser cuidadosamente estudada. Este estudo cuidadoso vai te ajudar a traduzir as regras subjacentes envolvidas na questão em uma forma que possa ser absorvida pelo seu subconsciente e, quando você menos esperar, o problema estará resolvido. Vamos lá!

O primeiro subproblema desta atividade diz respeito à leitura dos números. É sempre bom começar pela leitura. Afinal, como vamos resolver o problema se nem conseguimos fazer a leitura dos valores?

Para fazer a leitura, precisamos de variáveis e de algumas chamadas do comando **input**.

```
n = int(input('Quantos números serão lidos? '))
```

Agora que já conseguimos ler os números, vamos pensar no que podemos fazer para identificar o maior deles. Pode parecer difícil, mas temos uma técnica. O princípio dessa técnica é simples: sempre que encontrar um problema difícil, você deve tentar diminuí-lo ou simplificá-lo um pouco. Faça isso até que o problema fique simples demais para simplificar novamente e, depois, volte resolvendo os problemas maiores (mais complicados).

Por exemplo, o que acontece quando temos apenas um número? É bem mais fácil resolver este problema! O maior número em uma sequência de um número, é sempre o único número! Em Python, isto pode ser escrito da seguinte forma:

```
1.  num = int(input('Informe um número: '))
2.  maior = num;
3.  print("O maior número é: ", maior);
```

E se tivéssemos dois números? Qual deles poderia ser o maior? Há, obviamente, duas possibilidades e, para ter certeza, vamos precisar usar o comando de comparação **if**, estudado na Unidade III. O código a seguir implementa essa ideia.

```
1.  num = int(input('Informe um número: '))
2.  maior = num;
3.
4.  num = int(input('Informe um número: '))
5.  if maior < num:
6.      maior = num
7.  print("O maior número é: ", maior);
```

E se tivéssemos três números? Teríamos três possibilidades e precisaríamos de mais uma comparação. O código a seguir mostra uma implementação possível.

```
1.      num = int(input('Informe um número: '))
2.      maior = num;
3.
4.      num = int(input('Informe um número: '))
5.      if maior < num:
6.           maior = num
7.
8.      num = int(input('Informe um número: '))
9.      if maior < num:
10.          maior = num
11.
12.     print("O maior número é: ", maior);
```

Este exemplo é revelador. É possível perceber agora que para analisar quatro números, vamos precisar de mais uma comparação. Generalizando, podemos perceber que com **n** números vamos precisar analisar **n** possibilidades e podemos fazer isto com **n - 1** comparações (comandos **if**)!

Agora que descobrimos a estrutura do problema, podemos traduzir este conhecimento em um código com repetição utilizando o comando **while**. É só identificar a condição de parada e as ações que precisam ser repetidas.

Com um pouco de atenção, é possível perceber que o bloco de comandos

```
num = int(input('Informe um número: '))
if maior < num:
    maior = num
```

deve ser repetido n-1 vezes.

O código resultante é apresentado a seguir.

```
1.    n = int(input('Quantos números serão lidos? '))
2.    num = int(input('Informe um número: '))
3.    maior = num;
4.    i = 1
5.    while i <= n-1:
6.        num = int(input('Informe um número: '))
7.        if maior < num:
8.            maior = num
9.        i = i + 1
10.   print("O maior número é: ", maior);
```

2.2 Testando e simulando um código com lápis e papel

Há basicamente duas maneiras de testar informalmente se um programa está fazendo o que se espera: executá-lo no computador ou inspecioná-lo manualmente. Nesta seção, vamos aprender como realizar a inspeção manual, também chamada de teste de mesa (LEITE, 2006).

A inspeção manual é uma maneira de entender o que acontece em cada instrução de um programa e consiste em 4 (quatro) passos:

1. Identificar todas as variáveis.

2. Criar uma tabela com uma coluna para cada variável.

3. Percorrer o código linha a linha, preenchendo a tabela. Cada coluna de uma variável deve conter o respectivo valor dessa variável após a linha de código ser executada. Dica: Anote no canto da tabela a linha do código que está sendo analisada.

