Capítulo 1

Por que você deveria aprender a programar?

Escrever programas (ou programar) é uma atividade muito criativa e gratificante. Você pode escrever programas por várias razões, seja para ganhar a vida, para resolver um problema difícil de análises de dados, ou apenas se divertir ajudando alguém a solucionar um problema. Este livro assume que *todos* precisam saber como programar, e, uma vez sabendo, você vai descobrir o que deseja fazer com suas novas habilidades.

Em nosso dia-a-dia, estamos rodeados de computadores, de laptops a celulares. Nós podemos imaginar esses computadores como nossos "assistentes pessoais", que cuidam de muitas de nossas coisas. O hardware dos nossos computadores cotidianos é essencialmente construído para nos perguntar continuamente: "O que você deseja que eu faça agora?".

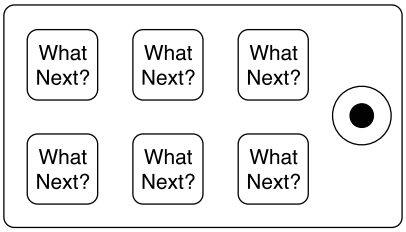

Figure 1.1: Personal Digital Assistant

Os programadores adicionam um sistema operacional e um conjunto de aplicativos ao hardware e nós acabamos tendo um Assistente Pessoal Digital que é bastante útil e capaz de nos ajudar em várias coisas.

Nossos computadores são rápidos e têm vastos espaços de memória e poderiam ser ainda mais úteis se nós {apenas} soubéssemos "falar" a língua para explicar ao computador o que nós "queremos que ele faça agora". Se nós soubéssemos essa linguagem, poderíamos mandar o computador realizar tarefas repetitivas ao nosso querer. Interessantemente, os tipos de coisas que computadores podem fazer melhor são muitas vezes as coisas que nós humanos achamos entendiantes e mentalmente exaustivas.

Por exemplo, leia os três primeiros parágrafos desse capítulo e me diga qual é a palavra mais usada e quantas vezes essa mesma palavra apareceu. Enquanto você era capaz de ler e entender as palavras em poucos segundos, contar o número de vezes que a palavra foi usada é quase que doloroso, pois esse não é o tipo de problema que mentes humanas foram feitas para resolver. Para um computador é o contrário, ler e entender o texto escrito num pedaço de papel é difícil, mas contar as palavras e dizer quantas vezes aquela palavra foi usada é muito fácil:

```
python words.py
Enter file:words.txt
de 8
```

Nosso "assistente pessoal de análises de informação" rapidamente nos diria que a palavra "de" foi usada oito vezes nos três primeiros parágrafos desse capítulo.

Esse mesmo fato de que computadores são bons em coisas que humanos não são é o motivo pelo qual você precisa se tornar hábil em falar a "linguagem computacional". Uma vez que você aprende essa nova linguagem, você pode designar tarefas mundanas para seu parceiro (o computador), sobrando mais tempo para fazer as coisas as quais você é unicamente adequado. Você traz criatividade, intuição e inventividade a essa parceria.

1.1 Criatividade e motivação

Embora este livro não seja destinado a programadores profissionais, a programação profissional pode ser um trabalho muito gratificante, tanto financeiramente quanto pessoalmente. Construir programas úteis, elegantes e inteligentes para os outros usarem é uma atividade que exige muita criatividade. Seu computador ou Personal Digital Assistant (Assistente Pessoal Digital - PDA) geralmente contém diversos programas de diferentes grupos de programadores, cada um competindo por sua atenção e interesse. Eles fazem o melhor para atender às suas necessidades e lhe oferecer uma ótima experiência no processo. Em algumas situações, quando você escolhe um software, os programadores são diretamente compensados por sua escolha.

Se pensarmos nos programas como o resultado da criatividade de grupos de programadores, talvez a seguinte figura seja uma versão mais sensata de nosso PDA:

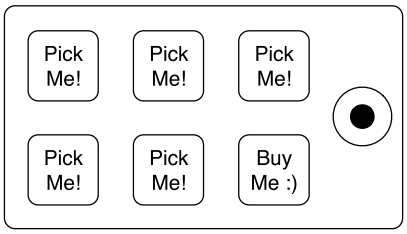

Figure 1.2: Programmers Talking to You

Por enquanto, nossa principal motivação não é ganhar dinheiro ou agradar os usuários finais, mas sim, sermos mais produtivos no manuseio dos dados e das informações que encontraremos em nossas vidas. Quando você começar, você será

o programador e o usuário final de seus programas. À medida que você ganha experiência como programador e a programação lhe parece mais criativa, seus pensamentos podem se voltar ao desenvolvimento de programas para os outros.

