**Introdução à Programação com Python**

**Algoritmos e lógica de programação para iniciantes 2a Edição**

**Introdução à Programação com Python**

**Algoritmos e lógica de programação para iniciantesNilo Ney Coutinho Menezes**

Novatec

Copyright © 2010, 2014 da Novatec Editora Ltda.

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998.

É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito,  do autor e da Editora.

Editor: Rubens Prates PY20170220

Revisão gramatical: Adriana Bernardino

Editoração eletrônica: Camila Kuwabata

Capa: Victor Bittow

ISBN: 978-85-7522-559-2

Histórico de edições impressas:

Fevereiro/2017 Quarta reimpressão

Junho/2016 Terceira reimpressão

Janeiro/2016 Segunda reimpressão

Maio/2015 Primeira reimpressão

Junho/2014 Segunda edição (ISBN: 978-85-7522-408-3)

Agosto/2013 Terceira reimpressão

Novembro/2012 Segunda reimpressão

Outubro/2011 Primeira reimpressão

Novembro/2010 Primeira edição (ISBN: 978-85-7522-250-8)

Novatec Editora Ltda.

Rua Luís Antônio dos Santos 110

02460-000 – São Paulo, SP – Brasil

Tel.: +55 11 2959-6529

E-mail: novatec@novatec.com.br

Site: www.novatec.com.br

Twitter: twitter.com/novateceditora

Facebook: facebook.com/novatec

LinkedIn: linkedin.com/in/novatec

*A minha esposa, Chris; e aos meus filhos, Igor, Hanna e Iris.*

**Sumário**

**Agradecimentos.....................................................................................................13 Prefácio da segunda edição ....................................................................................14 Prefácio da primeira edição ....................................................................................15 Introdução.............................................................................................................16**

**Capítulo 1** ■**Motivação ...........................................................................................19**

1.1 Você quer aprender a programar?.......................................................... 19 1.2 Como está seu nível de paciência? ........................................................20 1.3 Quanto tempo você pretende estudar?.................................................. 21 1.4 Programar para quê?............................................................................ 21

1.4.1 Escrever páginas web..................................................................... 21 1.4.2 Acertar seu relógio........................................................................22 1.4.3 Aprender a usar mapas..................................................................22 1.4.4 Mostrar para seus amigos que você sabe programar .......................22 1.4.5 Parecer estranho ...........................................................................22 1.4.6 Entender melhor como seu computador funciona..........................23 1.4.7 Cozinhar......................................................................................23 1.4.8 Salvar o mundo ............................................................................24 1.4.9 Software livre ...............................................................................24 1.5 Por que Python? ..................................................................................24

**Capítulo 2** ■ **Preparando o ambiente.......................................................................27**

2.1 Instalação do Python ...........................................................................27 2.1.1 Windows.......................................................................................28 2.1.2 Linux ...........................................................................................34 2.1.3 Mac OS X.....................................................................................34 2.2 Usando o interpretador .......................................................................34

**7**

**8 Introdução à Programação com Python**

2.3 Editando arquivos...............................................................................36 2.4 Cuidados ao digitar seus programas.....................................................40 2.5 Os primeiros programas...................................................................... 41 2.6 Conceitos de variáveis e atribuição.......................................................44

**Capítulo 3** ■ **Variáveis e entrada de dados................................................................49**

3.1 Nomes de variáveis...............................................................................49 3.2 Variáveis numéricas.............................................................................50 3.2.1 Representação de valores numéricos .............................................. 51 3.3 Variáveis do tipo Lógico .......................................................................53 3.3.1 Operadores relacionais ..................................................................53 3.3.2 Operadores lógicos.......................................................................55 3.4 Variáveis string ....................................................................................60 3.4.1 Operações com strings ..................................................................62 3.5 Sequências e tempo .............................................................................67 3.6 Rastreamento ......................................................................................68 3.7 Entrada de dados.................................................................................69 3.7.1 Conversão da entrada de dados......................................................70 3.7.2 Erros comuns...............................................................................72

**Capítulo 4** ■ **Condições............................................................................................75**

4.1 if.........................................................................................................75 4.2 else .....................................................................................................79 4.3 Estruturas aninhadas ..........................................................................80 4.4 elif......................................................................................................83

**Capítulo 5** ■ **Repetições...........................................................................................85**

5.1 Contadores..........................................................................................87 5.2 Acumuladores ..................................................................................... 91 5.3 Interrompendo a repetição ...................................................................93 5.4 Repetições aninhadas...........................................................................95

**Capítulo 6** ■ **Listas...................................................................................................98**

6.1 Trabalhando com índices.....................................................................101 6.2 Cópia e fatiamento de listas................................................................101 6.3 Tamanho de listas.............................................................................. 104 6.4 Adição de elementos.......................................................................... 105 6.5 Remoção de elementos da lista........................................................... 108

**Sumário 9**

6.6 Usando listas como filas .................................................................... 109 6.7 Uso de listas como pilhas....................................................................110 6.8 Pesquisa.............................................................................................112 6.9 Usando for .........................................................................................114 6.10 Range ...............................................................................................115 6.11 Enumerate.........................................................................................117 6.12 Operações com listas.........................................................................117 6.13 Aplicações.........................................................................................118 6.14 Listas com strings............................................................................. 120 6.15 Listas dentro de listas....................................................................... 120 6.16 Ordenação....................................................................................... 123 6.17 Dicionários ...................................................................................... 127 6.18 Dicionários com listas.......................................................................131 6.19 Tuplas.............................................................................................. 133

**Capítulo 7** ■ **Trabalhando com strings.................................................................... 137**

7.1 Verificação parcial de strings............................................................... 138 7.2 Contagem ......................................................................................... 140 7.3 Pesquisa de strings............................................................................. 140 7.4 Posicionamento de strings.................................................................. 144 7.5 Quebra ou separação de strings...........................................................145 7.6 Substituição de strings........................................................................145 7.7 Remoção de espaços em branco.......................................................... 146 7.8 Validação por tipo de conteúdo ...........................................................147 7.9 Formatação de strings........................................................................ 150

7.9.1 Formatação de números............................................................... 152 7.10 Jogo da forca .................................................................................... 156

**Capítulo 8** ■ **Funções............................................................................................. 161**

8.1 Variáveis locais e globais..................................................................... 168 8.2 Funções recursivas............................................................................. 170 8.3 Validação .......................................................................................... 173 8.4 Parâmetros opcionais..........................................................................174 8.5 Nomeando parâmetros...................................................................... 176 8.6 Funções como parâmetro................................................................... 178 8.7 Empacotamento e desempacotamento de parâmetros ......................... 179 8.8 Desempacotamento de parâmetros..................................................... 180 8.9 Funções Lambda ................................................................................181

**10 Introdução à Programação com Python**

8.10 Módulos...........................................................................................181 8.11 Números aleatórios........................................................................... 183 8.12 A função type................................................................................... 185

**Capítulo 9** ■ **Arquivos............................................................................................ 188**

9.1 Parâmetros da linha de comando .........................................................191 9.2 Geração de arquivos .......................................................................... 192 9.3 Leitura e escrita ................................................................................. 192 9.4 Processamento de um arquivo ............................................................ 193 9.5 Geração de HTML.............................................................................200 9.6 Arquivos e diretórios..........................................................................205 9.7 Um pouco sobre o tempo....................................................................210 9.8 Uso de caminhos................................................................................213 9.9 Visita a todos os subdiretórios recursivamente .....................................215

**Capítulo 10** ■ **Classes e objetos.............................................................................. 216**

10.1 Objetos como representação do mundo real ...................................... 216 10.2 Passagem de parâmetros...................................................................220 10.3 Exemplo de um banco...................................................................... 222 10.4 Herança........................................................................................... 227 10.5 Desenvolvendo uma classe para controlar listas................................. 230 10.6 Revisitando a agenda........................................................................ 243

**Capítulo 11** ■ **Banco de dados................................................................................ 260**

11.1 Conceitos básicos..............................................................................260 11.2 SQL................................................................................................. 262 11.3 Python & SQLite .............................................................................263 11.4 Consultando registros....................................................................... 270 11.5 Atualizando registros........................................................................ 273 11.6 Apagando registros........................................................................... 276 11.7 Simplificando o acesso sem cursores.................................................. 276 11.8 Acessando os campos como em um dicionário ..................................277 11.9 Gerando uma chave primária ........................................................... 278 11.10 Alterando a tabela...........................................................................280 11.11 Agrupando dados............................................................................ 281 11.12 Trabalhando com datas ...................................................................286 11.13 Chaves e relações.............................................................................289 11.14 Convertendo a agenda para utilizar um banco de dados.................... 292

**Sumário 11**

**Capítulo 12** ■ **Próximos passos............................................................................... 314**

12.1 Programação funcional......................................................................314 12.2 Algoritmos........................................................................................315 12.3 Jogos................................................................................................315 12.4 Orientação a objetos .........................................................................316 12.5 Banco de dados.................................................................................316 12.6 Sistemas web ....................................................................................316 12.7 Outras bibliotecas Python .................................................................317 12.8 Listas de discussão............................................................................317

**Apêndice A** ■**Mensagens de erro ........................................................................... 318**

A.1 SintaxError........................................................................................318 A.2 IdentationError .................................................................................319 A.3 KeyError .......................................................................................... 320 A.4 NameError....................................................................................... 320 A.5 ValueError.........................................................................................321 A.6 TypeError......................................................................................... 322 A.7 IndexError ....................................................................................... 322

**Referências.......................................................................................................... 323 Índice remissivo ................................................................................................... 324**

**Agradecimentos**

Este livro não seria possível sem o incentivo de minha esposa, Emília Christiane,  e a compreensão de meus filhos, Igor, Hanna e Iris. Para quem mora em um país  frio como a Bélgica, escrever no verão nem sempre é fácil.

Também agradeço o suporte oferecido por meus pais e avós durante meus estudos,  e os conselhos, compreensão e orientação que sempre recebi.

Não poderia me esquecer de meu irmão, Luís Victor, pela ajuda com as imagens  em português.

A Luciano Ramalho e aos colegas da lista python-brasil. Luciano, obrigado pelo  incentivo para publicar este livro e pelos comentários mais que pertinentes. À co munidade python-brasil pelo esforço e prova de civilidade ao manter as discussões  em altíssimo nível, agradando iniciantes, curiosos e profissionais de computação.

Agradecimentos também aos colegas, amigos e alunos da Fundação Matias  Machline, hoje Fundação Nokia de Ensino, onde tive oportunidade de estudar  e trabalhar como professor de lógica de programação. Aos amigos e colegas do  Centro Educacional La Salle e da Fundação Paulo Feitoza, onde ministrei cursos  de lógica de programação e de Python.

**13**

**Prefácio da segunda edição**

Revisar este livro foi um trabalho muito gratificante, especialmente pelas men sagens de apoio recebidas na lista Python-Brasil e por e-mail. Esta nova edição  traz um capítulo extra, cobrindo o básico de banco de dados, onde introduzimos  o uso do SQLite. O capítulo 10, que trata da orientação a objetos, também foi  expandido, e vários conceitos específicos ao Python foram adicionados. Muitas  pequenas revisões foram feitas por todo o livro, desde a atualização para Python  3.4 até a migração para Windows 8.1.

Apesar da tentação de cobrir tudo que se possa imaginar sobre Python, este  livro continua uma obra dedicada àqueles que dão seus primeiros passos na  programação de computadores. Nesta nova edição, conceitos mais específicos do  Python também foram abordados para tornar o texto e nossos programas mais  "Pythonicos".

**14**

**Prefácio da primeira edição**

Aprendi a programar usando a linguagem BASIC ainda em meados dos anos 1980.  Lembro-me de construir pequenos programas de desenho, agendas telefônicas e  jogos. Armazenamento só em fitas K-7. Antes da internet, e morando no norte do  Brasil, a forma de aprender a programar era lendo livros e, claro, programando. A  forma mais comum de obter novos programas era por meio de revistas de pro

gramação. Essas revistas eram verdadeiramente dedicadas à nova comunidade  de usuários de microcomputadores, termo usado na época para diferenciar os  computadores domésticos. Elas traziam listagens completas de programas, escritos  em BASIC ou *Assembly*. Em uma época em que o download mal era um sonho  distante, digitar esses programas era a única solução para poder executá-los. A  experiência de ler e digitar os programas foi muito importante para aprender a  programar, mas, infelizmente, poucas revistas técnicas hoje são acessíveis a ini ciantes. A complexidade dos programas de hoje também é muito maior, exigindo  mais tempo de estudo que antigamente. Embora a internet ajude muito, seguir  uma ordem planejada de aprendizado é muito importante.

Ao começar a ensinar programação, o desafio sempre foi encontrar um livro que  pudesse ser lido por alunos do ensino médio ou no início do ensino superior.  Embora várias obras tenham suprido essa necessidade, o uso de apostilas sempre  foi necessário, pois a ordem em que os novos conceitos eram apresentados era  quase sempre mais interessante para um dicionário que para o ensino de pro

gramação em si. Conceitos importantes para o iniciante eram completamente  esquecidos, e o foco maior era dado a assuntos mais complexos. Seguindo um  conceito apresentado e utilizado por meus professores do ensino médio, a lógica  de programação é mais importante que qualquer linguagem. Quem aprende a  programar uma vez fica mais apto a aprender outras linguagens de programação.  É nessa lógica de programação para iniciantes que este livro se concentra, apre sentando recursos do Python sempre que possível. O intuito é de iniciar o leitor  no mundo da programação e prepará-lo para cursos e conceitos mais avançados.  Acredito que depois de ler e estudar este livro, você estará apto a ler outras obras  de programação e a aprender novas linguagens por conta própria.

**15**

**Introdução**

Este livro foi escrito com o iniciante em programação em mente. Embora a  linguagem Python seja muito poderosa e rica em recursos modernos de progra mação, este livro não pretende ensinar apenas a linguagem em si, mas ensinar a  programar em qualquer linguagem. Alguns recursos da linguagem Python não  foram utilizados. O objetivo foi privilegiar os exercícios de lógica de programação  e melhor preparar o leitor para outras linguagens. Essa escolha não impediu que  recursos poderosos da linguagem fossem apresentados, mas este livro não é uma  referência da linguagem Python.

Os capítulos foram organizados de forma a apresentar progressivamente os con ceitos básicos de programação. A leitura do livro, feita perto de um computador  com o interpretador Python aberto, é extremamente recomendada para facilitar  a experimentação dos exemplos propostos.

Cada capítulo traz exercícios organizados de forma a explorar o conteúdo apresen tado. Alguns exercícios apenas modificam os exemplos do livro, enquanto outros  requerem a aplicação dos conceitos apresentados na criação de novos programas.  Tente resolver os exercícios na medida em que são apresentados. Embora seja  impossível não falar de matemática em um livro de programação, os exercícios  foram elaborados para o nível de conhecimento de um aluno do ensino médio,  e utiliza problemas comerciais ou do dia a dia. Essa escolha não foi feita para  evitar o estudo de matemática, mas para não misturar a introdução de conceitos  de programação com novos conceitos matemáticos.

É recomendado que você organize os programas gerados, com uma pasta (diretó rio) por capítulo, de preferência adicionando o número do exemplo ou exercício  aos nomes dos arquivos. Alguns exercícios alteram outros exercícios, mesmo  em capítulos diferentes. Uma boa organização desses arquivos vai facilitar seu  trabalho de estudo.

Um apêndice foi preparado para ajudar a entender as mensagens de erro que  podem ser geradas pelo interpretador Python.

