Web Scrapping

Rafael Alves Ribeiro

20 de Julho de 2018

Conteúdo

Capítulo 1

Introdução

1.1 O que é Web Scraping?

Web Scraping, ou raspagem de dados, consiste em um processo que se utiliza de técnicas de programação para a coleta automatizada de dados provenientes de Web Site.

O objetivo do Web Scraping é a extração de grandes volumes de dados não estruturados, normalmente em formato HTML, de serviços ou aplicativos que não oferecem uma interface de programação padrão(API), estruturando estes dados para que possam ser armazenados em bancos de dados ou em formatos padrão como CSV, XML ou JSON.

Neste processo, um software denominado *scraper* simula a interação realizada entre um browser operado por um humano e o Web Site, extraindo os dados de interesse, para tal, o *scraper* deverá executar as seguintes funções:

- **Acesso ao Web Site:** O scrapper estabelece comunicação com o Web Site através do protocolo HTTP, que cordena as transaões de requisição-resposta entre o scrapper e o Web Site. Os métodos mais utilizados para requisições HTTP são GET, utilizado em solicitações de dados, e POST, para o envio de dados como formulários e upload de arquivos.

- **Análise de HTML(parsing) e extração do conteúdo:** Uma vez que o conteúdo HTML é recebido através da requisição HTTP, o scraper pode extrair os dados de interesse. Para isto, pode utilizar expressões regulares, linguagens de consulta como XPATH ou CSS-selectors ou ainda bibliotecas/pacotes próprios para análise de HTML. Neste curso, utilizaremos o pacote Python chamado BeautifulSoup, para o qual dedicaremos um capítulo exclusivo.

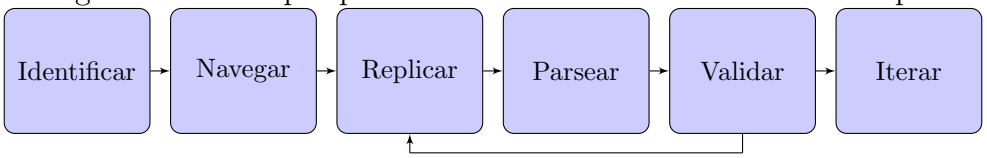
- **Estruturação dos resultados:** Por fim, o scraper deverá transformar o conteúdo extraído em uma representação estruturada que seja apropriada para análise e armazenamento.

De dados governamentais à resultados esportivos, é enorme a quantidade de dados disponíveis que podem ser extraídos, estruturados e utilizados para encontrar respostas para os mais diversos tipo de problemas.

1.2 Desenvolvendo um Scraper

O processo de desenvolvimento de um scraper pode ser dividido em 6 etapas (identificar, navegar, replicar, parsear, validar, iterar) organizadas conforme a abaixo:

Figura 1.1: 6 Etapas para o desenvolvimento de um Web Scraper

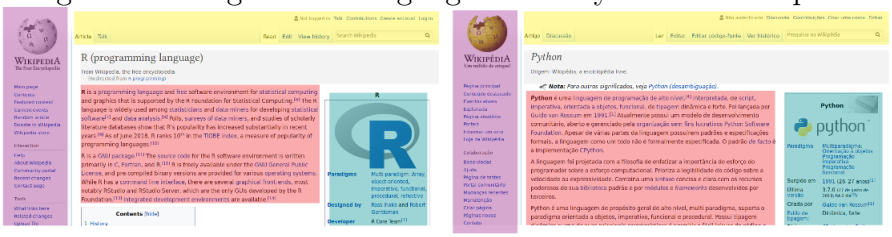


1.2.1 Identificar os padrões

No primeiro passo do processo de desenvolvimento de um scraper precisamos entender qual é a estrutura das páginas que queremos raspar e traçar um plano para extrair tudo que precisamos.

Na imagem abaixo temos o screenshot de duas páginas distintas da Wikipedia lado a lado, com marcações coloridas que evidenciam o padrão de organização dos elementos no site.

Figura 1.2: Páginas das linguagens R e Python na Wikipedia



Fica claro que a Wikipedia segue um padrão bem definido, e que provavelmente poderemos utilizar um mesmo scraper para extrair as informações de ambas as páginas. Como nem todos os web sites são bem estruturados como em nosso exemplo, essa análise inicial é importante para que tenhamos noção da complexidade do scraper que precisaremos contruir para obter as informações desejadas.

