

Web Scraping with Python

COLLECTING DATA FROM THE MODERN WEB

Ryan Mitchell

Web Scraping com Python

Coletando dados da ModernWeb

Ryan Mitchell

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Web Scraping com Python

por Ryan Mitchell

Copyright © 2015 Ryan Mitchell. Todos os direitos reservados. Impresso

nos Estados Unidos da América.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Os livros da O'Reilly podem ser adquiridos para uso educacional, comercial ou promocional de vendas. As edições online também estão disponíveis para a maioria dos títulos (<http://safaribooksonline.com>) Para obter mais informações, entre em contato com nosso departamento de vendas corporativo / institucional: 800-998-9938 ou corporate@oreilly.com.

Editores: Simon St. Laurent e Allyson MacDonald

Indexador: Lucie Haskins

Editor de produção: Kalapurakkal brilhante

Designer de interiores: David Futato

Editor de cópia: Jasmine Kwityn

Designer de capa: Karen Montgomery

Revisor: Carla Thornton

Ilustrador: Rebecca Demarest

Junho de 2015: Primeira edição

Histórico de revisão da primeira edição

10/06/2015: primeiro lançamento

Veja <http://oreilly.com/catalog/errata.csp?isbn=9781491910276> para detalhes de lançamento.

O logotipo O'Reilly é uma marca registrada da O'Reilly Media, Inc. *Web Scraping com Python*, a imagem da capa e a imagem comercial relacionada são marcas comerciais da O'Reilly Media, Inc.

Embora o editor e o autor tenham feito esforços de boa fé para garantir que as informações e instruções contidas neste trabalho sejam precisas, o editor e o autor se isentam de qualquer responsabilidade por erros ou omissões, incluindo, sem limitação, a responsabilidade por danos resultantes do uso de ou confiança neste trabalho. O uso das informações e instruções contidas neste trabalho é por sua própria conta e risco. Se qualquer amostra de código ou outra tecnologia que este trabalho contiver ou descrever estiver sujeita a licenças de código aberto ou direitos de propriedade intelectual de terceiros, é sua responsabilidade garantir que seu uso esteja em conformidade com tais licenças e / ou direitos.

978-1-491-91027-6

[LSI]

Índice

Prefácio.....	vii
---------------	-----

Parte I. Construindo Raspadores

1. Seu primeiro raspador de web.....	3
Conectando	3
Uma introdução ao BeautifulSoup	6
Instalando BeautifulSoup	6
Executando BeautifulSoup	8
Conexão confiável	9
2. Análise avançada de HTML.....	13
Você nem sempre precisa de um martelo	13
Outra porção de BeautifulSoup	14
find () e findAll () com BeautifulSoup	16
Outros objetos BeautifulSoup	18
Árvores de navegação	18
Expressões regulares	22
Expressões regulares e BeautifulSoup	27
Acessando Atributos	28
Expressões Lambda	28
Além do BeautifulSoup	29
3. Começando a engatinhar.....	31
Atravessando um único domínio	31
Rastreando um site inteiro	35
Coletando dados em um site inteiro	38
Rastreando pela Internet	40
Rastejando com Scrapy	45
4. Usando APIs.....	49
Como funcionam as APIs	50

Convenções Comuns	50
Métodos	51
Autenticação	52
Respostas	52
Chamadas API	53
Eco Nest	54
Alguns exemplos	54
Twitter	55
Começando	56
Alguns exemplos	57
APIs do Google	60
Começando	60
Alguns exemplos	61
Analisando JSON	63
Trazendo tudo de volta para casa	64
Mais sobre APIs	68
5. Armazenamento de dados.....	71
Arquivos de mídia	71
Armazenamento de dados em CSV	74
MySQL	76
Instalando MySQL	77
Alguns comandos básicos	79
Integrando com Python	82
Técnicas de banco de dados e boas práticas	85
“Six Degrees” no MySQL	87
O email	90
6. Leitura de documentos.....	93
Codificação de Documento	93
Texto	94
Codificação de texto e Internet global	94
CSV	98
Lendo arquivos CSV	98
PDF	100
Microsoft Word e .docx	102
<hr/>	
Parte II. Raspagem Avançada	
7. Limpando Seus Dados Sujos.....	109
Limpeza em código	109

Normalização de Dados	112
Limpeza após o fato	113
OpenRefine	114
8. Leitura e escrita de linguagens naturais.....	119
Resumindo Dados	120
Modelos Markov	123
Seis graus da Wikipedia: Conclusão	126
Kit de ferramentas de linguagem natural	129
Instalação e configuração	129
Análise Estatística com NLTK	130
Análise Lexicográfica com NLTK	132
Recursos adicionais	136
9. Rastreamento por meio de formulários e logins.....	137
Biblioteca de solicitações Python	137
Envio de um formulário básico	138
Botões de rádio, caixas de seleção e outras entradas	140
Envio de arquivos e imagens	141
Tratamento de logins e cookies	142
Autenticação de acesso básico HTTP	144
Outros problemas de formulário	144
10. Raspando JavaScript.....	147
Uma breve introdução ao JavaScript	148
Bibliotecas comuns de JavaScript	149
Ajax e HTML dinâmico	151
Executando JavaScript em Python com Selenium	152
Tratamento de redirecionamentos	158
11. Processamento de imagens e reconhecimento de texto.....	161
Visão geral das bibliotecas	162
Travesseiro	162
Tesseract	163
NumPy	164
Processando Texto Bem Formatado	164
Raspagem de texto de imagens em sites	166
Leitura de CAPTCHAs e Tesserato de Treinamento	169
Tesserato de treinamento	171
Recuperando CAPTCHAs e enviando soluções	174

12. Evitando armadilhas de raspagem.....	177
Uma nota sobre ética	177
Parcendo um Humano	178
Ajuste seus cabeçalhos	179
Manuseando Cookies	181
Tempo é tudo	182
Recursos comuns de segurança de formulários	183
Valores de campo de entrada ocultos	183
Evitando Honeypots	184
A Lista de Verificação Humana	186
13. Testando Seu Site com Scrapers.....	189
Uma introdução aos testes	189
O que são testes de unidade?	190
Python unittest	190
Testando a Wikipedia	191
Testando com Selênia	193
Interagindo com o site	194
Unittest ou selênia?	197
14. Raspar remotamente.....	199
Por que usar servidores remotos?	199
Evitando bloqueio de endereço IP	199
Portabilidade e extensibilidade	200
Tor	201
PySocks	202
Hospedagem Remota	203
Executando a partir de uma conta de hospedagem de site	203
Correndo da nuvem	204
Recursos adicionais	206
Seguindo em Frente	206
A. Python em um relance.....	209
B. Visão geral da Internet.....	213
C. As legalidades e a ética do Web Scraping.....	217
	Index.....
	231

Prefácio

Para aqueles que não desenvolveram essa habilidade, a programação de computadores pode parecer uma espécie de mágica. Se a programação é mágica, então *Raspagem da web* é magia; isto é, a aplicação de magia para particularmente impressionante e útil - mas surpreendentemente fácil

- façanhas.

Na verdade, em meus anos como engenheiro de software, descobri que muito poucas práticas de programação capturam a empolgação tanto de programadores quanto de leigos como a web scraping. A capacidade de escrever um bot simples que coleta dados e os transmite em um terminal ou os armazena em um banco de dados, embora não seja difícil, nunca deixa de fornecer uma certa emoção e senso de possibilidade, não importa quantas vezes você tenha feito isso antes .

É uma pena que, quando falo com outros programadores sobre web scraping, haja muitos mal-entendidos e confusão sobre a prática. Algumas pessoas não têm certeza se isso é legal (é) ou como lidar com a Web moderna, com todo seu JavaScript, multimídia e cookies. Alguns ficam confusos sobre a distinção entre APIs e web scrapers.

Este livro busca acabar com muitas dessas perguntas comuns e equívocos sobre web scraping, ao mesmo tempo que fornece um guia abrangente para as tarefas mais comuns de web scraping.

Começando em [Capítulo 1](#) , irei fornecer amostras de código periodicamente para demonstrar conceitos. Esses exemplos de código são de domínio público e podem ser usados com ou sem atribuição (embora o reconhecimento seja sempre apreciado). Todos os exemplos de código também estarão disponíveis em [o site](#) para visualização e download.

O que é Web Scraping?

A coleta automatizada de dados da Internet é quase tão antiga quanto a própria Internet. Apesar *Raspagem da web* não é um termo novo, nos últimos anos a prática era mais comumente conhecida como *screen scraping*, *data mining*, *web harvesting*, ou variações semelhantes. O consenso geral hoje parece favorecer *Raspagem da web*, então esse é o termo que usarei ao longo do livro, embora ocasionalmente me refira aos próprios programas de web scraping como *bots*.

Em teoria, web scraping é a prática de reunir dados por qualquer meio que não seja um programa interagindo com uma API (ou, obviamente, por meio de um ser humano usando um navegador da web). Isso é mais comumente realizado escrevendo um programa automatizado que consulta um servidor web, solicita dados (geralmente na forma de HTML e outros arquivos que compõem páginas web) e, em seguida, analisa esses dados para extrair as informações necessárias.

Na prática, web scraping engloba uma ampla variedade de técnicas e tecnologias de programação, como análise de dados e segurança da informação. Este livro cobrirá os conceitos básicos de web scraping e crawling ([Parte I](#)) e se aprofunde em alguns dos tópicos avançados em [parte II](#).

Por que Web Scraping?

Se a única maneira de acessar a Internet é por meio de um navegador, você está perdendo uma grande variedade de possibilidades. Embora os navegadores sejam úteis para executar JavaScript, exibir imagens e organizar objetos em um formato mais legível (entre outras coisas), os web scrapers são excelentes para reunir e processar grandes quantidades de dados (entre outras coisas). Em vez de visualizar uma página de cada vez através da janela estreita de um monitor, você pode visualizar bancos de dados abrangendo milhares ou até milhões de páginas de uma vez.

Além disso, os web scrapers podem ir a lugares que os mecanismos de pesquisa tradicionais não conseguem. Uma pesquisa no Google por “voos mais baratos para Boston” resultará em uma série de anúncios e sites populares de pesquisa de voos. O Google sabe apenas o que esses sites dizem em suas páginas de conteúdo, não os resultados exatos de várias consultas inseridas em um aplicativo de pesquisa de voos. No entanto, um web scraper bem desenvolvido pode traçar o custo de um voo para Boston ao longo do tempo, em uma variedade de sites, e dizer a você o melhor momento para comprar sua passagem.

Você pode estar perguntando: “A coleta de dados não é para que servem as APIs?” (Se você não estiver familiarizado com APIs, consulte [Capítulo 4](#).) Bem, as APIs podem ser fantásticas, se você encontrar uma que atenda aos seus propósitos. Eles podem fornecer um fluxo conveniente de dados bem formatados de um servidor para outro. Você pode encontrar uma API para muitos tipos diferentes de dados que você pode

deseja usar, como postagens do Twitter ou páginas da Wikipedia. Em geral, é preferível usar uma API (se houver), em vez de construir um bot para obter os mesmos dados. No entanto, existem vários motivos pelos quais uma API pode não existir:

- Você está coletando dados em uma coleção de sites que não possuem uma API coesa.
- Os dados que você deseja são um conjunto bastante pequeno e finito que o webmaster não achou que justificasse uma API.
- A fonte não possui infraestrutura ou capacidade técnica para criar uma API.

Mesmo quando uma API *faz* existem, os limites de volume e taxa de solicitação, os tipos de dados ou o formato de dados que ele fornece podem ser insuficientes para seus propósitos.

É aqui que entra o web scraping. Com poucas exceções, se você puder visualizá-lo em seu navegador, poderá acessá-lo por meio de um script Python. Se você puder acessá-lo em um script, poderá armazená-lo em um banco de dados. E se você pode armazená-lo em um banco de dados, você pode fazer praticamente qualquer coisa com esses dados.

Obviamente, há muitas aplicações extremamente práticas para ter acesso a dados quase ilimitados: previsão de mercado, tradução de linguagem de máquina e até diagnósticos médicos se beneficiaram enormemente com a capacidade de recuperar e analisar dados de sites de notícias, textos traduzidos e fóruns de saúde, respectivamente.

Mesmo no mundo da arte, web scraping abriu novas fronteiras para a criação. O projeto de 2006 "[Nós nos sentimos bem](#)" por Jonathan Harris e Sep Kamvar, vasculhou vários sites de blog em inglês em busca de frases que começassem com "I feel" ou "I am feeling". Isso levou a uma visualização de dados popular, descrevendo como o mundo estava se sentindo dia a dia e minuto a minuto.

Independentemente do seu campo, quase sempre há uma maneira de o web scraping guiar as práticas de negócios com mais eficácia, melhorar a produtividade ou até mesmo ramificar-se para um campo totalmente novo.

Sobre este livro

Este livro foi desenvolvido para servir não apenas como uma introdução à web scraping, mas como um guia abrangente para a extração de quase todos os tipos de dados da Web moderna. Embora use a linguagem de programação Python e cubra muitos fundamentos do Python, não deve ser usado como uma introdução à linguagem.

Se você não é um programador especialista e não conhece nenhum Python, este livro pode ser um desafio. Se, no entanto, você for um programador experiente, deverá achar o material fácil de aprender. [Apêndice A](#) cobre a instalação e o trabalho com o Python 3.x, que é usado ao longo deste livro. Se você só usou Python

2.x, ou não tem 3.x instalado, você pode querer revisar [Apêndice A](#).

Se você está procurando um recurso Python mais abrangente, o livro *Apresentando Python* de Bill Lubanovic é um guia muito bom, embora longo. Para aqueles com períodos de atenção mais curtos, a série de vídeos *Introdução ao Python* por Jessica McKellar é um excelente recurso.

apêndice C inclui estudos de caso, bem como uma análise dos principais problemas que podem afetar a forma como você pode operar raspadores legalmente nos Estados Unidos e usar os dados que eles produzem.

Livros técnicos muitas vezes são capazes de se concentrar em uma única linguagem ou tecnologia, mas web scraping é um assunto relativamente díspar, com práticas que exigem o uso de bancos de dados, servidores web, HTTP, HTML, segurança da Internet, processamento de imagens, ciência de dados, e outras ferramentas. Este livro tenta cobrir tudo isso com o propósito de reunir dados de fontes remotas na Internet.

Parte I cobre o assunto de web scraping e web crawling em profundidade, com um forte foco em um pequeno punhado de bibliotecas usadas ao longo do livro. **Parte I** pode ser facilmente usado como uma referência abrangente para essas bibliotecas e técnicas (com algumas exceções, onde referências adicionais serão fornecidas).

parte II cobre assuntos adicionais que o leitor pode achar úteis ao escrever web scrapers. Infelizmente, esses assuntos são muito amplos para serem agrupados em um único capítulo. Por causa disso, referências frequentes serão feitas a outros recursos para obter informações adicionais.

A estrutura deste livro é organizada para ser fácil pular entre os capítulos para encontrar apenas a técnica de raspagem da web ou as informações que você está procurando. Quando um conceito ou parte do código se baseia em outro mencionado no capítulo anterior, referirei explicitamente a seção em que foi abordado.

Convenções utilizadas neste livro

As seguintes convenções tipográficas são usadas neste livro:

itálico

Indica novos termos, URLs, endereços de e-mail, nomes de arquivos e extensões de arquivos.

Largura constante

Usado para listagens de programas, bem como dentro de parágrafos para se referir a elementos do programa, como nomes de variáveis ou funções, bancos de dados, tipos de dados, variáveis de ambiente, instruções e palavras-chave.

Largura constante em negrito

Mostra comandos ou outro texto que deve ser digitado pelo usuário.

Itálico de largura constante

Mostra o texto que deve ser substituído por valores fornecidos pelo usuário ou por valores determinados pelo contexto.



Este elemento significa uma dica ou sugestão.



Este elemento significa uma nota geral.



Este elemento indica um aviso ou cuidado.

Usando exemplos de código

O material complementar (exemplos de código, exercícios, etc.) está disponível para download em
<http://pythonscraping.com/code/>.

Este livro está aqui para ajudá-lo a realizar seu trabalho. Em geral, se o código de exemplo é oferecido com este livro, você pode usá-lo em seus programas e documentação. Você não precisa entrar em contato conosco para obter permissão, a menos que esteja reproduzindo uma parte significativa do código. Por exemplo, escrever um programa que usa vários pedaços de código deste livro não requer permissão. Vender ou distribuir um CD-ROM com exemplos de livros da O'Reilly requer permissão. Responder a uma pergunta citando este livro e citando um código de exemplo não requer permissão. Incorporar uma quantidade significativa de código de exemplo deste livro na documentação do seu produto requer permissão.

Agradecemos, mas não exigimos atribuição. Uma atribuição geralmente inclui o título, autor, editora e ISBN. Por exemplo: " *Web Scraping com Python* por Ryan Mitchell (O'Reilly). Copyright 2015 Ryan Mitchell, 978-1-491-91029-0. "

Se você acha que o uso de exemplos de código está fora do uso justo ou da permissão dada aqui, sinta-se à vontade para nos contatar em permission@oreilly.com .

Safari® Books Online



[®] **Safari Books Online** é uma biblioteca digital on-demand que oferece especialistas **conteúdo** na forma de livro e vídeo dos principais autores do mundo em tecnologia e negócios.

Profissionais de tecnologia, desenvolvedores de software, web designers e profissionais criativos e de negócios usam o Safari Books Online como seu principal recurso para pesquisa, solução de problemas, aprendizado e treinamento de certificação.

Safari Books Online oferece uma variedade de **misturas de produtos** e programas de preços para **organizações**, **agências governamentais**, e **indivíduos**. Os assinantes têm acesso a milhares de livros, vídeos de treinamento e manuscritos de pré-publicação em um banco de dados totalmente pesquisável de editoras como O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology e dezenas **Mais**. Para obter mais informações sobre o Safari Books Online, visite-nos **conectados**.

Como entrar em contato conosco

Envie comentários e perguntas sobre este livro ao editor:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (nos Estados Unidos ou Canadá)
707-829-0515 (internacional ou local)
707-829-0104 (fax)

Temos uma página web para este livro, onde listamos erratas, exemplos e qualquer informação adicional. Você pode acessar esta página em <http://oreil.ly/1ePG2Uj>.

Para comentar ou fazer perguntas técnicas sobre este livro, envie e-mail para bookquestions@oreilly.com.

Para obter mais informações sobre nossos livros, cursos, conferências e notícias, consulte nosso site em <http://www.oreilly.com>

Encontre-nos no Facebook: <http://facebook.com/oreilly>

Siga-nos no Twitter: <http://twitter.com/oreillymedia>

Veja-nos no YouTube: <http://www.youtube.com/oreillymedia>

Agradecimentos

Assim como alguns dos melhores produtos surgem de um mar de feedback do usuário, este livro nunca poderia ter existido em qualquer forma útil sem a ajuda de muitos colaboradores, líderes de torcida e editores. Obrigado à equipe da O'Reilly e seu apoio incrível por este assunto um tanto pouco convencional, aos meus amigos e familiares que ofereceram conselhos e toleraram leituras improvisadas, e aos meus colegas de trabalho da LinkeDrive, a quem provavelmente devo muitas horas de trabalho .

Obrigado, em particular, a Allyson MacDonald, Brian Anderson, Miguel Grinberg e Eric VanWyk por seus comentários, orientação e ocasional amor duro. Algumas seções e exemplos de código foram escritos como resultado direto de suas sugestões inspiradoras.

Obrigado a Yale Specht por sua paciência ilimitada ao longo dos últimos nove meses, fornecendo o incentivo inicial para prosseguir com este projeto e feedback estilístico durante o processo de escrita. Sem ele, este livro teria sido escrito na metade do tempo, mas não seria tão útil.

Finalmente, obrigado a Jim Waldo, que realmente começou tudo isso há muitos anos quando enviou uma caixa de Linux e *A Arte e Ciência de C* para um adolescente jovem e impressionável.

PARTE I

Raspadores de edifícios

Esta seção se concentra na mecânica básica de web scraping: como usar Python para solicitar informações de um servidor web, como realizar o tratamento básico da resposta do servidor e como começar a interagir com um site de maneira automatizada. No final, você estará navegando pela Internet com facilidade, construindo scrapers que podem saltar de um domínio para outro, coletar informações e armazenar essas informações para uso posterior.

Para ser honesto, o web scraping é um campo fantástico para entrar se você deseja um grande pagamento por um investimento inicial relativamente pequeno. Com toda a probabilidade, 90% dos projetos de web scraping que você encontrará se basearão em técnicas usadas apenas nos próximos seis capítulos. Esta seção cobre o que o público geral (embora tecnicamente experiente) tende a pensar quando pensa em "web scrapers":

- Recuperação de dados HTML de um nome de domínio
- Analisar esses dados para obter informações de destino
- Armazenar as informações do alvo
- Opcionalmente, ir para outra página para repetir o processo

Isso lhe dará uma base sólida antes de prosseguir para projetos mais complexos na parte II. Não se iluda pensando que esta primeira seção não é tão importante quanto alguns dos projetos mais avançados na segunda metade. Você usará quase todas as informações da primeira metade deste livro diariamente enquanto escreve web scrapers.

CAPÍTULO 1

Seu primeiro raspador de web

Depois de começar a fazer web scraping, você começa a apreciar todas as pequenas coisas que os navegadores fazem por nós. A Web, sem uma camada de formatação HTML, estilo CSS, execução de JavaScript e renderização de imagens, pode parecer um pouco intimidante no início, mas neste capítulo, assim como no próximo, vamos cobrir como formatar e interpretar dados sem a ajuda de um navegador.

Este capítulo começará com o básico do envio de uma solicitação GET a um servidor web para uma página específica, lendo a saída HTML dessa página e fazendo algumas extrações de dados simples para isolar o conteúdo que estamos procurando.

Conectando

Se você não passou muito tempo trabalhando em rede ou segurança de rede, a mecânica da Internet pode parecer um pouco misteriosa. Não queremos pensar sobre o que, exatamente, a rede está fazendo toda vez que abrimos um navegador e vamos para <http://google.com>, e, atualmente, não precisamos. Na verdade, eu diria que é fantástico que as interfaces de computador tenham avançado a ponto de a maioria das pessoas que usam a Internet não ter a menor ideia de como ela funciona.

No entanto, o web scraping requer a remoção de parte dessa cobertura de interface, não apenas no nível do navegador (como ele interpreta todo esse HTML, CSS e JavaScript), mas ocasionalmente no nível da conexão de rede.

Para dar uma ideia da infraestrutura necessária para levar as informações ao seu navegador, vamos usar o exemplo a seguir. Alice possui um servidor web. Bob usa um computador desktop, que está tentando se conectar ao servidor de Alice. Quando uma máquina deseja falar com outra máquina, ocorre algo como a seguinte troca:

1. O computador de Bob envia ao longo de um fluxo de bits 1 e 0, indicado por tensões altas e baixas em um fio. Esses bits formam algumas informações, contendo um cabeçalho e um corpo. O cabeçalho contém um destino imediato do endereço MAC de seu roteador local, com um destino final do endereço IP de Alice. O corpo contém sua solicitação para o aplicativo de servidor de Alice.
2. O roteador local de Bob recebe todos esses 1s e 0s e os interpreta como um pacote, do próprio endereço MAC de Bob e destinado ao endereço IP de Alice. Seu roteador marca seu próprio endereço IP no pacote como o endereço IP "de" e o envia pela Internet.
3. O pacote de Bob atravessa vários servidores intermediários, que direcionam seu pacote para o caminho físico / com fio correto, para o servidor de Alice.
4. O servidor de Alice recebe o pacote, em seu endereço IP.
5. O servidor de Alice lê o destino da porta do pacote (quase sempre a porta 80 para aplicativos da web, isso pode ser considerado algo como um "número do apartamento" para dados do pacote, onde o endereço IP é o "endereço da rua"), no cabeçalho e passa-o para o aplicativo apropriado - o aplicativo do servidor da web.
6. O aplicativo do servidor da Web recebe um fluxo de dados do processador do servidor. Esses dados dizem algo como:
 - Este é um pedido GET
 - O seguinte arquivo é solicitado: index.html
7. O servidor da web localiza o arquivo HTML correto, empacota-o em um novo pacote para enviar a Bob e o envia por meio de seu roteador local, para transporte de volta à máquina de Bob, pelo mesmo processo.

E *voilà!* Temos a Internet.

Então, onde nessa troca o navegador da web entrou em jogo? Absolutamente em lugar nenhum. Na verdade, os navegadores são uma invenção relativamente recente na história da Internet, quando o Nexus foi lançado em 1990.

Sim, o navegador da web é um aplicativo muito útil para criar esses pacotes de informações, enviá-los e interpretar os dados que você recebe como belas imagens, sons, vídeos e texto. No entanto, um navegador da web é apenas código, e o código pode ser desmontado, dividido em seus componentes básicos, reescrito, reutilizado e feito para fazer o que quisermos. Um navegador da web pode dizer ao processador para enviar alguns dados ao aplicativo que lida com sua interface sem fio (ou com fio), mas muitas linguagens têm bibliotecas que podem fazer isso da mesma forma.

Vamos dar uma olhada em como isso é feito em Python:

```
de urllib.request importar urlopen html = urlopen ( "http://pythonscraping.com/pages/page1.html"
)
impressão ( html . ler ())
```

Você pode salvar este código como *scrapetest.py* e execute-o em seu terminal usando o comando:

```
$ python scrapetest.py
```

Observe que se você também tiver o Python 2.x instalado em sua máquina, pode ser necessário chamar explicitamente o Python 3.x executando o comando desta forma:

```
$ python3 scrapetest.py
```

Isso produzirá o código HTML completo para a página em <http://pythonscraping.com/pages/page1.html>. Mais precisamente, isso gera o arquivo HTML *page1.html*, encontrado no diretório < raiz da web> / páginas, no servidor localizado no nome de domínio <http://pythonscraping.com>.

Qual é a diferença? A maioria das páginas da web modernas tem muitos arquivos de recursos associados a elas. Podem ser arquivos de imagem, arquivos JavaScript, arquivos CSS ou qualquer outro conteúdo ao qual a página que você está solicitando esteja vinculada. Quando um navegador da web atinge uma tag como

, o navegador sabe que precisa fazer outra solicitação ao servidor para obter os dados do arquivo *cuteKitten.jpg* para renderizar totalmente a página para o usuário. Lembre-se de que nosso script Python não tem a lógica de voltar e solicitar vários arquivos (ainda); ele só pode ler o único arquivo HTML que solicitamos.

Então, como isso acontece? Graças à natureza simples do Python, a linha

de urllib.request importar urlopen

significa o que parece significa: olha para a solicitação do módulo Python (encontrada na biblioteca `urllib`) e importa apenas a função `urlopen`.



urllib ou urllib2?

Se você usou a biblioteca `urllib2` no Python 2.x, deve ter notado que as coisas mudaram um pouco entre `urllib2` e `urllib`. No Python 3.x, `urllib2` foi renomeado como `urllib` e foi dividido em vários submódulos: `urllib.request`, `urllib.parse`, e `urllib.error`. Embora os nomes das funções permaneçam basicamente os mesmos, você pode querer observar quais funções foram movidas para submódulos ao usar o novo `urllib`.

`urllib` é uma biblioteca Python padrão (o que significa que você não precisa instalar nada extra para executar este exemplo) e contém funções para solicitar dados pela web, manipular cookies e até mesmo alterar metadados, como cabeçalhos e seu agente de usuário. Estaremos usando `urllib` extensivamente ao longo do livro, portanto, recomendamos que você leia a documentação do Python para a biblioteca (<https://docs.python.org/3/library/urllib.html>).

`urlopen` é usado para abrir um objeto remoto em uma rede e lê-lo. Por ser uma biblioteca bastante genérica (ela pode ler arquivos HTML, arquivos de imagem ou qualquer outro fluxo de arquivo com facilidade), iremos usá-la com bastante frequência ao longo do livro.

Uma introdução ao BeautifulSoup

“Linda Sopa, tão rica e verde, Esperando em
uma terrina quente!
Quem, por tais guloseimas, não se rebaixaria?
Sopa da noite, linda Sopa! ”

A biblioteca BeautifulSoup recebeu o nome de um poema de Lewis Carroll com o mesmo nome em *Alice no País das Maravilhas*. Na história, este poema é cantado por um personagem chamado Tartaruga Falsa (em si um trocadilho com o popular prato vitoriano Sopa Falsa de Tartaruga, feito não de tartaruga, mas de vaca).

Como seu homônimo do País das Maravilhas, BeautifulSoup tenta entender o que é absurdo; ajuda a formatar e organizar a bagunçada web corrigindo HTML incorreto e apresentando objetos Python facilmente percorríveis que representam estruturas XML.

Instalando BeautifulSoup

Como a biblioteca BeautifulSoup não é uma biblioteca Python padrão, ela deve ser instalada. Estaremos usando a biblioteca BeautifulSoup 4 (também conhecida como BS4) ao longo deste livro. As instruções completas para instalar o BeautifulSoup 4 podem ser encontradas em

[Crummy.com](#) ; no entanto, o método básico para Linux é:

```
$ sudo apt-get install python-bs4
```

e para Macs:

```
$ sudo pip easy_install
```

Isso instala o gerenciador de pacotes Python *pip*. Em seguida, execute o seguinte:

```
$ pip instalar beautifulsoup4
```

para instalar a biblioteca.

Novamente, observe que se você tiver Python 2.x e 3.x instalados em sua máquina, pode ser necessário chamar *python3* explicitamente:

```
$ python3 myScript.py
```

Certifique-se de usar isso também ao instalar pacotes, ou os pacotes podem ser instalados em Python 2.x, mas não em Python 3.x:

```
$ sudo python3 setup.py install
```

Se estiver usando *pip*, você também pode chamar *pip3* para instalar as versões Python 3.x dos pacotes:

```
$ pip3 instalar beautifulsoup4
```

A instalação de pacotes no Windows é quase idêntica ao processo para Mac e Linux. Baixe a versão mais recente do BeautifulSoup 4 no URL de download acima, navegue até o diretório em que você descompactou e execute:

```
> configuração python . py install
```

E é isso! BeautifulSoup agora será reconhecido como uma biblioteca Python em sua máquina. Você pode testar isso abrindo um terminal Python e importando-o:

```
$ Pitão  
> de bs4 importar BeautifulSoup
```

A importação deve ser concluída sem erros.

Além disso, existe um .exe [instalador para pip onWindows](#), para que você possa instalar e gerenciar facilmente os pacotes:

```
> pip instalar beautifulsoup4
```

Mantendo as bibliotecas em ordem com ambientes virtuais

Se você pretende trabalhar em vários projetos Python ou precisa de uma maneira de agrupar facilmente os projetos com todas as bibliotecas associadas, ou está preocupado com possíveis conflitos entre as bibliotecas instaladas, você pode instalar um ambiente virtual Python para manter tudo separado e fácil de gerenciar.

Ao instalar uma biblioteca Python sem um ambiente virtual, você a está instalando *globalmente*. Isso geralmente requer que você seja um administrador ou execute como root e que a biblioteca Python exista para cada usuário e cada projeto na máquina. Felizmente, criar um ambiente virtual é fácil:

```
$ virtualenv scrapingEnv
```

Isso cria um novo ambiente chamado *scrapingEnv*, que você deve ativar para usar:

```
$ CD scrapingEnv /  
$ fonte bin / ativar
```

Depois de ativar o ambiente, você verá o nome desse ambiente no prompt da linha de comando, lembrando-o de que está trabalhando com ele. Todas as bibliotecas que você instalar ou os scripts que executar estarão nesse ambiente virtual apenas.

Trabalhando no ambiente *scrapingEnv* recém-criado, posso instalar e usar o BeautifulSoup, por exemplo:

```
(scrapingEnv) ryan $ pip install beautifulsoup4 (scrapingEnv) ryan $  
python  
> from bs4 import BeautifulSoup  
>
```

Posso sair do ambiente com o comando deactivate, após o qual não posso mais acessar as bibliotecas que foram instaladas dentro do ambiente virtual:

```
(scrapingEnv) ryan $ deactivate  
ryan $ python  
> do bs4 import BeautifulSoup Traceback (última  
chamada mais recente):  
  
    Arquivo "<stdin>", linha 1, em <module> ImportError:  
    Nenhum módulo chamado 'bs4'
```

Manter todas as suas bibliotecas separadas por projeto também torna fácil compactar toda a pasta de ambiente e enviá-la para outra pessoa. Contanto que eles tenham a mesma versão do Python instalada em suas máquinas, seu código funcionará no ambiente virtual sem exigir que eles próprios instalem nenhuma biblioteca.

Embora não iremos instruí-lo explicitamente a usar um ambiente virtual em todos os exemplos deste livro, lembre-se de que você pode aplicar o ambiente virtual a qualquer momento, simplesmente ativando-o de antemão.

Executando BeautifulSoup

O objeto mais comumente usado na biblioteca BeautifulSoup é, apropriadamente, o objeto BeautifulSoup. Vamos dar uma olhada em ação, modificando o exemplo encontrado no início deste capítulo:

```
de urllib.request importar urlopen  
de bs4 importar BeautifulSoup html = urlopen ( "http://www.pythonscraping.com/pages/page1.html"  
)  
bsObj = BeautifulSoup ( html . ler ())  
impressão ( bsObj . h1 )
```

O resultado é:

```
<h1> Um título interessante </ h1 >
```

Como no exemplo anterior, estamos importando a biblioteca urlopen e chamando html.read () para obter o conteúdo HTML da página. Esse conteúdo HTML é então transformado em um objeto BeautifulSoup, com a seguinte estrutura:

- **html** → < *html*> <*head*> ... <*head*> <*body*> ... <*body*> </*html*>
 - **cabeça** → < *head*> <*title*> *Uma página útil* <*title*> </*head*>
 - **título** → < *title*> *Uma página útil* </*title*>
 - **corpo** → < *body*> <*h1*> *Um Int* ... <*h1*> <*div*> *Lorem ip* ... <*div*> </*body*>
 - **h1** → < *h1*> *Um título interessante* </*h1*>
 - **div** → < *div*> *Lorem Ipsum dolor* ... <*div*>

Observe que o < h1> a tag que extraímos da página foi aninhada duas camadas profundamente em nossa estrutura de objeto BeautifulSoup (html → corpo → h1). No entanto, quando realmente o buscamos do objeto, chamamos a tag h1 diretamente:

```
bsObj . h1
```

Na verdade, qualquer uma das seguintes chamadas de função produziria a mesma saída:

```
bsObj . html . corpo . h1  
bsObj . corpo . h1  
bsObj . html . h1
```

Esperamos que este pequeno gostinho de BeautifulSoup deu a você uma ideia do poder e da simplicidade desta biblioteca.

Praticamente qualquer informação pode ser extraída de qualquer arquivo HTML (ou XML), desde que tenha alguma marca de identificação ao seu redor ou próximo a ele. Dentro

Capítulo 3, vamos nos aprofundar em algumas chamadas de função BeautifulSoup mais complexas, bem como dar uma olhada nas expressões regulares e como elas podem ser usadas com BeautifulSoup para extrair informações de sites.

Conexão confiável

A web é uma bagunça. Os dados estão mal formatados, os sites caem e as tags de fechamento desaparecem. Uma das experiências mais frustrantes em web scraping é dormir com um scraper em execução, sonhando com todos os dados que você terá em seu banco de dados no dia seguinte - apenas para descobrir que o scraper encontrou um erro em algum formato de dados inesperado e interrompeu a execução logo depois que você parou de olhar para a tela. Em situações como essas, você pode ficar tentado a amaldiçoar o nome do desenvolvedor que criou o site (e os dados estranhamente formatados), mas a pessoa que você realmente deveria chutar é você mesmo, por não ter antecipado a exceção em primeiro lugar!

Vamos dar uma olhada na primeira linha de nosso raspador, após as instruções de importação, e descobrir como lidar com quaisquer exceções que isso possa lançar:

```
html = urlopen ( "http://www.pythonscraping.com/pages/page1.html" )
```

Existem duas coisas principais que podem dar errado nesta linha:

- A página não foi encontrada no servidor (ou ocorreu algum erro ao recuperá-la)
- O servidor não foi encontrado

Na primeira situação, um erro HTTP será retornado. Esse erro de HTTP pode ser “404 Página não encontrada”, “500 Erro interno do servidor” etc. Em todos esses casos, o urlopen

A função lançará a exceção genérica “HTTPError”. Podemos lidar com essa exceção da seguinte maneira:

```

experimentar :
    html = urlopen ( "http://www.pythonscraping.com/pages/page1.html" )
exceto Erro HTTP Como e :
    impressão ( e )
        # retornar nulo, interromper ou fazer algum outro "Plano B"
outro :
    # programa continua. Nota: Se você retornar ou quebrar no
    # exceção captura, você não precisa usar a instrução "else"

```

Se um código de erro HTTP for retornado, o programa agora imprime o erro e não executa o resto do programa no outro declaração.

Se o servidor não for encontrado (se, digamos, <http://www.pythonscraping.com> estava inativo ou o URL foi digitado incorretamente), urlopen retorna um Nenhum objeto. Este objeto é análogo a nulo em outras linguagens de programação. Podemos adicionar uma verificação para ver se o html retornado é Nenhum:

```

E se html é Nenhum :
    impressão ( " URL não encontrado " )
outro :
    # programa continua

```

Obviamente, se a página for recuperada com sucesso do servidor, ainda haverá o problema de o conteúdo da página não ser exatamente o que esperávamos. Cada vez que você acessa uma tag em um objeto BeautifulSoup, é inteligente adicionar uma verificação para ter certeza de que a tag realmente existe. Se você tentar acessar uma tag que não existe, BeautifulSoup retornará um Nenhum objeto. O problema é tentar acessar uma tag em um Nenhum o próprio objeto resultará em um AttributeError sendo jogado.

A seguinte linha (onde nonExistentTag é uma tag inventada, não o nome de uma função real do BeautifulSoup):

```
impressão ( bsObj . nonExistentTag )
```

retorna um Nenhum objeto. Este objeto é perfeitamente razoável de manusear e verificar. O problema surge se você não verificar, mas em vez disso, vá em frente e tente chamar alguma outra função no Nenhum objeto, conforme ilustrado a seguir:

```
impressão ( bsObj . nonExistentTag . algumaTag )
```

que retorna a exceção:

```
AttributeError : 'NoneType' objeto não tem atributo 'someTag'
```

Então, como podemos nos proteger contra essas duas situações? A maneira mais fácil é verificar explicitamente as duas situações:

```

experimentar :
    badContent = bsObj . nonExistingTag . outraTag
exceto AttributeError Como e :
    impressão ( " Tag não encontrada " )
outro :

```

```

E se badContent == Nenhum :
    impressão ( " Tag não encontrada " )
outro :
    impressão ( badContent )

```

Essa verificação e tratamento de cada erro parece trabalhoso no início, mas é fácil adicionar um pouco de reorganização a esse código para torná-lo menos difícil de escrever (e, mais importante, muito menos difícil de ler). Este código, por exemplo, é nosso mesmo raspador escrito de uma maneira ligeiramente diferente:

```

de urllib.request importar urlopen
de urllib.error importar Erro HTTP
de bs4 importar BeautifulSoup
def getTitle ( url ):
    experimentar :
        html = urlopen ( url )
    exceto Erro HTTP Como e :
        Retorna Nenhum
    experimentar :
        bsObj = BeautifulSoup ( html . ler ())
        título = bsObj . corpo . h1
    exceto AttributeError Como e :
        Retorna Nenhum
    Retorna título
    título = getTitle ( "http://www.pythonscraping.com/pages/page1.html" )
E se título == Nenhum :
    impressão ( " Título não encontrado " )
outro :
    impressão ( título )

```

Neste exemplo, estamos criando uma função getTitle, que retorna o título da página ou um Nenhum objeto se houve algum problema ao recuperá-lo. Dentro getTitle, nós verificamos por um Erro HTTP, como no exemplo anterior, e também encapsular duas das linhas BeautifulSoup dentro de uma experimentar declaração. A AttributeError pode ser lançado a partir de qualquer uma dessas linhas (se o servidor não existisse, html seria um Nenhum

objeto, e html.read () jogaria um AttributeError). Poderíamos, de fato, incluir quantas linhas quiséssemos em um experimentar declaração, ou chamar outra função inteiramente, que pode lançar um AttributeError em qualquer ponto.

Ao escrever scrapers, é importante pensar sobre o padrão geral de seu código para lidar com exceções e torná-lo legível ao mesmo tempo. Você provavelmente também desejará reutilizar fortemente o código. Tendo funções genéricas, como getSiteHTML e getTitle (completo com tratamento de exceções completo) torna mais fácil - e confiável - vasculhar a web.

Análise avançada de HTML

Quando Michelangelo foi questionado sobre como ele poderia esculpir uma obra de arte tão magistral quanto seu David, ele é famoso por ter dito: "É fácil. Você apenas lasca a pedra que não se parece com David."

Embora a raspagem da web seja diferente da escultura de mármore na maioria dos outros aspectos, devemos tomar uma atitude semelhante quando se trata de extrair as informações que buscamos de páginas da web complicadas. Existem muitas técnicas para remover o conteúdo que não se parece com o que procuramos, até chegarmos às informações que buscamos. Neste capítulo, examinaremos a análise de páginas HTML complicadas para extrair apenas as informações que estamos procurando.

Você nem sempre precisa de um martelo

Pode ser tentador, quando confrontado com um nó górdio de marcas, mergulhar de cabeça e usar declarações de várias linhas para tentar extrair suas informações. No entanto, lembre-se de que colocar em camadas as técnicas usadas nesta seção com abandono imprudente pode levar a códigos difíceis de depurar, frágeis ou ambos. Antes de começar, vamos dar uma olhada em algumas das maneiras pelas quais você pode evitar totalmente a necessidade de análise avançada de HTML!

Digamos que você tenha algum conteúdo de destino. Talvez seja um nome, estatística ou bloco de texto. Talvez esteja enterrado em 20 tags profundamente em um mingau HTML, sem tags úteis ou atributos HTML para serem encontrados.

Digamos que você mergulhe de cabeça e escreva algo como a seguinte linha para tentar a extração:

```
bsObj . encontrar tudo ( "mesa" ) [ 4 ] . encontrar tudo ( "tr" ) [ 2 ] . encontrar ( "td" ) . encontrar tudo ( "div" ) [ 1 ] . encontrar ( "uma" )
```

Isso não parece tão bom. Além da estética da linha, mesmo a menor alteração no site por um administrador do site pode quebrar seu raspador de web por completo. Então quais são suas opções?

- Procure um link "imprimir esta página", ou talvez uma versão móvel do site que tenha HTML melhor formatado (mais sobre se apresentar como um dispositivo móvel - e receber versões do site móvel - em [Capítulo 12](#))
- Procure as informações ocultas em um arquivo JavaScript. Lembre-se de que pode ser necessário examinar os arquivos JavaScript importados para fazer isso. Por exemplo, uma vez eu coletei endereços de ruas (junto com latitude e longitude) de um site em uma matriz perfeitamente formatada, observando o JavaScript do Google Map embutido que exibia um ponto exato sobre cada endereço.
- Isso é mais comum para títulos de página, mas as informações podem estar disponíveis no URL da própria página.
- Se as informações que você está procurando são exclusivas deste site por algum motivo, você está sem sorte. Caso contrário, tente pensar em outras fontes de onde você possa obter essas informações. Existe outro site com os mesmos dados? Este site está exibindo dados que extraiu ou agregou de outro site?

Especialmente quando se depara com dados ocultos ou mal formatados, é importante não apenas começar a cavar. Respire fundo e pense em alternativas. Se você tiver certeza de que não existem alternativas, o resto deste capítulo é para você.

Outra porção de BeautifulSoup

Dentro [Capítulo 1](#), demos uma olhada rápida na instalação e execução do BeautifulSoup, bem como na seleção de objetos um por vez. Nesta seção, discutiremos a busca de tags por atributos, trabalho com listas de tags e análise de navegação em árvore.

Quase todos os sites que você encontra contêm folhas de estilo. Embora você possa pensar que uma camada de estilo em sites projetada especificamente para navegador e interpretação humana pode ser uma coisa ruim, o advento do CSS é na verdade uma bênção para web scrapers. CSS depende da diferenciação de elementos HTML que, de outra forma, poderiam ter exatamente a mesma marcação para estilizá-los de maneira diferente. Ou seja, algumas tags podem ter a seguinte aparência:

```
<span classe = "verde" > </ span>
```

enquanto outros são assim:

```
<span classe = "vermelho" > </ span>
```

Os web scrapers podem separar facilmente essas duas tags diferentes com base em sua classe; por exemplo, eles podem usar BeautifulSoup para capturar todo o texto em vermelho, mas nenhum texto em verde. Como o CSS depende desses atributos de identificação para estilizar sites de maneira apropriada, é quase certo que esses atributos de classe e ID serão abundantes na maioria dos sites modernos.

Vamos criar um exemplo de raspador da web que raspe a página localizada em <http://www.pythonscraping.com/pages/warandpeace.html>.

Nesta página, as falas faladas pelos personagens da história estão em vermelho, enquanto os nomes dos próprios personagens estão em verde. Você pode ver o período tags, que se referem às classes CSS apropriadas, no seguinte exemplo do código-fonte da página:

```
"<span class = "red"> Céus! que ataque virulento! </span>" respondeu <span class = "green"> o príncipe </span>, nem um pouco desconcertado com a recepção.
```

Podemos pegar a página inteira e criar um objeto BeautifulSoup com ela usando um programa semelhante ao usado em [Capítulo 1](#):

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup html = urlopen ( "http://www.pythonscraping.com/pages/warandpeace.html"
)
bsObj = BeautifulSoup ( html )
```

Usando este objeto BeautifulSoup, podemos usar o encontrar tudo para extrair uma lista Python de nomes próprios encontrados selecionando apenas o texto dentro de ` ` Tag (encontrar tudo é uma função extremamente flexível que usaremos muito mais tarde neste livro):

```
lista de nomes = bsObj . encontrar tudo ( "periodo" , { "classe" : "verde" })
para nome dentro lista de nomes :
    impressão ( nome . get_text ())
```

Quando executado, deve listar todos os nomes próprios no texto, na ordem em que aparecem em *Guerra e Paz*. Então, o que está acontecendo aqui? Anteriormente, chamamos `bsObj.tagName` a fim de obter a primeira ocorrência dessa tag na página. Agora, estamos ligando `bsObj.findAll (tagName, tagAttributes)` para obter uma lista de todas as tags em a página, em vez de apenas a primeira.

Depois de obter uma lista de nomes, o programa percorre todos os nomes da lista e imprime `name.get_text ()` para separar o conteúdo das tags.



Quando obter_texto () e Quando preservar as tags

`.get_text ()` remove todas as marcas do documento com o qual você está trabalhando e retorna uma string contendo apenas o texto. Por exemplo, se você estiver trabalhando com um grande bloco de texto que contém muitos hiperlinks, parágrafos e outras marcas, todos eles serão removidos e você ficará com um bloco de texto sem marcas.

Lembre-se de que é muito mais fácil encontrar o que você procura em um objeto BeautifulSoup do que em um bloco de texto. Chamando `.get_text ()` deve ser sempre a última coisa a fazer, imediatamente antes de imprimir, armazenar ou manipular seus dados finais. Em geral, você deve tentar preservar a estrutura de tags de um documento o máximo possível.

find () e findAll () com BeautifulSoup

BeautifulSoup's encontrar() e encontrar tudo() são as duas funções que você provavelmente mais usará. Com eles, você pode facilmente filtrar páginas HTML para encontrar listas de tags desejadas, ou uma única tag, com base em seus vários atributos.

As duas funções são extremamente semelhantes, conforme evidenciado por suas definições na documentação do BeautifulSoup:

```
encontrar tudo ( etiqueta , atributos , recursivo , texto , limite , palavras-chave )
encontrar ( etiqueta , atributos , recursivo , texto , palavras-chave )
```

Com toda a probabilidade, 95% das vezes, você só precisará usar os dois primeiros argumentos: etiqueta e atributos.

No entanto, vamos dar uma olhada em todos os argumentos com mais detalhes.

o etiqueta argumento é aquele que vimos antes - você pode passar um nome de string de uma tag ou até mesmo uma lista Python de nomes de tag de string. Por exemplo, o seguinte retornará uma lista de todas as tags de cabeçalho em um documento:¹

```
. encontrar tudo ( { "h1" , "h2" , "h3" , "h4" , "h5" , "h6" } )
```

o atributos argumento pega um dicionário Python de atributos e combina tags que contêm qualquer um desses atributos. Por exemplo, a seguinte função retornaria *ambos* o verde e vermelho período tags no documento HTML:

```
. encontrar tudo ( "período" , { "classe" : "verde" , "classe" : "vermelho" } )
```

o recursivo argumento é um booleano. Quão profundamente no documento você deseja ir? E se recursão está configurado para Verdadeiro, a encontrar tudo A função analisa os filhos e os filhos dos filhos em busca de tags que correspondam aos seus parâmetros. Se for falso, ele olhará apenas para as marcas de nível superior em seu documento. Por padrão, encontrar tudo funciona recursivamente (recorrente está configurado para Verdadeiro); geralmente é uma boa ideia deixar como está, a menos que você realmente saiba o que precisa fazer e o desempenho seja um problema.

o texto O argumento é incomum porque corresponde com base no conteúdo de texto das tags, em vez de nas propriedades das próprias tags. Por exemplo, se quisermos encontrar o número de vezes que “o príncipe” foi cercado por tags na página de exemplo, podemos substituir nosso. encontrar tudo() função no exemplo anterior com as seguintes linhas:

```
lista de nomes = bsObj . encontrar tudo ( texto = "o príncipe" )
impressão ( len ( lista de nomes ) )
```

O resultado disso é “7”.

1 Se você deseja obter uma lista de todos h <some_level> tags no documento, existem maneiras mais sucintas de escrever esse código para realizar a mesma coisa. Vamos dar uma olhada em outras maneiras de abordar esses tipos de problemas na seção [BeautifulSoup e expressões regulares](#).

o limite argumento, é claro, só é usado no encontrar tudo método; encontrar é equivalente ao mesmo encontrar tudo chamada, com um limite de 1. Você pode definir isso se estiver interessado apenas em recuperar o primeiro x itens da página. Esteja ciente, porém, de que isso fornece os primeiros itens na página na ordem em que ocorrem, não necessariamente os primeiros que você deseja.

o palavra chave O argumento permite que você selecione tags que contêm um determinado atributo. Por exemplo:

```
allText = bsObj . encontrar tudo ( Eu iria = "texto" )
impressão ( allText [ 0 ] . get_text () )
```



Uma advertência para o argumento da palavra-chave

o palavra chave o argumento pode ser muito útil em algumas situações. No entanto, é tecnicamente redundante como um recurso BeautifulSoup. Tenha em mente que tudo o que pode ser feito com palavra chave também pode ser realizado usando técnicas que discutiremos mais tarde neste capítulo (ver [Expressões regulares](#) e [Expressões Lambda](#))

Por exemplo, as duas linhas a seguir são idênticas:

```
bsObj . encontrar tudo ( Eu iria = "texto" )
bsObj . encontrar tudo ( "" , { "Eu iria" : "texto" } )
```

Além disso, você pode ocasionalmente ter problemas ao usar palavra chave, mais notavelmente ao procurar elementos por seus classe atributo, porque classe é uma palavra-chave protegida em Python. Isso é, classe é uma palavra reservada em Python que não pode ser usada como uma variável ou nome de argumento (sem relação com BeautifulSoup.findAll ()) palavra chave argumento, discutido anteriormente).² Por exemplo, se você tentar a seguinte chamada, obterá um erro de sintaxe devido ao uso não padrão de classe:

```
bsObj . encontrar tudo ( classe = "verde" )
```

Em vez disso, você pode usar a solução um tanto desajeitada do BeautifulSoup, que envolve a adição de um sublinhado:

```
bsObj . encontrar tudo ( classe_ = "verde" )
```

Como alternativa, você pode incluir classe entre aspas:

```
bsObj . encontrar tudo ( "" , { "classe" : "verde" } )
```

Neste ponto, você pode estar se perguntando: “Mas espere, eu já não sei como obter uma lista de tags por atributo - passando atributos para a função em uma lista de dicionário?”

² A Referência da Linguagem Python fornece um [lista completa de palavras-chave protegidas](#).

Lembre-se de passar uma lista de tags para `encontrar tudo()` através da lista de atributos atua como um filtro “ou” (ou seja, ele seleciona uma lista de todas as tags que possuem tag1 ou tag2 ou tag3 ...). Se você tiver uma lista extensa de tags, pode acabar com muitas coisas que não quer. o palavra chave argumento permite que você adicione um filtro “e” adicional a ele.

Outros objetos BeautifulSoup

Até agora no livro, você viu dois tipos de objetos na biblioteca BeautifulSoup:

`bsObj . div . h1`

BeautifulSoup objetos

Visto nos exemplos de código anteriores como `bsObj`

Tag objetos

Obtido em listas ou individualmente por telefone encontrar e encontrar tudo com um BeautifulSoup objeto, ou detalhamento, como em:

No entanto, existem mais dois objetos na biblioteca que, embora menos usados, ainda são importantes para saber sobre:

NavigableString objetos

Usado para representar texto dentro de tags, ao invés das próprias tags (algumas funções operam e produzem, NavigableStrings, em vez de objetos de tag).

o Comente objeto

Usado para localizar comentários HTML em tags de comentários, `<! - como este ->`

Esses quatro objetos são os únicos que você encontrará (no momento em que este livro foi escrito) na biblioteca BeautifulSoup.

Árvores de navegação

o encontrar tudo A função é responsável por encontrar tags com base em seu nome e atributo. Mas e se você precisar encontrar uma tag com base em sua localização em um documento? É aí que a navegação em árvore é útil. Dentro [Capítulo 1](#), vimos como navegar em uma árvore BeautifulSoup em uma única direção:

`bsObj . etiqueta . subTag . outraSubTag`

Agora vamos ver como navegar para cima, através e diagonalmente através das árvores HTML usando nosso site de compras online altamente questionável <http://www.pythonscraping.com/pages/page3.html> como uma página de exemplo para raspar (ver [Figura 2-1](#)):



Totally Normal Gifts

Here is a collection of totally normal, totally reasonable gifts that your friends are sure to love! Our collection is hand-curated.

We haven't figured out how to make online shopping carts yet, but you can send us a check to:

123 Main St.

Abuja, Nigeria

We will then send your totally amazing gift, pronto! Please include an extra \$5.00 for gift wrapping.

Item Title	Description	Cost	Image
Vegetable Basket	This vegetable basket is the perfect gift for your health conscious (or overweight) friends! <i>Now with super-colorful bell peppers!</i>	\$15.00	
Russian Nesting Dolls	Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mean "extremely expensive"! <i>8 entire dolls per set! Octuple the presents!</i>	\$10,000.52	
Fish Painting	If something seems fishy about this painting, it's because it's a fish! <i>Also hand-painted by trained monkeys!</i>	\$10,005.00	
Dead Parrot	This is an ex-parrot! <i>Or maybe he's only resting?</i>	\$0.50	

Figura 2-1. Captura de tela de <http://www.pythonscraping.com/pages/page3.html>

O HTML para esta página, mapeado como uma árvore (com algumas tags omitidas para abreviar), é semelhante a:

- html
 - corpo
 - div.wrapper
 - h1
 - div.content
 - tabela # giftList
 - tr
 - o
 - o
 - o
 - o
 - tr.gift # gift1
 - td
 - td

```
- span.excitingNote
- td
- td

- img
. . . as linhas da tabela continuam ...
- div.footer
```

Usaremos essa mesma estrutura HTML como exemplo nas próximas seções.

Lidando com crianças e outros descendentes

Na ciência da computação e em alguns ramos da matemática, você costuma ouvir falar de coisas horribéis feitas a crianças: movê-las, armazená-las, removê-las e até matá-las. Felizmente, na BeautifulSoup, as crianças são tratadas de maneira diferente.

Na biblioteca BeautifulSoup, bem como em muitas outras bibliotecas, há uma distinção entre *crianças* e *descendentes*: assim como na árvore genealógica humana, os filhos estão sempre exatamente uma marca abaixo de um dos pais, enquanto os descendentes podem estar em qualquer nível na árvore abaixo de um dos pais. Por exemplo, o tr tags são filhos de mesa tag, enquanto tr, th, td, img, e período são todos descendentes de mesa tag (pelo menos em nossa página de exemplo). Todos os filhos são descendentes, mas nem todos os descendentes são filhos.

Em geral, as funções do BeautifulSoup sempre lidarão com os descendentes da tag atual selecionada. Por exemplo, bsObj.body.h1 seleciona o primeiro h1 tag que é descendente da tag body. Ele não encontrará tags localizadas fora do corpo.

Similarmente, bsObj.div.findAll ("img") vai encontrar o primeiro div tag no documento e, em seguida, recupere uma lista de todos img tags que são descendentes daquele div tag.

Se você deseja encontrar apenas descendentes que são filhos, você pode usar o. crianças tag:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup

html = urlopen ( "http://www.pythonscraping.com/pages/page3.html" )
bsObj = BeautifulSoup ( html )

para criança dentro bsObj . encontrar ( "mesa" , { "Eu iria" : "lista de presentes" } ) . crianças :
    impressão ( criança )
```

Este código imprime a lista de linhas de produtos no lista de presentes mesa. Se você fosse escrever usando o descendentes () função em vez do crianças() função, cerca de duas dúzias de tags seriam encontradas dentro da tabela e impressas, incluindo img Tag, período tags, e individual td Tag. Definitivamente, é importante diferenciar entre filhos e descendentes!

Lidando com irmãos

The BeautifulSoup next_siblings () função torna trivial coletar dados de tabelas, especialmente aquelas com linhas de título:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup html = urlopen ( "http://www.pythonscraping.com/pages/page3.html"
)
bsObj = BeautifulSoup ( html )

para irmão dentro bsObj . encontrar ( "mesa" , { "Eu iria" : "lista de presentes" }) . tr . next_siblings :
    impressão ( irmão )
```

A saída desse código é imprimir todas as linhas de produtos da tabela de produtos, exceto a primeira linha do título. Por que a linha do título é ignorada? Duas razões: primeiro, os objetos não podem ser irmãos de si mesmos. Sempre que você obtém irmãos de um objeto, o próprio objeto não será incluído na lista. Em segundo lugar, esta função chama Próximo irmãos apenas. Se tivéssemos que selecionar uma linha no meio da lista, por exemplo, e chamar next_siblings

nele, apenas os irmãos subsequentes (próximos) seriam retornados. Então, selecionando a linha do título e chamando irmãos_próximos, podemos selecionar todas as linhas da tabela, sem selecionar a própria linha do título.



Faça seleções específicas

Observe que o código anterior funcionará tão bem, se selecionarmos bsObj.table.tr ou mesmo apenas bsObj.tr para selecionar a primeira linha da tabela. No entanto, no código, eu passo por todos os problemas de escrever tudo de uma forma mais longa:

```
bsObj . encontrar ( "mesa" , { "Eu iria" : "lista de presentes" }) . tr
```

Mesmo que pareça que há apenas uma tabela (ou outra tag de destino) na página, é fácil perder coisas. Além disso, os layouts de página mudam o tempo todo. O que antes era o primeiro desse tipo na página, pode algum dia ser a segunda ou terceira tag desse tipo encontrada na página. Para tornar seus raspadores mais robustos, é melhor ser o mais específico possível ao fazer seleções de etiquetas. Aproveite as vantagens dos atributos de tag quando estiverem disponíveis.

Como complemento para irmãos_próximos, a irmãos_antigos A função geralmente pode ser útil se houver uma tag facilmente selecionável no final de uma lista de tags irmãs que você gostaria de obter.

E, claro, existem os next_sibling e anterior_sibling funções, que desempenham quase a mesma função que next_siblings e irmãos_antes, exceto que eles retornam uma única tag em vez de uma lista deles.

Lidando com seus pais

Ao raspar páginas, você provavelmente descobrirá que precisa encontrar pais de tags com menos frequência do que precisa encontrar seus filhos ou irmãos. Normalmente, quando olhamos para páginas HTML com o objetivo de rastreá-las, começamos observando a camada superior de tags e, em seguida, descobrimos como aprofundar até a parte exata de dados que desejamos. Ocasionalmente, no entanto, você pode se encontrar em situações estranhas que exigem as funções de localização dos pais da BeautifulSoup., pai e pais. Por exemplo:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup

html = urlopen ( "http://www.pythonscraping.com/pages/page3.html" )
bsObj = BeautifulSoup ( html )
impressão ( bsObj . encontrar ( "img" , { "src" : "../img/gifts/img1.jpg"
}) . pai . anterior_sibling . get_text () )
```

Este código irá imprimir o preço do objeto representado pela imagem no local .. / img / gifts / img1.jpg (neste caso, o preço é “\$ 15,00”).