4. Verificar se o resultado produzido é o esperado.

No programa que acabamos de escrever para encontrar o maior número digitado pelo usuário, há 4 (quatro) variáveis: **n**, **num**, **maior** e **i**.

Para simular este programa, vamos criar uma tabela com 4 (quatro) colunas e percorrer cada linha do código e atualizar o valor das variáveis.

O quadro a seguir mostra a tabela criada neste caso. Adicionamos uma coluna para guardar a linha do código que foi executada. Os traços indicam que nenhum valor foi armazenados nas variáveis.

	Variáveis			
Linha	**n**	**num**	**maior**	**I**
	-	-	-	-

Agora que a Tabela já foi criada, precisamos percorrer cada linha do código. Lembre-se de que as operações dentro do bloco de repetição deixam de ser executadas apenas quando a condição do comando **while** é falsa.

Após executar a instrução

```
n = int(input('Quantos números serão lidos? '))
```

o usuário informará a quantidade de números que serão digitados na sequência. Vamos assumir neste exemplo que o número digitado foi 3 (três). Este valor é armazenado na coluna da variável n.

	Variáveis			
Linha	**n**	**num**	**maior**	**I**
1	**3**	-	-	-

Observe que a coluna da variável **n** foi atualizada na Tabela.

Continuando a simulação, atualizamos a tabela após a instrução:

```
num = int(input('Informe um número: '))
```

	Variáveis			
Linha	**n**	**num**	**maior**	**i**
1	3	-	-	-
2	3	**6**	-	-

Assumimos que o usuário digitou o número 6.

Após a instrução:

```
maior = num
```

temos os seguintes valores na tabela:

	Variáveis			
Linha	**N**	**num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	**6**	-

E após a instrução

```
i = 1
```

a tabela terá os seguintes valores:

	Variáveis			
Linha	**N**	**num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	6	
4	3	6	6	**1**

Como chegamos ao comando de repetição, sabemos que a instrução

```
while i <= n-1:
```

poderá ser executada várias vezes.

Toda vez que esta linha for executada, testamos a condição

```
i <= n-1
```

Se esta condição for verdadeira, o bloco de códigos da repetição será executado, e executamos a linha 6. Se for condição for falsa, por outro lado, pulamos para a linha 10.

Para testar a condição, vamos olhar os valores de **i** e **n** na tabela. A variável **i** vale 1 e a variável **n** vale 3. Logo, 1 <= 2 é verdade.

Como a condição é verdadeira, vamos executar a instrução:

```
num = int(input('Informe um número: '))
```

Assumindo que o valor digitado pelo usuário é 4, a tabela contém os seguintes valores.

	Variáveis			
Linha	**N**	**num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	6	
4	3	6	6	1
6	3	**4**	6	1

Agora, a instrução

```
if maior < num:
```

será executada.

Como vimos na Unidade III, o bloco da condicional (linha 8) será executado apenas se a condição do comando if for verdadeira. Caso contrário, a instrução da linha 9 será executada.

Vamos verificar, então, se o valor da variável **maior** armazenado na tabela é menor que o valor da variável **num**. Como maior vale 6 e **num** vale 4, esta condição é falsa.

Como a condição é falsa, vamos executar a instrução:

```
i = i + 1
```

Após esta instrução, a tabela terá a seguinte configuração.

	Variáveis			
Linha	**N**	**Num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	6	
4	3	6	6	1
6	3	4	6	1
8	3	4	6	**2**

Como estamos executando o bloco de repetição, devemos voltar e executar novamente a instrução da linha 5:

```
while i <= n-1:
```

Mais uma vez, vamos testar a condição:

```
i <= n-1
```

Como fizemos lá atrás, vamos olhar os valores de **i** e **n** na tabela. A variável **i** agora vale 2, e a variável **n** vale 3. Logo, 2 <= 2 é verdade.