1.2 Arquitetura de hardware de computadores

Antes de começarmos a aprender a linguagem que usamos para dar instruções a computadores para o desenvolvimento de software, precisamos saber um pouco sobre como computadores são construídos. Se você por acaso desmontasse seu computador ou celular, encontraria as seguintes partes:

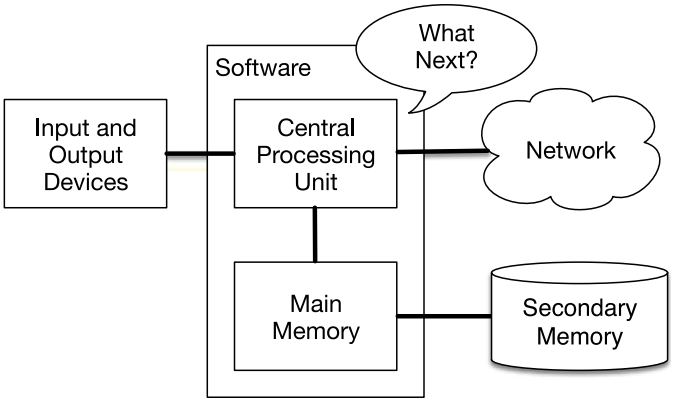

Figure 1.3: Hardware Architecture

As definições em alto nível dessas partes são as seguintes:

- A *Unidade de Processamento Central* (ou CPU) é a parte do computador que é construída para ser obcecada com a pergunta "E agora?". Se seu computador é avaliado em 3.0 Gigahertz, significa que ele irá perguntar "E agora?" três bilhões de vezes por segundo. Você terá que aprender como falar rápido para acompanhar a CPU.

- A *Memória Principal* é utilizada para armazenar informações que a CPU precisa com urgência. Sendo a memória quase tão rápida quanto a CPU. Mas a informação armazenada nessa memória desaparece quando o computador é desligado.

- A *Memória Secundária* também é utilizada para armazenar informação, mas é muito mais lenta que a memória principal. A vantagem é que os dados podem ser guardados até quando o computador está desligado. Exemplos de memória secundária são os discos rígidos ou as memórias flash (tipicamente encontrados em dispositivos USB e em reprodutores de música portáteis).

- Os *Dispositivos de Entrada e Saída* são simplesmente nosso monitor, teclado, mouse, microfone, alto-falante, touchpad, etc. Eles são todas as maneiras que temos para interagir com o computador.

- Atualmente, a maioria dos computadores também tem uma *Conexão de rede* para receber informações de uma rede. Podemos pensar nessa rede como um local muito devagar para armazenar e receber informação que nem sempre

pode estar "à disposição". Então, em resumo, essa rede é uma forma lenta e às vezes não confiável de *Memória Secundária*.

Embora seja melhor deixar para construtores de computadores a maioria dos detalhes de como esses componentes funcionam, ter algumas terminologias ajuda uma vez que podemos falar dessas partes à medida que escrevemos nossos programas.

Como um programador, seu trabalho é usar e orquestrar cada um desses recursos para resolver o problema que você precisa resolver e para analisar as informações obtidas da solução. Você vai, na maioria das vezes, estar "falando" com a CPU e dizendo a ela o que fazer agora. Algumas vezes dirá para usar a memória principal, a memória secundária, a rede, ou os dispositivos de entrada e saída.

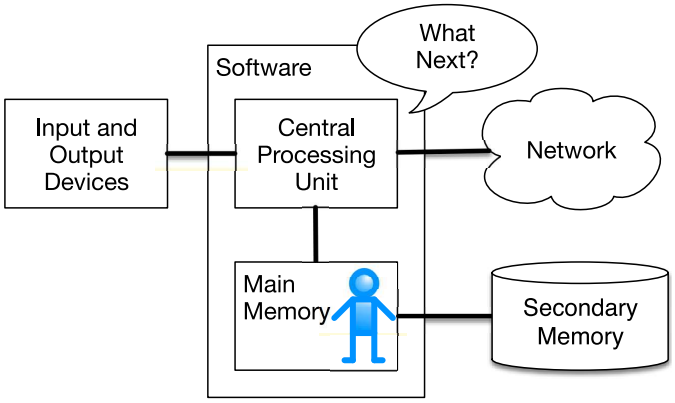

Figure 1.4: Where Are You?

Você precisa ser a pessoa que responde à pergunta "E agora?" da CPU. Mas seria bem desconfortável te encolher a um tamanho de 5 mm só para que você emita um comando três bilhões de vezes por segundo. Então, em vez disso, você deve escrever suas instruções antecipadamente. Chamamos essas instruções armazenadas de *programa* e o ato de escrevê-las e garantir que estejam corretas de *programar*.

1.3 Entendendo Programação

No resto do livro, nós tentaremos te transformar numa pessoa que tem habilidade na arte de programar. No final, você será um *programador* — talvez não um profissional, mas ao menos você terá a capacidade de olhar um problema relacionado a dados e/ou informações e conseguirá desenvolver um programa que solucione o problema.

De um modo ou de outro você precisará de duas habilidades para ser programador:

- Primeiro, você terá que aprender a linguagem de programação (Python), ou seja, você tem que saber o vocabulário e a gramática. Você tem que ser capaz de soletrar as palavras adequadamente nesta nova linguagem de forma a construir "sentenças" bem elaboradas.
- Segundo, você precisa "contar uma história". Quando se conta uma história se combina palavras e sentenças para se transmitir uma ideia ao leitor. Existe uma habilidade e uma arte na construção de uma história e a habilidade de

contá-la é aperfeiçoada quando se escreve e alguém avalia o seu trabalho dando um feedback sobre ele. Na área de programação, o nosso programa é a "história" e o problema que nós estamos querendo solucionar é a "ideia".

A partir do momento que você aprende uma linguagem de programação como Python, você terá muito mais facilidade em aprender uma segunda linguagem de programação como Javascript ou C++. A nova linguagem poderá ter gramática e vocabulário diferentes, mas as habilidades para resolver problemas serão as mesmas através de todas elas.