Como é que isso funciona? O diagrama a seguir representa a estrutura em árvore da parte da página HTML com a qual estamos trabalhando, com etapas numeradas:

- <tr>
 - <td>
 - <td>
 - <td> (3)
 - “\$ 15,00” (4)
 - s <td> (2)
 - (1)

1. A tag da imagem onde src = “.. / img / gifts / img1.jpg” é selecionado pela primeira vez.
2. Selecionamos o pai dessa tag (neste caso, o < td> tag).
3. Selecionamos o anterior_sibling do < td> tag (neste caso, o < td> marcar isso contém o valor em dólares do produto).
4. Selecionamos o texto dentro dessa tag, “\$ 15,00”

Expressões regulares

Como diz a velha piada da informática: “Digamos que você tenha um problema e decida resolvê-lo com expressões regulares. Bem, agora você tem dois problemas.”

Infelizmente, as expressões regulares (muitas vezes abreviadas para *regex*) são frequentemente ensinados usando grandes tabelas de símbolos aleatórios, amarrados juntos para parecer um monte de bobagens. este

tende a afastar as pessoas e, mais tarde, elas entram na força de trabalho e escrevem funções de pesquisa e filtragem desnecessariamente complicadas, quando tudo de que precisavam era uma expressão regular de uma linha em primeiro lugar!

Felizmente para você, as expressões regulares não são tão difíceis de começar e executar rapidamente, e podem ser facilmente aprendidas olhando e experimentando alguns exemplos simples.

Expressões regulares são assim chamadas porque são usadas para identificar strings regulares; ou seja, eles podem dizer definitivamente: "Sim, esta string que você me deu segue as regras e eu a devolverei" ou "Esta string não segue as regras e vou descartá-la". Isso pode ser extremamente útil para digitalizar rapidamente grandes documentos em busca de strings que se pareçam com números de telefone ou endereços de e-mail.

Observe que usei a frase *string regular*. O que é uma string regular? É qualquer string que pode ser gerada por uma série de regras lineares, ³ tal como:

1. Escreva a letra "a" pelo menos uma vez.
2. Anexe a isso a letra "b" exatamente cinco vezes.
3. Acrescente a isso a letra "c" qualquer número par de vezes.
4. Opcionalmente, escreva a letra "d" no final.

As strings que seguem essas regras são: "aaaabbbbcccd," "aabbbbcc" e assim por diante (há um número infinito de variações).

As expressões regulares são apenas uma forma abreviada de expressar esses conjuntos de regras. Por exemplo, aqui está a expressão regular para a série de etapas que acabamos de descrever:

$aa^* bbbb (cc)^* (d |)$

Esta string pode parecer um pouco assustadora no início, mas fica mais claro quando a dividimos em seus componentes:

aa^*

A carta *uma* é escrita, seguido por *uma** (lido como *uma estrela*) que significa "qualquer número de as, incluindo 0 deles." Desta forma, podemos garantir que a carta *uma* é escrito pelo menos uma vez.

$bbbb$

Sem efeitos especiais aqui - apenas cinco b's em uma fileira.

³ Você pode estar se perguntando: "Existem expressões 'irregulares'?" As expressões não regulares estão além do escopo deste livro, mas abrangem cadeias de caracteres como "escreva um número primo de a, seguido por exatamente o dobro desse número de b" ou "escreva um palíndromo". É impossível identificar strings desse tipo com uma expressão regular. Felizmente, nunca estive em uma situação em que meu raspador da web precisasse identificar esses tipos de strings.

(cc) *

Qualquer número par de coisas pode ser agrupado em pares, então, a fim de aplicar esta regra sobre coisas pares, você pode escrever dois *c*'s, coloque-os entre parênteses e escreva um asterisco depois dele, o que significa que você pode ter qualquer número de *pares* do *c*'s
(note que isso pode significar 0 pares, também).

(d /)

Adicionar uma barra no meio de duas expressões significa que pode ser “essa coisa *ou* aquela coisa.” Neste caso, estamos dizendo “adicone um *d* seguido por um espaço *ou* basta adicionar um espaço sem um *d*. “Desta forma podemos garantir que haja, no máximo, uma *d*, seguido por um espaço, completando o barbante.



Experimentando com RegEx

Ao aprender a escrever expressões regulares, é fundamental brincar com elas e ter uma ideia de como funcionam.

Se você não quiser abrir um editor de código, escrever algumas linhas e executar seu programa para ver se uma expressão regular funciona conforme o esperado, você pode ir a um site como [RegexPal](#) e teste suas expressões regulares na hora.

Um exemplo clássico de expressões regulares pode ser encontrado na prática de identificação de endereços de e-mail. Embora as regras exatas que regem os endereços de e-mail variem ligeiramente de servidor de e-mail para servidor de e-mail, podemos criar algumas regras gerais. A expressão regular correspondente para cada uma dessas regras é mostrada na segunda coluna:

<p><i>Regra 1</i></p> <p>A primeira parte de um endereço de e-mail contém pelo menos um dos seguintes: letras maiúsculas, letras minúsculas, números de 0 a 9, pontos (.), sinais de mais (+) ou sublinhados (_).</p>	<p>[A-Za-z0-9 _ +] +</p> <p>A abreviação da expressão regular é muito inteligente. Por exemplo, ele sabe que “AZ” significa “qualquer letra maiúscula, de A a Z.” Ao colocar todas essas sequências e símbolos possíveis entre colchetes (em oposição aos parênteses), estamos dizendo “este símbolo pode ser qualquer uma das coisas que listamos entre colchetes.” Observe também que o sinal + significa “esses caracteres podem ocorrer quantas vezes quiserem, mas devem ocorrer pelo menos uma vez”.</p>
<p><i>Regra 2</i></p> <p>Depois disso, o endereço de e-mail contém o símbolo @.</p>	<p>@</p> <p>Isso é bastante simples: o símbolo @ deve ocorrer no meio e deve ocorrer exatamente uma vez.</p>

<i>Regra 3</i> O endereço de e-mail deve conter pelo menos uma letra maiúscula ou minúscula.	[A-Za-z] + Podemos usar apenas letras na primeira parte do nome de domínio, após o símbolo @. Além disso, deve haver pelo menos um personagem.
<i>Regra 4</i> Isso é seguido por um ponto (.).	l. Você deve incluir um ponto (.) Antes do nome de domínio.
<i>Regra 5</i> Finalmente, o endereço de e-mail termina com <i>com, org, edu, ou internet</i> (na realidade, existem muitos possíveis domínios, mas esses quatro devem ser suficientes para fins de exemplo).	(com org edu net) Isso lista as possíveis sequências de letras que podem ocorrer após o ponto na segunda parte de um endereço de e-mail.

Ao concatenar todas as regras, chegamos à expressão regular:

`[A-Za-z0-9 _]+ @ [A-Za-z] + l. (Com | org | edu | net)`

Ao tentar escrever qualquer expressão regular do zero, é melhor primeiro fazer uma lista de etapas que descrevam concretamente a aparência de sua string de destino. Preste atenção aos casos extremos. Por exemplo, se você está identificando números de telefone, está considerando códigos de país e extensões?

Tabela 2-1 lista alguns símbolos de expressão regular comumente usados, com uma breve explicação e exemplo. Esta lista não está completa e, como mencionado antes, você pode encontrar pequenas variações de idioma para idioma. No entanto, esses 12 símbolos são as expressões regulares mais comumente usadas em Python e podem ser usados para encontrar e coletar praticamente qualquer tipo de string.

Tabela 2-1. Símbolos de expressão regular comumente usados

Significado do (s) símbolo (s)	Exemplo	Correspondências de exemplo
*	Corresponde ao caractere anterior, subexpressão ou caractere entre colchetes, a * b * mais vezes	0 ou aaaaaaaa, aabbbbb, bbbbb
+	Corresponde ao caractere anterior, subexpressão ou caractere entre colchetes, a + b + 1 ou mais vezes	aaaaaaaaab, aabbbbb, abbbbb

[]	Corresponde a qualquer caractere entre colchetes (ou seja, "Escolha qualquer uma dessas coisas")	[AZ] *	MACÃ, CAPITAIS, QWERTY
()	Uma subexpressão agrupada (estes são avaliados primeiro, na "ordem das operações" das expressões regulares)	(a * b) *	aabaab, abaaab, ababaaaaab
{m, n}	Corresponde ao caractere anterior, subexpressão ou caractere entre colchetes entre m e n vezes (inclusive)	a {2,3} b {2,3}	aabb, aaabb, aab
[^]	Corresponde a qualquer caractere único que seja <i>não</i> nos colchetes	[^ AZ] *	maçã, minúsculas, qwerty
	Corresponde a qualquer caractere, sequência de caracteres ou subexpressão, separados por "I" (observe que esta é uma barra vertical ou "barra vertical", não um "I" maiúsculo)	b (a i e) d	ruim, lance, cama
.	Corresponde a qualquer caractere único (incluindo símbolos, números, um espaço, etc.)	.bd	ruim, bzd, b \$ d, bd
^	Indica que um caractere ou subexpressão ocorre no início de uma string	^ a	apple, asdf, a
\	Um caractere de escape (permite que você use caracteres "especiais" como seu significado literal)	\. \ \\	. \
\$	Freqüentemente usado no final de uma expressão regular, significa "combine isso até o final da string". Sem ele, toda expressão regular tem um defacto ". **" No final dela, aceitando strings onde apenas a primeira parte da string corresponde. Isso pode ser considerado análogo ao símbolo ^.	[AZ] * [az] * \$	ABCabc, zzzyx, Bob
?	"Não contém." Este estranho par de símbolos, imediatamente precedendo um caractere (ou expressão regular), indica que aquele caractere não deve ser encontrado naquele local específico na string maior. Isso pode ser complicado de usar; afinal, o caractere pode ser encontrado em uma parte diferente da string. Se estiver tentando eliminar um caractere totalmente, use em conjunto com ^ e \$ em cada extremidade.	^(?! [AZ]). * \$	No caps-here, \$ ymb0ls a4e fl ne



Expressões regulares: nem sempre regulares!

A versão padrão das expressões regulares (aquela que estamos abordando neste livro, e que é usada por Python e BeautifulSoup) é baseada na sintaxe usada por Perl. A maioria das linguagens de programação modernas usa esta ou uma muito semelhante a ela. Esteja ciente, entretanto, que se você estiver usando expressões regulares em outro idioma, poderá encontrar problemas. Até mesmo algumas linguagens modernas, como Java, têm pequenas diferenças na maneira como lidam com expressões regulares. Na dúvida, leia a documentação!

Expressões regulares e BeautifulSoup

Se a seção anterior sobre expressões regulares pareceu um pouco desconexa da missão deste livro, é aqui que tudo se encaixa. BeautifulSoup e expressões regulares andam de mãos dadas quando se trata de copiar a web. Na verdade, a maioria das funções que aceitam um argumento de string (por exemplo, find (id = "aTagIdHere")) também terá uma expressão regular.

Vamos dar uma olhada em alguns exemplos, copiando a página encontrada em <http://www.pythonscraping.com/pages/page3.html>.

Observe que há muitas imagens de produtos no site - elas têm o seguinte formato:

```
<img src = "../ img / gifts / img3.jpg">
```

Se quiséssemos obter URLs para todas as imagens do produto, pode parecer bastante simples no início: apenas pegue todas as tags de imagem usando. findAll ("img"), certo? Mas há um problema. Além das óbvias imagens "extras" (por exemplo, logotipos), os sites modernos costumam ter imagens ocultas, imagens em branco usadas para espaçar e alinhar elementos e outras marcas de imagem aleatórias das quais você pode não estar ciente. Certamente, você não pode contar que as únicas imagens na página sejam imagens de produtos.

Vamos supor também que o layout da página pode mudar, ou que, por qualquer motivo, não queremos depender da posição da imagem na página para encontrar a tag correta. Esse pode ser o caso quando você está tentando obter elementos específicos ou pedaços de dados que estão espalhados aleatoriamente em um site. Por exemplo, pode haver uma imagem de produto em destaque em um layout especial no topo de algumas páginas, mas não em outras.

A solução é procurar algo que o identifique na própria tag. Nesse caso, podemos olhar o caminho do arquivo das imagens do produto:

```
de urllib.request
importar urlopenfrom bs4
importar BeautifulSoupimport re

html = urlopen ( "http://www.pythonscraping.com/pages/page3.html" )
bsObj = BeautifulSoup ( html )
imagens = bsObj . encontrar tudo ( "img" , { "src" : re . compilar ( "\. \. \. / img \ / gifts / img. * \. jpg" )})
para imagem dentro imagens :
    impressão ( imagem [ "src" ])
```

Isso imprime apenas os caminhos relativos da imagem que começam com .. / img / gifts / img e terminam em. jpg, cujo resultado é o seguinte:

```
.. /img/gifts/img1.jpg
.. /img/gifts/img2.jpg
.. /img/gifts/img3.jpg
.. /img/gifts/img4.jpg
.. /img/gifts/img6.jpg
```

Uma expressão regular pode ser inserida como qualquer argumento em uma expressão BeautifulSoup, permitindo uma grande flexibilidade na localização de elementos de destino.

Acessando Atributos

Até agora, vimos como acessar e filtrar tags e acessar o conteúdo dentro delas. No entanto, muitas vezes, em web scraping, você não está procurando o conteúdo de uma tag; você está procurando por seus atributos. Isso se torna especialmente útil para tags como `<a>`, onde o URL para o qual está apontando está contido no `href` atributo, ou o `` tag, onde a imagem alvo está contida no `src` atributo.

Com objetos de tag, uma lista Python de atributos pode ser acessada automaticamente chamando:

```
myTag.attrs
```

Lembre-se de que isso retorna literalmente um objeto de dicionário Python, o que torna trivial a recuperação e a manipulação desses atributos. O local de origem de uma imagem, por exemplo, pode ser encontrado usando a seguinte linha:

```
myImgTag.attrs['src']
```

Expressões Lambda

Se você tem educação formal em ciência da computação, provavelmente aprendeu sobre expressões lambda uma vez na escola e nunca mais as usou. Se não o fizer, eles podem ser desconhecidos para você (ou familiares apenas como "aquela coisa que tenho pretendido aprender em algum momento"). Nesta seção, não entraremos profundamente nessas funções extremamente úteis, mas veremos alguns exemplos de como elas podem ser úteis em web scraping.

Essencialmente, uma expressão lambda é uma função que é passada para outra função como uma variável; isto é, em vez de definir uma função como `f(x, y)`, você pode definir uma função como `f(g(x), y)`, ou mesmo `f(g(x), h(x))`.

BeautifulSoup nos permite passar certos tipos de funções como parâmetros para o encontrar tudo função. A única restrição é que essas funções devem tomar um objeto tag como argumento e retornar um booleano. Cada objeto de tag que BeautifulSoup encontra é avaliado nesta função, e as tags avaliadas como "verdadeiras" são retornadas enquanto o resto é descartado.

Por exemplo, o seguinte recupera todas as tags que têm exatamente dois atributos:

```
soup.findAll(tag=lambda: len(tag.attrs) == 2)
```

Ou seja, ele encontrará tags como as seguintes:

```
<div classe = "corpo" id = "conteúdo" > </div>
<span estilo = "cor vermelha" classe = "título" > </span>
```

Usando funções lambda no BeautifulSoup, os seletores podem atuar como um ótimo substituto para escrever uma expressão regular, se você se sentir confortável em escrever um pequeno código.

Além do BeautifulSoup

Apesar BeautifulSoup é usado ao longo deste livro (e é uma das bibliotecas HTML mais populares disponíveis para Python), tenha em mente que não é a única opção. Se o BeautifulSoup não atender às suas necessidades, verifique estas outras bibliotecas amplamente utilizadas:

/xm/

este [biblioteca](#) é usado para analisar documentos HTML e XML, e é conhecido por ter um nível muito baixo e fortemente baseado em C. Embora demore um pouco para aprender (uma curva de aprendizado íngreme na verdade significa que você aprende muito rápido), é muito rápido em analisar a maioria dos documentos HTML.

Analizador de HTML

Este é o Python embutido [biblioteca de análise](#). Como não requer instalação (exceto, obviamente, ter o Python instalado em primeiro lugar), pode ser extremamente conveniente de usar.

Começando a rastrear

Até agora, os exemplos no livro cobriram páginas estáticas únicas, com exemplos enlatados um tanto artificiais. Neste capítulo, começaremos a examinar alguns problemas do mundo real, com scrapers percorrendo várias páginas e até vários sites.

Os rastreadores da web são chamados assim porque rastreiam a web. Em seu núcleo está um elemento de recursão. Eles devem recuperar o conteúdo da página para um URL, examinar essa página para outro URL e recuperar *este* página, ad infinitum.

Porém, cuidado: só porque você pode rastrear a Web, não significa que você sempre deve fazer isso. Os raspadores usados nos exemplos anteriores funcionam muito bem em situações em que todos os dados de que você precisa estão em uma única página. Com os rastreadores da web, você deve estar extremamente consciente de quanta largura de banda está usando e fazer todos os esforços para determinar se há uma maneira de tornar a carga do servidor de destino mais fácil.

Atravessando um único domínio

Mesmo que você não tenha ouvido falar de “Seis Graus de Wikipedia”, quase certamente já ouviu falar de seu homônimo, “Seis Graus de Kevin Bacon”. Em ambos os jogos, o objetivo é ligar dois assuntos improváveis (no primeiro caso, artigos da Wikipedia que se ligam entre si, no segundo caso, atores que aparecem no mesmo filme) por uma cadeia contendo não mais do que seis no total (incluindo os dois assuntos originais).

Por exemplo, Eric Idle apareceu em *Dudley Do-Right* com Brendan Fraser, que apareceu em *O ar que eu respiro* com Kevin Bacon.¹ Nesse caso, a cadeia de Eric Idle a Kevin Bacon tem apenas três temas.

¹ Obrigado a [O Oráculo de Bacon](#) por satisfazer minha curiosidade sobre esta cadeia em particular.

Nesta seção, começaremos um projeto que se tornará um localizador de soluções dos "Seis Graus da Wikipedia". Ou seja, poderemos pegar [a página Eric Idle](#) e encontre o menor número de cliques em links que nos levará a [a página de Kevin Bacon](#).

Mas e quanto à carga do servidor da Wikipedia?

De acordo com a Wikimedia Foundation (a organização-mãe por trás da Wikipedia), as propriedades do site recebem aproximadamente 2.500 acessos por *segundo*, com mais de 99% deles no domínio da Wikipedia (ver [a seção "Volume de tráfego" da página "Wiki-mídia em figuras"](#)) Devido ao grande volume de tráfego, é improvável que seus web scrapers tenham qualquer impacto perceptível na carga do servidor da Wikipedia. No entanto, se você executar os exemplos de código neste livro extensivamente ou criar seus próprios projetos que limpam a Wikipedia, eu o encorajo a fazer [uma doação dedutível de impostos para a Fundação Wikimedia](#) - até mesmo alguns dólares compensarão a carga do servidor e ajudarão a disponibilizar recursos educacionais para todos os outros.

Você já deve saber como escrever um script Python que recupera uma página arbitrária da Wikipedia e produz uma lista de links nessa página:

```
de urllib.request importar urlopen  
de bs4 importar BeautifulSoup  
  
html = urlopen ( "http://en.wikipedia.org/wiki/Kevin_Bacon" )  
bsObj = BeautifulSoup ( html )  
para ligação dentro bsObj . encontrar tudo ( "uma" ):  
    E se ' href ' dentro ligação . attrs :  
        impressão ( ligação . attrs [ 'href' ])
```

Se você olhar a lista de links produzidos, perceberá que todos os artigos que você esperava estão lá: "Apollo 13," "Filadélfia," "Primetime Emmy Award," e assim por diante. No entanto, existem algumas coisas que também não queremos:

```
//wikimediafoundation.org/wiki/Privacy_policy  
//en.wikipedia.org/wiki/Wikipedia>Contact_us
```

Na verdade, a Wikipedia está cheia de links de barra lateral, rodapé e cabeçalho que aparecem em cada página, junto com links para as páginas de categoria, páginas de discussão e outras páginas que não contêm artigos diferentes:

```
/ wiki / Categoria: Articles_with_unsourced_statements_from_April_2014 / wiki / Talk: Kevin_Bacon
```

Recently a friend of mine, while working on a similar Wikipedia-scraping project, mentioned he had written a very large filtering function, with over 100 lines of code, in order to determine whether an internal Wikipedia link was an article page or not. Unfortunately, he had not spent much time up front trying to find patterns between "article links" and "other links," or he might have discovered the trick. If you examine

the links that point to article pages (as opposed to other internal pages), they all have three things in common:

- They reside within the div with the id set to bodyContent
- The URLs do not contain semicolons
- The URLs begin with / wiki/

We can use these rules to revise the code slightly to retrieve only the desired article links:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen ("http://en.wikipedia.org/wiki/Kevin_Bacon")
bsObj = BeautifulSoup ( html )
for link in bsObj . find ( "div" , { "id" : "bodyContent" } ) . findAll ( "a" ,
    href = re . compile ( "^/(wiki)((?:.)*$)" )):
    if ' href' in link . attrs :
        print ( link . attrs [ 'href' ])
```

If you run this, you should see a list of all article URLs that the Wikipedia article on Kevin Bacon links to.

Of course, having a script that finds all article links in one, hardcoded Wikipedia article, while interesting, is fairly useless in practice. We need to be able to take this code and transform it into something more like the following:

- A single function, getLinks, that takes in a Wikipedia article URL of the form / wiki/<Article_Name> and returns a list of all linked article URLs in the same form.
- A main function that calls getLinks with some starting article, chooses a random article link from the returned list, and calls getLinks again, until we stop the program or until there are no article links found on the new page.

Here is the complete code that accomplishes this:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import re

random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen("http://en.wikipedia.org" + articleUrl)
    bsObj = BeautifulSoup(html)
    return bsObj.findAll({"a": {"href": re.compile("^(/wiki/)((?:.).*)$")}})

links = getLinks("/wiki/Kevin_Bacon")
while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs["href"]
    print(newArticle)
    links = getLinks(newArticle)
```

The first thing the program does, after importing the needed libraries, is set the random number generator seed with the current system time. This practically ensures a new and interesting random path through Wikipedia articles every time the program is run.

Pseudorandom Numbers and Random Seeds

In the previous example, I used Python's random number generator to select an article at random on each page in order to continue a random traversal of Wikipedia. However, random numbers should be used with caution.

While computers are great at calculating correct answers, they're terrible at just making things up. For this reason, random numbers can be a challenge. Most random number algorithms strive to produce an evenly distributed and hard-to-predict sequence of numbers, but a "seed" number is needed to give these algorithms something to work with initially. The exact same seed will produce the exact same sequence of "random" numbers every time, so for this reason I've used the system clock as a starter for producing new sequences of random numbers, and, thus, new sequences of random articles. This makes the program a little more exciting to run.

For the curious, the Python pseudorandom number generator is powered by the *Mersenne Twister algorithm*. While it produces random numbers that are difficult to predict and uniformly distributed, it is slightly processor intensive. Random numbers this good don't come cheap!

Em seguida, ele define o `getLinks` função, que leva em um URL de artigo do formulário / `wiki` / precede o nome de domínio da Wikipedia, `http://en.wikipedia.org`, e recupera o objeto BeautifulSoup para o HTML nesse domínio. Em seguida, ele extrai uma lista de tags de link de artigo, com base nos parâmetros discutidos anteriormente, e os retorna.

O corpo principal do programa começa com a configuração de uma lista de tags de link de artigo (o `links` variável) à lista de links na página inicial: `https://en.wikipedia.org/wiki/Kevin_Bacon`. Em seguida, ele entra em um loop, encontrando uma tag de link de artigo aleatório na página, extraíndo o `href` atributo dele, imprimindo a página e obtendo uma nova lista de links do URL extraído.

Claro, há um pouco mais para resolver o problema dos “Seis Graus da Wikipedia” do que simplesmente construir um raspador que vai de página em página. Devemos também ser capazes de armazenar e analisar os dados resultantes. Para uma continuação da solução para este problema, consulte [capítulo 5](#).



Lide com suas exceções!

Embora estejamos omitindo a maior parte do tratamento de exceções nos exemplos de código para fins de brevidade nesses exemplos, esteja ciente de que existem muitas armadilhas potenciais que podem surgir: E se a Wikipedia mudasse o nome do `bodyContent` tag, por exemplo? (Dica: o código travaria.)

Portanto, embora esses scripts possam ser executados como exemplos observados de perto, o código de produção autônomo requer muito mais tratamento de exceções do que podemos acomodar neste livro. Olhe para trás para [Capítulo 1](#) para mais informações sobre isso.

Rastreando um site inteiro

Na seção anterior, demos um passeio aleatório por um site, indo de link em link. Mas e se você precisar catalogar ou pesquisar sistematicamente todas as páginas de um site? Rastrear um site inteiro, especialmente um grande, é um processo que consome muita memória e é mais adequado para aplicativos em que um banco de dados para armazenar os resultados do rastreamento esteja prontamente disponível. No entanto, podemos explorar o comportamento desses tipos de aplicativos sem realmente executá-los em escala total. Para saber mais sobre como executar esses aplicativos usando um banco de dados, consulte [capítulo 5](#).

The Dark e DeepWebs

Você provavelmente já ouviu os termos *deep web*, *dark web*, ou *teia oculta* sendo divulgado muito, especialmente na mídia recentemente. O que eles querem dizer?

A deep web é simplesmente qualquer parte da web que não faz parte da *superfície da web*. A superfície é parte da Internet que é indexada pelos motores de busca. As estimativas variam muito, mas a deep web quase certamente representa cerca de 90% da Internet. Porque o Google não pode fazer coisas como enviar formulários, encontrar páginas que não foram vinculadas por um domínio de nível superior ou investigar sites onde o *robots.txt* proíbe, a superfície da Web permanece relativamente pequena.

A dark Web, também conhecida como Darknet ou dark Internet, é outra besta inteiramente. Ele é executado na infraestrutura de rede existente, mas usa um cliente Tor com um protocolo de aplicativo que roda em cima do HTTP, fornecendo um canal seguro para a troca de informações. Embora seja possível raspar a Darknet, assim como você faria com qualquer outro site, fazer isso está fora do escopo deste livro.

Ao contrário da dark Web, a deep Web é relativamente fácil de raspar. Na verdade, existem muitas ferramentas neste livro que irão ensiná-lo a rastrear e extrair informações de muitos lugares que os bots do Google não podem ir.

Então, quando rastrear um site inteiro pode ser útil e quando pode realmente ser prejudicial? Web scrapers que percorrem um site inteiro são bons para muitas coisas, incluindo:

Gerando um mapa do site

Há alguns anos, me deparei com um problema: um cliente importante queria uma estimativa para o redesenho de um site, mas não queria fornecer à minha empresa acesso aos recursos internos de seu sistema de gerenciamento de conteúdo atual e não tinha um mapa do site disponível publicamente. Conseguí usar um rastreador para cobrir todo o site, reunir todos os links internos e organizar as páginas na estrutura de pastas real que tinham no site. Isso me permitiu encontrar rapidamente seções do site que eu nem sabia que existiam e contar com precisão quantos designs de página seriam necessários e quanto conteúdo precisaria ser migrado.

Juntando informação

Outro cliente meu queria reunir artigos (histórias, postagens de blog, artigos de notícias, etc.) para criar um protótipo funcional de uma plataforma de pesquisa especializada. Embora esses rastreamentos de sites não precisassem ser exaustivos, eles precisavam ser bastante extensos (havia apenas alguns sites dos quais estávamos interessados em obter dados). Conseguí criar crawlers que percorriam recursivamente cada site e coletavam apenas os dados encontrados nas páginas dos artigos.

A abordagem geral para um rastreamento exaustivo de sites é começar com uma página de nível superior (como a página inicial) e pesquisar uma lista de todos os links internos dessa página. Cada um desses links é rastreado e listas adicionais de links são encontradas em cada um deles, desencadeando outra rodada de rastreamento.

Obviamente, esta é uma situação que pode explodir muito rapidamente. Se cada página tiver 10 links internos, e um site tiver cinco páginas de profundidade (uma profundidade bastante típica para uma web de tamanho médio site), então o número de páginas que você precisa rastrear é 10 5, ou 100.000 páginas, antes de ter certeza de que cobriu exaustivamente o site. Estranhamente, embora “5 páginas de profundidade e 10 links internos por página” sejam dimensões bastante típicas para um site, existem muito poucos sites com 100.000 ou mais páginas. A razão, claro, é que a grande maioria dos links internos são duplicados.

Para evitar o rastreamento da mesma página duas vezes, é extremamente importante que todos os links internos descobertos sejam formatados de forma consistente e mantidos em uma lista em execução para facilitar as pesquisas, enquanto o programa está em execução. Apenas links que são “novos” devem ser rastreados e pesquisados para links adicionais:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar re

Páginas = conjunto ()
def getLinks ( URL da página ):
    global Páginas
    html = urlopen ( "http://en.wikipedia.org" + URL da página )
    bsObj = BeautifulSoup ( html )
    para ligação dentro bsObj . encontrar tudo ( "uma" , href = re . compilar ( "^ (/ wiki / )")):
        E se ' href ' dentro ligação . attrs :
            # Encontramos uma nova página
            nova página = ligação . attrs [ 'href' ]
            impressão ( nova página )
            Páginas . adicionar ( nova página )
            getLinks ( nova página )
    getLinks ( "" )
```

Para obter o efeito total de como funciona esse negócio de crawling na web, relaxei os padrões do que constitui um “link interno que estamos procurando” de exemplos anteriores. Em vez de limitar o raspador às páginas do artigo, ele procura todos os links que começam com / wiki / independentemente de onde estejam na página e independentemente de conterem dois pontos. Lembre-se: as páginas do artigo não contêm dois pontos, mas as páginas de upload de arquivo, páginas de discussão e outras contêm dois pontos na URL).

Inicialmente, getLinks é chamado com um URL vazio. Isso é traduzido como “a página inicial da Wikipedia” assim que o URL vazio é adicionado http://en.wikipedia.org dentro da função. Em seguida, cada link na primeira página é iterado e uma verificação é feita para ver se ele está no conjunto global de páginas (um conjunto de páginas que o script

já encontrou). Caso contrário, ele é adicionado à lista, impresso na tela e o `getLinks` função é chamada recursivamente nele.



AVISO Sobre Recursão

Este é um aviso raramente visto em livros de software, mas achei que você deveria estar ciente: se deixado em execução por tempo suficiente, o programa anterior quase certamente travará.

Python tem um limite de recursão padrão (quantas vezes os programas podem se chamar recursivamente) de 1.000. Como a rede de links da Wikipedia é extremamente grande, este programa acabará atingindo o limite de recursão e parará, a menos que você coloque um contador de recursão ou algo para evitar que isso aconteça.

Para sites "planos" com menos de 1.000 links de profundidade, esse método geralmente funciona muito bem, com algumas exceções incomuns. Por exemplo, certa vez encontrei um site que tinha uma regra para gerar links internos para postagens de blog. A regra era "pegue a URL da página atual em que estamos e anexe /blog/title_of_blog.php para isso."

O problema era que eles acrescentariam /blog/title_of_blog.php para URLs que foram já em uma página que tinha /blog/no URL. Assim, o site simplesmente adicionaria outro /blog/ em. Eventualmente, meu rastreador foi

```
indo      para      URLs      gostar:      /blog/blog/blog/blog.../blog/  
title_of_blog.php.
```

Eventualmente, eu tive que adicionar uma verificação para ter certeza de que os URLs não eram excessivamente ridículos, contendo segmentos repetidos que podem indicar um loop infinito. No entanto, se eu tivesse deixado essa execução desmarcada durante a noite, poderia facilmente travar.

Coletando dados em um site inteiro

É claro que os rastreadores da web seriam bastante enfadonhos se tudo que fizessem fosse pular de uma página para a outra. Para torná-los úteis, precisamos ser capazes de fazer algo na página enquanto estamos lá. Vejamos como construir um raspador que reúna o título, o primeiro parágrafo do conteúdo e o link para editar a página (se disponível).

Como sempre, a primeira etapa para determinar a melhor maneira de fazer isso é examinar algumas páginas do site e determinar um padrão. Ao olhar para um punhado de páginas da Wikipedia, tanto artigos como páginas que não são artigos, como a página da política de privacidade, as seguintes coisas devem ficar claras:

- Todos os títulos (em todas as páginas, independentemente de seu status como uma página de artigo, uma página de histórico de edição ou qualquer outra página) têm títulos sob `h1` → período tags, e essas são as únicas

`h1` tags na página.

- Como mencionado antes, todo o texto do corpo está sob o div # bodyContent tag. No entanto, se quisermos ser mais específicos e acessar apenas o primeiro parágrafo do texto, talvez seja melhor usar div # mw-content-text → p (selecionando apenas a tag do primeiro parágrafo). Isso é verdadeiro para todas as páginas de conteúdo, exceto páginas de arquivo (por exemplo: https://en.wikipedia.org/wiki/File:Orbit_of_274301_Wikipedia.svg), que não possuem seções de texto de conteúdo.
- Os links de edição ocorrem apenas nas páginas do artigo. Se ocorrerem, serão encontrados no li # ca-edit tag, sob li # ca-edit → período → uma.

Ao modificar nosso código de rastreamento básico, podemos criar um programa de combinação de rastreador / coleta de dados (ou, pelo menos, impressão de dados):

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar ré

Páginas = conjunto ()
def getLinks ( URL da página ):
    global Páginas
    html = urlopen ( "http://en.wikipedia.org" + URL da página )
    bsObj = BeautifulSoup ( html )
    experimentar :
        impressão ( bsObj . h1 . get_text () )
        impressão ( bsObj . encontrar ( Eu iria = "mw-content-text" ) . encontrar tudo ( "p" ) [ 0 ] )
        impressão ( bsObj . encontrar ( Eu iria = "ca-edit" ) . encontrar ( "periódico" ) . encontrar ( "uma" ) . atrs [ 'href' ] )
    exceto AttributeError :
        impressão ( " Esta página está faltando alguma coisa! Mas não se preocupe! " )

    para ligação dentro bsObj . encontrar tudo ( "uma" , href = ré . compilar ( "^ (/ wiki /)" )):
        E se ' href ' dentro ligação . atrs :
            E se ligação . atrs [ 'href' ] não em Páginas :
                # Encontramos uma nova página
                nova página = ligação . atrs [ 'href' ]
                impressão ( "-----\n" + nova página )
                Páginas . adicionar ( nova página )
                getLinks ( nova página )

    getLinks ( "" )
```

O loop neste programa é essencialmente o mesmo que no programa de rastreamento original (com a adição de alguns travessões impressos para maior clareza, separando o conteúdo impresso).

Como nunca podemos ter certeza de que todos os dados estão em cada página, cada declaração impressa é organizada na ordem em que é mais provável que apareça no site. Ou seja, o <h1> a tag de título aparece em todas as páginas (pelo que posso dizer, pelo menos), portanto, tentamos obter esses dados primeiro. O conteúdo do texto aparece na maioria das páginas (exceto para páginas de arquivo), de modo que é a segunda parte dos dados recuperados. O botão “editar” só aparece nas páginas onde já existem títulos e conteúdo de texto, mas não aparece em todas essas páginas.



Diferentes padrões para diferentes necessidades

Obviamente, existem alguns perigos envolvidos em agrupar várias linhas em um manipulador de exceções. Você não pode dizer qual linha gerou a exceção, para começar. Além disso, se por algum motivo uma página contivesse um botão “editar”, mas nenhum título, o botão “editar” nunca seria registrado. No entanto, é suficiente para muitos casos em que há uma ordem de probabilidade de itens aparecerem no site e, inadvertidamente, perder alguns pontos de dados ou manter registros detalhados não é um problema.

Você pode notar que neste e em todos os exemplos anteriores, não estivemos “coletando” dados, mas sim “imprimindo-os”. Obviamente, os dados em seu terminal são difíceis de manipular. Veremos mais como armazenar informações e criar bancos de dados em [capítulo 5](#).

Rastreando pela Internet

Sempre que dou uma palestra sobre web scraping, alguém inevitavelmente pergunta: “Como você constrói o Google?” Minha resposta é sempre dupla: “Primeiro, você recebe muitos bilhões de dólares para poder comprar os maiores data warehouses do mundo e colocá-los em locais ocultos em todo o mundo. Em segundo lugar, você constrói um rastreador da web.”

Quando o Google começou em 1994, eram apenas dois alunos de graduação de Stanford com um servidor antigo e um rastreador da web em Python. Agora que você sabe disso, oficialmente tem as ferramentas de que precisa para se tornar o próximo multimilionário em tecnologia!

Com toda a seriedade, os rastreadores da web estão no centro do que impulsiona muitas tecnologias modernas da web, e você não precisa necessariamente de um grande data warehouse para usá-los. Para fazer qualquer análise de dados entre domínios, você precisa criar rastreadores que possam interpretar e armazenar dados em uma infinidade de páginas diferentes na Internet.

Assim como no exemplo anterior, os rastreadores da web que iremos construir seguirão links de uma página para outra, construindo um mapa da web. Mas desta vez, eles não irão ignorar os links externos; eles irão segui-los. Como um desafio extra, veremos se podemos registrar algum tipo de informação sobre cada página à medida que a percorremos. Isso será mais difícil do que trabalhar com um único domínio, como fazíamos antes - diferentes sites têm layouts completamente diferentes. Isso significa que teremos que ser muito flexíveis no tipo de informação que procuramos e como procuramos.



Desconhecido Águas à Frente

Lembre-se de que o código da próxima seção pode ir *qualquer lugar* na internet. Se aprendemos alguma coisa com "Seis graus da Wikipedia", é que é perfeitamente possível ir de um site como <http://www.sesamestreet.org/> para algo menos saboroso em apenas alguns saltos.

Crianças, perguntam a seus pais antes de executar este código. Para aqueles com constituições sensíveis ou com restrições religiosas que podem proibir a visualização de textos de um site lascivo, leia os exemplos de código, mas tenha cuidado ao executá-los.

Antes de começar a escrever um rastreador que simplesmente segue todos os links externos, queira ou não, você deve se fazer algumas perguntas:

- Que dados estou tentando coletar? Isso pode ser feito raspando apenas alguns sites predefinidos (quase sempre a opção mais fácil) ou meu rastreador precisa ser capaz de descobrir novos sites que eu talvez não conheça?
- Quando meu rastreador chegar a um determinado site da Web, ele seguirá imediatamente o próximo link externo para um novo site ou ficará por um tempo e detalhará o site atual?
- Há alguma condição sob a qual eu não gostaria de raspar um determinado site? Estou interessado em conteúdo que não seja em inglês?
- Como estou me protegendo contra ações judiciais se meu rastreador da web chama a atenção de um webmaster em um dos sites que encontra? (Verificação de saída **Apêndice C** para obter mais informações sobre este assunto.)

Um conjunto flexível de funções Python que podem ser combinadas para realizar uma variedade de tipos diferentes de web scraping pode ser facilmente escrito em menos de 50 linhas de código:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar ré
importar data hora
importar aleatória

Páginas = conjunto ()
aleatória . semente ( data hora . data hora . agora ())

# Recupera uma lista de todos os links internos encontrados em uma página
def getInternalLinks ( bsObj , includeUrl ):
    internalLinks = []
    # Encontra todos os links que começam com "/"
    para ligação dentro bsObj . encontrar tudo ( "uma" , href = ré . compilar ( "^ ( / | . * " + includeUrl + ")") ):
        E se ligação . attrs [ 'href' ] não é Nenhum :
            E se ligação . attrs [ 'href' ] não em internalLinks :
                internalLinks . acrescentar ( ligação . attrs [ 'href' ])
    Retorna internalLinks
```

```

# Recupera uma lista de todos os links externos encontrados em uma página
def getExternalLinks ( bsObj , excludeUrl ):
    links externos = []
    # Encontra todos os links que começam com "http" ou "www" que fazem
    # não contém o URL atual
    para ligação dentro bsObj . encontrar tudo ( "uma" ,
        href = ré . compilar ( "^ (http | www) ((?!" + excludeUrl + ".)* $)" )):
        E se ligação . attrs [ 'href' ] não é Nenhum :
            E se ligação . attrs [ 'href' ] não em links externos :
                links externos . acrescentar ( ligação . attrs [ 'href' ] )
    Retorna links externos

def splitAddress ( endereço ):
    addressParts = endereço . substituir ( "http: //", "" ) . Dividido ( "/" )
    Retorna addressParts

def getRandomExternalLink ( página inicial ):
    html = urlopen ( página inicial )
    bsObj = BeautifulSoup ( html )
    links externos = getExternalLinks ( bsObj , splitAddress ( página inicial ) [ 0 ] )
    E se len ( links externos ) == 0 :
        internalLinks = getInternalLinks ( página inicial )
        Retorna getNextExternalLink ( internalLinks [ aleatória . Randint ( 0 ,
            len ( internalLinks ) - 1 )])
    outro :
        Retorna links externos [ aleatória . Randint ( 0 , len ( links externos ) - 1 )]

def followExternalOnly ( começandoSite ):
    link externo = getRandomExternalLink ( "http://oreilly.com" )
    impressão ( " O link externo aleatório é: " + link externo )
    followExternalOnly ( link externo )

    followExternalOnly ( "http://oreilly.com" )

```

O programa anterior começa em <http://oreilly.com> e pula aleatoriamente de link externo para link externo. Aqui está um exemplo da saída que ele produz:

```

O link externo aleatório é: http://igniteshow.com/
O link externo aleatório é: http://feeds.feedburner.com/oreilly/news
O link externo aleatório é: http://hire.jobvite.com/CompanyJobs/Careers.aspx?c=q319 O link externo aleatório é:
http://makerfaire.com/

```

Links externos nem sempre são garantidos para serem encontrados na primeira página de um site. Para encontrar links externos, neste caso, um método semelhante ao usado no exemplo de rastreamento anterior é empregado para pesquisar recursivamente em um site até encontrar um link externo.

[Figura 3-1](#) visualiza a operação como um fluxograma:

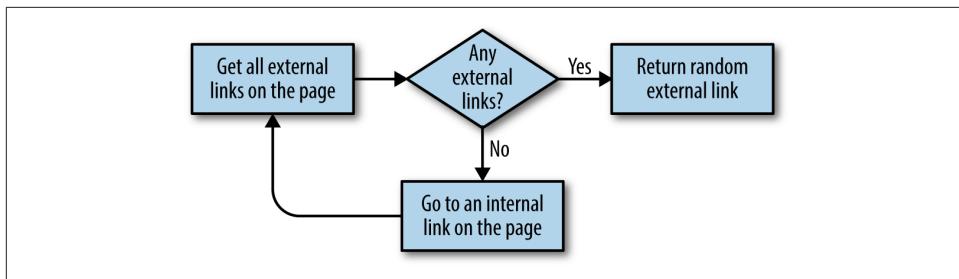


Figura 3-1. Fluxograma para script que rastreia diferentes sites na Internet



Não coloque programas de exemplo em produção

Continuo trazendo isso à tona, mas é importante para espaço e legibilidade, os programas de exemplo neste livro nem sempre contêm as verificações necessárias e o tratamento de exceções exigidos para o código pronto para produção.

Por exemplo, se um link externo não for encontrado em qualquer lugar em um site que este rastreador encontra (improvável, mas é provável que aconteça em algum ponto se você executá-lo por tempo suficiente), este programa continuará executando até atingir a recursão do Python limite.

Antes de executar este código para qualquer propósito sério, certifique-se de colocar verificações no local para lidar com armadilhas potenciais.

O bom de dividir as tarefas em funções simples, como “localizar todos os links externos nesta página”, é que o código pode ser facilmente refatorado posteriormente para executar uma tarefa de rastreamento diferente. Por exemplo, se nosso objetivo é rastrear um site inteiro em busca de links externos e tomar nota de cada um, podemos adicionar a seguinte função:

```

# Coleta uma lista de todos os URLs externos encontrados no site
allExtLinks = conjunto()
allIntLinks = conjunto()

def getAllExternalLinks( URL do site ):
    html = urlopen( URL do site )
    bsObj = BeautifulSoup( html )
    internalLinks = getInternalLinks( bsObj , splitAddress( URL do site ) [ 0 ] )
    links externos = getExternalLinks( bsObj , splitAddress( URL do site ) [ 0 ] )

    para ligação dentro links externos :
        E se ligação não em allExtLinks :
            allExtLinks . adicionar( ligação )
            impressão( ligação )

    para ligação dentro internalLinks :
        E se ligação não em allIntLinks :
            impressão( " Prestes a obter o link: " + ligação )
            allIntLinks . adicionar( ligação )
            getAllExternalLinks( ligação )

```

```
getAllExternalLinks ( "http://oreilly.com" )
```

Esse código pode ser considerado como dois loops - um reunindo links internos, outro reunindo links externos - trabalhando em conjunto um com o outro. O fluxograma parece algo como [Figura 3-2](#) :

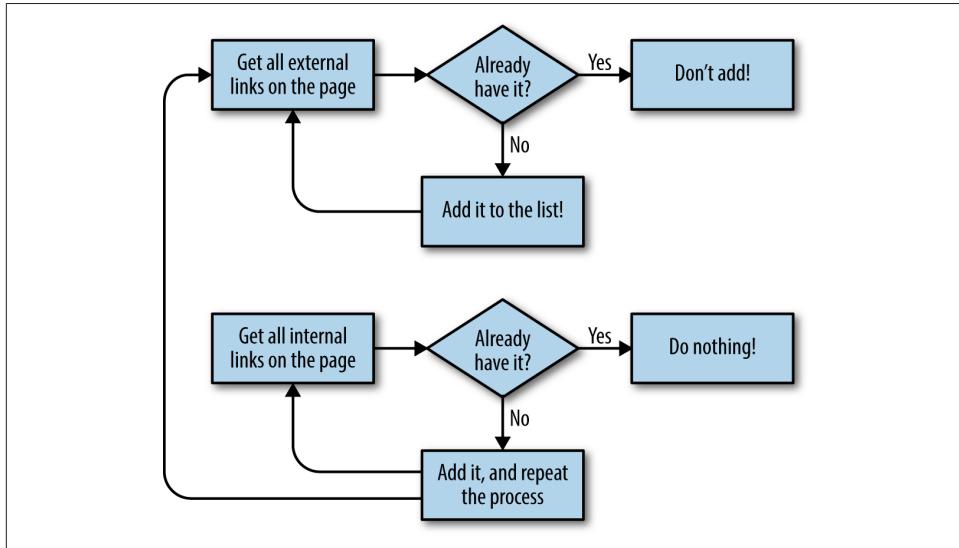


Figura 3-2. Diagrama de fluxo para o rastreador de site que coleta todos os links externos

Anotar ou fazer diagramas do que o código deve fazer antes de você escrever o código em si é um hábito fantástico que pode ser adquirido e que pode economizar muito tempo e frustração à medida que seus rastreadores ficam mais complicados.

Tratamento de redirecionamentos

Os redirecionamentos permitem que a mesma página da web seja visualizada em diferentes nomes de domínio. Os redirecionamentos vêm em dois sabores:

- Redirecionamentos do lado do servidor, onde o URL é alterado antes que a página seja carregada
- Redirecionamentos do lado do cliente, às vezes vistos com um tipo de mensagem "Você será direcionado em 10 segundos ...", em que a página é carregada antes de redirecionar para a nova.

Esta seção tratará dos redirecionamentos do lado do servidor. Para obter mais informações sobre redirecionamentos do lado do cliente, que são executados usando JavaScript ou HTML, consulte [Capítulo 10](#).

Para redirecionamentos do lado do servidor, você geralmente não precisa se preocupar. Se você estiver usando o `urllib` biblioteca com Python 3.x, ela lida com redirecionamentos automaticamente! Esteja ciente de que, ocasionalmente, o URL da página que você está rastreando pode não ser exatamente o URL em que você inseriu a página.

Rastejando com Scrapy

Um dos desafios de escrever rastreadores da web é que muitas vezes você está executando as mesmas tarefas repetidamente: encontrar todos os links em uma página, avaliar a diferença entre links internos e externos, ir para novas páginas. É útil conhecer esses padrões básicos e ser capaz de escrever do zero, mas existem opções se você quiser outra coisa para lidar com os detalhes para você.

Scrapy é uma biblioteca Python que lida com grande parte da complexidade de encontrar e avaliar links em um site, rastreando domínios ou listas de domínios com facilidade. Infelizmente, o Scrapy ainda não foi lançado para o Python 3.x, embora seja compatível com o Python 2.7.

A boa notícia é que várias versões do Python (por exemplo, Python 2.7 e 3.4) geralmente funcionam bem quando instaladas na mesma máquina. Se você quiser usar o Scrapy para um projeto, mas também quiser usar vários outros scripts Python 3.4, não deverá ter problemas para fazer os dois.

O site Scrapy oferece a ferramenta para [baixar](#) de seu site, bem como instruções para instalar o Scrapy com gerenciadores de instalação de terceiros, como pip. Lembre-se de que você precisará instalar o Scrapy usando Python 2.7 (não é compatível com 2.6 ou 3.x) e execute todos os programas usando Scrapy com Python 2.7 também.

Embora escrever rastreadores Scrapy seja relativamente fácil, há uma pequena configuração que precisa ser feita para cada rastreador. Para criar um novo projeto Scrapy no diretório atual, execute a partir da linha de comando:

```
$ scrapy startproject wikiSpider
```

`wikiSpider` é o nome do nosso novo projeto. Isso cria um novo diretório no diretório em que o projeto foi criado, com o título `wikiSpider`. Dentro desse diretório está a seguinte estrutura de arquivos:

- `scrapy.cfg`
- `wikiSpider`
 - `__init__.py`
 - `items.py`
 - `pipelines.py`
 - `settings.py`
 - `aranhas`

- __init.py__

Para criar um rastreador, adicionaremos um novo arquivo ao *wikiSpider / wikiSpider / spiders / articleSpider.py* chamado *items.py*. Além disso, vamos definir um novo item chamado Artigo dentro de *items.py* Arquivo.

Seu *items.py* o arquivo deve ser editado para ficar assim (com comentários gerados pelo Scrapy deixados no local, embora você possa se sentir à vontade para removê-los):

```
# -*- codificação: utf-8 -*-
# Defina aqui os modelos para seus itens raspados
#
# Veja a documentação em:
# http://doc.scrapy.org/en/latest/topics/items.html

de raspagem importar Item , Campo

classe Artigo ( Item ):
    # defina os campos para seu item aqui como:
    # name = scrapy.Field ()
    título = Campo ()
```

Cada Scrapy Item objeto representa uma única página no site. Obviamente, você pode definir quantos campos desejar (url, conteúdo, imagem de cabeçalho, etc.), mas estou simplesmente coletando o título campo de cada página, por enquanto.

Em seu recém-criado *articleSpider.py* arquivo, escreva o seguinte:

```
de scrapy.selector importar Seletor
de raspagem importar Aranha
de wikiSpider.items importar Artigo

classe ArticleSpider ( Aranha ):
    nome = "artigo"
    permission_domains = [ "en.wikipedia.org" ]
    start_urls = [ "http://en.wikipedia.org/wiki/Main_Page",
                   "http://en.wikipedia.org/wiki/Python_%28programming_language%29" ]

    def analisar ( auto , resposta ):
        item = Artigo ()
        título = resposta . xpath ( '// h1 / text ()' ) [ 0 ] . extrair ()
        impressão ( " O título é: " + título )
        item [ 'título' ] = título
        Retorna item
```

O nome deste objeto (ArticleSpider) é diferente do nome do diretório (*WikiSpider*), indicando que esta classe em particular é responsável por spidering apenas através de páginas de artigos, na categoria mais ampla de WikiSpider. Para sites grandes com muitos tipos de conteúdo, você pode ter itens Scrapy separados para cada tipo (blog

posts, comunicados à imprensa, artigos, etc.), cada um com campos diferentes, mas todos rodando no mesmo projeto Scrapy.

Você pode executar isto ArticleSpider de dentro do principal *WikiSpider* diretório digitando:

```
$ artigo scrapy crawl
```

Isso chama o raspador pelo nome do item de artigo (não a classe ou o nome do arquivo, mas o nome definido na linha: nome = "artigo" no ArticleSpider).

Junto com algumas informações de depuração, isso deve imprimir as linhas:

```
O título é: Página Principal
```

```
O título é: Python (linguagem de programação)
```

O raspador vai para as duas páginas listadas como start_urls, reúne informações e, em seguida, termina. Não é muito um rastreador, mas usar o Scrapy dessa forma pode ser útil se você tiver uma lista de URLs que precisa copiar. Para transformá-lo em um rastreador totalmente desenvolvido, você precisa definir um conjunto de regras que Scrapy pode usar para buscar novos URLs em cada página que encontrar:

```
de scrapy.contrib.spiders importar CrawlSpider , Regra
de wikiSpider.items importar Artigo
de scrapy.contrib.linkextractors.sgml importar SgmlLinkExtractor

classe ArticleSpider ( CrawlSpider ):
    nome = "artigo"
    permission_domains = [ "en.wikipedia.org" ]
    start_urls = [ "http://en.wikipedia.org/wiki/Python_
                    % 28 linguagem de programação % 29 "]
    regras = [ Regra ( SgmlLinkExtractor ( permitir = ( '(wiki)((?!:).)*$' ),),
                      ligue de volta = "parse_item" , Segue = Verdadeiro )]

    def parse_item ( auto , resposta ):
        item = Artigo ()
        título = resposta . xpath ( '// h1 / text ()' )[ 0 ] . extrair ()
        impressão ( " O título é: " + título )
        item [ 'título' ] = título
        Retorna item
```

Este rastreador é executado a partir da linha de comando da mesma forma que o anterior, mas não terminará (pelo menos não por muito, muito tempo) até que você interrompa a execução usando Ctrl + C ou fechando o terminal.



Registrando com Scrapy

As informações de depuração geradas pelo Scrapy podem ser úteis, mas geralmente são muito detalhadas. Você pode ajustar facilmente o nível de registro adicionando uma linha ao `settings.py` arquivo em seu projeto Scrapy:

```
LOG_LEVEL = 'ERROR'
```

Existem cinco níveis de registro no Scrapy, listados em ordem aqui:

- CRÍTICO
- ERRO
- ATENÇÃO
- DEPURAR
- INFO

Se o registro estiver definido para ERRO, só CRÍTICO e ERRO os logs serão exibidos. Se o registro estiver definido para INFO, todos os registros serão exibidos e assim por diante.

Para enviar registros para um arquivo de registro separado em vez do terminal, basta definir um arquivo de registro ao executar a partir da linha de comando:

```
$ artigo scrapy crawl -s ARQUIVO DE LOG = wiki.log
```

Isso criará um novo arquivo de log, se não existir, em seu diretório atual e produzirá todos os logs e instruções de impressão para ele.

Scrapy usa o Item objetos para determinar quais informações ele deve salvar das páginas que visita. Essas informações podem ser salvas pelo Scrapy de várias maneiras, como arquivos CSV, JSON ou XML, usando os seguintes comandos:

```
$ scrapy crawl article -o articles.csv -t csv  
$ scrapy crawl article -o articles.json -t json  
$ artigo scrapy crawl -o articles.xml -t xml
```

Claro, você pode usar o Item objetos você mesmo e gravá-los em um arquivo ou banco de dados da maneira que desejar, simplesmente adicionando o código apropriado à função de análise no rastreador.

O Scrapy é uma ferramenta poderosa que lida com muitos problemas associados ao rastreamento da web. Ele reúne automaticamente todos os URLs e os compara com regras predefinidas; certifica-se de que todos os URLs são exclusivos; normaliza URLs relativos onde necessário; e recorre para ir mais profundamente nas páginas.

Embora esta seção dificilmente arranhe a superfície do que o Scrapy é capaz, eu encorajo você a verificar a [documentação Scrapy](#) ou muitos dos outros recursos disponíveis online. Scrapy é uma biblioteca extremamente grande e extensa, com muitos recursos. Se houver algo que você gostaria de fazer com o Scrapy que não foi mencionado aqui, é provável que haja uma maneira (ou várias) de fazer isso!

CAPÍTULO 4

Usando APIs

Como muitos programadores que trabalharam em grandes projetos, tenho minha cota de histórias horíveis quando se trata de trabalhar com o código de outras pessoas. De problemas de espaço de nomes a problemas de tipo e mal-entendidos sobre a saída da função, simplesmente tentar obter informações do ponto A ao método B pode ser um pesadelo.

É aqui que as interfaces de programação de aplicativos são úteis: elas fornecem interfaces agradáveis e convenientes entre vários aplicativos distintos. Não importa se os aplicativos são escritos por programadores diferentes, com arquiteturas diferentes ou mesmo em linguagens diferentes - as APIs são projetadas para servir como uma língua franca entre diferentes partes de software que precisam compartilhar informações entre si.

Embora existam várias APIs para uma variedade de aplicativos de software diferentes, nos últimos tempos, "API" tem sido comumente entendida como significando "API de aplicativo da web". Tipicamente, um programador fará uma solicitação a uma API via HTTP para algum tipo de dados, e a API retornará esses dados na forma de XML ou JSON. Embora a maioria das APIs ainda suporte XML, JSON está rapidamente se tornando o protocolo de codificação de escolha.

Se tirar proveito de um programa pronto para uso para obter informações pré-empacotadas em um formato útil parece um pouco diferente do resto deste livro, bem, é e não é. Embora o uso de APIs não seja geralmente considerado web scraping pela maioria das pessoas, ambas as práticas usam muitas das mesmas técnicas (envio de solicitações HTTP) e produzem resultados semelhantes (obtenção de informações); eles freqüentemente podem ser muito complementares um ao outro.

Por exemplo, você pode querer combinar as informações coletadas de um web scraper com as informações de uma API publicada para torná-las mais úteis para você. Em um exemplo mais adiante neste capítulo, veremos a combinação de históricos de edição da Wikipedia (que contêm endereços IP) com uma API de resolução de endereço IP para obter a localização geográfica das edições da Wikipedia em todo o mundo.

Neste capítulo, vamos oferecer uma visão geral das APIs e como elas funcionam, dar uma olhada em algumas APIs populares disponíveis hoje e ver como você pode usar uma API em seus próprios web scrapers.

Como funcionam as APIs

Embora as APIs não sejam tão onipresentes quanto deveriam ser (uma grande motivação para escrever este livro, porque se você não conseguir encontrar uma API, ainda poderá obter os dados por meio de extração), você pode encontrar APIs para muitos tipos de informações. Interessado em música? Existem algumas APIs diferentes que podem fornecer músicas, artistas, álbuns e até mesmo informações sobre estilos musicais e artistas relacionados. Precisa de dados esportivos? A ESPN fornece APIs para informações de atletas, resultados de jogos e muito mais. O Google tem dezenas de APIs em seu

Seção de desenvolvedores para traduções de idiomas, análises, geolocalização e muito mais.

APIs são extremamente fáceis de usar. Na verdade, você pode experimentar uma simples solicitação de API apenas digitando o seguinte em seu navegador:¹

```
http://freegeoip.net/json/50.78.253.58
```

Isso deve produzir a seguinte resposta:

```
{"ip": "50.78.253.58", "Código do país": "NOS", "nome do país": "Estados Unidos", "Código Regional": "MA", "region_name": "Massachusetts", "cidade": "Chelmsford", "Código postal": "01824", "latitude": 42,5879, "longitude": -71,3498, "metro_code": "506", "Código de área": "978"}
```

Então, espere, você navega para um endereço da web na janela do seu navegador e ele produz algumas informações (embora, muito bem formatadas)? Qual é a diferença entre uma API e um site normal? Apesar do exagero em torno das APIs, a resposta geralmente é: não muito. As APIs funcionam via HTTP, o mesmo protocolo usado para buscar dados para sites, baixar um arquivo e fazer quase tudo na Internet. As únicas coisas que tornam uma API uma API é a sintaxe extremamente regulamentada que ela usa e o fato de que as APIs apresentam seus dados como JSON ou XML, em vez de HTML.

Convenções Comuns

Ao contrário dos assuntos da maioria dos web scraping, as APIs seguem um conjunto extremamente padronizado de regras para produzir informações e também produzem essas informações de uma forma extremamente padronizada. Por causa disso, é fácil aprender algumas regras básicas simples que o ajudarão a começar a trabalhar rapidamente com qualquer API, contanto que seja bem escrita.

¹ Esta API resolve endereços IP para localizações geográficas e é uma que usarei posteriormente neste capítulo também. Você pode aprender mais sobre isso em <http://freegeoip.net>.

Dito isso, lembre-se de que algumas APIs se desviam um pouco dessas regras, por isso é importante ler a documentação de uma API na primeira vez que usá-la, independentemente de quão familiarizado você esteja com as APIs em geral.

Métodos

Existem quatro maneiras de solicitar informações de um servidor da web via HTTP:

- PEGUE
- POSTAR
- COLOCAR
- EXCLUIR

PEGUE é o que você usa quando visita um site por meio da barra de endereço do navegador.

PEGUE é o método que você está usando quando faz uma chamada para <http://freegeoip.net/json/>

50.78.253.58. Você pode pensar em PEGUE como dizendo: "Ei, servidor da web, por favor, me dê essas informações."

POSTAR é o que você usa quando preenche um formulário ou envia informações, provavelmente para um script de back-end no servidor. Cada vez que você entra em um site, você está fazendo um

POSTAR pedido com seu nome de usuário e (espero) senha criptografada. Se você está fazendo um POSTAR solicitação com uma API, você está dizendo: "Por favor, armazene essas informações em seu banco de dados".

COLOCAR é menos comumente usado ao interagir com sites, mas é usado de vez em quando em APIs. UMA COLOCAR pedido é usado para atualizar um objeto ou informação. Uma API pode exigir um POSTAR solicitação para criar um novo usuário, por exemplo, mas pode precisar de um COLOCAR solicitar se você deseja atualizar o endereço de e-mail desse usuário.²

EXCLUIR é simples; é usado para excluir um objeto. Por exemplo, se eu enviar um EXCLUIR pedido para <http://myapi.com/user/23>, ele excluirá o usuário com o ID 23.

EXCLUIR métodos não são encontrados com frequência em APIs públicas, que são principalmente criadas para disseminar informações, em vez de permitir que usuários aleatórios removam essas informações de seus bancos de dados. No entanto, como o COLOCAR método, é bom conhecer.

Embora vários outros métodos HTTP sejam definidos sob as especificações para HTTP, esses quatro constituem a totalidade do que é usado em praticamente qualquer API que você possa encontrar.

² Na realidade, muitas APIs usam POSTAR pedidos em vez de COLOCAR solicitações ao atualizar informações. O fato de uma nova entidade ser criada ou uma antiga simplesmente ser atualizada depende de como a própria solicitação de API é estruturada. No entanto, ainda é bom saber a diferença e, muitas vezes, você encontrará COLOCAR solicitações em APIs comumente usadas.

Autenticação

Embora algumas APIs não usem nenhuma autenticação para operar (o que significa que qualquer pessoa pode fazer uma chamada de API gratuitamente, sem se registrar primeiro no aplicativo), muitas APIs modernas exigem algum tipo de autenticação antes de poderem ser usadas.

Algumas APIs exigem autenticação para cobrar dinheiro por chamada de API ou podem oferecer seu serviço em algum tipo de assinatura mensal. Outros se autenticam para “limitar a taxa” de usuários (restringi-los a um certo número de chamadas por segundo, hora ou dia) ou para restringir o acesso a certos tipos de informações ou tipos de chamadas de API para alguns usuários. Outras APIs podem não colocar restrições, mas podem querer controlar quais usuários estão fazendo chamadas para fins de marketing.

Todos os métodos de autenticação de API geralmente giram em torno do uso de um *símbolo* de algum tipo, que é passado para o servidor da web com cada chamada de API feita. Esse token é fornecido ao usuário quando o usuário se registra e é um acessório permanente das chamadas do usuário (geralmente em aplicativos de baixa segurança) ou pode mudar com frequência e é recuperado do servidor usando uma combinação de nome de usuário e senha.

Por exemplo, para fazer uma chamada para a API Echo Nest a fim de recuperar uma lista de músicas da banda Guns N' Roses, usaríamos:

```
http://developer.echonest.com/api/v4/artist/songs?api_key=<your api key here>%20&name=guns%20n%27%20roses&format=json&start=0&results=100
```

Isso fornece ao servidor um Chave API valor do que foi fornecido para mim no registro, permitindo que o servidor identifique o solicitante como Ryan Mitchell e forneça ao solicitante os dados JSON.

Além de passar tokens no URL da própria solicitação, os tokens também podem ser passados para o servidor por meio de um cookie no cabeçalho da solicitação. Discutiremos os cabeçalhos com mais detalhes posteriormente neste capítulo, bem como em [Capítulo 12](#), mas a título de exemplo, eles podem ser enviados usando o `urllib` pacote dos capítulos anteriores:

```
símbolo = "<sua chave de api>"  
webRequest = urllib.request.Request("http://myapi.com", headers = { "símbolo" : símbolo })  
html = urlopen ( webRequest )
```

Respostas

Como você viu no exemplo do FreeGeoIP no início do capítulo, um recurso importante das APIs é que elas têm respostas bem formatadas. Os tipos mais comuns de formatação de resposta são *Extensible Markup Language* (XML) e *JavaScript Object Notation* (JSON).

Nos últimos anos, o JSON se tornou muito mais popular do que o XML por alguns motivos principais. Primeiro, os arquivos JSON geralmente são menores do que os arquivos XML bem projetados. Compare, por exemplo, os dados XML:

```
<usuário> <primeiro> Ryan </ firstname> <lastname> Mitchell </ sobrenome> <nome de usuário> Kludgist  
<username> </user>
```

que registra 98 caracteres e os mesmos dados em JSON:

```
{ "do utilizador": { "primeiro nome": "Ryan", "último nome": "Mitchell", "nome do usuário": "Kludgist" }}
```

que tem apenas 73 caracteres, ou espantosos 36% menor que o XML equivalente. Claro, pode-se argumentar que o XML pode ser formatado assim:

```
<usuário primeiro nome = "ryan" sobrenome = "mitchell" username = "Kludgist" > </ usuário>
```

mas isso é considerado uma prática ruim porque não oferece suporte a aninhamento profundo de dados. Independentemente disso, ele ainda requer 71 caracteres, aproximadamente o mesmo comprimento que o JSON equivalente.

Outro motivo pelo qual JSON está rapidamente se tornando mais popular que XML é simplesmente devido a uma mudança nas tecnologias da web. No passado, era mais comum que um script do lado do servidor, como PHP ou .NET, recebesse uma API. Hoje em dia, é provável que um framework, como Angular ou Backbone, esteja enviando e recebendo chamadas API. As tecnologias do lado do servidor são um tanto agnósticas quanto à forma como seus dados vêm. Mas bibliotecas JavaScript como o Backbone acham o JSON mais fácil de manusear.

Embora a maioria das APIs ainda suporte saída XML, usaremos exemplos JSON neste livro. Independentemente disso, é uma boa ideia familiarizar-se com os dois, se ainda não o fez - é improvável que eles desapareçam tão cedo.

Chamadas API

A sintaxe de uma chamada de API pode variar muito de API para API, mas existem algumas práticas padrão que elas costumam ter em comum. Ao recuperar dados por meio de um PEGUE solicitação, o caminho do URL descreve como você gostaria de detalhar os dados, enquanto os parâmetros de consulta servem como filtros ou solicitações adicionais anexadas à pesquisa.

Por exemplo, em uma API hipotética, você pode solicitar o seguinte para recuperar todas as postagens do usuário com o ID 1234 durante o mês de agosto de 2014:

```
http://socialmediasite.com/users/1234/posts?from=08012014&to=08312014
```

Muitas outras APIs usam o caminho para especificar uma versão da API, o formato em que você gostaria que seus dados e outros atributos. Por exemplo, o seguinte retornaria os mesmos dados, usando a versão 4 da API, formatada em JSON:

```
http://socialmediasite.com/api/v4/json/users/1234/posts?from=08012014&to=08312014
```

Outras APIs exigem que você passe as informações de formatação e versão da API como um parâmetro de solicitação, como:

```
http://socialmediasite.com/users/1234/posts?format=json&from=08012014&to=08312014
```

Eco Nest

O Echo Nest é um exemplo fantástico de empresa que se baseia em web scrapers. Embora algumas empresas de música, como a Pandora, dependam da intervenção humana para categorizar e anotar música, The Echo Nest depende de inteligência automatizada e informações extraídas de blogs e artigos de notícias para categorizar artistas musicais, músicas e álbuns.

Melhor ainda, esta API está disponível gratuitamente para uso não comercial.³ Você não pode usar a API sem uma chave, mas pode obter uma chave indo para [A página "Criar uma conta" do Echo Nest](#) e registrar-se com um nome, endereço de e-mail e nome de usuário.

Alguns exemplos

A API Echo Nest é construída em torno de vários tipos básicos de conteúdo: artistas, músicas, faixas e gêneros. Exceto por gêneros, todos esses tipos de conteúdo têm IDs exclusivos, que são usados para recuperar informações sobre eles de várias formas, por meio de chamadas de API. Por exemplo, se eu quisesse recuperar uma lista de músicas executadas por Monty Python, faria a seguinte chamada para recuperar seu ID (lembre-se de substituir < sua chave de API> com sua própria chave de API):

```
http://developer.echonest.com/api/v4/artist/search?api_key=<your api key> & name = monty%  
20python
```

Isso produz o seguinte resultado:

```
{"resposta": { "status": { "versão": "4.2", "código": 0, "mensagem": "Sucesso" }, "artistas": [ { "Eu iria": "AR5HF791187B9ABAF4", "nome": "Monty Python" }, { "Eu iria": "ARWCIDE13925F19A33", "nome": "SPAMALOT do Monty Python" }, { "Eu iria": "ARVPRCC12FE0862033", "nome": "Graham Chapman de Monty Python" } ] }}
```

Eu também poderia usar esse ID para consultar uma lista de músicas:

```
http://developer.echonest.com/api/v4/artist/songs?api_key=<your api key> & id = AR5HF791187B9ABAF4 &  
format = json & start = 0 & results = 10
```

Que fornece alguns dos sucessos do Monty Python, junto com gravações menos conhecidas:

```
{ "resposta": { "status": { "versão": "4.2", "código": 0, "mensagem": "Sucesso" },  
"começar": 0, "total": 476, "canções": [ { "Eu iria": "SORDAUE12AF72AC547", "título": "Neville Shunt" }, { "Eu iria": "SORBMPW13129A9174D", "título": "Clássico (Silbury Hill)" } ] }}
```

³ Veja [A página de licenciamento Echo Nest](#) detalhes dos requisitos de restrição.

```
(Parte 2}" }, { "Eu iria" : "SOQXAYQ1316771628E" , "título" : "Questionário de Pessoa Famosa (Remix do The Final Rip Off)" }, { "Eu iria" : "SOUAMYZ133EB4E17E8" , "título" : "Sempre olhe pelo lado bom da vida - Monty Python" } , ... ]})
```

Como alternativa, posso fazer uma única chamada usando o nome monty% 20python no lugar do ID exclusivo e recupere as mesmas informações:

```
http://developer.echonest.com/api/v4/artist/songs?api_key=<your api key> 2 & name = monty% 20python & format = json & start = 0 & results = 10
```

Usando esse mesmo ID, posso solicitar uma lista de artistas semelhantes:

```
http://developer.echonest.com/api/v4/artist/similar?api_key=<your api key> & id = AR5HF791187B9ABAF4 & format = json & results = 10 & start = 0
```

Os resultados incluem outros artistas de comédia, como Eric Idle, que estava no grupo Monty Python:

```
{"resposta": { "status": { "versão": "4.2" , "código": 0 , "mensagem": "Sucesso" } , "artistas": [ { "nome": "Vida de Brian" , "Eu iria": "ARNZYOS1272BA7FF38" } , { "nome": "Eric Idle" , "Eu iria": "ARELDIS1187B9ABC79" } , { "nome": "Os Simpsons" , "Eu iria": "ARNR4B91187FB5027C" } , { "nome": "Tom Lehrer" , "Eu iria": "ARJMYTZ1187FB54669" } , ... ]}}
```

Observe que, embora esta lista de artistas semelhantes contenha algumas informações genuinamente interessantes (por exemplo, “Tom Lehrer”), o primeiro resultado é “The Life of Brian”, da trilha sonora do filme que Monty Python lançou. Um dos perigos de usar um banco de dados selecionado de muitas fontes com intervenção humana mínima é que você pode obter alguns resultados ligeiramente estranhos. Isso é algo para se ter em mente ao criar seu próprio aplicativo usando dados de APIs de terceiros.

Cobri apenas alguns exemplos de como a API Echo Nest pode ser usada. Para verificar a documentação completa [Visão geral da API Echo Nest](#).

O Echo Nest patrocina muitos hackathons e projetos de programação focados na interseção de tecnologia e música. Se você precisa de inspiração, [A página de demonstração do Echo Nest](#) é um bom lugar para começar.

Twitter

O Twitter é notoriamente protetor de sua API e com razão. Com mais de 230 milhões de usuários ativos e uma receita de mais de US \$ 100 milhões por mês, a empresa hesita em permitir que qualquer pessoa apareça e tenha os dados que desejar.

Os limites de taxa do Twitter (o número de chamadas que cada usuário pode fazer) se enquadram em duas categorias: 15 chamadas por período de 15 minutos e 180 chamadas por período de 15 minutos, dependendo do tipo de chamada. Por exemplo, você pode fazer até 12 chamadas por minuto para recuperar

informações básicas sobre os usuários do Twitter, mas apenas uma ligação por minuto para recuperar listas de seguidores do Twitter desses usuários.⁴

Começando

Além de limitar a taxa, o Twitter tem um sistema de autorização mais complicado do que outras APIs, como The Echo Nest, tanto para obter chaves de API quanto para usar essas chaves. Para obter uma chave API, você precisará, obviamente, de uma conta no Twitter; você pode criar um relativamente sem dor em [a página de inscrição](#). Além disso, você precisará registrar um novo “aplicativo” no Twitter em seu [site do desenvolvedor](#).

Após este registro, você será levado a uma página que contém as informações básicas do seu aplicativo, incluindo a sua chave de consumidor ([Figura 4-1](#)):

The screenshot shows the 'Application Settings' page for a Twitter application. At the top, there's a header with a lock icon and the URL <https://apps.twitter.com/app/6995837>. Below the header, there's a table with two rows: 'Organization website' (None) and 'Consumer Key (API Key)' (t7BsdDEtbrm6RNMJFW7LrID3ZE). The consumer key is linked to 'manage keys and access tokens'. The page then transitions into the 'Application Actions' section, which contains a single button labeled 'Manage keys and access tokens'.

Organization website	None
Consumer Key (API Key)	t7BsdDEtbrm6RNMJFW7LrID3ZE (manage keys and access tokens)

Application Actions

Figura 4-1. A página de configurações do aplicativo do Twitter fornece informações básicas sobre o seu novo aplicativo

Se você clicar na página “gerenciar chaves e tokens de acesso”, você será direcionado para uma página contendo mais informações ([Figura 4-2](#)):

⁴ Para uma lista completa de limites de taxa, consulte <https://dev.twitter.com/rest/public/rate-limits>.

The screenshot shows the Twitter developer application settings page. At the top, the URL is https://apps.twitter.com/app/6995837/keys. The application name is "PythonScraping". Below the title, there are tabs for "Details", "Settings", "Keys and Access Tokens" (which is selected), and "Permissions".

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	t7BsdDEtbn6RNMJFW7LriD3ZE
Consumer Secret (API Secret)	VslvKcTTldFsPEUfkp3ZvxUjoVIBEBYRCuFdmq04CUZHHW2QsE
Access Level	Read-only (modify app permissions)
Owner	Kludgist
Owner ID	14983299

Application Actions

[Regenerate Consumer Key and Secret](#) [Change App Permissions](#)

Figura 4-2. Você precisará de sua chave secreta para usar a API do Twitter

Esta página também contém um botão para regenerar automaticamente suas chaves caso sua chave secreta de alguma forma se torne pública (por exemplo, se você acidentalmente publicá-la como uma ilustração em um livro técnico).

Alguns exemplos

O sistema de autenticação do Twitter, que usa OAuth, é bastante complicado; é melhor embrulhá-lo em uma das bibliotecas prontamente disponíveis, em vez de tentar controlá-lo sozinho. Devido à relativa complexidade de trabalhar com a API do Twitter “manualmente”, os exemplos desta seção se concentrarão no uso de código Python para interagir com a API, em vez de trabalhar com a própria API.

No momento em que este livro foi escrito, havia muitas bibliotecas Python 2.x usadas para interagir com o Twitter, mas relativamente poucas bibliotecas Python 3.x disponíveis. Felizmente, uma das melhores bibliotecas do Twitter (apropriadamente chamada de Twitter) está disponível para Python 3.x. Você pode baixá-lo de [a página Python Twitter Tools](#) e instale-o da maneira usual:

```
$ cd twitter-x.xx.x  
$ python setup.py install
```



Permissões de credencial do Twitter

Por padrão, os tokens de acesso do aplicativo recebem permissão somente leitura. Isso é bom para a maioria dos propósitos, a menos que você queira que seu aplicativo faça um tweet em seu nome.

Para alterar as permissões de seus tokens para leitura / gravação, vá para a guia Permissões no painel Gerenciamento de aplicativos do Twitter. Os tokens precisarão ser regenerados para que a atualização da permissão ocorra.

Da mesma forma, você pode atualizar as permissões de token para poder acessar mensagens diretas para sua conta do Twitter, se necessário para seu aplicativo. Tenha cuidado, entretanto; você deve dar tokens apenas as permissões de que eles absolutamente precisam. Em geral, é uma boa prática gerar vários conjuntos de tokens para vários aplicativos, em vez de reutilizar tokens muito poderosos para aplicativos que não precisam de todas as suas permissões.

Em nosso primeiro exercício, procuramos tweets específicos. O código a seguir se conecta à API do Twitter e imprime uma lista JSON de tweets contendo a hashtag # Pitão. Lembre-se de substituir as strings na linha OAuth por suas credenciais reais:

[de Twitter importar Twitter](#)

```
t = Twitter ( auth = OAuth ( < Token de acesso > , < Segredo do token de acesso > ,
                           < Chave do consumidor > , < consumidor secreto > ))
pythonTweets = t . pesquisa . tweets ( q = "#Pitão" )
impressão ( pythonTweets )
```

Embora a saída desse script possa ser opressora, lembre-se de que você está recebendo muitas informações por tweet: a data e hora em que o tweet foi feito, detalhes sobre quaisquer retuítes ou favoritos e detalhes sobre a conta do usuário e imagem de perfil . Embora você possa estar procurando apenas um subconjunto desses dados, a API do Twitter foi projetada para desenvolvedores da web que desejam exibir os tweets recebidos por meio da API em seus próprios sites, portanto, há muito material extra envolvido!

Você pode ver um exemplo de saída de um único tweet ao fazer uma atualização de status por meio da API:

[de Twitter importar *](#)

```
t = Twitter ( auth = OAuth ( < Token de acesso > , < Segredo do token de acesso > ,
                           < Chave do consumidor > , < consumidor secreto > ))
atualização de status = t . status . atualizar ( status = 'Olá Mundo!' )
impressão ( atualização de status )
```

Aqui está a descrição de JSON do tweet:

```
{ 'created_at': 'Sun Nov 30 07:23:39 +0000 2014', 'place': None, 'in_reply_to_screen_name': None, 'id_str': '538956506478428160', 'in_reply_to_user_id': None, 'lan-
```

```
g ' en ' in_reply_to_user_id_str ': Nenhum,' usuário ': { profile_sidebar_border_color ': '000000 ',  
profile_background_image_url ': ' http://pbs.twimg.com/profile_background_images / 497094351076347904 /  
RXn8MUID.png ',' descrição ': Software Engine er @ LinkeDrive, Estudante de mestrado @HarvardEXT, graduação da  
@OlinCollege, escritor @OReillyMedia. Realmente alto. Tem cabelo rosa. Feminino, apesar do nome. ' Time_zone ':  
Ea stern Time (EUA e Canadá) ', local ': Boston, MA ', lang ': ' en ', url ': ' http://t.co/FM6dHXIolw ', profile_location ':  
Nenhum,' name ': Ryan Mitchell ', screen_name ': Kludgist ', protected ': False,' default_profile_image ': False,' id_str ':  
14983299 ', ' favourites_count ': 140, ' contributors_enabled ': False, ' profile_use_background_image ': True,  
'profile_background_image_url_https ': ' https://pbs.twimg.  
mg.com/profile_background_images/497094351076347904/RXn8_fill_side_side.png' bar ': '889654', 'profile_link_color':  
'0021B3', 'default_profile ': False, 'statuses_count ': 3344, 'profile_background_color ': 'FFFFFF', 'profile_image_url':  
' http://pbs.twimg.com/profile_images/496692905335984125/XJh_d.jpeg ', 'profile_background_tile ': True,' id ':  
14983299, ' friends_count ': 409, ' profile_image_url_https ': ' https://pbs.twimg.com/profile_images/496692905335984128  
/ XJh_d5f5_normal.jpeg ', ' seguinte ': False, ' created_at ': 'Seg 02 de junho 18:35:1
```

```
8 +0000 2008 ', 'is_translator ': False, 'geo_enabled ': True, 'is_translation_enabled ': False, 'follow_request_sent ': False,  
'Seguidores_count ': 2085, 'utc_offset ': -18000, 'verificado ': Falso, 'profile_text_color ': '383838', 'notificações ': Falso,  
'entidades': {'descrição': {'urls': []}, 'url': {'urls': [{"índices": [  
0, 22], 'url': 'http://t.co/FM6dHXIolw', 'extended_url': 'http://ryanemitchell.com', 'display_url': 'ryanemitchell.com'}]}},  
'contagem_listada': 22, 'profile_banner_url': 'https://pbs.twimg.com/profile_banners/14983299/1412961553'}, 'retuitado ':  
False, 'in_reply_to_status_id_str ': Nenhum, 'source ': '<a href = "http://ryanemitchell.com" rel = "não siga" >  
PythonScraping </a>', 'favorite_count ': 0, 'texto': 'Inferno  
o, mundo!', 'truncado ': False, 'id ': 538956506478428160, 'retweet_count ': 0, 'favorited ': False, 'in_reply_to_status_id':  
Nenhum, 'geo': Nenhum, 'entidades': {'user_mentions': [], 'hashtags': [], 'urls': [], 'símbolos': []}, 'coordenadas': Nenhum,  
'contribuidores': Nenhum}
```

Sim, este é o resultado do envio de um único tweet. Às vezes acho que o Twitter limita o acesso à sua API por causa da largura de banda necessária para responder a cada solicitação!

Para qualquer solicitação que recupere uma lista de tweets, você pode limitar o número de tweets recebidos especificando uma contagem:

```
pythonStatuses = t . status . user_timeline ( Nome do ecrã = "Monty Python" , contagem = 5 )  
impressão ( pythonStatuses )
```

Nesse caso, estamos solicitando os últimos cinco tweets que foram postados na linha do tempo de @montypython (isso inclui quaisquer retuítes que eles possam ter feito).

Embora esses três exemplos (busca de tweets, recuperação de tweets de um usuário específico e postagem de seus próprios tweets) cubram muito do que a maioria das pessoas faz com a API do Twitter, os recursos da biblioteca Python do Twitter são muito mais numerosos. Você pode pesquisar e manipular listas do Twitter, seguir e deixar de seguir usuários, pesquisar informações de perfil de usuários e muito mais. A documentação completa pode ser encontrada em [GitHub](#).

APIs do Google

O Google tem uma das coleções de APIs mais abrangentes e fáceis de usar da web hoje. Sempre que você está lidando com algum tipo de assunto básico, como tradução de idiomas, geolocalização, calendários ou mesmo genômica, o Google tem uma API para isso. Google também tem APIs para muitos de seus aplicativos populares, como Gmail, YouTube e Blogger.

Existem duas páginas de referência principais para navegar nas APIs do Google. O primeiro é [a página de produtos](#), que serve como um repositório organizado de suas APIs, kits de desenvolvimento de software e outros projetos que podem ser de interesse dos desenvolvedores de software. O outro é

[o console de APIs](#), que fornece uma interface conveniente para ativar e desativar serviços de API, visualizar limites de taxa e uso em um piscar de olhos e até mesmo ativar uma instância de computação em nuvem com tecnologia do Google, se desejar.

A maioria das APIs do Google é gratuita, embora algumas, como a API de pesquisa, exijam uma licença paga. O Google é bastante liberal com sua coleção de APIs gratuitas, permitindo de 250 solicitações por dia a 20 milhões de solicitações por dia com uma conta básica. Também existe a opção de aumentar os limites da taxa em algumas das APIs, verificando sua identidade com um cartão de crédito (o cartão não é cobrado). Por exemplo, a API do Google Places tem um limite de taxa básica de 1.000 solicitações por período de 24 horas, mas isso pode ser aumentado para 150.000 solicitações verificando sua identidade. Para mais informações, veja [a página Limites de uso e cobrança](#).

Começando

Se você tem uma conta do Google, pode visualizar a lista de APIs disponíveis e criar uma chave de API usando o [Google Developers Console](#). Se você não tem uma conta do Google, é fácil se inscrever por meio do [Crie sua página da Conta do Google](#).

Depois de fazer login ou criar sua conta, você pode ver as credenciais de sua conta, incluindo chaves de API, no [Página do console da API](#); clique em "Credenciais" no menu esquerdo ([Figura 4-3](#)):

The screenshot shows the Google Developers Console interface. On the left, there's a sidebar with navigation links for Projects, API Project (Overview, Permissions, Billing & settings), APIs & auth (APIs, Credentials - which is selected and highlighted in grey), Monitoring, Source Code, Compute (Compute Engine, VM instances, Disks), and a back arrow. The main content area has a title 'OAuth' under 'API Project'. It explains OAuth 2.0 and provides a 'Learn more' link. Below this is a large blue 'Create new Client ID' button. To the right, under 'Public API access', it says 'Use of this key does not require any user action or consent, does not grant access to any account information, and is not used for authorization.' It also has a 'Learn more' link and a 'Create new Key' button. On the far right, there's a table for 'Key for browser applications' with columns for API KEY, REFERERS, ACTIVATION DATE, and ACTIVATED BY. The table shows one entry: API KEY AlzaSyB030S7oOloX0hC9KD7Z90cs1FtWpg6C6l, REFERERS Any referer allowed, ACTIVATION DATE Jun 26, 2014 11:19 AM, and ACTIVATED BY ryan.e.mitchell@gmail.com (you). Below the table are three buttons: 'Edit allowed referrers', 'Regenerate key', and 'Delete'.

Figura 4-3. Página de credenciais de API do Google

Na página Credenciais, você pode clicar no botão “Criar nova chave” para criar uma chave API, com a opção de limitar o endereço IP de referência ou URL que pode usá-la. Para criar uma chave API que pode ser usada a partir de qualquer URL ou endereço IP, basta deixar a caixa “Aceitar solicitação desses endereços IP de servidor” em branco quando solicitado. Lembre-se, entretanto, de que é importante manter sua chave secreta se você não limitar o endereço IP de origem - todas as chamadas feitas usando sua chave de API serão contabilizadas em seu limite de taxa, mesmo se não forem autorizadas.

Você também pode criar várias chaves de API. Por exemplo, você pode ter uma chave API separada para cada projeto ou para cada domínio da web de sua propriedade. No entanto, os limites de taxa de API do Google são por conta, não por chave, portanto, embora essa possa ser uma maneira útil de gerenciar suas permissões de API, não funcionará para contornar a limitação de taxa!

Alguns exemplos

As APIs mais populares (e na minha opinião mais interessantes) do Google podem ser encontradas em sua coleção de APIs do Google Maps. Você pode estar familiarizado com esse recurso por meio do Google Maps embutido, encontrado em muitos sites. No entanto, as APIs do Google Maps vão muito além da incorporação de mapas - você pode resolver endereços de ruas em coordenadas de latitude / longitude, obter a elevação de qualquer ponto da Terra, criar uma variedade de visualizações baseadas em localização e obter informações de fuso horário para um localização, entre outros bits de informação.

Ao experimentar esses exemplos por conta própria, lembre-se de ativar cada API necessária no console de API do Google antes de enviar uma solicitação. O Google usa essas ativações da API para suas métricas (“quantos usuários esta API tem?”), Portanto, é necessário ativar explicitamente a API antes de usar.

Usando a API Geocode do Google, você pode fazer um simples PEGUE solicitação de seu navegador para resolver qualquer endereço (neste caso, o Museu de Ciência de Boston) para uma latitude e longitude:

```
https://maps.googleapis.com/maps/api/geocode/json?address=1+Science+Park+Boston  
+ MA + 02114 & key = <sua chave de API>  
  
"resultados": [{"address_components": [{"long_name": "Museum Of Science Drive way", "name.curto": "Museu da Ci\u00eancia", "tipos": ["rota"]}, {"long_name": "Boston", "name.curto": "Boston", "tipos": ["localidade", "pol\u00edtica"]}], {"long_name": "Massachusetts", "name.curto": "MA", "tipos": ["admin.istrative_area_level_1", "pol\u00edtico"]}, {"long_name": "Estados Unidos", "name.curto": "NOS", "tipos": ["pa\u00eds", "pol\u00edtico"]}], {"long_name": "02114", "name.curto": "02114", "tipos": ["C\u00f3digo postal"]}], "formatted_address":  
  
: "Museum Of Science Driveway, Boston, MA 02114, EUA", "geometria": { "limites":  
{ "nordeste": { "lat": 42.368454, "lng": -71.06961339999999 }, "sudoeste":  
{ "lat": 42.3672568, "lng": -71.0719624 }}, "localiza\u00e7\u00e3o": { "lat": 42.3677994,  
, "lng": -71.0708078 }, "Tipo de localiza\u00e7\u00e3o": "GEOMETRIC_CENTER", "janela de exibi\u00e7\u00e3o": { "n.orthoeste": { "lat": 42.3692043802915, "lng": -71.06943891970849 }, "sudoeste": { "lat": 42.3665064197085, "lng": -71.0721368802915 }},  
"tipos": ["rota"]}], "status": "EST\u00c1 BEM" }
```

Observe que o endereço que enviamos para a API não precisa ser especialmente bem formatado. Sendo Google, a API Geocode é excelente para pegar endereços que não contenham códigos postais ou informações de estado (ou mesmo endereços com erros ortográficos) e dar a você sua melhor estimativa sobre o endereço correto. Por exemplo, o argumento de solicitação escrito de forma lamentável 1 + Skience + Park + Boston + MA (mesmo sem um código postal) retorna o mesmo resultado.

Eu usei a API Geocode em várias ocasiões, não apenas para formatar endereços inseridos pelo usuário em um site, mas para rastrear a Web procurando coisas que se pareçam com endereços e usando a API para reformatá-los em algo mais fácil de armazenar e pesquisar.

Para obter as informações de fuso horário de nossa latitude e longitude recém-encontradas, você pode usar a API de fuso horário:

```
https://maps.googleapis.com/maps/api/timezone/json?location=42.3677994,-71.0708078 & timestamp = 1412649030  
& key = <sua chave de API>
```

Que fornece a resposta:

```
{"dstOffset": 3600, "rawOffset": -18.000, "status": "EST\u00c1 BEM", "timeZoneId": "America/New_York",  
timeZoneName": "Hor\u00e1rio de ver\u00e3o oriental"}
```

Um carimbo de data / hora Unix é necessário para fazer uma solicitação à API de fuso horário. Isso permite que o Google forneça a você um fuso horário devidamente ajustado para o horário de verão. Mesmo em áreas onde o fuso horário não é afetado pela época do ano (como Phoenix, que não usa o horário de verão), o carimbo de data / hora ainda é necessário em uma solicitação de API.

Para completar este breve tour pela coleção da API do Google Maps, você também pode buscar a elevação para esta latitude e longitude:

```
https://maps.googleapis.com/maps/api/elevation/json?locations=42.3677994,-71.0708078 & key = <sua chave de API>
```

Isso retorna a elevação em metros acima do nível do mar, junto com a “resolução”, que indica a distância em metros do ponto de dados mais distante do qual essa elevação foi interpolada. Um valor menor para resolução indica um maior grau de precisão na elevação dada:

```
{ "resultados": [ { "elevação": 5.127755641937256, "localização": { "lat": 42.3677994, "lng": -71.0708078 }, "resolução": 9.543951988220215 }, "status": "ESTÁ BEM" }
```

Analisando JSON

Neste capítulo, vimos vários tipos de APIs e como funcionam, e vimos alguns exemplos de respostas JSON dessas APIs. Agora vamos ver como podemos analisar e usar essas informações.

No início do capítulo, usei o exemplo do *freegeoip.net* IP, que resolve endereços IP para endereços físicos:

```
http://freegeoip.net/json/50.78.253.58
```

Posso pegar a saída desta solicitação e usar as funções de análise JSON do Python para decodificá-la:

```
importar json
de urllib.request importar urlopen

def getCountry ( endereço de IP ):
    resposta = urlopen ( "http://freegeoip.net/json/" + endereço de IP ) . ler ()
        . decodificar ( 'utf-8' )
    responseJson = json . cargas ( resposta )
    Retorna responseJson . pegue ( "Código do país" )

impressão ( getCountry ( "50.78.253.58" ))
```

Isso imprime o código do país para o endereço IP: *50.78.253.58*.

A biblioteca de análise JSON usada faz parte da biblioteca central do Python. Basta digitar `import json` no topo e está tudo pronto! Ao contrário de muitas linguagens que podem analisar JSON em

um objeto JSON especial ou nó JSON, o Python usa uma abordagem mais flexível e transforma objetos JSON em dicionários, arrays JSON em listas, strings JSON em strings e assim por diante. Dessa forma, é extremamente fácil acessar e manipular valores armazenados em JSON.

A seguir, uma rápida demonstração de como a biblioteca JSON do Python lida com os diferentes valores que podem ser encontrados em uma string JSON:

```
importar json

jsonString = '{"arrayOfNums": [{"número": 0}, {"número": 1}, {"número": 2}],
              "arrayOfFruits" : [{"fruta": "maçã"}, {"fruta": "banana"},
                                 {"fruta": "pera"}]}

jsonObj = json . cargas ( jsonString )

impressão ( jsonObj . pegue ( "arrayOfNums" ))
impressão ( jsonObj . pegue ( "arrayOfNums" ) [ 1 ])
impressão ( jsonObj . pegue ( "arrayOfNums" ) [ 1 ] . pegue ( "número" ) +
            jsonObj . pegue ( "arrayOfNums" ) [ 2 ] . pegue ( "número" ))
impressão ( jsonObj . pegue ( "arrayOfFruits" ) [ 2 ] . pegue ( "fruta" ))
```

O resultado é:

```
[{'número': 0}, {'número': 1}, {"número": 2}] {"número": 1}
3
pera
```

A linha 1 é uma lista de objetos de dicionário, a linha 2 é um objeto de dicionário, a linha 3 é um inteiro (a soma dos inteiros acessados nos dicionários) e a linha 4 é uma string.

Trazendo tudo de volta para casa

Apesar de *razão de ser* de muitos aplicativos da web modernos é pegar os dados existentes e formatá-los de uma forma mais atraente, eu diria que isso não é uma coisa muito interessante de se fazer na maioria dos casos. Se você estiver usando uma API como sua única fonte de dados, o melhor que você pode fazer é simplesmente copiar o banco de dados de outra pessoa que já existe e que está, essencialmente, publicado. O que pode ser muito mais interessante é pegar duas ou mais fontes de dados e combiná-las de uma maneira nova ou usar uma API como uma ferramenta para ver os dados extraídos de uma nova perspectiva.

Vejamos um exemplo de como dados de APIs podem ser usados em conjunto com web scraping: para ver quais partes do mundo contribuem mais para a Wikipedia.

Se você passou muito tempo na Wikipedia, provavelmente já se deparou com uma página de histórico de revisões de um artigo, que exibe uma lista de edições recentes. Se os usuários estiverem logados no Wikipedia quando fizerem a edição, seu nome de usuário será exibido. Se eles não estiverem logados, seu endereço IP é registrado, conforme mostrado em [Figura 4-4](#).

Article Talk Read Edit View history Search

Python: Revision history

[View logs for this page](#)

Browse history

From year (and earlier): 2014 From month (and earlier): all Tag filter: Go

For any version listed below, click on its date to view it. For more help, see Help:Page history and Help>Edit summary.

External tools: Revision history statistics · Revision history search · Edits by user · Number of watchers · Page view statistics

(cur) = difference from current version, (prev) = difference from preceding version, m = minor edit, → = section edit, ← = automatic edit summary
 (newest | oldest) View (newer 50 | older 50) (20 | 50 | 100 | 250 | 500)

[Compare selected revisions](#)

- (cur | prev) 00:42, 29 August 2014 Discospinster (talk | contribs) m . . (1,715 bytes) (-21) . . (Reverted edits by 121.97.110.145 (talk) to last revision by Bgwhite (HG)) (undo)
- (cur | prev) 00:41, 29 August 2014 121.97.110.145 (talk) . . (1,736 bytes) (+21) . . (→In computing) (undo)
- (cur | prev) 05:53, 10 June 2014 Bgwhite (talk | contribs) . . (1,715 bytes) (+49) . . (Reverted to revision 808657990 by Bgwhite: No content between TOC and first headline per WP:TOC and WP:LEAD. This is an accessibility issue for users of screen readers. (TW)) (undo)
- (cur | prev) 23:01, 9 June 2014 Nvmbz (talk | contribs) . . (1,670 bytes) (-62) . . (rephrase for link in the front) (undo)
- (cur | prev) 17:26, 9 June 2014 Bjennelle (talk | contribs) . . (1,732 bytes) (+17) . . (Link to the python snake when it's mentioned.) (undo)

Figura 4-4. O endereço IP de um editor anônimo na página de histórico de revisão para a entrada Python do Wiki-pedia

O endereço IP descrito na página do histórico é 121.97.110.145. Usando o free-geoip.net API, no momento em que este artigo foi escrito (o endereço IP pode ocasionalmente mudar geograficamente), esse endereço IP é de Queens, Filipinas.

Essa informação não é tão interessante por si só, mas e se pudéssemos reunir muitos, muitos pontos de dados geográficos sobre as edições da Wikipedia e onde elas ocorrem? Alguns anos atrás eu fiz exatamente isso e usei [Biblioteca Geochart do Google](#) para criar um gráfico interessante (<http://www.pythonscraping.com/pages/wikipedia.html>) que mostra de onde vêm as edições na Wikipédia em inglês, junto com as Wikipédias escritas em outros idiomas ([Figura 4-5](#))

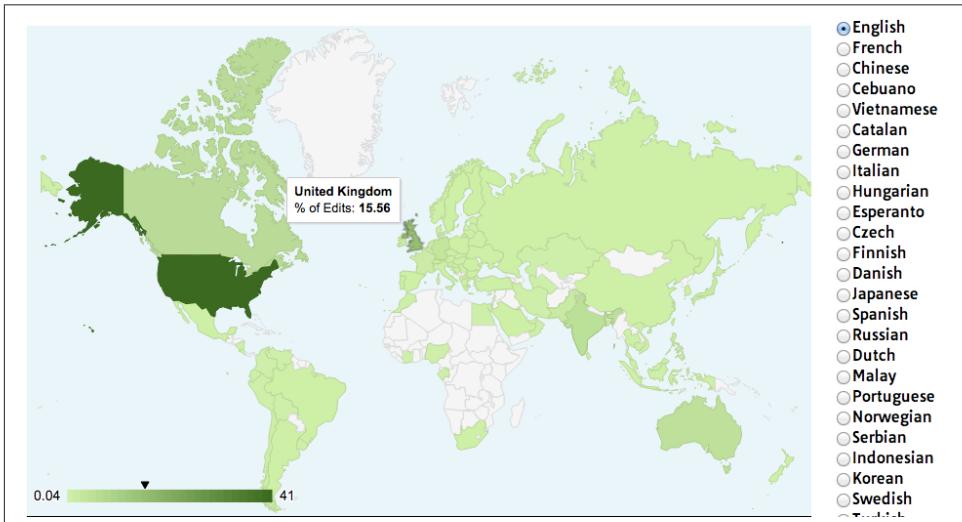


Figura 4-5. Visualização de edições da Wikipedia criadas usando a biblioteca Geochart do Google

Criar um script básico que rastreia a Wikipedia, procura por páginas de histórico de revisão e, em seguida, procura endereços IP nessas páginas de histórico de revisão não é difícil. Usando código modificado do [Capítulo 3](#), o seguinte script faz exatamente isso:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar data hora
importar aleatória
importar ré

aleatória . semente ( data hora . data hora . agora ())
def getLinks ( articleUrl ):
    html = urlopen ( "http://en.wikipedia.org" + articleUrl )
    bsObj = BeautifulSoup ( html )
    Retorna bsObj . encontrar ( "div" , { "Eu iria" : "bodyContent" }) . encontrar tudo ( "uma" ,
        href = ré . compilar ( "^ (/ wiki /)(?!:.)*$" ))

def getHistoryIPs ( URL da página ):
    # O formato das páginas do histórico de revisão é:
    # http://en.wikipedia.org/w/index.php?title=Title_in_URL&action=history
    URL da página = URL da página . substituir ( '/ wiki /' , "" )
    historyUrl = "http://en.wikipedia.org/w/index.php?title="
        + URL da página + "& ação = histórico"
    impressão ( " o url do histórico é: " + historyUrl )
    html = urlopen ( historyUrl )
    bsObj = BeautifulSoup ( html )
    # encontra apenas os links com a classe "mw-anonuserlink" que tem endereços IP
    # em vez de nomes de usuário
    ipAddresses = bsObj . encontrar tudo ( "uma" , { "classe" : "mw-anonuserlink" })
    addressList = conjunto ()
    para endereço de IP dentro ipAddresses :
        addressList . adicionar ( endereço de IP . get_text () )
    Retorna addressList

links = getLinks ( "/ wiki / Python_ (linguagem_de_programação)" )

enquanto ( len ( links ) > 0 ):
    para ligação dentro links :
        impressão ( "-----" )
        historyIPs = getHistoryIPs ( ligação . attrs [ "href" ])
        para historyIP dentro historyIPs :
            impressão ( historyIP )

newLink = links [ aleatória . Randint ( 0 , len ( links ) - 1 ) ] . attrs [ "href" ]
links = getLinks ( newLink )
```

Este programa usa duas funções principais: `getLinks` (que também foi usado em [Capítulo 3](#)), e o novo `getHistoryIPs`, que procura o conteúdo de todos os links com o

classe mw-anonuserlink (indicando um usuário anônimo com um endereço IP, em vez de um nome de usuário) e o retorna como um conjunto.

Vamos falar sobre conjuntos

Até este ponto do livro, confie quase que exclusivamente em duas estruturas de dados Python para armazenar vários dados: listas e dicionários. Com ambas as opções, por que usar um conjunto?

Os conjuntos Python não são ordenados, o que significa que você não deve fazer referência a uma posição específica no conjunto e esperar obter o valor que está procurando. A ordem em que você adiciona itens ao conjunto não é necessariamente a ordem em que você os receberá de volta. Uma boa propriedade dos conjuntos de que estou aproveitando no exemplo de código é que eles não armazenam múltiplos do mesmo item. Se você adicionar uma string a um conjunto quando essa string já existir, ela não será duplicada. Dessa forma, posso obter rapidamente uma lista apenas dos endereços IP exclusivos na página do histórico de revisão, desconsiderando várias edições do mesmo usuário.

Algumas coisas para ter em mente ao decidir entre conjuntos e listas no código que precisam ser escalonados: embora as listas sejam um pouco mais rápidas para iterar, os conjuntos são um pouco mais rápidos para fazer pesquisas (determinar se um objeto existe no conjunto ou não).

Este código também usa um padrão de pesquisa um tanto arbitrário (mas eficaz para os propósitos deste exemplo) para procurar artigos dos quais recuperar históricos de revisão. Ele começa recuperando as histórias de todos os artigos da Wikipedia vinculados pela página inicial (neste caso, o artigo sobre a linguagem de programação Python).

Posteriormente, ele seleciona uma nova página inicial aleatoriamente e recupera todas as páginas do histórico de revisão de artigos vinculados a essa página. Ele continuará até chegar a uma página sem links.

Agora que temos um código que recupera endereços IP como uma string, podemos combinar isso com o getCountry função da seção anterior para resolver esses endereços IP para países. Eu modifiquei getCountry ligeiramente, a fim de contabilizar endereços IP inválidos ou malformados que resultarão em um erro "404 não encontrado" (a partir desta redação, *freegeoip.net* não resolve IPv6, por exemplo, o que pode desencadear esse erro):

```
def getCountry ( endereço de IP ):  
    experimentar :  
        resposta = urlopen ( "http://freegeoip.net/json/"  
            + endereço de IP ) . ler () . decodificar ( 'utf-8' )  
    exceto Erro HTTP :  
        Retorna Nenhum  
        responseJson = json . cargas ( resposta )  
        Retorna responseJson . pegue ( "Código do país" )  
  
    links = getLinks ( "/ wiki / Python_ (linguagem_de_programação)" )
```

```

enquanto ( len ( links ) > 0 ):
    para ligação dentro links :
        impressão ( "-----" )
        historyIPs = getHistoryIPs ( ligação . attrs [ "href" ] )
        para historyIP dentro historyIPs :
            país = getCountry ( historyIP )
            E se país não é Nenhum :
                impressão ( historyIP + " é de " + país )

        newLink = links [ aleatória . Randint ( 0 , len ( links ) - 1 ) ] . attrs [ "href" ]
        links = getLinks ( newLink )

```

Aqui está um exemplo de saída:

o url do histórico é: http://en.wikipedia.org/w/index.php?title=Programming_paradigm&action=history

68.183.108.13 é dos EUA
 86.155.0.186 é de GB
 188.55.200.254 é de SA
 108.221.18.208 é dos EUA
 141.117.232.168 é da CA
 76.105.209.39 é dos EUA
 182.184.123.106 é de PK
 212.219.47.52 é de GB
 72.27.184.57 é de JM
 49.147.183.43 é de PH
 209.197.41.132 é dos EUA
 174.66.150.151 é dos EUA

Mais sobre APIs

Neste capítulo, vimos algumas maneiras como as APIs modernas são comumente usadas para acessar dados na Web, em particular usos de APIs que podem ser úteis para web scraping. No entanto, temo que isso não faça justiça ao amplo escopo que o “software compartilhando dados com softwares distintos” envolve.

Como este livro é sobre web scraping e não tem a intenção de ser um guia geral sobre coleta de dados, posso apenas indicar alguns recursos excelentes para pesquisas adicionais sobre o assunto, se necessário.

Leonard Richardson, Mike Amundsen e Sam Ruby *APIs da Web RESTful* fornece uma forte visão geral da teoria e prática do uso de APIs na web. Além disso, Mike Amundsen tem uma série de vídeos fascinantes, *Projetando APIs para a Web*, que ensina como criar suas próprias APIs, algo útil para saber se você decidir tornar seus dados copiados disponíveis ao público em um formato conveniente.

Web scraping e APIs da web podem parecer assuntos muito diferentes à primeira vista. No entanto, espero que este capítulo tenha mostrado que são habilidades complementares no mesmo continuum de coleta de dados. Em certo sentido, usar uma API da web pode até ser

pensado como um subconjunto do assunto web scraping. Afinal, você está escrevendo um script que coleta dados de um servidor web remoto e os analisa em um formato utilizável, como faria com qualquer web scraper.

CAPÍTULO 5

Armazenamento de dados

Embora imprimir para o terminal seja muito divertido, não é muito útil quando se trata de agregação e análise de dados. Para tornar a maioria dos web scrapers remotamente úteis, você precisa ser capaz de salvar as informações que eles copiam.

Neste capítulo, veremos três métodos principais de gerenciamento de dados que são suficientes para quase todas as aplicações imagináveis. Você precisa ativar o back-end de um site ou criar sua própria API? Você provavelmente vai querer que seus scrapers gravem em um banco de dados. Precisa de uma maneira rápida e fácil de coletar alguns documentos da Internet e colocá-los em seu disco rígido? Você provavelmente vai querer criar um fluxo de arquivos para isso. Precisa de alertas ocasionais ou dados agregados uma vez por dia? Envie um email para si mesmo!

Acima e além da web scraping, a capacidade de armazenar e interagir com grandes quantidades de dados é extremamente importante para praticamente qualquer aplicativo de programação moderno. Na verdade, as informações neste capítulo são necessárias para implementar muitos dos exemplos nas seções posteriores do livro. Eu recomendo que você pelo menos dê uma olhada se não estiver familiarizado com o armazenamento automatizado de dados.

Arquivos de mídia

Existem duas maneiras principais de armazenar arquivos de mídia: por referência e baixando o próprio arquivo. Você pode armazenar um arquivo por referência simplesmente armazenando a URL em que o arquivo está localizado. Isso tem várias vantagens:

- Os scrapers são executados com muito mais rapidez e exigem muito menos largura de banda, quando não é necessário fazer download de arquivos.
- Você economiza espaço em sua própria máquina, armazenando apenas os URLs.
- É mais fácil escrever código que armazena apenas URLs e não precisa lidar com downloads de arquivos adicionais.

- Você pode diminuir a carga no servidor host, evitando downloads de arquivos grandes. Aqui estão as desvantagens:
 - Incorporar esses URLs em seu próprio site ou aplicativo é conhecido como *hotlink-ing* e fazer isso é uma maneira muito rápida de colocá-lo em apuros na Internet.
 - Você não deseja usar os ciclos de servidor de outra pessoa para hospedar mídia para seus próprios aplicativos.
 - O arquivo hospedado em qualquer URL específico está sujeito a alterações. Isso pode levar a efeitos embaraçosos se, digamos, você estiver incorporando uma imagem com hotlink em um blog público. Se você estiver armazenando os URLs com a intenção de armazenar o arquivo posteriormente, para pesquisas adicionais, ele pode eventualmente desaparecer ou ser alterado para algo completamente irrelevante em uma data posterior.
 - Os navegadores reais não apenas solicitam o HTML de uma página e seguem em frente - eles também baixam todos os recursos exigidos pela página. Baixar arquivos pode ajudar a dar a aparência de que um humano real está navegando no site, o que pode ser uma vantagem.

Se você está debatendo se deve armazenar um arquivo ou simplesmente um URL para um arquivo, você deve se perguntar se é provável que realmente veja ou leia esse arquivo mais de uma ou duas vezes, ou se este banco de dados de arquivos está apenas indo ficar sentado juntando poeira eletrônica durante a maior parte de sua vida. Se a resposta for a última, provavelmente é melhor simplesmente armazenar o URL. Se for o primeiro, continue lendo!

No Python 3.x, `urllib.request.urlretrieve` pode ser usado para baixar arquivos de qualquer URL remoto:

```
de urllib.request importar urlretrieve
de urllib.request importar urlopen
de bs4 importar BeautifulSoup

html = urlopen ( "http://www.pythonscraping.com" )
bsObj = BeautifulSoup ( html )
imageLocation = bsObj . encontrar ( "uma" , { "Eu iria" : "logotipo" } ) . encontrar ( "img" ) [ "src" ]
urlretrieve ( imageLocation , "logo.jpg" )
```

Isso baixa o logotipo de <http://pythonscraping.com> e armazena como *logo.jpg* no mesmo diretório a partir do qual o script está sendo executado.

Isso funciona bem se você só precisar baixar um único arquivo e saber como chamá-lo e qual é a extensão do arquivo. Mas a maioria dos scrapers não baixa um único arquivo e termina o dia. O seguinte irá baixar todos os arquivos internos, vinculados por qualquer tag `src` atributo, da página inicial de <http://pythonscraping.com>:

```
importar os
de urllib.request importar urlretrieve
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
```

```

downloadDirectory = "baixado"
baseUrl = "http://pythonscraping.com"

def getAbsoluteURL ( baseUrl , fonte ):
    E se fonte . começa com ( "http: // www." ):
        url = "http: //" + fonte [ 11 :]
    elif fonte . começa com ( "http: //" ):
        url = fonte
    elif fonte . começa com ( "www." ):
        url = fonte [ 4 :]
        url = "http: //" + fonte
    outro :
        url = baseUrl + "/" + fonte
    E se baseUrl não em url :
        Retorna Nenhum
    Retorna url

def getDownloadPath ( baseUrl , absoluteUrl , downloadDirectory ):
    caminho = absoluteUrl . substituir ( "www." , "" )
    caminho = caminho . substituir ( baseUrl , "" )
    caminho = downloadDirectory + caminho
    diretório = os . caminho . dimame ( caminho )

    E se não os . caminho . existe ( diretório ):
        os . makedirs ( diretório )

    Retorna caminho

html = urlopen ( "http://www.pythonscraping.com" )
bsObj = BeautifulSoup ( html )
downloadList = bsObj . encontrar tudo ( src = Verdadeiro )

para baixar dentro downloadList :
    fileUrl = getAbsoluteURL ( baseUrl , baixar [ "src" ] )
    E se fileUrl não é Nenhum :
        impressão ( fileUrl )

urlretrieve ( fileUrl , getDownloadPath ( baseUrl , fileUrl , downloadDirectory ))

```



Corra com Cuidado

Você conhece todos aqueles avisos que ouve sobre o download de arquivos desconhecidos da Internet? Este script baixa tudo o que encontra para o disco rígido do seu computador. Isso inclui scripts de bash aleatórios. *Exe* arquivos e outro malware potencial.

Acha que está seguro porque nunca executaria nada enviado para sua pasta de downloads? Especialmente se você executar este programa como administrador, você está procurando problemas. O que acontece se você encontrar um arquivo em um site que se auto-envia para `.....usr/bin/python`? Na próxima vez que você executar um script Python na linha de comando, poderá realmente estar implantando malware em sua máquina!

Este programa foi escrito apenas para fins ilustrativos; ele não deve ser implantado aleatoriamente sem uma verificação mais extensa do nome do arquivo e deve ser executado apenas em uma conta com permissões limitadas. Como sempre, fazer backup de seus arquivos, não armazenar informações confidenciais em seu disco rígido e um pouco de bom senso ajudam muito.

Este script usa uma função lambda (introduzida em [Capítulo 2](#)) para selecionar todas as tags na página inicial que possuem o `src` e, em seguida, limpa e normaliza os URLs para obter um caminho absoluto para cada download (certificando-se de descartar links externos). Em seguida, cada arquivo é baixado para seu próprio caminho na pasta local `baixado` em sua própria máquina.

Observe que o módulo OS do Python é usado brevemente para recuperar o diretório de destino para cada download e criar diretórios ausentes ao longo do caminho, se necessário. O módulo os atua como uma interface entre o Python e o sistema operacional, permitindo-lhe manipular caminhos de arquivo, criar diretórios, obter informações sobre processos em execução e variáveis de ambiente e muitas outras coisas úteis.

Armazenamento de dados em CSV

CSV, ou *Valores Separados Por Virgula*, é um dos formatos de arquivo mais populares para armazenar dados de planilha. É suportado pelo Microsoft Excel e muitos outros aplicativos devido à sua simplicidade. A seguir está um exemplo de um arquivo CSV perfeitamente válido:

```
fruta, custo  
maçã, 1,00  
banana, 0,30  
péra, 1,25
```

Como no Python, o espaço em branco é importante aqui: cada linha é separada por um caractere de nova linha, enquanto as colunas dentro da linha são separadas por vírgulas (portanto, “separadas por vírgula”). Outras formas de arquivos CSV (às vezes chamados de “caracteres-

valores separados ") usam tabulações ou outros caracteres para separar linhas, mas esses formatos de arquivo são menos comuns e menos suportados.

Se você deseja baixar arquivos CSV diretamente da Web e armazená-los localmente, sem qualquer análise ou modificação, você não precisa desta seção. Simplesmente baixe-os como faria com qualquer outro arquivo e salve-os com o formato de arquivo CSV usando os métodos descritos na seção anterior.

Modificar um arquivo CSV, ou mesmo criar um inteiramente do zero, é extremamente fácil com a biblioteca csv do Python:

```
importar csv

csvFile = abrir ( "../files/test.csv" , 'w +' )
experimentar :
    escritor = csv . escritor ( csvFile )
    escritor . escritor ( ( 'número' , 'número mais 2' , 'número vezes 2' ) )
    para Eu dentro alcance ( 10 ):
        escritor . escritor ( ( Eu , Eu + 2 , Eu * 2 ) )
    finalmente :
        csvFile . Fechar ()
```

Um lembrete de precaução: a criação de arquivos em Python é bastante à prova de balas. E se *.. / arquivos / test.csv* ainda não existir, o Python criará o arquivo (mas não o diretório) automaticamente. Se já existir, Python irá sobrescrever *test.csv* com os novos dados.

Após a execução, você deverá ver um arquivo CSV:

```
número, número mais 2, número vezes 2
0,2,0
1,3,2
2,4,4
...
```

Uma tarefa comum de web scraping é recuperar uma tabela HTML e gravá-la como um arquivo CSV. [Comparação de editores de texto da Wikipedia](#) fornece uma tabela HTML bastante complexa, completa com codificação de cores, links, classificação e outros resíduos HTML que precisam ser descartados antes de serem gravados em CSV. Usando BeautifulSoup e o get_text ()

funcionar copiosamente, você pode fazer isso em menos de 20 linhas:

```
importar csv
de urllib.request importar urlopen
de bs4 importar BeautifulSoup

html = urlopen ( "http://en.wikipedia.org/wiki/Comparison_of_text_editors" )
bsObj = BeautifulSoup ( html )
# A principal tabela de comparação é atualmente a primeira tabela da página
mesa = bsObj . encontrar tudo ( "mesa" , { "classe" : "wikitable" } )[ 0 ]
filas = mesa . encontrar tudo ( "tr" )

csvFile = abrir ( "../files/editors.csv" , 'wt' )
escritor = csv . escritor ( csvFile )
```

```

experimentar :

para linha dentro filas :
    csvRow = []
    para célula dentro linha . encontrar tudo ([ 'td' , 'td' ]):
        csvRow . acrescentar ( célula . get_text ())
        escritor . escritor ( csvRow )
finalmente :
    csvFile . Fechar ()

```

Antes de implementar isso na vida real

Este script é ótimo para integrar em scrapers se você encontrar muitas tabelas HTML que precisam ser convertidas em arquivos CSV ou muitas tabelas HTML que precisam ser coletadas em um único arquivo CSV. No entanto, se você precisar fazer isso apenas uma vez, existe uma ferramenta melhor para isso: copiar e colar. Selecionar e copiar todo o conteúdo de uma tabela HTML e colá-lo no Excel fornecerá o arquivo CSV que você está procurando sem executar um script!

O resultado deve ser um arquivo CSV bem formatado salvo localmente, em *.. / files / editors.csv*

- perfeito para enviar e compartilhar com pessoas que ainda não pegaram o jeito do MySQL!

MySQL

MySQL (oficialmente pronunciado "My es-kew-el", embora muitos digam "My Sequel") é o sistema de gerenciamento de banco de dados relacional de código aberto mais popular atualmente. O que é incomum para um projeto de código aberto com grandes concorrentes, sua popularidade tem sido historicamente pescoco a pescoço com os dois outros principais sistemas de banco de dados de código fechado: SQL Server da Microsoft e DBMS da Oracle.

Sua popularidade não é sem causa. Para a maioria dos aplicativos, é muito difícil dar errado com o MySQL. É um DBMS muito escalonável, robusto e completo, usado pelos principais sites: YouTube,¹ Twitter,² e Facebook,³ entre muitos outros.

¹ Joab Jackson, "YouTube Scales MySQL with Go Code", PCWorld, 15 de dezembro de 2012 (<http://bit.ly/1LWVmc8>).

² Jeremy Cole e Davi Arnaut, "MySQL at Twitter," The Twitter Engineering Blog, 9 de abril de 2012 (<http://bit.ly/1KHDKns>).

³ "MySQL and Database Engineering: Mark Callaghan," 4 de março de 2012 (<http://on.fb.me/1RFMqvw>).

Por causa de sua onipresença, preço (“grátis” é um preço muito bom) e usabilidade pronta para uso, ele é um banco de dados fantástico para projetos de web scraping e nós o usaremos no restante deste livro.

Banco de dados “relacional”?

“Dados relacionais” são os dados que têm relações. Fico feliz que esclarecemos tudo!

Só brincando! Quando os cientistas da computação falam sobre dados relacionais, eles estão se referindo a dados que não existem no vácuo - eles têm propriedades que os relacionam a outras partes dos dados. Por exemplo, “O usuário A vai para a escola na Instituição B”, onde o Usuário A pode ser encontrado na tabela “usuários” no banco de dados e a Instituição B pode ser encontrado na tabela “instituições” no banco de dados.

Posteriormente neste capítulo, daremos uma olhada na modelagem de diferentes tipos de relações e como armazenar dados no MySQL (ou qualquer outro banco de dados relacional) de forma eficaz.

