Web scraping com Python para preguiçosos (unindo BeautifulSoup e Selenium) — Parte 1.

Iniciando no BeautifulSoup

Caio Estrella Follow
Apr 25, 2020 · 8 min read

> *"Eu escolho uma pessoa preguiçosa para fazer um trabalho duro. Porque uma pessoa preguiçosa irá encontrar uma forma fácil de o fazer." — Bill Gates*

Encontrar o melhor equilíbrio entre custo e benefício é questão de saber identificar a maneira mais simples de se resolver um problema. Claro que alguns desafios demandam mais estudo e desenvolvimento para a sua resolução. Às vezes não podemos nos dar ao luxo de sermos preguiçosos pois nem sempre o **simples** é o mais **adequado**. Mas em alguns casos, o prego e martelo da sua caixa de ferramentas já resolve muitos dos seus problemas de maneira elegante e pouco custosa.

Se você chegou nesse post provavelmente já sabe do que se trata web scraping (**caso esteja interessado em ver somente a demonstração, pule para a seção"mãos à obra"**). Mas vale a pena tentar uma definição clara sem muitas firulas. Web scraping é coletar dados da internet de **forma automática**. O que você cria para executar essa ação é um "bot" ou "web crawler" (ou simplesmente um "robô") e então você armazena esses dados, geralmente, em uma tabela e faz o que quiser com eles depois. É isso.

Aqui vamos presumir que você também já possui algum entendimento de programação, Python e Pandas. Estude essas matérias antes caso precise. Não seja **tão** preguiçoso 🐌. Mas caso não conheça, tudo isso serve para saber que no final teremos uma tabela clássica, com linhas e colunas, e vamos salva-la em um arquivo excel.

Você também precisa ter algum conhecimento de HTML, pois o que o seu robô vai fazer é ler a página web localizando tags para executar alguma ação ou coletar dados. No futuro você pode querer aprender um pouco de JSON e JavaScript e até ASP.NET para executar algumas ações… **Mas calma.** Esse ainda é um post para preguiçosos , então vamos com o básico somente. Quem sabe no futuro sai o "*Web Scraping para laboriosos*" 🤓.

Claro que esse post sozinho não responderá todas as duvidas que você possa ter. **Mas não deixe de ler as indicações no fim do post.**

Conhecendo o seu problema

Essa é a primeira parte onde vamos apresentar o BeautifulSoup e como ele pode resolver alguns desafios, enquanto outros serão apresentados em futuros textos. Então vamos separar os problemas em dois grupos:

Problemas tratados neste post:

- As informações estão em uma página que fornece seus dados na tela, sem nenhum arquivo de dados.
- Devemos transitar por diferentes URLs a fim de coletar os dados.
- Precisamos colocar esses dados em uma tabela (ou DataFrame) e então salva-la em um arquivo.

Problemas tratados em posts futuros:

- Precisaremos preencher um formulário para gerar os dados. Sempre de maneira automática.
- Não somos mais redirecionados para uma nova URL. A tela é atualizada, na mesma URL com os dados que pedimos acesso.
- Usando Selenium, precisaremos simular um usuário clicando e digitando, mas isso deverá ser feito **sem que o browser abra.**

Conhecendo as suas ferramentas.

BeautifulSoup:

É uma biblioteca que ajuda a organizar a estrutura HTML de uma página web em objetos que podem ser inspecionados pelo Python. Atua na manipulação de **páginas estáticas,** ou seja, ignorando JavaScript. É o foco deste post.

Selenium:

Acho que a melhor definição é a que existe em sua página: “O Selenium automatiza browsers. É isso! O que fazer com esse poder cabe a você.”. (https://www.selenium.dev). Veremos no futuro como ele nos será bastante útil na manipulação de **páginas dinâmicas**.

