PensePython2e

Tradução do livro Pense em Python (2ª ed.), de Allen B. Downey

View on GitHub

Capítulo 1: A jornada do programa

O objetivo deste livro é ensinar a pensar como um cientista da computação. Esta forma de pensar combina algumas das melhores características da matemática, da engenharia e das ciências naturais. Assim como os matemáticos, os cientistas da computação usam linguagens formais para denotar ideias (especificamente operações de computação). Como engenheiros, eles projetam coisas, reunindo componentes em sistemas e avaliando as opções de melhor retorno entre as alternativas à disposição. Como cientistas, observam o comportamento de sistemas complexos, formam hipóteses e testam previsões.

A habilidade específica mais importante de um cientista da computação é a resolução de problemas. Resolução de problemas significa a capacidade de formular problemas, pensar criativamente em soluções e expressar uma solução de forma clara e precisa. Assim, o processo de aprender a programar é uma oportunidade excelente para exercitar a habilidade de resolver problemas. É por isso que este capítulo se chama “A jornada do programa”.

Em um nível você aprenderá a programar, uma habilidade útil por si mesma. Em outro nível usará a programação como um meio para um fim. Conforme avançarmos, este fim ficará mais claro.

1.1 - O que é um programa?

Um **programa** é uma sequência de instruções que especifica como executar uma operação de computação. A operação de computação pode ser algo matemático, como solucionar um sistema de equações ou encontrar as raízes de um polinômio, mas também pode ser uma operação de computação simbólica, como a busca e a substituição de textos em um documento; ou algo gráfico, como o processamento de uma imagem ou a reprodução de um vídeo.

Os detalhes parecem diferentes em linguagens diferentes, mas algumas instruções básicas aparecem em quase todas as linguagens:

entrada
: Receber dados do teclado, de um arquivo, da rede ou de algum outro dispositivo.

saída
: Exibir dados na tela, salvá-los em um arquivo, enviá-los pela rede etc.

matemática
: Executar operações matemáticas básicas como adição e multiplicação.

execução condicional
: Verificar a existência de certas condições e executar o código adequado.

repetição
: Executar várias vezes alguma ação, normalmente com algumas variações.

Acredite ou não, isto é basicamente tudo o que é preciso saber. Cada programa que você já usou, complicado ou não, é composto de instruções muito parecidas com essas. Podemos então chegar à conclusão de que programar é o processo de quebrar uma tarefa grande e complexa em subtarefas cada vez menores, até que estas sejam simples o suficiente para serem executadas por uma dessas instruções básicas.

1.2 - Execução do Python

Um dos desafios de começar a usar Python é ter que instalar no seu computador o próprio programa e outros relacionados. Se tiver familiaridade com o seu sistema operacional, e especialmente se não tiver problemas com a interface de linha de comando, você não terá dificuldade para instalar o Python. Mas para principiantes pode ser trabalhoso aprender sobre administração de sistemas e programação ao mesmo tempo.

Para evitar esse problema, recomendo que comece a executar o Python em um navegador. Depois, quando você já conhecer o Python um pouco mais, darei sugestões para instalá-lo em seu computador.

Há uma série de sites que ajudam a usar e executar o Python. Se já tem um favorito, vá em frente e use-o. Senão, recomendo o PythonAnywhere. Apresento instruções detalhadas sobre os primeiros passos no link http://tinyurl.com/thinkpython2e.

Há duas versões do Python, o Python 2 e o Python 3. Como elas são muito semelhantes, se você aprender uma versão, é fácil trocar para a outra. Como é iniciante, você encontrará poucas diferenças. Este livro foi escrito para o Python 3, mas também incluí algumas notas sobre o

Python 2.

O **interpretador** do Python é um programa que lê e executa o código Python. Dependendo do seu ambiente, é possível iniciar o interpretador clicando em um ícone, ou digitando python em uma linha de comando. Quando ele iniciar, você deverá ver uma saída como esta:

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

As três primeiras linhas contêm informações sobre o interpretador e o sistema operacional em que está sendo executado, portanto podem ser diferentes para você. Mas é preciso conferir se o número da versão, que é 3.4.0 neste exemplo, começa com 3, o que indica que você está executando o Python 3. Se começar com 2, você está executando (adivinhe!) o Python 2.