**16**

**Introdução 17**

O uso de Python também libera alunos e professores para utilizar o sistema ope racional de sua escolha, seja Windows, Linux ou Mac OS X. Todos os exemplos do  livro requerem apenas a distribuição padrão da linguagem, que é disponibilizada  gratuitamente.

Embora todo esforço tenha sido realizado para evitar erros e omissões, não há  garantias de que o livro esteja isento de erros. Se você encontrar falhas no conteúdo,  envie um e-mail para *erros@nilo.pro.br*. Em caso de dúvidas, embora eu não possa  garantir resposta a todos os e-mails, envie sua mensagem para *duvidas@nilo.pro.br*.  Dicas, críticas e sugestões podem ser enviadas para *professores@nilo.pro.br*. O  código-fonte e eventuais correções deste livro podem ser encontrados no site  *http://python.nilo.pro.br*.

Um resumo do conteúdo de cada capítulo é apresentado a seguir:

**Capítulo 1 – Motivação**: visa a apresentar o desafio de aprender e estimular o estudo da  programação de computadores, apresentando problemas e aplicações do dia a dia.

**Capítulo 2 – Preparação do ambiente**: instalação do interpretador Python, introdução  ao editor de textos, apresentação do IDLE, ambiente de execução, como digitar  programas e fazer os primeiros testes com operações aritméticas no interpretador.

**Capítulo – 3 Variáveis e entrada de dados**: tipos de variáveis, propriedades de cada tipo,  operações e operadores. Apresenta o conceito de programa no tempo e uma téc nica simples de rastreamento. Entrada de dados pelo teclado, conversão de tipos  de dados e erros comuns.

**Capítulo 4 – Condições**: estruturas condicionais, conceitos de blocos e seleção de linhas  a executar com base na avaliação de expressões lógicas.

**Capítulo 5 – Repetições**: estrutura de repetição while, contadores, acumuladores. Apre senta o conceito de repetição da execução de um bloco e de repetições aninhadas.

**Capítulo 6 – Listas**: operações com listas, ordenação pelo método de bolhas, pesquisa,  utilização de listas como pilhas e filas.

**Capítulo 7 – Trabalhando com strings**: apresenta operações avançadas com strings. Explora  a classe string do Python. O capítulo traz também um jogo simples para fixar os  conceitos de manipulação de strings.

**Capítulo 8 – Funções**: noção de função e transferência de fluxo, funções recursivas,  funções lambda, parâmetros, módulos. Apresenta números aleatórios.

**18 Introdução à Programação com Python**

**Capítulo 9 – Arquivos**: criação e leitura de arquivos em disco. Geração de arquivos  HTML em Python, operações com arquivos e diretórios, parâmetros pela linha  de comando, caminhos.

**Capítulo 10 – Classes e objetos**: introdução à orientação a objetos. Explica os conceitos  de classe, objetos, métodos e herança. Prepara o aluno para continuar estudando  o tópico e melhor compreender o assunto.

**Capítulo 11 – Banco de dados**: introdução à linguagem SQL e ao banco de dados SQLite.

**Capítulo 12 – Próximos passos**: capítulo final, que lista os próximos passos em diversos  tópicos, como jogos, sistemas web, programação funcional, interfaces gráficas e  bancos de dados. Visa a apresentar livros e projetos open source pelos quais o  aluno pode continuar estudando, dependendo de sua área de interesse.

**Apêndice A – Mensagens de erro**: explica as mensagens de erro mais frequentes em  Python, mostrando suas causas e como resolvê-las.

**capítulo 1**

**Motivação**

*Então, você quer aprender a programar?*

Programar computadores é uma tarefa que exige tempo e dedicação para ser cor retamente aprendida. Muitas vezes não basta só estudar e fazer os exemplos, mas  também deixar a mente se acostumar com a nova forma de pensar. Para muitas  pessoas, o mais difícil é continuar gostando de programar. Elas desistem nas  primeiras dificuldades e não voltam mais a estudar. Outras são mais pacientes,  aprendem a não se irritar com a máquina e a assumir seus erros.

Para não sofrer dos males de quem não aprendeu a programar, você precisa res ponder a algumas perguntas antes de começar:

**1**. Você quer aprender a programar?

**2**. Como está seu nível de paciência?

**3**. Quanto tempo você pretende estudar?

**4**. Qual o seu objetivo ao programar?

**1.1 Você quer aprender a programar?**

Responda a essa questão, mas pense um pouco antes de chegar à resposta final. A  maneira mais difícil de aprender a programar é não querer programar. A vontade  deve vir de você, e não de um professor ou de um amigo. Programar é uma arte  e precisa de dedicação para ser dominada. Como tudo o que é desconhecido, é  muito difícil quando não a entendemos, mas se torna mais simples à medida que  a aprendemos.

**19**

**20 Introdução à Programação com Python**

Se você já decidiu aprender a programar, siga para a próxima parte. Se ainda  não se convenceu, continue lendo. Programar pode tornar-se um novo *hobby* e  até mesmo uma profissão. Se você estuda computação, precisa saber programar.  Isso não significa que você será um programador por toda a vida, ou que a pro

gramação limitará seu crescimento dentro da área de informática. Uma desculpa  que já ouvi muitas vezes é “*eu sei programar, mas não gosto”*. Vários alunos de  computação terminam seus cursos sem saber programar; isto é, sem realmente  saber programar. Programar é como andar de bicicleta, você não se esquece, mas  só aprende fazendo. Ao trocar de uma linguagem de programação para outra, se  você realmente aprendeu a programar, terá pouca dificuldade para aprender a  nova linguagem. Diferentemente de saber programar, a sintaxe de uma linguagem  de programação é esquecida muito facilmente. Não pense que saber programar  é decorar todos aqueles comandos, parâmetros e nomes estranhos. Programar é  saber utilizar uma linguagem de programação para resolver problemas, ou seja,  saber expressar uma solução por meio de uma linguagem de programação.

**1.2 Como está seu nível de paciência?**

Seja paciente.

Outro erro de quem estuda programação é querer fazer coisas difíceis logo de  início.

Qual será seu primeiro programa? Um editor de textos? Uma planilha eletrônica?  Uma calculadora?

Não! Será algo bem mais simples... Como somar dois números. É isso mesmo: somar dois números!

Com o tempo, a complexidade e o tamanho dos programas aumentarão. Seja paciente.

Programar exige muita paciência e, principalmente, atenção a detalhes. Uma  simples vírgula no lugar de um ponto ou aspas esquecidas podem arruinar seu  programa. No início, é comum perder a calma ou mesmo se desesperar até aprender  a ler o que realmente escrevemos em nossos programas. Nessas horas, paciência  nunca é demais. Leia novamente a mensagem de erro ou pare para entender o  que não está funcionando corretamente. Nunca pense que o computador está  contra você, nem culpe o dia ou o destino.

Seja paciente.

**Capítulo 1** ■ **Motivação 21 1.3 Quanto tempo você pretende estudar?**

Pode-se aprender a programar em poucas horas. Se você é o sujeito que progra ma o micro-ondas da sua tia, que abre vidros de remédio lendo as instruções da  tampa ou que brincou de Lego, programar é seu segundo nome.

Todos nós já programamos algo na vida, nem que seja ir ao cinema no sábado. A  questão é: quanto tempo você dedicará para aprender a programar computadores?

Como tudo na vida, nada de exageros. Na realidade, tanto o tempo como a forma  de estudar variam muito de pessoa para pessoa. Algumas rendem mais estudando  em grupo. Outras gostam de ter aulas.

O importante é incluir o estudo da programação em seu estilo preferido. Não  tente aprender tudo ou entender tudo rapidamente. Se isso ocorrer, parabéns, há  muito pela frente. Caso contrário, relaxe. Se não entender na segunda tentativa,  pare e volte a tentar amanhã.

Quando encontrar um problema, tenha calma. Veja o que você escreveu. Verifique  se você entende o que está escrito. Um erro comum é querer programar sem saber  escrever as instruções. É como querer escrever sem saber falar.

Inicie o estudo com sessões de uma ou no máximo duas horas por dia. Depois  ajuste esse tempo a seu ritmo.

**1.4 Programar para quê?**

Se você não precisa programar para seu trabalho ou estudo, vejamos algumas  outras razões:

**1.4.1 Escrever páginas web**

Hoje, todos estão expostos à web, à Internet e a seus milhares de programas. A  web só funciona porque permite a publicação de páginas e mais páginas de textos  e imagens com apenas um editor de textos. A mais complexa página que você já  acessou é um conjunto de linhas de texto reunidas para instruir um programa,  o navegador (*browser*), sobre como apresentar seu conteúdo.

**22 Introdução à Programação com Python**

**1.4.2 Acertar seu relógio**

Você conhece aquelas pessoas que nunca aprenderam a acertar seus relógios?  Consigo me lembrar de várias delas...

Seguir instruções é muito importante para tarefas tão simples quanto essas. A  sequência de passos para ajustar as horas, minutos e até mesmo a data de seu  relógio podem ser encaradas como um programa. Normalmente, aperta-se o botão  de ajuste até que um número comece a piscar. Depois, você pode usar um botão  para mudar a hora ou ir direto para o ajuste dos minutos. Isso se repete até que  você tenha ajustado todos os valores como segundos, dia, mês e, às vezes, o ano.

**1.4.3 Aprender a usar mapas**

Já se perdeu em uma cidade estranha? Já fez uma lista de passos para chegar a  algum lugar? Então você já programou antes. Só de procurar um mapa você já  mereceria um prêmio. Ao traçar um caminho de onde você está até onde deseja  chegar, você normalmente relaciona uma lista de ruas ou marcos desse caminho.  Normalmente é algo como passar três ruas à esquerda, dobrar à direita, dobrar  à esquerda... Ou algo como seguir reto até encontrar um sinal de trânsito ou um  rio. Isso é programar. Seguir seu programa é a melhor forma de saber se o que  você escreveu está correto ou se você agora está realmente perdido.

**1.4.4 Mostrar para seus amigos que você sabe programar**

Esta pode ser a razão mais complicada. Vamos ver isso como um subproduto do  aprendizado, e não seu maior objetivo. Se essa for a razão de aprender a programar,  é melhor continuar lendo e arranjar outra.

Programar é um esforço para você realizar algo. É uma tarefa que exige dedicação  e que traz muita satisfação pessoal. Seus programas podem ser legais, mas aí já  serão seus programas e não somente você.

**1.4.5 Parecer estranho**

Agora estamos conversando. Entender o que milhares de linhas de programa  significam pode realmente gerar uma fama como essa entre os leigos. Se esse é seu  objetivo, saiba que há maneiras mais fáceis, como parar de tomar banho, deixar

**Capítulo 1** ■ **Motivação 23**

as unhas crescerem, ter um cabelo de cor laranja ou roxa, parecer um roqueiro  sem nunca ter tocado em uma banda etc.

Embora boa parte dos programadores que conheço não seja exatamente o que  eu classifico de 100% normal, ninguém o é.

Saber programar não significa que você seja louco ou muito inteligente. Saber  programar também não significa que você não seja louco ou que não seja muito  inteligente. Imagine aprender a programar como qualquer outra coisa que você  já aprendeu.

Um dia, sua mente poderá ter pensamentos estranhos, mas, quando esse dia  chegar, pode apostar que você saberá.

**1.4.6 Entender melhor como seu computador funciona**

Programando você pode começar a entender por que aquela operação falhou  ou por que o programa simplesmente fechou sem nem mesmo avisar o motivo.

Programar também pode ajudá-lo a utilizar melhor sua planilha ou editor de  textos. O tipo de pensamento que se aprende programando servirá não só para  fazer programas, mas também para usar programas.

**1.4.7 Cozinhar**

Uma vez precisei cozinhar um prato, mas as instruções estavam escritas em ale mão. Não sei nada de alemão. Peguei o primeiro dicionário e comecei a traduzir  as principais palavras. Com as palavras traduzidas, tentei entender o que deveria  realmente fazer.

Naquela noite, o jantar foi apenas uma sopa instantânea. Receitas podem ser vis tas como programas. E como programas, só é possível segui-las se você entender  aquilo que foi escrito.

A simples sequência de instruções não ajuda uma pessoa que não entenda seus  efeitos.

Para algumas pessoas, programar é mais fácil que aprender alemão (ou qualquer  outro idioma estrangeiro). E como qualquer outra língua, não se aprende apenas  com um dicionário.

**24 Introdução à Programação com Python**

Idiomas humanos são ricos em contextos, e cada palavra costuma ter múltiplos  significados. A boa notícia: linguagens de programação são feitas para que má quinas possam entender o que ali foi representado. Isso significa que entender um  programa é muito fácil, quase como consultar um dicionário. Outra boa notícia  é que a maioria das linguagens contém conjuntos pequenos de “palavras”.

**1.4.8 Salvar o mundo**

Uma boa razão para aprender a programar é salvar o mundo. Isso mesmo!

Todos os dias, milhares de quilos de alimento são desperdiçados ou não chegam  onde deveriam por falta de organização. Programando você pode ajudar a criar  sistemas e até mesmo programas que ajudem outras pessoas a se organizar.

Outra boa ação é ajudar um projeto de software livre. Isso permitirá que muitas  pessoas que não podem pagar por programas de computador se beneficiem deles  sem cometer crime algum.

**1.4.9 Software livre**

Aliás, você tem licença de uso para todos os seus programas?

Se a resposta é não, saiba que programadores aprendem Linux e outros sistemas  operacionais muito mais rapidamente. Também conseguem tirar maior proveito  desses sistemas porque conseguem programá-los.

Se não existe, crie. Se é ruim, melhore.

Separar a programação em um mundo à parte pode ser sua primeira ideia estra nha entre muitas.

Criar mundos dentro de programas e computadores pode ser a segunda.

**1.5 Por que Python?**

A linguagem de programação Python é muito interessante como primeira lin guagem de programação devido à sua simplicidade e clareza. Embora simples, é  também uma linguagem poderosa, podendo ser usada para administrar sistemas  e desenvolver grandes projetos. É uma linguagem clara e objetiva, pois vai direto  ao ponto, sem rodeios.

**Capítulo 1** ■ **Motivação 25**

Python é software livre, ou seja, pode ser utilizada gratuitamente, graças ao tra balho da Python Foundation1 e de inúmeros colaboradores. Você pode utilizar  Python em praticamente qualquer arquitetura de computadores ou sistema ope racional, como Linux2, FreeBSD3, Microsoft Windows ou Mac OS X4.

Python vem crescendo em várias áreas da computação, como inteligência artifi cial, banco de dados, biotecnologia, animação 3D, aplicativos móveis (celulares),  jogos e mesmo como plataforma web. Isso explica porque Python é famosa por  ter “*batteries included*”, ou seja, baterias inclusas, fazendo referência a um produto  completo que pode ser usado prontamente (quem nunca ganhou um presente  de Natal que veio sem pilhas?). Hoje é difícil encontrar uma biblioteca que não  tenha *bindings* (versão) em Python. Esse fato torna o aprendizado da linguagem  muito mais interessante, uma vez que aprender a programar em Python é poder  continuar a utilizar os conhecimentos adquiridos mesmo depois de aprender a  programar para resolver problemas reais.

Uma grande vantagem de Python é a legibilidade dos programas escritos nessa  linguagem. Outras linguagens de programação utilizam inúmeras marcações,  como ponto (.) ou ponto e vírgula (;), no fim de cada linha, além dos marcadores  de início e fim de bloco como chaves ({ }) ou palavras especiais (begin*/*end). Esses  marcadores tornam os programas um tanto mais difíceis de ler e felizmente não  são usados em Python. Veremos mais sobre blocos e marcações nos capítulos  seguintes.

