Beginner's guide to web scraping with python's selenium

Lewis kori Sep 4 '19 *Updated on Jan 30, 2020* · 5 min read

#tutorial #beginners #python

web scraping techniques with python (3 Part Series)

1) Introduction to web scraping with python
2) Beginner's guide to web scraping with python's selenium
3) web scraping: Managing proxies and Captcha with scrapy and the Scrape...

In the first part of this series, we introduced ourselves to the concept of web scraping using two python libraries to achieve this task. Namely, requests and BeautifulSoup. The results were then stored in a JSON file. In this walkthrough, we'll tackle web scraping with a slightly different approach using the selenium python library. We'll then store the results in a CSV file using the pandas library.

The code used in this example is on github.

Why use selenium?

Selenium is a framework which is designed to automate test for web applications.
You can then write a python script to control the browser interactions automatically such as link clicks and form submissions. However, in addition to all this selenium comes in handy when we want to scrape data from javascript generated content from a webpage. That is when the data shows up after many ajax requests. Nonetheless, both

BeautifulSoup and scrapy are perfectly capable of extracting data from a webpage. The choice of library boils down to how the data in that particular webpage is rendered.

Other problems one might encounter while web scraping is the possibility of your IP address being blacklisted. I partnered with scraper API, a startup specializing in strategies that'll ease the worry of your IP address from being blocked while web scraping. They utilize IP rotation so you can avoid detection. Boasting over 20 million IP addresses and unlimited bandwidth.

In addition to this, they provide CAPTCHA handling for you as well as enabling a headless browser so that you'll appear to be a real user and not get detected as a web scraper. For more on its usage, check out my post on web scraping with scrapy. Although you can use it with both BeautifulSoup and selenium.

web scraping: Managing proxies and Captcha with scrapy and the Scraper API

Lewis kori · Oct 7 '19 · 8 min read

#startup #tutorial #javascript #python

Using this link and the code
SCRAPE201818, you'll get a 10% discount off your first purchase!

For additional resources to understand the selenium library and best practices, click here and here.

Setting up

We'll be using two python libraries. selenium and pandas. To install them simply run

```
pip install selenium pandas
```

In addition to this, you'll need a browser driver to simulate browser sessions.
Since I am on chrome, we'll be using that for the walkthrough.

Driver downloads

1. Chrome.
2. Firefox gecko driver

Getting started

For this example, we'll be extracting data from quotes to scrape which is specifically made to practise web scraping on.
We'll then extract all the quotes and their authors and store them in a CSV file.

```
from selenium.webdriver import Chrome
import pandas as pd

webdriver = "path_to_installed_driver_location"

driver = Chrome(webdriver)
```

The code above is an import of the chrome driver and pandas libraries.
We then make an instance of chrome by using

```
driver = Chrome(webdriver)
```

Note that the webdriver variable will point to the driver executable we downloaded previously for our browser of choice. If you happen to prefer firefox, import like so

```
from selenium.webdriver import Firefox
```

Main script

```
pages = 10

for page in range(1,pages):

    url = "http://quotes.toscrape.com/js/page/" + str(page) + "/"

    driver.get(url)

    items = len(driver.find_elements_by_class_name("quote"))

    total = []
    for item in range(items):
        quotes = driver.find_elements_by_class_name("quote")
        for quote in quotes:
            quote_text = quote.find_element_by_class_name('text').text
            author = quote.find_element_by_class_name('author').text
            new = ((quote_text,author))
            total.append(new)
    df = pd.DataFrame(total,columns=['quote','author'])
    df.to_csv('quoted.csv')
driver.close()
```

On close inspection of the sites URL, we'll notice that the pagination URL is

```
Http://quotes.toscrape.com/js/page/{{current_page_number}}/
```