A última linha é um prompt indicando que o interpretador está pronto para você digitar o código. Se digitar uma linha de código e pressionar Enter, o interpretador exibe o resultado:

```
>>> 1 + 1
2
```

Agora você está pronto para começar. Daqui em diante, vou supor que você sabe como inicializar o interpretador do Python e executar o código.

1.3 - O primeiro programa

Tradicionalmente, o primeiro programa que se escreve em uma nova linguagem chama-se “Hello, World!”, porque tudo o que faz é exibir as palavras “Hello, World!” na tela. No Python, ele se parece com isto:

```
>>> print('Hello, World!')
```

Este é um exemplo de uma instrução print (instrução de impressão), embora na realidade ela não imprima nada em papel. Ela exibe um resultado na tela. Nesse caso, o resultado são as palavras:

```
Hello, World!
```

As aspas apenas marcam o começo e o fim do texto a ser exibido; elas não aparecem no resultado.

Os parênteses indicam que o print é uma função. Veremos funções no Capítulo 3.

No Python 2, a instrução print é ligeiramente diferente; ela não é uma função, portanto não usa parênteses.

```
>>> print 'Hello, World!'
```

Esta distinção fará mais sentido em breve, mas isso é o suficiente para começar.

1.4 - Operadores aritméticos

Depois do “Hello, World”, o próximo passo é a aritmética. O Python tem operadores, que são símbolos especiais representando operações de computação, como adição e multiplicação.

Os operadores +, - e * executam a adição, a subtração e a multiplicação, como nos seguintes exemplos:

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
O operador / executa a divisão:
>>> 84 / 2
42.0
```

Pode ser que você fique intrigado pelo resultado ser 42.0 em vez de 42. Vou explicar isso na próxima seção.

Finalmente, o operador ** executa a exponenciação; isto é, eleva um número a uma potência:

```
>>> 6 ** 2 + 6
42
```

Em algumas outras linguagens, o ^ é usado para a exponenciação, mas no Python é um

operador bitwise, chamado XOR. Se não tiver familiaridade com operadores bitwise, o resultado o surpreenderá:

```
>>> 6 ^ 2
4
```

Não abordarei operadores bitwise neste livro, mas você pode ler sobre eles em http://wiki.python.org/moin/BitwiseOperators.

1.5 - Valores e tipos

Um valor é uma das coisas básicas com as quais um programa trabalha, como uma letra ou um número. Alguns valores que vimos até agora foram 2, 42.0 e ‘Hello, World!’.

Esses valores pertencem a tipos diferentes: 2 é um número inteiro, 42.0 é um número de ponto flutuante e ‘Hello, World!’ é uma string, assim chamada porque as letras que contém estão em uma sequência em cadeia.

Se não tiver certeza sobre qual é o tipo de certo valor, o interpretador pode dizer isso a você:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Hello, World!')
<class 'str'>
```

Nesses resultados, a palavra “class” [classe] é usada no sentido de categoria; um tipo é uma categoria de valores.

Como se poderia esperar, números inteiros pertencem ao tipo int, strings pertencem ao tipo str e os números de ponto flutuante pertencem ao tipo float.

E valores como ‘2’ e ‘42.0’? Parecem números, mas estão entre aspas como se fossem strings:

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

Então são strings.

Ao digitar um número inteiro grande, alguns podem usar a notação americana, com vírgulas entre grupos de dígitos, como em 1,000,000. Este não é um número inteiro legítimo no Python e resultará em:

```
>>> 1,000,000
(1, 0, 0)
```

O que não é de modo algum o que esperávamos! O Python interpreta 1,000,000 como uma sequência de números inteiros separados por vírgulas. Aprenderemos mais sobre este tipo de sequência mais adiante.