Você aprenderá o "vocabulário" e as "sentenças" de Python muito rápido. Será mais demorado para você aprender a escrever um programa coerente para solucionar um problema novo. Nós ensinamos programação assim como ensinamos a escrever. Nós começaremos lendo e explicando programas, depois nós escreveremos programas simples e logo após vamos passar para programas cada vez mais complexos conforme o tempo for passando. Em algum ponto você "pegará o gancho" e perceberá padrões por conta própria e verá mais claramente como pegar um problema e escrever um programa que o soluciona. Quando chegar nesse ponto, programar será um processo agradável e criativo.

Começaremos com o vocabulário e a estrutura dos programas em Python. Seja paciente, pois os exemplos te lembrarão como foi ler pela primeira vez.

1.4 Palavras e Frases

Ao contrário das línguas humanas, o vocabulário da Python é realmente muito pequeno. Chamamos de "vocabulário" as palavras "reservadas". Estas são palavras com um significado muito especial para Python. Quando ela as vê em um programa, elas tem um e apenas um significado para Python. Posteriormente, você escreverá programas com suas palavras próprias que chamará de variáveis. Você terá uma grande liberdade na escolha de nomes para as suas variáveis, mas não será possível utilizar as palavras reservadas do Python como um nome para uma variável.

Quando treinarmos um cão, usamos palavras especiais como "senta", "fica" e "pega". Quando você fala com um cão e não utiliza alguma destas palavras reservadas, eles encaram você com uma reação questionável em seu rosto até que você diga uma palavra reservada.
por exemplo, se você diz, "Eu desejo que mais pessoas caminhassem para melhorar sua saúde", o que a maioria dos cães ouve é, "blah blah blah *caminhar* blah blah blah blah." Isso porque "caminhar" é uma palavra reservada na liguagem canina. Muitos podem sugerir que a linguagem entre seres humanos e gatos não tenha palavras reservadas[1].

As palavras reservadas na língua onde os seres humanos falam com Python incluem as seguintes:

```
and       del       global      not       with
as        elif      if          or        yield
assert    else      import      pass
```

[1] http://xkcd.com/231/

```
break     except    in          raise
class     finally   is          return
continue  for       lambda      try
def       from      nonlocal    while
```

É isto, e ao contrário de um cão, Python já está completamente treinado. Quando você diz "tente", ele vai tentar toda vez que você falar, sem falhar. Nós iremos aprender essas palavras reservadas e como elas são usadas, mas por ora vamos nos concentrar no equivalente a "falar" Python (na linguagem entre homem e cachorro). A vantagem de pedir a Python para falar, é que podemos até mesmo dizer-lhe o que falar, dando-lhe uma mensagem em citações:

```
print('Hello world!')
```

E até escrevemos nossa primeira frase sintaticamente correta em Python. Nossa sentença começa com a palavra reservada *print* seguido por uma sequência de texto de nossa escolha entre aspas simples. Este conjunto de caracteres pode ser acompanhado de aspas simples ou duplas, sem distinção do funcionamento da função. Entretanto, é interessante utilizar-se de aspas duplas para os casos em que seja necessário utilizar as aspas simples como um apóstrofo.

1.5 Conversando com Python

Agora que nós já conhecemos uma palavra e uma simples sentença em Python, precisamos saber como iniciar uma conversa com ela para testar nossas novas habilidades linguísticas.

Antes de você poder conversar com Python, você deve primeiro instalar o software do Python no seu computador e aprender a como inicializá-lo. Isso possui detalhes demais para este capítulo, então eu sugiro que você consulte www.py4e.com onde eu tenho instruções detalhadas e screencasts sobre configuração e inicialização do Python nos sistemas Macintosh e Windows. Em certo ponto, você vai estar num terminal ou janela de comando e vai digitar *python* e o interpretador de Python vai começar a executar no modo interativo, onde aparece algo como o seguinte:

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25)
[MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

O >>> prompt é o modo como o interpretador da Python te pergunta, "O que você deseja que eu faça agora?" Ele está pronto para ter uma conversa com você. Tudo o que você tem que saber é como falar a sua linguagem .

Vamos dizer, por exemplo, que você não conheça nem as mais simples palavras ou sentenças da linguagem Python. Você pode querer usar a frase padrão que os astronautas usam quando aterrissam num planeta distante e tentam falar com os habitantes locais::

```
>>>  Venho em paz. Por favor, leve-me ao seu líder
File "<stdin>", line 1
   Venho em paz. Por favor, leve-me ao seu líder
       ^
SyntaxError: invalid syntax
>>>
```

Isso não está indo muito bem. A menos que você pense em algo rápido, provavelmente os habitantes do planeta irão te apunhalar com suas lanças, colocarão você num espeto, te assarão sobre o fogo e comerão sua carne no jantar.