Outro bom motivo para aprender Python é poder obter resultados em pouco  tempo. Como Python é uma linguagem completa, contando com bibliotecas  para acessar bancos de dados, processar arquivos XML, construir interfaces grá ficas e mesmo jogos, podemos utilizar muitas funções já existentes escrevendo  poucas linhas de código. Isso aumenta a produtividade do programador, pois ao  utilizarmos bibliotecas usamos programas desenvolvidos e testados por outras  pessoas. Isso reduz o número de erros e permite que você se concentre realmente  no problema que quer resolver.

Vejamos um pequeno programa escrito em Python na listagem 1.1. ► **Listagem 1.1 – Programa Olá Mundo**

**print** ("Olá!")

1 *http://www.python.org*

2 *http://www.kernel.org* ou *http://www.ubuntu.com* para obter o pacote completo 3 *http://www.freebsd.org*

4 *http://www.apple.com/macosx*

**26 Introdução à Programação com Python**

A listagem do programa 1.1 tem apenas uma linha de código. A palavra **print** é uma  função utilizada para enviar dados para a tela do computador. Ao escrevermos  **print** ("Olá"), ordenamos ao computador que exiba o texto “Olá!” na tela. Veja o  que seria exibido na tela ao se executar esse programa no computador:

Olá!

Observe que as aspas (") não aparecem na tela. Esse é um dos detalhes da pro gramação: precisamos marcar ou limitar o início e o fim de nossas mensagens  com um símbolo, nesse caso, aspas. Como podemos exibir praticamente qualquer  texto na tela, as primeiras aspas indicam o início da mensagem, e as seguintes,  o fim. Ao programar, não podemos esquecer as limitações do computador. Um  computador não interpreta textos como seres humanos. A máquina não conse gue diferenciar o que é programa ou mensagem. Se não utilizarmos as aspas, o  computador interpretará nossa mensagem como um comando da linguagem  Python, gerando um erro.

O interpretador Python é uma grande ferramenta para o aprendizado da lingua gem. O interpretador é o programa que permite digitar e testar comandos escritos  em Python e verificar os resultados instantaneamente. Veremos como utilizar o  interpretador na seção 2.2.

A linguagem Python foi escolhida para este livro por simplificar o trabalho de  aprendizado e fornecer grande poder de programação. Como é um software livre,  disponível praticamente para qualquer tipo de computador, sua utilização não  envolve a aquisição de licenças de uso, muitas vezes a um custo proibitivo.

Bem-vindo ao mundo da programação!

**capítulo 2**

**Preparando o ambiente**

Antes de começarmos, precisamos instalar o interpretador da linguagem Python.  O interpretador é um programa que aceita comandos escritos em Python e os  executa, linha a linha. É ele quem vai traduzir nossos programas em um formato  que pode ser executado pelo computador. Sem o interpretador Python, nossos  programas não podem ser executados, sendo considerados apenas como texto.  O interpretador também é responsável por verificar se escrevemos corretamente  nossos programas, mostrando mensagens de erro, caso encontre algum problema.

O interpretador Python não vem instalado com o Microsoft Windows: você  deverá instalá-lo fazendo um download da internet. Se você utiliza Mac OS X ou  Linux, provavelmente isso já foi feito. Veja, a seguir, como verificar se você tem a  versão correta.

**2.1 Instalação do Python**

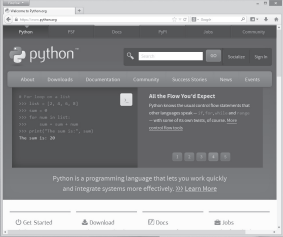
Neste livro, usamos o Python versão 3.4. A linguagem sofreu diversas modificações  entre as versões 2 e 3. A versão 3.4 foi escolhida por permitir a correta utilização  dos novos recursos, além do fato de que é interessante aprendermos a versão mais  nova. A versão 2.7 é também muito interessante, mas permite misturar a sintaxe  (forma de escrever) do Python 2 com a do Python 3. Para evitar complicações  desnecessárias, apresentaremos apenas as novas formas de escrever, e deixaremos  o Python 3.4 verificar se fizemos tudo corretamente. O Python 3.4 ainda não es tava disponível para todas as distribuições Linux na época de lançamento desta  edição. Você pode utilizar também o Python 3.3 com todos os exemplos do livro.

**27**

**28 Introdução à Programação com Python**

**2.1.1 Windows**

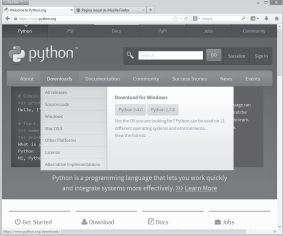
Python está disponível em todas as versões de Windows. Como é um software  livre, podemos baixar o interpretador Python gratuitamente no site *http://www. python.org*. O site deve parecer com o da figura 2.1. Veja que, no centro da página,  temos a opção **Downloads**. Passe o mouse sobre **Downloads** e espere o menu abrir.

*Figura 2.1 – Site da Python Foundation.*

A figura 2.2 mostra o menu da página de download aberto. Procure o texto Python  3.4.0. Se uma versão mais nova da série 3 estiver disponível, você deve escolhê-la.  Quando este livro foi escrito, a versão mais nova era a 3.4.0. Clique no botão para  iniciar o download do arquivo.

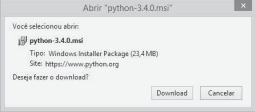
Se você não estiver utilizando o Firefox, o menu de download pode não aparecer.  Nesse caso, clique em Windows e siga as instruções nas páginas que irão se abrir.  Se sua versão de Windows é Windows Vista, 7, 8 ou 8.1 32 bits, escolha Python  3.4.0 Windows x86 MSI. Caso utilize o Windows Vista, 7, 8 ou 8.1 64 bits, escolha  Python 3.4.0 Windows x86-64 MSI Installer. Se você não conseguir encontrar esses  arquivos, tente digitar o endereço completo da página de download: *https://www.*

*python.org/downloads/release/python-340/*.

**Capítulo 2** ■ **Preparando o ambiente 29 ***Figura 2.2 – Página de download.*

Nos exemplos a seguir, faremos o download da versão 3.4.0 para Windows 8.1. A  versão do Python pode mudar, e o nome do arquivo, também. Se você escolheu  a versão de 32 ou 64 bits, o nome do arquivo será um pouco diferente, mas não  se preocupe: se você escolher a versão errada, ela simplesmente não funcionará.  Você pode então tentar outra versão.

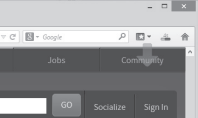
A figura 2.3 mostra a janela de download do Firefox. Essa janela também pode  variar de acordo com a versão de seu navegador de internet. Clique no botão  **download** para iniciar a transferência do arquivo. Aqui, o Firefox foi utilizado como  exemplo, mas você pode utilizar qualquer outro navegador de internet, como  Internet Explorer, Google Chrome, Opera ou Safari.



*Figura 2.3 – Janela de confirmação de download do Firefox.*

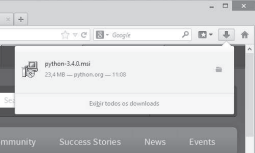
**30 Introdução à Programação com Python**

O interpretador Python ocupa aproximadamente 24 MB em disco (antes da insta lação), e a transferência do arquivo deve demorar alguns minutos, dependendo da  velocidade de conexão à internet. O Firefox exibe o download como na figura 2.4.



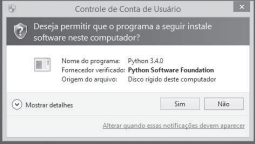
*Figura 2.4 – Janela de download com arquivo python-3.4.0.msi sendo baixado.*

Veja que a barra de transferência desaparece quando a transferência é concluída,  como na figura 2.5.



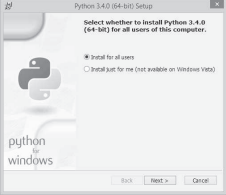
*Figura 2.5 – Janela de download com a transferência do arquivo finalizada.*

Clique no nome do arquivo que você acabou de baixar. Dependendo da versão do  seu Windows e de seu navegador de internet, você poderá receber um pedido de  confirmação do Firefox ou do Windows. Caso o Firefox pergunte se deseja executar  o programa, escolha executar. O Windows também pode pedir uma confirmação  de instalação, ilustrada na figura 2.6. Escolha Sim para aprovar a instalação. Essa  janela pode aparecer um pouco depois do início da instalação. Verifique também  se um ícone parecido com um escudo colorido começar a piscar; nesse caso, clique  no ícone para que a janela de permissão apareça.

**Capítulo 2** ■ **Preparando o ambiente 31 **

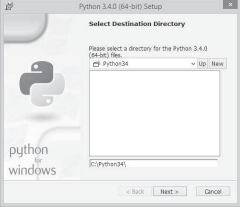
*Figura 2.6 – Janela de confirmação da execução do instalador Python.*

A figura 2.7 mostra a tela inicial do instalador. Até agora, não instalamos nada. É  esse programa que vai instalar o Python 3 em seu computador. Clique no botão  **Next** para continuar.



*Figura 2.7 – Janela do instalador do Python.*

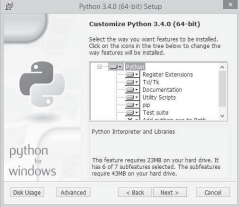
Você deve estar com uma janela parecida com a da figura 2.8. Essa janela permite  escolher onde instalar os arquivos do interpretador Python. Você pode mudar a  pasta ou simplesmente aceitar o local proposto (padrão). Clique no botão **Next** para continuar.

**32 Introdução à Programação com Python **

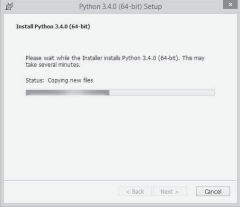
*Figura 2.8 – Janela do instalador pedindo confirmação da pasta (diretório) de  instalação do Python.*

A janela da figura 2.9 permite selecionar que partes do pacote do Python quere mos instalar. Essa janela é útil para utilizadores mais avançados da linguagem,  mas vamos simplesmente aceitar o padrão, clicando no botão **Next** para continuar.

Finalmente começamos a instalar os arquivos que precisamos. Uma barra de  progresso como a da figura 2.10 será exibida. Aguarde a finalização do processo.



*Figura 2.9 – Janela de seleção de pacotes do instalador Python.*

**Capítulo 2** ■ **Preparando o ambiente 33 **

*Figura 2.10 – Janela do instalador exibindo o progresso da instalação.*

A instalação foi concluída. Clique no botão **Finish**, mostrado na figura 2.11, para  terminar o instalador.



*Figura 2.11 – Janela de finalização do instalador.*

**34 Introdução à Programação com Python**

**2.1.2 Linux**

Grande parte das distribuições Linux inclui uma versão do interpretador Python;  porém, as versões mais utilizadas são a 2.6 ou a 2.7. Precisamos do interpretador na  versão 3.3 ou superior. Para verificar a versão de seu interpretador, digite python -V na linha de comando.

No Ubuntu (versão 14.04), digite:

sudo apt-get install python3.4

sudo apt-get install idle-python3.4

Você precisa de permissões de administrador (root) para efetuar a instalação. Se  a versão 3.4 ainda não estiver disponível para instalação via apt-get, você pode  utilizar a versão 3.3 sem problemas. Consulte o site do livro para mais detalhes.

**2.1.3 Mac OS X**

Os Mac OS X Leopard, Snow Leopard, Lion, Mountain Lion, Maverick e Yosemite  vêm com uma versão do interpretador Python da Apple. No entanto, essa versão não  é a 3.4. Para contornar o problema, instale o MacPorts (*http://www.macports.org/*),  fazendo o download do arquivo dmg, e, depois, instale o Python 3.4 com:

sudo port install python34

Para executar o interpretador recém-instalado, digite python3.4 na linha de coman do. Você também pode utilizar outros gerenciadores de pacote disponíveis para  Mac OS X, como Fink (*http://www.finkproject.org/*) e Homebrew (*http://brew.sh/*).  Uma versão instalável, em formato dmg, também está disponível no site da  *http://www.python.org*.

**2.2 Usando o interpretador**

Com o Python instalado, vamos começar a trabalhar.

O IDLE é uma interface gráfica para o interpretador Python, permitindo também  a edição e execução de nossos programas. No Windows Vista e no Windows 7,  você deve ter uma pasta no menu **Iniciar > Programas > Python 3.4**. Escolha **IDLE**. No  Windows 8 e 8.1, procure por Python 3.4 na lista de aplicativos e, então, por IDLE  (Python GUI).

**Capítulo 2** ■ **Preparando o ambiente 35**

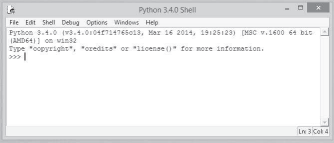
No Linux, abra o terminal e digite:

idle-python3.4 &

No Mac OS X, abra o terminal e digite:

IDLE3.4 &

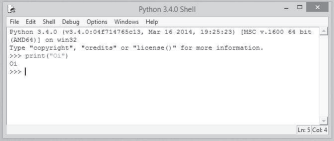
A janela inicial do IDLE, no Windows 8.1, é mostrada na figura 2.12. Se você utiliza  o Mac OS X, Linux ou uma versão diferente de Windows, essa janela não será  exatamente igual à da figura, mas muito parecida.

*Figura 2.12 – Janela inicial do IDLE.*

Observe que o cursor está posicionado dentro da janela Python Shell, e que a  linha de comandos é iniciada pela sequência >>>.

Digite:

**print**("Oi")

Pressione a tecla **ENTER**. Você deve obter uma tela parecida com a da figura 2.13. *Figura 2.13 – IDLE mostrando o resultado de print(“Oi”).*

**36 Introdução à Programação com Python**

Uma das grandes vantagens do Python é contar com o interpretador de coman dos. Você pode abrir uma janela como essa sempre que tiver alguma dúvida em  Python. O interpretador permite que você verifique o resultado de um comando  instantaneamente. Veremos mais adiante que esse tipo de verificação é muito  importante e facilita o aprendizado da linguagem.

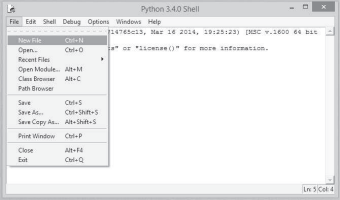
**2.3 Editando arquivos**

Nem só de experimentos vive o programador Python. Um programa nada mais  é que um arquivo-texto, escrito em um formato especial (linguagem).

No caso da linguagem Python, podemos usar qualquer editor de textos disponível:  PSPad ou Sublime, no Windows; TextMate, no Mac OS X; Vim ou Emacs, no Linux.

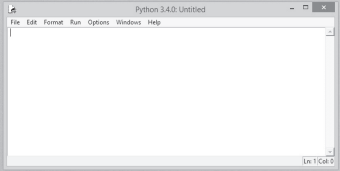
O que você não deve utilizar é um editor de textos como o Microsoft Word ou o  OpenOffice. Esses programas gravam seus arquivos em formatos especiais que  não podem ser utilizados para escrever programas, salvo se você escolher gravar  apenas texto ou simplesmente gravar no formato txt. Além disso, um editor de  textos comum não foi feito para escrever programas.

Você pode também utilizar o editor de textos incluído na instalação do interpre tador Python. Com o interpretador aberto, clique no menu **File** e depois selecione  a opção **New File**, como mostra a figura 2.14.