1.6 - Linguagens formais e naturais

As linguagens naturais são os idiomas que as pessoas falam, como inglês, espanhol e francês. Elas não foram criadas pelas pessoas (embora as pessoas tentem impor certa ordem a elas); desenvolveram-se naturalmente.

As linguagens formais são linguagens criadas pelas pessoas para aplicações específicas. Por exemplo, a notação que os matemáticos usam é uma linguagem formal especialmente boa para denotar relações entre números e símbolos. Os químicos usam uma linguagem formal para representar a estrutura química de moléculas. E o mais importante:

As linguagens de programação são idiomas formais criados para expressar operações de computação.

As linguagens formais geralmente têm regras de sintaxe estritas que governam a estrutura de declarações. Por exemplo, na matemática a declaração 3 + 3 = 6 tem uma sintaxe correta, mas não 3 + = 3$6. Na química, H2O é uma fórmula sintaticamente correta, mas 2Zz não é.

As regras de sintaxe vêm em duas categorias relativas a símbolos e estrutura. Os símbolos são os elementos básicos da linguagem, como palavras, números e elementos químicos. Um dos problemas com 3 + = 3$6 é que o $ não é um símbolo legítimo na matemática (pelo menos até onde eu sei). De forma similar, 2Zz não é legítimo porque não há nenhum elemento com a abreviatura Zz.

O segundo tipo de regra de sintaxe refere-se ao modo no qual os símbolos são combinados. A equação 3 + = 3 não é legítima porque, embora + e = sejam símbolos legítimos, não se pode ter

um na sequência do outro. De forma similar, em uma fórmula química o subscrito vem depois do nome de elemento, não antes.

Esta é um@ frase bem estruturada em portuguê$, mas com s*mbolos inválidos. Esta frase todos os símbolos válidos tem, mas estrutura válida sem.

Ao ler uma frase em português ou uma declaração em uma linguagem formal, é preciso compreender a estrutura (embora em uma linguagem natural você faça isto de forma subconsciente). Este processo é chamado de análise.

Embora as linguagens formais e naturais tenham muitas características em comum – símbolos, estrutura e sintaxe – há algumas diferenças:

ambiguidade
: As linguagens naturais são cheias de ambiguidade e as pessoas lidam com isso usando pistas contextuais e outras informações. As linguagens formais são criadas para ser quase ou completamente inequívocas, ou seja, qualquer afirmação tem exatamente um significado, independentemente do contexto.

redundância
: Para compensar a ambiguidade e reduzir equívocos, as linguagens naturais usam muita redundância. Por causa disso, muitas vezes são verborrágicas. As linguagens formais são menos redundantes e mais concisas.

literalidade
: As linguagens naturais são cheias de expressões e metáforas. Se eu digo “Caiu a ficha”, provavelmente não há ficha nenhuma na história, nem nada que tenha caído (esta é uma expressão para dizer que alguém entendeu algo depois de certo período de confusão). As linguagens formais têm significados exatamente iguais ao que expressam.

Como todos nós crescemos falando linguagens naturais, às vezes é difícil se ajustar a linguagens formais. A diferença entre a linguagem natural e a formal é semelhante à diferença entre poesia e prosa, mas vai além:

Poesia
: As palavras são usadas tanto pelos sons como pelos significados, e o poema inteiro cria um efeito ou resposta emocional. A ambiguidade não é apenas comum, mas muitas vezes proposital.

Prosa
: O significado literal das palavras é o mais importante e a estrutura contribui para este

significado. A prosa é mais acessível à análise que a poesia, mas muitas vezes ainda é ambígua.

Programas
: A significado de um programa de computador é inequívoco e literal e pode ser entendido inteiramente pela análise dos símbolos e da estrutura.

As linguagens formais são mais densas que as naturais, então exigem mais tempo para a leitura. Além disso, a estrutura é importante, então nem sempre é melhor ler de cima para baixo e da esquerda para a direita. Em vez disso, aprenda a analisar o programa primeiro, identificando os símbolos e interpretando a estrutura. E os detalhes fazem diferença. Pequenos erros em ortografia e pontuação, que podem não importar tanto nas linguagens naturais, podem fazer uma grande diferença em uma língua formal.