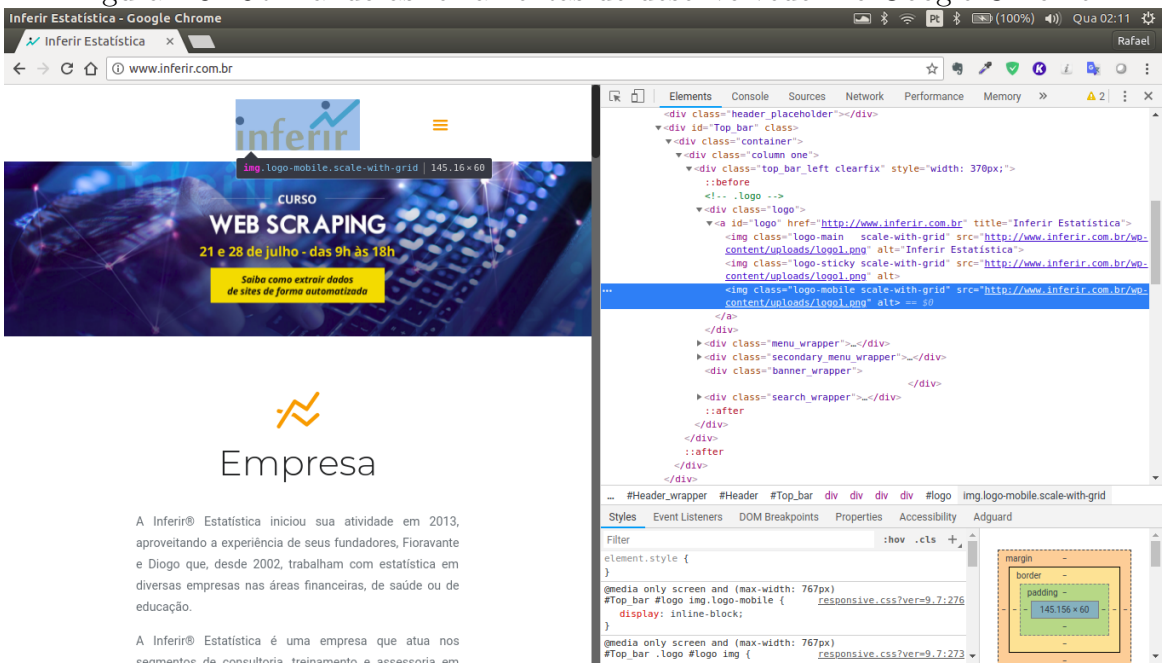
1.2.2 Navegar

Agora precisamos entender como localizar o dado que queremos extrair dentro do HTML da página. Esse passo pode ser extremamente simples, mas de vez em quando ele se tornará algo bastante complexo.

Usando as ferramentas de desenvolvedor do nosso navegador (tecla F12 ou clique com o botão esquerdo do mouse sobre um elemento da página e selecionar a opção Inspecionar) vamos navegar para encontrar a fonte dos dados.

No exemplo abaixo encontramos a logomarca da Inferir no código HTML do site. Podemos observar que a imagem está contida em uma tag do tipo img, que pertence à classe "logo-mobile scale-with-grid"e que a URL da imagem é `http://www.inferir.com.br/wp-content/uploads/logo1.png`. Todas estas informações poderiam ser utilizadas para construírmos um web scraper que buscasse esta imagem.