Como esta condição é verdadeira, vamos executar o bloco da repetição mais uma vez. Assumindo que o valor digitado pelo usuário foi 7, após a instrução

```
num = int(input('Informe um número: '))
```

a tabela agora contém os seguintes valores.

	Variáveis			
Linha	**n**	**Num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	6	
4	3	6	6	1
6	3	4	6	1
8	3	4	6	2
6	3	**7**	6	2

Novamente, será executada a instrução:

```
if maior < num:
```

Como vimos, o comando da linha 8 será executado apenas se a condição

```
maior < num
```

for verdadeira.

Como a condição é verdadeira (note que a variável **maior** vale 6 e **num** vale 7), vamos executar a instrução:

```
maior = num
```

Após a execução desta instrução, a tabela contém os seguintes valores.

	Variáveis			
Linha	**n**	**num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	6	
4	3	6	6	1
6	3	4	6	1
8	3	4	6	2
6	3	7	6	2
8	3	7	**7**	2

Após a instrução

```
i = i + 1
```

a tabela contém os seguintes valores.

	Variáveis			
Linha	**n**	**num**	**maior**	**i**
1	3	-	-	-
2	3	6	-	-
3	3	6	6	
4	3	6	6	1
6	3	4	6	1
8	3	4	6	2
6	3	7	6	2
8	3	7	7	2
9	3	7	7	**3**

Novamente, como estamos em um bloco de repetição, devemos voltar e executar a instrução da linha 5

```
while i <= n-1:
```

Vamos testar a condição:

```
i <= n-1
```

Conferindo a tabela, percebemos que a variável **i** vale 3, e a variável **n** também vale 3. Logo, 3 <= 2 não é verdade.

Como esta condição é falsa, vamos executar a instrução da linha 10.

Após instrução da linha 10

```
print("O maior número é: ", maior);
```

teremos a seguinte saída.

O maior número é: 7

Como não há mais instruções, o programa acaba. Observe que o programa produziu exatamente o resultado esperado. Foram digitados os números 6, 4 e 7, e o programa encontrou que o 7 é o maior deles.

Pensando um pouco na mecânica desse algoritmo, percebemos que ele funciona de modo semelhante a uma pessoa que seleciona a maior caixa em uma esteira. O programa pega a primeira caixa e, a cada momento, compara a caixa que está em sua mão com a que está passando na esteira. Se a caixa que está passando é maior, ele troca. No fim, não é difícil perceber que a maior caixa estará na mão do algoritmo.

Palavras Finais

Na sociedade contemporânea, as Tecnologias de Informação e Comunicação (TICs) desempenham um papel preponderante em diversas esferas de nossas vidas. Como dissemos no começo, o computador está em toda parte e, tal como a eletricidade, o passou a compor o nosso entorno sem que percebêssemos toda a complexidade envolvida.

Pensando nisso, entendemos que a educação não deve somente capacitar o indivíduo a trabalhar com alguma ferramenta específica, mas também capacitá-lo para perceber o computador como um instrumento de resolução de problemas que aumenta o seu poder cognitivo e operacional. Todo aluno deve ser capaz de criar e propor as suas próprias ferramentas.

Por isso, e considerando o papel que a informática tem representado no desenvolvimento dos países, entendemos que os estudantes deste curso devem compreender como os problemas são resolvidos por computador e, até mesmo, criar as suas próprias soluções.

Ao profissional licenciado para atuar com Tecnologias Educacionais cabe pensar em como vencer as barreiras impostas pelos alunos nos diferentes níveis de ensino e contribuir para o desenvolvimento de esquemas mentais que o capacitem a chegar à solução de problemas.

Só assim, o acesso ao computador, à internet e às TICs em geral será um indicador de inclusão, de exercício da cidadania e de transformação na sociedade em que vivemos.

Referências

LEITE, Mario. **Técnicas de Programação - Uma Abordagem Moderna.** Brasport, 2006.

UNIVERSIDADE FEDERAL
DE MATO GROSSO

SETEC
SECRETARIA DE
TECNOLOGIA EDUCACIONAL

UAB UNIVERSIDADE
ABERTA DO BRASIL