Python para Informáticos

Explorando a Informação

Version 2.7.2

Autor: Charles Severance
Tradução: @victorjabur

Histórico de Publicação:

Maio 2015: Checagem editorial obrigado a Sue Blumenberg.

Outubro 2013: Revisão principal dos Capítulos 13 e 14 para mudar para JSON e usar OAuth. Novo capítulo adicionado na Visualização.

Setembro 2013: Livro publicado na Amazon CreateSpace

Janeiro 2010: Livro publicado usando a máquina da Universidade de Michigan Espresso Book.

Dezembro 2009: Revisão principal dos capítulos 2-10 de *Think Python: How to Think Like a Computer Scientist* e escrita dos capítulos 1 e 11-15 para produzir *Python for Informatics: Exploring Information*

Junho 2008: Revisão prncipal, título alterado para *Think Python: How to Think Like a Computer Scientist*.

Agosto 2007: Revisão principal, título alterado para *How to Think Like a (Python) Programmer*.

Abril 2002: Primeira edição de *How to Think Like a Computer Scientist*.

O código fonte LaTeX para a *Think Python: How to Think Like a Computer Scientist* versão deste livro está disponível em `http://www.thinkpython.com`.

Prefácio

Python para Informáticos: Adaptação de um livro aberto

É muito comum que acadêmicos, em sua profissão, necessitem publicar continuamente materiais ou artigos quando querem criar algo do zero. Este livro é um experimento em não partir da estaca zero, mas sim "remixar" o livro entitulado *Think Python: How to Think Like a Computer Scientist* escrito por Allen B. Downey, Jeff Elkner, e outros.

Em dezembro de 2009, quando estava me preparando para ministrar a disciplina **SI502 - Programação para Redes** na Universidade de Michigan para o quinto semestre e decidi que era hora de escrever um livro de Python focado em explorar dados ao invés de entender algoritmos e abstrações. Minha meta em SI502 é ensinar pessoas a terem habilidades na manipulação de dados para a vida usando Python. Alguns dos meus estudantes estavam planejando tornarem-se profissionais em programação de computadores. Ao invés disso, eles escolheram ser bibliotecários, gerentes, advogados, biólogos, economistas, etc., e preferiram utilizar habilmente a tecnologia nas áreas de suas escolhas.

Eu nunca consegui encontrar o livro perfeito sobre Python que fosse orientado a dados para utilizar no meu curso, então eu comecei a escrever o meu próprio. Com muita sorte, em uma reunião eventual três semanas antes de eu começar a escrever o meu novo livro do zero, em um descanso no feriado, Dr. Atul Prakash me mostrou o *Think Python* livro que ele tinha usado para ministrar seu Curso de Python naquele semestre. Era um texto muito bem escrito sobre Ciência da Computação com foco em explicações diretas e simples de se aprender.

Toda a estrutura do livro foi alterada, visando a resolução de problemas de análise de dados de um modo tão simples e rápido quanto possível, acrescido de uma série de exemplos executáveis e exercícios sobre análise de dados desde o início.

Os capítulos 2–10 são similares ao livro *Think Python* mas precisaram de muitas alterações. Exemplos com numeração e exercícios foram substituídos por exercícios orientados a dados. Tópicos foram apresentados na ordem necessária para construir soluções sofisticadas em análise de dados. Alguns tópicos tais como `try` e `except` foram movidos mais para o final e apresentados como parte do capítulo de condicionais. Funções foram necessárias para simplificar a complexidade na manipulação dos programas introduzidos anteriormente nas primeiras

lições em abstração. Quase todas as funções definidas pelo usuário foram removidas dos exemplos do código e exercícios, com exceção do Capítulo 4. A palavra "recursão"[1] não aparece no livro inteiro.

Nos capítulos 1 e 11–16, todo o material é novo, focado em exemplos reais de uso e exemplos simples de Python para análise de dados incluindo expressões regulares para busca e transformação, automação de tarefas no seu computador, recuperação de dados na internet, extração de dados de páginas web, utilização de web services, transformação de dados em XML para JSON, e a criação e utilização de bancos de dados utilizando SQL (Linguagem estruturada de consulta em bancos de dados).

O último objetivo de todas estas alterações é a mudança de foco, de Ciência da Computação para uma Informática que inclui somente tópicos que podem ser utilizados em uma turma de primeira viagem (iniciantes) que podem ser úteis mesmo se a escolha deles não for seguir uma carreira profissional em programação de computadores.

Estudantes que acharem este livro interessante e quiserem se aprofundar devem olhar o livro de Allen B. Downey's *Think Python*. Porque há muita sinergia entre os dois livros, estudantes irão rapidamente desenvolver habilidades na área com a técnica de programação e o pensamento em algoritmos, que são cobertos em *Think Python*. Os dois livros possuem um estilo de escrita similar, é possível mover-se para o livro *Think Python* com o mínimo de esforço.

Com os direitos autorais de *Think Python*, Allen me deu permissão para trocar a licença do livro em relação ao livro no qual este material é baseado de GNU Licença Livre de Documentação para a mais recente Creative Commons Attribution — Licença de compartilhamento sem ciência do autor. Esta baseia-se na documentação aberta de licenças mudando da GFDL para a CC-BY-SA (i.e., Wikipedia). Usando a licença CC-BY-SA, os mantenedores deste livro recomendam fortemente a tradição "copyleft" que incentiva os novos autores a reutilizarem este material da forma como considerarem adequada.