Figura 1.3: Utilizando as ferramentas de desenvolvedor no Google Chrome

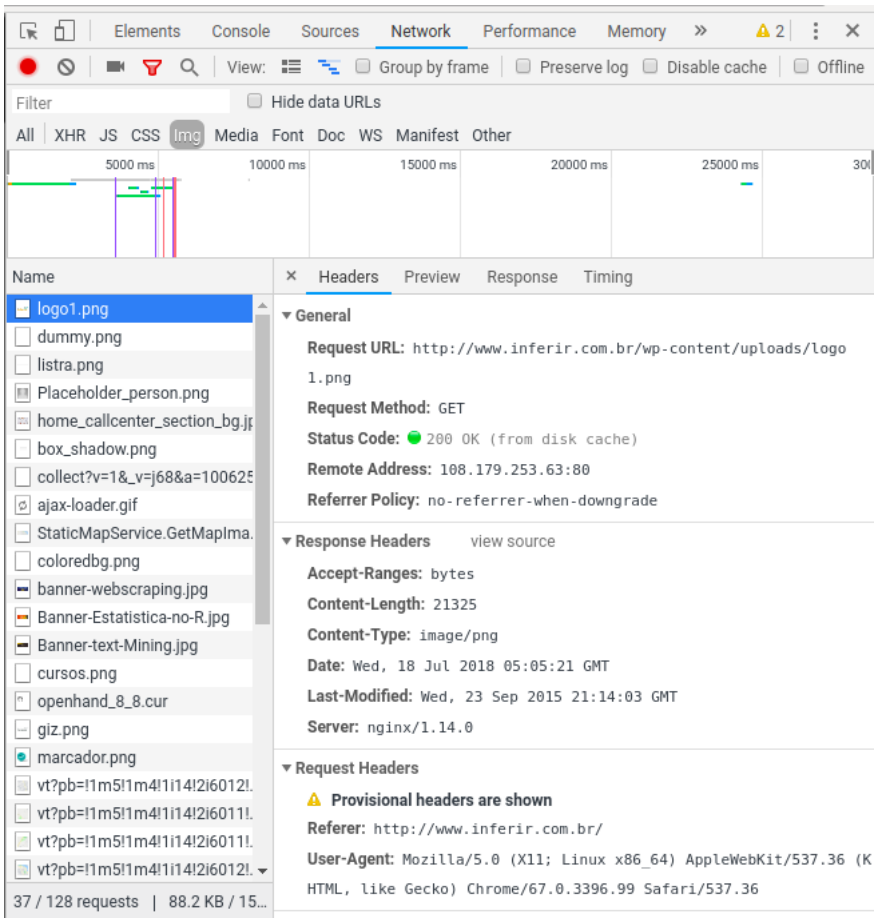


Com as ferramentas de desenvolvedor poderíamos ainda analisar o networking do navegador para entender as chamadas HTTP que são feitas, estudar os resultados das funções JavaScript invocadas pela página e assim por diante. Abordaremos o uso desta ferramento em profundidade próximos capítulos.

1.2.3 Replicar

Se tivéssemos que fazer várias requisições HTTP para chegar até a informação que queremos, seria aqui em que tentaríamos replicar essas chamadas. Neste passo é importante compreender absolutamente tudo que a página está fazendo para trazer o conteúdo até você, então é necessário analisar o seu networking a fim de entender tais requests e seus respectivos parâmetros.

Figura 1.4: Utilizando a Ferramenta de Desenvolvedor para verificar a requisição HTTP referente à logomarca da Inferir



No exemplo acima, verificamos que para obter a logomarca da Inferir o browser realizou uma requisição HTTP da URL `http://www.inferir.com.br/wp-content/uploads/logo1.png` utilizando o método GET.

1.2.4 Parsear

O anglicismo parsear vem do verbo *to parse*, que quer dizer algo como analisar ou estudar, mas que, no contexto do Web Scraping, significa extrair os dados desejados de um arquivo HTML.

Vamos utilizar as informações obtidas nos passos anteriores para extrair o código HTML da logomarca Inferir utilizando um script Python.

```
from bs4 import BeautifulSoup
import requests

# enviamos uma requisão HTTP utilizando o método GET
response = requests.get('http://www.inferir.com.br')
html = response.text

# realizamos o parse do HTML utilizando BeautifulSoup
soup = BeautifulSoup(html, 'html.parser')

# buscamos a imagem utilizando a tag e a classe
# encontradas na Ferramenta de Desenvolvedor
classe_logo = 'logo-mobile scale-with-grid'
logo_inferir = soup.find('img', {'class': classe_logo})
print(logo_inferir)
```

O resultado do script acima imprimirá na tela o código HTML da imagem:

```
<img alt="" class="logo-mobile scale-with-grid"
 ↪  src="http://www.inferir.com.br/wp-content/uploads/logo1.png"/>
```

1.2.5 Validar

Se tivermos feito tudo certo até agora, validar os resultados será uma tarefa simples. Precisamos apenas reproduzir o procedimento descrito até agora para algumas outras páginas de modo verificar se estamos de fato extraindo corretamente tudo o que queremos.

Caso encontremos algo de errado precisamos voltar ao passo 3 para tentar replicar corretamente o comportamento do site e parsear os dados certos nas páginas.