*Figura 2.14 – IDLE com o menu File aberto e a opção New File selecionada.*

**Capítulo 2** ■ **Preparando o ambiente 37**

Como utilizamos a versão 3.4 do interpretador, é muito importante que você use  um editor de textos correto. Isso é, um editor que suporte a edição de textos em  UTF-8. Se você não sabe o que é UTF-8, ou se o seu editor de textos suporta esse  formato de codificação, utilize o IDLE como editor de textos. Além de suportar a  edição no formato UTF-8, que permitirá a utilização de acentos em nossos pro

gramas, ele é preparado para os colorir em Python, tornando a leitura mais fácil. *Figura 2.15 – Janela do editor de textos do IDLE.*

Uma nova janela, como a da figura 2.15, deverá aparecer. É nessa janela que escre veremos nossos programas. Observe que, embora parecida com a janela principal  do IDLE, a janela tem opções de menu diferentes da outra. Para esclarecer essa  separação, chamaremos a primeira de janela do interpretador; e a segunda, de  janela do editor de textos. Se você ainda estiver em dúvida, a janela do editor de  textos é a que apresenta a opção **Run** no menu.

Experimente um pouco, escrevendo:

**print**("Oi")

Sua janela do editor de textos deverá ser semelhante à mostrada na figura 2.16.

**38 Introdução à Programação com Python** *Figura 2.16 – Editor de textos com o programa já digitado.*

Agora que escrevemos nosso pequeno programa, vamos salvá-lo. Escolha a opção  **Save** do menu **File**, como mostra a figura 2.17.

*Figura 2.17 – Janela do editor de textos com o menu File aberto e a opção Save  selecionada.*

Uma janela-padrão de gravação de arquivos será exibida. A janela muda com a  versão do sistema operacional, mas se você utiliza Windows 8.1, ela parecerá com  a da figura 2.18. Escreva *teste.py* no nome do arquivo e clique no botão para salvar.  Atenção: a extensão *.py* não é adicionada automaticamente pelo IDLE. Lembre-se  de sempre gravar seus programas escritos em Python com a extensão *.py*.

**Capítulo 2** ■ **Preparando o ambiente 39** *Figura 2.18 – Janela de gravação de arquivos do Windows 8.1.*

Agora clique no menu **Run** para executar seu programa. Você pode também  pressionar a tecla F5 com o mesmo efeito. Essa operação pode ser visualizada na  figura 2.19.

*Figura 2.19 – Janela do interpretador mostrando a opção Run Module selecionada.*

Seu programa será executado na outra janela, a do interpretador. Veja que uma  linha com a palavra RESTART apareceu, como na figura 2.20. Observe que obti vemos o mesmo resultado de nosso primeiro teste.

**40 Introdução à Programação com Python** *Figura 2.20 – Janela do interpretador mostrando a execução do programa.*

Experimente modificar o texto entre aspas e executar novamente o programa,  pressionando a tecla **F5** na janela do Editor ou selecionando **Run** no menu.

Agora estamos com tudo instalado para continuar aprendendo a programar. Du rante todo o livro, programas de exemplo serão apresentados. Você deve digitá-los  na janela do editor de textos, gravá-los e executá-los. Você pode voltar a ler as  seções 2.2 e 2.3 sempre que precisar. Com o tempo, essas tarefas serão memori zadas e realizadas sem esforço.

Experimente gravar seus programas em um diretório específico em seu compu tador. Isso ajudará a encontrá-los mais tarde. Uma dica é criar um diretório para  cada capítulo do livro, mas você pode organizar os exemplos como quiser. Você  pode também fazer cópias dos exemplos e alterá-los sem receio. Embora você  possa baixar todos os exemplos já digitados, é altamente recomendado que você  leia o livro e digite cada programa. Essa prática ajuda a memorizar as construções  da linguagem e facilita a leitura dos programas.

**2.4 Cuidados ao digitar seus programas**

Ao digitar um programa no editor de textos, verifique se você o copiou exatamente  como apresentado. Em Python, você deve tomar cuidado com os seguintes itens:

**1**. Letras maiúsculas e minúsculas são diferentes. Assim, **print** e Print são  completamente diferentes, causando um erro caso você digite o P maiúsculo.  No início, é comum termos o hábito de escrever a letra inicial de cada linha  em letra maiúscula, causando erros em nossos programas. Em caso de erro,  leia atentamente o que você digitou e compare com a listagem apresentada  no livro.

**Capítulo 2** ■ **Preparando o ambiente 41**

**2**. Aspas são muito importantes e não devem ser esquecidas. Toda vez que  abrir aspas, não se esqueça de fechá-las. Se você se esquecer, seu programa  não funcionará. Observe que o IDLE muda a cor do texto entre aspas,  facilitando essa verificação.

**3**. Parênteses não são opcionais em Python. Não remova os parênteses dos  programas e preste a mesma atenção dada às aspas para abri-los e fechá-los.  Todo parêntese aberto deve ser fechado.

**4**. Espaços são muito importantes. A linguagem Python se baseia na quantida de de espaço em branco antes do início de cada linha para realizar diversas  operações, explicadas posteriormente no livro. Não se esqueça de digitar  o texto dos programas com o mesmo alinhamento apresentado no livro.  Observe também que o IDLE ajuda nesses casos, avisando sobre problemas  de alinhamento. Nunca junte duas linhas em uma só até sentir-se seguro  sobre como escrever corretamente em Python.

**2.5 Os primeiros programas**

Vamos analisar nosso primeiro programa. A figura 2.21 mostra a separação entre o  nome da função, os parênteses, a mensagem e as aspas. É muito importante saber  o nome de cada uma dessas partes para o correto entendimento dos conceitos e  programas apresentados.

parênteses

mensagem

print ( " Oi " )

função aspas

*Figura 2.21 – Nomes dos elementos do primeiro programa.*

A função **print** informa que vamos exibir algo na tela. Pode-se dizer que a função  exibe uma mensagem na tela do computador. Sempre que quisermos mostrar  algo para o usuário do computador, como uma mensagem, uma pergunta ou o  resultado de uma operação de cálculo, utilizaremos a função **print**.

Para separar os programas do texto explicativo do livro, todo trecho de programa,  funções, variáveis e demais itens em Python serão apresentados com uma fonte  de texto diferente (monoespaçada), como a utilizada para explicar a função **print**.

**42 Introdução à Programação com Python**

Voltando à figura 2.21, temos as aspas como elementos importantes. Elas são  utilizadas para separar textos destinados ao usuário do computador do resto do  programa. Utilizamos aspas para indicar o início e o fim do texto de nossa men sagem. A mensagem é nosso texto em si e será exibida exatamente como digitada.  Os parênteses são utilizados para separar os parâmetros de uma função, no caso,  os de **print**. Um parâmetro é um valor passado para uma função: no caso da  função **print**, a mensagem a imprimir.

O interpretador Python também pode ser utilizado como calculadora. Experi mente digitar 2+3 no interpretador, como mostra a listagem 2.1.

► **Listagem 2.1 – Usando o interpretador como calculadora**

>>> 2+3

5

Não se esqueça de pressionar a tecla **Enter** para que o interpretador saiba que  você terminou a digitação. O resultado deve ser apresentado na linha seguinte.  Podemos também subtrair valores, como na listagem 2.2.

► **Listagem 2.2 – Subtração**

>>> 5-3

2

Podemos combinar adição e subtração na mesma linha, como mostra a listagem  2.3.

► **Listagem 2.3 – Adição e subtração**

>>> 10-4+2

8

A multiplicação é representada por um asterisco (\*); e a divisão, pela barra (/),  como mostra a listagem 2.4.

► **Listagem 2.4 – Multiplicação e divisão**

>>> 2\*10

20

>>> 20/4

5.0

**Capítulo 2** ■ **Preparando o ambiente 43**

Para elevar um número a um expoente, utilizaremos dois asteriscos (\*\*) para repre sentar a operação de exponenciação. Observe que não há qualquer espaço entre  os dois asteriscos. Assim, para calcularmos 23, escreveremos de forma semelhante  ao conteúdo da listagem 2.5.

► **Listagem 2.5 – Exponenciação**

>>> 2\*\*3

8

Podemos também obter o resto da divisão de dois números usando o símbolo  %. Assim, para calcularmos o resto da divisão entre 10 e 3, digitaríamos como  mostrado na listagem 2.6.

► **Listagem 2.6 – Resto da divisão inteira**

>>> 10 % 3

1

>>> 16 % 7

2

>>> 63 % 8

7

Os parênteses são utilizados em Python da mesma forma que em expressões  matemáticas, ou seja, para alterar a ordem de execução de uma operação. Para  relembrar a ordem de precedência das operações, temos as seguintes prioridades:

**1**. Exponenciação ou potenciação (\*\*).

**2**. Multiplicação (\*) e divisão (/ e %).

**3**. Adição (+) e subtração (-).

A expressão 1500 + (1500 \* 5 ∕ 100) é equivalente a: 

1500 +

1500 × 5 100

Não se esqueça de que, tanto em Python como na matemática, as operações de  mesma prioridade são realizadas da esquerda para direita. Utilize parênteses  sempre que precisar alterar a ordem de execução das operações e também para  aumentar a clareza da fórmula.

**44 Introdução à Programação com Python**

**Exercício 2.1** Converta as seguintes expressões matemáticas para que possam ser  calculadas usando o interpretador Python.

10 + 20 × 30

42 ÷ 30

(94 + 2) × 6 - 1

**Exercício 2.2** Digite a seguinte expressão no interpretador:

10 % 3 \* 10 \*\* 2 + 1 - 10 \* 4 / 2

Tente resolver o mesmo cálculo, usando apenas lápis e papel. Observe como a  prioridade das operações é importante.

**2.6 Conceitos de variáveis e atribuição**

Além de operações simples de cálculo, o interpretador também pode ser usado para  realizar operações mais complexas e mesmo executar programas completos. Antes  de continuarmos, é importante observar o conceito de variáveis e como podemos  usá-las em um programa. Em matemática, aprendemos o conceito de variável para  representar incógnitas em equações do tipo x + 1 = 2, onde devemos determinar  o valor de x, resolvendo a equação. Em programação, variáveis são utilizadas para  armazenar valores e para dar nome a uma área de memória do computador onde  armazenamos dados. Variáveis serão mais bem estudadas no capítulo 3. Por enquan to, podemos imaginar a memória do computador como uma grande estante, onde  cada compartimento tem um nome. Para armazenar algo nesses compartimentos,  usaremos o símbolo de igualdade (=) entre o nome do compartimento e o valor  que queremos armazenar. Chamaremos essa operação de atribuição, onde um  valor é atribuído a uma variável. Quando lermos nosso programa, as operações  de atribuição serão chamadas de “recebe”, ou seja, uma variável recebe um valor.

A fim de simplificar as explicações de como um programa funciona, utilizaremos  bolas pretas ❶ com números para relacionar uma determinada linha a um texto  explicativo. Esses símbolos não fazem parte do programa e não devem ser digi tados nem no interpretador nem no editor de textos.

Como quase tudo na vida, aprende-se a programar programando. Vamos escrever  outro programa. Observe a listagem 2.7:

**Capítulo 2** ■ **Preparando o ambiente 45**

► **Listagem 2.7 – O primeiro programa com variáveis**

a = 2 ❶

b = 3 ❷

**print** (a + b) ❸

Vejamos o que cada linha significa. Em ❶, temos a = 2. Leia “a recebe 2”. Essa linha  diz que uma variável chamada a receberá o valor 2. Variáveis em programação  têm o mesmo significado que em matemática. Você pode entender uma variável  como uma forma de guardar valores na memória do computador. Toda variável  precisa ter um nome para que seu valor possa ser utilizado posteriormente. Esse  conceito ficará mais claro um pouco mais adiante.

Em ❷, temos b = 3. Leia “b recebe 3”. Essa linha realiza um trabalho muito parecido  com o da linha anterior, mas a variável se chama b, e o valor é o número 3. Para  entender o que faz essa linha, imagine que criamos um espaço na memória do  computador para guardar outro valor, no caso, 3. Para podermos usar esse valor  mais tarde, chamamos esse espaço de “b”.

A linha ❸ solicita que o resultado da soma do conteúdo da variável a com o con teúdo da variável b seja exibido na tela. A função **print** realiza a impressão, mas,  antes, o resultado de a + b é calculado. Veja que nessa linha estamos ordenando  ao programa que calcule a + b e que exiba o resultado na tela. Como em matemá tica, passamos parâmetros ou valores para uma função usando parênteses. Esses  parênteses são requeridos pelo interpretador Python. Vale se lembrar de *f*(*x*); onde  f é o nome da função, e x um parâmetro. No exemplo anterior, **print** é o nome da  função; e o resultado de a + b, o valor passado como parâmetro. No decorrer deste  livro, veremos diversas funções disponíveis no Python para realizar operações  com o computador, como ler valores do teclado ou gravar dados em um arquivo.

Você pode experimentar o programa da listagem 2.7 na janela do interpretador  Python, como mostra a seção 2.2. O resultado desse programa pode ser visto na  listagem 2.8.

► **Listagem 2.8 – Exemplo mostrado no interpretador**

>>> a = 2

>>> b = 3

>>> **print** ( a + b )

5

As duas primeiras linhas não enviam nada para a tela; por isso, apenas o resultado  da terceira linha é mostrado.

**46 Introdução à Programação com Python**

Você pode estar-se perguntando por que criamos duas variáveis, a e b, para somar  dois números? Poderíamos ter obtido o mesmo resultado de diversas formas,  como na listagem 2.9.

► **Listagem 2.9 – Outra forma de resolver o problema**

**print** (2 + 3)

Ou mesmo como mostra a listagem 2.10.

► **Listagem 2.10 – Outra forma de resolver o problema**

**print** (5)

Então, por que escolhemos resolver o problema usando variáveis? Primeiramente,  para podermos falar de variáveis, mas também para exemplificar uma grande  diferença entre resolver um problema no papel e por meio de um computador.  Quando estamos resolvendo um problema de matemática no papel, como somar  dois números, realizamos diversos cálculos mentalmente e escrevemos parte desse  processo no papel, quando necessário. Depois de escrito no papel, mudar os va

lores não é tão simples. Ao programarmos um computador, estamos transferindo  esse cálculo para o computador. Como programar é descrever os passos para a  solução do problema, é aconselhável escrevermos programas o mais claramente  possível, de forma que possamos alterá-los caso precisemos e, mais importante,  que possamos entendê-los mais tarde.

Na listagem 2.9, o problema foi representado como sendo a soma de 2 e 3. Se  precisarmos mudar as parcelas dessa soma, teremos de escrever outro programa.  Isso também é válido para o primeiro programa, mas observe que, ao utilizar mos variáveis, estamos dando nome aos valores de entrada de nosso problema,  aumentando, assim, a facilidade de entendermos o que o programa faz.

Já a solução apresentada na listagem 2.10 não descreve o problema em si. Estamos  apenas ordenando ao computador que imprima o número 5 na tela. Não fizemos  qualquer registro do que estávamos fazendo, ou de que nosso problema era somar  dois números. Isso ficará mais claro no exemplo a seguir. Veja a listagem 2.11.

► **Listagem 2.11 – Cálculo de aumento de salário**

salário = 1500 ❶

aumento = 5 ❷

**print** (salário + (salário \* aumento / 100)) ❸

**Capítulo 2** ■ **Preparando o ambiente 47**

Em ❶ temos uma variável que é chamada salário, recebendo o valor 1500. Em ❷,  outra variável, aumento, recebe o valor 5. Finalmente, em ❸ descrevemos a fórmula  que calculará o valor do novo salário depois de receber um aumento. Teríamos,  então, um resultado como o da listagem 2.12.