Felizmente, você levou uma cópia deste livro para sua viagem, abre ele nesta exata página e tenta de novo:

```
>>> print('Alô Mundo!')
Alô Mundo!
```

Isso está parecendo bem melhor, então você tenta se comunicar um pouco mais:

```
>>> print (`Você deve ser o lendário Deus que vem do céu')
Você deve ser o lendário Deus que vem do céu
>>> print (`Nós estávamos te esperando há um longo tempo')
Nós estávamos te esperando há um longo tempo
>>> print (`Nossa lenda diz que você fica bastante saboroso com mostarda')
Nossa lenda diz que você fica bastante saboroso com mostarda
>>> print `Nós teremos um banquete esta noite a menos que você diga
  File "<stdin>", line 1
    print `Nós teremos um banquete esta noite a menos que você diga
                                                                   ^
SyntaxError: Missing parentheses in call to 'print'
             (Falta de parênteses no uso de 'print')
>>>
```

A conversa estava indo tão bem por um momento, até que você cometeu o menor erro na linguagem Python e ele pegou as lanças de volta.

A essa altura, você também deve ter percebido que, enquanto Python é maravilhosamente complexo e poderoso e muito exigente sobre a sintaxe que você usa para se comunicar com ele, o mesmo *não* é inteligente. Você está apenas tendo uma conversa consigo mesmo, porém usando uma sintaxe apropriada.

De certo modo, quando você usa um programa escrito por outras pessoas, a conversa é entre você e esses outros programadores, com o Python atuando como intermediário. Ele é um meio pelo qual os criadores de programas expressam como a conversa deve proceder. E em alguns poucos capítulos, você será um desses programadores usando-a para conversar com os usuários do seu programa.

Antes de nós acabarmos nossa primeira conversa com o interpretador de Python, você provavelmente deve saber como, adequadamente, dizer “good-bye”, quando se está interagindo com os habitantes do planeta Python:

```
>>> good-bye
(tchau)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'good' is not defined
>>> if you don't mind, I need to leave
(se você não se importa, eu preciso sair)
File "<stdin>", line 1
  if you don't mind, I need to leave
           ^
SyntaxError: invalid syntax
>>> quit()
```

Você irá perceber que os erros são diferentes para cada uma das duas falhas tentativas. O segundo erro é diferente porque *if* é uma palavra reservada e o Python viu isso e pensou que estávamos tentando dizer algo, mas teríamos errado na sintaxe da frase.

A maneira certa se despedir do Python é inserir *quit()* na parte interativa do >>> prompt. Provavelmente, você teria demorado um pouco para adivinhar essa, então ter um livro em mãos será bem útil.

1.6 Terminologia: Interpretador e Compilador

Python é uma linguagem de *alto-nível* que pretende ser relativamente simples para um humano ler e escrever e para um computador ler e processar. Algumas outras linguagens de alto-nível são: Java, C++, PHP, Ruby, Basic, Perl, JavaScript, and many more. O Hardware dentro da Unidade de Processamento Central (CPU) não entende nenhuma dessas linguagens de alto-nível.

A CPU entende uma linguagem que chamamos de *linguagem de máquina*. Ela é muito simples e francamante bem cansativa de escrever, uma vez que é representada totalmente por zeros e uns.

```
00101000111010010010101010000001111
11100110000011101010010101101101
...
```

Linguagem de máquina parece ser bem simples superficialmente, visto que somente há zeros e uns, mas sua sintaxe é bem mais complexa e muito mais complicada que Python. Então, poucos trabalhadores escrevem nessa linguagem. Em vez disso, construímos vários tradutores para permitir que programadores escrevam em linguagens de alto-nível, como Python ou JavaScript, e esses tradutores convertem os programas para linguagem de máquina para a real execução pela CPU.

Uma vez que a linguagem de máquina está amarrada ao hardware do computador, ela não é *portátil* para diferentes tipos de hardware. Programas escritos em linguagens de alto-nível podem ser movidos entre diferentes computadores usando um diferente interpretador na nova máquina ou recompilando o código para criar uma versão da linguagem de máquina para o novo computador.

Esses tradutores de linguagem de programação caem em duas categorias gerais: (1) interpretadores e (2) compiladores.

Um * interpretador* lê o código-fonte do programa como o programador escreveu, analisa o código e interpreta as instruções em tempo real. %%Aqui adaptei um pouco visto a repetição excessiva da palavra Python, que não cai muito bem no português Python é um interpretador e quando estamos executando Python interativamente, podemos digitar uma linha (uma sentença) e ela é processada imediatamente e já podemos digitar outra linha de Python.

Algumas das linhas de Python falam para ele que você quer aquilo para lembrar de algum valor para depois. Precisamos escolher um nome para esse valor que será lembrado e podemos usar esse nome simbólico para recuperar o valor depois. Usamos o termo *variável* para nos referir a esses rótulos que se referem à informação armazenada.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

Nesse exemplo, pedimos para o Python que lembre o valor seis e use o rótulo x para que possamos recuperar o valor depois. Nós verificamos que ele realmente lembrou o valor usando *print*. Então, pedimos que o interpretador recupere x e multiplique por sete, colocando o valor recentemente computado em y. Então, pedimos ao Python para mostrar o valor que está atualmente em y.

Apesar de escrevermos esses comandos em Python uma linha por vez, ele trata os comandos como uma sequência ordenada de afirmações com afirmações posteriores capazes de recuperar informações criadas em afirmações anteriores. Estamos escrevendo nosso primeiro parágrafo simples com quatro sentenças em uma ordem lógica e significativa.

É da natureza de um *interpretador* ser capaz de ter uma conversa interativa como mostrada acima. Um *compilador* precisa receber todo o programa em um arquivo, e então executa um processo para traduzir o código-fonte de alto-nível para linguagem de máquina e então o compilador põe a linguagem de máquina resultante em um arquivo para posterior execução.