Instalando MySQL

Se você é novo no MySQL, instalar um banco de dados pode parecer um pouco intimidante (se você for um veterano nisso, sinta-se à vontade para pular esta seção). Na realidade, é tão simples quanto instalar praticamente qualquer outro tipo de software. Basicamente, o MySQL é alimentado por um conjunto de arquivos de dados, armazenados em seu servidor ou máquina local, que contém todas as informações armazenadas em seu banco de dados. A camada de software MySQL em cima disso fornece uma maneira conveniente de interagir com os dados, por meio de uma interface de linha de comando. Por exemplo, o comando a seguir pesquisará os arquivos de dados e retornará uma lista de todos os usuários em seu banco de dados cujo primeiro nome é “Ryan”:

SELECIONE * DE Comercial ONDE primeiro nome = "Ryan"

Se você estiver no Linux, instalar o MySQL é tão fácil quanto:

```
$ sudo apt-get install mysql-server
```

Fique atento ao processo de instalação, aprove os requisitos de memória e insira uma nova senha para o novo usuário root quando solicitado.

Para Mac OS X e Windows, as coisas são um pouco mais complicadas. Caso ainda não tenha feito isso, você precisará criar uma conta Oracle antes de baixar o pacote.

Se você estiver no Mac OS X, precisará primeiro obter [o pacote de instalação](#).

Selecione os *.dmg* pacote e faça login ou crie sua conta Oracle para baixar o arquivo. Após a abertura, você deve ser guiado por um assistente de instalação bastante simples ([Figura 5-1](#)):

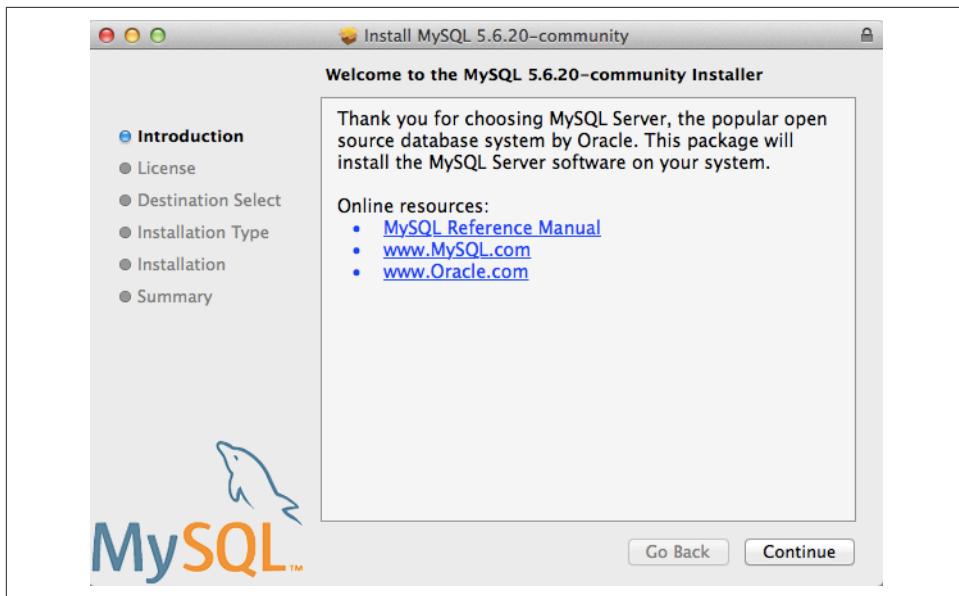


Figura 5-1. O instalador do Mac OS X MySQL

As etapas de instalação padrão devem bastar e, para os propósitos deste livro, assumirei que você possui uma instalação MySQL padrão.

Se baixar e executar um instalador parece um pouco tedioso, você sempre pode instalar o gerenciador de pacotes, [Homebrew](#). Com o Homebrew instalado, você também pode instalar o MySQL executando:

```
$ brew install mysql
```

Homebrew é um ótimo projeto de código aberto com uma integração de pacote Python muito boa. Na verdade, a maioria dos módulos Python de terceiros usados neste livro podem ser instalados facilmente com o Homebrew. Se você ainda não o tem, recomendo dar uma olhada!

Depois que o MySQL estiver instalado no Mac OS X, você pode iniciar o servidor MySQL da seguinte maneira:

```
$ cd /usr/local/mysql  
$ sudo ./bin/mysqld_safe
```

No Windows, instalar e executar o MySQL é um pouco mais complicado, mas a boa notícia é que há [um instalador conveniente](#) que simplifica o processo. Depois de baixado, ele o guiará pelas etapas que você precisará seguir (consulte [Figura 5-2](#)):



Figura 5-2. O MySQL Windows Installer

Você deve conseguir instalar o MySQL usando as seleções padrão, com uma exceção: na página Tipo de instalação, recomendo que você escolha “Somente servidor” para evitar a instalação de muitos softwares e bibliotecas adicionais da Microsoft.

A partir daí, você deve ser capaz de usar as configurações de instalação padrão e seguir os prompts para iniciar o servidor MySQL.

Alguns comandos básicos

Depois que o servidor MySQL estiver em execução, você terá muitas opções para interagir com o banco de dados. Existem muitas ferramentas de software que atuam como intermediárias para que você não tenha que lidar com os comandos do MySQL (ou pelo menos lidar com eles com menos frequência). Ferramentas como phpMyAdmin e MySQL Workbench podem facilitar a visualização, classificação e inserção rápida de dados. No entanto, ainda é importante conhecer a linha de comando.

Exceto para nomes de variáveis, o MySQL não faz distinção entre maiúsculas e minúsculas; por exemplo, SELEÇÃO é o mesmo que sElEcT. No entanto, por convenção, todas as palavras-chave do MySQL estão em maiúsculas quando você está escrevendo uma instrução MySQL. Por outro lado, a maioria dos desenvolvedores prefere nomear suas tabelas e bancos de dados em letras minúsculas, embora esse padrão seja frequentemente ignorado.

Quando você faz login pela primeira vez no MySQL, não há bancos de dados aos quais adicionar dados. Você pode criar um:

> CRIAR BASE DE DADOS raspagem ;

Como cada instância do MySQL pode ter vários bancos de dados, antes de começarmos a interagir com um banco de dados, precisamos especificar para o MySQL qual banco de dados queremos usar:

> USAR raspagem ;

Deste ponto em diante (pelo menos até fecharmos a conexão MySQL ou mudarmos para outro banco de dados), todos os comandos inseridos serão executados no novo banco de dados de “raspagem”.

Tudo isso parece bastante simples. Deve ser igualmente fácil criar uma tabela no banco de dados, certo? Vamos tentar criar uma tabela para armazenar uma coleção de páginas da web copiadas:

> CRIAR A TABELA Páginas ;

Isso resulta no erro:

ERROR 1113 (42000): Uma tabela deve ter pelo menos 1 coluna

Ao contrário de um banco de dados, que pode existir sem tabelas, uma tabela no MySQL não pode existir sem colunas. Para definir colunas no MySQL, você deve inseri-las em uma lista delimitada por vírgulas, entre parênteses, após o CRIAR TABELA <tablename> declaração:

```
> CRIAR A TABELA Páginas ( Eu iria BIGINT (7) NÃO NULO INCREMENTO AUTOMÁTICO , título VARCHAR ( 200 ),  
conteúdo VARCHAR ( 10.000 ), criada TIMESTAMP DEFAULT CURRENT_TIMESTAMP , CHAVE PRIMÁRIA  
( Eu iria ));
```

Cada definição de coluna tem três partes:

- O nome (id, título, criado, etc.)
- O tipo de variável (BIGINT (7), VARCHAR, TIMESTAMP)
- Opcionalmente, quaisquer atributos adicionais (NOT NULL AUTO_INCREMENT)

No final da lista de colunas, você deve definir a “chave” de uma tabela. O MySQL usa chaves para organizar o conteúdo na tabela para pesquisas rápidas. Posteriormente neste capítulo, descreverei como usar essas chaves a seu favor para bancos de dados mais rápidos, mas, por enquanto, usando uma tabela Eu iria coluna como a chave geralmente é o melhor caminho a seguir.

Após a execução da consulta, você pode ver como a estrutura da tabela se parece a qualquer momento usando DESCERVER:

```
> DESCREVER páginas;
+-----+-----+-----+-----+
| Campo | Tipo          | Nulo | Chave | Padrão           | Extra      |
+-----+-----+-----+-----+
| Eu iria | bigint (7)    | NÃO   | PRI   | NULO             | auto_increment |
| título  | varchar (200)  | SIM   |       | NULO             |
| conteúdo | varchar (10000) | SIM   |       | NULO             |
| criado  | timestamp     | NÃO   |       | CURRENT_TIMESTAMP |
+-----+-----+-----+-----+
4 linhas em conjunto (0,01 s)
```

Claro, esta ainda é uma mesa vazia. Você pode inserir alguns dados de teste na tabela de páginas, usando a seguinte linha:

```
> INSIRA DENTRO DE Páginas ( título , conteúdo ) VALORES ( " Título da página de teste " , "Este é o conteúdo da página principal. Pode ter até 10.000 caracteres." );
```

Observe que embora a tabela tenha quatro colunas (id, título, conteúdo, criado), você precisa definir apenas dois deles (título e conteúdo) para inserir uma linha. É porque o Eu iria coluna é autoincrementada (o MySQL adiciona automaticamente 1 cada vez que uma nova linha é inserida) e geralmente pode cuidar de si mesmo. Além disso, o timestamp coluna é definida para conter a hora atual como padrão.

Claro, nós podemos substituir esses padrões:

```
INSIRA DENTRO DE Páginas ( Eu iria , título , conteúdo , criada ) VALORES ( 3 , "Título da página de teste" , "Este é o conteúdo da página de teste. Pode ter até 10.000 caracteres." , "201409-21 10:25:32" );
```

Contanto que o número inteiro que você fornece para o Eu iria coluna ainda não existe no banco de dados, esta substituição funcionará perfeitamente bem. No entanto, geralmente é uma prática ruim fazer isso; é melhor deixar o MySQL lidar com o Eu iria e timestamp colunas, a menos que haja um motivo convincente para fazê-lo de forma diferente.

Agora que temos alguns dados na tabela, você pode usar uma ampla variedade de métodos para selecionar esses dados. Aqui estão alguns exemplos de SELECIONE afirmações:

```
> SELECIONE * DE Páginas ONDE Eu iria = 2 ;
```

Esta declaração diz ao MySQL, "Selecione todas as páginas onde id é igual a 2." O asterisco (*) atua como um caractere curinga, retornando todas as linhas onde a cláusula (onde id é igual a 2) é verdade. Ele retorna a segunda linha da tabela, ou um resultado vazio se não houver linha com um Eu iria de 2. Por exemplo, a seguinte consulta que não diferencia maiúsculas de minúsculas retorna todas as linhas em que o campo de título contém "teste" (o símbolo% atua como um caractere curinga em strings MySQL):

```
> SELECIONE * DE Páginas ONDE título GOSTAR "% teste%" ;
```

Mas e se você tiver uma tabela com muitas colunas e quiser apenas um determinado dado retornado? Em vez de selecionar todos, você pode fazer algo assim:

```
> SELECIONE Eu iria , título DE Páginas ONDE conteúdo GOSTAR "% Conteúdo da página%" ;
```

Isso retorna apenas o Eu iria e título onde o conteúdo contém a frase "conteúdo da página".

EXCLUIR declarações têm praticamente a mesma sintaxe que SELECIONE afirmações:

```
> EXCLUIR DE Páginas ONDE Eu iria = 1 ;
```

Por este motivo, é uma boa ideia, especialmente ao trabalhar em bancos de dados importantes que não podem ser facilmente restaurados, escrever qualquer EXCLUIR declarações como um SELECIONE declaração primeiro (neste caso, SELECIONE * DAS páginas ONDE id = 1), teste para garantir que apenas as linhas que deseja excluir sejam retornadas e, em seguida, substitua SELECIONE * com EXCLUIR. Muitos programadores têm histórias horríveis de erros de codificação da cláusula em um EXCLUIR declaração, ou pior, deixando-a inteiramente de fora quando eles estavam com pressa e estragando os dados do cliente. Não deixe isso acontecer com você!

Precauções semelhantes devem ser tomadas com ATUALIZAR afirmações:

```
> ATUALIZAR Páginas CONJUNTO título = "Um novo título" , conteúdo = "Algum novo conteúdo" ONDE Eu iria = 2 ;
```

Para os propósitos deste livro, trabalharemos apenas com instruções simples do MySQL, fazendo seleção, inserção e atualização básicas. Se você estiver interessado em aprender mais comandos e técnicas com esta poderosa ferramenta de banco de dados, eu recomendo Paul DuBois *Livro de receitas MySQL*.

Integrando com Python

Infelizmente, o suporte Python para MySQL não está embutido. No entanto, há um bom número de bibliotecas de código aberto que você pode usar, tanto com Python 2.x e Python 3.x, que permitem interagir com um banco de dados MySQL. Um dos mais populares deles é

[PyMySQL](#).

No momento em que este documento foi escrito, a versão atual do PyMySQL é 0.6.2 e você pode baixar e instalar esta versão com os seguintes comandos:

```
$ curl -L https://github.com/PyMySQL/PyMySQL/tarball/pymysql-0.6.2 | tar xz  
$ CD PyMySQL-PyMySQL-f953785 /  
$ python setup.py install
```

Lembre-se de verificar a versão mais recente do PyMySQL no site e atualizar o número da versão na primeira linha deste comando, conforme necessário.

Após a instalação, você deve ter acesso ao pacote PyMySQL automaticamente, e enquanto seu servidor MySQL local estiver em execução, você deve ser capaz de executar o seguinte script com sucesso (lembre-se de adicionar a senha root para seu banco de dados):

```
importar pymysql  
con = pymysql . conectar ( hospedeiro = '127.0.0.1' , unix_socket = '/tmp/mysql.sock' ,  
                           do utilizador = 'raiz' , senha = Nenhum , db = 'mysql' )  
cur = con . cursor ()  
cur . executar ( "USE raspagem" )
```

```
cur . executar ( "SELEÇÃO * DAS páginas ONDE id = 1" )
impressão ( cur . fetchone ())
cur . Fechar ()
con . Fechar ()
```

Existem dois novos tipos de objetos em ação neste exemplo: o objeto Connection (conn), e o objeto Cursor (cur).

O modelo de conexão / cursor é comumente usado na programação de banco de dados, embora alguns usuários possam achar difícil diferenciar os dois a princípio. A conexão é responsável por, bem, conectar-se ao banco de dados, é claro, mas também enviar as informações do banco de dados, lidar com rollbacks (quando uma consulta ou conjunto de consultas precisa ser abortada e o banco de dados precisa retornar ao seu estado anterior) e criando novos objetos de cursor.

Uma conexão pode ter vários cursores. Um cursor acompanha certas *Estado* informações, como qual banco de dados está usando. Se você tiver vários bancos de dados e precisar gravar informações em todos eles, poderá ter vários cursores para lidar com isso. Um cursor também contém os resultados da última consulta que executou. Chamando funções no cursor, como cur.fetchone (), você pode acessar essas informações.

É importante que o cursor e a conexão sejam fechados após terminar de usá-los. Não fazer isso pode resultar em *vazamentos de conexão*, um acúmulo de conexões não fechadas que não estão mais sendo usadas, mas o software não consegue fechar porque tem a impressão de que você ainda pode usá-las. Esse é o tipo de coisa que desativa os bancos de dados o tempo todo, então lembre-se de fechar suas conexões!

A coisa mais comum que você provavelmente desejará fazer, no início, é ser capaz de armazenar seus resultados de scraping em um banco de dados. Vamos dar uma olhada em como isso poderia ser feito, usando um exemplo anterior: o raspador da Wikipedia.

Lidar com texto Unicode pode ser difícil durante o web scraping. Por padrão, o MySQL não suporta Unicode. Felizmente, você pode ativar esse recurso (lembre-se de que isso aumentará o tamanho do seu banco de dados). Como estamos fadados a encontrar uma variedade de caracteres coloridos na Wikipedia, agora é um bom momento para dizer ao seu banco de dados para esperar algum Unicode:

```
ALTER DATABASE raspagem PERSONAGEM CONJUNTO = utf8mb4 COLLATE = utf8mb4_unicode_ci ;
ALTERAR A TABELA Páginas CONVERTER EM PERSONAGEM CONJUNTO utf8mb4 COLLATE utf8mb4_unicode_ci ;
ALTERAR A TABELA Páginas MUDANÇA título VARCHAR ( 200 ) PERSONAGEM CONJUNTO utf8mb4 COLLATE
utf8mb4_unicode_ci ;
ALTERAR A TABELA Páginas MUDANÇA conteúdo de conteúdo VARCHAR ( 10.000 ) PERSONAGEM CONJUNTO utf8mb4 CO LLATE utf8mb4_unicode_ci ;
```

Essas quatro linhas mudam o seguinte: o conjunto de caracteres padrão para o banco de dados, para a tabela e para ambas as colunas, de utf8mb4 (ainda tecnicamente Unicode, mas com suporte notoriamente terrível para a maioria dos caracteres Unicode) para utf8mb4_unicode_ci.

Você saberá que foi bem-sucedido se tentar inserir alguns tremas ou caracteres do mandarim no título ou conteúdo no banco de dados e é bem-sucedido sem erros.

Agora que o banco de dados está preparado para aceitar uma ampla variedade de tudo o que a Wikipedia pode oferecer, você pode executar o seguinte:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar data hora
importar aleatória
importar pymysql

con = pymysql . conectar ( hospedeiro = '127.0.0.1' , unix_socket = '/tmp/mysql.sock' ,
                           do utilizador = 'raiz' , senha = Nenhum , db = 'mysql' , charset = 'utf8' )
cur = con . cursor ()
cur . executar ( "USE raspagem" )

aleatória . semente ( data hora . data hora . agora () )

def loja ( título , conteúdo ):
    cur . executar ( "INSERIR NAS páginas (título, conteúdo) VALORES (%s , "
                     "%s)" , ( título , conteúdo ) )
    cur . conexão . comprometer ()

def getLinks ( articleUrl ):
    html = urlopen ( "http://en.wikipedia.org" + articleUrl )
    bsObj = BeautifulSoup ( html )
    título = bsObj . encontrar ( "h1" ) . encontrar ( "periodo" ) . get_text ()
    conteúdo = bsObj . encontrar ( "div" , { "Eu iria" : "mw-content-text" }) . encontrar ( "p" ) . get_text ()
    loja ( título , conteúdo )
    Retorna bsObj . encontrar ( "div" , { "Eu iria" : "bodyContent" }) . encontrar tudo ( "uma" ,
        href = re . compilar ( "^( / wiki / )(?!:).*$" ))

links = getLinks ( "/ wiki / Kevin_Bacon" )
experimentar :
    enquanto len ( links ) > 0 :
        novo artigo = links [ aleatória . Randint ( 0 , len ( links ) - 1 )] . attrs [ "href" ]
        impressão ( novo artigo )
        links = getLinks ( novo artigo )
finalmente :
    cur . Fechar ()
    con . Fechar ()
```

Há algumas coisas a serem observadas aqui: primeiro, "charset = 'utf8'" é adicionado à string de conexão do banco de dados. Isso diz à conexão que ela deve enviar todas as informações ao banco de dados como UTF-8 (é, é claro, o banco de dados já deve estar configurado para lidar com isso).

Em segundo lugar, observe a adição de um loja função. Isso leva em duas variáveis de string, título e conteúdo, e os adiciona a um INSERIR declaração que é executada pelo

cursor e confirmado pela conexão do cursor. Este é um excelente exemplo da separação do cursor e da conexão; enquanto o cursor possui algumas informações armazenadas sobre o banco de dados e seu próprio contexto, ele precisa operar através da conexão para enviar informações de volta ao banco de dados e inserir algumas informações.

Por último, você verá que um finalmente instrução foi adicionada ao loop principal do programa, na parte inferior do código. Isso irá garantir que, independentemente de como o programa é interrompido ou das exceções que podem ser lançadas durante sua execução (e, claro, porque a Web é confusa, você deve sempre assumir que exceções serão lançadas), o cursor e o a conexão será encerrada imediatamente antes do término do programa. É uma boa ideia incluir um tente ... finalmente sempre que você estiver acessando a Web e tiver uma conexão de banco de dados aberta.

Embora PyMySQL não seja um pacote enorme, há um bom número de funções úteis que este livro simplesmente não pode acomodar. Confira esta documentação <http://bit.ly/1KHzoga>.

Técnicas de banco de dados e boas práticas

Existem pessoas que passam a carreira inteira estudando, ajustando e inventando bancos de dados. Eu não sou um deles, e este não é esse tipo de livro. No entanto, como acontece com muitos assuntos em ciência da computação, existem alguns truques que você pode aprender rapidamente para, pelo menos, tornar seus bancos de dados suficientes e suficientemente rápidos para a maioria dos aplicativos.

Primeiro, com muito poucas exceções, sempre adicione Eu iria colunas às suas tabelas. Todas as tabelas no MySQL devem ter pelo menos uma chave primária (a coluna chave que o MySQL classifica), para que o MySQL saiba como ordená-la, e muitas vezes pode ser difícil escolher essas chaves de forma inteligente. O debate sobre se deve usar um criado artificialmente Eu iria coluna para esta chave ou algum atributo exclusivo, como nome do usuário assola cientistas de dados e engenheiros de software há anos, embora eu tenha a tendência de me inclinar para o lado da criação Eu iria colunas. As razões para fazer isso de uma forma ou de outra são complicadas, mas para sistemas não empresariais, você deve sempre usar um Eu iria coluna como uma chave primária autoincrementada.

Em segundo lugar, use a indexação inteligente. Um dicionário (como o livro, não o objeto Python) é uma lista de palavras indexadas em ordem alfabética. Isso permite pesquisas rápidas sempre que você precisar de uma palavra, desde que você saiba como ela é escrita. Você também pode imaginar um dicionário organizado em ordem alfabética pela definição da palavra. Isso não seria tão útil, a menos que você estivesse jogando algum jogo estranho de *Perigo!* onde uma definição foi apresentada e você precisava encontrar a palavra. Mas, no mundo das pesquisas de banco de dados, esse tipo de situação acontece. Por exemplo, você pode ter um campo em seu banco de dados que frequentemente consultará:

```
> SELECIONE * DO dicionário WHERE definição = "Um pequeno animal peludo que diz miau";
```

Eu iria	palavra definição
200 gato	Um pequeno animal peludo que diz miau

1 linha no conjunto (0,00 s)

Você pode muito bem querer adicionar uma chave adicional a esta tabela (além da chave presumivelmente já colocada no Eu iria) para tornar as pesquisas na coluna de definição mais rápidas. Infelizmente, adicionar indexação adicional requer mais espaço para o novo índice, bem como algum tempo de processamento adicional ao inserir novas linhas. Para tornar isso um pouco mais fácil, você pode dizer ao MySQL para indexar apenas os primeiros caracteres no valor da coluna. Este comando cria um índice nos primeiros 16 caracteres no campo de definição:

CRIAR ÍNDICE definição EM dicionário (Eu iria , definição (16));

Este índice tornará suas pesquisas muito mais rápidas ao pesquisar palavras por sua definição completa, e também não adicionará muito espaço extra e tempo de processamento inicial.

Sobre o assunto do tempo de consulta versus tamanho do banco de dados (um dos atos de equilíbrio fundamentais na engenharia de banco de dados), um dos erros comuns cometidos, especialmente com web scraping, onde muitas vezes você tem grandes quantidades de dados de texto natural, é armazenar muitos dados repetidos . Por exemplo, digamos que você queira medir a frequência de certas frases que aparecem em sites. Essas frases podem ser encontradas em uma determinada lista ou geradas automaticamente por meio de algum algoritmo de análise de texto. Você pode ficar tentado a armazenar os dados como algo assim:

Field	Tipo	Nulo	Chave	Padrão	Extra
Eu iria	int (11)	NÃO	PRI	NULO	auto_increment
url	varchar (200)	SIM		NULO	
frase	varchar (200)	SIM		NULO	

Isso adiciona uma linha ao banco de dados cada vez que você encontra uma frase em um site e registra o URL em que foi encontrada. No entanto, ao dividir os dados em três tabelas separadas, você pode reduzir enormemente seu conjunto de dados:

```
> DESCRIEVER frases
+-----+-----+-----+-----+
| Field | Tipo          | Nulo | Chave | Padrão | Extra |
+-----+-----+-----+-----+
| Eu iria    | int (11)      | NÃO   | PRI    | NULO    | auto_increment |
| frase | varchar (200) | SIM   |        | NULO    |           |
+-----+-----+-----+-----+

```

```
> DESCRIBE urls
+-----+-----+-----+-----+
| Field | Tipo          | Nulo | Chave | Padrão | Extra |
+-----+-----+-----+-----+
| Eu iria    | int (11)      | NÃO   | PRI    | NULO    | auto_increment |
| url       | varchar (200) | SIM   |        | NULO    |           |
+-----+-----+-----+-----+

```

```
> DESCRIBE foundInstances
+-----+-----+-----+-----+
| Campo     | Tipo          | Nulo | Chave | Padrão | Extra |
+-----+-----+-----+-----+
| Eu iria    | int (11) | NÃO   | PRI    | NULO    | auto_increment |
| urlId     | int (11) | SIM   |        | NULO    |           |
| fraseId   | int (11) | SIM   |        | NULO    |           |
| ocorrências | int (11) | SIM   |        | NULO    |           |
+-----+-----+-----+-----+

```

Embora as definições da tabela sejam maiores, você pode ver que a maioria das colunas são apenas campos de ID de inteiros. Eles ocupam muito menos espaço. Além disso, o texto completo de cada URL e frase é armazenado exatamente uma vez.

A menos que você instale algum pacote de terceiros ou mantenha registros meticulosos, pode ser impossível dizer quando um dado foi adicionado, atualizado ou removido de seu banco de dados. Dependendo do espaço disponível para seus dados, a frequência das alterações e a importância de determinar quando essas alterações aconteceram, você pode querer considerar a manutenção de vários carimbos de data / hora: “criado”, “atualizado” e “excluído”.

“Seis Graus” no MySQL

Dentro [Capítulo 3](#), Eu apresentei o problema dos “Seis graus da Wikipedia”, em que o objetivo é encontrar a conexão entre dois artigos da Wikipedia por meio de uma série de links (ou seja, encontrar uma maneira de ir de um artigo da Wikipedia a outro apenas clicando em links de uma página para a próxima).

Para resolver esse problema, é necessário não apenas construir bots que possam rastrear o site (o que já fizemos), mas armazenar as informações de uma maneira arquitetonicamente sólida para facilitar a análise de dados posteriormente.

Autoincrementado Eu iria colunas, carimbos de data / hora e várias tabelas: todos eles entram em jogo aqui. Para descobrir como armazenar melhor essas informações, você precisa pensar

abstratamente. Um link é simplesmente algo que conecta a Página A à Página B. Ele poderia facilmente conectar a Página B à Página A, mas seria um link separado. Podemos identificar um link de forma exclusiva, dizendo: "Existe um link na página A, que se conecta à página B. Ou seja, INSIRA DENTRO DE links (fromPageId, toPageId) VALORES (A, B); (onde "A" e "B" são os IDs exclusivos para as duas páginas).

Um sistema de duas tabelas projetado para armazenar páginas e links, junto com datas de criação e IDs exclusivos, pode ser construído da seguinte maneira:

```
CRIAR A TABELA `wikipedia` . `páginas` (
    `id` INT NÃO NULO INCREMENTO AUTOMÁTICO ,
    `url` VARCHAR( 255 ) NÃO NULO ,
    `criado` TIMESTAMP NÃO NULO DEFAULT CURRENT_TIMESTAMP ,
    CHAVE PRIMÁRIA ( ` id` );
```

```
CRIAR A TABELA `wikipedia` . `links` (
    `id` INT NÃO NULO INCREMENTO AUTOMÁTICO ,
    `fromPageId` INT NULO ,
    `toPageId` INT NULO ,
    `criado` TIMESTAMP NÃO NULO DEFAULT CURRENT_TIMESTAMP ,
    CHAVE PRIMÁRIA ( ` id` );
```

Observe que, ao contrário dos rastreadores anteriores que imprimem o título da página, nem mesmo estou armazenando o título da página na tabela de páginas. Por que é que? Bem, gravar o título da página requer que você realmente visite a página para recuperá-lo. Se quisermos construir um rastreador da web eficiente para preencher essas tabelas, queremos ser capazes de armazenar a página, bem como os links para ela, mesmo que ainda não tenhamos necessariamente visitado a página.

Claro, embora isso não seja verdade para todos os sites, o bom dos links e títulos de páginas da Wikipedia é que um pode ser transformado no outro por meio de uma simples manipulação. Por exemplo, http://en.wikipedia.org/wiki/Monty_Python indica que o título da página é "Monty Python".

O seguinte irá armazenar todas as páginas da Wikipedia que têm um "número Bacon" (o número de links entre ele e a página de Kevin Bacon, inclusive) de 6 ou menos: de
urllib.request import urlopen.

```
de bs4 importar BeautifulSoup
importar re
importar pymysql

con = pymysql . conectar ( hospedeiro = '127.0.0.1' , unix_socket = '/tmp/mysql.sock' , do utilizador =
    'raiz' , senha = Nenhum , db = 'mysql' , charset = 'utf8' )
cur = con . cursor ()
cur . executar ( "USE wikipedia" )

def insertPageIfNotExists ( url ):
    cur . executar ( "SELECIONE * DAS páginas ONDE url = % s " , ( url ))
    E se cur . Contagem de linhas == 0 :
        cur . executar ( "INSERT INTO páginas (url) VALUES ( % s ) " , ( url ))
```

```

        con . comprometer ()
        Retorna cur . lastrowid
    outro :
        Retorna cur . fetchone () [ 0 ]

def insertLink ( fromPageld , toPageld ):
    cur . executar ( "SELECT * FROM links WHERE fromPageld = % s AND toPageld = % s " ,
                    ( int ( fromPageld ), int ( toPageld )))

    E se cur . Contagem de linhas == 0 :
        cur . executar ( "INSERT INTO links (fromPageld, toPageld) VALUES ( % s , % s ) " ,
                        ( int ( fromPageld ), int ( toPageld )))

    con . comprometer ()

Páginas = conjunto ()
def getLinks ( URL da página , recursionLevel ):
    global Páginas
    E se recursionLevel > 4 :
        Retorna ;
    pageld = insertPageIfNotExists ( URL da página )
    html = urlopen ( "http://en.wikipedia.org" + URL da página )
    bsObj = BeautifulSoup ( html )
    para ligação dentro bsObj . encontrar tudo ( "uma" ,
                                                href = ré . compilar ( "^ (/ wiki /)(?!:.)*$" )):
        insertLink ( pageld ,
                     insertPageIfNotExists ( ligação . attrs [ 'href' ]))

    E se ligação . attrs [ 'href' ] não em Páginas :
        # Encontramos uma nova página, adicione-a e pesquise os links
        nova página = ligação . attrs [ 'href' ]
        Páginas . adicionar ( nova página )
        getLinks ( nova página , recursionLevel + 1 )

    getLinks ( "/ wiki / Kevin_Bacon" , 0 )
    cur . Fechar ()
    con . Fechar ()

```

A recursão é sempre uma coisa complicada de implementar em um código projetado para ser executado por um longo tempo. Neste caso, um recursionLevel variável é passada para o getLinks função, que rastreia quantas vezes aquela função foi repetida (cada vez que é chamada, recursionLevel é incrementado). Quando recursionLevel atingir 5, a função retorna automaticamente sem pesquisar mais. Esse limite garante que nunca ocorra um estouro de pilha.

Lembre-se de que esse programa provavelmente levaria dias para ser concluído. Embora eu realmente o tenha executado, meu banco de dados contém uma mera fração das páginas com um número de Kevin Bacon de 6 ou menos, para o bem dos servidores da Wikipedia. No entanto, isso é suficiente para nossa análise de como encontrar caminhos entre os artigos da Wikipedia vinculados.

Para a continuação deste problema e a solução final, consulte [Capítulo 8](#) na resolução de problemas de gráfico direcionado.

O email

Assim como as páginas da web são enviadas por HTTP, o e-mail é enviado por SMTP (Simple Mail Transfer Protocol). E, assim como você usa um cliente de servidor da Web para enviar páginas da Web por HTTP, os servidores usam vários clientes de email, como Sendmail, Postfix ou Mailman, para enviar e receber emails.

Embora enviar e-mail com Python seja relativamente fácil de fazer, isso requer que você tenha acesso a um servidor executando SMTP. Configurar um cliente SMTP em seu servidor ou máquina local é complicado e está fora do escopo deste livro, mas existem muitos recursos excelentes para ajudar nesta tarefa, particularmente se você estiver executando Linux ou Mac OS X.

Nos exemplos de código a seguir, assumirei que você está executando um cliente SMTP localmente. (Para modificar este código para um cliente SMTP remoto, basta alterar localhost para o endereço do seu servidor remoto.)

Enviar um e-mail com Python requer apenas nove linhas de código:

```
importar smtplib  
de email.mime.text importar MIMEText  
  
msg = MIMEText ( "O corpo do e-mail está aqui" )  
  
msg [ 'Sujeito' ] = "Um alerta de email"  
msg [ 'De' ] = " ryan@pythonscraping.com "  
msg [ 'Para' ] = " webmaster@pythonscraping.com "  
  
s = smtplib . SMTP ( 'localhost' )  
s . enviar mensagem ( msg )  
s . Sair ( )
```

Python contém dois pacotes importantes para enviar e-mails: *smtp/lib* e *o email*.

O módulo de e-mail do Python contém funções de formatação úteis para a criação de “pacotes” de e-mail a serem enviados. o *MIMEText* objeto, usado aqui, cria um e-mail vazio formatado para transferência com o protocolo MIME (Multipurpose Internet Mail Extensions) de baixo nível, através do qual as conexões SMTP de alto nível são feitas. o *MIMEText* objeto, msg.

contém endereços de e-mail de / para, bem como um corpo e um cabeçalho, que o Python usa para criar um e-mail formatado corretamente.

O pacote *smtplib* contém informações para lidar com a conexão com o servidor. Assim como uma conexão com um servidor MySQL, esta conexão deve ser interrompida toda vez que for criada, para evitar a criação de muitas conexões.

Esta função básica de e-mail pode ser estendida e tornada mais útil ao incluí-la em uma função:

```
importar smtplib  
de email.mime.text importar MIMEText
```

```

de bs4 importar BeautifulSoup
de urllib.request importar urlopen
importar Tempo

def enviar correio ( sujeito , corpo ):
    msg = MIMEText ( corpo )
    msg [ 'Sujeito' ] = sujeito
    msg [ 'De' ] = " christmas_alerts@pythonscraping.com "
    msg [ 'Para' ] = " ryan@pythonscraping.com "

    s = smtplib . SMTP ( 'localhost' )
    s . enviar mensagem ( msg )
    s . Sair ()

bsObj = BeautifulSoup ( urlopen ( "https://isitchristmas.com/" ))
enquanto ( bsObj . encontrar ( "uma" , { "Eu iria" : "responda" }) . attrs [ 'título' ] == "NÃO" ):
    impressão ( " Ainda não é Natal. ")
    Tempo . dormir ( 3600 )
bsObj = BeautifulSoup ( urlopen ( "https://isitchristmas.com/" ))
enviar correio ( "É Natal!" ,
    "De acordo com http://itischristmas.com, é Natal!" )

```

Este script específico verifica o site <https://isitchristmas.com> (cuja principal característica é um gigante “SIM” ou “NÃO”, dependendo do dia do ano) uma vez por hora. Se vir qualquer coisa diferente de um “NÃO”, ele enviará um e-mail alertando que é Natal.

Embora este programa em particular possa não parecer muito mais útil do que um calendário pendurado na parede, ele pode ser ligeiramente ajustado para fazer uma variedade de coisas extremamente úteis. Ele pode enviar alertas por e-mail em resposta a interrupções no site, falhas de teste ou até mesmo a aparência de um produto fora de estoque que você está esperando na Amazon - nada do que seu calendário de parede pode fazer.

Lendo Documentos

É tentador pensar na Internet principalmente como uma coleção de sites baseados em texto intercalados com o novo conteúdo multimídia da web 2.0 que pode ser ignorado para fins de web scraping. No entanto, isso ignora o que a Internet é mais fundamentalmente: um veículo independente de conteúdo para a transmissão de arquivos.

Embora a Internet exista de uma forma ou de outra desde o final dos anos 1960, o HTML não foi lançado até 1992. Até então, a Internet consistia principalmente em e-mail e transmissão de arquivos; o conceito de páginas da web como as conhecemos hoje realmente não existia. Em outras palavras, a Internet não é uma coleção de arquivos HTML. É uma coleção de informações, com arquivos HTML geralmente sendo usados como um quadro para exibi-la. Sem poder ler uma variedade de tipos de documentos, incluindo texto, PDF, imagens, vídeo, e-mail e muito mais, estamos perdendo uma grande parte dos dados disponíveis.

Este capítulo aborda como lidar com documentos, quer você esteja baixando-os para uma pasta local ou lendo-os e extraíndo dados. Também veremos como lidar com vários tipos de codificação de texto, que podem tornar possível até mesmo ler páginas HTML em idiomas estrangeiros.

Codificação de Documento

A codificação de um documento informa aos aplicativos - sejam eles o sistema operacional do seu computador ou seu próprio código Python - como lê-lo. Esta codificação geralmente pode ser deduzida de sua extensão de arquivo, embora esta extensão de arquivo não seja obrigatória por sua codificação. Eu poderia, por exemplo, salvar *myImage.jpg* Como *myImage.txt* sem problemas - pelo menos até meu editor de texto tentar abri-lo. Felizmente, essa situação é rara, e a extensão de arquivo de um documento geralmente é tudo que você precisa saber para lê-lo corretamente.

Em um nível fundamental, todos os documentos são codificados em 0s e 1s. Além disso, existem algoritmos de codificação que definem coisas como "quantos bits por caractere" ou

"Quantos bits representam a cor de cada pixel" (no caso de arquivos de imagem). Além disso, você pode ter uma camada de compactação ou algum algoritmo de redução de espaço, como é o caso dos arquivos PNG.

Embora lidar com arquivos não HTML possa parecer intimidante no início, tenha certeza de que, com a biblioteca certa, o Python estará devidamente equipado para lidar com qualquer formato de informação que você desejar. A única diferença entre um arquivo de texto, um arquivo de vídeo e um arquivo de imagem é como seus 0s e 1s são interpretados. Neste capítulo, abordarei vários tipos de arquivos comumente encontrados: texto, PDFs, PNGs e GIFs.

Texto

É um tanto incomum ter arquivos armazenados como texto simples online, mas é popular entre os sites básicos ou da "velha escola" ter grandes repositórios de arquivos de texto. Por exemplo, a Internet Engineering Task Force (IETF) armazena todos os seus documentos publicados como HTML, PDF e arquivos de texto (consulte <https://www.ietf.org/rfc/rfc1149.txt> como um exemplo). A maioria dos navegadores exibirá esses arquivos de texto muito bem e você poderá copiá-los sem problemas.

Para a maioria dos documentos de texto básicos, como o arquivo de prática localizado em <http://www.pythonscraping.com/pages/warandpeace/chapter1.txt>, você pode usar o seguinte método:

```
de urllib.request importar urlopen
(
    "http://www.pythonscraping.com/pages/warandpeace/chapter1.txt")
impressão ( textPage . ler ())
```

Normalmente, quando recuperamos uma página usando urlopen, nós o transformamos em um objeto BeautifulSoup para analisar o HTML. Nesse caso, podemos ler a página diretamente. Transformar em um objeto BeautifulSoup, embora perfeitamente possível, seria contraproducente - não há HTML para analisar, então a biblioteca seria inútil. Depois que o arquivo de texto é lido como uma string, você apenas tem que analisá-lo como faria com qualquer outra string lida no Python. A desvantagem aqui, é claro, é que você não tem a capacidade de usar tags HTML como dicas de contexto, apontando na direção do texto que você realmente precisa, em vez do texto que você não deseja. Isso pode representar um desafio quando você está tentando extrair certas informações de arquivos de texto.

Codificação de texto e Internet global

Lembra quando eu disse que uma extensão de arquivo era tudo que você precisava para ler um arquivo corretamente? Bem, curiosamente, essa regra não se aplica ao mais básico de todos os documentos: o *.TXT* Arquivo.

Nove em cada 10 vezes, a leitura de texto usando os métodos descritos anteriormente funcionará perfeitamente. No entanto, lidar com texto na Internet pode ser um negócio complicado.

A seguir, cobriremos os fundamentos da codificação em inglês e em idiomas estrangeiros, de ASCII a Unicode a ISO, e como lidar com eles.

Uma breve visão geral dos tipos de codificação

No início da década de 1990, uma organização sem fins lucrativos chamada The Unicode Consortium tentou criar um codificador de texto universal estabelecendo codificações para cada caractere que precisa ser usado em qualquer documento de texto, em qualquer idioma. O objetivo era incluir tudo, desde o alfabeto latino em que este livro foi escrito, ao cirílico (кириллица) e pictogramas chineses (象形), símbolos matemáticos e lógicos (•, ≥), e até mesmo emoticons e símbolos "diversos", como o sinal de risco biológico (✿) e o símbolo da paz (⊕).

O codificador resultante, como você já deve saber, foi denominado UTF-8, que significa, confusamente, "você Conjunto de caracteres universais - Transformação Format 8 mordeu". É um equívoco comum que UTF-8 armazene todos os caracteres em 8 bits. No entanto, o "8 bits" refere-se ao menor tamanho que um caractere requer para ser exibido, não o máximo

Tamanho. (Se todos os caracteres em UTF-8 fossem realmente armazenados em 8 bits, isso permitiria apenas 2⁸, ou 256 caracteres possíveis. Claramente, não há espaço suficiente para tudo, desde um pictograma chinês a um símbolo da paz.)

Na realidade, cada caractere em UTF-8 começa com um indicador que diz "apenas um byte é usado para codificar esse caractere" ou "os próximos dois bytes codificam um único caractere", com até quatro bytes sendo possíveis. Como esses quatro bytes também contêm as informações sobre quantos bytes estão sendo usados para codificar o caractere, os 32 bits completos ($32 = 4 \text{ bytes} \times 8 \text{ bits por byte}$) não são usados; em vez disso, podem ser usados até 21 bits de informação, para um total de 2.097.152 caracteres possíveis, dos quais 1.114.112 são usados atualmente.

Embora o Unicode tenha sido uma dádiva de Deus para muitos aplicativos, alguns hábitos são difíceis de quebrar e o ASCII ainda é uma escolha popular para muitos usuários.

ASCII, um padrão de codificação de texto usado desde 1960, usa 7 bits para codificar cada um de seus caracteres, para um total de 2⁷, ou 128 caracteres. Isso é suficiente para o alfabeto latino (maiúsculas e minúsculas), pontuação e, essencialmente, todos os caracteres encontrados no teclado de um falante de inglês comum.

Em 1960, é claro, a diferença entre armazenar um arquivo de texto com 7 bits por caractere e 8 bits por caractere era significativa porque o armazenamento era muito caro. Os cientistas da computação da época discutiam se um bit extra deveria ser adicionado para a conveniência de ter um bom número redondo em comparação com a praticidade de arquivos que exigem menos espaço de armazenamento. No final, venceram 7 bits. No entanto, na computação moderna, cada sequência de 7 bits é preenchida com um "0" extra no início, deixando-nos com o pior dos dois mundos - arquivos 14% maiores e a falta de flexibilidade de apenas 128 caracteres.

¹ Esse pedaço de "enchimento" voltará para nos assombrar com os padrões ISO um pouco mais tarde.

Quando o UTF-8 foi projetado, os criadores decidiram usar este "bit de preenchimento" em documentos ASCII a seu favor, declarando todos os bytes começando com um "0" para indicar que apenas um byte é usado no caractere e codificando os dois esquemas para ASCII e UTF-8 idênticos. Portanto, os seguintes caracteres são válidos em UTF-8 e ASCII:

01000001 - A
01000010 - B
01000011 - C

E os seguintes caracteres são válidos apenas em UTF-8 e serão processados como "não imprimíveis" se o documento for interpretado como um documento ASCII:

11000011 10000000 - À
11000011 10011111 - ß
11000011 10100111 - ç

Além de UTF-8, existem outros padrões UTF, como UTF-16, UTF-24 e UTF-32, embora documentos codificados nesses formatos raramente sejam encontrados, exceto em circunstâncias incomuns, que estão fora do escopo deste livro.

O problema com todos os padrões Unicode, é claro, é que qualquer documento escrito em uma única língua estrangeira é muito maior do que deveria ser. Embora seu idioma possa exigir o uso de apenas 100 ou mais caracteres, você precisará de pelo menos 16 bits para cada caractere, em vez de apenas 8 bits, como é o caso do ASCII específico do inglês. Isso torna os documentos de texto em idioma estrangeiro cerca de duas vezes o tamanho dos documentos de texto em inglês, pelo menos para idiomas estrangeiros que não usam o conjunto de caracteres latinos.

ISO resolve este problema criando codificações específicas para cada idioma. Como o Uni-code, ele tem as mesmas codificações que o ASCII, mas usa o bit 0 de "preenchimento" no início de cada caractere para permitir a criação de caracteres especiais para todas as línguas que os exigem. Isso funciona melhor para idiomas europeus que também dependem fortemente do alfabeto latino (que permanecem nas posições 0-127 na codificação), mas requerem alguns caracteres especiais adicionais. Isso permite que o ISO-8859-1 (projeto para o alfabeto latino) tenha símbolos como frações (por exemplo, $\frac{1}{2}$) ou o símbolo de copyright (©).

Outros conjuntos de caracteres ISO, como ISO-8859-9 (turco), ISO-8859-2 (alemão, entre outros idiomas) e ISO-8859-15 (francês, entre outros idiomas) também podem ser encontrados com alguma regularidade.

Embora a popularidade dos documentos codificados por ISO tenha diminuído nos últimos anos, cerca de 9% dos sites na Internet ainda são codificados com algum tipo de ISO,² tornando essencial conhecer e verificar as codificações antes de copiar um site.

² De acordo com http://w3techs.com/technologies/history_overview/character_encoding, que usa rastreadores da web para coletar esses tipos de estatísticas.

Codificações em ação

Na seção anterior, usamos as configurações padrão para urlopen para ler. *TXT* documentos que você pode encontrar na Internet. Isso funciona muito bem para a maioria dos textos em inglês. No entanto, assim que você encontrar russo, árabe ou mesmo uma palavra como currículo, poderá ter problemas.

Pegue o seguinte código, por exemplo:

```
de urllib.request importar urlopen
textPage = urlopen
(
    "http://www.pythonscraping.com/pages/warandpeace/chapter1-ru.txt")
impressão (textPage . ler ())
```

Lê-se no primeiro capítulo do original *Guerra e Paz* (escrito em russo e francês) e imprime na tela. Este texto da tela lê, em parte:

b \| xd0 \| xa7 \| xd0 \| x90 \| xd0 \| xa1 \| xd0 \| xa2 \| xd0 \| xac \| xd0 \| x9f \| xd0 \| x95 \| xd0 \| xa0 \| xd0 \| x92 \| xd0 \| x90 \| xd0 \| x \| nl \| n \| n \| xe2 \| x80 \| x94 Eh bien, mon principe.

Além disso, visitar esta página na maioria dos resultados de navegadores em rabiscos (consulte Figura 6-1).

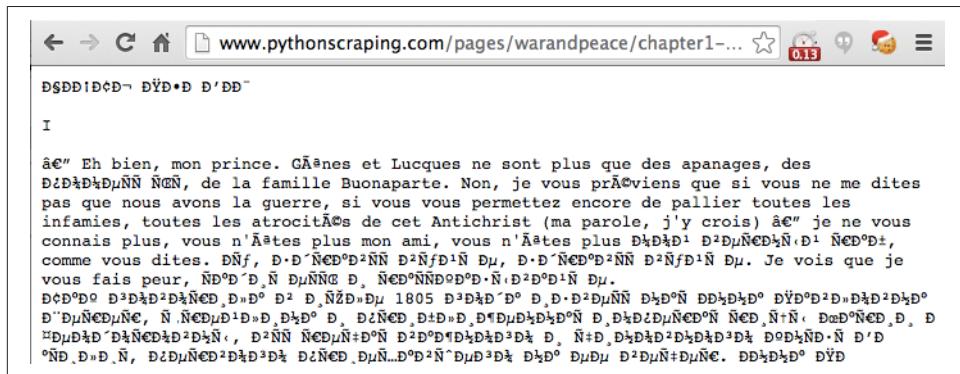


Figura 6-1. Texto em francês e cirílico codificado em ISO-8859-1, a codificação de documento de texto padrão em muitos navegadores

Mesmo para falantes nativos de russo, isso pode ser um pouco difícil de entender. O problema é que o Python está tentando ler o documento como um documento ASCII, enquanto o navegador está tentando lê-lo como um documento codificado ISO-8859-1. Nenhum dos dois, é claro, percebe que é um documento UTF-8.

Podemos definir explicitamente a string como UTF-8, o que formata corretamente a saída em caracteres cirílicos:

```
de urllib.request importar urlopen

textPage = urlopen (
    "http://www.pythonscraping.com/pages/warandpeace/chapter1-ru.txt",
    impressão ( str ( textPage . ler (), 'utf-8' ))
```

Usando este conceito em BeautifulSoup e Python 3.x tem esta aparência:

```
html = urlopen( "http://en.wikipedia.org/wiki/Python_(programming_language)" )
bsObj = BeautifulSoup( html )
conteúdo = bsObj . encontrar ( "div" , { "Eu iria" : "mw-content-text" }) . get_text ()
conteúdo = bytes ( conteúdo , "UTF-8" )
conteúdo = conteúdo . decodificar ( "UTF-8" )
```

Você pode ficar tentado a usar a codificação UTF-8 para cada raspador da web que escrever. Afinal, o UTF-8 também manipulará caracteres ASCII sem problemas. No entanto, é importante lembrar os 9% dos sites que usam alguma versão da codificação ISO. Infelizmente, no caso de documentos de texto, é impossível determinar concretamente que codificação um documento possui. Existem algumas bibliotecas que podem examinar o documento e fazer uma melhor suposição (usando um pouco de lógica para perceber que “Ñ € Ð ° ÑÑÐ°Ð ° Ð · Ñ” provavelmente não é uma palavra), mas muitas vezes está errado.

Felizmente, no caso de páginas HTML, a codificação geralmente está contida em uma tag encontrada no < cabeçá> seção do site. A maioria dos sites, principalmente sites em inglês, tem a tag:

```
<meta charset = "utf-8" />
```

Considerando que a [Site da European Computer Manufacturers Association](#) tem esta tag:³

```
<META HTTP-EQUIV = "Tipo de conteúdo" CONTEÚDO = "text / html; charset = iso-8859-1" >
```

Se você planeja fazer muito scraping na web, especialmente em sites internacionais, pode ser sábio procurar essa metatag e usar a codificação que ela recomenda ao ler o conteúdo da página.

CSV

Ao fazer web scraping, é provável que você encontre um arquivo CSV ou um colega de trabalho que gosta de dados formatados dessa forma. Felizmente, Python tem um fantástico [biblioteca](#) para ler e gravar arquivos CSV. Embora esta biblioteca seja capaz de lidar com muitas variações de CSV, vou me concentrar principalmente no formato padrão. Se você tiver um caso especial que precise tratar, consulte a documentação!

Lendo arquivos CSV

A biblioteca csv do Python é voltada principalmente para trabalhar com arquivos locais, pressupondo que os dados CSV de que você precisa estão armazenados em sua máquina. Infelizmente, nem sempre é esse o caso, especialmente quando você está copiando na web. Existem várias maneiras de contornar isso:

³ Ecma foi um dos contribuidores originais do padrão ISO, então não é surpresa que seu site seja codificado com um sabor de ISO.

- Baixe o arquivo localmente manualmente e aponte o Python para o local do arquivo local
- Escreva um script Python para baixar o arquivo, lê-lo e (opcionalmente) excluí-lo após a recuperação
- Recupere o arquivo como uma string da Web e envolva a string em um StringIO objeto para que se comporte como um arquivo

Embora as duas primeiras opções sejam viáveis, ocupar espaço no disco rígido com arquivos quando você poderia facilmente mantê-los na memória não é uma prática. É muito melhor ler o arquivo como uma string e envolvê-lo em um objeto que permita ao Python tratá-lo como um arquivo, sem nunca realmente salvá-lo. O script a seguir recupera um arquivo CSV da Internet (neste caso, uma lista de álbuns do Monty Python em <http://pythonscraping.com/files/MontyPythonAlbums.csv>) e imprime, linha por linha, no terminal:

```
de urllib.request importar urlopen
de io importar StringIO
importar csv

dados = urlopen( "http://pythonscraping.com/files/MontyPythonAlbums.csv" )
    . ler () . decodificar ( 'ascii' , 'ignorar' )
dataFile = StringIO ( dados )
csvReader = csv . leitor ( dataFile )

para linha dentro csvReader :
    impressão ( linha )
```

A saída é muito longa para imprimir por completo, mas deve ser semelhante a:

```
[ 'Nome' , 'Ano' ]
[ "Monty Python's Flying Circus" , '1970' ][ 'Outro registro do
Monty Python' , '1971' ][ "Registro anterior de Monty Python" , '1972'
]
...
```

Como você pode ver no exemplo de código, o objeto leitor retornado por csv.reader é iterável e composto de objetos de lista Python. Por causa disso, cada linha no csvReader objeto é acessível da seguinte maneira:

```
para linha dentro csvReader :
    impressão ( " O álbum \" + linha [ 0 ] + "\" foi lançado em " + str ( linha [ 1 ] ))
```

O que dá a saída:

```
O álbum "Nome" foi liberado dentro Ano o álbum "Monty Python's Flying Circus" foi
liberado dentro 1970
O álbum "Outro registro do Monty Python" foi liberado dentro 1971
O álbum "Registro anterior de Monty Python" foi liberado dentro 1972
...
```

Observe a primeira linha: O álbum 'Name' foi lançado no ano. Embora este possa ser um resultado fácil de ignorar ao escrever código de exemplo, você não quer que isso entre em seus dados no mundo real. Um programador inferior pode simplesmente pular a primeira linha do

a csvReader objeto, ou escrever em algum caso especial para manipulá-lo. Felizmente, há uma alternativa para o csv.reader função que cuida de tudo isso para você automaticamente. Entrar DictReader:

```
de urllib.request importar urlopen
de io importar StringIO
importar csv

dados = urlopen ( "http://pythonscraping.com/files/MontyPythonAlbums.csv" )
    . ler () . decodificar ( 'ascii' , 'ignorar' )
dataFile = StringIO ( dados )
dictReader = csv . DictReader ( dataFile )

impressão ( dictReader . nomes de campo )

para linha dentro dictReader :
    impressão ( linha )
```

csv.DictReader retorna os valores de cada linha no arquivo CSV como objetos de dicionário em vez de objetos de lista, com nomes de campos armazenados na variável nomes de dictReader.field e como chaves em cada objeto do Dicionário:

```
[Nome', 'Ano']
{'Nome': "Monty Python's Flying Circus", 'Ano': '1970'} {'Nome': 'Outro registro do Monty
Python', 'Ano': '1971'} {'Nome': "Registro anterior do Monty Python ", 'Ano': '1972'}
```

A desvantagem, claro, é que leva um pouco mais de tempo para criar, processar e imprimir esses DictReaders em oposição a csvReaders, mas a conveniência e usabilidade geralmente compensam a sobrecarga adicional.

PDF

Como usuário do Linux, conheço a dor de ser enviado um. docx arquivo que meu software não-Microsoft destrói e lutando para encontrar os codecs para interpretar algum novo formato de mídia da Apple. De certa forma, a Adobe foi revolucionária na criação de seu Portable Document Format em 1993. Os PDFs permitiam que usuários em diferentes plataformas visualizassem imagens e documentos de texto exatamente da mesma maneira, independentemente da plataforma em que estavam visualizando.

Embora o armazenamento de PDFs na Web seja um tanto ultrapassado (por que armazenar conteúdo em um formato estático de carregamento lento quando você poderia escrevê-lo como HTML?), Os PDFs permanecem onipresentes, especialmente ao lidar com formulários e arquivamentos oficiais.

Em 2009, um britânico chamado Nick Innes foi notícia quando solicitou informações públicas sobre o resultado de um teste de estudante ao Conselho da cidade de Buckinghamshire, que estava disponível sob a versão do Reino Unido da Lei de Liberdade de Informação. Após alguns

repetidos pedidos e recusas, ele finalmente recebeu a informação que estava procurando - na forma de 184 documentos PDF.

Embora Innes tenha persistido e eventualmente recebido um banco de dados formatado de forma mais adequada, se ele fosse um especialista em web scraper, provavelmente poderia ter economizado muito tempo nos tribunais e usado os documentos PDF diretamente, com um dos muitos módulos de análise de PDF do Python.

Infelizmente, muitas das bibliotecas de análise de PDF construídas para Python 2.x não foram atualizadas com o lançamento do Python 3.x. No entanto, como o PDF é um formato de documento relativamente simples e de código aberto, existem muitas bibliotecas Python decentes, mesmo no Python 3.x, que podem lê-los.

PDFMiner3K é uma dessas bibliotecas relativamente fácil de usar. É muito flexível, permitindo o uso da linha de comando ou integração com o código existente. Ele também pode lidar com uma variedade de codificações de linguagem - novamente, algo que geralmente é útil na web.

Você pode baixar este [Módulo Python](#) e instale-o descompactando a pasta e executando:

```
$ python setup.py install
```

A documentação está localizada em `/pdfminer3k-1.3.0/docs/index.html` dentro da pasta extraída, embora a documentação atual tenda a ser mais voltada para a interface de linha de comando do que para a integração com o código Python.

Esta é uma implementação básica que permite ler PDFs arbitrários em uma string, dado um objeto de arquivo local:

```
de urllib.request importar urlopen
de pdfminer.pdfinterp importar PDFResourceManager , process_pdf
de pdfminer.converter importar TextConverter
de pdfminer.layout importar LAParams
de io importar StringIO
de io importar abrir

def readPDF ( ficheiro PDF ):
    rsrcmgr = PDFResourceManager ()
    retstr = StringIO ()
    laparams = LAParams ()
    dispositivo = TextConverter ( rsrcmgr , retstr , laparams = laparams )

    process_pdf ( rsrcmgr , dispositivo , ficheiro PDF )
    dispositivo . Fechar ()

    conteúdo = retstr . Obter valor ()
    retstr . Fechar ()
    Retorna conteúdo

ficheiro PDF = urlopen ( "http://pythonscraping.com/pages/warandpeace/chapter1.pdf" );
outputString = readPDF ( ficheiro PDF )
```

```
impressão ( outputString )
ficheiro PDF . Fechar ()
```

O bom dessa função é que se você estiver trabalhando com arquivos localmente, pode simplesmente substituir um objeto de arquivo Python regular por aquele retornado por urlopen, e use a linha:

```
ficheiro PDF = abrir ( "..\pages\warandpeace\chapter1.pdf" , 'rb' )
```

A saída pode não ser perfeita, especialmente para PDFs com imagens, texto com formatação estranha ou texto organizado em tabelas ou gráficos. No entanto, para a maioria dos PDFs somente de texto, a saída não deve ser diferente do que se o PDF fosse um arquivo de texto.

Microsoft Word e .docx

Correndo o risco de ofender meus amigos da Microsoft: não gosto do Microsoft Word. Não porque seja necessariamente um software ruim, mas devido à maneira como os usuários o utilizam de maneira inadequada. Ele tem um talento particular para transformar o que deveriam ser documentos de texto simples ou PDFs em bestas grandes, lentas e difíceis de abrir que muitas vezes perdem toda a formatação de máquina para máquina e são, por qualquer motivo, editáveis quando o conteúdo é frequentemente direcionado para ser estático. Os arquivos do Word nunca foram feitos para transmissão frequente. No entanto, eles são onipresentes em certos sites, contendo documentos importantes, informações e até mesmo gráficos e multimídia; enfim, tudo o que pode e deve ser criado em HTML.

Antes de 2008, os produtos do Microsoft Office usavam o proprietário. *.doc* formato de arquivo. Este formato de arquivo binário era difícil de ler e mal suportado por outros processadores de palavras. Em um esforço para acompanhar o tempo e adotar um padrão que era usado por muitos outros softwares, a Microsoft decidiu usar o padrão baseado em Open Office XML, que tornava os arquivos compatíveis com código aberto e outros softwares.

Infelizmente, o suporte do Python para este formato de arquivo, usado pelo Google Docs, Open Office e Microsoft Office, ainda não é bom. Existe a [biblioteca python-docx](#), mas isso só dá aos usuários a capacidade de criar documentos e ler apenas os dados básicos do arquivo, como o tamanho e o título do arquivo, não o conteúdo real. Para ler o conteúdo de um arquivo do Microsoft Office, precisaremos lançar nossa própria solução.

A primeira etapa é ler o XML do arquivo:

```
de arquivo zip importar ZipFile
de urllib.request importar urlopen
de io importar BytesIO

arquivo de palavra = urlopen ( "http://pythonscraping.com/pages/AWordDocument.docx" ) . ler ()
arquivo de palavra = BytesIO ( arquivo de palavra )
documento = ZipFile ( arquivo de palavra )
xml_content = documento . ler ( 'word / document.xml' )
impressão ( xml_content . decodificar ( 'utf-8' ))
```

Isso lê um documento remoto do Word como um objeto de arquivo binário (BytesIO é análogo a StringIO, usado anteriormente neste capítulo), descompacta-o usando a biblioteca de arquivos zip do Python (todos. docx arquivos são compactados para economizar espaço) e, em seguida, lê o arquivo descompactado, que é XML.

O documento do Word em <http://pythonscraping.com/pages/AWordDocument.docx> é mostrado em Figura 6-2 .

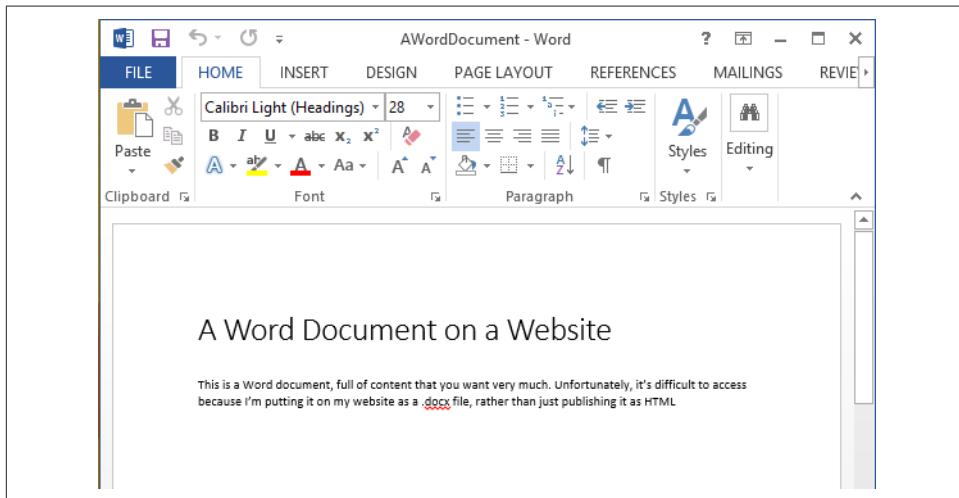


Figura 6-2. Este é um documento do Word que está cheio de conteúdo que você pode querer muito, mas é difícil de acessar porque estou colocando-o no meu site como um arquivo .docx em vez de publicá-lo como HTML

A saída do script Python lendo meu documento simples do Word é a seguinte:

```
<? -? xml version = "1.0" encoding = "UTF-8" standalone = "sim"? ->
<w: documento mc: ignorável = "w14 w15 wp14" xmlns: m = "http: //schemas.openxmlformats.org/officeDocument/2006/math" xmlns: mc = "http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns: o = "urn: schemas-microsoft-oft-com: office: office" xmlns: r = "http://schemas.openxmlformats.org/officeDocument / 2006 / relationships" xmlns: v = "urn: schemas-microsoft-com: vm l" xmlns: w = "http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns: w10 = "urn: schemas-microsoft-com: office: word" xmlns: w14 = "http://schemas.microsoft.com/office/word/2010/wordml" xmlns: w15 = "http://schemas.microsoft.com/office/word/2012/wordml" xmlns: wne = "http://schemas.microsoft.com/office/word/2006/wordml" xmlns: wp = "http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing" xmlns: wp14 = "http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing" xmlns: wpc = "http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas" xmlns: wpg = "http://schemas.microsoft.com/office/word/2010/wordprocessingGroup" xmlns: wpi = "http://schemas.microsoft.com/office/word / 2010 / wordprocessingInk" xmlns: wps = "http://schemas.microsoft.com/office/word / 2010 / wordprocessingShape" > < w: body> <w: p w: rsidp = "00764658" w: r
```

```

sidr = "00764658" w: rsidrdefault = "00764658" > < w: ppr > < w: pstyle w: val = "Tit le" > </ w: pstyle> </ w:
ppr > < w: t> < w: t> Um documento do Word em um site </ w: t> </ w: r> < w: bookmarkstart w: id = "0" w:
nome = "_Volte" > </ w: bookmarkstart> < w: b
ookmarkend w: id = "0" > </ w: bookmarkend> </ w: p> < w: p w: rsidp = "00764658" w: r sidr = "00764658" w:
rsidrdefault = "00764658" > </ w: p> < w: p w: rsidp = "00764658" w: rsidr = "00764658" w: rsidrdefault = "00764658"
w: rsidrpr = "00764658" >
< w: r> < w: t> Este é um documento Word, cheio de conteúdos que você deseja muito. Infelizmente, é
difícil de acessar porque estou colocando no meu site como um. </ w: t> </ w: r> < w: prooferr w: tipo = "spellStart"
> </
w: prooferr> < w: r> < w: t> docx </ w: t> </ w: r> < w: prooferr w: tipo = "spellEnd" > </
w: prooferr> < w: r> < w: t xml: espaço = "preservar" > arquivo, em vez de apenas publicá-lo como
HTML </ w: t> </ w: r> </ w: p> < w: sectpr w: rsidr = "00764658"
w: rsidrpr = "00764658" > < w: pgsize: h = "15840" w: w = "12240" > </ w: pgsize> < w: pgm
ar w: inferior = "1440" w: rodapé = "720" w: sarjeta = "0" w: cabeçalho = "720" w: esquerda =
"1440" w: direito = "1440" w: top = "1440" > </ w: pgmar> < w: cols w: espaço = "720" > <
w: cols E g; < w: docgrid w: linepitch = "360" > </ w: docgrid> </ w: sectpr> < / w: corpo> </ w: documento>

```

Claramente, há muitas informações aqui, mas estão enterradas. Felizmente, todo o texto do documento, incluindo o título na parte superior, está contido em `< w: t>` tags, o que torna mais fácil pegar:

```

de arquivo zip importar ZipFile
de urllib.request importar urlopen
de io importar BytesIO
de bs4 importar BeautifulSoup

arquivo de palavra = urlopen( "http://pythonscraping.com/pages/AWordDocument.docx" ) . ler ()
arquivo de palavra = BytesIO ( arquivo de palavra )
documento = ZipFile ( arquivo de palavra )
xml_content = documento . ler ( 'word / document.xml' )

wordObj = BeautifulSoup ( xml_content . decodificar ( 'utf-8' ) )
textStrings = wordObj . encontrar tudo ( "w: t" )
para textElem dentro textStrings :
    impressão ( textElem . texto )

```

A saída não é perfeita, mas está chegando lá e imprimindo cada `< w: t>` tag em uma nova linha torna mais fácil ver como o Word está dividindo o texto:

```

Um documento do Word em um site
Este é um documento Word, cheio de conteúdos que você deseja muito. Infelizmente, é difícil de acessar porque estou
colocando no meu site como um. arquivo docx, em vez de apenas publicá-lo como HTML

```

Observe que a palavra “docx” está em sua própria linha. No XML original, ele é cercado pela tag `< w: proofErr w: type = "spellStart" />`. Esta é a maneira que o Word usa de destacar “docx” com o sublinhado ondulado em vermelho, indicando que ele acredita que há um erro de grafia no nome de seu próprio formato de arquivo.

O título do documento é precedido pela tag do descritor de estilo <w: pstyle w: val = "Título">. Embora isso não torne extremamente fácil para nós identificarmos títulos (ou outro texto estilizado) como tal, usar os recursos de navegação do BeautifulSoup pode ser útil:

```
textStrings = wordObj . encontrar tudo ( "w: t" )
para textElem dentro textStrings :
    closeTag = ""
    experimentar :
        estilo = textElem . pai . irmão anterior . encontrar ( "w: pstyle" )
        E se estilo não é Nenhum e estilo [ "w: val" ] == "Título" :
            impressão ( "<h1>" )
            closeTag = "</h1>"
    exceto AttributeError :
        # Nenhuma etiqueta para imprimir
        passar
    impressão ( textElem . texto )
    impressão ( closeTag )
```

Esta função pode ser facilmente expandida para imprimir marcas em uma variedade de estilos de texto diferentes ou etiquetá-los de alguma outra maneira.

PARTE II

Raspagem Avançada

Você estabeleceu algumas bases de raspagem da web; Agora vem a parte divertida. Até este ponto, nossos raspadores de web têm sido relativamente burros. Eles são incapazes de recuperar informações, a menos que sejam imediatamente apresentadas a eles em um formato agradável pelo servidor. Eles pegam todas as informações pelo seu valor nominal e simplesmente as armazenam sem qualquer análise. Eles se confundem com formulários, interação com o site e até mesmo JavaScript. Em suma, eles não são bons para recuperar informações, a menos que essas informações realmente desejem ser recuperadas.

Esta parte do livro ajudará você a analisar dados brutos para obter a história por trás dos dados
- a história de que os sites muitas vezes escondem sob camadas de JavaScript, formulários de login e medidas anti-fraude.

Você aprenderá a usar web scrapers para testar seus sites, automatizar processos e acessar a Internet em grande escala. Ao final desta seção, você deverá ter as ferramentas para reunir e manipular quase todos os tipos de dados, em qualquer formato, em qualquer parte da Internet.

Limpando Seus Dados Sujos

Até agora, neste livro, ignoramos o problema de dados mal formatados usando fontes de dados geralmente bem formatadas, descartando os dados inteiramente se eles se desviarem do que esperávamos. Mas, frequentemente, em web scraping, você não pode ser muito exigente sobre de onde obtém seus dados.

Devido à pontuação incorreta, capitalização inconsistente, quebras de linha e erros ortográficos, os dados incorretos podem ser um grande problema na web. Neste capítulo, abordarei algumas ferramentas e técnicas para ajudá-lo a evitar o problema na fonte, alterando a maneira como você escreve o código e limpando os dados assim que estiverem no banco de dados.

Limpeza em código

Assim como você escreve código para lidar com exceções evidentes, você deve praticar a codificação defensiva para lidar com o inesperado.

Em linguística, um *n-grama* é uma sequência de *n* palavras usadas em texto ou fala. Ao fazer a análise de linguagem natural, muitas vezes pode ser útil quebrar um trecho de texto procurando n-gramas comumente usados ou conjuntos recorrentes de palavras que geralmente são usados juntos.

Nesta seção, vamos nos concentrar em obter n-gramas formatados adequadamente, em vez de usá-los para fazer qualquer análise. Mais tarde, em [Capítulo 8](#), você pode ver 2 gramas e 3 gramas em ação para fazer o resumo e a análise do texto.

O seguinte retornará uma lista de 2 gramas encontrada no artigo da Wikipedia sobre a linguagem de programação Python:

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup

def ngrams ( entrada , n ):
    entrada = entrada . Dividido ( " ")
```

```

resultado = []
para Eu dentro alcance ( len ( entrada ) - n + 1 ):
    resultado . acrescentar ( entrada [ Eu : Eu + n ])
Retorna resultado

html = urlopen ( "http://en.wikipedia.org/wiki/Python_(programming_language)" )
bsObj = BeautifulSoup ( html )
conteúdo = bsObj . encontrar ( "div" , { "Eu iria" : "mw-content-text" }) . get_text ()
ngrams = ngrams ( conteúdo , 2 )
impressão ( ngrams )
impressão ( " A contagem de 2 gramas é: " + str ( len ( ngrams )))

o ngrams A função recebe uma string de entrada, divide-a em uma sequência de palavras (assumindo que todas as palavras
são separadas por espaços) e adiciona o n-grama (neste caso, um 2-gramas) que cada palavra começa em uma matriz.

```

Isso retorna alguns 2 gramas genuinamente interessantes e úteis do texto:

```
['of', 'free'], ['free', 'and'], ['and', 'open-source'], ['open-source', 'softwa re']
```

mas também muito lixo:

```

'software \ nOutline \ nSPDX \ n\ \n\ \n\ \n\ \n\ \n\ \nOperando', 'sistema \ nfamílias \ n\ \n\ \nAROS \ nBSD \ nDarwin \
neCos \ nFreeDOS \ nGNU \ nHaiku \ nInferno \ nLinux \ nMach \ nMINIX \ nAbreSolaris \ nPlan ', [' sistema \ nfamilias \ n\ \n\ \
nAROS \ nBSD \ nDarwin \ nCos \ nFreeDOS \ nGNU \ nHaiMach \ nInferno \ nLinux \ nLinux nMINIX \ nOpenSolaris \ nPlan ','9 \
nReactOS \ nTUD: OS \ n\ \n\ \n\ \n\ \n\ \nDesenvolvimento \ n\ \n\ \n\ nBasic ], ['9 \ nReactOS \ nTUD: OS \ n\ \n\ \n\ \n\ \
\ n\ \n\ \nDesenvolvimento \ n\ \n\ \n\ nBasic ','Para ']
```

Além disso, como há 2 gramas criados para cada palavra encontrada (exceto a última), há 7.411 2 gramas no artigo no momento desta redação. Não é um conjunto de dados muito gerenciável!

Usando algumas expressões regulares para remover caracteres de escape (como \ n) e filtrando para remover quaisquer caracteres Unicode, podemos limpar um pouco a saída:

```

def ngrams ( entrada , n ):
    conteúdo = ré . sub ( "\n+" , "" , conteúdo )
    conteúdo = ré . sub ( '+' , "" , conteúdo )
    conteúdo = bytes ( conteúdo , "UTF-8" )
    conteúdo = conteúdo . decodificar ( "ascii" , "ignorar" )
    impressão ( conteúdo )
    entrada = entrada . Dividido ( " " )
    resultado = []
    para Eu dentro alcance ( len ( entrada ) - n + 1 ):
        resultado . acrescentar ( entrada [ Eu : Eu + n ])
    Retorna resultado

```

Isso primeiro substitui todas as instâncias do caractere de nova linha (ou vários caracteres de nova linha) por um espaço e, em seguida, substitui todas as instâncias de vários espaços em uma linha por um único espaço, garantindo que todas as palavras tenham um espaço entre eles. Em seguida, os caracteres de escape são eliminados codificando o conteúdo com UTF-8.

Essas etapas melhoraram muito o resultado da função, mas ainda existem alguns problemas:

```
[Pythoneers. [43] [44], 'Sintaxe'], [7, '/'], [", '3'], [3, '=='], [' == ',' 2 ']
```

Nesse ponto, as decisões que precisam ser tomadas para processar esses dados tornam-se mais interessantes. Existem mais algumas regras que podemos adicionar para nos aproximar dos dados ideais:

- As "palavras" de um único caractere devem ser descartadas, a menos que esse caractere seja "i" ou "a"
- As marcas de citação da Wikipedia (números entre colchetes) devem ser descartadas
- Os sinais de pontuação devem ser descartados (nota: esta regra é um tanto simplificadora e será explorada em mais detalhes em [Capítulo 9](#), mas é bom para o propósito deste exemplo)

Agora que a lista de "tarefas de limpeza" está ficando mais longa, é melhor removê-las e colocá-las em uma função separada, cleanInput:

```
def urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar ré
importar corda

def cleanInput( entrada ):
    entrada = ré . sub ( '\n + ' , "" , entrada )
    entrada = ré . sub ( '\[[0-9] * \]' , "" , entrada )
    entrada = ré . sub ( '+ ' , "" , entrada )
    entrada = bytes ( entrada , "UTF-8" )
    entrada = entrada . decodificar ( "ascii" , "ignorar" )
    cleanInput = []
    entrada = entrada . Dividido ( " " )
    para item dentro entrada :
        item = item . faixa ( corda . pontuação )
        E se len ( item ) > 1 ou ( item . mais baixo () == 'uma' ou item . mais baixo () == 'Eu' ):
            cleanInput . acrescentar ( item )
    Retorna cleanInput

def ngrams ( entrada , n ):
    entrada = cleanInput ( entrada )
    resultado = []
    para Eu dentro alcance ( len ( entrada ) - n + 1 ):
        resultado . acrescentar ( entrada [ Eu : Eu + n ])
    Retorna resultado
```

Observe o uso de importar string e string.punctuation para obter uma lista de todos os caracteres de pontuação em Python. Você pode ver a saída de string.punctuation de um terminal Python:

```
>>> importar string
>>> imprimir (string.punctuation)
! "# $% & '() * +, - / ; <=> ? @ [ ] ^ _ ` { } ~
```

Usando item.strip (string.punctuation) dentro de um loop que itera por todas as palavras no conteúdo, quaisquer caracteres de pontuação em qualquer lado da palavra serão removidos, embora palavras hifenizadas (onde o caractere de pontuação é delimitado por letras em ambos os lados) permanecerão intactas.

O resultado desse esforço resulta em 2 gramas muito mais limpos:

```
['Linux', 'Fundação'], ['Fundação', 'Mozilla'], ['Mozilla', 'Fundação'], ['Fundação', 'Aberto'], ['Aberto', 'Conhecimento'],
['Conhecimento', 'Fundação'], ['Fundação', 'Aberto'], ['Aberto', 'Fonte']
```

Normalização de Dados

Todo mundo encontrou um formulário da web mal projetado: “Digite o seu número de telefone. Seu número de telefone deve estar no formato ‘xxx-xxx-xxxx’.”

Como um bom programador, você provavelmente pensará consigo mesmo: “Por que eles simplesmente não retiram os caracteres não numéricos que coloquei lá e fazem isso sozinhos?” A normalização de dados é o processo de garantir que as strings que são linguística ou logicamente equivalentes umas às outras, como os números de telefone “(555) 123-4567” e “555.123.4567,” sejam exibidas, ou pelo menos comparadas , como equivalente.

Usando o código n-gram da seção anterior, podemos adicionar alguns recursos de normalização de dados.

Um problema óbvio com esse código é que ele contém muitos 2 gramas duplicados. Cada 2 gramas que encontra é adicionado à lista, sem nenhum registro de sua frequência. Não é apenas interessante registrar a frequência desses 2 gramas, em vez de apenas sua existência, mas pode ser útil para mapear os efeitos das alterações nos algoritmos de limpeza e normalização de dados. Se os dados forem normalizados com sucesso, o número total de n-gramas exclusivos será reduzido, enquanto a contagem total de n-gramas encontrados (ou seja, o número de itens exclusivos ou não exclusivos identificados como n-gramas) não será reduzida . Em outras palavras, haverá menos “baldes” para o mesmo número de n-gramas.

Infelizmente, para o propósito deste exercício, os dicionários Python não estão classificados. “Classificar um dicionário” não faz sentido, a menos que você esteja copiando os valores do dicionário para algum outro tipo de conteúdo e classificando-o. Uma solução fácil para este problema é o OrderedDict, da biblioteca de coleções do Python:

```
das coleções importar OrderedDict  
...  
  
ngrams = ngrams (conteúdo, 2)  
ngrams = OrderedDict (Sorted (ngrams.items (), key = lambda t: t [1], reverse = True)) print (ngrams)
```

Aqui estou aproveitando [Python's classificado função](#) para colocar os itens em um novo OrderedDict objeto, classificado pelo valor. Os resultados:

(["Software", "Fundação"], 40), (["Python", "Software"], 38), (["de", "o"], 35), (["Fundação", "Recuperado"], 34), (["de", "Python"], 28), (["em", "o"], 21), (["van", "Rossum"], 18)

No momento em que este livro foi escrito, havia 7.696 no total de 2 gramas e 6.005 2 gramas exclusivos, com o 2 gramas mais popular sendo "Software Foundation", seguido por "Python Software". No entanto, a análise dos resultados mostra que "Python Software" na verdade aparece na forma de "Python software" mais duas vezes. Da mesma forma, "van Rossum" e "Van Rossum" aparecem na lista separadamente.

Adicionando a linha:

```
entrada = entrada . superior ()
```

ao cleanInput A função mantém o número total de 2 gramas encontrado estável em 7.696, enquanto diminui o número de 2 gramas exclusivos para 5.882.

Além disso, geralmente é bom parar e considerar quanto poder de computação você deseja gastar para normalizar os dados. Existem várias situações em que diferentes grafias de palavras são equivalentes, mas para resolver essa equivalência, você precisa verificar cada palavra para ver se ela corresponde a alguma de suas equivalências pré-programadas.

Por exemplo, "Python primeiro" e "Python primeiro" aparecem na lista de 2 gramas. No entanto, fazer uma regra geral que diz "Todos 'primeiro', 'segundo', 'terceiro', etc. serão resolvidos para 1 °, 2 °, 3 °, etc. (ou vice-versa)" resultaria em um adicional 10 ou mais verificações por palavra.

Da mesma forma, o uso inconsistente de hífens ("coordenado" versus "coordenado"), erros ortográficos e outras incongruências de linguagem natural afetarão os agrupamentos de *n*-gramas, e pode turvar os resultados da saída se as incongruências forem comuns o suficiente.

Uma solução, no caso de palavras hifenizadas, pode ser remover totalmente os hífens e tratar a palavra como uma única string, o que exigiria apenas uma única operação. No entanto, isso também significa que as frases hifenizadas (uma ocorrência muito comum) serão tratadas como uma única palavra. Seguir o outro caminho e tratar os hífens como espaços pode ser uma opção melhor. Esteja preparado para o ocasional "ataque coordenado" e "ordenado" para entrar em ação!

Limpeza após o fato

Há tantas coisas que você pode ou deseja fazer no código. Além disso, você pode estar lidando com um conjunto de dados que não criou ou um conjunto de dados que seria um desafio até mesmo saber como limpar sem vê-lo primeiro.

Uma reação automática que muitos programadores têm nesse tipo de situação é "escrever um script", que pode ser uma solução excelente. No entanto, também existem ferramentas de terceiros,

como o OpenRefine, que são capazes não apenas de limpar dados de forma rápida e fácil, mas permitem que seus dados sejam facilmente vistos e usados por não programadores.

OpenRefine

OpenRefine é um projeto de código aberto iniciado por uma empresa chamada Metaweb em 2009. O Google adquiriu a Metaweb em 2010, mudando o nome do projeto de Freebase Gridworks para Google Refine. Em 2012, o Google abandonou o suporte para Refine e mudou o nome novamente, para OpenRefine, onde qualquer pessoa é bem-vinda para contribuir com o desenvolvimento do projeto.

Instalação

O OpenRefine é incomum porque embora sua interface seja executada em um navegador, é tecnicamente um aplicativo de desktop que deve ser baixado e instalado. Você pode baixar o aplicativo para Linux, Windows e Mac OS X de seu [local na rede Internet](#).



Se você for um usuário Mac e tiver problemas para abrir o arquivo, vá para Preferências do Sistema → Segurança e Privacidade → Geral → e marque “Qualquer lugar” em “Permitir download de aplicativos de”. Infelizmente, durante a transição de um projeto do Google para um projeto de código aberto, o OpenRefine parece ter perdido sua legitimidade aos olhos da Apple.

Para usar o OpenRefine, você precisará salvar seus dados como um CSV (consulte o arquivo "Armazenando Dados em CSV" em [capítulo 5](#) se você precisar de uma atualização sobre como fazer isso). Alternativamente, se você tiver seus dados armazenados em um banco de dados, poderá exportá-los para um arquivo CSV.

Usando OpenRefine

Nos exemplos a seguir, usaremos dados extraídos de [Tabela “Comparação de editores de texto” da Wikipedia](#) ; Veja Figura 7-1 . Embora esta tabela seja relativamente bem formatada, ela contém muitas edições feitas por pessoas por um longo tempo, portanto, tem algumas pequenas inconsistências de formatação. Além disso, como seus dados devem ser lidos por humanos em vez de máquinas, algumas das opções de formatação (por exemplo, usar “Grátis” em vez de “\$ 0,00”) são inadequadas para entradas de programação.

75 rows

Show as: rows records Show: 5 10 25 50 rows Extensions: Firebase ▾

All	Name	Creator	First public rel.	Latest stable ver.	Programming language	Cost (US\$)	Software license	Open source
1.	Acme	Rob Pike	1993	Plan 9 and Inferno	C	\$0	LPL (OSI approved)	Yes
2.	AkellPad	Alexey Kuznetsov, Alexander Shengals	2003	4.9.0	C	\$0	BSD	Yes
3.	Alphatik	Vince Darley	1999	8.3.3		\$40	Proprietary, with BSD components	No
4.	Aquamacs	David Reitter	2005	3.0a	C, Emacs Lisp	\$0	GPL	Yes
5.	Atom	Github	2014	0.132.0	HTML, CSS, JavaScript, C++	\$0	MIT	Yes
6.	BBlueEdit	Rich Siegel	1992-04	10.5.12	Objective-C, Objective-C++	\$49.99	Proprietary	No
7.	Bluelish	Bluelish Development Team	1999	2.2.6	C	\$0	GPL	Yes
8.	Coda	Panic	2007	2.0.12	Objective-C	\$99	Proprietary	No
9.	ConTEXT	ConTEXT Project Ltd	1999	0.98.6	Object Pascal (Delphi)	\$0	BSD	Yes
10.	Crimson Editor	Ingyu Kang, Emerald Editor Team	1999	3.72	C++	\$0	GPL	Yes
11.	Diakonos	Pistos	2004	0.9.2	Ruby	\$0	MIT	Yes
12.	E Text Editor	Alexander Stigsen	2005	2.0.2		\$46.95	Proprietary, with BSD components	No
13.	ed	Ken Thompson	1970	unchanged from original	C	\$0	?	Yes
14.	EditPlus	Sangil Kim	1998	3.5	C++	\$35	Shareware	No
15.	Editra	Cody Precord	2007	0.6.77	Python	\$0	wxWindows license	Yes

Figura 7-1. Dados da “comparação de editores de texto” da Wikipedia, conforme mostrado na tela principal do OpenRefine

A primeira coisa a observar sobre OpenRefine é que cada rótulo de coluna possui uma seta ao lado dele. Esta seta fornece um menu de ferramentas que podem ser usadas com aquela coluna para filtrar, classificar, transformar ou remover dados.

Filtrando. A filtragem de dados pode ser realizada usando dois métodos: filtros e facetas. Filtros são bons para usar expressões regulares para filtrar os dados; por exemplo, “Mostrar apenas dados que contenham quatro ou mais linguagens de programação separadas por vírgula na coluna Linguagem de programação”, visto em [Figura 7-2](#).

Facet / Filter Undo / Redo 1

Refresh Reset All Remove All

Programming language

.+, .+, +

case sensitive regular expression

5 matching rows (75 total)

Show as: rows records Show: 5 10 25 50 rows

All	Name	Creator	First public re
5.	Atom	Github	201
28.	Komodo Edit	Activestate	open-sourced 2007
29.	Komodo IDE	Activestate	200
59.	Sublime Text	Jon Skinner	200
74.	Zed	Zef Hemel	201

Figura 7-2. A expressão regular “.+, .+, +” Seleciona valores que têm pelo menos três itens separados por vírgula

Os filtros podem ser combinados, editados e adicionados facilmente, manipulando os blocos na coluna da direita. Eles também podem ser combinados com facetas.

As facetas são ótimas para incluir ou excluir dados com base em todo o conteúdo da coluna. (por exemplo, “Mostrar todas as linhas que usam a licença GPL ou MIT, e foram lançadas pela primeira vez depois de 2005,” visto em [Figura 7-3](#)) Eles têm ferramentas de filtragem integradas. Por exemplo, filtro

usar um valor numérico fornece barras deslizantes para selecionar o intervalo de valores que deseja incluir.

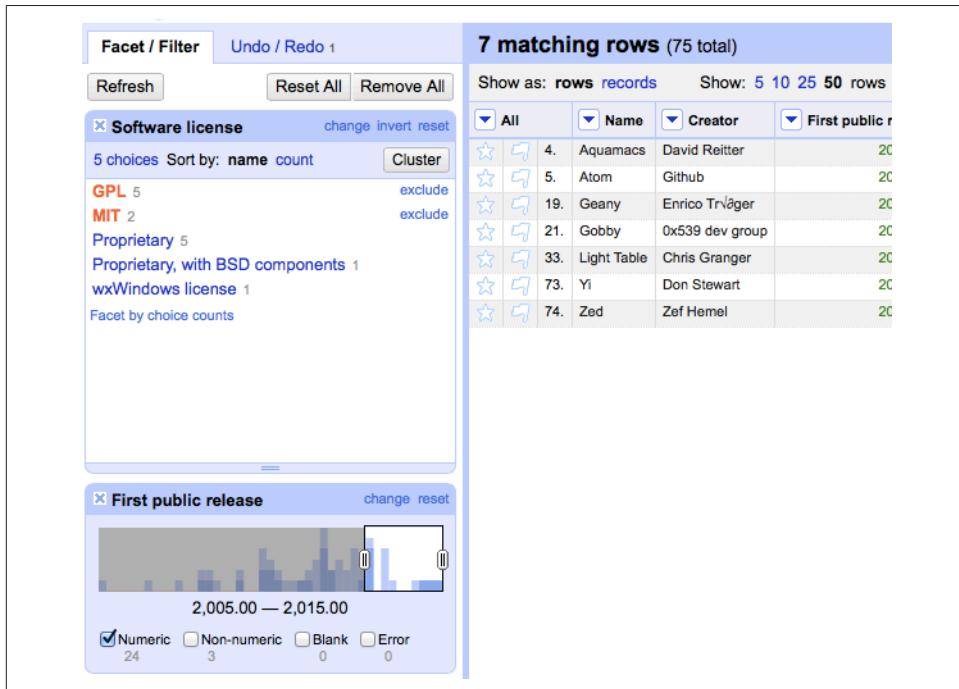


Figura 7-3. Isso exibe todos os editores de texto usando a licença GPL ou MIT que tiveram seu primeiro lançamento público após 2005

Independentemente de como você filtra seus dados, eles podem ser exportados a qualquer ponto para um dos vários tipos de formatos que o OpenRefine suporta. Isso inclui CSV, HTML (uma tabela HTML), Excel e vários outros formatos.

Limpeza. A filtragem de dados pode ser feita com sucesso apenas se os dados estiverem relativamente limpos para começar. Por exemplo, no exemplo de faceta da seção anterior, um editor de texto com data de lançamento "01-01-2006" não teria sido selecionado na faceta "Primeiro lançamento público", que procurava um valor de "2006" e ignorando valores que não eram assim.

A transformação de dados é realizada no OpenRefine usando a linguagem de expressão OpenRefine, chamada GREL (o "G" é o resto do nome anterior do OpenRefine, Google Refine). Essa linguagem é usada para criar funções lambda curtas que transformam os valores nas células com base em regras simples. Por exemplo:

```
if (value.length ()!= 4, "inválido", valor)
```

Quando esta função é aplicada à coluna “Primeira versão estável”, ela preserva os valores das células em que a data está no formato “AAAA” e marca todas as outras colunas como “inválidas”.

As instruções GREL arbitrárias podem ser aplicadas clicando na seta para baixo ao lado do rótulo de qualquer coluna e indo para editar células → transformar.

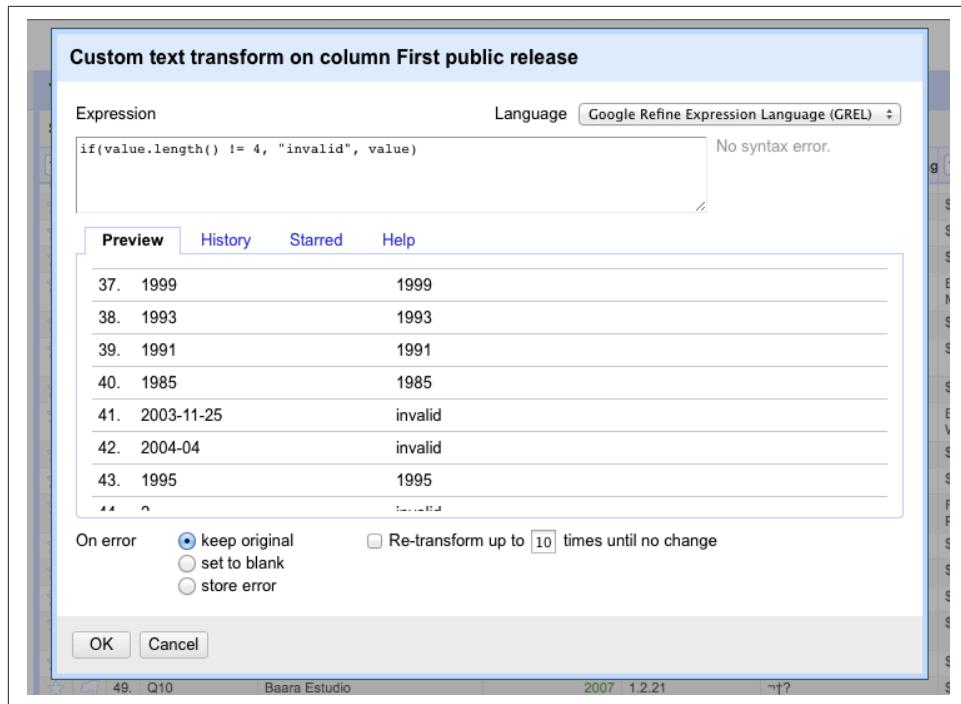


Figura 7-4. Inserindo uma declaração GREL em um projeto (uma exibição de visualização é mostrada abaixo da declaração)

No entanto, marcar todos os valores abaixo do ideal como inválidos, ao mesmo tempo que os torna fáceis de identificar, não nos ajuda muito. Preferimos tentar recuperar informações dos valores mal formatados, se possível. Isso pode ser feito usando GREL's partida função:

```
value.match ("^.*([0-9]{4}).*$").get(0)
```

Isso tenta combinar o valor da string com a expressão regular fornecida. Se a expressão regular corresponder à string, uma matriz será retornada. Quaisquer substrings que correspondam ao “grupo de captura” na expressão regular (demarcada pelos parênteses na expressão, neste exemplo, “[0-9]{4}”) são retornados como valores de matriz.

Este código, na verdade, encontra todas as instâncias de quatro decimais em uma linha e retorna a primeira. Geralmente, isso é suficiente para extrair anos de texto ou datas mal formatadas. isto

também tem a vantagem de retornar “nulo” para datas inexistentes. (GREL não lança uma exceção de ponteiro nulo ao realizar operações em uma variável nula)

Muitas outras transformações de dados são possíveis com edição de células e GREL. Um guia completo do idioma pode ser encontrado em [Página GitHub do OpenRefine](#).

Ler e escrever linguagens naturais

Até agora, os dados com os quais trabalhamos geralmente estavam na forma de números ou valores contáveis. Na maioria dos casos, simplesmente armazenamos os dados sem realizar nenhuma análise após o fato. Neste capítulo, tentaremos abordar o assunto delicado da língua inglesa.¹

Como o Google sabe o que você está procurando quando digita “gatinho fofo” na Pesquisa de imagens? Por causa do texto que envolve as imagens de gatinhos fofo. Como o YouTube sabe que deve trazer um certo esboço do Monty Python quando você digita “dead par- rot” na barra de pesquisa? Por causa do título e do texto de descrição que acompanha cada vídeo enviado.

Na verdade, mesmo digitar termos como “pássaro falecido monty python” imediatamente traz o mesmo esboço “Papagaio morto”, embora a página em si não contenha nenhuma menção das palavras “falecido” ou “pássaro”. O Google sabe que um “cachorro-quente” é uma comida e que um “cachorrinho fervente” é algo totalmente diferente. Quão? São tudo estatísticas!

Embora você possa achar que a análise de texto não tem nada a ver com o seu projeto, entender os conceitos por trás dela pode ser extremamente útil para todos os tipos de aprendizado de máquina, bem como a capacidade mais geral de modelar problemas do mundo real de forma probabilística e termos algorítmicos.

¹ Embora muitas das técnicas descritas neste capítulo possam ser aplicadas a todos ou à maioria dos idiomas, por enquanto, está tudo bem se concentrar no processamento de linguagem natural apenas em inglês. Ferramentas como o Natural Language Toolkit do Python, por exemplo, se concentram no inglês. Cinquenta e seis por cento da Internet ainda está em inglês (com o alemão seguindo apenas 6%, de acordo com http://w3techs.com/technologies/overview/content_language/all). Mas quem sabe? O domínio do inglês na maior parte da Internet quase certamente mudará no futuro, e novas atualizações podem ser necessárias nos próximos anos.

Por exemplo, o serviço de música Shazam pode identificar o áudio como contendo uma determinada gravação de música, mesmo se esse áudio contiver ruído ambiente ou distorção. O Google está trabalhando na legendagem automática de imagens com base em nada além da própria imagem.²

Ao comparar imagens conhecidas de, digamos, cachorros-quentes com outras imagens de cachorros-quentes, o mecanismo de pesquisa pode aprender gradualmente a aparência de um cachorro-quente e observar esses padrões em imagens adicionais mostradas.

Resumindo Dados

Dentro [Capítulo 7](#), analisamos a divisão do conteúdo de texto em n-gramas, ou conjuntos de frases que são *n*-palavras em comprimento. Em um nível muito básico, isso pode ser usado para determinar quais conjuntos de palavras e frases tendem a ser mais comumente usados em uma seção de texto. Além disso, ele pode ser usado para criar resumos de dados que parecem naturais, voltando ao texto original e extrairindo frases em torno de algumas dessas frases mais populares.

Um exemplo de texto que usaremos para fazer isso é o discurso de posse do nono presidente dos Estados Unidos, William Henry Harrison. A presidência de Harrison bate dois recordes na história do cargo: um para o discurso de posse mais longo e outro para o tempo mais curto no cargo, 32 dias.

Usaremos o texto completo deste [discurso](#) como a fonte para muitos dos exemplos de código neste capítulo.

Modificando ligeiramente o n-gramma usado para encontrar o código em [Capítulo 7](#), podemos produzir código que procura conjuntos de 2 gramas e os classifica usando a função de classificação do Python no módulo "operador":

```
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar ré
importar corda
importar operador

def cleanInput( entrada ):
    entrada = ré . sub ( '\n + ' , "" , entrada ) . mais baixo ( )
    entrada = ré . sub ( '\[[0-9] * \]' , "" , entrada )
    entrada = ré . sub ( '+ ' , "" , entrada )
    entrada = bytes ( entrada , "UTF-8" )
    entrada = entrada . decodificar ( "ascii" , "ignorar" )
    cleanInput = []
    entrada = entrada . Dividido ( " " )
    para item dentro entrada :
```

² Ver "Uma imagem vale mais que mil palavras (coerentes): Construindo uma descrição natural das imagens", novembro 17, 2014 (<http://bit.ly/1HEJ8kX>).

```

item = item . faixa ( corda . pontuação )
E se len ( item ) > 1 ou ( item . mais baixo () == 'uma' ou item . mais baixo () == 'Eu' ):
    cleanInput . acrescentar ( item )
Retorna cleanInput

def ngrams ( entrada , n ):
    entrada = cleanInput ( entrada )
    resultado = {}
    para Eu dentro alcance ( len ( entrada ) - n + 1 ):
        ngramTemp = "" . Junte-se ( entrada [ Eu : Eu + n ] )
        E se ngramTemp não em resultado :
            resultado [ ngramTemp ] = 0
            resultado [ ngramTemp ] += 1
    Retorna resultado

conteúdo = str (
    urlopen ( "http://pythonscraping.com/files/inaugurationSpeech.txt" ) . ler (),
    'utf-8' )

ngrams = ngrams ( conteúdo , 2 )
SortNGrams = classificado ( ngrams . Itens () , chave = operador . itemgetter ( 1 ) , reverter = Verdadeiro )
impressão ( SortNGrams )

```

A saída produz, em parte:

```

[('da', 213), ('na', 65), ('na', 61), ('pela', 41), ('a constituição', 34), ('da nosso', 29), ('ser', 26), ('de', 24
), ('o povo', 24), ('e o', 23), ('é', 23), ('que o', 2
3), ('de um', 22), ('deles', 19)

```

Destes 2 gramas, “a constituição” parece um assunto razoavelmente popular no discurso, mas “do”, “no” e “para o” não parecem especialmente dignos de nota. Como você pode se livrar automaticamente de palavras indesejadas de maneira precisa?

Felizmente, existem pessoas por aí que estudam cuidadosamente as diferenças entre palavras “interessantes” e palavras “desinteressantes”, e seu trabalho pode nos ajudar a fazer exatamente isso. Mark Davies, professor de lingüística da Brigham Young University, mantém o

Corpus do inglês americano contemporâneo, uma coleção de mais de 450 milhões de palavras da última década ou mais de publicações americanas populares.

A lista de 5.000 palavras encontradas com mais frequência está disponível gratuitamente e, felizmente, isso é muito mais do que o suficiente para funcionar como um filtro básico para eliminar os 2 gramas mais comuns. Apenas as primeiras 100 palavras melhoraram muito os resultados, com a adição de um é comum função:

```

def é comum ( ngram ):
    palavras comuns = [ "a" , "estar" , "e" , "do" , "uma" , "dentro" , "para" , "ter" , "isto" ,
    "Eu" , "este" , "para" , "vocês" , "ele" , "com" , "em" , "Faz" , "dizer" , "esta" ,
    "eles" , "é" , "a" , "em" , "mas" , "nós" , "dele" , "de" , "este" , "não" ,
    "de" , "ela" , "ou" , "Como" , "o que" , "ir" , "deles" , "posso" , "quem" , "pegue" ,
    "E se" , "seria" , "dela" , "todos" , "meu" , "faço" , "sobre" , "conhecer" , "vai" ,
    "Como" , "acima" , "1" , "Tempo" , "tem" , "fui" , "há" , "ano" , "então" ,
    "pensar" , "quando" , "qual" , "eles" , "alguns" , "mim" , "pessoas" , "levar" ,

```

"Fora" , "para dentro" , "somente" , "Vejo" , "ele" , "seu" , "venha" , "poderia" , "agora" ,
"do que" , "gostar" , "de outros" , "quão" , "então" , "Está" , "nosso" , "dois" , "Mais" ,
"estes" , "quer" , "caminho" , "Veja" , "primeiro" , "Além disso" , "Novo" , "Porque" ,
"dia" , "Mais" , "usar" , "não" , "homem" , "encontrar" , "aqui" , "coisa" , "dar" ,
"muitos" , "bem"]

para palavra dentro ngram :

E se palavra dentro palavras comuns :

Retorna Verdadeiro

Retorna Falso

Isso produz os seguintes 2 gramas que foram encontrados mais de duas vezes no corpo do texto:

('Estados Unidos' , 10), ('departamento executivo' , 4), ('governo geral ent' , 4), ('chamado a' , 3), ('o governo deveria' , 3), (' país inteiro ' , 3), (' Sr. Jefferson ' , 3), (' magistrado chefe ' , 3), (' mesmas causas ' , 3), ('corpo legislativo' , 3)

Apropriadamente, os dois primeiros itens da lista são “Estados Unidos” e “departamento executivo”, que esperaríamos em um discurso de posse presidencial.

É importante notar que estamos usando uma lista de palavras comuns de tempos relativamente modernos para filtrar os resultados, o que pode não ser apropriado, visto que o texto foi escrito em 1841. No entanto, porque estamos usando apenas as primeiras 100 ou mais palavras na lista, que podemos assumir são mais estáveis ao longo do tempo do que, digamos, as últimas 100 palavras

- e parece que estamos obtendo resultados satisfatórios, provavelmente podemos nos poupar do esforço de rastrear ou criar uma lista das palavras mais comuns de 1841 (embora tal esforço possa ser interessante).

Agora que alguns tópicos importantes foram extraídos do texto, como isso nos ajuda a escrever resumos de texto? Uma maneira é pesquisar a primeira frase que contém cada n-grama “popular”, a teoria é que a primeira instância fornecerá uma visão geral satisfatória do corpo do conteúdo. Os cinco primeiros 2 gramas mais populares rendem estes pontos:

- A Constituição dos Estados Unidos é o instrumento que contém esta outorga de poder aos diversos departamentos que compõem o governo.
- Tal foi concedido pelo departamento executivo constituído pela Constituição.
- O governo geral não se apoderou de nenhum dos direitos reservados dos estados.
- Chamado de uma aposentadoria que eu supunha que continuaria pelo resto da minha vida para ocupar o cargo de executivo desta grande e livre nação, apresento-me diante de vocês, concidadãos, para fazer os juramentos que a constituição prescreve como um qualificação necessária para o desempenho de suas funções; e em obediência a um costume contemporâneo com nosso governo e o que acredito serem suas expectativas, passo a apresentar-lhes um resumo dos princípios que me regerão no cumprimento dos deveres que serei chamado a cumprir.

- As prensas no emprego necessário do governo nunca devem ser usadas para inocentar o culpado ou para envernar o crime.

Claro, pode não ser publicado em CliffsNotes em breve, mas considerando que o documento original tinha 217 frases e a quarta frase ("Chamado de uma aposentadoria ...") condensa o assunto principal muito bem, não é muito ruim para uma primeira passagem.

Modelos Markov

Você já deve ter ouvido falar de geradores de texto Markov. Eles se tornaram populares para fins de entretenimento, como no [App Twitov](#), bem como seu uso para gerar e-mails de spam que parecem reais para enganar os sistemas de detecção.

Todos esses geradores de texto são baseados no modelo de Markov, que é freqüentemente usado para analisar grandes conjuntos de eventos aleatórios, onde um evento discreto é seguido por outro evento discreto com uma certa probabilidade.

Por exemplo, podemos construir um modelo de Markov de um sistema climático conforme ilustrado em

Figura 8-1.

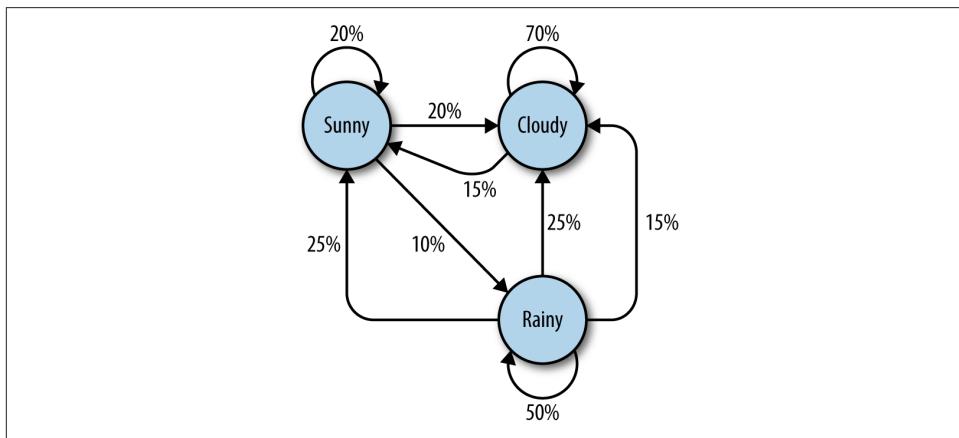


Figura 8-1. Modelo de Markov que descreve um sistema meteorológico teórico

Neste modelo, cada dia de sol tem 70% de chance de o dia seguinte também ser ensolarado, com 20% de chance de o dia seguinte estar nublado com apenas 10% de chance de chuva. Se o dia estiver chuvoso, há 50% de chance de chuva no dia seguinte, 25% de chance de sol e 25% de chance de nuvens.

Nota:

- Todas as porcentagens que conduzem para fora de qualquer nó devem somar exatamente 100%. Não importa o quão complicado seja o sistema, sempre deve haver 100% de chance de que ele leve a outro lugar na próxima etapa.
- Embora existam apenas três possibilidades para o clima em um determinado momento, você pode usar este modelo para gerar uma lista infinita de estados do tempo.
- Apenas o estado do nó atual em que você está influencia para onde você irá em seguida. Se você estiver no nó "ensolarado", não importa se os 100 dias anteriores foram ensolarados ou chuvosos - as chances de sol no dia seguinte são exatamente as mesmas: 70%.
- Pode ser mais difícil alcançar alguns nós do que outros. A matemática por trás disso é razoavelmente complicada, mas deve ser bastante fácil ver que "chuvisco" (com menos de "100%" de setas apontando em direção a ele) é um estado muito menos provável de alcançar neste sistema, em qualquer dado ponto no tempo, do que "ensolarado" ou "nublado".

Obviamente, este é um sistema muito simples, e os modelos de Markov podem aumentar de tamanho arbitrariamente. Na verdade, o algoritmo de classificação de página do Google é baseado parcialmente em um modelo de Markov, com sites representados como nós e links de entrada / saída representados como conexões entre nós. A "probabilidade" de pousar em um nó específico representa a popularidade relativa do site. Ou seja, se nosso sistema meteorológico representasse uma Internet extremamente pequena, "chuvisco" teria um page rank baixo, enquanto "nublado" teria um page rank alto.

Com tudo isso em mente, vamos voltar a um exemplo mais concreto: analisar e escrever um texto.

Again using the inauguration speech of William Henry Harrison analyzed in the previous example, we can write the following code that generates arbitrarily long Markov chains (with the chain length set to 100) based on the structure of its text:

```
from urllib.request import urlopen
from random import randint

def wordListSum ( wordList ):
    sum = 0
    for word , value in wordList . items ():
        sum += value
    return sum

def retrieveRandomWord ( wordList ):

    randIndex = randint ( 1 , wordListSum ( wordList ) )
    for word , value in wordList . items ():
        randIndex -= value
    if randIndex <= 0 :
        return word

def buildWordDict ( text ):
```

```

# Remove newlines and quotes
text = text . replace ( "\n" , " " );
text = text . replace ( "\t" , " " );

# Make sure punctuation marks are treated as their own "words,"
# so that they will be included in the Markov chain
punctuation = [ '!', '.', ',', ';' ]
for symbol in punctuation :
    text = text . replace ( symbol , " " + symbol + " " );

words = text . split ( " " )
# Filter out empty words
words = [ word for word in words if word != "" ]

wordDict = {}
for i in range ( 1 , len ( words ) ):
    if words [ i - 1 ] not in wordDict :
        # Create a new dictionary for this word
        wordDict [ words [ i - 1 ] ] = {}
    if words [ i ] not in wordDict [ words [ i - 1 ] ]:
        wordDict [ words [ i - 1 ] ][ words [ i ] ] = 0
    wordDict [ words [ i - 1 ] ][ words [ i ] ] = wordDict [ words [ i - 1 ] ][ words [
        i ] ] + 1

return wordDict

text = str ( urlopen ( "http://pythonscraping.com/files/inaugurationSpeech.txt" )
            . read () , 'utf-8' )
wordDict = buildWordDict ( text )

# Generate a Markov chain of length 100
length = 100
chain = ""
currentWord = "|"
for i in range ( 0 , length ):
    chain += currentWord + " "
    currentWord = retrieveRandomWord ( wordDict [ currentWord ] )

print ( chain )

```

The output of this code changes every time it is run, but here's an example of the uncannily nonsensical text it will generate:

I sincerely believe in Chief Magistrate to make all necessary sacrifices and oppression of the remedies which we may have occurred to me in the arrangement and disbursement of the democratic claims them , consolatory to have been best political power in fervently commanding every other addition of legislation , by the interests which violate that the Government would compare our aboriginal neighbors the people to its accomplishment . The latter also susceptible of the Constitution not much mischief , disputes have left to betray . The maxim which may sometimes be an impartial and to prevent the adoption or

So what's going on in the code?

The function `buildWordDict` takes in the string of text, which was retrieved from the Internet. It then does some cleaning and formatting, removing quotes, and putting spaces around other punctuation so it is effectively treated as a separate word. After this, it builds a two-dimensional dictionary—a “dictionary of dictionaries”—that has the following form:

```
{word_a : {word_b : 2, word_c : 1, word_d : 1}, word_e : {word_b : 5,  
word_d : 2},...}
```

In this example dictionary, “`word_a`” was found four times, two instances of which were followed by “`word_b`,” one instance followed by “`word_c`,” and one instance followed by “`word_d`.” “`Word_e`” was followed seven times, five times by “`word_b`” and twice by “`word_d`.”

If we were to draw a node model of this result, the node representing “`word_a`” would have a 50% arrow pointing toward “`word_b`” (which followed it two out of four times), a 25% arrow pointing toward “`word_c`,” and a 25% arrow pointing toward “`word_d`.”

Once this dictionary is built up, it can be used as a lookup table to see where to go next, no matter which word in the text you happen to be on.³ Using the sample dictionary of dictionaries, we might currently be on “`word_e`,” which means that we’ll pass

the dictionary `{ word_b : 5, word_d: 2 }` to the `retrieveRandomWord` function. This function in turn retrieves a random word from the dictionary, weighted by the number of times it occurs.

By starting with a random starting word (in this case, the ubiquitous “I”), we can traverse through the Markov chain easily, generating as many words as we like.

Six Degrees of Wikipedia: Conclusion

In [Chapter 3](#), we created a scraper that collects links from one Wikipedia article to the next, starting with the article on Kevin Bacon, and stores them in a database. Why are we bringing it up again? Because it turns out the problem of choosing a path of links that starts on one page and ends up on the target page (i.e., finding a string of pages between https://en.wikipedia.org/wiki/Kevin_Bacon and https://en.wikipedia.org/wiki/Eric_Idle) is the same as finding a Markov chain where both the first word and last word are defined. These sorts of problems are *directed graph* problems, where $A \rightarrow B$ does not necessarily mean that $B \rightarrow A$. The word “football” might often be followed by the word “player,” but you’ll find that the word “player” is much less often followed

³ The exception is the last word in the text because nothing follows the last word. In our example text, the last word is a period (.), which is convenient because it has 215 other occurrences in the text and so does not represent a dead end. However, in real-world implementations of the Markov generator, the last word of the text might be something you need to account for.

by the word “football.” Although Kevin Bacon’s Wikipedia article links to the article on his home city, Philadelphia, the article on Philadelphia does not reciprocate by linking back to him.

In contrast, the original Six Degrees of Kevin Bacon game is an *undirected graph* problem. If Kevin Bacon starred in *Flatliners* with Julia Roberts, then Julia Roberts necessarily starred in *Flatliners* with Kevin Bacon, so the relationship goes both ways (it has no “direction”). Undirected graph problems tend to be less common in computer science than directed graph problems, and both are computationally difficult to solve.

Although much work has been done on these sorts of problems and multitudes of variations on them, one of the best and most common ways to find shortest paths in a directed graph—and thus find paths between the Wikipedia article on Kevin Bacon and all other Wikipedia articles—is through a *breadth-first search*.

A breadth-first search is performed by first searching all links that link directly to the starting page. If those links do not contain the target page (the page you are searching for), then a second level of links—pages that are linked by a page that is linked by the starting page—is searched. This process continues until either the depth limit (6 in this case) is reached or the target page is found.

A complete solution to the breadth-first search, using a table of links as described in [Chapter 5](#), is as follows:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import pymysql

conn = pymysql.connect ( host = '127.0.0.1' , unix_socket = '/tmp/mysql.sock' ,
                        user = 'root' , passwd = None , db = 'mysql' , charset = 'utf8' )
cur = conn . cursor ()
cur . execute ( "USE wikipedia" )

class SolutionFound ( RuntimeError ):
    def __init__ ( self , message ):
        self . message = message

    def getLinks ( fromPagId ):
        cur . execute ( "SELECT toPagId FROM links WHERE fromPagId = %s " , ( fromPagId ) )
        if cur . rowcount == 0 :
            return None
        else :
            return [ x [ 0 ] for x in cur . fetchall ()]

    def constructDict ( currentPagId ):
        links = getLinks ( currentPagId )
        if links :
            return dict ( zip ( links , [ {} ] * len ( links ) ) )
        return {}


```

```

# The link tree may either be empty or contain multiple links
def searchDepth ( targetPagId , currentPagId , linkTree , depth ):
    if depth == 0 :
        # Stop recursing and return, regardless
        return linkTree
    if not linkTree :
        linkTree = constructDict ( currentPagId )
    if not linkTree :
        # No links found. Cannot continue at this node
        return {}
    if targetPagId in linkTree . keys ():

        print ( " TARGET " + str ( targetPagId ) + " FOUND!" )
        raise SolutionFound ( "PAGE: " + str ( currentPagId ) )

    for branchKey , branchValue in linkTree . items ():

        try :
            # Recuse here to continue building the tree
            linkTree [ branchKey ] = searchDepth ( targetPageId , branchKey ,
                branchValue , depth - 1 )
        except SolutionFound as e :
            print ( e . message )
            raise SolutionFound ( "PAGE: " + str ( currentPagId ) )
    return linkTree

try :
    searchDepth ( 134951 , 1 , {} , 4 )
    print ( " No solution found" )
except SolutionFound as e :
    print ( e . message )

```

The functions getLinks and constructDict are helper functions that retrieve links from the database given a page, and format those links into a dictionary. The main function, searchDepth, works recursively to simultaneously construct and search a tree of links, working one level at a time. It operates on the following rules:

- If the given recursion limit has been reached (i.e., if it has called itself too many times), return without doing any work.
- If the dictionary of links it has been given is empty, populate it with links for the current page. If the current page has no links, return.
- If the current page contains a link to the page we are searching for, throw an exception that alerts copies of itself on up the stack that the solution has been found. Each stack then prints the current page it is on, and throws the exception again, resulting in a perfect list of pages leading to the solution being printed on the screen.
- If the solution is not found, call itself while subtracting one from the depth count in order to search the next level of links.

The output for searching for a link between the page on Kevin Bacon (page ID 1, in my database) and the page on Eric Idle (page ID 78520 in my database) is:

```
TARGET 134951 FOUND!
PAGE: 156224
PAGE: 155545
PAGE: 3
PAGE: 1
```

This translates into the relationship of links: Kevin Bacon → San Diego Comic Con International → Brian Froud → Terry Jones → Eric Idle

In addition to solving “6 degree” problems and modeling which words tend to follow which other words in sentences, directed and undirected graphs can be used to model a variety of different situations encountered in web scraping. Which websites link to which other websites? Which research papers cite which other research papers? Which products tend to be shown with which other products on a retail site? What is the strength of this link? Is the link reciprocal?

Recognizing these fundamental types of relationships can be extremely helpful for making models, visualizations, and predictions based on scraped data.

Natural Language Toolkit

So far, this chapter has focused primarily on the statistical analysis of words in bodies of text. Which words are most popular? Which words are unusual? Which words are likely to come after which other words? How are they grouped together? What we are missing is understanding, to the extent that we can, what the words represent.

The Natural Language Toolkit (NLTK) is a suite of Python libraries designed to identify and tag parts of speech found in natural English text. Its development began in 2000, and over the past 15 years dozens of developers around the world have contributed to the project. Although the functionality it provides is tremendous (entire books are devoted to NLTK), this section will focus on just a few of its uses.

Installation and Setup

The NLTK module can be installed the same as other Python modules, either by downloading the package through the NLTK website directly or by using any number of third-party installers with the keyword “nltk.” For complete installation instructions, you see the [NLTK website](#).

After installing the module it’s a good idea to download its preset text repositories so you can try out some of the features more easily. Type this on the Python command line:

```
>>> import nltk
>>> nltk.download()
```

This opens the NLTK Downloader ([Figure 8-2](#)).

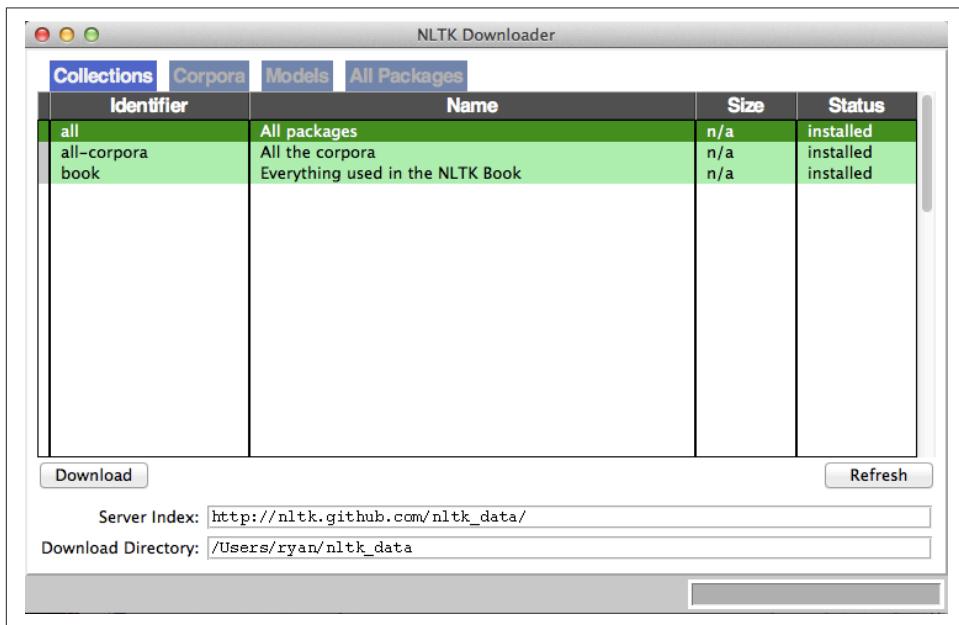


Figure 8-2. The NLTK Downloader lets you browse and download optional packages and text libraries associated with the NLTK module

I recommend installing all of the available packages. Because everything is text based the downloads are very small; you never know what you'll end up using, and you can easily uninstall packages at any time.

Statistical Analysis with NLTK

NLTK is great for generating statistical information about word counts, word frequency, and word diversity in sections of text. If all you need is a relatively straightforward calculation (e.g., the number of unique words used in a section of text), importing NLTK might be overkill—it's a very large module. However, if you need to do relatively extensive analysis of a text, you have a number of functions at your fingertips that will give you just about any metric you want.

Analysis with NLTK always starts with the `Text` object. `Text` objects can be created from simple Python strings in the following way:

```
from nltk import word_tokenize
from nltk import Text

tokens = word_tokenize( "Here is some not very interesting text" )
text = Text( tokens )
```

The input for the word_tokenize function can be any Python text string. If you don't have any long strings handy but still want to play around with the features, NLTK has quite a few books already built into the library, which can be accessed using the import function:

```
from nltk.book import *
```

This loads the nine books:

```
*** Introductory Examples for the NLTK Book *** Loading text1, ..., text9  
and sent1, ..., sent9 Type the name of the text or sentence to view it. Type:  
'texts()' or 'sents()' to list the materials. text1: Moby Dick by Herman Melville  
1851
```

```
text2: Sense and Sensibility by Jane Austen 1811 text3: The Book of  
Genesis  
text4: Inaugural Address Corpus text5: Chat  
Corpus  
text6: Monty Python and the Holy Grail text7: Wall Street  
Journal  
text8: Personals Corpus  
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

We will be working with text6, "Monty Python and the Holy Grail" (the screenplay for the 1975 movie) in all of the following examples.