► **Listagem 2.12 – Resultado do aumento de salário no interpretador**

>>> salário = 1500

>>> aumento = 5

>>> **print** (salário + (salário \* aumento / 100))

1575.0

O programa da listagem 2.11 pode ser escrito de forma mais direta, utilizando  outra fórmula sem variáveis. Veja a alternativa da listagem 2.13.

► **Listagem 2.13 – Alternativa para o cálculo de aumento de salário**

**print** (1500 + (1500 \* 5 / 100))

O objetivo desse exemplo é apresentar a diferença entre descrevermos o proble ma de forma genérica, separando os valores de entrada do cálculo. O resultado  é idêntico: a diferença está na clareza da representação de nosso problema. Se  mudarmos o valor do salário, na primeira linha da listagem 2.11, obteremos o re sultado correto na saída do programa, sem precisar nos preocuparmos novamente  com a fórmula do cálculo. Observe também que, se fizermos a mesma coisa no  programa da listagem 2.13, teremos que mudar o valor de salário em duas posições  diferentes da fórmula, aumentando nossas chances de nos esquecermos de uma  delas e, consequentemente, de recebermos um resultado incorreto.

Ao utilizarmos variáveis, podemos referenciar o mesmo valor várias vezes, sem nos  esquecer de que podemos utilizar nomes mais significativos que simples x ou y para aumentar a clareza do programa. Por exemplo, na listagem 2.11, registramos  a fórmula para o cálculo do aumento especificando o nome de cada variável,  facilitando a leitura e o entendimento.

Se você já utilizou uma planilha eletrônica, como Microsoft Excel ou OpenOffice  Calc, o conceito de variável pode ser entendido como as células de uma planilha  eletrônica. Você pode escrever as fórmulas de sua planilha sem utilizar outras  células, mas teria de reescrevê-las toda vez que os valores mudassem. Assim como  as células de uma planilha eletrônica, as variáveis de um programa podem ser  utilizadas diversas vezes e em lugares diferentes.

**48 Introdução à Programação com Python Exercício 2.3** Faça um programa que exiba seu nome na tela.

**Exercício 2.4** Escreva um programa que exiba o resultado de *2a × 3b*, onde *a* vale  3 e *b* vale 5.

**Exercício 2.5** Modifique o primeiro programa, listagem 2.7, de forma a calcular a  soma de três variáveis.

**Exercício 2.6** Modifique o programa da listagem 2.11, de forma que ele calcule um  aumento de 15% para um salário de R$ 750.

**capítulo 3**

**Variáveis e entrada de dados**

O capítulo anterior apresentou o conceito de variáveis, mas há mais por desco brir. Já sabemos que variáveis têm nomes que permitem acessar os valores dessas  variáveis em outras partes do programa. Neste capítulo, vamos ampliar nosso  conhecimento sobre variáveis, estudando novas operações e novos tipos de dados.

**3.1 Nomes de variáveis**

Em Python, nomes de variáveis devem iniciar obrigatoriamente com uma letra,  mas podem conter números e o símbolo sublinha (\_). Vejamos exemplos de nomes  válidos e inválidos em Python na tabela 3.1.

*Tabela 3.1 – Exemplo de nomes válidos e inválidos para variáveis*

|  |  |  |
| --- | --- | --- |
| **Nome** | **Válido** | **Comentários** |
| a1 | Sim | Embora contenha um número, o nome a1 inicia com letra. |
| velocidade | Sim | Nome formado por letras. |
| velocidade90 | Sim | Nome formado por letras e números, mas iniciado por letra. |
| salário\_médio | Sim | O símbolo sublinha (\_) é permitido e facilita a leitura de nomes grandes. |
| salário médio | Não | Nomes de variáveis não podem conter espaços em branco. |
| \_b | Sim | O sublinha (\_) é aceito em nomes de variáveis, mesmo no início. |
| 1a | Não | Nomes de variáveis não podem começar com números. |

A versão 3 da linguagem Python permite a utilização de acentos em nomes de  variáveis, pois, por padrão, os programas são interpretados utilizando-se um  conjunto de caracteres chamado UTF-8 (*http://pt.wikipedia.org/wiki/Utf-8*), capaz  de representar praticamente todas as letras dos alfabetos conhecidos.

**49**

**50 Introdução à Programação com Python**

Variáveis têm outras propriedades além de nome e conteúdo. Uma delas é conhe cida como tipo e define a natureza dos dados que a variável armazena. Python  tem vários tipos de dados, mas os mais comuns são números inteiros, números de  ponto flutuante e strings. Além de poder armazenar números e letras, as variáveis  em Python também armazenam valores como verdadeiro ou falso. Dizemos que  essas variáveis são do tipo lógico. Veremos mais sobre variáveis do tipo lógico na  seção 3.3.

|  |
| --- |
| **TRÍVIA**  A maior parte das linguagens de programação foi desenvolvida nos Es tados Unidos, ou considerando apenas nomes escritos na língua inglesa  como nomes válidos. Por isso, acentos e letras consideradas especiais  não são aceitos como nomes válidos na maior parte das linguagens  de programação. Essa restrição é um problema não só para falantes  do português, mas de muitas outras línguas que utilizam acentos ou  mesmo outros alfabetos. O padrão americano é baseado no conjunto de  caracteres ASCII (\*), desenvolvido na década de 1960. Com a globalização  da economia e o advento da internet, as aplicações mais modernas são  escritas para trabalhar com um conjunto de caracteres dito universal,  chamado Unicode (\*\*). |
| (\*) *http://pt.wikipedia.org/wiki/Ascii*  (\*\*) *http://pt.wikipedia.org/wiki/Unicode* |

**3.2 Variáveis numéricas**

Dizemos que uma variável é numérica quando armazena números inteiros ou  de ponto flutuante.

Os números inteiros são aqueles sem parte decimal, como 1, 0, -5, 550, -47, 30000.

Números de ponto flutuante ou decimais são aqueles com parte decimal, como  1.0, 5.478, 10.478, 30000.4. Observe que 1.0, mesmo tendo zero na parte decimal, é  um número de ponto flutuante.

Em Python, e na maioria das linguagens de programação, utilizamos o ponto, e  não a vírgula, como separador entre a parte inteira e fracionária de um número.  Essa é outra herança da língua inglesa. Observe também que não utilizamos nada  como separador de milhar. Exemplo: 1.000.000 (um milhão) é escrito 1000000.

**Capítulo 3** ■ **Variáveis e entrada de dados 51**

**Exercício 3.1** Complete a tabela a seguir, marcando inteiro ou ponto flutuante  dependendo do número apresentado.

|  |  |
| --- | --- |
| **Número** | **Tipo numérico** |
| 5 | 🔾 inteiro 🔾 ponto flutuante |
| 5.0 | 🔾 inteiro 🔾 ponto flutuante |
| 4.3 | 🔾 inteiro 🔾 ponto flutuante |
| -2 | 🔾 inteiro 🔾 ponto flutuante |
| 100 | 🔾 inteiro 🔾 ponto flutuante |
| 1.333 | 🔾 inteiro 🔾 ponto flutuante |

**3.2.1 Representação de valores numéricos**

Internamente, todos os números são representados utilizando o sistema binário,  ou seja, de base 2. Esse sistema permite apenas os dígitos 0 e 1. Para representar  números maiores, combinamos vários dígitos, exatamente como fazemos com o  sistema decimal, ou de base 10, que utilizamos.

Vejamos primeiro como isso funciona na base 10:

531 = 5 x 102 + 3 x 101 + 1 x 100

 = 5 x 100 + 3 x 10 + 1 x 1

 = 500 + 30 + 1

 = 531

Multiplicamos cada dígito pela base elevada a um expoente igual ao número de  casas à direita do dígito em questão. Como em 531 o 5 tem 2 dígitos à direita,  multiplicamos 5 × 102. Para o 3, temos apenas outro dígito à direita, logo, 3 × 101.  Finalmente, para o 1, sem dígitos à direita, temos 1 × 100. Somando esses compo

nentes, temos o número 531. Fazemos isso tão rápido que é natural ou automático  pensarmos desse jeito.

A mudança para o sistema binário segue o mesmo processo, mas a base agora é  2, e não 10. Assim, 1010 em binário representa:

1010 = 1 x 23 + 0 x 22 + 1 x 21 + 0 x 20

 = 1 x 8 + 0 x 4 + 1 x 2 + 0 x 1

 = 8 + 2

 = 10

**52 Introdução à Programação com Python**

A utilização do sistema binário é transparente em Python, ou seja, se você não so licitar explicitamente que esse sistema seja usado, tudo será apresentado na base 10  utilizada no dia a dia. A importância da noção de diferença de base é importante,  pois ela explica os limites da representação. O limite de representação é o valor  mínimo e máximo que pode ser representado em uma variável numérica. Esse  limite é causado pela quantidade de dígitos que foram reservados para armazenar  o número em questão. Vejamos como funciona na base 10.

Se você tem apenas 5 dígitos para representar um número, o maior número é  99999. E o menor seria (-99999). O mesmo acontece no sistema binário, sendo  que lá reservamos um dígito para registrar os sinais de positivo e negativo.

Para números inteiros, Python utiliza um sistema de precisão ilimitada que  permite a representação de números muito grandes. É como se você sempre  pudesse escrever novos dígitos à medida que for necessário. Você pode calcular  em Python valores como 21000000 (2 \*\* 1000000) sem problemas de representação,  mesmo quando o resultado é um número de 301030 dígitos.

Em ponto flutuante, temos limite e problemas de representação. Um número  decimal é representado em ponto flutuante utilizando-se uma mantissa e um  expoente (*sinal* × *mantissa* × *base*expoente). Tanto a mantissa quanto o expoente têm  um número de dígitos máximos que limita os números que podem ser represen

tados. Você não precisa se preocupar com isso no momento, pois esses valores  são bem grandes e você não terá problemas na maioria de seus programas. Você  pode obter mais informações acessando *http://pt.wikipedia.org/wiki/Ponto\_flutuante*.

A versão 3.1.2 do Python, rodando em Mac OS X, tem como limites  2.2250738585072014 × 10-308 e 1.7976931348623157 × 10308, suficientemente grandes  e pequenos para quase qualquer tipo de aplicação. É possível encontrar problemas  de representação em função de como os números decimais são convertidos em  números de ponto flutuante. Esses problemas são bem conhecidos e afetam todas  as linguagens de programação, não sendo um problema específico do Python.

Vejamos um exemplo: o número 0.1 não tem nada de especial no sistema decimal,  mas é uma dízima periódica no sistema binário. Você não precisa se preocupar  com esses detalhes agora, mas pode investigá-los mais tarde quando precisar (nor malmente cursos de computação apresentam uma disciplina, chamada cálculo  numérico, para abordar esses tipos de problemas). Digite no interpretador 3 \* 0.1

Você deve ter obtido como resultado 0.30000000000000004, e não 0.3, como  esperado. Não se assuste: não é um problema de seu computador, mas de repre sentação. Se for necessário, durante seus estudos, cálculos mais precisos, ou se os  resultados em ponto flutuante não satisfizerem os requisitos de precisão esperados,

**Capítulo 3** ■ **Variáveis e entrada de dados 53**

verifique os módulos decimals e fractions. A documentação do Python traz uma  página específica sobre esse tipo de problema: *http://docs.python.org/py3k/tutorial/ floatingpoint.html*.

**3.3 Variáveis do tipo Lógico**

Muitas vezes, queremos armazenar um conteúdo simples: verdadeiro ou falso em  uma variável. Nesse caso, utilizaremos um tipo de variável chamado tipo lógico ou booleano. Em Python, escreveremos **True** para verdadeiro e **False** para falso  (Listagem 3.1). Observe que o T e o F são escritos com letras maiúsculas.

► **Listagem 3.1 – Exemplo de variáveis do tipo lógico**

resultado = **True**

aprovado = **False**

**3.3.1 Operadores relacionais**

Para realizarmos comparações lógicas, utilizaremos operadores relacionais. A lista  de operadores relacionais suportados em Python é apresentada na tabela 3.2.

*Tabela 3.2 – Operadores relacionais*

|  |  |  |
| --- | --- | --- |
| **Operador** | **Operação** | **Símbolo matemático** |
| == | igualdade | = |
| > | maior que | > |
| < | menor que | < |
| != | diferente | ≠ |
| >= | maior ou igual | ≥ |
| <= | menor ou igual | ≤ |

O resultado de uma comparação é um valor do tipo lógico, ou seja, **True** (verda deiro) ou **False** (falso). Utilizaremos o verbo “avaliar” para indicar a resolução de  uma expressão.

► **Listagem 3.2 – Exemplo de uso de operadores relacionais**

>>> a = 1 # a recebe 1

>>> b = 5 # b recebe 5

**54 Introdução à Programação com Python**

>>> c = 2 # c recebe 2

>>> d = 1 # d recebe 1

>>> a == b # a é igual a b ?

**False**

>>> b > a # b é maior que a?

**True**

>>> a < b # a é menor que b?

**True**

>>> a == d # a é igual a d?

**True**

>>> b >= a # b é maior ou igual a a?

**True**

>>> c <= b # c é menor ou igual a b?

**True**

>>> d != a # d é diferente de a?

**False**

>>> d != b # d é diferente de b?

**True**

Esses operadores são utilizados como na matemática. Especial atenção deve ser  dada aos operadores >= e <=. O resultado desses operadores é realmente maior ou  igual e menor ou igual, ou seja, 5 >= 5 é verdadeiro, assim como 5 <= 5.

Observe que, na listagem 3.2, utilizamos o símbolo de cerquilha (#) para escrever  comentários na linha de comando. Todo texto à direita do cerquilha é ignorado  pelo interpretador Python, ou seja, você pode escrever o que quiser. Leia nova mente a listagem 3.2 e veja como é mais fácil entender cada linha quando a co mentamos. Comentários não são obrigatórios, mas são muito importantes. Você  pode e deve utilizá-los em seus programas para facilitar o entendimento e oferecer  uma anotação para você mesmo. É comum lermos um programa alguns meses  depois de escrito e termos dificuldade de lembrar o que realmente queríamos  fazer. Você também não precisa comentar todas as linhas de seus programas ou  escrever o óbvio. Uma dica é identificar os programas com seu nome, a data em  que começou a ser escrito e mesmo a listagem ou capítulo do livro onde você o  encontrou.

Variáveis de tipo lógico também podem ser utilizadas para armazenar o resultado  de expressões e comparações.

**Capítulo 3** ■ **Variáveis e entrada de dados 55**

► **Listagem 3.3 – Exemplo do uso de operadores relacionais com variáveis do tipo lógico**

nota = 8

média = 7

aprovado = nota > média

**print**(aprovado)

Se uma expressão contém operações aritméticas, estas devem ser calculadas antes  que os operadores relacionais sejam avaliados. Quando avaliamos uma expressão,  substituímos o nome das variáveis por seu conteúdo e só então verificamos o  resultado da comparação.

**Exercício 3.2** Complete a tabela abaixo, respondendo True ou False. Considere  a = 4, b = 10, c = 5.0, d = 1 e f = 5.

|  |  |
| --- | --- |
| **Expressão** | **Resultado** |
| a == c | 🔾 True 🔾 False |
| a < b | 🔾 True 🔾 False |
| d > b | 🔾 True 🔾 False |
| c! = f | 🔾 True 🔾 False |
| a == b | 🔾 True 🔾 False |
| c < d | 🔾 True 🔾 False |
| b > a | 🔾 True 🔾 False |
| c >= f | 🔾 True 🔾 False |
| f >= c | 🔾 True 🔾 False |
| c <= c | 🔾 True 🔾 False |
| c <= f | 🔾 True 🔾 False |

**3.3.2 Operadores lógicos**

Para agrupar operações com lógica booleana, utilizaremos operadores lógicos.  Python suporta três operadores básicos: **not** (não), **and** (e), **or** (ou). Esses operadores  podem ser traduzidos como não (¬ negação), e (Λ conjunção) e ou (V disjunção).