Eu sinto que este livro serve de exemplo sobre como materiais abertos (gratuitos) são importantes para o futuro da educação, e quero agradecer ao Allen B. Downey e à editora da Universidade de Cambridge por sua decisão de tornar este livro disponível sob uma licença aberta de direitos autorais. Eu espero que eles fiquem satisfeitos com os resultados dos meus esforços e eu desejo que você leitor esteja satisfeito com *nosso* esforço coletivo.

Eu quero fazer um agradecimento ao Allen B. Downey e Lauren Cowles por sua ajuda, paciência, e instrução em lidar com este trabalho e resolver os problemas de direitos autorais que cercam este livro.

Charles Severance
www.dr-chuck.com

[1] Com exceção, naturalmente, desta linha.

Ann Arbor, MI, USA
9 de Setembro de 2013

Charles Severance é um Professor Associado à Escola de Informação da Universidade de Michigan.

Tradução:
@victorjabur

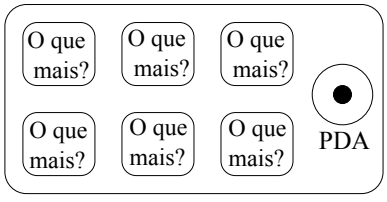
Sumário

Capítulo 1

Por que você deve aprender a escrever programas ?

Escrever programas (ou programação) é uma atividade muito criativa e recompensadora. Você pode escrever programas por muitas razões, que vão desde resolver um difícil problema de análise de dados a se divertir ajudando alguém a resolver um problema. Este livro assume que *qualquer pessoa* precisa saber como programar, e uma vez que você sabe como programar, você irá imaginar o que você quer fazer com suas novas habilidades.

Nós estamos cercados no nosso dia a dia por computadores, desde notebooks até celulares. Nós podemos achar que estes computadores são nossos "assistentes pessoais" que podem cuidar de muitas coisas a nosso favor. O hardware desses computadores no nosso dia a dia é essencialmente construído para nos responder a uma pergunta, "O que você quer que eu faça agora ?"



Programadores adicionam um sistema operacional e um conjunto de aplicações ao hardware e nós terminamos com um Assistente Pessoal Digital que é muito útil e capaz de nos ajudar a fazer diversas coisas.

Nossos computadores são rápidos, tem vasta quantidade de memória e pode ser muito útil para nós, somente se conhecermos a linguagem falada para explicar para um computador o que nós gostaríamos de fazer "em seguida". Se nós conhecemos esta linguagem, nós podemos pedir ao computador para fazer tarefas repetitivas a nosso favor. Curiosamente, as coisas que os computadores podem fazer melhor são frequentemente aquelas coisas que humanos acham chatas e entediantes.

Por exemplo, olhe para os três primeiros parágrafos deste capítulo e me diga qual é a palavra mais usada e quantas vezes. Contá-las é muito doloroso porque não é o tipo de problema que mentes humanas foram feitas para resolver. Para um computador o oposto é verdade, ler e entender o texto de um pedaço de papel é difícil, mas contar palavras dizendo a você quantas vezes ela aparece é muito fácil:

```
python palavras.py
Digite o nome do arquivo: palavras.txt
para 16
```

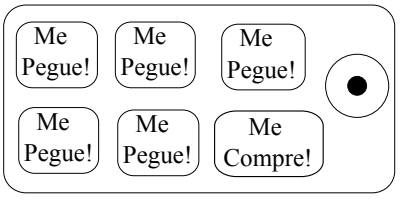
Nosso "assistente de análise pessoal de informações" rapidamente conta para nós que a palavra "para" foi utilizada dezesseis vezes nos primeiros três parágrafos deste capítulo.

Este fato de que os computadores são bons em coisas que humanos não são é a razão pela qual você precisa tornar-se qualificado em falar a "linguagem do computador". Uma vez que você aprende esta nova linguagem, pode delegar tarefas mundanas para o seu parceiro (o computador), ganhando mais tempo para fazer coisas que você foi especialmente adaptado para fazer. Você agrega criatividade, intuição e originalidade para o seu parceiro.

1.1 Criatividade e motivação

Embora este livro não se destine a programadores profissionais, programação profissional pode ser um trabalho muito gratificante, tanto financeiramente quanto pessoalmente. Construir programas úteis, elegantes, inteligentes para que outros utilizem é uma atividade criativa. Seu computador ou assistente pessoal digital (PDA) geralmente contém muitos programas diferentes feitos por diversos grupos de programadores, todos competindo por sua atenção e seu interesse. Eles tentam dar o seu melhor para atender suas necessidades e dar a você uma boa experiência de usabilidade no processo. Em algumas situações, quando você executa um trecho de software, os programadores são diretamente recompensados por sua escolha.

Se nós pensarmos em programas como resultado criativo de grupos de programadores, então talvez a figura a seguir seja uma versão mais sensata de nosso PDA:

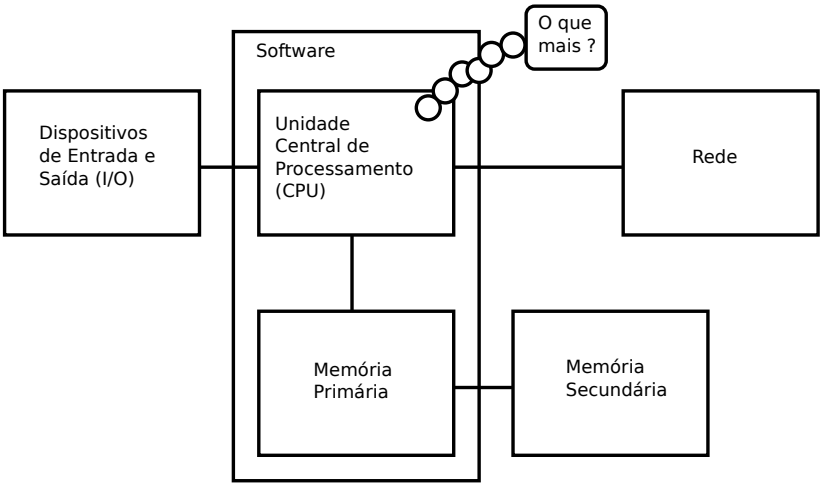


Por enquanto, nossa motivação primária não é ganhar dinheiro ou agradar usuários finais, para nós sermos mais produtivos na manipulação de dados e informações que nós encontraremos em nossas vidas. Quando você começar, você será tanto