Text objects can be manipulated much like normal Python arrays, as if they were an array containing words of the text. Using this property, you can count the number of unique words in a text and compare it against the total number of words:

```
>>> len ( text6 ) / len ( words )  
7.833333333333333
```

The preceding shows that each word in the script was used about eight times on average. You can also put the text into a frequency distribution object to see what some of the most common words are and the frequencies for various words:

```
>>> from nltk import FreqDist  
>>> fdist = FreqDist ( text6 )  
>>> fdist . most_common ( 10 )  
[( 'I' , 1197 ), ( 'I' , 816 ), ( 'I' , 801 ), ( 'I' , 731 ), ( "" , 421 ), ( 'T' , 319 ), ( 'T' , 312 ), ( 'the' , 299 ), ( 'I' , 255 ), ( 'ARTHUR' , 225 )]  
>>> fdist [ "Grail" ]  
34
```

Because this is a screenplay, some artifacts of how it is written can pop up. For instance, "ARTHUR" in all caps crops up frequently because it appears before each of King Arthur's lines in the script. In addition, a colon (:) appears before every single line, acting as a separator between the name of the character and the character's line. Using this fact, we can see that there are 1,197 lines in the movie!

What we have called 2-grams in previous chapters NLTK refers to as bigrams (from time to time you might also hear 3-grams referred to as “trigrams,” but I personally prefer n-gram rather than bigram or trigram). You can create, search, and list 2- grams extremely easily:

```
>>> from nltk import bigrams  
>>> bigrams = bigrams( text6 )  
>>> bigramsDist = FreqDist( bigrams )  
>>> bigramDist[("Sir", "Robin")]  
18
```

To search for the 2-grams “Sir Robin” we need to break it up into an array (“Sir”, “Robin”), to match the way the 2-grams are represented in the frequency distribution. There is also a trigrams module that works in the exact same way. For the general case, you can also import the ngrams module:

```
>>> from nltk import ngrams  
>>> fourgrams = ngrams( text6, 4 )  
>>> fourgramsDist = FreqDist( fourgrams )  
>>> fourgramsDist[("father", "smelt", "of", "elderberries")]  
1
```

Here, the ngrams function is called to break up a text object into n-grams of any size, governed by the second parameter. In this case, I’m breaking the text into 4- grams. Then, I can demonstrate that the phrase “father smelt of elderberries” occurs in the screenplay exactly once.

Frequency distributions, text objects, and n-grams also can be iterated through and operated on in a loop. The following prints out all 4-grams that begin with the word “coconut,” for instance:

```
from nltk.book import *  
from nltk import ngrams fourgrams = ngrams  
( text6, 4 )  
for fourgram in fourgrams :  
    if fourgram[0] == "coconut" :  
        print( fourgram )
```

The NLTK library has a vast array of tools and objects designed to organize, count, sort, and measure large swaths of text. Although we’ve barely scratched the surface of their uses, most of these tools are very well designed and operate rather intuitively for someone familiar with Python.

Lexicographical Analysis with NLTK

So far, we’ve compared and categorized all the words we’ve encountered based only on the value they represent by themselves. There is no differentiation between homonyms or the context in which the words are used.

Although some people might be tempted to dismiss homonyms as rarely problematic, you might be surprised at how frequently they crop up. Most native English speakers probably don't even register that a word is a homonym, much less consider that it might possibly be confused for another word in a different context.

"He was objective in achieving his objective of writing an objective philosophy, pri- marily using verbs in the objective case" is easy for humans to parse but might make a web scraper think the same word is being used four times and cause it to simply dis- card all the information about the meaning behind each word.

In addition to sussing out parts of speech, being able to distinguish a word being used in one way versus another might be useful. For example, you might want to look for company names made up of common English words, or analyze someone's opinions about a company "ACME Products is good" and "ACME Products is not bad" can have the same meaning, even if one sentence uses "good" and the other uses "bad".

Penn Treebank's Tags

NLTK uses by default a popular system of tagging parts of speech developed by the University of Pennsylvania's [Penn Treebank Project](#). Although some of the tags make sense (e.g., CC is a coordinating conjunction), others can be confusing (e.g., RP is a particle). Use the following as a reference for the tags referred to in this section:

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential "there"
FW	Foreign word
IN	Preposition, subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural

PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	"to"
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle Verb,
VBN	past participle
VBP	Verb, non-third person singular present Verb,
VBZ	third person singular present wh-determiner
WDT	
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb

In addition to measuring language, NLTK can assist in finding meaning in the words based on context and its own very large dictionaries. At a basic level, NLTK can identify parts of speech:

```
>>> from nltk.book import *
>>> from nltk import word_tokenize
>>> text = word_tokenize ("Strange women lying in ponds distributing swords is no
basis for a system of government . Supreme executive power derives from a mandate
from the masses , not from some farcical aquatic ceremony .")
>>> from nltk import pos_tag
>>> pos_tag ( text )
[('Strange', 'NNP'), ('women', 'NNS'), ('lying', 'VBG'), ('in', 'IN'),
 ('ponds', 'NNS'), ('distributing', 'VBG'), ('swords', 'NNS'), ('is',
 'VBZ'), ('no', 'DT'), ('basis', 'NN'), ('for', 'IN'), ('a', 'DT'),
 ('system', 'NN'), ('of', 'IN'), ('government', 'NN'), ('.', '.'), ('$',
 '$')] 
```

```
supremo ',' NNP ), ( ' executivo ',' NN ), ( ' poder ',' NN ), ( ' deriva ',' N NS ), ( ' de ',' DENTRO ), ( ' uma ',' DT ), ( ' mandato
',' NN ), ( ' de ',' DENTRO ),  
    ( 'a' , 'DT' ), ( 'massas' , 'NNS' ), ( ' ', ' ' ), ( 'não' , 'RB' ), ( 'de'  
    , ' DENTRO' ), ( 'alguns' , 'DT' ), ( 'farsesco' , 'JJ' ), ( 'aquático' , 'JJ' ), ( 'cer  
emony' , 'NN' ), ( ' . ' , ' ' )]
```

Cada palavra é separada em um *tupla* contendo a palavra e uma tag que identifica a classe gramatical (consulte a barra lateral de Penn Treebank Tags anterior para obter mais informações sobre essas tags). Embora isso possa parecer uma pesquisa muito direta, a complexidade necessária para executar a tarefa corretamente se torna aparente com o seguinte exemplo:

```
>>> texto = word_tokenize( "A poeira estava grossa, então ele teve que espanar" )
>>> pos_tag( texto )
[( 'O' , 'DT' ), ( 'poeira' , 'NN' ), ( 'estava' , 'VBD' ), ( 'Grosso' , 'JJ' ), ( 'então' , ' RB' ), ( 'ele' , 'PRP' ), ( 'teve' , 'VBD' ), ( 'para
',' PARA' ), ( 'poeira' , 'VB' )
]
```

Observe que a palavra “poeira” é usada duas vezes na frase: uma vez como substantivo e novamente como verbo. O NLTK identifica ambos os usos corretamente, com base em seu contexto na frase. NLTK identifica classes gramaticais usando um *gramática livre de contexto* definido pela língua inglesa. Gramáticas livres de contexto são, essencialmente, conjuntos de regras que definem quais coisas têm permissão para seguir quais outras coisas em listas ordenadas. Nesse caso, eles definem quais classes gramaticais podem seguir quais classes gramaticais. Sempre que uma palavra ambígua como “poeira” é encontrada, as regras da gramática livre de contexto são consultadas e uma classe gramatical apropriada que segue as regras é selecionada.

Aprendizado de máquina e treinamento de máquina

Você pode fazer com que o NLTK gere novas gramáticas livres de contexto ao treiná-lo, por exemplo, em uma língua estrangeira. Se você marcar grandes seções de texto à mão na linguagem usando os Penn Treebank Tags apropriados, você pode alimentá-los de volta no NLTK e treiná-lo para marcar apropriadamente outro texto que possa encontrar. Este tipo de treinamento é um componente necessário de qualquer atividade de aprendizado de máquina que revisitaremos em [Capítulo 11](#), ao treinar scrapers para reconhecer caracteres CAPTCHA.

Então, de que adianta saber se uma palavra é um verbo ou um substantivo em um determinado contexto? Pode ser legal em um laboratório de pesquisa de ciência da computação, mas como isso ajuda na eliminação da web?

Um problema muito comum em web scraping está relacionado à pesquisa. Você pode estar copiando texto de um site e deseja pesquisar por instâncias da palavra “google”, mas apenas quando está sendo usada como um verbo, não um substantivo próprio. Ou você pode estar procurando apenas por instâncias da empresa Google e não quer contar com o uso correto de

letras maiúsculas para localizar essas instâncias. Aqui o pos_tag função pode ser extremamente útil:

```
de nltk importar word_tokenize , sent_tokenize , frases pos_tag = sent_tokenize ( "O Google é uma das melhores empresas do mundo." )
```

```
Eu constantemente procuro no Google para ver o que eu estou fazendo. "
```

```
substantivos = [ 'NN' , 'NNS' , 'NNP' , 'NNPS' ]
```

```
para frase dentro frases :
```

```
    E se " Google" dentro frase . mais baixo ():
```

```
        taggedWords = pos_tag ( word_tokenize ( frase ))
```

```
        para palavra dentro taggedWords :
```

```
            E se palavra [ 0 ] . mais baixo () == "Google" e palavra [ 1 ] dentro substantivos :
```

```
                impressão ( frase )
```

Isso imprime apenas frases que contenham a palavra “google” (ou “Google”) como algum tipo de substantivo, não um verbo. Claro, você poderia ser mais específico e exigir que apenas as instâncias do Google marcadas com “NNP” (um nome próprio) sejam impressas, mas até mesmo o NLTK comete erros às vezes, e pode ser bom ter um pouco de espaço para mexer, dependendo no aplicativo.

Muito da ambigüidade da linguagem natural pode ser resolvida usando o NLTK pos_tag função. Ao pesquisar o texto não apenas por ocorrências de sua palavra ou frase alvo, mas por ocorrências de sua palavra ou frase alvo *mais* sua etiqueta, você pode aumentar muito a precisão e eficácia das pesquisas do seu raspador.

Recursos adicionais

Processar, analisar e compreender a linguagem natural por máquina é uma das tarefas mais difíceis na ciência da computação, e incontáveis volumes e trabalhos de pesquisa foram escritos sobre o assunto. Espero que a cobertura aqui o inspire a pensar além da web scraping convencional, ou pelo menos dê alguma orientação inicial sobre por onde começar ao realizar um projeto que requer análise de linguagem natural.

Existem muitos recursos excelentes sobre o processamento introdutório de linguagem e o Natural Language Toolkit do Python. Em particular, o livro de Steven Bird, Ewan Klein e Edward Loper *Processamento de linguagem natural com Python* apresenta uma abordagem abrangente e introdutória ao tópico.

Além disso, James Pustejovsky e Amber Stubbs ' *Anotações de linguagem natural para aprendizado de máquina* fornece um guia teórico ligeiramente mais avançado. Você precisará de um conhecimento de Python para implementar as lições; os tópicos abordados funcionam perfeitamente com o Natural Language Toolkit do Python.

Rastreamento por meio de formulários e logins

Uma das primeiras perguntas que surge quando você começa a ir além do básico de web scraping é: "Como faço para acessar informações atrás de uma tela de login?" A Web está cada vez mais se movendo em direção à interação, mídia social e conteúdo gerado pelo usuário. Formulários e logins são parte integrante desses tipos de sites e quase impossíveis de evitar. Felizmente, eles também são relativamente fáceis de lidar.

Até este ponto, a maioria de nossas interações com servidores da web em nossos scrapers de exemplo consistia no uso de HTTP PEGUE para solicitar informações. Neste capítulo, vamos nos concentrar no POSTAR método que envia informações a um servidor web para armazenamento e análise.

Os formulários basicamente fornecem aos usuários uma maneira de enviar um POSTAR solicitar que o servidor da web possa entender e usar. Assim como as tags de link em um site ajudam os usuários a formatar PEGUE pedidos, os formulários HTML os ajudam a POSTAR solicitações de. Claro, com um pouco de codificação, é possível simplesmente criar essas solicitações nós mesmos e enviá-las com um raspador.

Biblioteca de solicitações Python

Embora seja possível navegar em formulários da web usando apenas as bibliotecas centrais do Python, às vezes um pouco de açúcar sintático torna a vida muito mais agradável. Quando você começa a fazer mais do que um básico PEGUE pedido com `urllib` pode ajudar olhar para fora das bibliotecas centrais do Python.

o [Biblioteca de solicitações](#) é excelente para lidar com solicitações HTTP complicadas, cookies, cabeçalhos e muito mais.

Aqui está o que o criador do Requests, Kenneth Reitz, tem a dizer sobre as principais ferramentas do Python:

Padrão do Python urllib2 O módulo fornece a maioria dos recursos HTTP de que você precisa, mas a API está totalmente corrompida. Foi construído para uma época diferente - e uma rede diferente. Requer uma enorme quantidade de trabalho (até mesmo substituições de métodos) para realizar as tarefas mais simples.

As coisas não deveriam ser assim. Não em Python.

Como acontece com qualquer biblioteca Python, a biblioteca Requests pode ser instalada com qualquer gerenciador de biblioteca Python de terceiros, como pip, ou baixando e instalando o [arquivo fonte](#).

Envio de um formulário básico

A maioria dos formulários da web consiste em alguns campos HTML, um botão de envio e uma página de “ação”, onde o processamento real do formulário é feito. Os campos HTML geralmente consistem em texto, mas também podem conter um upload de arquivo ou algum outro conteúdo não textual.

Os sites mais populares bloqueiam o acesso aos seus formulários de login em seus *robots.txt* Arquivo ([apêndice C](#) discute a legalidade da remoção de tais formulários), então, para jogar pelo seguro, construí uma série de diferentes tipos de formulários e logins em pythonscraping.com contra os quais você pode executar seus web scrapers. O mais básico desses formulários está localizado em <http://pythonscraping.com/pages/files/form.html>.

Todo o formulário é:

```
<formulário método = "postar" ação = "processing.php" >
Primeiro nome: < entrada tipo = "texto" nome = "primeiro nome" > < br>
Último nome: < entrada tipo = "texto" nome = "último nome" > < br> <input tipo = "enviar"
valor = "Enviar" >
</form>
```

Algumas coisas a serem observadas aqui: primeiro, o nome dos dois campos de entrada são primeiro nome e último nome. Isso é importante. Os nomes desses campos determinam os nomes dos parâmetros variáveis que serão POSTAR ed para o servidor quando o formulário é enviado. Se você quiser imitar a ação que o formulário executará quando POSTAR com seus próprios dados, você precisa se certificar de que os nomes das variáveis correspondem.

A segunda coisa a notar é que a ação do formulário está realmente em *process-ing.php* (o caminho absoluto é <http://pythonscraping>) Qualquer postar

as solicitações ao formulário devem ser feitas em *esta* página, não na página que o próprio formulário reside. Lembre-se: o objetivo dos formulários HTML é apenas ajudar os visitantes do site a formular solicitações adequadas para enviar à página que executa a ação real. A menos que você esteja fazendo uma pesquisa para formatar a solicitação, não precisa se preocupar muito com a página em que o formulário pode ser encontrado.

O envio de um formulário com a biblioteca Requests pode ser feito em quatro linhas, incluindo a importação e a instrução para imprimir o conteúdo (sim, é muito fácil):

```

importar solicitações de

params = { 'primeiro nome' : 'Ryan' , 'último nome' : 'Mitchell' }
r = solicitações de . postar ( "http://pythonscraping.com/files/processing.php" , dados = params )
impressão ( r . texto )

```

Após o envio do formulário, o script deverá retornar com o conteúdo da página:

```
Olá, Ryan Mitchell!
```

Este script pode ser aplicado a muitos formulários simples encontrados na Internet. O formulário para se inscrever no boletim informativo da O'Reilly Media, por exemplo, é assim:

```

<formulário ação = "http://post.oreilly.com/client/o/oreilly/forms/
    quicksignup.cgi" id = "example_form2" método = "POSTAR" >
    <input nome = "client_token" tipo = "escondido" valor = "oreilly" />
    <input nome = "se inscrever" tipo = "escondido" valor = "optin" />
    <input nome = "success_url" tipo = "escondido" valor = "http://oreilly.com/store/
        newsletter-thankyou.html" />
    <input nome = "error_url" tipo = "escondido" valor = "http://oreilly.com/store/
        newsletter-signup-error.html" />
    <input nome = "topic_or_dod" tipo = "escondido" valor = "1" />
    <input nome = "fonte" tipo = "escondido" valor = "orm-home-t1-dotd" />
    <fieldset>
        <input classe = "email_address long" maxlength = "200" nome =
            "email_addr" tamanho = "25" tipo = "texto" valor =
            "Digite seu email aqui" />
        <botão alt = "Junte-se" classe = "magro" nome = "enviar" onclick =
            "return addClickTracking ('orm', 'ebook', 'rightrail', 'dod'
            );" valor = "enviar" > Junte-se </ botão>
    </fieldset>
</form>

```

Embora possa parecer assustador no início, lembre-se de que, na maioria dos casos (veremos as exceções mais tarde), você está procurando apenas duas coisas:

- O nome do campo (ou campos) que você deseja enviar com os dados (neste caso, o nome é endereço de e-mail)
- O atributo de ação do próprio formulário; ou seja, a página em que o formulário realmente posta (neste caso, <http://post.oreilly.com/client/o/oreilly/forms/quicksignup.cgi>)

Basta adicionar as informações necessárias e executá-las:

```

importar solicitações de
params = { 'email_addr' : 'ryan.e.mitchell@gmail.com' }
r = solicitações de . postar ( "http://post.oreilly.com/client/o/oreilly/forms/
    inscrição rápida . cgi" , dados = parâmetros)
impressão ( r . texto )

```

Nesse caso, o site retornado é simplesmente outro formulário a ser preenchido, antes que você possa realmente entrar na lista de mala direta de O'Reilly, mas o mesmo conceito pode ser aplicado a

essa forma também. No entanto, eu solicitaria que você use seus poderes para o bem, e não envie spam para o editor com inscrições inválidas, se quiser tentar isso em casa.

Botões de rádio, caixas de seleção e outras entradas

Obviamente, nem todos os formulários da web são uma coleção de campos de texto seguidos por um botão de envio. O HTML padrão contém uma grande variedade de campos de entrada de formulário possíveis: botões de rádio, caixas de seleção e caixas de seleção, para citar alguns. No HTML5, há a adição de controles deslizantes (campos de entrada de intervalo), e-mail, datas e muito mais. Com campos JavaScript personalizados, as possibilidades são infinitas, com seletores de cores, calendários e tudo o mais que os desenvolvedores criarem.

Independentemente da aparente complexidade de qualquer tipo de campo de formulário, existem apenas duas coisas com as quais você precisa se preocupar: o nome do elemento e seu valor. O nome do elemento pode ser facilmente determinado observando o código-fonte e encontrando o nome atributo. O valor às vezes pode ser mais complicado, pois pode ser preenchido por JavaScript imediatamente antes do envio do formulário. Colorpickers, como exemplo de um campo de formulário bastante exótico, provavelmente terá um valor de algo como # F03030.

Se você não tiver certeza do formato do valor de um campo de entrada, há uma série de ferramentas que você pode usar para rastrear o PEGUE e POSTAR solicitações que seu navegador está enviando de e para sites. A melhor e talvez mais óbvia maneira de rastrear PEGUE solicitações, como mencionado antes, é simplesmente olhar para a URL de um site. Se o URL for algo como:

```
http://domainname.com?thing1=foo&thing2=bar
```

Você sabe que isso corresponde a uma forma deste tipo:

```
<formulário método = "PEGUE" ação = "someProcessor.php" >
<input tipo = "someCrazyInputType" nome = "Coisa 1" valor = "foo" />
<input tipo = "anotherCrazyInputType" nome = "coisa2" valor = "Barra" />
<input tipo = "enviar" valor = "Enviar" />
</form>
```

Que corresponde ao objeto de parâmetro Python:

```
{'coisa1': 'foo', 'coisa2': 'bar'}
```

Você pode ver isso em [Figura 9-1](#).

Se você está preso a um POSTAR formulário e você deseja ver exatamente quais parâmetros seu navegador está enviando ao servidor, a maneira mais fácil é usar o inspetor do navegador ou a ferramenta de desenvolvedor para visualizá-los.

The screenshot shows a browser window with the URL `localhost:8888/someProcessor.php`. Below the address bar is a form with two input fields: `foo` and `bar`, followed by a `Submit` button. At the bottom of the browser, the developer tools Network tab is open, showing a list of requests. One request, `someProcessor.php`, is selected. The right pane displays the request details: Remote Address: `[::1]:8888`, Request URL: `http://localhost:8888/someProcessor.php`, Request Method: `POST`, Status Code: `200 OK`. A red box highlights the `Form Data` section under `Request Headers (11)`, which contains the key-value pairs `thing1: foo` and `thing2: bar`.

Figura 9-1. A seção Form Data, destacada em uma caixa, mostra os parâmetros POST “thing1” e “thing2” com seus valores “foo” e “bar”

A ferramenta de desenvolvedor do Chrome pode ser acessada por meio do menu em Exibir → Desenvolvedor → Ferramentas do desenvolvedor. Ele fornece uma lista de todas as consultas que seu navegador produz enquanto interage com o site atual e pode ser uma boa maneira de visualizar a composição dessas consultas em detalhes.

Envio de arquivos e imagens

Embora os uploads de arquivos sejam comuns na Internet, eles não são usados com frequência em web scraping. É possível, entretanto, que você queira escrever um teste para seu próprio site que envolva o upload de um arquivo. De qualquer forma, é uma coisa útil saber fazer.

Existe um formulário de upload de arquivo de prática em <http://pythonscraping/files/form2.html>. O formulário na página possui a seguinte marcação:

```
<formulário ação = "processing2.php" método = "postar" enctype = "multipart / form-data" >
    Envie um jpg, png ou gif: < entrada tipo = "Arquivo" nome = "imagem" > < br > < input tipo = "enviar"
    valor = "Subir arquivo" >
</form>
```

Exceto para o `< entrada>` tag tendo o atributo type Arquivo, parece essencialmente igual aos formulários baseados em texto usados nos exemplos anteriores. Felizmente, a maneira como os formulários são usados pela biblioteca Python Requests também é muito semelhante:

```
importar solicitações de  
  
arquivos = { 'subir arquivo' : abrir ( '..files/Python-logo.png' , 'rb' )}  
r = solicitações de . postar ( "http://pythonscraping.com/pages/processing2.php" ,  
    arquivos = arquivos )  
impressão ( r . texto )
```

Observe que, em vez de uma string simples, o valor enviado para o campo do formulário (com o nome subir arquivo) agora é um objeto Arquivo Python, conforme retornado pelo abrir função. Neste exemplo, estou enviando um arquivo de imagem, armazenado em minha máquina local, no caminho *.. files / Python-logo.png*, em relação ao local de onde o script Python está sendo executado.

Sim, é realmente tão fácil!

Tratamento de logins e cookies

Até agora, discutimos principalmente os formulários que permitem enviar informações a um site ou visualizar as informações necessárias na página imediatamente após o formulário. Como isso é diferente de um formulário de login, que permite que você exista em um estado permanente de “conectado” durante a sua visita ao site?

A maioria dos sites modernos usa cookies para rastrear quem está conectado e quem não está. Depois que um site autentica suas credenciais de login, ele armazena em seu navegador um cookie, que geralmente contém um token gerado pelo servidor, tempo limite e informações de rastreamento. O site então usa esse cookie como uma espécie de prova de autenticação, que é mostrado a cada página que você visita durante sua permanência no site. Antes do uso generalizado de cookies em meados dos anos 90, manter os usuários autenticados com segurança e rastreá-los era um grande problema para os sites.

Embora os cookies sejam uma ótima solução para desenvolvedores da web, eles podem ser problemáticos para os web scrapers. Você pode enviar um formulário de login o dia todo, mas se você não controlar o cookie que o formulário envia de volta para você depois, a próxima página que você visitar agirá como se você nunca tivesse feito login.

Eu criei um formulário de login simples em <http://pythonscraping.com/pages/cookies/login.html> (o nome de usuário pode ser qualquer coisa, mas a senha deve ser “senha”).

Este formulário é processado em <http://pythonscraping.com/pages/cookies/welcome.php>, e contém um link para a página do “site principal”, <http://pythonscraping.com/pages/cookies/profile.php>.

Se você tentar acessar a página de boas-vindas ou a página de perfil sem fazer login primeiro, receberá uma mensagem de erro e instruções para fazer login antes de continuar. Na página de perfil, é feita uma verificação nos cookies do seu navegador para ver se o cookie foi definido na página de login.

Manter o controle dos cookies é fácil com a biblioteca Requests:

```

importar solicitações de

params = { 'nome do usuário' : 'Ryan' , 'senha' : 'senha' }
r = solicitações de . postar ( "http://pythonscraping.com/pages/cookies/welcome.php" , params )
impressão ( " Cookie está definido para: " )
impressão ( r . biscoitos . get_dict () )
impressão ( "-----" )
impressão ( " Indo para a página de perfil ... " )
r = solicitações de . pegue ( "http://pythonscraping.com/pages/cookies/profile.php" ,
    biscoitos = r . biscoitos )
impressão ( r . texto )

```

Aqui estou enviando os parâmetros de login para a página de boas-vindas, que atua como processadora do formulário de login. Eu recupero os cookies dos resultados da última solicitação, imprimo o resultado para verificação e, em seguida, envio-os para a página de perfil configurando o biscoitos argumento.

Isso funciona bem para situações simples, mas e se você estiver lidando com um site mais complicado que freqüentemente modifica os cookies sem aviso, ou se você preferir nem pensar nos cookies para começar? Os Pedidos sessão função funciona perfeitamente neste caso:

```

importar solicitações de

sessão = solicitações de . Sessão ()

params = { 'nome do usuário' : 'nome do usuário' , 'senha' : 'senha' }
s = sessão . postar ( "http://pythonscraping.com/pages/cookies/welcome.php" , params )
impressão ( " Cookie está definido para: " )
impressão ( s . biscoitos . get_dict () )
impressão ( "-----" )
impressão ( " Indo para a página de perfil ... " )
s = sessão . pegue ( "http://pythonscraping.com/pages/cookies/profile.php" )
impressão ( s . texto )

```

Neste caso, o objeto de sessão (recuperado chamando `solicitações.Sessão()`) mantém o controle das informações da sessão, como cookies, cabeçalhos e até mesmo informações sobre os protocolos que você pode estar executando em cima do HTTP, como `HTTPAdapters`.

Requests é uma biblioteca fantástica, perdendo talvez apenas para Selenium (que abordaremos em [Capítulo 10](#)) na integridade do que ele lida sem que os programadores tenham que pensar sobre isso ou escrever o código eles próprios. Embora possa ser tentador sentar e deixar a biblioteca fazer todo o trabalho, é extremamente importante estar sempre ciente da aparência dos cookies e o que eles estão controlando ao escrever web scrapers. Isso pode economizar muitas horas de depuração dolorosa ou descobrir por que um site está se comportando de maneira estranha!

Autenticação de acesso básico HTTP

Antes do advento dos cookies, uma maneira popular de lidar com logins era com HTTP *autenticação de acesso básico*. Você ainda vê isso de vez em quando, especialmente em sites corporativos ou de alta segurança e com algumas APIs. Eu criei uma página em <http://pythonscraping.com/pages/auth/login.php> que tem este tipo de autenticação (Figura 9-2)

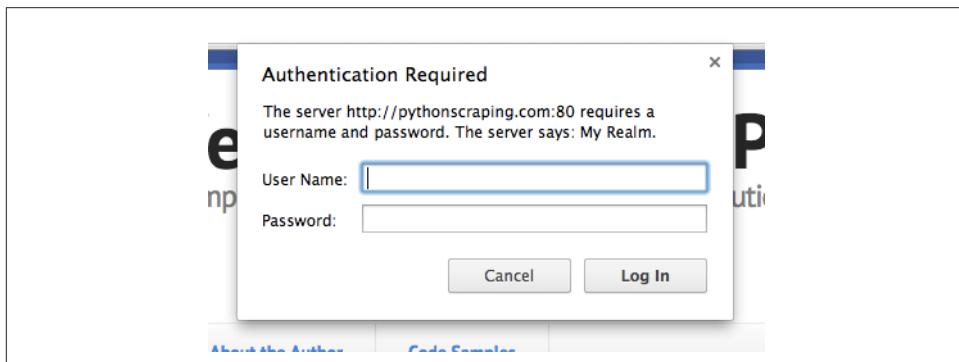


Figura 9-2. O usuário deve fornecer um nome de usuário e senha para chegar à página protegida por autenticação de acesso básico

Como de costume com esses exemplos, você pode fazer login com qualquer nome de usuário, mas a senha deve ser "senha".

O pacote de solicitações contém um auth módulo projetado especificamente para lidar com a autenticação HTTP:

```
importar solicitações de
de request.auth importar AuthBase
de request.auth importar HTTPBasicAuth

auth = HTTPBasicAuth ('ryan' , 'senha')
r = solicitações de . postar ( url = "http://pythonscraping.com/pages/auth/login.php" , auth =
                                auth )
impressão ( r . texto )
```

Embora isso pareça ser normal POSTAR pedido, um HTTPBasicAuth objeto é passado como o auth argumento no pedido. O texto resultante será a página protegida pelo nome de usuário e senha (ou uma página Acesso negado, se a solicitação falhou).

Outros FormProblems

Os formulários da Web são um ponto de entrada importante para bots maliciosos. Você não quer bots criando contas de usuário, ocupando um tempo valioso de processamento do servidor ou enviando comentários de spam em um blog. Por esse motivo, muitas vezes há uma série de recursos de segurança que são

incorporados em formulários HTML em sites modernos que podem não ser imediatamente aparentes.

Para obter ajuda com CAPTCHAs, verifique [Capítulo 11](#), que cobre o processamento de imagens e reconhecimento de texto em Python.

Se você encontrar um erro misterioso ou o servidor estiver rejeitando o envio do seu formulário por um motivo desconhecido, verifique [Capítulo 12](#), que cobre potes de mel, campos ocultos e outras medidas de segurança que os sites tomam para proteger seus formulários.

Raspando JavaScript

Linguagens de script do lado do cliente são linguagens que são executadas no próprio navegador, em vez de em um servidor da web. O sucesso de uma linguagem do lado do cliente depende da capacidade do seu navegador de interpretar e executar a linguagem corretamente. (É por isso que é tão fácil desativar o JavaScript em seu navegador.)

Em parte devido à dificuldade de fazer com que todos os fabricantes de navegadores concordem com um padrão, há muito menos linguagens do lado do cliente do que do lado do servidor. Isso é bom quando se trata de web scraping: quanto menos idiomas houver para lidar, melhor.

Na maioria das vezes, existem apenas duas linguagens que você encontrará com frequência online: ActionScript (que é usado por aplicativos Flash) e JavaScript. O ActionScript é usado com muito menos frequência hoje do que há 10 anos e é frequentemente usado para transmitir arquivos multimídia, como uma plataforma para jogos online ou para exibir páginas de "introdução" para sites que não receberam a dica de que não alguém quer assistir a uma página de introdução. De qualquer forma, como não há muita demanda para copiar páginas em Flash, irei me concentrar na linguagem do lado do cliente que é onipresente nas páginas da web modernas: JavaScript.

JavaScript é, de longe, a linguagem de script do lado do cliente mais comum e mais bem suportada na Web hoje. Ele pode ser usado para coletar informações para rastreamento do usuário, enviar formulários sem recarregar a página, incorporar multimídia e até mesmo criar jogos online inteiros. Mesmo as páginas aparentemente simples podem conter várias partes de JavaScript. Você pode encontrá-lo incorporado entre <script> tags no código-fonte da página:

```
< roteiro >
    alerta ( "Isso cria um pop-up usando JavaScript" );
< / script>
```

Uma breve introdução ao JavaScript

Ter pelo menos alguma ideia do que está acontecendo no código que você está copiando pode ser extremamente útil. Com isso em mente, é uma boa ideia se familiarizar com JavaScript.

JavaScript é uma linguagem fracamente tipada, com uma sintaxe que geralmente é comparada a C ++ e Java. Embora certos elementos da sintaxe, como operadores, loops e arrays, possam ser semelhantes, a digitação fraca e a natureza semelhante a script da linguagem podem torná-la uma besta difícil de lidar para alguns programadores.

Por exemplo, o seguinte calcula recursivamente os valores na sequência de Fibonacci e os imprime no console do desenvolvedor do navegador:

```
< roteiro >
função Fibonacci ( uma , b ) {
    var nextNum = uma + b ;
    console . registro ( nextNum + "está na sequência de Fibonacci" );
    E se ( nextNum < 100 ) {
        Fibonacci ( b , nextNum );
    }
}
Fibonacci ( 1 , 1 );
< / script>
```

Observe que todas as variáveis são demarcadas por um precedente var. Isso é semelhante ao \$ assinar em PHP ou a declaração de tipo (int, String, List, etc.) em Java ou C ++. Python é incomum por não ter esse tipo de declaração explícita de variável.

JavaScript também é extremamente bom para passar funções como variáveis:

```
< roteiro >
var Fibonacci = função () {
    var uma = 1 ;
    var b = 1 ;
    função de retorno () {
        var temp = b ;
        b = uma + b ;
        uma = temp ;
        Retorna b ;
    }
}
var fibInstance = Fibonacci ();
console . registro ( fibInstance () + "está na sequência de Fibonacci" );
console . registro ( fibInstance () + "está na sequência de Fibonacci" );
console . registro ( fibInstance () + "está na sequência de Fibonacci" );
< / script>
```

Isso pode parecer assustador no início, mas se torna simples se você pensar em termos de expressões lambda (abordadas em [Capítulo 2](#)) A variável Fibonacci é definido como uma função. O valor de sua função retorna uma função que imprime cada vez mais grande

valores na sequência de Fibonacci. Cada vez que é chamado, ele retorna a função de cálculo de Fibonacci, que é executada novamente e aumenta os valores na função.

Embora possa parecer complicado à primeira vista, alguns problemas, como o cálculo de valores de Fibonacci, tendem a se prestar a padrões como este. Passar funções como variáveis também é extremamente útil quando se trata de lidar com ações do usuário e callbacks, e vale a pena se familiarizar com este estilo de programação quando se trata de ler JavaScript.

Bibliotecas comuns de JavaScript

Embora seja importante conhecer a linguagem JavaScript central, você não pode ir muito longe na Web moderna sem usar pelo menos uma das muitas bibliotecas de terceiros da linguagem. Você pode ver uma ou mais dessas bibliotecas comumente usadas ao examinar o código-fonte da página.

Executar JavaScript usando Python pode consumir muito tempo e consumir muito do processador, especialmente se você estiver fazendo isso em grande escala. Conhecer o JavaScript e ser capaz de analisá-lo diretamente (sem a necessidade de executá-lo para adquirir as informações) pode ser extremamente útil e poupar muitas dores de cabeça.

jQuery

jQuery é uma biblioteca extremamente comum, usada por 70% dos sites mais populares da Internet e cerca de 30% do restante da Internet.¹ Um site que usa jQuery é prontamente identificável porque conterá uma importação para jQuery em algum lugar de seu código, como:

```
<script src = "http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js" ></script>
```

Se você descobrir que jQuery é encontrado em um site, deve ter cuidado ao copiá-lo. jQuery é adepto da criação dinâmica de conteúdo HTML que aparece somente depois que o JavaScript é executado. Se você copiar o conteúdo da página usando métodos tradicionais, irá recuperar apenas a página pré-carregada que aparece antes de o JavaScript ter criado o conteúdo (vamos cobrir este problema de eliminação em mais detalhes em “[Ajax e HTML dinâmico](#)” na [página 151](#))

Além disso, é mais provável que essas páginas contenham animações, conteúdo interativo e mídia incorporada que podem tornar a raspagem um desafio.

¹ Postagem no blog de Dave Methvin, “[The State of jQuery 2014](#)”, 13 de janeiro de 2014, contém uma análise mais detalhada das estatísticas.

Google Analytics

O Google Analytics é usado por cerca de 50% de todos os sites,² tornando-a talvez a biblioteca JavaScript mais comum e a ferramenta de rastreamento de usuário mais popular na Internet. Na verdade, ambos <http://pythonscraping.com> e <http://www.oreilly.com/> use o Google Analytics.

É fácil dizer se uma página está usando o Google Analytics. Ele terá JavaScript na parte inferior semelhante ao seguinte (retirado do site da O'Reilly Media):

```
</ - Google Analytics -> < tipo de script = "text /  
javascript" >  
  
var _gaq = _gaq || [];  
_gaq . empurrar ([ '_setAccount' , 'UA-4591498-1' ]);  
_gaq . empurrar ([ '_setDomainName' , 'oreilly.com' ]);  
_gaq . empurrar ([ '_addIgnoredRef' , 'oreilly.com' ]);  
_gaq . empurrar ([ '_setSiteSpeedSampleRate' , 50 ]);  
_gaq . empurrar ([ '_trackPageview' ]);  
  
( função () { var ga = documento . createElement ( 'roteiro' ); ga . tipo =  
'text / javascript' ; ga . assíncrono = verdadeiro ; ga . src = ( 'https:' ==  
documento . localização . protocolo ? 'https://ssl' : 'http://www' ) +  
'google-analytics.com / ga.js' ; var s =  
documento . getElementsByTagName ( 'roteiro' ) [ 0 ];  
s . parentNode . insertBefore ( ga , s ); } ) ();  
  
< / script>
```

Este script lida com o Google Analytics, cookies específicos usados para rastrear sua visita página a página. Às vezes, isso pode ser um problema para web scrapers projetados para executar JavaScript e manipular cookies (como os que usam Selenium, discutidos posteriormente neste capítulo).

Se um site usa o Google Analytics ou um sistema de análise da web semelhante e você não deseja que o site saiba que está sendo rastreado ou raspado, descarte todos os cookies usados para análises ou descarte os cookies completamente.

Google Maps

Se você já passou algum tempo na Internet, quase certamente já viu o Google Maps embutido em um site. Sua API torna extremamente fácil incorporar mapas com informações personalizadas em qualquer site.

Se você estiver copiando qualquer tipo de dados de localização, entender como o Google Maps funciona torna mais fácil obter coordenadas de latitude / longitude bem formatadas e até

² W3Techs, "Estatísticas de uso e participação de mercado do Google Analytics para sites" (<http://w3techs.com/technologies/details/ta-googleanalytics/all/all>).

endereços. Uma das maneiras mais comuns de denotar um local no Google Maps é por meio de um *marcador* (também conhecido como alfinete).

Os marcadores podem ser inseridos em qualquer mapa do Google usando códigos como o seguinte:

```
var marcador = Novo Google . mapas . Marcador ({  
    posição : Novo Google . mapas . LatLng ( - 25,363882 , 131.044922 ),  
    mapa : mapa ,  
    título : 'Algum texto do marcador'  
});
```

Python facilita a extração de todas as instâncias de coordenadas que ocorrem entre google.maps.LatLng (e) para obter uma lista de coordenadas de latitude / longitude.

Usando API de “geocodificação reversa” do Google , você pode resolver esses pares de coordenadas em endereços bem formatados para armazenamento e análise.

Ajax e HTML dinâmico

Até agora, a única maneira que tínhamos de nos comunicar com um servidor da web é enviando a ele algum tipo de solicitação HTTP por meio da recuperação de uma nova página. Se você já enviou um formulário ou recuperou informações de um servidor sem recarregar a página, provavelmente já usou um site que usa *Ajax*.

Ao contrário do que alguns acreditam, Ajax não é uma linguagem, mas um grupo de tecnologias usadas para realizar uma determinada tarefa (muito parecido com web scraping, pensando bem). Ajax significa Asynchronous JavaScript and XML, e é usado para enviar e receber informações de um servidor da web sem fazer uma solicitação de página separada. Observação: você nunca deve dizer “Este site será escrito em Ajax”. Seria correto dizer: “Este formulário usará Ajax para se comunicar com o servidor da web”.

Como Ajax, *HTML dinâmico* ou DHTML é uma coleção de tecnologias usadas para um propósito comum. DHTML é o código HTML, a linguagem CSS ou ambos que mudam devido a scripts do lado do cliente que alteram os elementos HTML na página. Um botão pode aparecer apenas depois que o usuário move o cursor, uma cor de fundo pode mudar com um clique ou uma solicitação Ajax pode acionar um novo bloco de conteúdo para carregar.

Observe que embora a palavra “dinâmico” esteja geralmente associada a palavras como “movendo” ou “mudando”, a presença de componentes HTML interativos, imagens em movimento ou mídia incorporada não necessariamente torna uma página DHTML, mesmo que possa parecer dinâmico. Além disso, algumas das páginas mais chatas e de aparência estática da Internet podem ter processos DHTML em execução nos bastidores que dependem do uso de JavaScript para manipular HTML e CSS.

Se você vasculhar um grande número de sites diferentes, logo se deparará com uma situação em que o conteúdo que você está visualizando em seu navegador não corresponde ao conteúdo que você vê no código-fonte que está recuperando do site. Você pode ver a saída de

seu raspador e coçar a cabeça, tentando descobrir para onde tudo o que você está vendo na mesma página em seu navegador desapareceu.

A página da web também pode ter uma página de carregamento que parece redirecioná-lo para outra página de resultados, mas você notará que o URL da página nunca muda quando esse redirecionamento acontece.

Ambos são causados por uma falha do seu raspador em executar o JavaScript que está fazendo a mágica acontecer na página. Sem o JavaScript, o HTML simplesmente fica lá, e o site pode parecer muito diferente do que parece em seu navegador da web, que executa o JavaScript sem problemas.

Existem vários prêmios de que uma página pode estar usando Ajax ou DHTML para alterar / carregar o conteúdo, mas em situações como esta, existem apenas duas soluções: raspar o conteúdo diretamente do JavaScript ou usar pacotes Python capazes de executar o próprio JavaScript, e raspe o site conforme você o visualiza em seu navegador.

Executando JavaScript em Python com Selenium

Selênio é uma ferramenta poderosa de web scraping desenvolvida originalmente para teste de sites. Hoje em dia, ele também é usado quando o retrato preciso dos sites - como eles aparecem em um navegador - é necessário. O Selenium funciona automatizando os navegadores para carregar o site, recuperar os dados necessários e até mesmo fazer capturas de tela ou afirmar que certas ações acontecem no site.

O Selenium não contém seu próprio navegador; ele requer integração com navegadores de terceiros para funcionar. Se você fosse executar o Selenium com o Firefox, por exemplo, você veria literalmente uma instância do Firefox abrindo em sua tela, navegaria até o site da Web e executaria as ações que você especificou no código. Embora possa ser legal de assistir, eu prefiro que meus scripts sejam executados silenciosamente em segundo plano, então eu uso uma ferramenta chamada **PhantomJS** em vez de um navegador real.

O PhantomJS é conhecido como um navegador “sem cabeça”. Ele carrega sites na memória e executa JavaScript na página, mas sem qualquer renderização gráfica do site para o usuário. Ao combinar Selenium com PhantomJS, você pode executar um web scraper extremamente poderoso que lida com cookies, JavaScript, cabeçalhos e tudo o mais que você precisa com facilidade.

Você pode baixar a biblioteca Selenium em [seu site](#) ou use um instalador de terceiros, como pip para instalá-lo a partir da linha de comando.

PhantomJS pode ser baixado de seu [local na rede Internet](#). Como o PhantomJS é um navegador completo (embora sem cabeçalho) e não uma biblioteca Python, ele requer um download e instalação para ser usado e não pode ser instalado com pip.

Embora existam muitas páginas que usam Ajax para carregar dados (principalmente o Google), criei uma página de amostra em <http://pythonscraping.com/pages/javascript/ajaxDemo.html> para

executar nossos raspadores contra. Esta página contém algum texto de amostra, codificado no HTML da página, que é substituído pelo conteúdo gerado por Ajax após um atraso de dois segundos. Se extraíssemos os dados desta página usando métodos tradicionais, obteríamos apenas a página de carregamento, sem realmente obter os dados que desejamos.

A biblioteca Selenium é uma API chamada no a *WebDriver*. O WebDriver é um pouco como um navegador, pois pode carregar sites, mas também pode ser usado como um objeto BeautifulSoup para localizar elementos da página, interagir com os elementos da página (enviar texto, clicar, etc.) e realizar outras ações para conduzir os raspadores de teia.

O código a seguir recupera o texto atrás de uma "parede" Ajax na página de teste:

```
de selênio importar webdriver
importar Tempo

motorista = webdriver . PhantomJS ( executable_path = " ")
motorista . pegue ( "http://pythonscraping.com/pages/javascript/ajaxDemo.html" )
Tempo . dormir ( 3 )
impressão ( motorista . find_element_by_id ( "conteúdo" ) . texto )
motorista . Fechar ()
```

Selenium Selenium

Nos capítulos anteriores, selecionamos elementos de página usando seletores BeautifulSoup, como encontrar e encontrar tudo. O Selenium usa um conjunto inteiramente novo de seletores para localizar um elemento no DOM de um WebDriver, embora eles tenham nomes bastante simples.

No exemplo, usamos o seletor `find_element_by_id`, embora os outros seletores a seguir também funcionassem:

```
motorista . find_element_by_css_selector ( "#conteúdo" )
motorista . find_element_by_tag_name ( "div" )
```

Claro, se você deseja selecionar vários elementos na página, a maioria desses seletores de elemento pode retornar uma lista Python de elementos simplesmente usando elementos (ou seja, torná-lo plural):

```
motorista . find_elements_by_css_selector ( "#conteúdo" )
motorista . find_elements_by_css_selector ( "div" )
```

Claro, se você ainda deseja usar o BeautifulSoup para analisar este conteúdo, você pode, usando o WebDriver fonte da página função, que retorna a fonte da página, conforme visualizada pelo DOM naquele momento, como uma string:

```
fonte da página = motorista . fonte da página
bsObj = BeautifulSoup ( fonte da página )
impressão ( bsObj . encontrar ( Eu iria = "conteúdo" ) . get_text ())
```

Isso cria um novo SeleniumWebDriver, usando a biblioteca PhantomJS, que diz ao WebDriver para carregar uma página e, em seguida, pausa a execução por três segundos antes de olhar para a página para recuperar o conteúdo (provavelmente carregado).

Dependendo da localização da instalação do PhantomJS, você também pode precisar apontar explicitamente o Selenium na direção certa ao criar um novo PhantomJS Web Driver:

```
motorista = webdriver . PhantomJS ( executable_path = '/caminho / para / download /  
fantasmas - 1.9 . 8 - Mac OS X / bin / fantasmas ' )
```

Se tudo estiver configurado corretamente, o script deve levar alguns segundos para ser executado e resultar no seguinte texto:

Aqui está um texto importante que você deseja recuperar! Um botão para clicar!

Observe que, embora a própria página contenha um botão HTML, o do Selenium. texto A função recupera o valor do texto do botão da mesma forma que recupera todos os outros conteúdos da página.

Se o hora de dormir a pausa é alterada para um segundo em vez de três, o texto retornado muda para o original:

Este é um conteúdo que aparecerá na página durante o carregamento. Você não se preocupa em raspar isso.

Embora essa solução funcione, ela é um tanto ineficiente e sua implementação pode causar problemas em grande escala. Os tempos de carregamento da página são inconsistentes, dependendo da carga do servidor em qualquer milissegundo específico, e ocorrem variações naturais na velocidade de conexão. Embora o carregamento da página demore pouco mais de dois segundos, estamos dando a ele três segundos inteiros para garantir que carregue completamente. Uma solução mais eficiente seria verificar repetidamente a existência de algum elemento em uma página totalmente carregada e retornar somente quando esse elemento existir.

Este código usa a presença do botão com id loadButton para declarar que o a página foi totalmente carregada: do selênia import webdriver.

```
de selenium.webdriver.common.by importar De  
de selenium.webdriver.support.ui importar WebDriverWait  
de selenium.webdriver.support importar condições_esperadas Como CE  
  
motorista = webdriver . PhantomJS ( executable_path = " )  
motorista . pegue ( "http://pythonscraping.com/pages/javascript/ajaxDemo.html" )  
experimentar :  
    elemento = WebDriverWait ( motorista , 10 ) . até (  
        CE . Presence_of_element_located ( ( De . EU IRIA , "carregadoButton" ))  
    finalmente :  
        impressão ( motorista . find_element_by_id ( "conteúdo" ) . texto )  
    motorista . Fechar ()
```

Existem várias novas importações neste script, principalmente WebDriverWait e condições_esperadas, ambos são combinados aqui para formar o que o Selênio chama de *espera implícita*.

Uma espera implícita difere de uma espera explícita porque espera que algum estado no DOM ocorra antes de continuar, enquanto uma espera explícita define um tempo codificado como no exemplo anterior, que tem uma espera de três segundos. Em uma espera implícita, o estado de disparo do DOM é definido por condição_esperada (observe que a importação é convertida para CE aqui, uma convenção comum usada para brevidade). As condições esperadas podem ser muitas coisas na biblioteca Selenium, entre elas:

- Uma caixa de alerta aparece
- Um elemento (como uma caixa de texto) é colocado em um estado "selecionado"
- O título da página muda, ou algum texto agora é exibido na página ou em um elemento específico
- Um elemento agora está visível para o DOM ou um elemento desaparece do DOM

Obviamente, a maioria dessas condições esperadas exige que você especifique um elemento a ser observado em primeiro lugar. Os elementos são especificados usando *localizadores*. Observe que os localizadores não são iguais aos seletores (consulte a barra lateral anterior para obter mais informações sobre os seletores). Um localizador é uma linguagem de consulta abstrata, usando o De objeto, que pode ser usado de várias maneiras, incluindo para fazer seletores.

No código de exemplo a seguir, um localizador é usado para encontrar elementos com o id carregado
Botão:

```
CE . Presence_of_element_located (( De . EU IRIA , "carregadoButton" ))
```

Localizadores também podem ser usados para criar seletores, usando o find_element Função WebDriver:

```
impressão ( motorista . find_element ( De . EU IRIA , "conteúdo" ) . texto )
```

O que é, obviamente, funcionalmente equivalente à linha no código de exemplo:

```
impressão ( motorista . find_element_by_id ( "conteúdo" ) . texto )
```

Se você não precisa usar um localizador, não; isso vai economizar uma importação. No entanto, é uma ferramenta muito útil, usada para uma variedade de aplicações e com um alto grau de flexibilidade.

As seguintes estratégias de seleção de localizador podem ser usadas com o De objeto:

EU IRIA

Usado no exemplo; encontra elementos por seu HTML Eu iria atributo.

NOME DA CLASSE

Usado para encontrar elementos por seu HTML classe atributo. Por que esta função

NOME DA CLASSE e não simplesmente CLASSE? Usando o formulário object.CLASS criaria problemas para a biblioteca Java do Selenium, onde. classe é um método reservado. Para manter a sintaxe do Selenium consistente entre as diferentes linguagens, NOME DA CLASSE foi usado em seu lugar.

CSS_SELECTOR

Encontre elementos por seus classe, id, ou etiqueta nome, usando o # idName, .className, tagName convenção.

TEXTO DO LINK

Encontra HTML < a> tags pelo texto que contêm. Por exemplo, um link que diz “Próximo” pode ser selecionado usando (PorLINK_TEXT, “Próximo”).

PARTIAL_LINK_TEXT

Igual a TEXTO DO LINK, mas corresponde a uma string parcial.

NOME

Encontra tags HTML por seu nome atributo. Isso é útil para formulários HTML.

TAG_NAME

Encontra tags HTML por seu nome de tag.

XPATH

Usa um XPath expressão (cuja sintaxe é descrita na próxima barra lateral) para selecionar os elementos correspondentes.

Sintaxe XPath

XPath (abreviação de XML Path) é uma linguagem de consulta usada para navegar e selecionar partes de um documento XML. Foi fundado pelo W3C em 1999 e é ocasionalmente usado em linguagens como Python, Java e C # ao lidar com documentos XML.

Embora o BeautifulSoup não ofereça suporte a XPath, muitas das outras bibliotecas deste livro oferecem. Muitas vezes, pode ser usado da mesma forma que os seletores CSS (como `mytag # idname`), embora seja projetado para funcionar com documentos XML mais generalizados, em vez de documentos HTML em particular.

Existem quatro conceitos principais na sintaxe XPath:

- *Nós raiz versus nós não raiz*
 - `/ div` irá selecionar o nó div apenas se estiver na raiz do documento
 - `// div` seleciona todos os divs em qualquer lugar do documento
- *Seleção de atributo*
 - `// @ href` seleciona qualquer nó com o atributo href
 - `//a[@href='http://google.com ']` seleciona todos os links no documento que apontam para o Google
- *Seleção de nós por posição*
 - `// a [3]` seleciona o terceiro link no documento
 - `// tabela [última ()]` seleciona a última tabela do documento
 - `// a [posição () <3]` seleciona os três primeiros links no documento
- *Asteriscos (*) correspondem a qualquer conjunto de caracteres ou nós e podem ser usados em uma variedade de situações*
 - `// tabela / tr / *` seleciona todos os filhos de tr tags em todas as tabelas (isso é bom para selecionar células usando ambos o e td Tag)
 - `// div [@ *]` seleciona tudo div tags que possuem quaisquer atributos

Claro, existem muitos recursos avançados da sintaxe XPath. Ao longo dos anos, ele se desenvolveu em uma linguagem de consulta relativamente complicada, com lógica booleana, funções (como `posição()`), e uma variedade de operadores não discutidos aqui.

Se você tiver um problema de seleção de HTML ou XML que não pode ser resolvido pelas funções mostradas aqui, consulte [Página de sintaxe XPath da Microsoft](#).

Tratamento de redirecionamentos

Redirecionamentos do lado do cliente são redirecionamentos de página que são executados em seu navegador por Java- Script, ao invés de um redirecionamento executado no servidor, antes que o conteúdo da página seja enviado. Às vezes, pode ser difícil perceber a diferença ao visitar uma página no navegador da web. O redirecionamento pode acontecer tão rápido que você não percebe nenhum atraso no tempo de carregamento e assume que um redirecionamento do lado do cliente é na verdade um redirecionamento do lado do servidor.

No entanto, ao vasculhar a Web, a diferença é óbvia. Um redirecionamento do lado do servidor, dependendo de como é tratado, pode ser facilmente percorrido pelo Python urllib biblioteca sem qualquer ajuda do Selenium (para obter mais informações sobre como fazer isso, consulte [Capítulo 3](#)) Os redirecionamentos do lado do cliente não serão tratados de forma alguma, a menos que algo esteja realmente executando o JavaScript.

O Selenium é capaz de lidar com esses redirecionamentos de JavaScript da mesma maneira que lida com outras execuções de JavaScript; no entanto, o principal problema com esses redirecionamentos é quando interromper a execução da página - ou seja, como saber quando o redirecionamento de uma página termina. Uma página de demonstração em <http://pythonscraping.com/pages/javascript/redirectDemo2.html> dá um exemplo desse tipo de redirecionamento, com uma pausa de dois segundos.

Podemos detectar esse redirecionamento de uma maneira inteligente, “observando” um elemento no DOM quando a página é carregada inicialmente e, em seguida, chamando esse elemento repetidamente até que o Selenium lance um StaleElementReferenceException ou seja, o elemento não está mais anexado ao DOM da página e o site redirecionou:

```
de selénio importar webdriver
importar Tempo
de selenium.webdriver.remote.webelement importar WebElement
de selenium.common.exceptions importar StaleElementReferenceException

def waitForLoad ( motorista ):
    elem = motorista . find_element_by_tag_name ( "html" )
    contagem = 0
    enquanto Verdadeiro :
        contagem + = 1
        E se contagem > 20 :
            impressão ( " Timing out after 10 seconds and returning" )
            return
        time . sleep ( . 5 )
    try :
        elem == driver . find_element_by_tag_name ( "html" )
    except StaleElementReferenceException :
        return

driver = webdriver . PhantomJS ( executable_path = '<Path to Phantom JS>' )
driver . get ( "http://pythonscraping.com/pages/javascript/redirectDemo1.html" )
waitForLoad ( driver )
print ( driver . page_source )
```

This script checks the page every half second, with a timeout of 10 seconds, although the times used for the checking time and timeout can be easily adjusted up or down as needed.

Processamento de imagem e reconhecimento de texto

Dos carros autônomos do Google a máquinas de venda automática que reconhecem moedas falsas, a visão de máquina é um campo enorme com objetivos e implicações de longo alcance. Neste capítulo, vamos nos concentrar em um aspecto muito pequeno do campo: reconhecimento de texto, especificamente como reconhecer e usar imagens baseadas em texto encontradas online usando uma variedade de bibliotecas Python.

Usar uma imagem no lugar do texto é uma técnica comum quando você não quer que o texto seja encontrado e lido por bots. Isso geralmente é visto em formulários de contato quando um endereço de e-mail é parcial ou totalmente processado como uma imagem. Dependendo de quanto habilmente isso é feito, pode nem ser perceptível para visualizadores humanos, mas os bots têm muita dificuldade em ler essas imagens e a técnica é suficiente para impedir que a maioria dos spammers obtenham seu endereço de e-mail.

Os CAPTCHAs, é claro, tiram vantagem do fato de que os usuários podem ler imagens de segurança, mas a maioria dos bots não. Alguns CAPTCHAs são mais difíceis do que outros, um problema que abordaremos posteriormente neste livro.

Mas os CAPTCHAs não são o único lugar na Web onde os scrapers precisam de ajuda na tradução de imagem para texto. Mesmo nos dias de hoje, muitos documentos são simplesmente digitalizados de cópias impressas e colocados na Web, tornando esses documentos inacessíveis no que diz respeito à Internet, embora estejam “escondidos à vista de todos”. Sem os recursos de imagem para texto, a única maneira de tornar esses documentos acessíveis é um humano digitá-los à mão - e ninguém tem tempo para isso.

A tradução de imagens em texto é chamada *reconhecimento óptico de caracteres*, ou *OCR*. Existem algumas bibliotecas principais que são capazes de realizar OCR e muitas outras bibliotecas que as suportam ou são construídas sobre elas. Este sistema de bibliotecas pode ficar bastante complicado às vezes, então eu recomendo que você leia a próxima seção antes de tentar qualquer um dos exercícios deste capítulo.