1.7 - Depuração

Os programadores erram. Por um capricho do destino, erros de programação são chamados de bugs (insetos) e o processo de rastreá-los chama-se depuração (debugging).

Programar, e especialmente fazer a depuração, às vezes traz emoções fortes. Se tiver dificuldade com certo bug, você pode ficar zangado, desesperado ou constrangido.

Há evidências de que as pessoas respondem naturalmente a computadores como se fossem pessoas. Quando funcionam bem, pensamos neles como parceiros da equipe, e quando são teimosos ou grosseiros, respondemos a eles do mesmo jeito que fazemos com pessoas grosseiras e teimosas (Reeves e Nass, *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places* — *A equação da mídia: como as pessoas tratam os computadores, a televisão e as novas mídias como se fossem pessoas e lugares reais*).

Prepare-se para essas reações, pois isso pode ajudar a lidar com elas. Uma abordagem é pensar no computador como um funcionário com certas vantagens, como velocidade e precisão, e certas desvantagens, como a falta de empatia e a incapacidade de compreender um contexto mais amplo.

Seu trabalho é ser um bom gerente: encontrar formas de aproveitar as vantagens e atenuar as desvantagens. E também encontrar formas de usar suas emoções para lidar com o problema sem deixar suas reações interferirem na sua capacidade de trabalho.

Aprender a depurar erros pode ser frustrante, mas é uma habilidade valiosa, útil para muitas atividades além da programação. No fim de cada capítulo há uma seção como esta, com as

minhas sugestões para fazer a depuração. Espero que sejam úteis!

1.8 - Glossário

resolução de problemas
: O processo de formular um problema, encontrar uma solução e expressá-la.

linguagem de alto nível
: Uma linguagem de programação como Python, que foi criada com o intuito de ser fácil para os humanos escreverem e lerem.

linguagem de baixo nível
: Uma linguagem de programação criada para o computador executar com facilidade; também chamada de “linguagem de máquina” ou “linguagem assembly”.

portabilidade
: A propriedade de um programa de poder ser executado em mais de um tipo de computador.

interpretador
: Um programa que lê outro programa e o executa.

prompt
: Caracteres expostos pelo interpretador para indicar que está pronto para receber entradas do usuário.

programa
: Conjunto de instruções que especificam uma operação de computação.

instrução print
: Uma instrução que faz o interpretador do Python exibir um valor na tela.

operador
: Um símbolo especial que representa uma operação de computação simples como adição, multiplicação ou concatenação de strings.

valor
: Uma das unidades básicas de dados, como um número ou string, que um programa manipula.

tipo

Uma categoria de valores. Os tipos que vimos por enquanto são números inteiros (tipo int), números de ponto flutuante (tipo float) e strings (tipo str).

inteiro
Um tipo que representa números inteiros.

ponto flutuante
Um tipo que representa números com partes fracionárias.

string
Um tipo que representa sequências de caracteres.

linguagem natural
Qualquer linguagem que as pessoas falam e que se desenvolveu naturalmente.

linguagem formal
Qualquer linguagem que as pessoas criaram com objetivos específicos, como representar ideias matemáticas ou programas de computador; todas as linguagens de programação são linguagens formais.

símbolo
Um dos elementos básicos da estrutura sintática de um programa, análogo a uma palavra em linguagem natural.

sintaxe
As regras que governam a estrutura de um programa.

análise
Examinar um programa e sua estrutura sintática.

bug
Um erro em um programa.

depuração
O processo de encontrar e corrigir (depurar) bugs.

1.9 - Exercícios

Exercício 1.1

É uma boa ideia ler este livro em frente a um computador para testar os exemplos durante a leitura.

Sempre que estiver testando um novo recurso, você deve tentar fazer erros. Por exemplo, no programa “Hello, World!”, o que acontece se omitir uma das aspas? E se omitir ambas? E se você soletrar a instrução print de forma errada?

Este tipo de experimento ajuda a lembrar o que foi lido; também ajuda quando você estiver programando, porque assim conhecerá o significado das mensagens de erro. É melhor fazer erros agora e de propósito que depois e acidentalmente.