o programador quanto o usuário final de seus programas. Conforme você ganhar habilidades como programador e melhorar a criatividade em seus próprios programas, mais você pode pensar em programar para os outros.

1.2 Arquitetura física do Computador - Hardware

Antes de começar a estudar a linguagem, nós falamos em dar instruções aos computadores para desenvolver software, nós precisamos aprender um pouco mais sobre como os computadores são construídos. Se você desmontar seu computador ou celular e olhar por dentro, você encontrará as seguintes partes:



As definições resumidas destas partes são:

- A **Unidade Central de Processamento** (ou CPU) é a parte do computador que é feita para sempre te perguntar: "O que mais ?" Se seu computador possui uma frequência de 3.0 Gigahertz, significa que a CPU irá te perguntar "O que mais ?" três bilhões de vezes por segundo. Você irá aprender como conversar tão rápido com a CPU.

- A **Memória Principal** é utilizada para armazenar informação que a CPU precisa com muita pressa. A memória principal é aproximadamente tão rápida quanto a CPU. Mas a informação armazenada na memória principal se perde quando o computador é desligado (volátil).

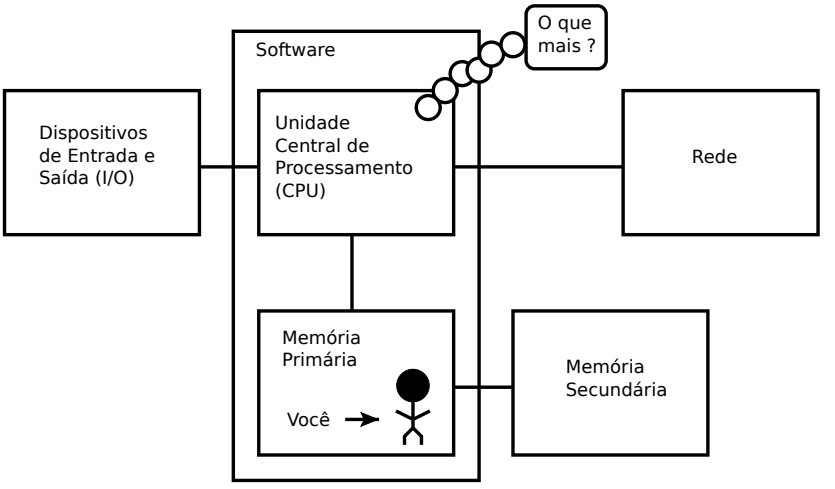
- A **Memória Secundária** é também utilizada para armazenar informação, mas ela é muito mais lenta que a memória principal. A vantegem da memória secundária é que ela pode armazenar informação que não se perde quando o computador é desligado. Exemplos de memória secundária são discos rígidos (HD), pen drives, cartões de memória (sd card) (tipicamente) encontradas no formato de USB e portáteis.

- Os **Dispositivos de Entrada e Saídas** são simplesmente nosso monitor (tela), teclado, mouse, microfone, caixa de som, touchpad, etc. Eles são todas as formas com as quais interagimos com o computador.

- Atualmente, a maioria dos computadores tem uma **Conexão de Rede** para buscar informação em uma rede. Nós podemos pensar a rede como um lugar muito lento para armazenar e buscar dados que podem não estar "disponíveis". Em essência, a rede é mais lenta e às vezes parece uma forma não confiável de **Memória Secundária**.

É melhor deixar a maior parte dos detalhes de como estes componentes funcionam para os construtores dos computadores, isso nos ajuda a ter alguma terminologia que podemos utilizar para conversar sobre as diferentes partes nos programas que vamos escrever.

Como um programador, seu trabalho é usar e orquestrar cada um destes recursos para resolver um problema que você precisa resolver e analisar os dados que você obtém da solução. Como um programador você irá "conversar" com a CPU e contar a ela o que fazer em um próximo passo. Algumas vezes você irá dizer à CPU para usar a memória principal, a memória secundária, a rede ou os dispositivos de entrada e saída.



Você precisa ser a pessoa que responde à pergunta "O que mais ?" para a CPU. Mas seria muito desconfortável se você fosse encolhido para uma altura de apenas 5 mm e inserido dentro de um computador e ainda ter que responder uma pergunta três bilhões de vezes por segundo. Então, ao invés disso, você deve escrever suas instruções previamente. Nós invocamos as instruções armazenadas de um **programa** e o ato de escrita dessas instruções precisa ser correto.

1.3 Entendendo programação

No restante deste livro, nós iremos tentar fazer de você uma pessoa com habilidades na arte da programação. No final você será um **programador**, no entanto não um programador profissional, mas pelo menos você terá os conhecimentos para analisar os problemas de dados/informações e desenvolver um programa para resolver tais problemas.