Se você possui um sistema Windows, frequentemente esses programas executáveis em linguagem de máquina têm um sufixo “.exe” ou “.dll” que correspondem a “executável” e “biblioteca de vínculo dinâmico” respectivamente. No Linux e Macintosh, não há sufixo que marque exclusivamente um arquivo como executável.

Se você for abrir um arquivo executável em um editor de texto, ele vai parecer completamente louco e será ilegível:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
```

```
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

Não é fácil ler ou escrever em linguagem de máquina, então é ótimo termos os *interpretadores* e *compiladores* que nos permitem escrever em linguagens de alto-nível como Python ou C.

Agora, a esse ponto de nossa discussão de compiladores e interpretadores, você deveria estar se perguntando um pouco sobre o interpretador Python em si. Em qual linguagem ele é escrito? Ele é escrito em uma linguagem compilada? Quando nós digitamos "python", o que está exatamente acontecendo?

O interpretador Python é escrito em uma linguagem de alto-nível chamada C. Você pode dar uma olhada no código-fonte real do interpretador Python indo em www.python.org e trilhando seu caminho pelo código-fonte. Então, Python é propriamente um programa e é compilado em código de máquina. Quando você instalou o Python no seu computador (ou o fornecedor instalou), você copiou uma cópia de um código de máquina do programa Python traduzido no seu sistema. No Windows, o código de máquina executável para o Python em si está provavelmente em um arquivo com um nome do tipo:

```
C:\Python35\python.exe
```

Isso é mais do que você realmente precisa saber para ser um programador de Python, mas às vezes vale a pena responder a essas perguntas irritantes logo no começo.

1.7 Escrevendo um Programa

Escrever comandos no Interpretador de Python é um excelente meio de experimentar os recursos da linguagem, mas não é recomendado para resolução de problemas complexos.

Quando queremos escrever um programa, nós utilizamos um editor de texto para escrever as instruções em Python em um arquivo, isto é chamado de um *script*. Por convenção, scripts em Python possuem nomes que terminam com `.py`.

Para executar o script, você fala ao interpretador de Python o nome do arquivo. Em uma janela de comando, você escreveria `python Alo.py` como a seguir:

```
$ cat Alo.py
print(''Alô Mundo!')
$ python Alo.py
Alô Mundo!
```

O "$" é o prompt do sistema operacional, e o "cat Alo.py" nos mostra que um arquivo "Alo.py" contém um programa em Python de uma linha que imprime uma string.

Nós chamamos o interpretador de Python e falamos com ele para que leia o código-fonte do arquivo "Alo.py", ao invés de nos pedir linhas de código Python interativamente.

Você notará que não há necessidade de *quit()* no final do programa. Quando o Python está lendo seu código-fonte a partir de um arquivo, ele sabe parar quando chega ao final do script.

1.8 O que é um programa?

A definição de um *programa* em seu básico é uma sequência de instruções do Python que foram criadas para fazer algo. Até mesmo nosso simples script *Alo.py* é um programa. Trata-se de um programa de uma única linha e particularmente não é útil, mas na definição mais estrita, é sim um programa em Python.

Pode ser mais fácil entender o que é um programa pensando em um problema que um programa poderia ser criado para resolver, e então olhar para um que solucionaria este problema.

Digamos que você está fazendo uma pesquisa de Computação Social em posts do Facebook e que você está interessado nas palavras mais usadas frequentemente em uma série de posts. Você poderia imprimir o conteúdo dos posts do Facebook e examinar o texto procurando pela palavra mais comum, mas isso levaria muito tempo e seria muito propenso a erros. Seria mais inteligente escrever um programa em Python para lidar com esta tarefa, de forma rápida e precisa, para então você poder passar o fim de semana fazendo algo divertido.

Por exemplo, veja o seguinte texto sobre um palhaço e um carro. Veja o texto e descubra qual é a palavra mais comum e quantas vezes ela aparece.

```
O palhaço correu atrás do carro e o carro entrou na tenda e a
tenda caiu sobre o palhaço e o carro.
```

Imagine então que você está fazendo essa tarefa olhando para milhões de linhas de texto. Francamente seria mais rápido para você aprender Python e escrever um programa nesta linguagem para fazer a contagem do que seria examinar manualmente as palavras.

A boa notícia é que eu já escrevi um programa simples para encontrar a palavra mais comum em um arquivo de texto. Eu escrevi, testei, e agora o estou lhe dando para você usar e economizar algum tempo.

```
name = input('Enter file:')
handle = open(name, 'r')
counts = dict()

for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1
```

```
bigcount = None
bigword = None
for word, count in list(counts.items()):
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)

# Code: http://www.py4e.com/code3/words.py
```

Você nem mesmo precisa saber Python para usar este programa. Você irá precisar passar do Capítulo 10 desse livro para entender completamente as impressionantes técnicas de Python que nós usamos para fazer esse programa. Você é o usuário final, você simplesmente usa o programa e se maravilha com sua inteligência e como isso poupou você de muito esforço manual. Você apenas digita o código em um arquivo chamado *words.py* ou faz o download do código-fonte de http://www.py4e.com/code3/, e o executa.