Visão geral das bibliotecas

Python é uma linguagem fantástica para processamento e leitura de imagens, aprendizado de máquina baseado em imagens e até mesmo criação de imagens. Embora haja um grande número de bibliotecas que podem ser usadas para processamento de imagens, vamos nos concentrar em duas: Pillow e Tesseract.

Qualquer um pode ser instalado baixando de seus sites e instalando a partir da fonte (<http://pillow.readthedocs.org/installation.html> e <https://pypi.python.org/pypi/pyteseract>) ou usando um instalador Python de terceiros, como pip e usando as palavras-chave “pillow” e “pytesseract”, respectivamente.

Travesseiro

Embora a Pillow possa não ser a biblioteca de processamento de imagem com mais recursos, ela tem todos os recursos de que você provavelmente precisará e alguns, a menos que você esteja fazendo pesquisas ou reescrevendo o Photoshop em Python ou algo parecido. É também um código muito bem documentado e extremamente fácil de usar.

Separado da Python Imaging Library (PIL) para Python 2.x, Pillow adiciona suporte para Python 3.x. Como seu antecessor, o Pillow permite que você importe e manipule facilmente imagens com uma variedade de filtros, máscaras e até mesmo transformações específicas de pixel:

`de PIL importar Imagem, ImageFilter`

```
gatinho = Imagem . abrir ( "gatinho.jpg" )
borrado gatinho = gatinho . filtro ( ImageFilter . GaussianBlur )
borrado gatinho . Salve • ( "gatinho_blurred.jpg" )
borrado gatinho . exposição ()
```

No exemplo anterior, a imagem *gatinho.jpg* irá abrir no seu visualizador de imagens padrão com um desfoco adicionado a ele e também será salvo em seu estado desfocado como *gatinho_blurred.jpg* no mesmo diretório.

Usaremos o Pillow para realizar o pré-processamento de imagens para torná-las mais legíveis por máquina, mas, como mencionado antes, há muitas outras coisas que você pode fazer com a biblioteca além dessas manipulações simples. Para obter mais informações, consulte o

[Documentação de travesseiro](#).

Tesseract

Tesseract é uma biblioteca OCR. Patrocinado pelo Google (uma empresa obviamente bem conhecida por suas tecnologias de OCR e aprendizado de máquina), o Tesseract é amplamente considerado o melhor e mais preciso sistema de OCR de código aberto disponível.

Além de ser preciso, é extremamente flexível. Ele pode ser treinado para reconhecer qualquer número de fontes (desde que essas fontes sejam relativamente consistentes entre si, como veremos em breve) e pode ser expandido para reconhecer qualquer caractere Unicode.

Ao contrário das bibliotecas que usamos até agora neste livro, Tesseract é uma ferramenta de linha de comando escrita em Python, ao invés de uma biblioteca usada com um importar declaração. Após a instalação, deve ser executado com o tesserao comando, de fora do Python.

Instalando Tesseract

Para usuários do Windows, há um conveniente [instalador executável](#). No momento em que este livro foi escrito, a versão atual é 3.02, embora versões mais recentes também devam funcionar.

Os usuários do Linux podem instalar o Tesseract com apt-get:

```
$ sudo apt-get tesseract-ocr
```

Instalar o Tesseract em um Mac é um pouco mais complicado, embora possa ser feito facilmente com muitos instaladores de terceiros, como [Homebrew](#), que foi usado em [Capítulo 5](#) para instalar o MySQL. Por exemplo, você pode instalar o Homebrew e usá-lo para instalar o Tesseract em duas linhas:

```
$ ruby -e "$( curl -fsSL https://raw.githubusercontent.com/Homebrew/install/ | sed '/#-instalar/d' )"
```

O Tesseract também pode ser instalado a partir da fonte, no [página de download do projeto](#).

Para usar alguns recursos do Tesseract, como treinar o software para reconhecer novos personagens posteriormente nesta seção, você também precisará definir uma nova variável de ambiente, \$ TESSDATA_PREFIX, para informá-lo onde os arquivos de dados estão armazenados.

Você pode fazer isso na maioria dos sistemas Linux e no Mac OS X usando:

```
$ export TESSDATA_PREFIX=/usr/local/share/
```

Observe que /usr/local/share/ é o local de dados padrão do Tesseract, embora você deva verificar se esse é o caso para sua própria instalação.

Da mesma forma, no Windows, você pode usar o seguinte para usar a variável de ambiente:

```
# setx TESSDATA_PREFIX C:\Arquivos de programas\Tesseract OCR\
```

NumPy

Embora o NumPy não seja necessário para o OCR direto, você precisará dele se quiser treinar o Tesseract para reconhecer conjuntos de caracteres ou fontes adicionais introduzidos posteriormente neste capítulo. NumPy é uma biblioteca muito poderosa usada para álgebra linear e outras aplicações matemáticas de grande escala. NumPy funciona bem com Tesseract por causa de sua capacidade de representar matematicamente e manipular imagens como grandes matrizes de pixels.

Como sempre, o NumPy pode ser instalado usando qualquer instalador Python de terceiros, como pip:

```
$ pip instalar numpy
```

Processando texto bem formatado

Com alguma sorte, a maior parte do texto que você precisará processar estará relativamente limpo e bem formatado. Um texto bem formatado geralmente atende a vários requisitos, embora a linha entre o que é “confuso” e o que é “bem formatado” possa ser subjetiva.

Em geral, texto bem formatado:

- Está escrito em uma fonte padrão (excluindo fontes manuscritas, fontes cursivas ou fontes excessivamente “decorativas”)
- Se copiado ou fotografado tem linhas extremamente nítidas, sem artefatos de cópia ou manchas escuras
- Está bem alinhado, sem letras inclinadas
- Não sai da imagem, nem há texto cortado ou margens nas bordas da imagem

Algumas dessas coisas podem ser corrigidas no pré-processamento. Por exemplo, as imagens podem ser convertidas para tons de cinza, o brilho e o contraste podem ser ajustados e a imagem pode ser cortada e girada conforme necessário. No entanto, existem algumas limitações fundamentais que podem exigir um treinamento mais extenso. Vejo [“Leitura de CAPTCHAs e Tesseract de Treinamento” na página 169](#).

Figura 11-1 um exemplo ideal de texto bem formatado.

This is some text, written in Arial, that will be read by Tesseract. Here are some symbols: !@#\$%^&*()

Figura 11-1. Texto de amostra salvo como um arquivo .tiff, para ser lido pelo Tesseract

Você pode executar o Tesseract na linha de comando para ler este arquivo e gravar os resultados em um arquivo de texto:

```
$ tesseract text.tif textoutput | cat textoutput.txt
```

A saída é uma linha de informações sobre a biblioteca Tesseract para indicar que ela está em execução, seguida pelo conteúdo da biblioteca recém-criada *textoutput.txt*:

Tesseract Open Source OCR Engine v3.02.02 com Leptonica Este é um texto, escrito
em Arial, que será lido por Tesseract. Aqui estão alguns símbolos : ! @ # \$% "& '()

Você pode ver que os resultados são em sua maioria precisos, embora os símbolos “^” e “*” tenham sido interpretados como aspas duplas e simples, respectivamente. No geral, porém, isso permite que você leia o texto de maneira bastante confortável.

Depois de desfocar o texto da imagem, criar alguns artefatos de compressão JPG e adicionar um leve gradiente de fundo, os resultados ficam muito piores (veja **Figura 11-2**)

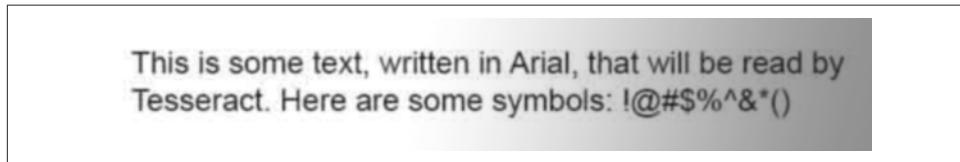


Figura 11-2. Infelizmente, muitos dos documentos que você encontrará na Internet serão mais parecidos com este do que com o exemplo anterior

O Tesseract não é capaz de lidar com esta imagem quase tão bem principalmente devido ao gradiente de fundo e produz a seguinte saída:

Este é algum texto, escrito em Arial, que "Tesseract. Aqui
estão alguns símbolos: _

Observe que o texto é cortado assim que o gradiente de fundo torna o texto mais difícil de distinguir e que o último caractere de cada linha está errado, pois Tesseract tenta inutilmente dar sentido a ele. Além disso, os artefatos JPG e o desfoque tornam difícil para o Tesseract distinguir entre um “i” minúsculo e um “l” maiúsculo e o número “1”.

É aqui que usar um script Python para limpar suas imagens pela primeira vez é útil. Usando a biblioteca Pillow, podemos criar um filtro de limite para eliminar o cinza do fundo, trazer o texto e tornar a imagem mais clara para Tesseract ler:

```
de PIL importar Imagem
importar subprocesso

def clearFile( caminho de arquivo , newFilePath ):
    imagem = Imagem . abrir ( caminho de arquivo )

    # Defina um valor limite para a imagem e salve
    imagem = imagem . ponto ( lambda x : 0 E se x < 143 outro 255 )
    imagem . Salve • ( newFilePath )

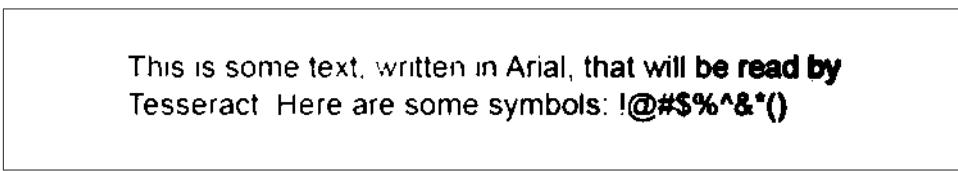
    # chamar tesseract para fazer OCR na imagem recém-criada
    subprocesso . ligar ([ "tesseract" , newFilePath , "resultado" ])
```

```
# Abra e leia o arquivo de dados resultante
arquivo de saída = abrir ( "output.txt" , 'r' )
impressão ( arquivo de saída . ler () )
arquivo de saída . Fechar ()

cleanFile ( "text_2.png" , "text_2_clean.png" )
```

A imagem resultante, criada automaticamente como *text_2_clean.png*, é mostrado em

Figura 11-3 :



This is some text, written in Arial, that will be read by
Tesseract. Here are some symbols: !@#\$%^&*()

Figura 11-3. Esta imagem foi criada passando a versão anterior "confusa" da imagem por um filtro de limite

Além de alguma pontuação pouco legível ou ausente, o texto é legível, pelo menos para nós. Tesseract oferece sua melhor chance:

Este é um texto 'escrito' em Anal, que será lido por Tesseract. Aqui estão alguns símbolos :! @ # \$ % "& '()

Os pontos e vírgulas, sendo extremamente pequenos, são as primeiras vítimas dessa disputa de imagens e quase desaparecem, tanto de nossa vista quanto de Tesserato. Há também a infeliz interpretação errônea de "Arial" como "Anal", o resultado de Tesseract interpretar o "r" e o "i" como o único caractere "n".

Ainda assim, é uma melhoria em relação à versão anterior, na qual quase metade do texto foi cortado.

A maior fraqueza de Tesseract parece ser fundos com brilho variável. Os algoritmos do Tesseract tentam ajustar o contraste da imagem automaticamente antes de ler o texto, mas você provavelmente pode obter melhores resultados fazendo isso sozinho com uma ferramenta como a biblioteca Pillow.

As imagens que você definitivamente deve corrigir antes de enviar para o Tesseract são aquelas que são inclinadas, têm grandes áreas sem texto ou outros problemas.

Raspagem de texto de imagens em sites

Usar o Tesseract para ler texto de uma imagem em seu disco rígido pode não parecer muito empolgante, mas pode ser uma ferramenta muito poderosa quando usado com um raspador de web. As imagens podem ofuscar inadvertidamente o texto em sites (como com a cópia JPG de um menu em um site de restaurante local), mas também podem ocultar o texto propositalmente, como mostrarei no próximo exemplo.

Embora Amazon's *robots.txt* permite copiar as páginas de produtos do site, visualizações de livros normalmente não são detectadas por bots de passagem. Isso porque as visualizações do livro são carregadas por meio de scripts Ajax acionados pelo usuário, e as imagens são cuidadosamente escondidas sob camadas de divs; na verdade, para o visitante médio do site, eles provavelmente parecem mais com uma apresentação em flash do que com arquivos de imagem. Claro, mesmo se pudéssemos chegar às imagens, há a questão não tão pequena de lê-las como texto.

O script a seguir realiza exatamente esta façanha: ele navega para a edição em letras grandes¹ de Tolstoi *Guerra e Paz*, abre o leitor, coleta URLs de imagens e, em seguida, baixa, lê e imprime sistematicamente o texto de cada uma. Como este é um código relativamente complexo que se baseia em vários conceitos dos capítulos anteriores, adicionei comentários para tornar um pouco mais fácil entender o que está acontecendo:

```
importar Tempo
de urllib.request importar urlretrieve
importar subprocesso
de selénio importar webdriver

# Crie um novo driver Selenium
motorista = webdriver . PhantomJS ( executable_path = '<Caminho para Phantom JS>' )
# Às vezes, descobri que o PhantomJS tem problemas para encontrar elementos neste
# página que o Firefox não possui. Se for esse o caso quando você executa,
# tente usar um navegador Firefox com Selenium removendo o comentário desta linha:
# driver = webdriver.Firefox ()

motorista . pegue (
    "http://www.amazon.com/War-Peace-Leo-Nikolayevich-Tolstoy/dp/1427030200" )
Tempo . dormir ( 2 )

# Clique no botão de visualização do livro
motorista . find_element_by_id ( "sitbLogoImg" ) . clique ()
imageList = conjunto ( )

# Espere a página carregar
Tempo . dormir ( 5 )
# Enquanto a seta para a direita está disponível para clicar, vire as páginas
enquanto " ponteiro " dentro motorista . find_element_by_id ( "sitbReaderRightPageTurner" )
    . get_attribute ( "estilo" ):
        motorista . find_element_by_id ( "sitbReaderRightPageTurner" ) . clique ()
Tempo . dormir ( 2 )
# Obtenha todas as novas páginas que foram carregadas (várias páginas podem ser carregadas de uma vez,
# mas as duplicatas não serão adicionadas a um conjunto)
Páginas = motorista . find_elements_by_xpath ( " // div [@ class = 'pageImage' ] / div / img" )
para página dentro Páginas :
```

¹ Quando se trata de processamento de texto no qual não foi treinado, o Tesseract se sai muito melhor com edições de livros em formato grande, especialmente se as imagens forem pequenas. Na próxima seção, discutiremos como treinar o Tesseract em diferentes fontes, o que pode ajudá-lo a ler tamanhos de fonte muito menores, incluindo visualizações para edições de livros com impressão menor!

```

        imagem = página . get_attribute ( "src" )
        imageList . adicionar ( imagem )

motorista . Sair ()

# Comece a processar as imagens para as quais coletamos URLs com o Tesseract

para imagem dentro classificado ( imageList ):
    urlretrieve ( imagem , "page.jpg" )
    p = subprocesso . Popen ([ "tesseract" , "page.jpg" , "página" ],
                           saída padrão = subprocesso . TUBO , stderr = subprocesso . TUBO )
    p . esperar ()
    f = abrir ( "page.txt" , "r" )
    impressão ( f . ler ())

```

Como já experimentamos com o leitor do Tesseract antes, ele imprime muitas passagens longas do livro perfeitamente, como visto na visualização da página 6:

6

"Um conselho amigável, mon cher. Saia o mais rápido que puder, é tudo o que tenho a lhe dizer. Feliz aquele que tem ouvidos para ouvir. Adeus, meu caro amigo. Oh, por falar nisso! " ele gritou pela porta atrás de Pierre, "é verdade que a condessa caiu nas garras dos santos padres da Sociedade de Jesus?"

Pierre não respondeu e saiu do quarto de Rostopchin mais taciturno e zangado do que jamais havia se mostrado.

No entanto, quando o texto aparece em um fundo colorido na capa e contracapa do livro, torna-se incompreensível:

WEI 'nrrd Peace
Len Nlkelayevldu lol fl uy

Readmg shmdd be ax
wlnvame asnoxxble Wen fl er
um mm m nosso cram: Lihvary

- Leo Tmsloy era um russo rwovelwst l e moval ph fl
mopher méd lur

A ms Ideas 01 reswslace não violento m 5 Nós variamos 0, "e"

Claro, você pode usar a biblioteca Pillow para limpar imagens seletivamente, mas isso pode ser extremamente trabalhoso para um processo que foi projetado para ser o mais livre de humanos possível.

A próxima seção discute outra abordagem para resolver o problema do texto mutilado, especialmente se você não se importar em investir um pouco de tempo adiantado no treinamento do Tesseract. Ao fornecer ao Tesseract uma grande coleção de imagens de texto com valores conhecidos, o Tesseract pode ser “ensinado” a reconhecer a mesma fonte no futuro com muito maior precisão e exatidão, mesmo apesar dos ocasionais problemas de fundo e posicionamento no texto.

Leitura de CAPTCHAs e Tesserato de Treinamento

Embora a palavra “CAPTCHA” seja familiar para a maioria, muito menos pessoas sabem o que ela significa: Teste de Turing público automatizado por computador para distinguir computadores de humanos. Seu acrônimo pesado sugere seu papel bastante difícil de obstruir interfaces da web perfeitamente utilizáveis, já que tanto humanos quanto robôs não humanos freqüentemente lutam para resolver os testes CAPTCHA.

O teste de Turing foi descrito pela primeira vez por Alan Turing em seu artigo de 1950, “Computing Machinery and Intelligence”. No artigo, ele descreveu uma configuração na qual um ser humano pode se comunicar tanto com humanos quanto com programas de inteligência artificial por meio de um terminal de computador. Se o humano fosse incapaz de distinguir os humanos dos programas de IA durante uma conversa casual, os programas de IA seriam considerados como tendo passado no teste de Turing, e a inteligência artificial, raciocinou Turing, estaria genuinamente “pensando” para todos os efeitos e propósitos.

É irônico que nos últimos 60 anos tenhamos passado de usar esses testes para testar máquinas para usá-los para testar a nós mesmos, com resultados mistos. O reCAPTCHA do Google, notoriamente difícil, atualmente o mais popular entre os sites preocupados com a segurança, bloqueia até 25% dos usuários humanos legítimos de acessar um site.²

A maioria dos outros CAPTCHAs são um pouco mais fáceis. Drupal, um popular sistema de gerenciamento de conteúdo baseado em PHP, por exemplo, tem um popular [Módulo CAPTCHA](#), que pode gerar imagens CAPTCHA de dificuldade variada. A imagem padrão parece [Figura 11-4](#).

² Ver <http://bit.ly/1HGTbGf>.

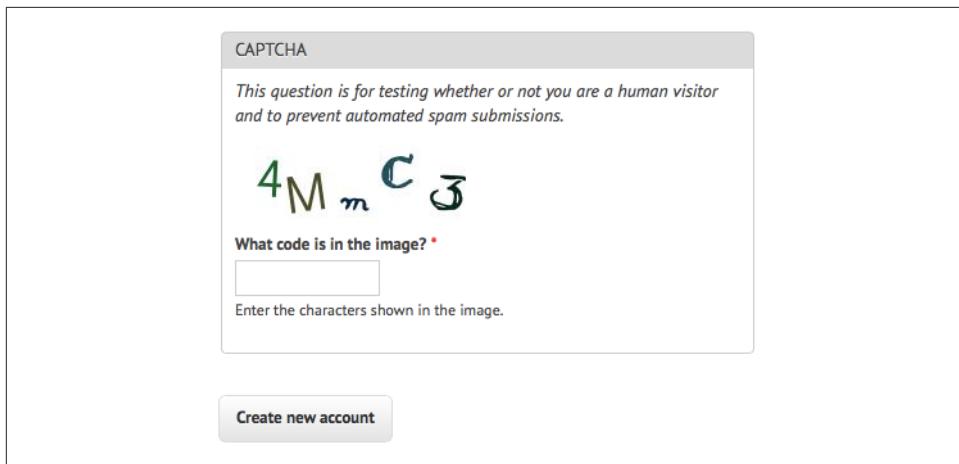


Figura 11-4. Um exemplo de texto padrão CAPTCHA para o projeto CAPTCHA do Drupal

O que torna este CAPTCHA tão fácil de ler para humanos e máquinas, em comparação com outros CAPTCHAs?

- Os personagens não se sobrepõem, nem se cruzam no espaço um do outro horizontalmente. Ou seja, é possível desenhar um retângulo bem organizado ao redor de cada caractere sem sobrepor nenhum outro caractere.
- Não há imagens de fundo, linhas ou outro lixo perturbador que possa confundir um programa de OCR.
- Não é óbvio nesta imagem, mas existem algumas variações na fonte que o CAPTCHA usa. Ele alterna entre uma fonte sans-serif limpa (como visto nos caracteres "4" e "M") e uma fonte de estilo manuscrito, (como visto nos caracteres "m", "C" e "3")
- Existe um alto contraste entre o fundo branco e os caracteres escuros.

Este CAPTCHA apresenta algumas curvas, porém, que tornam a leitura difícil para os programas de OCR:

- São usados letras e números, aumentando o número de caracteres potenciais.
- A inclinação aleatória das letras pode confundir o software OCR, mas permanece muito fácil de ler para humanos.
- A fonte de escrita à mão relativamente estranha apresenta desafios específicos, com linhas extras no "C" e "3" e um "m" minúsculo incomumente pequeno, exigindo treinamento extra para os computadores se familiarizarem.

Quando executamos o Tesseract sobre esta imagem usando o comando:

```
$ tesseract saída captchaExample.png
```

nós entendemos isso *output.txt* Arquivo:

4N \ ... C <3

Ele acertou o 4, o C e o 3, mas claramente não será capaz de preencher um campo protegido por CAPTCHA tão cedo.

Tesseract de treinamento

Para treinar o Tesseract a reconhecer a escrita, seja uma fonte obscura e difícil de ler ou um CAPTCHA, você precisa dar a ele vários exemplos de cada caractere.

Esta é a parte em que você pode querer colocar um bom podcast ou filme na fila, porque vai ser um trabalho bastante chato por algumas horas. A primeira etapa é baixar vários exemplos de seu CAPTCHA em um único diretório. O número de exemplos que você compilar dependerá da complexidade do CAPTCHA; Usei 100 arquivos de amostra (um total de 500 caracteres, ou cerca de 8 exemplos por símbolo, em média) para meu treinamento CAPTCHA e isso parece funcionar muito bem.

Dica: Eu recomendo nomear a imagem após a solução CAPTCHA que ela representa (ou seja, *4MmC3.jpg*). Descobri que isso ajuda a fazer uma verificação de erros rápida em um grande número de arquivos de uma vez - você pode ver todos os arquivos como miniaturas e comparar a imagem com o nome da imagem facilmente. Isso também ajuda muito na verificação de erros nas etapas subsequentes.

O segundo passo é dizer a Tesseract exatamente o que cada personagem é e onde está na imagem. Isso envolve a criação de arquivos de caixa, um para cada imagem CAPTCHA. Um arquivo de caixa tem a seguinte aparência:

```
4 15 26 33 55 0  
M 38 13 67 45 0  
m 79 15 101 26 0 C 111  
33 136 60 0 3 147 17 176  
45 0
```

O primeiro símbolo é o caractere representado, os próximos quatro números representam coordenadas para uma caixa retangular delineando a imagem, e o último número é um "número de página" usado para treinamento com documentos de várias páginas (0 para nós).

Obviamente, esses arquivos box não são divertidos de criar manualmente, mas há uma variedade de ferramentas para ajudá-lo. Eu gosto da ferramenta online [Chopper Tesseract OCR](#) porque não requer instalação ou bibliotecas adicionais, roda em qualquer máquina que tenha um navegador e é relativamente fácil de usar: carregue a imagem, clique no botão "adicionar" na parte inferior se você precisar de caixas adicionais, ajuste o tamanho das caixas, se necessário, e copie e cole o texto do novo arquivo de caixa em um novo arquivo.

Os arquivos de caixa devem ser salvos em texto simples, com o. *caixa* extensão de arquivo. Tal como acontece com os arquivos de imagem, é útil nomear os arquivos de caixa pelas soluções CAPTCHA que eles representam (por exemplo, *4MmC3.box*). Novamente, isso torna mais fácil verificar novamente o conteúdo do. *caixa* Arquivo

texto em relação ao nome do arquivo e, em seguida, novamente em relação ao arquivo de imagem com o qual ele é pareado, se você classificar todos os arquivos em seu diretório de dados pelo nome do arquivo.

Novamente, você precisará criar cerca de 100 desses arquivos para garantir que tenha dados suficientes. Além disso, o Tesseract ocasionalmente descarta arquivos como ilegíveis, então você pode querer algum espaço de buffer além disso. Se você achar que seus resultados de OCR não são tão bons quanto você gostaria, ou se Tesseract está tropeçando em certos personagens, é uma boa etapa de depuração para criar dados de treinamento adicionais e tentar novamente.

Depois de criar uma pasta de dados cheia de *caixa* arquivos e arquivos de imagem, copie esses dados em uma pasta de backup antes de fazer qualquer manipulação adicional. Embora a execução de scripts de treinamento sobre os dados provavelmente não exclua nada, é melhor prevenir do que remediar quando horas de trabalho valem a pena. *caixa* a criação do arquivo está envolvida. Além disso, é bom poder descartar um diretório bagunçado cheio de dados compilados e tentar novamente.

Existem meia dúzia de etapas para realizar todas as análises de dados e criar os arquivos de treinamento necessários para o Tesseract. Existem ferramentas que fazem isso para você com a imagem de origem e *caixa* arquivos, mas nenhum no momento para o Tesseract 3.02, infelizmente.

eu escrevi [uma solução em Python](#) que opera sobre um arquivo contendo arquivos de imagem e caixa e cria todos os arquivos de treinamento necessários automaticamente.

As principais configurações e etapas que este programa executa podem ser vistas em seu a Principal e executar tudo métodos:

```
def a Principal ( auto ):  
    languageName = "eng"  
    fontName = "captchaFont"  
    diretório = "<caminho para imagens>"  
  
def runAll ( auto ):  
    auto . createFontFile ()  
    auto . cleanImages ()  
    auto . renameFiles ()  
    auto . extractUnicode ()  
    auto . runShapeClustering ()  
    auto . runMFTTraining ()  
    auto . runCnTraining ()  
    auto . createTessData ()
```

As únicas três variáveis que você precisa definir aqui são bastante diretas:

languageName

O código de idioma de três letras que o Tesseract usa para entender para qual idioma ele está olhando. Na maioria dos casos, você provavelmente desejará usar "eng" para inglês.

fontName

O nome da fonte escolhida. Pode ser qualquer coisa, mas deve ser uma única palavra sem espaços.

O diretório que contém todos os seus arquivos de imagem e caixa

Eu recomendo que você torne este um caminho absoluto, mas se você usar um caminho relativo, ele precisará ser relativo ao local de onde você está executando o código Python. Se for absoluto, você pode executar o código de qualquer lugar em sua máquina.

Vamos dar uma olhada nas funções individuais usadas.

createFontFile cria um arquivo necessário, *font_properties*, que permite ao Tesseract saber sobre a nova fonte que estamos criando:

```
captchaFont 0 0 0 0
```

Este arquivo consiste no nome da fonte, seguido por 1s e 0s indicando se itálico, negrito ou outras versões da fonte devem ser consideradas (fontes de treinamento com essas propriedades é um exercício interessante, mas infelizmente fora do escopo deste livro) .

cleanImages cria versões de alto contraste de todos os arquivos de imagem encontrados, converte-os em tons de cinza e executa outras operações que tornam os arquivos de imagem mais fáceis de ler por programas de OCR. Se você estiver lidando com imagens CAPTCHA com lixo visual que podem ser fáceis de filtrar no pós-processamento, aqui seria o lugar para adicionar esse processamento adicional.

renameFiles renomeia todos os seus. *caixa* arquivos e seus arquivos de imagem correspondentes com os nomes exigidos pelo Tesseract (os números dos arquivos aqui são dígitos sequenciais para manter vários arquivos separados):

- <*languageName*>. <*fontName*> .exp <*fileNumber*> .box
- <*languageName*>. <*fontName*> .exp <*fileNumber*> .tiff

extractUnicode olha para todos os criados. *caixa* arquivos e determina o conjunto total de caracteres disponíveis para serem treinados. O arquivo Unicode resultante dirá quantos caracteres diferentes você encontrou e pode ser uma boa maneira de ver rapidamente se está faltando alguma coisa.

As próximas três funções, runShapeClustering, runMfTraining, e runCtTrain ing, crie os arquivos modelável, pftable, e normproto, respectivamente. Todos eles fornecem informações sobre a geometria e a forma de cada personagem, bem como fornecem informações estatísticas que Tesseract usa para calcular a probabilidade de que um determinado personagem seja um tipo ou outro.

Por fim, o Tesseract renomeia cada uma das pastas de dados compiladas para serem acrescentadas ao nome do idioma necessário (por exemplo, *modelável* é renomeado para *eng.shapetable*) e compila todos esses arquivos no arquivo de dados de treinamento final *eng.traineddata*.

A única etapa que você precisa realizar manualmente é mover o *eng.traineddata* arquivo para o seu *tessdata* pasta raiz usando os seguintes comandos no Linux e Mac:

```
$ cp /path/to/data/eng.traineddata $TESSDATA_PREFIX/tessdata
```

Seguindo essas etapas, você não deverá ter problemas para resolver CAPTCHAs do tipo para o qual o Tesseract foi treinado. Agora, quando peço a Tesseract para ler a imagem do exemplo, obtenho a resposta correta:

```
$ saída tesseract captchaExample.png ; cat output.txt 4MmC3
```

Sucesso! Uma melhoria significativa em relação à interpretação anterior da imagem como "4N \,,, C <3"

Esta é apenas uma visão geral rápida de todo o poder do treinamento de fontes e recursos de reconhecimento do Tesseract. Se você estiver interessado em treinar extensivamente o Tesseract, talvez começar sua própria biblioteca pessoal de arquivos de treinamento CAPTCHA ou compartilhar novos recursos de reconhecimento de fonte com o mundo, recomendo verificar o [documentação](#).

Recuperando CAPTCHAs e enviando soluções

Muitos sistemas populares de gerenciamento de conteúdo são frequentemente envenenados por spam com registros por bots que são pré-programados com a localização conhecida dessas páginas de registro do usuário. Em <http://pythonscraping.com>, por exemplo, mesmo um CAPTCHA (reconhecidamente fraco) faz pouco para diminuir o fluxo de registros.

Então, como esses bots fazem isso? Resolvemos CAPTCHAs com sucesso em imagens em nosso disco rígido, mas como fazemos um bot totalmente funcional? Esta seção reúne muitas técnicas abordadas nos capítulos anteriores. Se ainda não o fez, você deve pelo menos folhear os capítulos sobre envio de formulários e download de arquivos.

A maioria dos CAPTCHAs baseados em imagens tem várias propriedades:

- Eles são imagens geradas dinamicamente, criadas por um programa do lado do servidor. Eles podem ter fontes de imagem que não se parecem com imagens tradicionais, como
- A solução para a imagem é armazenada em um banco de dados do lado do servidor.
- Muitos CAPTCHAs expiram se você demorar muito para resolvê-los. Isso geralmente não é um problema para bots, mas enfileirar soluções CAPTCHA para uso posterior, ou outra prática

Ces que podem atrasar o tempo entre a solicitação do CAPTCHA e o envio da solução podem não ser bem-sucedidos.

A abordagem geral para isso é fazer o download do arquivo de imagem CAPTCHA em seu disco rígido, limpá-lo, usar o Tesseract para analisar a imagem e retornar a solução no parâmetro de formulário apropriado.

Eu criei uma página em <http://pythonscraping.com/humans-only> com um formulário de comentário protegido por CAPTCHA com o propósito de escrever um bot para derrotar. O bot se parece com o seguinte:

```
de urllib.request importar urlretrieve
de urllib.request importar urlopen
de bs4 importar BeautifulSoup
importar subprocess
importar solicitações de
de PIL importar Imagem
de PIL importar ImageOps

def cleanImage ( imagePath ):
    imagem = Imagem . abrir ( imagePath )
    imagem = imagem . ponto ( lambda x : 0 E se x < 143 outro 255 )
    borderImage = ImageOps . expandir ( imagem , fronteira = 20 , preencher = 'branco' )
    borderImage . Salve • ( imagePath )

html = urlopen ( "http://www.pythonscraping.com/humans-only" )
bsObj = BeautifulSoup ( html )
# Reúna valores de formulário pré-preenchidos
imageLocation = bsObj . encontrar ( "img" , { "título" : "Image CAPTCHA" })[ "src" ]
formBuildId = bsObj . find ( "input" , { "name" : "form_build_id" })[ "value" ]
captchaSid = bsObj . find ( "input" , { "name" : "captcha_sid" })[ "value" ]
captchaToken = bsObj . find ( "input" , { "name" : "captcha_token" })[ "value" ]

captchaUrl = "http://pythonscraping.com" + imageLocation
urllib . urlopen ( captchaUrl , "captcha.jpg" )
cleanImage ( "captcha.jpg" )
p = subprocess . Popen ( [ "tesseract" , "captcha.jpg" , "captcha" ] , stdout =
    subprocess . PIPE , stderr = subprocess . PIPE )
p . wait ()
f = open ( "captcha.txt" , "r" )

# Clean any whitespace characters
captchaResponse = f . read () . replace ( " " , "" ) . replace ( "\n" , "" )
print ( "Capcha solution attempt: " + captchaResponse )

if len ( captchaResponse ) == 5 :
    params = { "captcha_token" : captchaToken , "captcha_sid" : captchaSid ,
        "form_id" : "comment_node_page_form" , "form_build_id" : formBuildId ,
        "captcha_response" : captchaResponse , "name" : "Ryan Mitchell" ,
        "subject" : "I come to seek the Grail" ,
        "comment_body[und][0][value]" :
            "...and I am definitely not a bot" }
```

```

r = requests . post ( "http://www.pythonscraping.com/comment/reply/10" ,
                     data = params )
responseObj = BeautifulSoup ( r . text )
if responseObj . find ( "div" , { "class" : "messages" }) is not None :
    print ( responseObj . find ( "div" , { "class" : "messages" }) . get_text () )
else :
    print ( " There was a problem reading the CAPTCHA correctly!" )

```

Note there are two conditions on which this script fails: if Tesseract did not extract exactly five characters from the image (because we know that all valid solutions to this CAPTCHA must have five characters), or if it submits the form but the CAPTCHA was solved incorrectly. The first case happens approximately 50% of the time, at which point it does not bother submitting the form and fails with an error message. The second case happens approximately 20% of the time, for a total accuracy rate of about 30% (or about 80% accuracy for each character encountered, over 5 characters).

Although this may seem low, keep in mind that there is usually no limit placed on the number of times users are allowed to make CAPTCHA attempts, and that most of these incorrect attempts can be aborted without needing to actually send the form. When a form is actually sent, the CAPTCHA is accurate most of the time. If that doesn't convince you, also keep in mind that simple guessing would give you an accuracy rate of .0000001%. Running a program three or four times rather than guessing 900 million times is quite the time saver!

Avoiding Scraping Traps

There are few things more frustrating than scraping a site, viewing the output, and not seeing the data that's so clearly visible in your browser. Or submitting a form that should be perfectly fine but gets denied by the web server. Or getting your IP address blocked by a site for unknown reasons.

These are some of the most difficult bugs to solve, not only because they can be so unexpected (a script that works just fine on one site might not work at all on another, seemingly identical, site), but because they purposefully don't have any tell tale error messages or stack traces to use. You've been identified as a bot, rejected, and you don't know why.

In this book, I've written about a lot of ways to do tricky things on websites (submitting forms, extracting and cleaning difficult data, executing JavaScript, etc.). This chapter is a bit of a catchall in that the techniques stem from a wide variety of subjects (HTTP headers, CSS, and HTML forms, to name a few). However, they all have something in common: they are meant to overcome an obstacle put in place for the sole purpose of preventing automated web scraping of a site.

Regardless of how immediately useful this information is to you at the moment, I highly recommend you at least skim this chapter. You never know when it might help you solve a very difficult bug or prevent a problem altogether.

A Note on Ethics

In the first few chapters of this book, I discussed the legal gray area that web scraping inhabits, as well as some of the ethical guidelines to scrape by. To be honest, this chapter is, ethically, perhaps the most difficult one for me to write. My websites have been plagued by bots, spammers, web scrapers, and all manner of unwanted virtual guests, as perhaps yours have been. So why teach people how to build better bots?

There are a few reasons why I believe this chapter is important to include:

- There are perfectly ethical and legally sound reasons to scrape some websites that do not want to be scraped. In a previous job I had as a web scraper, I performed an automated collection of information from websites that were publishing clients' names, addresses, telephone numbers, and other personal information to the Internet without their consent. I used the scraped information to make formal requests to the websites to remove this information. In order to avoid competition, these sites guarded this information from scrapers vigilantly. However, my work to ensure the anonymity of my company's clients (some of whom had stalkers, were the victims of domestic violence, or had other very good reasons to want to keep a low profile) made a compelling case for web scraping, and I was grateful that I had the skills necessary to do the job.
- Although it is almost impossible to build a "scraper proof" site (or at least one that can still be easily accessed by legitimate users), I hope that the information in this chapter will help those wanting to defend their websites against malicious attacks. Throughout, I will point out some of the weaknesses in each web scraping technique, which you can use to defend your own site. Keep in mind that most bots on the Web today are merely doing a broad scan for information and vulnerabilities, and employing even a couple of simple techniques described in this chapter will likely thwart 99% of them. However, they are getting more sophisticated every month, and it's best to be prepared.
- Like most programmers, I simply don't believe that withholding any sort of educational information is a net positive thing to do.

While you're reading this chapter, keep in mind that many of these scripts and described techniques should not be run against every site you can find. Not only is it not a nice thing to do, but you could wind up receiving a cease-and-desist letter or worse. But I'm not going to pound you over the head with this every time we discuss a new technique. So, for the rest of this book—as the philosopher Gump once said—“That's all I have to say about that.”

Looking Like a Human

The fundamental challenge for sites that do not want to be scraped is figuring out how to tell bots from humans. Although many of the techniques sites use (such as CAPTCHAs) can be difficult to fool, there are a few fairly easy things you can do to make your bot look more human.

Adjust Your Headers

In [Chapter 9](#), we used the `requests` module to handle forms on a website. The `requests` module is also excellent for setting headers. HTTP headers are a list of attributes, or preferences, sent by you every time you make a request to a web server. HTTP defines dozens of obscure header types, most of which are not commonly used. The following seven fields, however, are consistently used by most major browsers when initiating any connection (shown with example data from my own browser):

Host	<code>https://www.google.com/</code>
Connection	<code>keep-alive</code>
Accept	<code>text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8</code>
User-Agent	<code>Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36</code>
Referrer	<code>https://www.google.com/</code>
Accept-Encoding	<code>gzip, deflate, sdch</code>
Accept-Language	<code>en-US,en;q=0.8</code>

And here are the headers that a typical Python scraper using the default `urllib` library might send:

Accept-Encoding	<code>identity</code>
User-Agent	<code>Python-urllib/3.4</code>

If you're a website administrator trying to block scrapers, which one are you more likely to let through?

Installing Requests

We installed the `Requests` module in [Chapter 9](#), but if you haven't done so, you can find download links and instructions on the [module's website](#) or use any third-party Python module installer.

Fortunately, headers can be completely customized using the `requests` module. The website <https://www.whatismybrowser.com> is great for testing browser properties viewable by servers. We'll scrape this website to verify our cookie settings with the following script:

```

import requests
from bs4 import BeautifulSoup

session = requests.Session()
headers = { "User-Agent" : "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)
                           AppleWebKit/537.36 (KHTML, like Gecko) Chrome ",
            "Accept" : "text/html,application/xhtml+xml,application/xml;
                        q=0.9, image/webp,*/*;q=0.8" }
url = "https://www.whatismybrowser.com/
       developers/what-http-headers-is-my-browser-sending"
req = session.get(url, headers=headers)

bsObj = BeautifulSoup(req.text)
print(bsObj.find("table", {"class": "table-striped"}).get_text())

```

The output should show that the headers are now the same ones set in the headers dictionary object in the code.

Although it is possible for websites to check for “humanness” based on any of the properties in HTTP headers, I’ve found that typically the only setting that really matters is the User-Agent. It’s a good idea to keep this one set to something more inconspicuous than Python-urllib/3.4, regardless of what project you are working on. In addition, if you ever encounter an extremely suspicious website, populating one of the commonly used but rarely checked headers such as Accept-Language might be the key to convincing it you’re a human.

Headers Change the Way You See the World

Let’s say you want to write a machine learning language translator for a research project, but lack large amounts of translated text to test with. Many large sites present different translations of the same content, based on the indicated language preferences in your headers. Simply changing Accept-Language:en-US to Accept-Language:fr in your headers might get you a “Bonjour” from websites with the scale and budget to handle translation (large international companies are usually a good bet).

Headers also can prompt websites to change the format of the content they are presenting. For instance, mobile devices browsing the Web often see a very pared-down version of sites, lacking banner ads, Flash, and other distractions. If you try changing your User-Agent to something like the following, you might find that sites get a little easier to scrape!

User-Agent:Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like Mac OS X
AppleWebKit/537.51.2 (KHTML, like Gecko) Version/7.0 Mobile/11D257 Safari/9537.53

Handling Cookies

Handling cookies correctly can alleviate many of scraping problems, although cookies can also be a double-edged sword. Websites that track your progression through a site using cookies might attempt to cut off scrapers that display abnormal behavior, such as completing forms too quickly, or visiting too many pages. Although these behaviors can be disguised by closing and reopening connections to the site, or even changing your IP address (see [Chapter 14](#) for more information on how to do that), if your cookie gives your identity away, your efforts of disguise might be futile.

Cookies can also be very necessary to scrape a site. As shown in [Chapter 9](#), staying logged in on a site requires that you be able to hold and present a cookie from page to page. Some websites don't even require that you actually log in and get a new version of a cookie every time—merely holding an old copy of a "logged in" cookie and visiting the site is enough.

If you are scraping a single targeted website or a small number of targeted sites, I recommend examining the cookies generated by those sites and considering which ones you might want your scraper to handle. There are a number of browser plug-ins that can show you how cookies are being set as you visit and move around a site. [Edit-ThisCookie](#), a Chrome extension, is one of my favorites.

Check out the code samples in "[Handling Logins and Cookies](#)" on page 142 in Chapter 9 for more information about handling cookies using the requests module. Of course, because it is unable to execute JavaScript, the requests module will be unable to handle many of the cookies produced by modern tracking software, such as Google Analytics, which are set only after the execution of client-side scripts (or, sometimes, based on page events, such as button clicks, that happen while browsing the page). In order to handle these, you need to use the Selenium and PhantomJS packages (we covered their installation and basic usage in [Chapter 10](#)).

You can view cookies by visiting any site (<http://pythonscraping.com>, in this example) and calling `get_cookies()` on the webdriver:

```
from selenium import webdriver
driver = webdriver.PhantomJS(executable_path='<Path to PhantomJS>')
driver.get("http://pythonscraping.com")
driver.implicitly_wait(1)
print(driver.get_cookies())
```

This provides the fairly typical array of Google Analytics cookies:

```
[{"value": "1", "httponly": False, "name": "__gat", "path": "/", "expiry": 1422806785, "expires": "Sun, 01 Feb 2015 16:06:25 GMT", "secure": False, "domain": ".pythonscraping.com"}, {"value": "GA1.2.161952506 2.1422806186", "httponly": False, "name": "__ga", "path": "/", "expiry": 1485878185, "expires": "Tue, 31 Jan 2017 15:56:25 GMT", "secure": False, "domain": ".pythonscraping.com"}, {"value": "1", "httponly": F
```

```
alse, 'name': 'has_js', 'path': '/', 'expiry': 1485878185, 'expires': 'Tue, 31 Jan 2017 15:56:25 GMT',
'secure': False, 'domain': 'pythonscraping.com'}]
```

To manipulate cookies, you can call the `delete_cookie()`, `add_cookie()`, and `delete_all_cookies()` functions. In addition, you can save and store cookies for use in other web scrapers. Here's an example to give you an idea how these functions work together:

```
from selenium import webdriver

driver = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver.get("http://pythonscraping.com")
driver.implicitly_wait(1)
print(driver.get_cookies())

savedCookies = driver.get_cookies()

driver2 = webdriver.PhantomJS(executable_path='<Path to Phantom JS>')
driver2.get("http://pythonscraping.com")
driver2.delete_all_cookies()
for cookie in savedCookies:
    driver2.add_cookie(cookie)

driver2.get("http://pythonscraping.com")
driver.implicitly_wait(1)
print(driver2.get_cookies())
```

In this example, the first webdriver retrieves a website, prints the cookies, and then stores them in the variable `savedCookies`. The second webdriver loads the same web- site (technical note: it must load the website first so that Selenium knows which web- site the cookies belong to, even if the act of loading the website does nothing useful for us), deletes all of its cookies and replaces them with the saved cookies from the first webdriver. When loading the page again, the timestamps, codes, and other infor- mation in both cookies should be identical. According to Google Analytics, this sec- ond webdriver is now identical to the first one.

Timing Is Everything

Some well-protected websites might prevent you from submitting forms or interact- ing with the site if you do it too quickly. Even if these security features aren't in place, downloading lots of information from a website significantly faster than a normal human might is a good way to get yourself noticed, and blocked.

Therefore, although multithreaded programming might be a great way to load pages faster—allowing you to process data in one thread while repeatedly loading pages in another—it's a terrible policy for writing good scrapers. You should always try to keep individual page loads and data requests to a minimum. If possible, try to space them out by a few seconds, even if you have to add in an extra:

```
time.sleep(3)
```

Although web scraping often involves rule breaking and boundary pushing in order to get data, this is one rule you don't want to break. Not only does consumingordinate server resources put you in a legally vulnerable position, but you might cripple or take down smaller sites. Bringing down websites is not an ethically ambiguous situation: it's just wrong. So watch your speed!

Common Form Security Features

There are many litmus tests that have been used over the years, and continue to be used, with varying degrees of success, to separate web scrapers from browser-using humans. Although it's not a big deal if a bot downloads some articles and blog posts that were available to the public anyway, it is a big problem if a bot creates thousands of user accounts and starts spamming all of your site's members. Web forms, especially forms that deal with account creation and logins, pose a significant threat to security and computational overhead if they're vulnerable to indiscriminate use by bots, so it's in the best interest of many site owners (or at least they think it is) to try to limit access to the site.

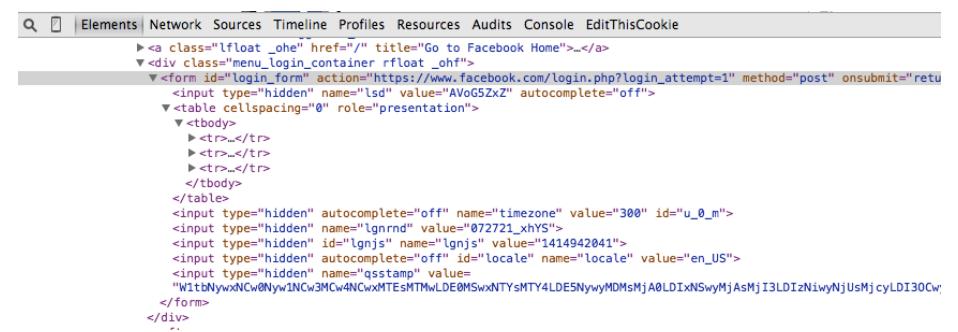
These anti-bot security measures centered on forms and logins can pose a significant challenge to web scrapers.

Keep in mind that this is only a partial overview of some of the security measures you might encounter when creating automated bots for these forms. Review [Chapter 11](#), on dealing with CAPTCHAs and image processing, as well as [Chapter 14](#), on dealing with headers and IP addresses, for more information on dealing with well-protected forms.

Hidden Input Field Values

"Hidden" fields in HTML forms allow the value contained in the field to be viewable by the browser but invisible to the user (unless they look at the site's source code). With the increase in use of cookies to store variables and pass them around on websites, hidden fields fell out of favor for a while before another excellent purpose was discovered for them: preventing scrapers from submitting forms.

[Figure 12-1](#) shows an example of these hidden fields at work on a Facebook login page. Although there are only three visible fields in the form (username, password, and a submit button), the form conveys a great deal of information to the server behind the scenes.



```
<form id="login_form" action="https://www.facebook.com/login.php?login_attempt=1" method="post" onsubmit="retu
  <input type="hidden" name="lsd" value="AVoG5ZXZ" autocomplete="off">
  <table cellspacing="0" role="presentation">
    <tbody>
      <tr></tr>
      <tr></tr>
      <tr></tr>
    </tbody>
  </table>
  <input type="hidden" autocomplete="off" name="timezone" value="300" id="u_0_m">
  <input type="hidden" name="lgnrnd" value="072721_xhYSI">
  <input type="hidden" id="lgnjs" name="lgnjs" value="1414942041">
  <input type="hidden" autocomplete="off" id="locale" name="locale" value="en_US">
  <input type="hidden" name="qsstamp" value=
    "W1tBNywxNCv0Ny1NCw3NCw4NCwxHTEsMTMwLDE0MSwxNTYsMTY4LDE5NywyMDMsMjA0LDIxNswyMjAsMjI3LDIxNiwyNjUsMjcyLDI3OCw
  </form>
</div>
<div>
```

Figure 12-1. The Facebook login form has quite a few hidden fields

There are two main ways hidden fields are used to prevent web scraping: a field can be populated with a randomly generated variable on the form page that the server is expecting to be posted to the form processing page. If this value is not present in the form, then the server can reasonably assume that the submission did not originate organically from the form page, but was posted by a bot directly to the processing page. The best way to get around this measure is to scrape the form page first, collect the randomly generated variable and then post to the processing page from there.

The second method is a "honey pot" of sorts. If a form contains a hidden field with an innocuous name, such as "username" or "email address," a poorly written bot might fill out the field and attempt to submit it, regardless of whether it is hidden to the user or not. Any hidden fields with actual values (or values that are different from their defaults on the form submission page) should be disregarded, and the user may even be blocked from the site.

In short: It is sometimes necessary to check the page that the form is on to see if you missed anything that the server might be expecting. If you see several hidden fields, often with large, randomly generated string variables, it is likely that the web server will be checking for their existence on form submission. In addition, there might be other checks to ensure that the form variables have been used only once, are recently generated (this eliminates the possibility of simply storing them in a script and using them over and over again over time), or both.

Avoiding Honeypots

Although CSS for the most part makes life extremely easy when it comes to differentiating useful information from nonuseful information (e.g., by reading the id and class tags), it can occasionally be problematic for web scrapers. If a field on a web form is hidden from a user via CSS, it is reasonable to assume that the average user visiting the site will not be able to fill it out because it doesn't show up in the browser. If the form is populated, there is likely a bot at work and the post will be discarded.

Para obter informações mais completas sobre a instalação e atualização do Python em todas as plataformas, visite o [Página de downloads Python](#).

Ao contrário do Java, Python pode ser usado como linguagem de script sem criar novas classes ou funções. Abrindo um novo arquivo de texto, escrevendo:

```
impressão ( " Olá, Internet! " )
```

e salvá-lo como *ola.py* é perfeitamente legítimo. Se você quiser ser um pouco mais elaborado, pode criar uma nova função e usá-la para fazer a mesma coisa:

```
def Olá():
    impressão ( " Olá, Internet! " )
Olá ()
```

Existem algumas coisas a serem observadas aqui:

Python não usa ponto-e-vírgula para indicar o final de uma linha, nem usa colchetes para indicar o início e o final de um loop ou função. Em vez disso, ele conta com quebras de linha e guias para controlar a execução. Abrindo uma linha com def, seguido por um nome de função, argumentos (neste caso, temos um()), indicando que não há argumentos), e dois pontos indicam que o corpo de uma função virá em seguida. Cada linha de uma função seguindo uma declaração de função *devo* ser indentado e outdenting indica o fim do corpo da função.

Se a ideia de usar o espaço em branco dessa forma parece um pouco pedante (ou insana) à primeira vista, continue. Na minha experiência, escrever Python na verdade melhora a legibilidade do meu código escrito em outras linguagens. Lembre-se de adicionar ponto-e-vírgula ao voltar para algo como Java ou C!

Python é uma linguagem fracamente tipada, o que significa que os tipos de variáveis (string, inteiro, objeto, etc.) não precisam ser declarados explicitamente quando as variáveis são inicializadas. Tal como acontece com outras linguagens com tipagem fraca, isso pode ocasionalmente causar problemas de depuração, mas torna a declaração de variável muito fácil:

```
cumprimento = "Olá, Internet!"
impressão ( cumprimento )
```

E é isso! Você é um Pythoner!

Obviamente, isso não é exatamente verdade, mas uma das melhores coisas sobre Python é que é uma linguagem tão simples que os programadores de outras linguagens podem ler e interpretá-la sem muita exposição prévia. Este aspecto da linguagem pode provavelmente ser melhor resumido no mais famoso “Ovo de Páscoa” do Python:

```
importar esta
```

O resultado é:

O Zen do Python, de Tim Peters

Belo é melhor do que feio. Explícito é melhor do que implícito. Simples é melhor que complexo. Complexo é melhor do que complicado. Plano é melhor do que aninhado.

O esparso é melhor do que o denso. A legibilidade conta.

Casos especiais não são especiais o suficiente para quebrar as regras. Embora a praticidade supere a pureza.

Os erros nunca devem passar silenciosamente. A menos que seja explicitamente silenciado.

Diante da ambigüidade, recuse a tentação de adivinhar.

Deve haver uma - e de preferência apenas uma - maneira óbvia de fazer isso. Embora esse caminho possa não ser óbvio no início, a menos que você seja holandês.¹

Agora é melhor do que nunca.

Embora nunca seja sempre melhor do que * agora *.

Se a implementação for difícil de explicar, é uma má ideia.

Se a implementação for fácil de explicar, pode ser uma boa ideia. Os namespaces são uma ótima ideia - vamos fazer mais disso!

¹ Esta linha pode ser uma referência ao cientista da computação holandês Edsger Dijkstra, que disse em uma palestra em 1978: "Eu pensei que era um princípio firme de design de linguagem ... que em todos os aspectos programas equivalentes deveriam ter poucas possibilidades para diferentes representações ... Caso contrário, estilos completamente diferentes de programação surgem desnecessariamente, dificultando a manutenção, a legibilidade e tudo o mais" (<http://www.cs.utexas.edu/~EWD/transcriptions/EWD06xx/EWD660.html>). Ou pode ser simplesmente devido ao fato de o criador original do Python, Guido van Rossum, ser holandês. Ninguém parece estar totalmente certo sobre esse assunto, entretanto.

A Internet num relance

À medida que os tipos de transações que a Internet precisa lidar se tornam cada vez mais complexos, os termos e tecnologias usados para descrever essas transações também aumentam em complexidade. Longe de suas raízes como uma forma de trocar mensagens de pesquisa, a Internet agora deve lidar com uploads de arquivos grandes, streaming de vídeo, transações bancárias seguras, compras com cartão de crédito e a transmissão de documentos corporativos confidenciais.

Apesar dessas camadas extras de complexidade, no entanto, a Internet permanece, em seu núcleo, uma série de mensagens. Algumas mensagens contêm solicitações de informações, algumas contêm mensagens destinadas a um destinatário distante, algumas contêm informações de arquivo ou instruções para um aplicativo específico na máquina para a qual está sendo enviado. Essas solicitações são enviadas de uma máquina cliente (desktop ou dispositivo móvel) para um servidor e vice-versa. Eles também podem ser enviados entre os próprios servidores, talvez para reunir mais informações solicitadas por um cliente.

Figura B-1 descreve alguns tipos comuns de transações na Internet: uma solicitação de localização de um servidor em um nome de domínio específico, a solicitação de uma página da web e seu arquivo de imagem associado em dois servidores e o upload de um arquivo de imagem.

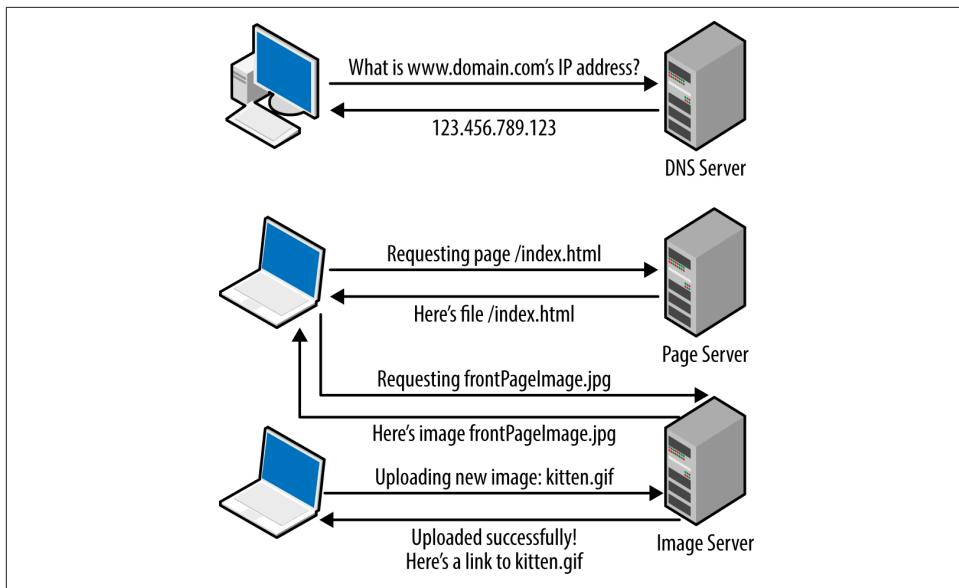


Figura B-1. Alguns tipos de transações comuns da Internet entre clientes e servidores

Existem muitos protocolos ou linguagens diferentes que governam essas comunicações entre clientes e servidores. Você pode receber seu e-mail por meio do protocolo SMTP ao fazer uma chamada telefônica por VOIP e enviar arquivos por FTP. Todos esses protocolos definem campos diferentes para cabeçalhos, codificações de dados, endereços ou nomes de envio e recebimento e outros tipos de coisas. O protocolo usado para solicitar, enviar e receber sites é o Hypertext Transfer Protocol (HTTP).

Para a grande maioria dos scrapers deste livro (e, provavelmente, a maioria dos scrapers que você escreverá), o HTTP é usado para se comunicar com servidores web remotos. Por esse motivo, é importante examinar o protocolo um pouco mais detalhadamente.

Uma mensagem HTTP contém duas partes principais: campos de cabeçalho e um campo de dados. Cada campo de cabeçalho contém um título e um valor. Os títulos possíveis para esses campos são predefinidos pelo padrão HTTP. Por exemplo, você pode ter um campo de cabeçalho como:

Conteúdo - Tipo : inscrição / json

indicando que os dados no pacote HTTP usarão o formato JSON. Existem mais de 60 campos de cabeçalho possíveis que podem ser encontrados em um pacote HTTP, mas usaremos apenas um pequeno conjunto deles neste livro. A tabela a seguir mostra alguns dos campos de cabeçalho HTTP com os quais você deve estar familiarizado:

Nome	Descrição	Exemplo
Do utilizador/Agente	Uma string que indica de qual navegador e sistema operacional você está fazendo a solicitação.	Mozilla / 5.0 (X11; Ubuntu; Linux x86_64; rv: 28.0) Gecko / 20100101 Firefox / 28.0
Bolacha	Variáveis usadas pelo aplicativo da web para conter dados da sessão e outras informações.	"__Ultma: 20549163.147923691.1398729710.1398729710.1398858679.2"
Status	Um código que indica o sucesso ou falha da solicitação de página.	"200" (OK), "404" (não encontrado)

Quando um pacote, transportado por HTTP, chega ao seu navegador, o conteúdo do pacote deve ser interpretado como um site. A estrutura dos sites é regida por HTML, ou *Linguagem de marcação de hipertexto*. Embora HTML seja frequentemente referido como uma linguagem de programação, HTML é um *marcação* língua. Ele define a estrutura de um documento usando tags para demarcar itens como título, conteúdo, barra lateral, rodapé e muito mais.