1.2.6 Iterar

O último passo consiste em colocar o nosso scraper em produção. Aqui, ele já deve estar funcionando corretamente para todos os casos desejados e estar pronto para raspar todos os dados dos quais precisamos.

Na maior parte dos casos isso consiste em encapsular o scraper em uma função que recebe uma série de links e aplica o mesmo procedimento em cada um. Se quisermos aumentar a eficiência desse processo, podemos paralelizar ou distribuir o nosso raspador.

Capítulo 2

Requests

2.1 Requests: HTTP for Humans

Requests é um pacote Python que permite o acesso à serviços web, sem a necessidade de conhecimento avançados sobre o protocolo de comunicação HTTP. Empresas como Amazon, Google, Mozilla, PayPal, The Washington Post e Twitter utilizam Requests, que é um dos pacotes Python mais baixados de todos os tempos, com mais de 11 milhões de downloads mensais.

Para instalar o Requests, utilize o commando pip install no terminal de comando:

```
pip install requests
```

2.2 Solicitando informações a um servidor web

Um dos destaques do pacote é a simplicidade para obter as informações. No exemplo abaixo, fazemos o download das informações da Base de Dados do Comércio Exterior Brasileiro disponibilizadas no site do Ministério da Indústria, Comércio Exterior e Serviços e imprimimos na tela a primeira linha do arquivo csv.

```
>>> import requests
>>>
>>> url = "http://www.mdic.gov.br/balanca/bd/ncm/EXP_2018.csv"
>>> # envia uma requisição HTTP utilizando o método GET
>>> response = requests.get(url)
>>> # acessa o texto da resposta enviada pelo servidor,
>>> # seleciona a primeira linha e imprime na tela
>>> print(response.text.splitlines()[0])
"CO\_UNID";"CO\_PAIS";"CO\_UF";"CO\_PORTO";"CO\_VIA";"QT\_ESTAT";
"KG\_LIQUIDO";"VL\_FOB"''
```

A saída reproduzida corresponde à primeira linha do arquivo csv.

2.3 Manipulando a resposta do servidor

Após receber e iterpretar a requisição enviada, o servidor web retorna uma resposta HTTP contendo diversas informações úteis. O Requests realiza o tratamento destes dados e os armazena em um objeto chamado Response. Em nosso exemplo, utilizamos apenas o texto da resposta. Vamo realizar uma nova requisição, desta vez solicitando uma imagem, e explorar as informações contidas no objeto Response.

```
>>> import requests
>>>
>>> url = "https://www.python.org/static/community\_logos/python-logo.png"
>>> response = requests.get(url)
>>> print(response)
<Response [200]>
```

O códido de status da resposta enviada pelo servidor é 200, o que quer dizer que obtivemos sucesso em nossa requisição. Caso a URL não existisse no servidor receberíamos o status 404. A lista completa dos status possíveis pode ser encontrada em `https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`.

```
>>> print(response.status_code)
200
```

Podemos também acessar o cabeçalho da resposta enviada pelor servidor, que pode ser obtido atraver da propriedade headers. O cabeçalho é um objeto do tipo dicionário (dict) e contém diversas informações sobre o conteúdo da resposta, o servidor e a requisição.

```
>>> print(response.headers)
{'Server': 'nginx', 'Content-Type': 'image/png', 'Last-Modified': 'Wed, 21
    ↪  Oct 2015 03:51:54 GMT', 'ETag': '"56270bda-b083"', 'Cache-Control':
    ↪  'max-age=604800, public', 'X-Clacks-Overhead': 'GNU Terry Pratchett',
    ↪  'Via': '1.1 varnish, 1.1 varnish', 'Content-Length': '45187',
    ↪  'Accept-Ranges': 'bytes', 'Date': 'Tue, 17 Jul 2018 03:05:49 GMT', 'Age':
    ↪  '439970', 'Connection': 'keep-alive', 'X-Served-By': 'cache-iad2132-IAD,
    ↪  cache-gru17121-GRU', 'X-Cache': 'HIT, HIT', 'X-Cache-Hits': '1, 1',
    ↪  'X-Timer': 'S1531796749.401601,VS0,VE2', 'Strict-Transport-Security':
    ↪  'max-age=63072000; includeSubDomains'}
```