Mãos à obra

Instalação

No **Windows**, já com o Python3 instalado, abra uma prompt de comando e digite `pip install beautifulsoup4` . Em **Linux Debian ou Ubuntu** `sudo apt-get install python3-bs4` . Para **Mac** `sudo easy_install pip` e então `pip install beautifulsoup4`

Lendo a página

Usando ou um Jupyter Notebook ou uma IDE da sua preferência digite:

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

html = urlopen("http://www.pagina_exemplo.com")
bs = BeautifulSoup(html, 'html.parser')
```

Em `html` você terá basicamente toda a estrutura HTML da página que foi incluida em `urlopen()` (Duvida? digite `print(bs)` ou melhor ainda `print(bs.prettify())` 😱. Com `bs` começaremos a inspecionar os elementos, seus atributos, extrair textos incluídos entre as tags entre outras coisas.

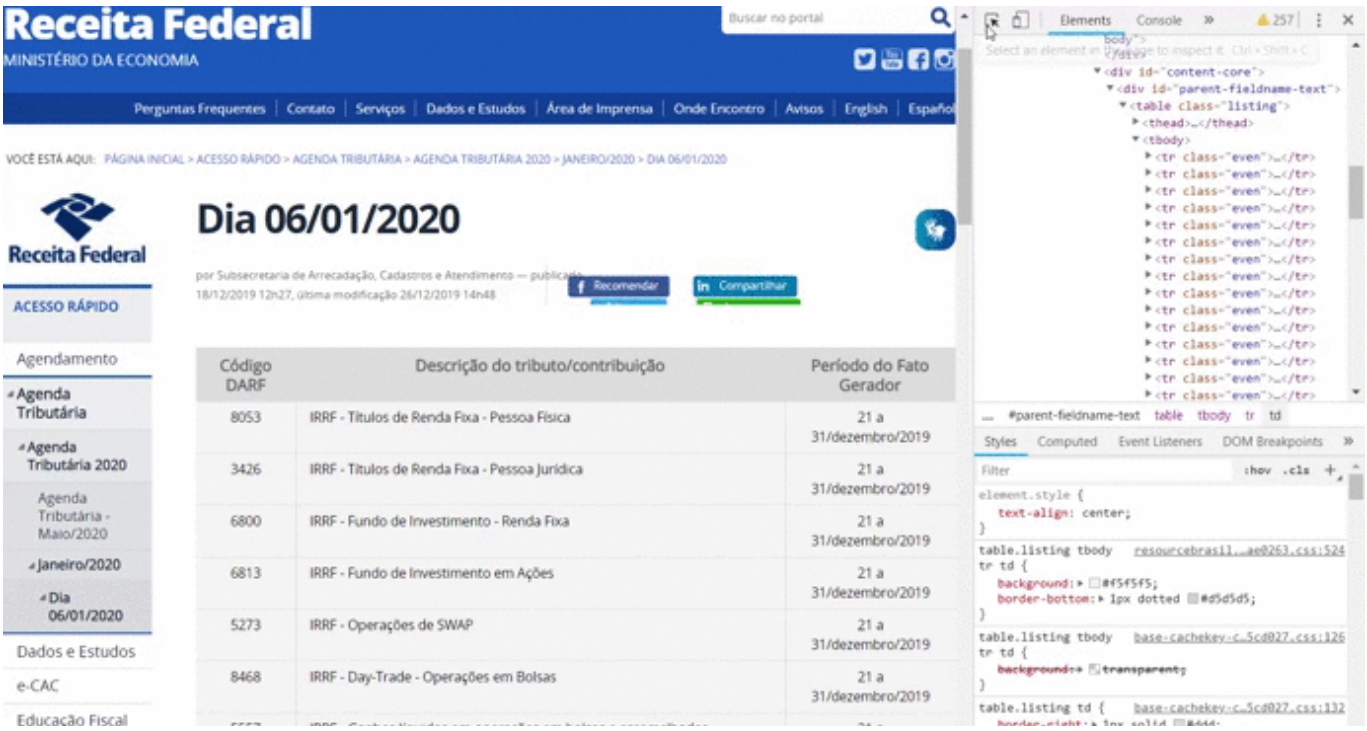
Lembremos que **não temos aqui uma página que gera conteúdo dinamicamente ao clicarmos em algum link.** Faça um teste e repare se a barra de URL muda quando você clica em algo. Enquanto estivermos transitando por diferentes URLs para coletar os dados, o BeautifulSoup dará conta do recado sozinho (Bem, na verdade, é possível lidar com uma página dinâmica só com o BS. Mas não seria nada elegante e bem mais trabalhoso).

Investigando uma página

Vamos supor que após aplicar alguma lógica para acessar um link de maneira automática tenhamos parado em uma página de nosso interesse. Usaremos no lugar da página_exemplo (**do código anterior**) a URL (https://receita.economia.gov.br/acesso-rapido/agenda-tributaria/agenda-tributaria-2020/agenda-tributaria-janeiro-2020/agenda-tributaria-janeiro-2020) da Receita Federal que nos fornece uma tabela com alguns dados públicos na tela.

Aperte F12 e abrirá o modo de inspeção de página. Para inspecionar os elementos da página ao passar o mouse, clique no ícone no canto superior esquerdo da caixa ou digite CTRL+Shift+C.

Em um rápido estudo de como a tabela está organizada, podemos ver que existe uma estrutura <tbody> e que cada linha da tabela está em um <tr class = 'even'> conforme o GIF abaixo.



Nesse ponto existem diversas maneiras de se atacar o problema. O leitor poderia descobrir uma solução mais **preguiçosa**, inclusive.

Seria bom se pudéssemos acessar todos os elementos <tr> que possuem a classe 'even', e com o BS fica fácil:

```
linhas = bs.find_all('tr', {'class':'even'})
```

Aqui temos uma lista com todos os elementos e a armazenamos na variável 'linhas'. Inspecionando mais a fundo na página, percebemos que em cada linha existem três filhas, que são as tags <td> que representam as colunas. Reparem em negrito o texto contido em cada uma:

```
<tr class="even">
 <td style="text-align: center; ">
  8053
 </td>
 <td style="text-align: justify; ">
  IRRF – Títulos de Renda Fixa – Pessoa Física
 </td>
 <td style="text-align: center; ">
  21 a 31/dezembro/2019
 </td>
</tr>
```

O nosso objetivo é coletar esse texto. Mas antes, quero testar se realmente todos os elementos que queríamos extrair estão aqui mesmo. Portanto vou imprimir qualquer texto que exista em cada uma das 'linhas'. Existem maneiras diferentes de se fazer isso:

```
## Imprime todo texto contido em cada linha ##
for i in linhas:
    print(i.text)

## Imprime o texto de cada uma das tags filhas ##
for i in linhas:
    filhas = i.findChildren("td")
```

```
print(filhas[0])
print(filhas[1])
print(filhas[2])
```

A segunda maneira utiliza `findChildren("td")` para armazenar em `filhas` todas as tags <td> que são filhas de <tr class = 'even'>, e já sabemos que existem três por linha. Isso é interessante pois cada uma dessas 'filhas' deve estar em uma coluna diferente da tabela que vamos gerar, e essa abordagem nos permite manipular cada uma das colunas individualmente.

Confirmando que o seu código está imprimindo os valores corretamente, agora vamos adicioná-los à listas distintas.

```
codigo, descricao, periodo = [], [], []

for i in linhas:
    children = i.findChildren("td")
    codigo.append(children[0].text)
    descricao.append(children[1].text)
    periodo.append(children[2].text)
```

Agora que já temos as nossas futuras colunas em listas, é só criar a nossa tabela no Pandas:

```
import pandas as pd

df = pd.DataFrame({'Código': codigo, 'Descrição': descricao,
'Período': periodo})

df.head()
```

	Código	Descrição	Período
0	8053	IRRF - Títulos de Renda Fixa - Pessoa Física	21 a 31/dezembro/2019
1	3426	IRRF - Títulos de Renda Fixa - Pessoa Jurídica	21 a 31/dezembro/2019
2	6800	IRRF - Fundo de Investimento - Renda Fixa	21 a 31/dezembro/2019
3	6813	IRRF - Fundo de Investimento em Ações	21 a 31/dezembro/2019

4	5273	IRRF - Operações de SWAP	21 a 31/dezembro/2019

Para salva-la em um arquivo excel : `df.to_excel('caminho_do_arquivo.xlsx')`

A página fornece os dados para um unico dia nesta URL. Para repetir o processo e criar uma tabela para outro dia, basta que se mude o `href` e repita o processo. Repare que tudo isso pode ser feito em um único loop. Conjuntos de dados organizados desta maneira **geralmente** tem só o ultimo trecho das URLs alterada, como aqui, onde começamos em "…agenda-tributaria-janeiro-2020/**dia-06–01–2020**" mas podemos ver outra tabela em "…agenda-tributaria-janeiro-2020/**dia-08–01–2020**"

Não sendo preguiçoso

Observações

As tags e seus atributos em um documento HTML estão organizadas em um estrutura de árvore, conhecida como DOM. Se familiarize com a estrutura da sua página antes de raspar os dados. Comece identificando onde está o dado que você quer e quais são as tags em volta dele; quem são seus “pais”, “filhos” e “irmãos”. Pois essa é a abordagem utilizada pelas funções do BeautifulSoup para você realizar web scraping. Você passará a pensar por exemplo: “Gostaria de acessar a primeira tag irmã de toda tag com classe = x”. E você verá que o BS já tem uma solução simples para esse problema (veja mais na seção ‘Indicações’).

O BS permite que você acesse qualquer tag. Existem diversas maneiras de se fazer isso. Vale a pena estudar algumas das principais funções da biblioteca e testar seu comportamento antes de incluí-lo no trecho de código do seu projeto.

O BeautifulSoup não é difícil. O maior desafio continua sendo a lógica para fazer loops, manipular strings e etc. Você pode se ver tentando contornar um erro achando que é algo no BS quando na verdade é culpa da lógica empregada. Ou, como infelizmente

ocorre, devido ao fato das tags em cada URL dentro do mesmo site não estarem organizadas em um padrão previsível. Ao mudar para uma URL que atualiza os dados, uma tag que você contava estar de um jeito pode estar de outro ou sequer existir.

Por exemplo você pode querer selecionar tags pelo seu atributo, mas depois da enésima iteração o seu loop se depara com uma tag **sem** aquele atributo. O BS simplesmente retornará um erro como `KeyError` ou `AttributeError` e você não saberá em que página ele ocorreu.

Portanto, teste antes a sua função em algo mais simples e direto, e veja como ela se comporta. Considere utilizar `try/except` com frequência em casos onde o erro já é esperado. Também mande imprimir mensagens ao longo do código, para entender em que parte seu loop está e o que está sendo armazenado nas suas variáveis. Isso facilitará a sua vida e reduzirá sua curva de aprendizado.

Indicações

Seguem algumas indicações sobre pontos que podem ser mais bem explorados.

- Protocolo HTTP e métodos GET e POST: http://devfuria.com.br/php/como-funcionam-os-metodos-get-e-post/
- Documentação do BeautifulSoup em português: https://www.crummy.com/software/BeautifulSoup/bs4/doc.ptbr/
- Possivelmente o melhor ajudante para usuários de BeautifulSoup: https://kite.com/python/docs/bs4.BeautifulSoup
  Contém uma lista de funções e diversas perguntas frequentes com respostas.
- Existe um ótimo livro sobre o assunto de leitura fácil: *Web Scraping With Python — Ryan Mitchell.*
- Um ótimo projeto traduzido para português : https://medium.com/machina-sapiens/raspagem-de-dados-com-python-e-beautifulsoup-1b1b7019774c

Web Crawling Python Beautifulsoup Selenium