Este é um bom exemplo de como Python e a Linguagem Python estão atuando como intermediário entre você (o usuário final) e eu (o programador). Python é uma forma para nós trocarmos sequências de instruções úteis (i.e., programas) em uma linguagem comum que pode ser usada por qualquer um que o instale em seu computador. Então nenhum de nós está falando *para o Python*, ao invés disso estamos nos comunicando uns com os outros *através* dele.

1.9 A Construção de blocos de programas

Nos próximos capítulos nós iremos aprender sobre vocabulário, estrutura de sentenças, parágrafos e a estrutura da história do Python. Nós aprenderemos sobre a poderosa capacidade desta linguagem e como integrar essas capacidades em conjunto para criar programas eficientes.

Existem alguns padrões conceituais de baixo nível que usamos para construir programas. Essas construções não são apenas para programas em Python, elas fazem parte de todas as linguagens de programação, desde a linguagem da máquina até as linguagens de alto nível.

entrada (input) Obter dados do "mundo externo". Isso pode ser ler dados de um arquivo, ou até de algum tipo de sensor como um microfone ou GPS. Nos nossos programas iniciais, nossa entrada virá do usuário digitando dados em seu teclado.

saída (output) Mostrar resultados de um programa numa tela, ou guardá-los em um arquivo, ou talvez gravá-los em um dispositivo como um alto falante, para que ele toque música ou fale algum texto.

execução sequencial Realizar instruções uma após a outra na ordem na qual são encontradas no script.

execução condicional Checar certas condições para que uma certa sequência de instruções seja executada ou ignorada. execução repetitiva:] Realizar algum conjunto de instruções repetidamente, geralmente com alguma variação.

reuso Escrever um conjunto de instruções atribuindo um nome a ele para que estas instruções sejam reutilizadas quando necessárias durante o programa.

Parece quase simples demais para ser verdade, e é claro que nunca é tão simples. É como dizer que andar é simplesmente "colocar um pé na frente do outro". A "arte" de escrever um programa é compor e tecer esses elementos básicos juntos muitas vezes para produzir algo que é útil para seus usuários.

O programa de contagem de palavras acima usa diretamente todos esses conceitos, exceto um.

1.10 O que poderia dar errado?

Como vimos em nossas primeiras conversas com Python, nós devemos nos comunicar precisamente quando estamos escrevendo o código. O menor desvio ou erro fará com que o Python desista de olhar para o seu programa.

Os programadores iniciantes costumam encarar o fato de que Python não deixa espaço para erros como prova de que ele é mau, odioso e cruel. Enquanto Python parece gostar de todos os outros programadores, ele os conhece pessoalmente e guarda rancor contra eles. Por causa desse rancor, Python encara nossos programas perfeitamente escritos e os rejeita como "impróprios", só para nos atormentar.

```
>>> primt 'Olá Mundo!'
File "<stdin>", line 1
  primt 'Olá Mundo!'
                ^
SyntaxError: invalid syntax (sintaxe inválida)
>>> primt ('Olá Mundo')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'primt' is not defined (nome 'primpt' não está definido)
>>> Eu te odeio Python!
File "<stdin>", line 1
  Eu te odeio Python!
     ^
SyntaxError: invalid syntax (sintaxe inválida)
>>> se você saísse daí, eu te ensinaria uma lição
File "<stdin>", line 1
  se você saísse daí, eu te ensinaria uma lição
        ^
SyntaxError: invalid syntax (sintaxe inválida)
>>>
```

Há pouco a ser ganho por discutir com o Python. É apenas uma ferramenta. Ele não tem emoções e é feliz e pronto para atendê-lo sempre que você precisar dele. Suas mensagens de erro soam ásperas, mas elas são apenas pedidos de ajuda. Ele olhou para o que você digitou, e simplesmente não pôde entender o que você inseriu.

Python é muito mais como um cão, te amando incondicionalmente, tendo algumas Palavras-chave que ele entende, olhando para você com um olhar doce em sua cara (>>>), e esperando que você diga algo que ele compreenda. Quando Python diz "SyntaxError: invalid syntax" (Erro de sintaxe: sintaxe inválida), ele está simplesmente abanando sua cauda e dizendo: "você parecia dizer alguma coisa, só que eu não entendi o que você quis dizer, mas por favor continue falando comigo (>>>)."

A medida que seus programas se tornam cada vez mais sofisticados, você vai encontrar três tipos gerais de erros:

Erros de Sintaxe Estes são os primeiros erros que você vai cometer e os mais fáceis para corrigir. Um erro de sintaxe significa que você violou as regras da "gramática" do Python. Assim, ele faz o seu melhor para apontar diretamente para a linha e caractere em que ele percebeu estar confuso. A única complicação nos erros de sintaxe é que às vezes o erro se encontra na verdade um pouco antes de onde ele havia *notado* que estava confuso. Assim, a linha e caractere que o Python indica pode ser apenas o ponto de partida da sua investigação.

Erros de Lógica Um erro lógico é quando o programa tem uma boa sintaxe, mas há um erro na ordem das afirmações ou talvez um erro na forma como as afirmações se relacionam entre si. UM bom exemplo de um erro lógico pode ser, "abra sua garrafa e beba um pouco d'água, coloque-a em sua mochila, caminhe até a biblioteca e, somente em seguida, tampe a garrafa."