O campo Content-Type do cabeçalho nos mostra que o servidor retornou uma imagem no formato png, cujo tamanho é 45.187 bytes de acordo com o campo Content-Length. Agora que confirmamos estas informações, podemos salvar a imagem em um arquivo, utilizando o conteúdo binário da resposta obtido através da propriedade content:

```
>>> arquivo_destino = 'logo-python.png'
>>> with open(arquivo_destino, 'w') as pngfile:
...     pngfile.write(r.content)
```

Ao abrirmos o arquivo que criamos veremos logotipo do Python:

Figura 2.1: Logotipo do Python

Para facilitar a reprodução deste processo podemos definir uma função no Python para salvar o conteúdo de uma URL em um determinado arquivo.

```
1  import requests
2
3  def download_url(url, nome_arquivo):
4      ''' Acessa a url utilizando o método HTTP GET, verifica o tipo de
   ↪   conteúdo e armazena no arquivo passado como parâmetro '''
5
6      response = requests.get(url)
7      response.raise_for_status()
8
9      content_type = response.headers.get('Content-Type', '')
10     if content_type.startswith('text'):
11         mode = 'w'  # escrita de arquivo texto
12     else:
13         mode = 'wb'  # escrita de arquivo binário
14
15     with open(nome_arquivo, mode) as f:
16         f.write(response.content)
17
18     msg = ('Download do conteúdo {} da url {} armazenado'
19            'com sucesso no arquivo {}.')
20     print(msg.format(content_type, url, nome_arquivo))
21
22     return response
```

2.4 Enviando informações para o servidor

O método POST é uma das principais formas de envio de informações para um servidor web. Com ele podemos enviar dados de formulários, realizar logins e upload de arquivos.

2.4.1 Utilizando Sessions

O objeto *Session* possibilita persistir dados entre as requisições. Esta funcionalidade é necessária em sites que utilizam cookies para gerenciar estado da sessão do usuário. Para criar uma *Session* basta instanciar o objeto:

```
>>> import requests
>>>
>>> session = requests.Session()
>>> session
<requests.sessions.Session object at 0x7fbc186205c0>
```

Podemos utilizar o método *get* da conforme apresentado nas seções anteriores:

```
>>> response = session.get('http://www.google.com')
>>> response
<Response [200]>
```

2.5 Acessando APIs

Capítulo 3

BeautifulSoup

3.1 Introdução ao BeautifulSoup

Publicado pela primeira vez em 2004, o BeautifulSoup é um pacote Python para extração de informações em arquivos HTML e XML. Funciona com diferentes parsers como lxml e html5lib, oferecendo uma linguagem clara e intuitiva para navegar, buscar e modificar a árvore de análise. Outra funcionalidade interessante é a detecção e conversão automática da codificação dos documentos analisados, reduzindo o tempo de desenvolvimento e a quantidade de erros.

Como curiosidade, vale dizer que o nome BeautifulSoup foi retirado de um poema de Lewis Carrol encontrado no livro Alice no País das Maravilhas:

Beautiful Soup, so rich and green,	Linda Sopa, tão rica e verdinha
Waiting in a hot tureen!	Esperando em uma tigela quente!
Who for such dainties would not stoop?	Quem não se entregaria a tamanha iguaria?
Soup of the evening, beautiful Soup!	Sopa da noite, sopa tão linda!
Soup of the evening, beautiful Soup	Sopa da noite, sopa tão linda!

Sob a inspiração non-sense de Lewis Carrol, o BeautifulSoup foi criado para nos ajudar a formatar e organizar a sopa de conteúdos da web, corrigindo HTML inválido e nos apresentado um estrutura de fácil análise.

Para instalar o BeautifulSoup, abra um terminal e utilize o pip:

```
1 pip install beautifulsoup4
```

3.2 Analisando HTML

Para analisar um documento, devemos utilizá-lo como parâmetro no construtor do BeautifulSoup. Podemos utilizar uma string ou um arquivo:

```
>>> from bs4 import BeautifulSoup
>>>
>>> # utilizando um arquivo
>>> with open('index.html') as fp:
>>>     soup = BeautifulSoup(fp)
>>>
>>> # utilizando uma string
>>> soup = BeautifulSoup('<html>data</html>')
```