Resumidamente, você precisa de duas qualidades para ser um programador:

- Primeiramente, você precisa conhecer uma linguagem de programação (Python) - você precisa conhecer o vocabulario e a gramática. Você precisa saber pronunciar as palavras desta nova linguagem corretamente e conhecer como construir sentenças bem formadas nesta linguagem.

- Segundo, você precisa "contar uma estória". Na escrita da estória, você combina palavras e sentenças para convencer o leitor. É necessário qualidade e arte na construção da estória, adquiri-se isto através da prática de contar estórias e obter um feedback. Na programação, nosso programa é a "estória" e o problema que você quer resolver é a "idéia".

Uma vez que você aprende uma linguagem de programação, como o Python, você irá achar muito mais fácil aprender a segunda linguagem de programação, tal como JavaScript ou C++. A nova linguagem de programação possuirá um vocabulário e gramática bastante diferente, mas as habilidades na resolução do problemas serão as mesmas em qualquer linguagem.

Você aprenderá o "vocabulário" e "sentenças" do Python rapidamente. Levará muito tempo para você tornar-se hábil em escrever programas coerentes para resolver um novo problema. Nós ensinamos programação assim como ensinamos a escrever. Nós estaremos lendo e explicando programas, nós escreveremos programas simples, e então nós aumentaremos a complexidade dos programas ao longo do tempo. Em algum momento, você "deslancha" e vê os padrões por si próprio e pode visualizar com maior naturalidade como escrever um programa para resolver o problema. Uma vez que você chega neste pontoprogramar torna-se um processo muito agradável e criativo.

Nós iniciamos com o vocabulário e a estrutura de programas em Python. Seja paciente com os exemplos simples, relembre quando você iniciou a leitura pela primeira vez.

1.4 Programas e Sentenças

Diferentemente dos idiomas humanos, o vocabulário do Python é atualmente muito pequeno. Nós chamamos isto de "vocabulário" e "palavras reservadas".

Estas palavras tem um significado especial no Python. Quando o Python encontra estas palavras em um programa, elas possuem um e somente um significado para o Python. Quando você escreve seus programas você irá definir suas próprias palavras com significado, são chamadas **variáveis**. Você pode escolher muitos nomes diferentes para as suas variáveis, mas você não pode usar qualquer palavra reservada do Python como o nome de uma variável.

Quando nós treinamos um cachorro, nós usamos palavras especiais, tais como: "sentado", "fique" e "traga". Quando você conversar com um cachorro e não usar qualquer uma destas palavras reservadas, eles ficarão olhando para você com um olhar curioso até que você diga uma palavra reservada. Por exemplo, se você disser: "Eu desejo que mais pessoas possam caminhar para melhorar a sua saúde", o que os cachorros vão ouvir será: "blah blah blah **walk** blah blah blah blah." Isto porque "caminhar" é uma palavra reservada na linguagem dos cachorros. Muitos podem sugerir que a linguagem entre humanos e gatos não tem palavras reservadas[1].

As palavras reservadas na linguagem onde os humanos conversam com o Python, incluem o seguinte:

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

É isto, e ao contrário do cachorro, o Python é completamente treinado. Quando você diz "try", o Python irá tentar todas as vezes que você pedir sem desobedecer.

Nós aprenderemos as palavras reservadas e como elas são usadas mais adiante, por enquanto nós iremos focar no equivalente ao Python de "falar" (na linguagem humano-para-cachorro). Uma coisa legal sobre pedir ao Python para falar é que nós podemos até mesmo pedir o que nós queremos através de uma mensagem entre aspas:

```
print 'Hello world!'
```

E finalmente nós escrevemos a nossa primeira sentença sintaticamente correta em Python. Nossa sentença inicia com uma palavra reservada **print** seguida por uma cadeia de caracteres textuais de sua escolha entre aspas simples.

1.5 Conversando com Python

Agora que você tem uma palavra e uma simples sentença que nós conhecemos em Python, nós precisamos saber como iniciar uma conversação com Python para testar nossas habilidades na nova linguagem.

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

Antes de você conversar com o Python, você deve primeiramente instalar o programa Python em seu computador e aprender como inicializá-lo. Isto é muita informação para este capítulo, então eu sugiro que você consulte www.pythonlearn.com onde se encontra instruções e screencasts de preparação e inicialização do Python em sistemas Windows e Macintosh. Em algum momento, você estará no interpretador Python e estará executando o modo interativo e aparecer algo assim:

```
Python 2.6.1 (r261:67515, Jun 24 2010, 21:47:49)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

O >>> prompt é a forma do interpretador Python perguntar o que você deseja: "O que você quer que eu faça agora ?" Python está pronto para ter uma conversação com você. Tudo o que você deve conhecer é como falar a linguagem Python.

Digamos, por exemplo, que você não conhece nem mesmo as mais simples palavras ou sentenças da linguagem Python. Você pode querer usar a linha padrão que os astronautas usam quando eles estão em uma terra distante do planeta e tentam falar com os habitantes do planeta:

```
>>> Eu venho em paz, por favor me leve para o seu líder
  File "<stdin>", line 1
    Eu venho em paz, por favor me leve para o seu líder
          ^
SyntaxError: invalid syntax
>>>
```

Isto não deu muito certo. A menos que você pense algo rapidamente, os habitantes do planeta provavelmente irão apunhalá-lo com uma lança, colocando-o em um espeto, assando-o no fogo, e comê-lo no jantar.

A sorte é que você trouxe uma cópia deste livro em sua viagem, e caiu exatamente nesta página, tente novamente:

```
>>> print 'Ola Mundo!'
Ola Mundo!
```

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