Erros de Semântica Um erro semântico é quando a descrição dos passos a serem tomados é sintaticamente perfeita e na ordem certa, mas há um erro no programa. Ele está perfeito, mas não faz o que você pretendia que fizesse. Um exemplo simples seria se você estivesse dando a uma pessoa direções para um restaurante e dissesse: "... quando você chegar no cruzamento com o posto de gasolina, vire à esquerda e vá um kilômetro de distância e o restaurante é um edifício vermelho à sua esquerda." Seu amigo está muito atrasado e te liga para dizer que eles estão em uma fazenda, andando por trás de um celeiro, sem sinal de um restaurante.
Então você diz "você virou à esquerda ou direita no posto de gasolina?" e Ele diz, "Eu segui suas direções perfeitamente, eu as escrevi todas: diz para virar à esquerda e seguir um kilômetro em direção ao posto de gasolina." Então você diz, "Eu sinto muito, porque enquanto minhas instruções foram sintaticamente corretas, elas infelizmente continham um erro semântico pequeno, mas não detectado."

Novamente em todos os três tipos de erros, Python está tentando ao máximo apenas fazer exatamente o que você pediu.

1.11 Debugging

Quando seu Python emite um erro ou até mesmo quando ele te dá um resultado diferente do esperado, você inicia a "caça" em busca da causa de tal problema. Debugging é o processo de encontrar essa causa em seu código. Quando você está fazendo o *Debugging* em seu programa, especialmente se for um erro difícil de encontrar, existem quatro coisas a se tentar:

Ler Leia seu código. Fale-o em voz alta para si próprio e verifique se ele diz mesmo o que você quer que ele diga.

Testar Experimente fazer mudanças e rodar diferentes versões do seu programa. Geralmente, quando você coloca a coisa certa no lugar certo, o problema se torna óbvio, porém, às vezes, você precisará levar um tempo para encontrar o que deve ser ajustado.

Refletir Tire um tempo para pensar! Que tipo de erro está acontecendo: sintaxe, semântica, em tempo de execução? Que informações você pode extrair das mensagens de erro, ou da saída do programa? Que tipo de erro poderia causar o problema que você está vendo? Qual foi a última coisa que você alterou antes do problema aparecer?

Retroceder Em certo momento, o melhor a se fazer é voltar atrás. Desfazer mudanças recentes, até retornar a um programa que funcione e que você o entenda. Depois você pode começar a reconstruir o código.

Programadores iniciantes às vezes ficam emperrados em uma dessas atividades e acabam esquecendo as outras. Para encontrar um erro complexo é preciso ler, testar, refletir e retroceder. Se você ficar preso em uma dessas, tente as outras. Cada uma corresponde a um estilo de problema diferente.

Por exemplo, ler seu código pode ajudar caso o problema seja um erro tipográfico, mas não se ele for uma má compreensão conceitual. Se você não entende o que o programa faz, você pode lê-lo 100 vezes e nunca verá o erro, porque o erro está na sua cabeça.

Testar diferentes versões pode ajudar, principalmente se forem testes simples e pequenos. Porém, se você faz os testes sem pensar ou ler o seu código, você pode enquadrar-se num padrão que eu chamo de "programação randômica", que é a arte de fazer mudanças aleatórias até que seu programa funcione da maneira certa. Obviamente, programação randômica pode lhe custar um bom tempo.

Você precisa tirar um tempo para pensar. Debugging é uma ciência experimental. Você deveria ter ao menos hipóteses sobre o que é o problema. Se existem duas ou mais possibilidades, tente pensar em um teste que eliminaria uma dessas.

Tirar uma folga ajuda no pensar. Pratique o falar também. Se você explicar o problema para outra pessoa (ou mesmo para si próprio), você às vezes encontrará a resposta antes de terminar a pergunta.

Mas até mesmo as melhores técnicas de *Debugging* irão falhar se houverem muitos erros, ou se o código que você está tentando concertar for muito grande e complicado. Às vezes, o melhor a se fazer é retroceder, simplificando o programa até que você retorne a algo que funcione e que você entenda.

A maioria dos programadores iniciantes são relutantes a retroceder, porque eles não suportam o fato de deletar uma linha de código (mesmo se estiver errada). Se isso te fizer se sentir melhor, copie seu código em outro arquivo antes de começar a desmontá-lo. Dessa maneira, você pode colar as peças de volta em pouquíssimo tempo.

1.12 A jornada do aprendizado

Conforme for progredindo sobre os tópicos abordados no restante do livro, não se deixe abalar se sentir que os conceitos não estão se encaixando perfeitamente na primeira impressão. Quando você está aprendendo a falar uma língua nova, não é um problema que os seus primeiros anos de prática não passem além do que singelas tentativas gorgolejantes. E está tudo em ordem se você leva um semestre apenas para evoluir de um vocabulário simples para pequenas sentenças, e se levar anos para transformar estas sentenças em parágrafos, e alguns anos a mais para conseguir escrever uma pequena fábula interessante com suas próprias palavras.