**56 Introdução à Programação com Python**

*Tabela 3.3 – Operadores lógicos*

|  |  |
| --- | --- |
| **Operador Python** | **Operação** |
| **not** | não |
| **and** | e |
| **or** | ou |

Cada operador obedece a um conjunto simples de regras, expresso pela tabela  verdade desse operador. A tabela verdade demonstra o resultado de uma operação  com um ou dois valores lógicos ou operandos. Quando o operador utiliza apenas  um operando, dizemos que é um operador unário. Ao utilizar dois operandos, é  chamado operador binário. O operador de negação (**not**) é um operador unário.  **or** (ou) e **and** (e) são operadores binários, precisando, assim, de dois operandos.

**3.3.2.1 Operador not**

O operador **not** (não) é o mais simples, pois precisa apenas de um operador. A  operação de negação também é chamada de inversão, pois um valor verdadeiro  negado se torna falso e vice-versa. A tabela verdade do operador **not** (não) é apre sentada na tabela 3.4.

*Tabela 3.4 – Tabela verdade do operador not (não)*

|  |  |
| --- | --- |
| **V1** | **not V1** |
| V | F |
| F | V |

► **Listagem 3.4 – Operador not**

>>> **not True**

**False**

>>> **not False**

**True**

**3.3.2.2 Operador and**

O operador **and** (e) tem sua tabela verdade representada na tabela 3.5. O operador  **and** (e) resulta verdadeiro apenas quando seus dois operadores forem verdadeiros.

**Capítulo 3** ■ **Variáveis e entrada de dados 57**

*Tabela 3.5 – Tabela verdade do operador and (e)*

|  |  |  |
| --- | --- | --- |
| **V1** | **V2** | **V1 and V2** |
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

► **Listagem 3.5 – Operador and**

>>> **True and True**

**True**

>>> **True and False**

**False**

>>> **False and True**

**False**

>>> **False and False**

**False**

**3.3.2.3 Operador or**

A tabela verdade do operador **or** (ou) é apresentada na tabela 3.6. A regra funda mental do operador **or** (ou) é que ele resulta em falso apenas se seus dois opera dores também forem falsos. Se apenas um de seus operadores for verdadeiro, ou  se os dois forem, o resultado da operação será verdadeiro.

*Tabela 3.6 – Tabela verdade do operador or(ou)*

|  |  |  |
| --- | --- | --- |
| **V1** | **V2** | **V1 or V2** |
| V | V | V |
| V | F | V |
| F | V | V |
| F | F | F |

► **Listagem 3.6 – Operador or**

>>> **True or True**

**True**

**58 Introdução à Programação com Python**

>>> **True or False**

**True**

>>> **False or True**

**True**

>>> **False or False**

**False**

**Exercício 3.3** Complete a tabela a seguir utilizando a = True, b = False e c = True.

|  |  |
| --- | --- |
| **Expressão** | **Resultado** |
| a and a | 🔾 True 🔾 False |
| b and b | 🔾 True 🔾 False |
| not c | 🔾 True 🔾 False |
| not b | 🔾 True 🔾 False |
| not a | 🔾 True 🔾 False |
| a and b | 🔾 True 🔾 False |
| b and c | 🔾 True 🔾 False |
| a or c | 🔾 True 🔾 False |
| b or c | 🔾 True 🔾 False |
| c or a | 🔾 True 🔾 False |
| c or b | 🔾 True 🔾 False |
| c or c | 🔾 True 🔾 False |
| b or b | 🔾 True 🔾 False |

**3.3.3 Expressões lógicas**

Os operadores lógicos podem ser combinados em expressões lógicas mais com plexas. Quando uma expressão tiver mais de um operador lógico, avalia-se o  operador **not** (não) primeiramente, seguido do operador **and** (e) e, finalmente, **or** (ou). Vejamos a seguir a ordem de avaliação da expressão, onde a operação sendo  avaliada é sublinhada; e o resultado, mostrado na linha seguinte.

**Capítulo 3** ■ **Variáveis e entrada de dados 59**

True or False and not True

True or False and False

True or False

True

Os operadores relacionais também podem ser utilizados em expressões com  operadores lógicos.

salário > 1000 and idade > 18

Nesses casos, os operadores relacionais devem ser avaliados primeiramente. Fa çamos salário = 100 e idade = 20. Teremos:

salário > 1000 and idade > 18

100 > 1000 and 20 > 18

False and True

False

A grande vantagem de escrever esse tipo de expressão é representar condições  que podem ser avaliadas com valores diferentes. Por exemplo: imagine que  salário > 1000 **and** idade > 18 seja uma condição para um empréstimo de compra  de um carro novo. Quando salário = 100 e idade = 20, sabemos que o resultado  da expressão é falso, e podemos interpretar que, nesse caso, a pessoa não recebe

ria o empréstimo. Avaliemos a mesma expressão com salário = 2000 e idade = 30.

salário > 1000 and idade > 18

2000 > 1000 and 30 > 18

True and True

True

Agora o resultado é **True** (verdadeiro) e poderíamos dizer que a pessoa atende às  condições para obter o empréstimo.

**Exercício 3.4** Escreva uma expressão para determinar se uma pessoa deve ou não  pagar imposto. Considere que pagam imposto pessoas cujo salário é maior que  R$ 1.200,00.

**60 Introdução à Programação com Python**

**Exercício 3.5** Calcule o resultado da expressão A > B and C or D, utilizando os  valores da tabela a seguir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **Resultado** |
| 1 | 2 | True | False |  |
| 10 | 3 | False | False |  |
| 5 | 1 | True | True |  |

**Exercício 3.6** Escreva uma expressão que será utilizada para decidir se um aluno foi  ou não aprovado. Para ser aprovado, todas as médias do aluno devem ser maiores  que 7. Considere que o aluno cursa apenas três matérias, e que a nota de cada uma  está armazenada nas seguintes variáveis: matéria1, matéria2 e matéria3.

**3.4 Variáveis string**

Variáveis do tipo string armazenam cadeias de caracteres como nomes e textos em  geral. Chamamos cadeia de caracteres uma sequência de símbolos como letras,  números, sinais de pontuação etc. Exemplo: João e Maria comem pão. Nesse caso,  João é uma sequência com as letras J, o, ã, o. Para simplificar o texto, utilizaremos  o nome string para mencionar cadeias de caracteres. Podemos imaginar uma string  como uma sequência de blocos, onde cada letra, número ou espaço em branco  ocupa uma posição, como mostra a figura 3.1.

String

J o ã o e M a r i a c o m e m p ã o *Figura 3.1 – Representação de uma string.*

Para possibilitar a separação entre o texto do seu programa e o conteúdo de  uma string, utilizaremos aspas (") para delimitar o início e o fim da sequência  de caracteres.

Voltando ao exemplo anterior, escreveremos "João e Maria comem pão". Veja que  nesse caso não há qualquer problema em utilizarmos espaços. Na verdade, o  computador ignora praticamente tudo que escrevemos entre aspas, mas veremos  mais tarde que não é bem assim.

**Capítulo 3** ■ **Variáveis e entrada de dados 61**

As variáveis do tipo string são utilizadas para armazenar sequências de caracteres,  normalmente utilizadas em textos ou mensagens. O tipo string é muito útil e  bastante utilizado para exibir mensagens ou mesmo para gerar outros arquivos.

Uma string em Python tem um tamanho associado, assim como um conteúdo  que pode ser acessado caractere a caractere. O tamanho de uma string pode ser  obtido utilizando-se a função len. Essa função retorna o número de caracteres na  string. Dizemos que uma função retorna um valor quando podemos substituir o  texto da função por seu resultado. A função len retorna um valor do tipo inteiro,  representando a quantidade de caracteres contidos na string. Se a string é vazia  (representada simplesmente por "", ou seja, duas aspas sem nada entre elas, nem  mesmo espaços em branco), seu tamanho é igual a zero. Façamos alguns testes,  como mostra a listagem 3.7.

► **Listagem 3.7 – A função len**

>>> **print** (len("A"))

1

>>> **print** (len("AB"))

2

>>> **print** (len(""))

0

>>> **print** (len("O rato roeu a roupa"))

19

Como dito anteriormente, outra característica de strings é poder acessar seu conte údo caractere a caractere. Sabendo que uma string tem um determinado tamanho,  podemos acessar seus caracteres utilizando um número inteiro para representar sua  posição. Esse número é chamado de índice, e começamos a contar de zero. Isso quer  dizer que o primeiro caractere da string é de posição ou índice 0. Observe a figura 3.2.

String 

0 1 2 3 4 5 6 7 8

A B C D E F G H I

Índice

Conteúdo 

*Figura 3.2 – Índices e conteúdo de uma variável string.*

Para acessar os caracteres de uma string, devemos informar o índice ou posição  do caractere entre colchetes ([ ]). Como o primeiro caractere de uma string é o  de índice 0, podemos acessar valores de 0 até o tamanho da string menos 1. Logo,

**62 Introdução à Programação com Python**

se a string contiver 9 caracteres, poderemos acessar os caracteres de 0 a 8. Veja o  resultado de alguns testes com strings na listagem 3.8. Se tentarmos acessar um  índice maior que a quantidade de caracteres da string, o interpretador emitirá  uma mensagem de erro.

► **Listagem 3.8 – Manipulação de strings no interpretador**

>>> a = "ABCDEF"

>>> **print**(a[0])

A

>>> **print**(a[1])

B

>>> **print**(a[5])

F

>>> **print**(a[6])

Traceback (most recent call last):

 File "<stdin>", line 1, in <module>

IndexError: string index out of range

>>> **print**(len(a))

6

**3.4.1 Operações com strings**

As variáveis de tipo string suportam operações como fatiamento, concatenação e  composição. Por fatiamento, podemos entender a capacidade de utilizar apenas  uma parte de uma string, ou uma fatia. A concatenação nada mais é que poder  juntar duas ou mais strings em uma nova string maior. A composição é muito  utilizada em mensagens que enviamos à tela e consiste em utilizar strings como  modelos onde podemos inserir outras strings. Veremos cada uma dessas opera ções nas seções a seguir.

**3.4.1.1 Concatenação**

O conteúdo de variáveis string podem ser somados, ou melhor, concatenados.  Para concatenar duas strings, utilizamos o operador de adição (+). Assim,  "AB" + "C" é igual a "ABC". Um caso especial de concatenação é a repetição de  uma string várias vezes. Para isso, utilizamos o operador de multiplicação (\*):  "A" \* 3 é igual a "AAA". Vejamos alguns exemplos na listagem 3.9.

**Capítulo 3** ■ **Variáveis e entrada de dados 63**

► **Listagem 3.9 – Exemplo de concatenação**

>>> s = "ABC"

>>> **print** (s + "C")

ABCC

>>> **print** (s + "D" \* 4)

ABCDDDD

>>> **print** ("X" + "-"\*10 + "X")

X----------X

>>> **print** (s+"x4 = "+s\*4)

ABCx4 = ABCABCABCABC

**3.4.1.2 Composição**

Juntar várias strings para construir uma mensagem nem sempre é prático. Por  exemplo, exibir que "João tem X anos", onde X é uma variável numérica.

Usando a composição de strings do Python, podemos escrever de forma simples  e clara:

"João tem %d anos" % X

Onde o símbolo de % foi utilizado para indicar a composição da string anterior  com o conteúdo da variável X. O %d dentro da primeira string é o que chamamos  de marcador de posição. O marcador indica que naquela posição estaremos co locando um valor inteiro, daí o %d.

Python suporta diversas operações com marcadores. Veremos mais sobre marcadores  em outras partes do livro. A tabela 3.7 apresenta os principais tipos de marcadores.  Veja que eles são diferentes, de acordo com o tipo de variável que vamos utilizar.

*Tabela 3.7 – Marcadores*

|  |  |
| --- | --- |
| **Marcador** | **Tipo** |
| %d | Números inteiros |
| %s | Strings |
| %f | Números decimais |

Imagine que precisamos apresentar um número como 001 ou 002, mas que tam bém pode ser algo como 050 ou 561. Nesse caso, estamos querendo apresentar  um número com três posições, completando com zeros à esquerda se o número

**64 Introdução à Programação com Python**

for menor. Podemos realizar essa operação utilizando "%03d" % X. Observe que  adicionamos 03 entre o % e o d. Se você precisar apenas que o número ocupe três  posições, mas não desejar zeros à esquerda, basta retirar o zero e utilizar "%3d" % X.  Isso é muito importante quando estamos gravando dados em um arquivo ou sim

plesmente exibindo informações na tela. Vejamos alguns exemplos na listagem 3.10. ► **Listagem 3.10 – Exemplo de composição com marcadores**

>>> idade = 22

>>> **print**("[%d]" % idade)

[22]

>>> **print**("[%03d]" % idade)

[022]

>>> **print**("[%3d]" % idade)

[ 22]

>>> **print**("[%-3d]" % idade)

[22 ]

Quando formatamos números decimais, podemos utilizar dois valores entre o sím bolo de % e a letra f. O primeiro indica o tamanho total em caracteres a reservar; e o  segundo, o número de casas decimais. Assim, %5.2f diz que estaremos imprimindo  um número decimal utilizando cinco posições, sendo que duas são para a parte  decimal. Isso é muito interessante para exibir o resultado de cálculos ou representar  dinheiro. Por exemplo, para exibir R$ 5, você pode utilizar "R$%f" % 5, mas o resultado  não é bem o que esperamos, pois normalmente utilizamos apenas dois dígitos após  a vírgula quando falamos de dinheiro. Vejamos alguns exemplos na listagem 3.11.

► **Listagem 3.11 – Exemplos de composição com números decimais**

>>> **print**("%5f" % 5)

5.000000

>>> **print**("%5.2f" % 5)

 5.00

>>> **print**("%10.5f" % 5)

 5.00000

O poder da composição realmente aparece quando precisamos combinar vários  valores em uma nova string. Imagine que João tem 22 anos e apenas R$ 51,34 no  bolso. Para exibir essa mensagem, podemos utilizar:

"%s tem %d anos e apenas R$%5.2f no bolso" % ("João", 22, 51.34)

**Capítulo 3** ■ **Variáveis e entrada de dados 65**

Python suporta diversas operações com marcadores. Veremos mais sobre marca dores em outras partes do livro. Quando temos mais de um marcador na string,  somos obrigados a escrever os valores a substituir entre parênteses. Agora, vejamos  exemplos com outros tipos, e utilizando mais de uma variável na composição,  na listagem 3.12.

► **Listagem 3.12 – Exemplo de composição de string**

>>> nome = "João"

>>> idade = 22

>>> grana = 51.34

>>> **print**("%s tem %d anos e R$%f no bolso." % (nome, idade, grana)) João tem 22 anos e R$51.340000 no bolso.

>>> **print**("%12s tem %3d anos e R$%5.2f no bolso." % (nome, idade, grana))  João tem 22 anos e R$51.34 no bolso.

>>> **print**("%12s tem %03d anos e R$%5.2f no bolso." % (nome, idade, grana))  João tem 022 anos e R$51.34 no bolso.

>>> **print**("%-12s tem %-3d anos e R$%-5.2f no bolso." % (nome, idade, grana)) João tem 22 anos e R$51.34 no bolso.