1. Em uma instrução print, o que acontece se você omitir um dos parênteses ou ambos?
2. Se estiver tentando imprimir uma string, o que acontece se omitir uma das aspas ou ambas?
3. Você pode usar um sinal de menos para fazer um número negativo como -2. O que acontece se puser um sinal de mais antes de um número? E se escrever assim: 2++2?
4. Na notação matemática, zeros à esquerda são aceitáveis, como em 02. O que acontece se você tentar usar isso no Python?
5. O que acontece se você tiver dois valores sem nenhum operador entre eles?

Exercício 1.2

Inicialize o interpretador do Python e use-o como uma calculadora.

1. Quantos segundos há em 42 minutos e 42 segundos?
2. Quantas milhas há em 10 quilômetros? Dica: uma milha equivale a 1,61 quilômetro.
3. Se você correr 10 quilômetros em 42 minutos e 42 segundos, qual é o seu passo médio (tempo por milha em minutos e segundos)? Qual é a sua velocidade média em milhas por hora?

PensePython2e

Tradução do livro Pense em Python (2ª ed.), de Allen B. Downey

View on GitHub

Capítulo 2: Variáveis, expressões e instruções

Um dos recursos mais eficientes de uma linguagem de programação é a capacidade de manipular variáveis. Uma variável é um nome que se refere a um valor.

2.1 - Instruções de atribuição

Uma instrução de atribuição cria uma nova variável e dá um valor a ela:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.141592653589793
```

Esse exemplo faz três atribuições. A primeira atribui uma string a uma nova variável chamada message; a segunda dá o número inteiro 17 a n; a terceira atribui o valor (aproximado) de π a pi.

Uma forma comum de representar variáveis por escrito é colocar o nome com uma flecha apontando para o seu valor. Este tipo de número é chamado de diagrama de estado porque mostra o estado no qual cada uma das variáveis está (pense nele como o estado de espírito da variável). A Figura 2.1 mostra o resultado do exemplo anterior.

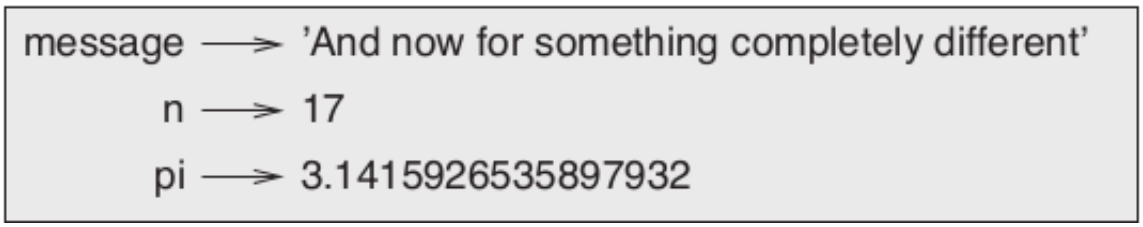

Figura 2.1 – Diagrama de estado.

2.2 - Nomes de variáveis

Os programadores geralmente escolhem nomes significativos para as suas variáveis – eles documentam o uso da variável.

Nomes de variáveis podem ser tão longos quanto você queira. Podem conter tanto letras como números, mas não podem começar com um número. É legal usar letras maiúsculas, mas a convenção é usar apenas letras minúsculas para nomes de variáveis.

O caractere de sublinhar (_) pode aparecer em um nome. Muitas vezes é usado em nomes com várias palavras, como your_name ou airspeed_of_unladen_swallow.

Se você der um nome ilegal a uma variável, recebe um erro de sintaxe:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

`76trombones` é ilegal porque começa com um número. `more@` é ilegal porque contém um caractere ilegal, o `@`. Mas o que há de errado com `class`?

A questão é que class é uma das palavras-chave do Python. O interpretador usa palavras-chave para reconhecer a estrutura do programa e elas não podem ser usadas como nomes de variável.

O Python 3 tem estas palavras-chave:

```
and         del         from        None        True
as          elif        global      nonlocal    try
assert      else        if          not         while
break       except      import      or          with
class       False       in          pass        yield
continue    finally     is          raise
def         for         lambda      return
```

Você não precisa memorizar essa lista. Na maior parte dos ambientes de desenvolvimento, as palavras-chave são exibidas em uma cor diferente; se você tentar usar uma como nome de variável, vai perceber.

2.3 - Expressões e instruções

Uma expressão é uma combinação de valores, variáveis e operadores. Um valor por si mesmo é considerado uma expressão, assim como uma variável, portanto as expressões seguintes são todas legais:

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

Quando você digita uma expressão no prompt, o interpretador a avalia, ou seja, ele encontra o valor da expressão. Neste exemplo, o n tem o valor 17 e n + 25 tem o valor 42.

Uma instrução é uma unidade de código que tem um efeito, como criar uma variável ou exibir um valor.

```
>>> n = 17
>>> print(n)
```

A primeira linha é uma instrução de atribuição que dá um valor a n. A segunda linha é uma instrução de exibição que exibe o valor de n.

Quando você digita uma instrução, o interpretador a executa, o que significa que ele faz o que a instrução diz. Em geral, instruções não têm valores.

2.4 - Modo script

Até agora executamos o Python no modo interativo, no qual você interage diretamente com o interpretador. O modo interativo é uma boa forma de começar, mas se estiver trabalhando com mais do que algumas linhas do código, o processo pode ficar desorganizado.

A alternativa é salvar o código em um arquivo chamado script e então executar o interpretador no modo script para executá-lo. Por convenção, os scripts no Python têm nomes que terminam com .py.

Se souber como criar e executar um script no seu computador, você está pronto. Senão, recomendo usar o PythonAnywhere novamente. Inseri instruções sobre como executar programas no modo script em http://tinyurl.com/thinkpython2e.

Como o Python oferece os dois modos, você pode testar pedaços do código no modo interativo antes de colocá-los em um script. Mas há diferenças entre o modo interativo e o modo script que podem confundir as pessoas.

Por exemplo, se estiver usando o Python como uma calculadora, você poderia digitar:

```
>>> miles = 26.2
>>> miles * 1.61
42.182
```

A primeira linha atribui um valor a miles, mas não tem efeito visível. A segunda linha é uma expressão, então o interpretador a avalia e exibe o resultado. No fim, chega-se ao resultado de que uma maratona tem aproximadamente 42 quilômetros.

Mas se você digitar o mesmo código em um script e executá-lo, não recebe nenhuma saída. Uma expressão, por conta própria, não tem efeito visível no modo script. O Python, na verdade, avalia a expressão, mas não exibe o valor a menos que você especifique:

```
miles = 26.2
print(miles * 1.61)
```

Este comportamento pode confundir um pouco no início.

Um script normalmente contém uma sequência de instruções. Se houver mais de uma instrução, os resultados aparecem um após o outro, conforme as instruções sejam executadas.

Por exemplo, o script

```
print(1)
x = 2
print(x)
```

produz a saída

```
1
2
```

A instrução de atribuição não produz nenhuma saída.

Para verificar sua compreensão, digite as seguintes instruções no interpretador do Python e veja o que fazem:

```
5
x = 5
x + 1
```

Agora ponha as mesmas instruções em um script e o execute. Qual é a saída? Altere o script transformando cada expressão em uma instrução de exibição e então o execute novamente.

2.5 - Ordem das operações

Quando uma expressão contém mais de um operador, a ordem da avaliação depende da ordem das operações. Para operadores matemáticos, o Python segue a convenção matemática. O acrônimo PEMDAS pode ser útil para lembrar das regras:

- Os Parênteses têm a precedência mais alta e podem ser usados para forçar a avaliação de uma expressão na ordem que você quiser. Como as expressões em parênteses são avaliadas primeiro, `2 * (3-1)` é 4, e `(1+1)**(5-2)` é 8. Também é possível usar parênteses para facilitar a leitura de uma expressão, como no caso de `(minute * 100) / 60`, mesmo se o resultado não for alterado.
- A Exponenciação tem a próxima precedência mais alta, então `1 + 2**3` é 9, não 27, e `2 * 3**2` é 18, não 36.
- A Multiplicação e a Divisão têm precedência mais alta que a Adição e a Subtração. Assim, `2 * 3 - 1` é 5, não 4, e `6 + 4 / 2` é 8, não 5.
- Os operadores com a mesma precedência são avaliados da esquerda para a direita (exceto na exponenciação). Assim, na expressão `degrees / 2 * pi`, a divisão acontece primeiro e o resultado é multiplicado por `pi`. Para dividir por 2π, você pode usar parênteses ou escrever `degrees / 2 / pi`.

Eu não fico sempre tentando lembrar da precedência de operadores. Se a expressão não estiver clara à primeira vista, uso parênteses para fazer isso.

2.6 - Operações com strings

Em geral, não é possível executar operações matemáticas com strings, mesmo se elas parecerem números, então coisas assim são ilegais:

```
'2'-'1'    'eggs'/'easy'    'third'*'a charm'
```

Mas há duas exceções, `+` e `*`.

O operador `+` executa uma concatenação de strings, ou seja, une as strings pelas extremidades. Por exemplo:

```
>>> first = 'throat'
>>> second = 'warbler'
>>> first + second
throatwarbler
```

O operador `*` também funciona em strings; ele executa a repetição. Por exemplo, 'Spam' `*` 3 é 'SpamSpamSpam'. Se um dos valores for uma string, o outro tem de ser um número inteiro.

Este uso de + e `*` faz sentido por analogia com a adição e a multiplicação. Tal como `4 * 3` é equivalente a `4 + 4 + 4`, esperamos que `'Spam' * 3` seja o mesmo que 'Spam'+'Spam'+'Spam', e assim é. Por outro lado, há uma diferença significativa entre a concatenação de strings e a repetição em relação à adição e à multiplicação de números inteiros. Você consegue pensar em uma propriedade que a adição tem, mas a concatenação de strings não tem?

2.7 - Comentários

Conforme os programas ficam maiores e mais complicados, eles são mais difíceis de ler. As linguagens formais são densas e muitas vezes é difícil ver um pedaço de código e compreender o que ele faz ou por que faz isso.

Por essa razão, é uma boa ideia acrescentar notas aos seus programas para explicar em linguagem natural o que o programa está fazendo. Essas notas são chamadas de comentários, e começam com o símbolo `#`:

```
# computa a percentagem da hora que passou
percentage = (minute * 100) / 60
```

Nesse caso, o comentário aparece sozinho em uma linha. Você também pode pôr comentários no fim das linhas:

```
percentage = (minute * 100) / 60    # percentagem de uma hora
```

Tudo do `#` ao fim da linha é ignorado – não tem efeito na execução do programa.

Os comentários tornam-se mais úteis quando documentam algo no código que não está óbvio. Podemos supor que o leitor compreenda o que o código faz; assim, é mais útil explicar porque faz o que faz.

Este comentário é redundante em relação ao código, além de inútil:

```
v = 5    # atribui 5 a v
```

Este comentário contém informações úteis que não estão no código:

```
v = 5    # velocidade em metros/segundo.
```

Bons nomes de variáveis podem reduzir a necessidade de comentários, mas nomes longos podem tornar expressões complexas difíceis de ler, então é preciso analisar o que vale mais a pena.

2.8 - Depuração

Há três tipos de erros que podem ocorrer em um programa: erros de sintaxe, erros de tempo de execução e erros semânticos. É útil distinguir entre eles para rastreá-los mais rapidamente.

Erro de sintaxe

A “sintaxe” refere-se à estrutura de um programa e suas respectivas regras. Por exemplo, os parênteses devem vir em pares correspondentes, então (1 + 2) é legal, mas 8) é um erro de sintaxe.

Se houver um erro de sintaxe em algum lugar no seu programa, o Python exibe uma mensagem de erro e para, e não será possível executar o programa. Nas primeiras poucas semanas da sua carreira em programação, você pode passar muito tempo rastreando erros de sintaxe. Ao adquirir experiência, você fará menos erros e os encontrará mais rápido.

Erro de tempo de execução

O segundo tipo de erro é o erro de tempo de execução, assim chamado porque o erro não aparece até que o programa seja executado. Esses erros também se chamam de exceções porque normalmente indicam que algo excepcional (e ruim) aconteceu.

Os erros de tempo de execução são raros nos programas simples que veremos nos primeiros capítulos, então pode demorar um pouco até você encontrar algum.

Erro semântico

O terceiro tipo do erro é "semântico", ou seja, relacionado ao significado. Se houver um erro semântico no seu programa, ele será executado sem gerar mensagens de erro, mas não vai fazer a coisa certa. Vai fazer algo diferente. Especificamente, vai fazer o que você disser para fazer.

Identificar erros semânticos pode ser complicado, porque é preciso trabalhar de trás para a frente, vendo a saída do programa e tentando compreender o que ele está fazendo.

2.9 - Glossário

variável

Um nome que se refere a um valor.

atribuição

Uma instrução que atribui um valor a uma variável.

diagrama de estado

Uma representação gráfica de um grupo de variáveis e os valores a que se referem.

palavra-chave

Uma palavra reservada, usada para analisar um programa; não é possível usar palavras-chave como if, def e while como nomes de variáveis.

operando

Um dos valores que um operador produz.

expressão

Uma combinação de variáveis, operadores e valores que representa um resultado único.

avaliar

Simplificar uma expressão executando as operações para produzir um valor único.

instrução
: Uma seção do código que representa um comando ou ação. Por enquanto, as instruções que vimos são instruções de atribuições e de exibição.

executar
: Executar uma instrução para fazer o que ela diz.

modo interativo
: Um modo de usar o interpretador do Python, digitando o código no prompt.

modo script
: Um modo de usar o interpretador do Python para ler código em um script e executá-lo.

script
: Um programa armazenado em um arquivo.

ordem das operações
: As regras que governam a ordem na qual as expressões que envolvem vários operadores e operandos são avaliadas.

concatenar
: Juntar dois operandos pelas extremidades.

comentários
: Informações em um programa destinadas a outros programadores (ou qualquer pessoa que leia o texto fonte) que não têm efeito sobre a execução do programa.

erro de sintaxe
: Um erro em um programa que torna sua análise impossível (e por isso impossível de interpretar).

exceção
: Um erro que se descobre quando o programa é executado.

semântica
: O significado de um programa.

erro semântico
: Um erro que faz com que um programa faça algo diferente do que o programador pretendia.

2.10 - Exercícios

Exercício 2.1

Repetindo o meu conselho do capítulo anterior, sempre que você aprender um recurso novo, você deve testá-lo no modo interativo e fazer erros de propósito para ver o que acontece.

- Vimos que n = 42 é legal. E 42 = n?
- Ou x = y = 1?
- Em algumas linguagens, cada instrução termina em um ponto e vírgula ;. O que acontece se você puser um ponto e vírgula no fim de uma instrução no Python?
- E se puser um ponto no fim de uma instrução?
- Em notação matemática é possível multiplicar x e y desta forma: xy. O que acontece se você tentar fazer o mesmo no Python?

Exercício 2.2

Pratique o uso do interpretador do Python como uma calculadora:

1. O volume de uma esfera com raio r é $\frac{4}{3} \pi r^3$. Qual é o volume de uma esfera com raio 5?
2. Suponha que o preço de capa de um livro seja R$ 24,95, mas as livrarias recebem um desconto de 40%. O transporte custa R$ 3,00 para o primeiro exemplar e 75 centavos para cada exemplar adicional. Qual é o custo total de atacado para 60 cópias?
3. Se eu sair da minha casa às 6:52 e correr 1 quilômetro a um certo passo (8min15s por quilômetro), então 3 quilômetros a um passo mais rápido (7min12s por quilômetro) e 1 quilômetro no mesmo passo usado em primeiro lugar, que horas chego em casa para o café da manhã?