Nosso objetivo é que você aprenda Python de uma maneira muito mais rápida, por isso ensinamos tudo simultaneamente ao longo dos próximos capítulos. Todavia, é como aprender uma nova língua que leva um certo tempo para absorver e compreender antes que a sinta natural. Isto pode causar uma desorientação conforme visitamos e revisitamos alguns tópicos com o intuito de lhe fazer observar aquilo de uma visão macroscópica, isto enquanto estamos definindo uma pequena fração do que compõe o imenso todo. Ao passo que o livro for se desenvolvendo linearmente, e, se você está cursando a disciplina, ela também irá progredir linearmente, não hesite de abordar o material de uma maneira completamente diferente e não-linear. Leia coisas do começo ao fim, de acordo com sua necessidade e curiosidade. Ao ler de relance tópicos mais avançados, mesmo sem compreendê-los bem detalhadamente, você pode chegar à um entendimento melhor do "por que?" da programação. Ao revisar o material prévio, e até refazendo exemplos anteriores, você perceberá que na verdade houve um grande aprendizado sobre o material mesmo que o que é estudado atualmente pareça um pouco impenetrável.

Usualmente, quando você está aprendendo sua primeira linguagem de programação, existem alguns momentos maravilhosos de "Ah Hah!", onde você pode se afastar e observar com um martelo e formão nas mãos que o pedaço de pedra que você está trabalhando, está se transformando numa escultura encantadora.

Se algumas vezes a tarefa de programar parecer ser particularmente difícil, não existe vantagem alguma em permanecer a noite em claro encarando o problema. Pare, relaxe, coma alguma coisa, explique o problema verbalmente com alguma pessoa (ou até mesmo o seu cachorro), e somente depois retorne a ele com uma nova perspectiva. Eu lhe garanto que no momento em que você compreender os conceitos de programação presentes neste livro, poderá olhar para trás e observar como tudo isto era realmente fácil e elegante, e que precisou apenas de um pouco do seu esforço e tempo para que tudo pudesse ser absorvido.

1.13 Glossário

análise sintática Processo de examinar um programa e analisar a estrutura sintática.

bug Um erro em um programa.

central processing unit Unidade central de processamento, considerada o coração de qualquer computador. É o que roda o software que escrevemos, também chamado de "CPU" ou "processador".

código de máquina A linguagem de nível mais baixo para software, que é a linguagem que é diretamente executada pela unidade central de processamento (CPU).

código fonte Um programa em uma linguagem de alto nível

compilar Compilar. Ação de traduzir um programa escrito em uma linguagem de alto nível em uma linguagem de baixo nível, tudo em preparação, para a execução posterior.

erro de semântica Um erro em um programa que faz com que, na execução, ele faça algo diferente do que o programador intencionou.

função print Instrução que faz com que o interpretador Python exiba um valor na tela.

interpretar Executar um programa em uma linguagem de alto nível, traduzindo-o uma linha por vez.

linguagem de alto nível Uma linguagem de programação como o Python, projetada para ser fácil para os humanos lerem e escrever.

linguagem de baixo nível Uma linguagem de programação projetada para ser fácil para um computador executar; também chamado de "código de máquina" ou "linguagem de montagem(assembly)".

memória principal. Armazena programas e dados. A memória principal perde sua informação quando a energia é desligada.

memória secundária Armazena programas e dados e retém suas informações mesmo quando a fonte de alimentação está desligada. Geralmente mais lento que a memória principal. Exemplos de memória secundária incluem drives de disco e memória flash em pendrives.

modo interativo Uma maneira de usar o interpretador Python digitando comandos e expressões no prompt.interpret: Para executar um programa em uma linguagem de alto nível, traduzindo-o uma linha por vez.

portabilidade Uma propriedade de um programa que pode ser executado em mais de um tipo de computador.

programa Um conjunto de instruções que especifica a computação a ser executada pela máquina.

prompt Quando um programa exibe uma mensagem e pausa para o usuário digitar alguma entrada para o programa.

resolução de problemas O processo de formular um problema, encontrar uma solução e expressar a resolução.

semântica O significado de um programa.

1.14 Exercícios

Exercício 1: Qual é a função da memória secundária em um computador?

a) Executar toda computação e lógica do programa
b) Recuperar páginas de web através da internet
c) Armazenar informações para o longo prazo,mesmo além de um ciclo de energia
d) Tomar a entrada do usuário

Exercício 2: O que é um programa?

Exercício 3: Qual é a diferença entre um compilador e um interpretador?

Exercício 4: Qual das opções seguintes contém "código de máquina"?

a) O interpretador de Python
b) O teclado
c) Arquivo de origem do Python
d) Um documento de processamento de texto

Exercício 5: O que há de errado com o código a seguir:

```
>>> primt 'Alô mundo!'
File "<stdin>", line 1
  primt 'Alô mundo!'
                    ^
SyntaxError: invalid syntax (sintaxe inválida)
>>>
```

Exercício 6: Onde é armazenada a variável "X" em um computador, após a seguinte linha de código terminar ?

```
x = 123
```

a) Unidade central de processamento
b) Memória principal
c) Memória secundária
d) Dispositivo de entrada
e) Dispositivo de saída

Exercício 7: Qual será o resultado mostrado em tela do seguinte programa:

```
x = 43
x = x + 1
print(x)
```

a) 43
b) 44
c) x + 1
d) Erro, porque x = x + 1 não é possível matematicamente

Exercício 8: Explique cada uma das seguintes estruturas utilizando como exemplo uma habilidade humana: (1) Unidade central de processamento (CPU), (2) Memória principal, (3) Memória secundária, (4) Dispositivo de entrada, (5) Dispositivo de saída. Por exemplo, "O quê do ser humano é equivalente à Unidade Central de Processamento"?

Exercício 9: Como você corrige um "Syntax Error(Erro de Sintaxe)"