```
>>> print 'Voce deve ser um Deus lendario que veio do ceu'
Voce deve ser um Deus lendario que veio do ceu
>>> print 'Nos estivemos esperando voce por um longo tempo'
Nos estivemos esperando voce por um longo tempo
>>> print 'Nossa lenda nos conta que voce seria muito apetitoso com mostarda'
Nossa lenda nos conta que voce seria muito apetitoso com mostarda
>>> print 'Nos teremos uma festa hoje a noite a menos que voce diga
  File "<stdin>", line 1
    print 'Nos teremos uma festa hoje a noite a menos que voce diga
                                                                   ^
SyntaxError: EOL while scanning string literal
>>>
```

A conversa foi bem por um momento, até que você cometeu o pequeno erro no uso da linguagem e o Python trouxe a lança de volta.

Até o momento, você deve ter percebido que o Python é incrivelmente complexo e poderoso e muito exigente em relação à sintaxe que você utiliza para se comunicar com ele, Python *não* é inteligente. Você está na verdade tendo uma conversa com você mesmo, mas usando uma sintaxe apropriada.

De certa forma, quando você usa um programa escrito por alguém, a conversa ocorre entre você e os programadores, neste caso o Python atuou como um intermediário. Python é uma forma para que os criadores de programas se expressem sobre como uma conversa deve proceder. E em poucos capítulos, você será um dos programadores usando Python para conversar com os usuários de seus programas.

Antes de sairmos da nossa primeira conversa com o interpretador do Python, você deve conhecer o modo correto de dizer "ate-logo" quando interagir com os habitantes do Planeta Python.

```
>>> ate-logo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ate' is not defined

>>> se voce nao se importa, eu preciso ir embora
  File "<stdin>", line 1
    se voce nao se importa, eu preciso ir embora
          ^
SyntaxError: invalid syntax

>>> quit()
```

Você pode perceber que o erro é diferente nas duas primeiras tentativas incorretas. No primeiro erro, por tratar-se de uma palavra simples, o Python não pode encontrar nenhuma função ou variável com este nome. No segundo erro, existe um erro de sintaxe, não sendo reconhecida a frase como válida.

O jeito correto de se dizer "ate-logo" para o Python é digitar **quit()** no prompt do interpretador interativo. É provável que você tenha perdido certo tempo tentado fazer isso, ter um livro em mãos irá tornar as coisas mais fáceis e pode ser bastante útil.

1.6 Terminologia: interpretador e compilador

Python é uma linguagem **alto nível** cujo objetivo é ser relativamente fácil para humanos ler e escrever e para computadores ler e processar. Outras linguagens alto nível incluem Java, C++, PHP, Ruby, Basic, Perl, JavaScript, e muito mais. O atual hardware dentro da Unidade Central de Processamento (CPU) não é capaz de entender nenhum destes comando em alto nível.

A CPU entende a linguagem que chamamos de **linguagem de máquina**. Linguagem de máquina é muit simples e francamente cansativa de se escrever porque ela é representada em zeros e uns:

```
01010001110100100101010000001111
11100110000011101010010101101101
...
```

Linguagem de máquina parece simples olhando-se de um modo superficial, dado que são apenas zeros e uns, mas sua sintaxe é muito mais complexa e mais distante que o Python. Poucos programadores escrevem linguagem de máquina. Invés disso, nós usamos vários tradutores para permitir que os programadores escrevam linguagem de máquina em linguagens de alto nível como o Python ou o JavaScript e estes tradutores convertem os programas para linguagem de máquina para execução pela CPU.

Visto que linguagem de máquina é vinculada ao hardware do computador, linguagem de máquina não é **portável** entre diferentes tipos de hardware. Programas que foram escritos em linguagens de alto nível podem mover-se entre diferentes computadores usando um interpretador diferente em cada máquina ou então recompilando o código para criar uma versão de linguagem de máquina do programa para a nova máquina.

Os tradutores das linguagens de programação se enquadram em duas características gerais: (1) interpretadores e (2) compiladores

Um **interpretador** lê o código fonte de um programa da forma como foi escrito pelo programador, analisa, e interpreta as instruções em tempo de execução. Python é um interpretador e quando ele está rodando Python no modo interativo, nós podemos digitar uma linha de Python (uma sentença) e o Python processa imediatamente e está ponto para receber outra linha de Python.

Algumas das linhas de Python diz a ele que você quer armazenar algum valor para resgatar depois. Nós precisamos dar um nome para um valor de forma que possa ser armazenado e resgatado através deste nome simbólico. Nós usamos o termo **variável** para se referir aos apelidos que nós demos ao dado que foi armazenado.

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

Neste exemplo, nós pedimos ao Python para armazenar o valor seis e usar um apelido **x** assim nós podemos resgatar o valor mais tarde. Nós verificamos que o Python realmente lembrou dos valores quando usamos a função **print**. Então nós perguntamos ao Python para resgatar **x** e multiplicar por sete a armazenar de novo

em uma variável **y**. Então nós pedimos ao Python para imprimir o valor corrente em **y**.

Mesmo que nós digitemos estes comandos em uma única linha de Python por vez, o Python está processando elas em sequência, mantendo a ordem, de forma que as instruções seguintes consigam recuperar os dados criados pelas anteriores. Nós escrevemos nosso primeiro parágrafo simples com quatro sentenças com uma ordem lógica e com um significado.

É da natureza de um **interpretador** ser capaz de ter uma conversa interativa, como foi mostrado acima. Um **compilador** precisa ter em mãos o programa completo em um arquivo, e então ele roda um processo para traduzir um código fonte em alto nível para uma linguagem de máquina e então o compilador coloca o resultado deste processo em um outro arquivo para posterior execução.

Se você está em um sistema Windows, frequentemente este programa executável em código de máquina tem o sufixo ".exe" ou ".dll", o qual é chamado de "executável" ou "biblioteca de link dinâmico", respectivamente. Em Linux e Macintosh, não há um sufixo que marca um arquivo unicamente como executável.

Se você abrir um arquivo executável em um editor de texto, verá algo completamente doido e 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 nada fácil ler ou escrever código de máquina, assim é bom que tenhamos **interpretadores** e **compiladores** que nos permitam escrever em linguagens de alto nível assim como o Python ou o C.

Agora neste ponto em nossa discussão de compiladores e interpretadores, você deve estar com algumas dúvidas sobre o funcionamento do interpretador Python. Em que linguagem é escrito ? É escrito em uma linguagem compilada ? Quando nós digitamos "python", o que exatamente acontece ?

O interpretador Python é escrito em uma linguagem de alto nível, chamada "C". Você pode dar uma olhada no código fonte atual do interpretador através do endereço `www.python.org` e trabalhar como você quiser com o código fonte. Então Python é um programa compilado em uma linguagem de máquina. Quando você instalou Python em seu computador (ou o fornecedor instalou), você copiou um código de máquina do programa Python traduzido para o seu sistema. Em Windows, o código de máquina executável para o Python encontra-se em um arquivo com um nome a seguir:

```
C:\Python27\python.exe
```

Isto é mais do que você realmente precisa conhecer para ser um programador Python, mas às vezes, isto ajuda a entender questões que intrigam justamente no início.

1.7 Escrevendo um programa

Digitar comandos em um Interpretador Python é uma boa maneira de experimentar as características da linguagem, mas isto não é recomendado para resolver problemas mais complexos.

Quando nós queremos escrever um programa, usamos um editor de texto para escrever as instruções Python em um arquivo, o qual chamamos de **script**. Por convenção, scripts Python tem nomes que terminam com `.py`.

Para executar o script, você tem que dizer ao interpretador do Python o nome do arquivo. Em uma janela de comandos Unix ou Windows, você digita `python hello.py` como a seguir:

```
csev$ cat hello.py
print 'Ola Mundo!'
csev$ python hello.py
Ola Mundo!
csev$
```

O "csev$" é o prompt do sistema operacional, e o "cat hello.py" é para nos mostrar que o arquivo "hello.py" tem uma linha de programa Python para imprimir uma string.

Nós chamamos o interpretador Python e pedimos a ele para ler o código fonte do arquivo "hello.py" ao invés dele nos perguntar quais são as próximas linhas de modo interativo.

Você notará que não é preciso ter o **quit()** no fim do programa Python no arquivo. Quando o Python está lendo o seu código fonte de um arquivo, ele sabe que deve parar quando chegar ao fim do arquivo.

1.8 O que é um programa ?

A definição de um **programa** em sua forma mais básica é uma sequência de comandos Python que foram criados para fazer algo. Mesmo o nosso simples script **hello.py** é um programa. É um programa de uma linha e não é particularmente útil, mas na estrita definição, é um programa Python.

Pode ser mais fácil entender o que é um programa, imaginando qual problema ele foi construído para resolver, e então olhar para o programa que resolve um problema.

Vamos dizer que você está fazendo uma pesquisa de computação social em posts do Facebook e está interessado nas palavras mais frequentes em uma série de posts. Você não pode imprimir o stream de posts do Facebook e debruçar-se sobre o texto procurando pela palavra mais comum, mas pode levar um longo tempo e ser muito propenso a erros. Você pode ser inteligente para escrever um programa Python para tratar disso rapidamente a com acurácia, então você pode passar seu final de semana fazendo algo divertido.

Por exemplo, olhe para o seguinte texto sobre o palhaço e o carro. Olhe para o texto e imagine qual é a palavra mais comum e quantas vezes ela aparece:

```
O palhaço correu atrás do carro e o carro correu para a tenda
e a tenda caiu em cima do palhaço e do carro
```

Então imagine que você está fazendo esta tarefa olhando para milhões de linhas de texto. Francamente será mais rápido para você aprender Python e escrever um programa Python para contar as palavras do que você manualmente escanear as palavras.

A notícia ainda melhor é que eu já fiz para você um simples programa para encontrar a palavra mais comum em um arquivo texto. Eu escrevi, testei e agora eu estou dando isto para que você use e economize algum tempo.

```
name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()