No restante desta seção utilizaremos o site do Governo Federal (`http://www.brasil.gov.br`) como exemplo. O primeiro passo é obter o HTML da página:

```
>>> import requests
>>>
>>> url = 'http://www.brasil.gov.br'
>>>
>>> user_agent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:61.0)
 ↪  Gecko/20100101 Firefox/61.0'
>>> headers = {'User-Agent': user_agent}
>>> response = requests.get(url, headers=headers)
>>> response
<Response [200]>
>>> HTML = response.text
>>> print(HTML[:100])
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br" xml:lang="pt-br">
<head>
```

Ao passarmos o HTML para o BeautifulSoup recebemos um objeto *BeautifulSoup*, que representa o documento como uma estrutura de dados aninhada:

```
>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(HTML, 'html.parser')
>>> print(soup.prettify())
<!DOCTYPE html>
<html lang="pt-br" xml:lang="pt-br" xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
  <base href="http://www.brasil.gov.br/pagina-inicial/"/>
  <!--[if lt IE 7]></base><![endif]-->
  <link href="http://www.brasil.gov.br/portal_css/Sunburst%20Theme
  /resourceportalbrasilswiper.min-cachekey-ef22025c2b9a48d88369591fb7
  b99e24.css" media="screen" rel="stylesheet" type="text/css"/>
```

Podemos explorar a estrutura utilizando diferentes métodos:

```
>>> soup.title
<title>Página Inicial | Governo do Brasil</title>
>>> soup.title.name
'title'
>>> soup.title.string
'Página Inicial | Governo do Brasil'
>>> soup.title.parent.name
'head'
>>> soup.p
<p class="hiddenStructure">
<a accesskey="2" href="#conteudo">Skip to content.</a> |
    <a accesskey="3" href="#main-navigation">Skip to navigation</a>
</p>
```

```
>>> soup.p['class']
['hiddenStructure']
>>> soup.a
<a href="http://brasil.gov.br" style="font-family:sans,sans-serif;
 ↪  text-decoration:none; color:white;">Portal do Governo Brasileiro</a>
>>> soup.find_all('a')[:3]
[<a href="http://brasil.gov.br" style="font-family:sans,sans-serif;
 ↪  text-decoration:none; color:white;">Portal do Governo Brasileiro</a>, <a
 ↪  href="http://epwg.governoeletronico.gov.br/barra/atualize.html"
 ↪  style="font-family:sans,sans-serif; text-decoration:none;
 ↪  color:white;">Atualize sua Barra de Governo</a>, <a accesskey="2"
 ↪  href="#conteudo">Skip to content.</a>]
>>> soup.find(id='portal-logo')
<a href="http://www.brasil.gov.br" id="portal-logo" title="Portal oficial do
 ↪  governo federal. Acompanhe as notícias e informações sobre as ações,
 ↪  serviços e programas do Poder Executivo Federal.">
<span id="portal-title-1">Governo do</span>
<div class="corto" id="portal-title">Brasil</div>
<span id="portal-description"></span>
</a>
```

Uma tarefa comum é extrair todas as URLs encontradas dentro das tags ¡a¿ de uma pagína:

```
>>> for link in soup.find_all('a'):
...     href = link.get('href')
...     # ignoramos as âncoras internas e tags sem href
...     if href is not None and href.startswith('http'):
...         print(href)
...
http://brasil.gov.br
http://epwg.governoeletronico.gov.br/barra/atualize.html
http://www.brasil.gov.br
http://www.vlibras.gov.br/
http://www.brasil.gov.br/editoria
http://www.brasil.gov.br/editoria/ultimas-noticias
```

Outra tarefa comum é extrair todo o texto de um elemento:

```
>>> soup.find(id='content').get_text()
'\n\n\n\n\n\n\n\nFies\nTire dúvidas sobre troca de
↪ dados\n\n\n\n\n\n\n\nVacina contra gripe\nMais de 90% do público-alvo
↪ está imunizado\n\n\n\n\n\nJustiça Eleitoral\nCadastro de eleitores em
↪ trânsito é aberto\n\n\n\n\n\n\n\n\n\nSUS\nAvisos de consulta virão pelo
↪ celular\n\n\n\n\n\n\n\n\nCadastro Ambiental Rural\nInscrição de
↪ terrenos deve ser feita até dezembro\n\n\n\n\n\n\n\n\n\nAlimentação
↪ saud
```

Capítulo 4

Selenium

Capítulo 5

Informações em diversos formatos

Capítulo 6

Boas práticas em projetos de Web Scrapping