**3.4.1.3 Fatiamento**

O fatiamento em Python é muito poderoso. Imagine nossa string de exemplo  da figura 3.2. Podemos fatiá-la de forma a escrever apenas seus dois primeiros  caracteres AB utilizando como índice [0:2]. O fatiamento funciona com a utili zação de dois pontos no índice da string. O número à esquerda dos dois pontos  indica a posição de início da fatia; e o à direita, do fim. No entanto, é preciso ter  atenção ao final, pois no exemplo anterior utilizamos 2; e o C, que é o caractere  na posição 2; não foi incluído. Dizemos que isso acontece porque o final da fatia  não é incluído na mesma, sendo deixado de fora. Entenda [0:2] como a fatia de  caracteres da posição 0 até a posição 2, sem incluí-la, ou o intervalo fechado em  0 e aberto em 2.

Vejamos outros exemplos de fatias na listagem 3.13.

► **Listagem 3.13 – Exemplo de fatiamento**

>>> s="ABCDEFGHI"

>>> **print** (s[0:2])

AB

**66 Introdução à Programação com Python**

>>> **print** (s[1:2])

B

>>> **print** (s[2:4])

CD

>>> **print** (s[0:5])

ABCDE

>>> **print** (s[1:8])

BCDEFGH

Podemos também omitir o número da esquerda ou o da direita para representar  do início ou até o final. Assim, [:2] indica do início até o segundo caractere (sem  incluí-lo), e [1:] indica do caractere de posição 1 até o final da string. Observe que,  nesse caso, nem precisamos saber quantos caracteres a string contém.

Se omitirmos o início e o fim da fatia, estaremos fazendo apenas uma cópia de  todos os caracteres da string para uma nova string.

Podemos também utilizar valores negativos para indicar posições a partir da  direita. Assim -1 é o último caractere; -2, o penúltimo; e assim por diante. Veja o  resultado de testes com índices negativos na listagem 3.14.

► **Listagem 3.14 – Exemplo de fatiamento com omissão de valores e com índices negativos**

>>> s="ABCDEFGHI"

>>> **print** (s[:2])

AB

>>> **print** (s[1:])

BCDEFGHI

>>> **print** (s[0:-2])

ABCDEFG

>>> **print** (s[:])

ABCDEFGHI

>>> **print** (s[-1:])

I

>>> **print** (s[-5:7])

EFG

>>> **print** (s[-2:-1])

H

Veremos mais sobre strings em Python no capítulo 7.

**Capítulo 3** ■ **Variáveis e entrada de dados 67 3.5 Sequências e tempo**

Um programa é executado linha por linha pelo computador, executando as  operações descritas no programa uma após a outra. Quando trabalhamos com  variáveis, devemos nos lembrar de que o conteúdo de uma variável pode mudar  com o tempo. Isso porque a cada vez que alteramos o valor de uma variável, o  valor anterior é substituído pelo novo.

Observe o programa da listagem 3.15. A variável dívida foi utilizada para registrar  quanto alguém estava devendo; e a variável compra, o valor de novas despesas dessa  pessoa. Como somos justos, a pessoa começou sem dívidas em ❶.

► **Listagem 3.15 – Exemplo de sequência e tempo**

dívida = 0 ❶

compra = 100 ❷

dívida = dívida + compra ❸

compra = 200 ❹

dívida = dívida + compra ❺

compra = 300 ❻

dívida = dívida + compra ❼

compra = 0 ❽

**print**(dívida) ❾

Em ❷, temos a primeira compra no valor de R$ 100. No entanto, o valor da dí vida continua sendo 0, pois ainda não alteramos seu valor de forma a adicionar  a compra realizada. Isso é feito em ❸. Observe que estamos atualizando o valor  da dívida com o valor atual mais a compra.

Em ❹, registramos uma nova compra no valor de R$ 200. Nesse ponto, compra  tem seu valor substituído por R$ 200, causando a perda do valor anterior R$ 100.  Como já somamos o valor anterior na variável dívida, essa perda não representará  um problema.

❺ é idêntica a ❸, mas o resultado é bem diferente. Nesse momento, compra vale  R$ 200; e dívida, R$ 100.

Em ❻, alteramos o valor de compra novamente. Dessa vez, a compra foi de R$ 300.

❼ é idêntica a ❸ e ❺, mas seu resultado é diferente, pois nesse momento temos  compra valendo R$ 300; e dívida igual a R$ 300 (100 + 200), sendo atualizada  para R$ 600 (300 + 300).

**68 Introdução à Programação com Python**

Em ❽, simplesmente dizemos que a compra foi 0, representando que a pessoa  não comprou mais nada.

❾ exibe o conteúdo da variável dívida na tela (600).

A figura 3.3 mostra a evolução do conteúdo de nossas duas variáveis em função  do tempo, representado pelo número da linha.

tempo 1 2 3 4 5 6 7 8 9 

compra dívida

100 200 300 0

0 100 300 600 600 *Figura 3.3 – Mudanças no valor de duas variáveis no tempo.*

**3.6 Rastreamento**

Uma das principais diferenças entre ler um texto e um programa é justamente  seguir as mudanças de valores de cada variável conforme o programa é executado.  Entender que o valor das variáveis pode mudar durante a execução do programa  não é tão natural, mas é fundamental para a programação de computadores. Um  programa não pode ser lido como um texto, mas cuidadosamente analisado linha  a linha. Ao escrever seus programas, verifique linha a linha os efeitos e mudanças  causados no valor de cada variável. Para programar corretamente, você deve ser  capaz de entender o que cada linha do programa significa e os efeitos que produz.  Essa atividade, chamada de rastreamento, é muito importante para entender novos  programas e para encontrar erros nos programas que você escreverá.

Para rastrear um programa, utilize lápis, borracha e uma folha de papel. Escreva  o nome de suas variáveis na folha de papel, como se fossem títulos de colunas,  deixando espaço para ser preenchido embaixo desses nomes. Leia uma linha do  programa de cada vez e escreva o valor atribuído a cada variável na outra folha, na  mesma coluna em que escreveu o nome da variável. Se o valor da variável mudar,  escreva o novo valor e risque o anterior, um embaixo do outro, formando uma  coluna. Ao exibir algo na tela, escreva também na outra folha, como se ela fosse  a sua tela (você pode desenhar a tela como se fosse uma variável, mas se lembre  de deixar um pouco mais de espaço). Um exemplo de como ficaria o resultado  do rastreamento do programa da listagem 3.15 é apresentado na figura 3.4. O

**Capítulo 3** ■ **Variáveis e entrada de dados 69**

rastreamento vai ajudá-lo a entender melhor as mudanças de valores de suas  variáveis e a acompanhar a execução do programa, como mais tarde será feito  pelo computador. É um processo detalhado que precisa de atenção. Não tente  simplificá-lo ou começar a rastrear no meio de um programa. Você deve rastrear  linha a linha, do início ao fim do programa. Se você encontrar um erro, pode  parar o rastreamento e corrigi-lo, mas lembre-se de recomeçar do início sempre  que alterar o programa ou os valores sendo rastreados.

*Figura 3.4 – Exemplo de rastreamento no papel.*

Dominar o rastreamento de um programa é essencial para programar e ajuda  muito a entender como os programas realmente funcionam. Lembre-se de que  programar é detalhar, e que simplesmente ler o texto de um programa não é  suficiente. Você deve rastreá-lo para entendê-lo. Embora pareça óbvio, esse é um  dos erros mais comuns quando se começa a programar. Se um programa não  funciona ou se você não entendeu exatamente o que ele faz, o rastreamento é a  melhor ferramenta.

**3.7 Entrada de dados**

Até agora nossos programas trabalharam apenas com valores conhecidos, escritos  no próprio programa. No entanto, o melhor da programação é poder escrever a  solução de um problema e aplicá-la várias vezes. Para isso, precisamos melhorar  nossos programas de forma a permitir que novos valores sejam fornecidos durante  sua execução, de modo que poderemos executá-los com valores diferentes sem  alterar os programas em si.

Chamamos de entrada de dados o momento em que seu programa recebe dados  ou valores por um dispositivo de entrada de dados (como o teclado do compu tador) ou de um arquivo em disco.

**70 Introdução à Programação com Python**

A função **input** é utilizada para solicitar dados do usuário. Ela recebe um parâ metro, que é a mensagem a ser exibida, e retorna o valor digitado pelo usuário.  Vejamos um exemplo na listagem 3.16.

► **Listagem 3.16 – Entrada de dados**

x = **input**("Digite um número: ")

**print**(x)

► **Listagem 3.17 – Saída na tela, tendo o 5 como exemplo de número digitado pelo usuário**

Digite um número: 5

5

Vejamos outro exemplo na listagem 3.18.

► **Listagem 3.18 – Exemplo de entrada de dados**

nome = **input**("Digite seu nome:") ❶

**print**("Você digitou %s" % nome)

**print**("Olá, %s!" % nome)

Em ❶, solicitamos a entrada de dados, no caso, o nome do usuário. A mensagem  “Digite seu nome:” é exibida, e o programa para até que o usuário digite ENTER.  Só então o resto do programa é executado. Vejamos a saída de dados quando  digitamos João como nome na listagem 3.19.

► **Listagem 3.19 – Resultado da entrada de dados**

Digite seu nome:João

Você digitou João

Olá, João!

Execute o programa outras vezes digitando, por exemplo, 123 como nome. Obser ve que o programa não se importa com os valores digitados pelo usuário. Essa  verificação deve ser feita por seu programa. Veremos como fazer isso em outro  capítulo, como validação de dados.

**3.7.1 Conversão da entrada de dados**

A função **input** sempre retorna valores do tipo string, ou seja, não importa se di gitamos apenas números, o resultado sempre é string. Para resolver esse pequeno

**Capítulo 3** ■ **Variáveis e entrada de dados 71**

problema, vamos utilizar a função int para converter o valor retornado em um  número inteiro, e a função float para convertê-lo em número decimal ou de ponto  flutuante. Vejamos outro exemplo usando essas funções, no qual devemos calcular  o valor de um bônus por tempo de serviço na listagem 3.20.

► **Listagem 3.20 – Cálculo de bônus por tempo de serviço**

anos = int(**input**("Anos de serviço: "))

valor\_por\_ano = float(**input**("Valor por ano: "))

bônus = anos \* valor\_por\_ano

**print**("Bônus de R$ %5.2f" % bônus)

Vejamos o resultado se testarmos 10 anos e R$ 25 por ano na tela da listagem 3.21.  Observe que escrevemos apenas 25, e não R$ 25. Isso porque 25 é um número,  e R$ 25 é uma string. Por enquanto, não vamos misturar dois tipos de dados na  mesma entrada de dados, para simplificar nossos programas.

► **Listagem 3.21 – Resultado do cálculo para 10 anos e R$ 25 por ano**

Anos de serviço: 10

Valor por ano: 25

Bônus de R$ 250.00

Execute o programa novamente com outros valores. Experimente digitar uma  letra em anos de serviço ou em valor por ano. Você receberá uma mensagem de  erro, pois a conversão de letras em números não é automática. A próxima seção  explica melhor o problema.

**Exercício 3.7** Faça um programa que peça dois números inteiros. Imprima a soma  desses dois números na tela.

**Exercício 3.8** Escreva um programa que leia um valor em metros e o exiba con vertido em milímetros.

**Exercício 3.9** Escreva um programa que leia a quantidade de dias, horas, minutos  e segundos do usuário. Calcule o total em segundos.

**72 Introdução à Programação com Python**

**Exercício 3.10** Faça um programa que calcule o aumento de um salário. Ele deve  solicitar o valor do salário e a porcentagem do aumento. Exiba o valor do aumento  e do novo salário.

**Exercício 3.11** Faça um programa que solicite o preço de uma mercadoria e o per centual de desconto. Exiba o valor do desconto e o preço a pagar.

**Exercício 3.12** Escreva um programa que calcule o tempo de uma viagem de carro.  Pergunte a distância a percorrer e a velocidade média esperada para a viagem.

**Exercício 3.13** Escreva um programa que converta uma temperatura digitada em  °C em °F. A fórmula para essa conversão é:

*F* = 9 × *C*

5 + 32

**Exercício 3.14** Escreva um programa que pergunte a quantidade de km percorridos  por um carro alugado pelo usuário, assim como a quantidade de dias pelos quais  o carro foi alugado. Calcule o preço a pagar, sabendo que o carro custa R$ 60 por  dia e R$ 0,15 por km rodado.

**Exercício 3.15** Escreva um programa para calcular a redução do tempo de vida de  um fumante. Pergunte a quantidade de cigarros fumados por dia e quantos anos  ele já fumou. Considere que um fumante perde 10 minutos de vida a cada cigarro,  calcule quantos dias de vida um fumante perderá. Exiba o total em dias.

**3.7.2 Erros comuns**

A entrada de dados é um ponto frágil em nossos programas. Como não temos  como prever o que o usuário vai digitar, temos que nos preparar para reconhecer  os erros mais comuns. Vejamos um programa que lê três valores na listagem 3.22.

**Capítulo 3** ■ **Variáveis e entrada de dados 73**

► **Listagem 3.22 – Entrada de dados com conversão de tipos**

nome = **input**("Digite seu nome: ")

idade = int(**input**("Digite sua idade: "))

saldo = float(**input**("Digite o saldo da sua conta bancária: "))

**print**(nome)

**print**(idade)

**print**(saldo)

A listagem 3.23 mostra o resultado da execução do programa quando todos os  valores são digitados corretamente. Correto significa uma ausência de erros  durante a função **input** ou durante a conversão do valor retornado pela função.

► **Listagem 3.23 – Exemplo de entrada de dados**

Digite seu nome: João

Digite sua idade: 42

Digite o saldo da sua conta bancária: 15756.34

João

42

15756.34

A listagem 3.24 mostra outro exemplo de entrada de dados bem-sucedida. Observe  que, como utilizamos a função float para converter o saldo, mesmo inserindo 34  o valor foi convertido para 34.0.

► **Listagem 3.24 – Exemplo de entrada de dados**

Digite seu nome: Maria

Digite sua idade: 28

Digite o saldo da sua conta bancária: 34

Maria

28

34.0

Já na listagem 3.25, temos um exemplo de erro durante a entrada de dados. No  caso, digitamos letras (abc) que não podem ser convertidas em um valor inteiro.  Observe que o erro interrompe nosso programa, exibindo a linha em que ocorreu  e o nome do erro. Nesse caso, o erro aconteceu na linha 2 e seu nome é ValueError:  invalid literal for int with base 10: 'abc'.

**74 Introdução à Programação com Python**

► **Listagem 3.25 – Erro de conversão**

Digite seu nome: Minduim

Digite sua idade: abc

Traceback (most recent call last):

 File "input/input2.py", line 2, in <module>

 idade = int(**input**("Digite sua idade: "))

ValueError: invalid literal for int() with base 10: 'abc'

A listagem 3.26 mostra outro erro de conversão, mas, dessa vez, durante a conversão  para número decimal, usando a função float. A entrada de dados é um pouco  rústica, parando em caso de erro. Mais adiante, aprenderemos sobre exceções em  Python e como tratar esse tipo de erro. Por enquanto, basta saber que não estamos  validando a entrada e que nossos programas ainda são frágeis.