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

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

print bigword, bigcount
```

Você nem precisa conhecer Python para usar este programa. Você precisará chegar até o capítulo 10 deste livro para entender completamente as impressionantes técnicas Python que foram utilizadas para fazer o programa. Você é o usuário final, você simplesmente usa o programa e admira-se com a inteligência e em como ela poupou seus esforços manuais. Você simplesmente digitou o código em um arquivo chamado **words.py** e executou ou então fez o download do código fonte no site `http://www.pythonlearn.com/code/` e executou.

Este é um bom exemplo de como o Python e sua linguagem podem atuar como um intermediário entre você (o usuário final) e eu (o programador). Python é uma

forma para trocarmos úteis sequências de instruções (i.e., programas) em uma linguagem comum que pode ser usada por qualquer que instalar Python em seu computador. Então nenhum de nós está conversando *com o Python* mas sim nos comunicando uns com os outros *através* de Python.

1.9 A construção de blocos de programas

Em poucos capítulos, nós iremos aprender mais sobre o vocabulário, estrutura das sentenças, dos parágrafos e da história do Python. Nós iremos aprender sobre as capacidades poderosas do Python e como compor estas capacidades juntas para criar programas úteis.

Há alguns padrões conceituais de baixo nível que nós usamos para construir programas. Estas construções não são apenas para programas Python, elas são parte de todas as linguagens de programação desde linguagens de baixo nível até as de alto nível.

input: Obter dados do "mundo externo". Estes dados podem ser lidos de um arquivo ou mesmo de algum tipo de sensor como um microfone ou um GPS. Em nossos primeiros programas, nosso input virá de um usuário que digita dados no teclado.

output: Exibe os resultados do programa em uma tela ou armazena-os em um arquivo ou talvez os escreve em algum dispositivo tal como um alto falante para tocar música ou falar o texto.

execução sequencial: Executa instruções uma após a outra respeitando a sequência encontrada no script.

execução condicional: Avalia certas condições e as executa ou pula a sequência de instruções.

execução repetitiva: Executa algumas instruções repetitivamente, geralmente com alguma variação.

reúso: Escrever um conjunto de instruções uma única vez, dar um nome a elas e reusar estas instruções em várias partes de um programa.

Parece simples demais para ser verdade, e naturalmente que isto nunca é tão simples. É como dizer que caminhar é simplesmente "colocar um pé na frente do outro". A "arte" de escrever um programa é compor e costurar estes elementos básicos muitas vezes para produzir algo que seja útil aos usuários.

O programa de contar palavras acima usa todos estes padrões exceto um.

1.10 O que pode dar errado ?

Como vimos em nossa última conversa com o Python, devemos nos comunicar de modo preciso quando escrevemos código Python. O mínimo desvio ou erro fará com que o Python pare de executar o seu programa.

Programadores iniciantes mutas vezes tomam o fato de que o Python não deixa espaço para erros, como prova de que ele é malvado e cruel. Enquanto o Python parece gostar de todo mundo, ele os conhece pessoalmente e guarda um ressentimento contra eles. Devido a este ressentimento, o Python leva avalia nossos programas perfeitamente escritos e os rejeita como "incorretos" apenas para nos atormentar.