Todas as páginas HTML (pelo menos todas as bem formatadas) são cercadas por abrir e fechar < html></html> tags, com < cabeça> e < corpo> tags entre. Outras tags preenchem esses cabeçalhos e corpos de página para formar a estrutura e o conteúdo da página:

```
<html>
<head>
<title> Uma página de exemplo </ título>
</head>
<body>
<h1> Uma página de exemplo < h1>
<div classe = "corpo" >
Alguns exemplos de conteúdo estão aqui
</div>
</body>
</html>
```

Neste exemplo, o título da página (este é o texto que é visto em uma guia em seu navegador) é “Uma página de exemplo”, com esse mesmo título ecoado no cabeçalho < h1> tag. Imediatamente abaixo disso está uma tag div (“divider”) da classe “body”, contendo o que pode ser um artigo principal ou um texto mais longo.

Analisando sites para fácil raspagem

Observar quando o conteúdo da página é copiado exatamente em áreas diferentes da página pode ser útil na web scraping - parte da página pode ser mais conveniente ou mais confiável de raspar do que outra parte da página.

Em “Uma página de exemplo”, é claro, o título é copiado de forma bastante óbvia em duas áreas separadas da página, mas casos mais sutis são comuns na web. Você pode estar descartando os nomes de funcionários de um site corporativo, por exemplo, e perceber que os nomes estão formatados de forma inconsistente:

```
< período Eu iria = "mary_smith" > Dr . Mary Smith , CEO </ período > < período Eu iria = "john_jones" > Presidente das  
Finanças, Sr. John E . Jones </ período > < período Eu iria = "stacy_roberts" > Stacy Roberts III , Marketing </ período >
```

Separar o conteúdo dessas tags em pares de palavras “nome, sobrenome” seria difícil. Como você lida com nomes do meio, títulos, posições e outras porcarias nessa bagunça de formatação?

Felizmente, parece que o Eu iria as tags são bem formatadas e separá-las em Python se torna uma questão trivial de dividir a string no caractere sublinhado.

CSS, ou *Cascading Style Sheets*, ande de mãos dadas com o HTML para definir a aparência de um site. CSS define coisas como cor, posição, tamanho e plano de fundo de vários objetos em um site.

Usando o HTML no exemplo anterior, isso pode ser parecido com:

```
h1 {  
    cor : 'vermelho' ;  
    Fonte - Tamanho : 1,5 em ;  
},  
div . corpo {  
    fronteira : 2 px sólido ;  
}
```

Isso cria um título vermelho de tamanho médio, com uma borda ao redor do texto do conteúdo no site.

Infelizmente, entrar em maiores detalhes sobre HTTP, HTML ou CSS está fora do escopo deste livro. No entanto, se você não estiver familiarizado com os assuntos, recomendo que dê uma olhada em [W3Schools](#) para procurar termos ou linhas desconhecidas de HTML / CSS enquanto você lê o livro. Usar o recurso “view source” do seu navegador é uma grande ajuda para começar a se familiarizar com a sintaxe também.

A legalidade e a ética do Web Scraping

Em 2010, o engenheiro de software Pete Warden construiu um web crawler para coletar dados do Facebook. Ele coletou dados de aproximadamente 200 milhões de usuários do Facebook - nomes, informações de localização, amigos e interesses. Claro, o Facebook percebeu e enviou-lhe cartas para parar e desistir, as quais ele obedeceu. Quando questionado sobre por que acatou o cesse e desista, ele disse: "Big data? Barato. Advogados? Não muito."

Neste capítulo, veremos as leis dos EUA (e algumas internacionais) que são relevantes para web scraping e aprenderemos como analisar a legalidade e a ética de uma determinada situação de web scraping.

Antes de ler esta seção, considere o óbvio: sou engenheiro de software, não advogado. Não interprete nada que você ler aqui ou em qualquer outro capítulo do livro como aconselhamento jurídico profissional, nem aja de acordo com isso. Embora eu acredite que posso discutir a legalidade e a ética da web scraping com conhecimento, você deve consultar um advogado (não um engenheiro de software) antes de empreender qualquer projeto de web scraping legalmente ambíguo.

Marcas registradas, direitos autorais, patentes, OhMy!

É hora de alguma Propriedade Intelectual 101! Existem três tipos básicos de PI: marcas registradas (indicadas por um símbolo ™ ou ®), direitos autorais (o onipresente ©) e patentes (às vezes indicadas por texto que descreve que a invenção é protegida por patente, mas muitas vezes por nada) .

As patentes são usadas para declarar propriedade apenas sobre invenções. Você não pode patentear imagens, textos ou qualquer informação em si. Embora algumas patentes, como patentes de software, sejam menos tangíveis do que o que consideramos "invenções", tenha em mente que é o *coisa* (ou técnica) que é patenteado - não as informações contidas na patente. A menos que você esteja construindo coisas a partir de projetos raspados ou alguém patenteie um

método de web scraping, é improvável que você infrinja inadvertidamente uma patente ao copiar a web.

As marcas registradas também dificilmente serão um problema, mas ainda assim algo que deve ser considerado. De acordo com o US Patent and Trademark Office:

UMA **marca comercial** é uma palavra, frase, símbolo e / ou design que identifica e distingue a origem dos bens de uma parte daqueles de outras partes. UMA **marca de serviço** é uma palavra, frase, símbolo e / ou design que identifica e distingue a origem de um serviço em vez de mercadorias. O termo "marca registrada" é freqüentemente usado para se referir a marcas registradas e marcas de serviço.

Além das palavras / símbolos tradicionais em que pensamos quando pensamos em marcas registradas, outros atributos descritivos podem ser marcas registradas. Isso inclui, por exemplo, o formato de um recipiente (pense em garrafas de Coca-Cola) ou mesmo uma cor (mais notavelmente, a cor rosa do isolamento de fibra de vidro Pink Panther da Owens Corning).

Ao contrário das patentes, a propriedade de uma marca depende muito do contexto em que ela é usada. Por exemplo, se eu quiser publicar uma postagem de blog com uma imagem do logotipo da Coca-Cola, eu poderia fazer isso (contanto que não esteja sugerindo que minha postagem de blog foi patrocinada ou publicada pela Coca-Cola) Se eu quisesse fabricar um novo refrigerante com o mesmo logotipo da Coca-Cola exibido na embalagem, isso seria claramente uma violação de marca registrada. Da mesma forma, embora eu pudesse embalar meu novo refrigerante em rosa pantera rosa, não poderia usar essa mesma cor para criar um produto de isolamento doméstico.

Lei de direitos autorais

Tanto as marcas quanto as patentes têm algo em comum, pois devem ser registradas formalmente para serem reconhecidas. Ao contrário da crença popular, isso não é verdade com material protegido por direitos autorais. O que torna imagens, textos, músicas, etc. protegidos por direitos autorais? Não é o aviso "Todos os direitos reservados" na parte inferior da página, nem nada especial sobre material "publicado" versus material "não publicado". Cada pedaço de material que você cria está automaticamente sujeito à lei de direitos autorais assim que você o coloca em existência.

A Convenção de Berna para a Proteção de Obras Literárias e Artísticas, em homenagem a Berna, Suíça, onde foi adotada pela primeira vez em 1886, é o padrão internacional para direitos autorais. Esta convenção diz, em essência, que todos os países membros devem reconhecer a proteção de direitos autorais das obras de cidadãos de outros países membros como se fossem cidadãos de seu próprio país. Na prática, isso significa que, como cidadão americano, você pode ser responsabilizado nos Estados Unidos por violar os direitos autorais de material escrito por alguém, digamos, na França (e vice-versa).

Obviamente, os direitos autorais são uma preocupação para os web scrapers. Se eu extrair conteúdo do blog de alguém e publicar em meu próprio blog, posso muito bem estar me abrindo para um processo judicial. Felizmente, tenho várias camadas de proteção que podem tornar meu projeto de scraping de blog defensável, dependendo de como ele funciona.

Primeiro, a proteção de direitos autorais se estende apenas a trabalhos criativos. Não cobre estatísticas ou fatos. Felizmente, muito do que os web scrapers procuram *está* fatos e estatísticas. Embora um web scraper que reúne poesia de toda a web e exiba essa poesia em seu próprio site possa estar violando as leis de direitos autorais, um web scraper que reúne informações sobre a frequência de postagens de poesia ao longo do tempo não o é. A poesia, em sua forma bruta, é uma obra criativa. A contagem média de palavras de poemas publicados em um site, por mês, não é um trabalho criativo.

Mesmo o conteúdo postado na íntegra (em oposição ao conteúdo agregado / calculado a partir de dados não processados) pode não estar violando as leis de direitos autorais se esses dados forem preços, nomes de executivos da empresa ou alguma outra informação factual.

Mesmo conteúdo protegido por direitos autorais pode ser usado diretamente, dentro do razoável, de acordo com a Lei de Direitos Autorais do Milênio Digital. O DMCA descreve algumas regras para o tratamento automatizado de material protegido por direitos autorais. O DMCA é longo, com muitas regras específicas que regem tudo, de e-books a telefones. No entanto, existem três seções principais que são de particular interesse:

- Sob a proteção de "porto seguro", se você extrair material de uma fonte que você acredita conter apenas material livre de direitos autorais, mas algum usuário enviou material protegido por direitos autorais, você está protegido desde que tenha removido a cópia - material corrigido quando notificado.
- Você não pode contornar medidas de segurança (como proteção por senha) para coletar conteúdo.
- Você pode usar o conteúdo sob a regra de "uso justo", que leva em consideração a porcentagem da obra protegida por direitos autorais usada e a finalidade para a qual está sendo usada.

Resumindo, você nunca deve publicar material protegido por direitos autorais sem a permissão do autor original ou do detentor dos direitos autorais. Se você estiver armazenando material protegido por direitos autorais ao qual tem acesso livre em seu próprio banco de dados não público para fins de análise, tudo bem. Se você estiver publicando esse banco de dados em seu site para visualização ou download, não está bem. Se você estiver analisando esse banco de dados e publicando estatísticas sobre contagens de palavras, uma lista de autores por produtividade ou alguma outra meta-análise dos dados, tudo bem. Se você estiver acompanhando essa meta-análise com algumas citações selecionadas ou breves amostras de dados para demonstrar seu ponto de vista, provavelmente também não há problema, mas você pode querer examinar a cláusula de "uso justo" do DMCA para ter certeza.

Trespass to Chattels

A invasão de bens móveis é fundamentalmente diferente do que consideramos como "leis de violação", pois se aplica não a imóveis ou terras, mas a bens móveis (como um servidor). Aplica-se quando o seu acesso à propriedade é interferido de alguma forma que não permite que você acesse ou use.

Nesta era da computação em nuvem, é tentador não pensar nos servidores da web como recursos reais e tangíveis. No entanto, os servidores não apenas consistem em componentes caros, mas precisam ser armazenados, monitorados, resfriados e fornecidos com grande quantidade de eletricidade. Segundo algumas estimativas, 10% do uso global de eletricidade é consumido por computadores (se uma pesquisa de sua própria eletrônica não convence você, considere os vastos farms de servidores do Google, todos os quais precisam ser conectados a grandes usinas de energia).

Embora os servidores sejam recursos caros, eles são interessantes do ponto de vista jurídico, pois os webmasters em geral *quer* pessoas para consumir seus recursos (ou seja, acessar seus sites); eles apenas não querem que eles consumam seus recursos *demais*. Verificar um site através do seu navegador é bom, lançar um DDOS em escala real contra ele, obviamente, não.

Existem três critérios que precisam ser atendidos para que um raspador de web viole a invasão de bens móveis:

Falta de consentimento

Como os servidores da web estão abertos a todos, eles geralmente estão “dando consentimento” também aos web scrapers. No entanto, os acordos de Termos de Serviço de muitos sites proíbem especificamente o uso de raspadores. Além disso, quaisquer notificações de cessar e desistir obviamente revogam esse consentimento.

Dano real

Os servidores são caros. Além dos custos do servidor, se seus scrapers derrubarem um site ou limitarem sua capacidade de servir a outros usuários, isso pode aumentar o “dano” que você causa.

Intencionalmente

Se você está escrevendo o código, sabe o que ele faz!

Você deve atender a todos esses três critérios para aplicação de invasão de bens móveis. No entanto, se você está violando um contrato de Termos de Serviço, mas não está causando danos reais, não pense que está imune a ações judiciais. Você pode muito bem estar violando a lei de direitos autorais, o DMCA, a Lei de Fraude e Abuso de Computador (mais sobre isso mais tarde), ou uma das outras inúmeras leis que se aplicam a web scrapers.

Estrangulando seus bots

Antigamente, os servidores da web eram muito mais poderosos do que os computadores pessoais. Na verdade, parte da definição de “servidor” era “grande computador”. Agora, a situação mudou um pouco. Meu computador pessoal, por exemplo, tem um processador de 3,5 GHz e 8 GB de RAM. Em contraste, uma instância média da Amazon (na época da escrita deste livro) tem 4 GB de RAM e cerca de 3 GHz de capacidade de processamento.

Com uma conexão de Internet decente e uma máquina dedicada, até mesmo um único computador pessoal pode sobrecarregar muitos sites, até mesmo incapacitando-os ou tomado

para baixo completamente. A menos que haja uma emergência médica e a única cura seja agregar todos os dados do site de Joe Schmo em dois segundos, não há motivo para martelar um site.

Um bot monitorado nunca termina. Às vezes é melhor deixar os rastreadores funcionando durante a noite do que no meio da tarde ou noite por alguns motivos:

- Se você tem cerca de oito horas, mesmo no ritmo glacial de dois segundos por página, pode rastrear mais de 14.000 páginas. Quando o tempo é menos problemático, você não fica tentado a aumentar a velocidade de seus rastreadores.
- Supondo que o público-alvo do site esteja em sua localização geral (ajuste de acordo para públicos-alvo remotos), a carga de tráfego do site é provavelmente muito menor durante a noite, o que significa que seu rastreamento não estará agravando o congestionamento do tráfego de pico.
- Você economiza tempo dormindo, em vez de verificar constantemente seus registros em busca de novas informações.
Pense em como você ficará animado ao acordar de manhã com dados totalmente novos!

Considere os seguintes cenários:

- Você tem um rastreador da web que percorre o site de Joe Schmo, agregando alguns ou todos os seus dados
- Você tem um rastreador que percorre centenas de pequenos sites, agregando alguns ou todos os seus dados.
- Você tem um rastreador da web que percorre um site muito grande, como a Wikipedia.

No primeiro cenário, é melhor deixar o bot funcionando lentamente e durante a noite.

No segundo cenário, é melhor rastrear cada site de maneira round-robin, em vez de rastreá-los lentamente, um de cada vez. Dependendo de quantos sites você está rastreando, isso significa que você pode coletar dados tão rápido quanto sua conexão com a Internet e sua máquina podem gerenciar, embora a carga seja razoável para cada servidor remoto individual. Você pode fazer isso programaticamente, usando vários threads (em que cada thread individual rastreia um único site e pausa sua própria execução) ou usando listas Python para manter o controle dos sites.

No segundo cenário, é improvável que a carga que sua conexão com a Internet e sua máquina doméstica podem colocar em um site como a Wikipedia seja notada ou tenha muita importância. No entanto, se você estiver usando uma rede distribuída de máquinas, isso é obviamente um assunto diferente. Tenha cuidado e pergunte a um representante da empresa sempre que possível.

A Lei de Fraude e Abuso de Computador

No início da década de 1980, os computadores começaram a sair da academia e entrar no mundo dos negócios. Pela primeira vez, vírus e worms foram vistos como mais do que uma inconveniência (ou mesmo um hobby divertido) e como um assunto criminal grave que poderia causar reais

danos monetários. Em resposta, a Lei de Fraude e Abuso de Computador foi criada em 1986.

Embora você possa pensar que o ato se aplica apenas a alguma versão estereotipada de um hacker malicioso que libera vírus, o ato também tem fortes implicações para os web scrapers. Imagine um raspador que varre a web em busca de formulários de login com senhas fáceis de adivinhar ou coleta segredos do governo incidentalmente deixados em um local oculto, mas público. Todas essas atividades são ilegais (e com razão) segundo o CFAA.

A lei define sete crimes principais, que podem ser resumidos da seguinte forma:

- O conhecimento de acesso não autorizado a computadores de propriedade do governo dos EUA e a obtenção de informações desses computadores.
- O conhecimento de acesso não autorizado a um computador, obtenção de informações financeiras.
- O conhecimento de acesso não autorizado a um computador de propriedade do governo dos EUA, afetando o uso desse computador pelo governo.
- Acessar conscientemente qualquer computador protegido com a tentativa de fraude.
- Acessar conscientemente um computador sem autorização e causar danos a esse computador.
- Compartilha ou trazega senhas ou informações de autorização para computadores usados pelo governo dos Estados Unidos ou computadores que afetam o comércio interestadual ou estrangeiro.
- Tentativas de extorquir dinheiro ou “qualquer coisa de valor” causando danos ou ameaçando causar danos a qualquer computador protegido.

Resumindo: fique longe de computadores protegidos, não acesse computadores (incluindo servidores web) aos quais você não tenha acesso e, principalmente, fique longe de computadores governamentais ou financeiros.

robots.txt e Termos de Serviço

Os termos de serviço de um site e *robots.txt* os arquivos estão em um território interessante, legalmente falando. Se um site for acessível ao público, o direito do webmaster de declarar qual software pode e não pode acessá-lo é discutível. Dizer “está tudo bem se você usar seu navegador para visualizar este site, mas não se você usar um programa que escreveu para visualizá-lo” é complicado.

A maioria dos sites tem um link para seus Termos de Serviço no rodapé de cada página. O TOS contém mais do que apenas regras para rastreadores da web e acesso automatizado; frequentemente contém informações sobre o tipo de informação que o site coleta, o que faz com elas e, geralmente, uma isenção de responsabilidade legal de que os serviços prestados pelo site são fornecidos sem qualquer garantia expressa ou implícita.

Se você está interessado em otimização de mecanismo de pesquisa (SEO) ou tecnologia de mecanismo de pesquisa, provavelmente já ouviu falar do *robots.txt* Arquivo. Se você for a qualquer grande web

site e procure por seu *robots.txt* arquivo, você o encontrará na pasta raiz da web: <http://website.com/robots.txt>.

A sintaxe para *robots.txt* files foi desenvolvida em 1994 durante o boom inicial da tecnologia de mecanismo de pesquisa na web. Foi nessa época que os motores de busca que vasculhavam toda a Internet, como AltaVista e DogPile, começaram a competir seriamente com listas simples de sites organizados por assunto, como a curadoria do Yahoo! Esse crescimento da pesquisa na Internet significou uma explosão não apenas no número de rastreadores da web, mas na disponibilidade das informações coletadas por esses rastreadores para o cidadão comum.

Embora possamos considerar esse tipo de disponibilidade um dado adquirido hoje, alguns webmasters ficaram chocados quando as informações que publicaram nas profundezas da estrutura de arquivos de seu site tornaram-se disponíveis na página inicial dos resultados de pesquisa nos principais mecanismos de pesquisa. Em resposta, a sintaxe para *robots.txt* arquivos, chamados de Robots Exclusion Standard, foi desenvolvida.

Ao contrário do TOS, que frequentemente fala sobre rastreadores da web em termos gerais e em uma linguagem muito humana, *robots.txt* pode ser analisado e usado por programas automatizados com extrema facilidade. Embora possa parecer o sistema perfeito para resolver o problema de bots não autorizados de uma vez por todas, tenha em mente o seguinte:

- Não existe um órgão oficial de gestão para a sintaxe de *robots.txt*. É uma convenção comumente usada e geralmente bem seguida, mas não há nada que impeça ninguém de criar sua própria versão de um *robots.txt* (além do fato de que nenhum bot irá reconhecê-lo ou obedecê-lo até que se torne popular). Dito isso, é uma convenção amplamente aceita, principalmente porque é relativamente direta e não há incentivo para as empresas inventarem seu próprio padrão ou tentarem melhorá-lo.
- Não há como fazer cumprir uma *robots.txt* Arquivo. É apenas um sinal que diz "Por favor, não vá para estas partes do site". Existem muitas bibliotecas de web scraping disponíveis que obedecem *robots.txt* (embora esta seja apenas uma configuração padrão que pode ser substituída). Além disso, muitas vezes existem mais barreiras para seguir um *robots.txt* (afinal, você precisa raspar, analisar e aplicar o conteúdo da página à lógica do código) do que simplesmente seguir em frente e raspar qualquer página que desejar.

A sintaxe do Robot Exclusion Standard é bastante direta. Como no Python (e em muitas outras linguagens), os comentários começam com um símbolo #, terminam com uma nova linha e podem ser usados em qualquer lugar no arquivo.

A primeira linha do arquivo, além de quaisquer comentários, é iniciada com Agente de usuário:, que especifica o usuário ao qual as seguintes regras se aplicam. Isso é seguido por um conjunto de regras, Permitir: ou Disallow :, dependendo se o bot é permitido nessa seção do site ou não. Um asterisco (*) indica um caractere curinga e pode ser usado para descrever um Agente de usuário ou um URL.

Se uma regra segue uma regra que parece contradizer, a última regra tem precedência. Por exemplo:

```
# Bem-vindo ao meu arquivo robots.txt!
```

```
Do utilizador - agente : *
```

```
Não permitir : *
```

```
Do utilizador - agente : Googlebot
```

```
Permitir : *
```

```
Não permitir : / privado
```

Nesse caso, todos os bots não são permitidos em qualquer lugar do site, exceto para o bot do Google, que é permitido em qualquer lugar, exceto para o / *privado* diretório.

No Twitter *robots.txt* arquivo, ele contém instruções explícitas para os bots do Google, Yahoo !, Yandex (um mecanismo de pesquisa russo popular), Microsoft e outros bots ou mecanismos de pesquisa não cobertos por nenhuma das categorias anteriores. A seção do Google (que parece idêntica às permissões concedidas a todas as outras categorias de bots) se parece com:

```
# User-agent do robô do mecanismo de pesquisa  
do Google: Googlebot
```

```
Permitir: /? _ Escaped_fragment_
```

```
Permitir: /? Lang =
```

```
Permitir: / hashtag / *? Src =
```

```
Permitir: / search? Q =% 23
```

```
Disallow: / search / realtime
```

```
Disallow: / search / users
```

```
Disallow: / search / * / grid
```

```
Disallow: / *?
```

```
Disallow: / * / Seguidores
```

```
Disallow: / * / following
```

Observe que o Twitter restringe o acesso às partes de seu site para as quais ele possui uma API. Como o Twitter tem uma API bem regulamentada (e com a qual pode lucrar com o licenciamento), é do interesse da empresa proibir quaisquer “APIs caseiras” que coletam informações rastreando independentemente seu site.

Embora um arquivo que diga ao seu rastreador onde ele não pode ir possa parecer restritivo no início, pode ser uma bênção disfarçada para o desenvolvimento do rastreador da web. Se você encontrar um *robots.txt* arquivo que não permite o rastreamento em uma seção específica do site, o webmaster está dizendo, essencialmente, que eles aceitam rastreadores em todas as outras seções do site (afinal, se eles não estivessem bem com isso, eles teriam acesso restrito quando eles estavam escrevendo *robots.txt* em primeiro lugar).

Por exemplo, a seção da Wikipedia *robots.txt* arquivo que se aplica a web scrapers em geral (ao contrário de mecanismos de pesquisa) é maravilhosamente permissivo. Ele chega a conter texto legível por humanos para dar as boas-vindas aos bots (somos nós!) E bloqueia o acesso a apenas algumas páginas, como a página de login, a página de pesquisa e a página de “artigo aleatório”:

```
#  
# Bots amigáveis e de baixa velocidade são bem-vindos vendo as páginas dos artigos, mas não  
# páginas geradas dinamicamente, por favor.  
#  
# O "Slurp" de Inktomi pode ler um atraso mínimo entre os acessos; se o seu bot suporta  
# tal coisa usando 'Crawl-delay' ou outra instrução, por favor, deixe-nos  
# conhecer.  
#  
# Há uma exceção especial para API de visualização móvel para permitir web móvel dinâmica e  
# visualizações do aplicativo para carregar o conteúdo da seção.  
# Essas visualizações não são armazenadas em cache de HTTP, mas usam o cache do analisador de forma agressiva e não  
# expor especial: páginas etc.  
#  
Agente de usuário: *  
Permitir: /w/api.php?action=mobileview&  
Disallow: / w /  
Disallow: / trap /  
Disallow: / wiki / Especial: Pesquisa  
Disallow: / wiki / Especial% 3ASearch  
Disallow: / wiki / Special: Coleção  
Disallow: / wiki / Spezial: Sammlung  
Disallow: / wiki / Special: Random  
Disallow: / wiki / Special% 3ARandom  
Disallow: / wiki / Special: Search  
Disallow: / wiki / Special% 3ASearch  
Disallow: / wiki / Spesial: Pesquisa  
Disallow: / wiki / Spesial% 3ASearch  
Disallow: / wiki / Spezial: Pesquisa  
Disallow: / wiki / Spezial% 3ASearch  
Disallow: / wiki / Specjalna: Pesquisa  
Disallow: / wiki / Specjalna% 3ASearch  
Disallow: / wiki / Speciaal: Pesquisa  
Disallow: / wiki / Speciaal% 3ASearch  
Disallow: / wiki / Speciaal: Random  
Disallow: / wiki / Speciaal% 3ARandom  
Disallow: / wiki / Speciel: Pesquisa  
Disallow: / wiki / Speciel% 3ASearch  
Disallow: / wiki / Speciale: Pesquisa  
Disallow: / wiki / Speciale% 3ASearch  
Disallow: / wiki / Istimewa: Search  
Disallow: / wiki / Istimewa% 3ASearch  
Disallow: / wiki / Toiminnot: Pesquisar  
Disallow: / wiki / Toiminnot% 3ASearch
```

Se você optar por escrever rastreadores da web que obedecem *robots.txt* depende de você, mas eu recomendo fortemente, principalmente se você tiver rastreadores que rastreiam indiscriminadamente a web.

ThreeWeb Scrapers

Como o web scraping é um campo ilimitado, há um número impressionante de maneiras de cair em águas quentes legais. Nesta seção, veremos três casos que abordaram alguma forma de lei que geralmente se aplica aos web scrapers e como ela foi usada nesse caso.

eBay versus Edge do Licitante e Trespass to Chattels

Em 1997, o mercado do Beanie Baby estava em alta, o setor de tecnologia estava fervilhando e as casas de leilão online eram uma novidade na Internet. Uma empresa chamada Bidder's Edge formou e criou um novo tipo de site de meta-leilão. Em vez de forçá-lo a ir de site de leilão em site de leilão, comparando preços, ele agregaria dados de todos os leilões atuais de um produto específico (digamos, uma nova boneca Furby ou uma cópia de

Spice World) e apontam para o site com o menor preço.

O Bidder's Edge conseguiu isso com um exército de web scrapers, fazendo solicitações constantes aos servidores da web dos vários sites de leilão para obter informações sobre preços e produtos. De todos os sites de leilão, o eBay era o maior, e o Bidder's Edge atingia os servidores do eBay cerca de 100.000 vezes por dia. Mesmo para os padrões de hoje, é muito tráfego. De acordo com o eBay, isso representava 1,53% de seu tráfego total de Internet na época, e certamente não estava feliz com isso.

O eBay enviou à Bidder's Edge uma carta de cessar e desistir, juntamente com uma oferta para licenciar seus dados. No entanto, as negociações para esse licenciamento falharam e a Bidder's Edge continuou a rastrear o site do eBay.

O eBay tentou bloquear os endereços IP usados pelo Bidder's Edge, bloqueando 169 endereços IP diferentes, embora o Bidder's Edge tenha conseguido contornar isso usando servidores proxy (servidores que encaminham solicitações em nome de outra máquina, mas usando o próprio endereço IP do servidor proxy). Como tenho certeza de que você pode imaginar, esta foi uma solução frustrante e insustentável para ambas as partes - a Bidder's Edge estava constantemente tentando encontrar novos servidores proxy e comprar novos endereços IP enquanto os antigos eram bloqueados, enquanto o eBay era forçado a manter grandes listas de firewall (e adicionando sobrecarga de comparação de endereço IP computacionalmente cara a cada verificação de pacote).

Finalmente, em dezembro de 1999, o eBay processou a Bidder's Edge sob invasão de bens móveis.

Como os servidores do eBay eram recursos reais e tangíveis de sua propriedade, e não apreciava o uso anormal deles pela Bidder's Edge, a invasão de bens móveis parecia a lei ideal a ser usada. Na verdade, nos tempos modernos, a invasão de bens móveis anda de mãos dadas com processos judiciais de web scraping e, na maioria das vezes, é considerada uma lei de TI.

Os tribunais decidiram que, para que o eBay ganhe o caso usando invasão de bens móveis, o eBay deve mostrar duas coisas:

- O Bidder's Edge não tinha permissão para usar os recursos do eBay
- eBay sofreu perda financeira como resultado das ações da Bidder's Edge

Dado o registro das cartas de cessar e desistir do eBay, juntamente com os registros de TI que mostram o uso do servidor e os custos reais associados aos servidores, isso foi relativamente fácil para o eBay fazer. Obviamente, nenhuma grande batalha judicial termina facilmente: contra-ações foram movidas, muitos advogados foram pagos e a questão foi resolvida fora do tribunal por uma quantia não revelada em março de 2001.

Então, isso significa que qualquer uso não autorizado do servidor de outra pessoa é automaticamente uma violação de invasão de bens móveis? Não necessariamente. Bidder's Edge foi um caso extremo; estava usando tantos recursos do eBay que a empresa teve que comprar servidores adicionais, pagar mais pela eletricidade e talvez contratar pessoal adicional (embora 1,53% pode não parecer muito, nas grandes empresas pode somar uma quantia significativa)

Em 2003, a Suprema Corte da Califórnia decidiu sobre outro caso, Intel Corp versus Hamidi, no qual um ex-funcionário da Intel (Hamidi) enviou e-mails que a Intel não gostou, em servidores da Intel, para funcionários da Intel. O tribunal disse:

A alegação da Intel falha não porque o e-mail transmitido pela Internet goza de imunidade única, mas porque a invasão de bens móveis - ao contrário das causas de ação mencionadas - não pode, na Califórnia, ser provada sem evidências de um dano ao autor da ação. propriedade pessoal ou interesse jurídico sobre eles.

Essencialmente, a Intel não conseguiu provar que os custos de transmissão dos seis e-mails enviados por Hamidi a todos os funcionários (cada um, curiosamente, com a opção de ser removido da lista de mala direta de Hamidi - pelo menos ele foi educado!) Contribuíram para qualquer financiamento lesão social para eles. Isso não privou a Intel de qualquer propriedade ou uso de sua propriedade.

Estados Unidos v. Auernheimer e The Computer Fraud and Abuse Act

Se houver informações prontamente acessíveis na Internet para um humano usando um navegador da web, é improvável que acessar as mesmas informações exatas em uma moda automatizada o levaria a uma situação difícil com os federais. No entanto, tão fácil quanto pode ser para uma pessoa suficientemente curiosa encontrar um pequeno vazamento de segurança, esse pequeno vazamento de segurança pode rapidamente se tornar muito maior e muito mais perigoso quando raspadores automatizados entram em cena.

Em 2010, Andrew Auernheimer e Daniel Spitzer notaram um bom recurso dos iPads: quando você visitava o site da AT&T neles, a AT&T o redirecionava para um URL contendo o número de identificação exclusivo do seu iPad:

<https://dcp2.att.com/OEPClient/openPage?ICCID=<idNumber>&IMEI=>

Esta página conteria um formulário de login, com o endereço de e-mail do usuário cujo número de identificação estava na URL. Isso permitiu que os usuários obtivessem acesso às suas contas simplesmente digitando sua senha.

Embora houvesse um número muito grande de números de ID de iPad em potencial, era possível, com scrapers da web suficientes, iterar através dos números possíveis, reunindo endereços de e-mail ao longo do caminho. Ao fornecer aos usuários esse conveniente recurso de login, a AT&T, em essência, tornou os endereços de e-mail de seus clientes públicos na web.

Auernheimer e Spitzer criaram um raspador que coletou 114.000 desses endereços de e-mail, entre eles os endereços de e-mail privados de celebridades, CEOs e funcionários do governo. Auernheimer (mas não Spitzer) enviou a lista e as informações sobre como ela foi obtida para a Gawker Media, que publicou a história (mas não a lista) sob o título: "A pior violação de segurança da Apple: 114.000 proprietários de iPad expostos".

Em junho de 2011, a casa de Auernheimer foi invadida pelo FBI em conexão com a coleta do endereço de e-mail, embora eles acabassem prendendo-o por acusações de drogas. Em novembro de 2012, ele foi considerado culpado de fraude de identidade e conspiração para acessar um computador sem autorização, e algum tempo depois foi condenado a 41 meses em prisão federal e condenado a pagar \$ 73.000 em restituição.

Seu caso chamou a atenção do advogado de direitos civis Orin Kerr, que se juntou à sua equipe jurídica e apelou para o Tribunal de Apelações do Terceiro Circuito. Em 11 de abril de 2014 (esses processos legais podem demorar um pouco), o The Third Circuit concordou com o recurso, dizendo:

A condenação de Auernheimer na contagem 1 deve ser anulada porque visitar um site disponível publicamente não é um acesso não autorizado de acordo com a Lei de Fraude e Abuso de Computador, 18 USC § 1030 (a) (2) (C). A AT&T optou por não empregar senhas ou quaisquer outras medidas de proteção para controlar o acesso aos endereços de e-mail de seus clientes. É irrelevante que a AT&T desejasse subjetivamente que estranhos não tropeçassem nos dados ou que Auernheimer caracterizasse hiperbolicamente o acesso como um "roubo". A empresa configurou seus servidores para disponibilizar as informações a todos e, assim, autorizou o público em geral a visualizar as informações. O acesso aos endereços de e-mail através do site público da AT&T foi autorizado pela CFAA e, portanto, não constituiu crime.

Assim, a sanidade prevaleceu no sistema legal, Auernheimer foi libertado da prisão no mesmo dia e todos viveram felizes para sempre.

Embora tenha sido decidido que Auernheimer não violou a Lei de Fraude e Abuso de Computador, ele teve sua casa invadida pelo FBI, gastou muitos milhares de dólares em custas judiciais e passou três anos entrando e saindo de tribunais e prisões. Como web scrapers, que lições podemos tirar disso para evitar situações semelhantes?

A coleta de qualquer tipo de informação sensível, seja dados pessoais (neste caso, endereços de e-mail), segredos comerciais ou segredos governamentais, provavelmente não é algo que você deseja fazer sem ter um advogado na discagem rápida. Mesmo se estiver disponível publicamente, pense:

"O usuário médio de computador seria capaz de acessar facilmente essas informações se quisesse vê-las?" "Isso é algo que a empresa deseja que os usuários vejam?"

Em várias ocasiões, liguei para empresas para relatar vulnerabilidades de segurança em seus sites e aplicativos da web. Esta linha faz maravilhas: "Olá, sou um profissional de segurança que descobri uma vulnerabilidade de segurança potencial em seu site. Você poderia me indicar alguém para que eu possa denunciá-lo e resolver o problema?" Além da satisfação imediata do reconhecimento por seu gênio hacker (chapéu branco), você poderá obter assinaturas gratuitas, recompensas em dinheiro e outras vantagens com isso!

Além disso, a divulgação das informações de Auernheimer para a Gawker Media (antes de notificar a AT&T) e sua exibição em torno da exploração da vulnerabilidade também o tornaram um alvo especialmente atraente para os advogados da AT&T.

Se você encontrar vulnerabilidades de segurança em um site, a melhor coisa a fazer é alertar os proprietários do site, não a mídia. Você pode ficar tentado a escrever uma postagem no blog e anunciar a ao mundo, especialmente se uma solução para o problema não for implementada imediatamente. No entanto, você precisa lembrar que é responsabilidade da empresa, não sua. A melhor coisa que você pode fazer é tirar seus web scrapers (e, se aplicável, sua empresa) do site!

Campo v. Google: Copyright e robots.txt

Blake Field, um advogado, entrou com um processo contra o Google com base no fato de que seu recurso de cache de sites violava a lei de direitos autorais ao exibir uma cópia de seu livro depois que ele o removeu de seu site. A lei de direitos autorais permite que o criador de uma obra criativa original tenha controle sobre a distribuição dessa obra. O argumento de Field era que o cache do Google (depois que ele o removeu de seu site) removeu sua capacidade de controlar sua distribuição.



O GoogleWeb Cache

Quando os web scrapers do Google (também conhecidos como "bots do Google") rastreiam sites, eles fazem uma cópia do site e a hospedam na Internet. Qualquer pessoa pode acessar esse cache, usando o formato de URL:

<http://webcache.googleusercontent.com/search?q=cache:http://pythonscraping.com/>

Se um site que você está procurando ou copiando não estiver disponível, você pode verificar se existe uma cópia utilizável!

Saber sobre o recurso de cache do Google e não tomar providências não ajudou no caso de Field. Afinal, ele poderia ter evitado que os bots do Google armazenassem em cache seu site simplesmente adicionando o *robots.txt* arquivo, com diretivas simples sobre quais páginas devem e não devem ser raspadas.

Mais importante ainda, o tribunal concluiu que a disposição do DMCS Safe Harbor permitia ao Google armazenar e exibir legalmente sites como o de Field: “[um] provedor de serviços não será responsável por compensação monetária ... por violação de direitos autorais em razão de armazenamento intermediário e temporário de material em um sistema ou rede controlada ou operada por ou para o provedor de serviços.”

Índice

Símbolos

" (aspas), [17](#)
\$ (cifrão), [27](#)
() (parênteses), [25](#)
* (asterisco), [25 , 157](#)
+ (sinal de mais), [25](#)
- (hífen), [113](#)
. (período), [25](#)
Erro 403 proibido, [187](#)
Erro 404 de página não encontrada, [9](#)
500 Erro interno do servidor, [9](#)
; (ponto e vírgula), [210](#)
?! (não contém), [27](#)
[] (colchetes), [25](#)
\ (barra), [27](#)
^ (circunflexo), [27](#)
_ (sublinhado), [17](#)
| (tubo), [25](#)

UMA

uma etiqueta, [28 , 156](#)
Aceite o cabeçalho, [179](#)
Cabeçalho Accept-Encoding, [179](#)
Cabeçalho Accept-Language, [179](#)
cadeias de ação, [195](#)
ActionScript, [147](#)
função add_cookie, [182](#)
Ajax (JavaScript assíncrono e XML),
[151 - 152](#)

sobre, [49 - 50 , 68](#)
autenticação e, [52](#)
convenções comuns, [50 - 52](#)
Exemplo de Echo Nest, [52 , 54 - 55](#)
Exemplos do Google, [50 , 60 - 63](#)
Métodos HTTP e, [51](#)
analizando JSON, [63](#)
respostas, [52](#)
Exemplo do Twitter, [55 - 59](#)
Exemplo da Wikipedia, [64](#)
Codificação de caracteres ASCII, [95 - 98](#)
asserções (testes de unidade), [190](#)
asterisco (*), [25 , 157](#)
JavaScript assíncrono e XML (Ajax),
[151 - 152](#)
Exceção AttributeError, [10](#)
atributos
 acessando, [14 , 28](#)
 encontrar tags com base em, [15 - 18](#)
Auernheimer, Andrew, [227](#)
módulo de autenticação, [144](#)
autenticação
 sobre, [52](#)
 lidar com logins, [142 - 144](#)
 Acesso básico HTTP, [144](#)
 Exemplo do Twitter, [57](#)

B

fazendo backup de dados, [172](#)
Biblioteca BeautifulSoup
 sobre, [6](#)
 função children (), [20](#)
 descendants () function, [20](#)
 função find (), [16 - 18](#)

- função findAll (), 15 - 18 , 28
função get_text (), 15
instalando, 6
função next_siblings (), 21
função previous_siblings (), 21
expressões regulares e, 27
corrida, 8
procurando por tags, 14 - 22
XPath e, 157
- Objeto BeautifulSoup, 15 , 18 , 192
Convenção de Berna para a Proteção de Literatura e Obras Artísticas, 218
- tag do corpo, 215
. extensão de arquivo box, 171
- construção de web scrapers
análise avançada de HTML, 13 - 29
rastreando pela web, 31 - 48
primeiro raspador de teia, 3 - 11
lendo documentos, 93
armazenando dados, 71 - 91
usando APIs, 49 - 69
- Por objeto, 155
Objeto BytesIO, 103
- C**
- Caracteres CAPTCHA
sobre, 161 , 169 - 171
arrastando, 197
treinamento de máquina e, 135 , 171 - 174
recuperando, 174 - 176
acento circunflexo (^), 27
Carroll, Lewis, 6
Cascading Style Sheets (CSS)
sobre, 14 , 216
HTML dinâmico e, 151
campos ocultos e, 184
CGI (Common Gateway Interface), 204
lista de verificação, humana, 186
filhos (tags), 20
função children (), 20
Ferramenta de desenvolvedor do Chrome, 141
atributo de classe, 14 , 17 , 156
limpando dados sujos
limpeza após o fato, 113 - 118
limpeza em código, 109 - 113
processamento do lado do cliente
lidar com redirecionamentos, 44 , 158
línguagens de script e, 147
computação em nuvem, 204
- selecionadores de cores, 140
arquivos de valores separados por vírgula (CSV)
lendo, 98 - 100
armazenar dados para, 74 - 76
- Objeto de comentário, 18
Interface de gateway comum (CGI), 204
Lei de Fraude e Abuso de Computador, 221 , 227 - 229
Cabeçalho de conexão, 179
Objeto de conexão, 83
modelo de conexão / cursor, 83
gramáticas livres de contexto, 135
biscoitos
tratamento, 142 - 143 , 181 - 182
verificar as configurações, 179
lei de direitos autorais, 218 - 219 , 229
- software cPanel, 204
rastreando pela Web (consulte rastreadores da web)
- Instrução CREATE DATABASE, 80
Declaração CREATE INDEX, 86
Instrução CREATE TABLE, 80
credenciais
Contas do Google, 60
Contas do Twitter, 58
- CSS (Cascading Style Sheets)
sobre, 14 , 216
HTML dinâmico e, 151
campos ocultos e, 184
Arquivos CSV (valores separados por vírgula)
lendo , 98 - 100
armazenar dados para, 74 - 76
biblioteca csv, 98 - 100
Objeto cursor, 83
- D**
- Dark Web, 36
coleta de dados, 36 , 38 - 40
gestão de dados
sobre, 71
e-mail e, 90 - 91
MySQL e, 76 - 89
armazenamento de dados em CSV, 74 - 76
armazenamento de arquivos de mídia, 71 - 74
normalização de dados, 112 - 113
armazéns de dados, 40
tamanho do banco de dados versus tempo de consulta, 86
Davies, Mark, 121
desativar o comando, 8
Rede profunda, 36
Método DELETE (HTTP), 51

Declaração DELETE, 82
função delete_all_cookies, 182
função delete_cookie, 182
delimitadores, 74
descendentes (tags), 20
descendants () function, 20
Declaração DESCRIBE, 80
DHTML (HTML dinâmico), 151 - 152
dicionários, 85
Objeto DictReader, 100
Digital Millennium Copyright Act (DMCA),
 219
problemas de gráfico direcionado, 126
display: nenhum atributo, 185
computação distribuída, 201
DMCA (Digital Millennium Copyright Act),
 219
. formato doc, 102
documentos, leitura (ver leitura de documentos)
. formato docx, 102 - 105
cifrão (\$), 27
baixando arquivos da Internet, 74
interfaces de arrastar e soltar, 196
HTML dinâmico (DHTML), 151 - 152

E

Ovos de pascoa, 210
eBay x Bidder's Edge, 226
API Echo Nest, 52 , 54 - 55
Extensão do Chrome EditThisCookie, 181
elementos (selênio), 153 , 194
o email
 identificando endereços, 24
 enviando e recebendo, 90 - 91
pacote de e-mail, 90
codificação (documento)
 sobre, 93
 arquivos de texto e, 94 - 98
variáveis ambientais, 163
personagens de escape, 27 , 110
diretrizes éticas, 177 - 178 , 217 - 230
manipulação de exceção
 links externos, 43
 lidar com redirecionamentos, 158
 conexões de rede, 9
 sugestões para, 35 , 40
espera explícita, 155
eXtensible Markup Language (XML), 52
links externos

cuidados usando, 41
rastreando pela Internet, 40 - 45
rastrejando com Scrapy, 45 - 48
encontrando, 42

F

Site de mídia social do Facebook, 217
cláusula de uso justo, 219
raspagem rápida, 182 , 187
Sequência de Fibonacci, 149
Field v. Google, 229
atributo de arquivo, 141
Objeto de arquivo, 142
uploads de arquivos, 141
filtrando dados, 115 - 116 , 165
finalmente declaração, 85
função find (), 16 - 18
função findAll (), 15 - 18 , 28
para loops, 39
formulários
 sobre, 137
 uploads de arquivos e, 141
lidar com logins e cookies, 142 - 144
campos ocultos em, 183 - 186
imagens em, 141 , 161
campos de entrada suportados, 140
bots maliciosos, 144
considerações de segurança, 183 - 186
submetendo o básico, 138 - 140
barra (), 27
distribuições de frequência, 131 - 132
funções
 manipulação em JavaScript, 148
 expressões lambda e, 28

G

juntando informação, 36 , 38 - 40
Método GET (HTTP)
 sobre, 51
 Exemplo do Google, 62
 recuperando dados, 53
 pedidos de rastreamento, 140
função get_cookies, 181
função get_text (), 15
Google
 Exemplos de API, 50 , 60 - 63
 construção, 40
 Exemplo de modelo de Markov, 124
Google Analytics, 150 , 181

Google Maps, 150
GREL (linguagem de expressão OpenRefine), 116

H

tag h1, 9 , 39
tag de cabeça, 98 , 215
cabeçalhos (HTTP), 179 - 180 , 187
campos ocultos em formulários, 183 - 186
Gerenciador de pacotes Homebrew, 78
homônimos, 133
honeypots, 184 - 186
Cabeçalho do host, 179
hotlinking, 72
atributo href, 28
HTML (linguagem de marcação de hipertexto), 215
Biblioteca HTML Parser, 29
Análise de HTML
 acessando atributos, 28
 evitando a necessidade de, 13
 Exemplo de BeautifulSoup, 14 - 22
 expressões lambda, 28
 expressões regulares, 22 - 28
tag html, 215
HTTP (protocolo de transferência de hipertexto)
 Funcionalidade da API e, 50
 autenticação de acesso básico, 144
 Manipulação de erros, 9 , 187
 cabeçalhos suportados, 179 - 180
 métodos suportados, 51
Objeto HTTPBasicAuth, 144
lista de verificação humana, 186
HyperText Markup Language (HTML), 215
Protocolo de transferência de hipertexto (ver HTTP) hífen
(-), 113

Eu

atributo id, 156
processamento de imagem
 raspar texto de imagens, 166 - 169
 enviar arquivos de imagem, 141
 reconhecimento de texto e, 161 - 176
tag img, 28
espera implícita, 155
indexação, 85
Innes, Nick, 100
tag de entrada, 141
Instrução INSERT INTO, 81 , 84
Intel Corp v. Hamidi, 227
propriedade intelectual, 217 - 219

links internos
 rastreando um site inteiro, 35 - 40
 rastrejando com Scrapy, 45 - 48
 atravessando um único domínio, 31 - 35

Internet

sobre, 213 - 216
alerta para baixar arquivos de, 74
rastrejando, 40 - 45
avançando, 206
Bloqueio de endereço IP, evitando, 199 - 200
Conjuntos de caracteres ISO, 96 - 98
função is_displayed, 186
Objeto de item, 46 , 48
arquivo items.py, 46

J

JavaScript
 sobre, 147 - 149
 bibliotecas comuns, 149 - 151
 executando com selênio, 152 - 156
 lidar com redirecionamentos, 158
 importar arquivos, 14
JavaScript Object Notation (JSON)
 sobre, 52
 análise, 63
biblioteca jQuery, 149
JSON (JavaScript Object Notation)
 sobre, 52
 análise, 63

K

Kerr, Orin, 228
palavras-chave, 17

eu

expressões lambda, 28 , 74
legalidades de web scraping, 217 - 230
análise lexicográfica com NLTK, 132 - 136
bibliotecas
 agrupamento com projetos, 7
 Suporte OCR, 161 - 164
 registrando com o Scrapy, 48
logins
 sobre, 137
 tratamento, 142 - 143
 solução de problemas, 187
biblioteca lxml, 29

M

aprendizado de máquina, 135 , 180
treinamento da máquina, 135 , 171 - 174
Geradores de texto Markov, 123 - 129
arquivos de mídia, armazenamento, 71 - 74
Algoritmo Mersenne Twister, 34
métodos (HTTP), 51
Microsoft SQL Server, 76
Microsoft Word, 102 - 105
MIME (Multipurpose Internet Mail Extensions) protocolo, 90
Objeto MIMEText, 90
MySQL
 sobre, 76
 comandos básicos, 79 - 82
 técnicas de banco de dados, 85 - 87
 instalando, 77 - 79
 integração com Python, 82 - 85
 Exemplo da Wikipedia, 87 - 89

N

atributo de nome, 140
processamento de linguagem natural
 sobre, 119
 Recursos adicionais, 136
 Modelos Markov, 123 - 129
 Natural Language Toolkit, 129 - 136
 resumindo dados, 120 - 123
Kit de ferramentas de linguagem natural (NLTK)
 sobre, 129
 instalação e configuração, 129
 análise lexicográfica, 132 - 136
 análise estatística, 130 - 132
Objeto NavigableString, 18
navegando em árvores, 18 - 22
conexões de rede
 sobre, 3 - 5
 conectando-se de forma confiável, 9 - 11
 considerações de segurança, 181
função next_siblings (), 21
módulo ngrams, 132
n-gramas, 109 - 112 , 120
NLTK (Natural Language Toolkit)
 sobre, 129
 instalação e configuração, 129
 análise lexicográfica, 132 - 136
 análise estatística, 130 - 132
Interface do Downloader NLTK, 130
Módulo NLTK, 129

Nenhum objeto, 10
normalizando dados, 112 - 113
Biblioteca NumPy, 164

O

Autenticação OAuth, 57
OCR (reconhecimento óptico de caracteres)
 sobre, 161
 suporte de biblioteca, 162 - 164
OpenRefine Expression Language (GREL), 116
Ferramenta OpenRefine
 sobre, 114
 limpeza de dados, 116 - 118
 filtrando dados, 115 - 116
 instalando, 114
 considerações de uso, 114
 reconhecimento óptico de caracteres (OCR)
 sobre, 161
 suporte de biblioteca, 162 - 164
Oracle DBMS, 76
Objeto OrderedDict, 112
módulo OS, 74

P

tempos de carregamento da página, 154 , 182
parênteses (), 25
país (tags), 20 , 22
análise de páginas HTML (consulte análise de HTML) análise de JSON, 63
patentes, 217
instâncias de computação de pagamento por hora, 205
Arquivos PDF, 100 - 102
Biblioteca PDFMiner3K, 101
Projeto Penn Treebank, 133
ponto final (), 25
Peters, Tim, 211
Ferramenta PhantomJS, 152 - 155 , 203
PIL (Python Imaging Library), 162
Biblioteca de almofadas
 sobre, 162
 processamento de texto bem formatado, 165 - 169
tubo () , 25
sinal de mais (+), 25
Método POST (HTTP)
 sobre, 51
 pedidos de rastreamento, 140
 solução de problemas, 186
 nomes de variáveis e, 138
 visualizar os parâmetros do formulário, 140

função previous_siblings (), 21
chaves primárias em tabelas, 85
linguagens de programação, expressões regulares e, 27
projeto, agrupamento com bibliotecas, 7
geradores de números pseudo-aleatórios, 34
Método PUT (HTTP), 51
Biblioteca PyMySQL, 82 - 85
Módulo PySocks, 202
Python Imaging Library (PIL), 162
Linguagem Python, instalando, 209 - 211

Q

tempo de consulta versus tamanho do banco de dados, 86
aspas (""), 17

R

geradores de números aleatórios, 34
sementes aleatórias, 34

limites de taxa

sobre, 52
APIs do Google, 60
API do Twitter, 55

lendo documentos

codificação de documentos, 93
Microsoft Word, 102 - 105
Arquivos PDF, 100
arquivos de texto, 94 - 98

limite de recursão, 38 , 89

redireciona, 44 , 158

Cabeçalho de referência, 179

Site RegexPal, 24

expressões regulares

sobre, 22 - 27
Exemplo de BeautifulSoup, 27
símbolos comumente usados, 25
linguagens de programação e, 27

dados relacionais, 77

hospedagem remota

executando a partir de uma conta de hospedagem de site, 203

correndo da nuvem, 204

servidores remotos

evitando o bloqueio do endereço IP, 199 - 200
extensibilidade e, 200
portabilidade e, 200
PySocks e, 202
Tor e, 201 - 202

Biblioteca de solicitações

sobre, 137
módulo de autenticação, 144
instalando, 138 , 179
envio de formulários, 138
cookies de rastreamento, 142 - 143
módulo de solicitações, 179 - 181
respostas, chamadas de API e, 52
Padrão de exclusão de robôs, 223
arquivo robots.txt, 138 , 167 , 222 - 225 , 229

S

proteção de porto seguro, 219 , 230
Biblioteca Scrapy, 45 - 48
screenshots, 197
tag de script, 147

otimização de mecanismo de pesquisa (SEO), 222
pesquisando dados de texto, 135

considerações de segurança

lei de direitos autorais e, 219
formulários e, 183 - 186

manipulação de cookies, 181

Instrução SELECT, 79 , 81

Biblioteca Selenium

sobre, 143
elementos e, 153 , 194
executando JavaScript, 152 - 156
lidar com redirecionamentos, 158
considerações de segurança, 185
exemplo de teste, 193 - 198

Supporte Tor, 203

ponto e vírgula (;), 210

SEO (otimização de mecanismos de pesquisa), 222

processamento do lado do servidor

lidar com redirecionamentos, 44 , 158

linguagens de script e, 147

conjuntos, 67

irmãos (tags), 21

Protocolo de transferência de correio simples (SMTP), 90

mapas do site, 36

Seis graus da Wikipedia, 31 - 35

SMTP (protocolo de transferência de correio simples), 90

pacote smtplib, 90

função classificada, 112

tag span, 15

Spitzer, Daniel, 227

SQL Server (Microsoft), 76

colchetes [], 25

atributo src, 28 , 72 , 74

StaleElementReferenceException, 158

análise estatística com NLTK, 130 - 132
armazenamento de dados (ver gerenciamento de dados) objeto StringIO, 99
strings, expressões regulares e, 22 - 28
folhas de estilo
sobre, 14 , 216
HTML dinâmico e, 151
campos ocultos e, 184
Surface Web, 36

T

mesas
criando em bancos de dados, 80
inserir dados em, 81
chaves primárias e, 85
Exemplo da Wikipedia, 88
Objeto de tag, 18
Tag
acessando atributos, 14 , 28
encontrar com base na localização no documento,
18 - 22
encontrar com base no nome e atributo, 15 - 18
preservando, 15
Termos de serviço, 222 - 225
Biblioteca Tesseract
sobre, 163
instalando, 163
processamento de texto bem formatado, 164 - 169
Treinamento, 171 - 174
Ferramenta Tesseract OCR Chopper, 171
testando
sobre, 189
Exemplo de selênio, 193 - 198
tests de unidade, 190 , 197
módulo unittest, 190 , 197
Exemplo da Wikipedia, 191 - 193
Objeto de texto, 130
processamento de texto
tradução de imagem para texto, 161 - 176
lendo arquivos de texto, 94 - 98
raspar texto de imagens, 166 - 169
pesquisando dados de texto, 135
strings e expressões regulares, 22 - 28
texto bem formatado, 164 - 169
The Onion Network (Tor), 201 - 202
filtros de limite, 165
timestamps, 87
tokens, 52 , 58
Tor (The Onion Network), 201 - 202

marcas registradas, 218
atravessando a navegação em árvore da Web (consulte
os rastreadores da Web), 18 - 22
invasão de bens móveis, 219 - 220 , 226
módulo trigrama, 132
tente ... declaração finalmente, 85
Aplicativo Twitov, 123
API do Twitter, 55 - 59

você

sublinhado (_), 17
problemas de gráficos não direcionados, 127
Padrão Unicode, 83 , 95 - 98 , 110
tests de unidade, 190 , 197
Estados Unidos v. Auernheimer, 227 - 229
módulo unittest, 190 , 197
Declaração UPDATE, 82
biblioteca urllib, 5 , 45
módulo urllib.error, 5
módulo urllib.parse, 5
módulo urllib.request, 5 , 72
biblioteca urllib2, 5
função urlopen, 5 , 97
função urlretrieve, 72
Instrução USE, 80
Cabeçalho do agente do usuário, 179
Padrões UTF, 95 , 110

V

variáveis
meio Ambiente, 163
manipulação em JavaScript, 148
expressões lambda e, 28
versões, múltiplas, 45
ambientes virtuais, 7

W

Diretor, Pete, 217
rastreadores da web
sobre, 31
cuidados usando, 41
rastreando pela Internet, 40 - 45
rastreando um site inteiro, 35 - 40
rastejando com Scrapy, 45 - 48
atravessando um único domínio, 31 - 35
considerações de uso, 220
Raspagem da web, viii - ix
WebDriver, 153 - 155 , 181

- sites**
- analisando por raspagem, [216](#)
 - rastejando inteiro, [35 - 40](#)
 - executando a partir de contas de hospedagem, [203](#)
 - raspar texto de imagens em, [166 - 169](#)
 - teste com raspadores, [189 - 198](#)
 - texto bem formatado, [164 - 169](#)
 - espaço em branco, [74 , 210](#)
- Wikipedia**
- limpando dados sujos, [109 - 112](#)
 - Exemplo de modelo de Markov, [126 - 129](#)
 - Exemplo de MySQL, [87 - 89](#)
- x**
- XML (eXtensible Markup Language), [52](#)
 - XPath (caminho XML), [157](#)

Sobre o autor

Ryan Mitchell é engenheira de software na LinkeDrive em Boston, onde desenvolve a API da empresa e ferramentas de análise de dados. Ela é formada pelo Olin College of Engineering e é estudante de mestrado na Harvard University School of Extension Studies. Antes de ingressar na LinkeDrive, ela construiu web scrapers e bots na Abine Inc. e presta consultoria regularmente em projetos de web scraping, principalmente nos setores financeiro e de varejo.

Colofão

O animal na capa de *Web Scraping com Python* é um pangolim moído (*Smutsia temminckii*). O pangolim é um mamífero solitário e noturno, intimamente relacionado aos tatus, preguiças e tamanduás. Eles podem ser encontrados no sul e no leste da África. Existem três outras espécies de pangolins na África e todas são consideradas em perigo crítico.

Os pangolins terrestres crescidos podem ter em média 12-39 polegadas de comprimento e pesar entre apenas 3,5-73 libras. Eles se assemelham a tatus, cobertos por escamas protetoras que são escuras, marrom claro ou verde-oliva. As escamas imaturas do pangolim são mais rosadas. Quando ameaçadas, as escamas da cauda podem atuar mais como uma arma ofensiva, pois são capazes de cortar e ferir os atacantes. O pangolim também tem uma estratégia de defesa semelhante à dos gambás, em que secretam um ácido fedorento de glândulas localizadas próximas ao ânus. Isso serve como um alerta para possíveis invasores, mas também ajuda o pangolim a marcar território. A parte de baixo do pangolim não é coberta por escamas, mas por pequenos pedaços de pêlo.

Como seus parentes tamanduás, as dietas do pangolim consistem em formigas e cupins. Suas línguas incrivelmente longas permitem que eles vasculhem toras e formigueiros para suas refeições. A língua é mais longa que o corpo e se retrai para a cavidade torácica durante o repouso.

Embora sejam animais solitários, uma vez amadurecidos, o pangolim terrestre vive em grandes tocas que ficam no subsolo. Em muitos casos, as tocas pertencem a porcos-da-terra e javalis, e o pangolim simplesmente assumiu a residência abandonada. Com as três garras longas e curvas encontradas em seus pés dianteiros, no entanto, os pangolins não têm problemas em cavar suas próprias tocas, se necessário.

Muitos dos animais nas capas da O'Reilly estão em perigo; todos eles são importantes para o mundo. Para saber mais sobre como você pode ajudar, vá para animals.oreilly.com.

A imagem da capa é de *História Natural Real de Lydekker*. As fontes de capa são URW Typewriter e Guardian Sans. A fonte do texto é Adobe Minion Pro; a fonte do título é Adobe Myriad Condensed; e a fonte do código é Ubuntu Mono da Dalton Maag.