► **Listagem 3.26 – Erro de conversão: letras no lugar de números**

Digite seu nome: Juanito

Digite sua idade: 31

Digite o saldo da sua conta bancária: abc

Traceback (most recent call last):

 File "input/input2.py", line 3, in <module>

 saldo = float(**input**("Digite o saldo da sua conta bancária: ")) ValueError: could not convert string to float: abc

► **Listagem 3.27 – Erro de conversão: vírgula no lugar de ponto**

Digite seu nome: Mary

Digite sua idade: 25

Digite o saldo da sua conta bancária: 17,4

Traceback (most recent call last):

 File "input/input2.py", line 3, in <module>

 saldo = float(**input**("Digite o saldo da sua conta bancária: ")) ValueError: invalid literal for float(): 17,4

O erro mostrado na listagem 3.27 é muito comum em países onde se usa a vírgula  e não o ponto como separador entre a parte inteira e fracionária de um número.  Em Python, você deve sempre digitar valores decimais usando o ponto, e não  a vírgula como em português. Assim, 17,4 é um valor inválido, pois deveria ter  sido digitado como 17.4. Existem recursos em Python para resolver esse tipo de  problema, mas ainda é cedo para abordarmos o assunto.

**capítulo 4**

**Condições**

*Executar ou não executar? Eis a questão...*

Nem sempre todas as linhas dos programas serão executadas. Muitas vezes, será  mais interessante decidir que partes do programa devem ser executadas com base  no resultado de uma condição. A base dessas decisões consistirá em expressões  lógicas que permitam representar escolhas em programas.

**4.1 if**

As condições servem para selecionar quando uma parte do programa deve ser  ativada e quando deve ser simplesmente ignorada. Em Python, a estrutura de  decisão é o **if**. Seu formato é apresentado na listagem 4.1.

► **Listagem 4.1 – Formato da estrutura de condicional if**

**if** <condição>:

bloco verdadeiro

O **if** nada mais é que o nosso “se”. Poderemos então entendê-lo em português da  seguinte forma: se a condição for verdadeira, faça alguma coisa.

Vejamos um exemplo: ler dois valores e imprimir o maior deles, apresentado na  listagem 4.2.

► **Listagem 4.2 – Condições**

a = int(**input**("Primeiro valor: "))

b = int(**input**("Segundo valor: "))

**75**

**76 Introdução à Programação com Python**

**if** a > b: ❶

**print** ("O primeiro número é o maior!") ❷

**if** b > a: ❸

**print** ("O segundo número é o maior!") ❹

Em ❶, temos a condição a > b. Essa expressão será avaliada, e, se o seu resultado  for verdadeiro, a linha ❷ será executada. Se for falso, a linha ❷ será ignorada. O  mesmo acontece para a condição b > a da linha ❸. Se o seu resultado for verda deiro, a linha ❹ será executada. Se for falso, ignorada.

A sequência de execução do programa é alterada de acordo com os valores digi tados como o primeiro e segundo valores. Digite o programa da listagem 4.2 e  execute-o duas vezes. Na primeira vez, digite um valor maior primeiro e um menor  em segundo. Na segunda vez, inverta esses valores e verifique se a mensagem na  tela também mudou.

Quando o primeiro valor é maior que o segundo, temos as seguintes linhas sendo  executadas: ❶, ❷, ❸. Quando o primeiro valor é menor que o segundo, temos  outra sequência: ❶, ❸, ❹. É importante entender que a linha com a condição  em si é executada mesmo se o resultado da expressão for falso.

As linhas ❶ e ❸ foram terminadas com o símbolo dois pontos (:). Quando isso  acontece, temos o anúncio de um bloco de linhas a seguir. Em Python, um bloco  é representado deslocando-se o início da linha para a direita. O bloco continua  até a primeira linha com deslocamento diferente.

Observe que começamos a escrever a linha ❷ alguns caracteres mais à direita da  linha anterior ❶ que iniciou o bloco. Como a linha ❸ foi escrita mais à esquerda,  dizemos que o bloco da linha ❷ foi terminado.

|  |
| --- |
| **TRÍVIA**  Python é uma das poucas linguagens de programação que utiliza o  deslocamento do texto à direita (recuo) para marcar o início e o fim de  um bloco. Outras linguagens contam com palavras especiais para isso,  como BEGIN e END, em Pascal; ou as famosas chaves ({ e }), em C e Java. |

**Exercício 4.1** Analise o programa da listagem 4.2. Responda o que acontece se o  primeiro e o segundo valores forem iguais? Explique.

**Capítulo 4** ■ **Condições 77**

Vejamos outro exemplo, onde solicitaremos a idade do carro do usuário e, em  seguida, escreveremos novo se o carro tiver menos de três anos; ou velho, em  caso contrário.

► **Listagem 4.3 – Carro novo ou velho, dependendo da idade**

idade = int(**input**("Digite a idade do seu carro: "))

**if** idade <= 3:

**print**("Seu carro é novo") ❶

**if** idade > 3:

**print**("Seu carro é velho") ❷

Execute o programa da listagem 4.3 e verifique o que aparece na tela. Você pode  executá-lo várias vezes com as seguintes idades: 1, 3 e 5. A primeira condição é  idade <= 3. Essa condição decide se a linha com função **print** ❶ será ou não exe cutada. Como é uma condição simples, podemos entender que só exibiremos a  mensagem do carro novo para as idades 0, 1, 2 e 3. A segunda condição, idade > 3,  é o inverso da primeira. Se você observar de perto, não há um só número que  torne ambas verdadeiras ao mesmo tempo. A segunda decisão é responsável por  decidir a impressão da mensagem do carro velho ❷.

Embora óbvio que um carro não poderia ter valores negativos como idade, o  programa não trata desse problema. Vamos alterá-lo mais adiante para verificar  valores inválidos.

**Exercício 4.2** Escreva um programa que pergunte a velocidade do carro de um  usuário. Caso ultrapasse 80 km/h, exiba uma mensagem dizendo que o usuário  foi multado. Nesse caso, exiba o valor da multa, cobrando R$ 5 por km acima de  80 km/h.

Um bloco de linhas em Python pode ter mais de uma linha, embora o último  exemplo mostre apenas dois blocos com uma linha em cada. Se você precisar  de duas ou mais no mesmo bloco, escreva essas linhas na mesma direção ou na  mesma coluna da primeira linha do bloco. Isso basta para representá-lo.

Um problema comum é quando temos que pagar Imposto de Renda. Normalmen te, pagamos o Imposto de Renda por faixa de salário. Imagine que para salários  menores que R$ 1.000,00 não teríamos imposto a pagar, ou seja, alíquota 0%. Para  salários entre R$ 1.000,00 e R$ 3.000,00 pagaríamos 20%. Acima desses valores,  a alíquota seria de 35%. Esse problema se pareceria muito com o anterior, salvo

**78 Introdução à Programação com Python**

se o imposto não fosse cobrado diferentemente para cada faixa, ou seja, quem  ganha R$ 4.000,00 tem os primeiros R$ 1.000,00 isentos de imposto; com o mon tante entre R$ 1.000,00 e R$ 3.000,00 pagando 20%, e o restante pagando os 35%.  Vejamos a solução na listagem do programa 4.4.

► **Listagem 4.4 – Cálculo do Imposto de Renda**

salário=float(**input**("Digite o salário para cálculo do imposto: ")) base = salário ❶

imposto = 0

**if** base > 3000: ❷

imposto = imposto + ((base - 3000) \* 0.35) ❸

base = 3000 ❹

**if** base > 1000: ❺

imposto = imposto + ((base - 1000) \* 0.20) ❻

**print**("Salário: R$%6.2f Imposto a pagar: R$%6.2f" % (salário, imposto))

O programa da listagem 4.4 é bem interessante. Tente executá-lo algumas vezes e  compare o valor impresso com o valor calculado por você. Rastreie o programa  e tente entender o que ele faz antes de ler o parágrafo seguinte. Verifique o que  acontece para salários de R$ 500,00, R$ 1.000,00 e R$ 1,500,00.

Em ❶ temos a variável base recebendo uma cópia de salário. Isso é necessário  porque, quando atribuímos um novo valor para uma variável, o valor anterior é  substituído (e perdido se não o guardarmos em outro lugar). Como vamos utilizar  o valor do salário digitado para exibi-lo na tela, não podemos perdê-lo; por isso,  a necessidade de uma variável auxiliar chamada aqui de base.

Em ❷ verificamos se a base é maior que R$ 3.000,00. Se verdadeiro, executamos  as linhas ❸ e ❹. Em ❸, calculamos 35% do valor superior a R$ 3.000,00. O re sultado é armazenado na variável imposto. Como essa variável contém o valor a  pagar para essa quantia, atualizaremos o valor de base para R$ 3.000,00 ❹, pois  o que ultrapassa esse valor já foi tarifado.

Em ❺ verificamos se o valor de base é maior que R$ 1.000,00, calculando 20% de  imposto em ❻, caso verdadeiro.

Vejamos o rastreamento para um salário de R$ 500,00:

|  |  |  |
| --- | --- | --- |
| **salário** | **base** | **imposto** |
| 500 | 500 | 0 |

**Capítulo 4** ■ **Condições 79** Para um salário de R$ 1.500,00:

|  |  |  |
| --- | --- | --- |
| **salário** | **base** | **imposto** |
| 1500 | 1500 | ~~0~~ |
|  |  | 100 |

Para um salário de R$ 3.000,00:

|  |  |  |
| --- | --- | --- |
| **salário** | **base** | **imposto** |
| 3000 | 3000 | ~~0~~ |
|  |  | 400 |

Para um salário de R$ 5.000,00:

|  |  |  |
| --- | --- | --- |
| **salário** | **base** | **imposto** |
| 5000 | ~~5000~~ | ~~0~~ |
|  | 3000 | ~~700~~ |
|  |  | 1100 |

**Exercício 4.3** Escreva um programa que leia três números e que imprima o maior  e o menor.

**Exercício 4.4** Escreva um programa que pergunte o salário do funcionário e calcule  o valor do aumento. Para salários superiores a R$ 1.250,00, calcule um aumento de  10%. Para os inferiores ou iguais, de 15%.

**4.2 else**

Quando há problemas, como a mensagem do carro velho (Listagem 4.3), onde  a segunda condição é simplesmente o inverso da primeira, podemos usar outra  forma de **if** para simplificar os programas. Essa forma é a cláusula **else** para  especificar o que fazer caso o resultado da avaliação da condição seja falso, sem  precisarmos de um novo **if**. Vejamos como ficaria o programa reescrito para usar  **else** na listagem 4.5.

**80 Introdução à Programação com Python**

► **Listagem 4.5 – Carro novo ou velho, dependendo da idade com else**

idade = int(**input**("Digite a idade de seu carro: "))

**if** idade <= 3:

**print**("Seu carro é novo")

**else**: ❶

**print**("Seu carro é velho") ❷

Veja que em ❶ utilizamos “:” após **else**. Isso é necessário porque **else** inicia um  bloco, da mesma forma que **if**. É importante notar que devemos escrever **else** na mesma coluna do **if**, ou seja, com o mesmo recuo. Assim, o interpretador  reconhece que **else** se refere a um determinado **if**. Você obterá um erro caso não  alinhe essas duas estruturas na mesma coluna.

A vantagem de usar **else** é deixar os programas mais claros, uma vez que podemos  expressar o que fazer caso a condição especificada em **if** seja falsa. A linha ❷ só  é executada se a condição idade <= 3 for falsa.

**Exercício 4.5** Execute o programa (Listagem 4.5) e experimente alguns valores.  Verifique se os resultados foram os mesmos do programa anterior (Listagem 4.3).

**Exercício 4.6** Escreva um programa que pergunte a distância que um passageiro  deseja percorrer em km. Calcule o preço da passagem, cobrando R$ 0,50 por km  para viagens de até de 200 km, e R$ 0,45 para viagens mais longas.

**4.3 Estruturas aninhadas**

Nem sempre nossos programas serão tão simples. Muitas vezes, precisaremos  aninhar vários **if** para obter o comportamento desejado do programa. Aninhar,  nesse caso, é utilizar um **if** dentro de outro.

Vejamos o exemplo de calcular a conta de um telefone celular da empresa Tchau.  Os planos da empresa Tchau são bem interessantes e oferecem preços diferen ciados de acordo com a quantidade de minutos usados por mês. Abaixo de 200  minutos, a empresa cobra R$ 0,20 por minuto. Entre 200 e 400 minutos, o preço  é de R$ 0,18. Acima de 400 minutos, o preço por minuto é de R$ 0,15. O programa  da listagem 4.6 resolve esse problema.

**Capítulo 4** ■ **Condições 81**

► **Listagem 4.6 – Conta de telefone com três faixas de preço**

minutos=int(**input**("Quantos minutos você utilizou este mês:"))

**if** minutos < 200: ❶

preço = 0.20 ❷

**else**:

**if** minutos < 400: ❸

 preço = 0.18 ❹

**else**: ❺

 preço = 0.15 ❻

**print**("Você vai pagar este mês: R$%6.2f" % (minutos \* preço))

Em ❶, temos a primeira condição: minutos < 200. Se a quantidade de minutos for  menor que 200, atribuímos 0,20 ao preço em ❷. Até aqui, nada de novo. Observe  que **if** de ❸ está dentro de **else** da linha anterior: dizemos que está aninhado  dentro de **else**. A condição de ❸, minutos < 400, decide se vamos executar a linha  de ❹ ou a de ❻. Observe que **else** de ❺ está alinhado com **if** de ❸. No final,  calculamos e imprimimos o preço na tela. Lembre-se de que o alinhamento do  texto é muito importante em Python.

Vejamos, por exemplo a situação em que cinco categorias são necessárias. Faça mos um programa que leia a categoria de um produto e determine o preço pela  tabela 4.1.

*Tabela 4.1 – Categorias de produto e preço*

|  |  |
| --- | --- |
| **Categoria** | **Preço** |
| 1 | 10,00 |
| 2 | 18,00 |
| 3 | 23,00 |
| 4 | 26,00 |
| 5 | 31,00 |

À esquerda da listagem 4.7, você encontrará os números de linha do programa,  numeradas de 1 a 19. Esses números servem apenas para ajudar o entendimento  da explicação a seguir: lembre-se de não digitá-los.

**82 Introdução à Programação com Python**

► **Listagem 4.7 – Categoria x preço**

1 categoria = int(**input**("Digite a categoria do produto:"))

2 **if** categoria == 1:

3 preço = 10

4 **else**:

5 **if** categoria == 2:

6 preço = 18

7 **else**:

8 **if** categoria == 3:

9 preço = 23

10 **else**:

11 **if** categoria == 4:

12 preço = 26

13 **else**:

14 **if** categoria == 5:

15 preço = 31

16 **else**:

17 **print**("Categoria inválida, digite um valor entre 1 e 5!") 18 preço = 0

19 **print**("O preço do produto é: R$%6.2f" % preço)

Observe que o alinhamento se tornou um grande problema, uma vez que tivemos  que deslocar à direita a cada **else**.

No programa da listagem 4.7, introduzimos o conceito de validação da entrada.  Dessa vez, se o usuário digitar um valor inválido, receberá uma mensagem de erro  na tela. Nada muito prático ou bonito.

Vejamos a execução das linhas dependendo da categoria digitada na tabela 4.2.

*Tabela 4.2 – Linhas executadas*

|  |  |
| --- | --- |
| **Categoria** | **Linhas executadas** |
| 1 | 1,2,3,19 |
| 2 | 1,2,4,5,6,19 |
| 3 | 1,2,4,5,7,8,9,19 |
| 4 | 1,2,4,5,7,8,10,11,12,19 |
| 5 | 1,2,4,5,7,8,10,11,13,14,15,19 |
| outras | 1,2,4,5,7,8,10,11,13,14,16,17,18,19 |