```
>>> primt 'Ola mundo!'
  File "<stdin>", line 1
    primt 'Ola mundo!'
                  ^
SyntaxError: invalid syntax
>>> primt 'Ola mundo'
  File "<stdin>", line 1
    primt 'Ola mundo'
                 ^
SyntaxError: invalid syntax
>>> Eu te odeio Python!
  File "<stdin>", line 1
    Eu te odeio Python!
       ^
SyntaxError: invalid syntax
>>> se você vier aqui fora, vou te dar uma lição
  File "<stdin>", line 1
    se você vier aqui fora, vou te dar uma lição
             ^
SyntaxError: invalid syntax
>>>
```

Não se ganha muita coisa discutindo com o Python. Ele é somente uma ferramenta. Ele não tem emoções e também não fica feliz e pronto para te servir quando você precisa dele. Suas mensagens de erro parecem ásperas, mas elas apenas tentam nos ajudar. Ele recebeu o seu comando e simplesmente não conseguiu entender o que você digitou.

Python se parece muito com um cachorro, te ama incondicionalmente, consegue entender apenas algumas poucas palavras, olha para você com um olhar doce na face (>>>), e fica esperando você dizer algo que ele entenda. Quando o Python diz "SyntaxError: invalid syntax", está simplesmente abanando o rabo e dizendo, "Parece que você disse algo que eu não consegui entender, por favor, continue conversando comigo (>>>)."

Conforme seu programa vai se tornando mais sofisticado, você encontrará três tipos genéricos de erro:

Erros de Sintaxe: Estes são os primeiros erros que você cometerá e os mais fáceis de se consertar. Um erro de sintaxe significa que você violou as "regras gramaticais" do Python. Python dá o seu melhor para apontar a linha correta e o caractere que o confundiu. A única parte complicada dos erros de sintaxe é que às vezes os erros que precisam de conserto na verdade ocorrem um pouco antes de onde o Python *indica* e isso confunde um pouco. Desta forma, a linha e caractere que o Python indica no erro de sintaxe pode ser que seja apenas um ponto de início e precisa da sua investigação.

Erros de Lógica: Um erro de lógica é quando o seu programa tem uma boa sintaxe mas há um erro na ordem das instruções ou às vezes um erro em como uma instrução se relaciona com as demais. Um bom exemplo de erro de lógica pode ser, "tome um gole de sua garrafa de água, coloque-a na mochila, caminhe para a biblioteca, e depois coloque a tampa de volta na garrafa."

Erros de Semântica: Um erro de semântica é quando a descricao dos passos estão sintaticamente corretos, na ordem certa, mas há existe um erro no programa. O programa está perfeitamente correto, mas ele não faz o que você *deseja* que ele faça. Um exemplo simples poderia ser quando você instrui uma pessoa a chegar até um restaurante e diz, "quando você cruzar a estação de gás, vire à esquerda e ande por um quilômetro e o restaurante estará no prédio vermelho à sua esquerda." Seu amigo está muito atrasado e liga para você para dizer que está em uma fazenda, passando atrás de um celeiro, sem o sinal da existência de um restaurante. Então você diz "você virou à esquerda ou à direita na estação de gás ?" e ele diz: "Eu segui suas instruções perfeitamente, as escrevi em um papel, e dizia para virar à esquerda e andar por um quilômetro até a estação de gás." Então você diz: "Eu sinto muito, embora minhas instruções estivessem sintaticamente corretas, elas infelizmente tinham um pequeno erro semântico indetectável."

Novamente em todos os três tipos de erros, o Python está se esforçando para fazer tudo aquilo que você pediu.

1.11 A jornada do aprendizado

Enquanto você progride para o restante do livro, não tenha medo se os conceitos não parecem se encaixar tão bem em um primeiro momento. Quando você aprender a falar, verá que os problemas que você enfrentou valeram a pena e não se lembrará mais das dificuldades. Estará tudo certo se você levar seis meses para se mover de um simples vocabulário até simples sentenças e levar mais 5-6 anos para mover-se de sentenças a parágrafos, e uns anos mais para estar habilitado a escrever uma interessante e curta estória com suas próprias mãos.

Nós queremos que você aprenda Python muito mais rápido, então nós ensinamos tudo ao mesmo tempo nos próximos capítulos. Mas aprender uma nova linguagem

leva tempo para ser absorver e entender antes de se tornar natural. Este processo pode gerar alguma confusão conforme nós visitamos e revisitamos os tópicos para tentar dar a você uma visão completa, nós definimos pequenos fragmentos que aos poucos irão formando a visão completa. Este livro é dividido em capítulos sequenciais e à medida que você avança vai aprendendo diversos assuntos, não se sinta preso na sequência do livro, avance capítulos e depois recue se for preciso, o que importa é o seu aprendizado e em como você sente que deve ser. Ao estudar superficialmente materiais mais avançados sem entender completamente os detalhes, você pode obter um melhor entendimento do "porque?" programar. Revisando materiais mais básicos e até mesmo refazendo exercícios anteriores, você irá perceber que aprendeu muito, até mesmo com aqueles materiais que pareciam impenetráveis de tão difíceis.

Normalmente, quando você aprende sua primeira linguagem de programação, ocorrem vários momentos "Ah Hah!". Aqueles em que você está trabalhando arduamente e quando para para prestar atenção e dar um descanso percebe que está construindo algo maravilhoso.

Se algo estiver particularmente difícil, saiba que não vale a pena ficar acordado a noite inteira encarando o problema. Faça uma pausa, tire um cochilo, faça um lanche, compartilhe o seu problema com alguém (com seu cão talvez) e então retorne ao problema com a mente descansada. Eu asseguro a você que uma vez que você aprender os conceitos de programação neste livro, irá olhar para trás e perceber que tudo foi muito fácil, elegante e tão simples que tomou de você apenas um tempo para absorver o aprendizado.

1.12 Glossário

bug: Um erro em um programa.

unidade central de processamento: O coração de qualquer computador. É ela que executa o software que nós escrevemos; também chamada de "CPU" ou de "processador".

compilação: Traduzir um programa escrito em uma linguagem de alto nível em uma linguagem de baixo nível tudo de uma vez, em preparação para uma posterior execução.

linguagem de alto nível: Uma linguagem de programação como o Python que é desenhada para ser fácil para humanos ler e escrever.

modo interativo: Um modo de usar o interpretador Python digitando comandos e expressões no prompt.

interpretação: Executar um programa em uma linguagem de alto nível traduzindo uma linha por vez.

linguagem de baixo nível: Uma linguagem de programação que é desenhada para que seja fácil um computador executar; também chamada "código de máquina" ou "linguagem assembly".

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

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

parse: Examinar um programa e analisar a estrutura sintática.

portabilidade: Uma propriedade de um programa que roda em mais de um tipo de computador.

instrução print: Uma instrução que faz com que o interpretador Python exiba um valor na tela.

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

programa: Um conjunto de instruções que especifica uma computação.

prompt: Quando um programa exibe uma mensagem e aguarda o usuário digitar algo para o programa.

memória secundária: Armazena programas e dados e retém a informação mesmo quando a energia é desligada. Geralmente mais devagar em relação à memória principal. Exemplos de memória secundária são discos rígidos e memória flash nos pendrives USB.

semântica: O significado de um programa.

erro semântico: Um erro em um programa que faz algo diferente daquilo que o programador desejava.

código fonte: Um programa em uma linguagem de alto nível.

1.13 Exercícios

Exercício 1.1 Qual é a função da memória secundária em um computador ?

a) Executar todas as computações e lógica de um programa
b) Obter páginas web da internet
c) Armazenar informação por um longo período – mesmo se faltar energia
d) Receber o input de um usuário

Exercício 1.2 O que é um programa ?

Exercício 1.3 Qual é a diferença entre um compilador e um interpretador ?

Exercício 1.4 Qual destas opções a seguir contém "código de máquina"?

a) O interpretador Python
b) O teclado
c) Arquivo de código fonte Python
d) Um documento do processador de texto

Exercício 1.5 O que está errado no código a seguir:

```
>>> primt 'Ola mundo!'
  File "<stdin>", line 1
    primt 'Ola mundo!'
                      ^
SyntaxError: invalid syntax
>>>
```

Exercício 1.6 Em qual lugar do computador existe uma variável "X" armazenada depois que a seguinte linha de Python finaliza ?

```
x = 123
```

a) Unidade central de processamento
b) Memória Principal
c) Memória Secundária
d) Dispositivos de Entrada
e) Dispositivos de Saída

Exercício 1.7 O que o seguinte programa irá imprimir:

```
x = 43
x = x + 1
print x
```

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

Exercício 1.8 Explique cada item a seguir usando como exemplo uma capacidade humana: (1) Unidade central de processamento, (2) Memória principal, (3) Memória secundária, (4) Dispositivo de entrada, e (5) Dispositivo de saída. Por exemplo, "Qual é a capacidade humana equivalente a Unidade central de processamento"?

Exercício 1.9 Como se conserta um "Erro de Sintaxe"?

Índice Remissivo