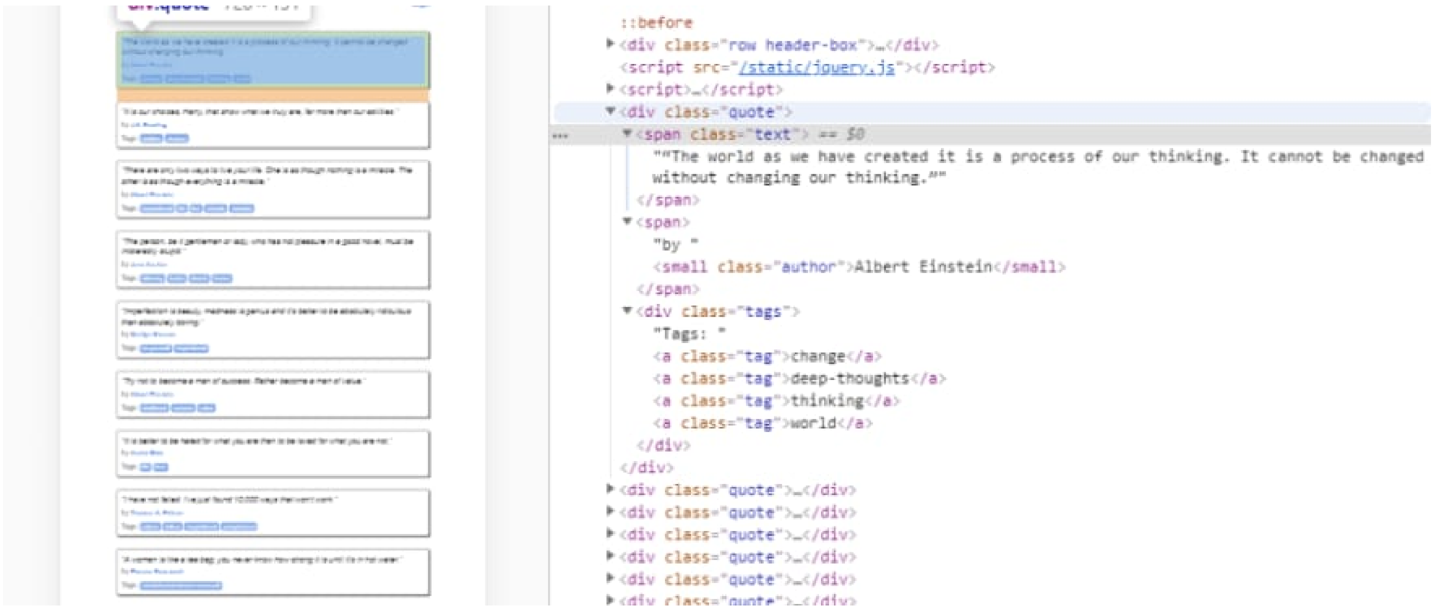
where the last part is the current page number. Armed with this information, we can proceed to make a page variable to store the exact number of web pages to scrape data from. In this instance, we'll be extracting data from just 10 web pages in an iterative manner.

The

```
driver.get(url)
```

command makes an HTTP get request to our desired webpage.
From here, it's important to know the exact number of items to extract from the webpage.
From our previous walkthrough, we defined web scraping as

> This is the process of extracting information from a webpage by taking advantage of patterns in the web page's underlying code. We can use web scraping to gather unstructured data from the internet, process it and store it in a structured format.



```
  ::before
▶<div class="row header-box">…</div>
  <script src="/static/jquery.js"></script>
▶<script>…</script>
▼<div class="quote">
  ▼<span class="text"> == $0
      ""The world as we have created it is a process of our thinking. It cannot be changed
      without changing our thinking.""
    </span>
  ▼<span>
      "by "
      <small class="author">Albert Einstein</small>
    </span>
  ▼<div class="tags">
      "Tags: "
      <a class="tag">change</a>
      <a class="tag">deep-thoughts</a>
      <a class="tag">thinking</a>
      <a class="tag">world</a>
    </div>
  </div>
▶<div class="quote">…</div>
▶<div class="quote">…</div>
▶<div class="quote">…</div>
▶<div class="quote">…</div>
▶<div class="quote">…</div>
▶<div class="quote">…</div>
```

On inspecting each quote element, we observe that each quote is enclosed within a div with the class name of quote. By running the directive

```
driver.get_elements_by_class("quote")
```

we get a list of all elements within the page exhibiting this pattern.

The command is then wrapped with a len() function to get the exact number of quotes within that page and store it in the item variable to make our iterator.

Final step

```
quotes = driver.find_elements_by_class_name("quote")
for quote in quotes:
    quote_text = quote.find_element_by_class_name('text').text[1:]
    author = quote.find_element_by_class_name('author').text
    new = ((quote_text,author))
    total.append(new)
```

To begin extracting the information from the webpages, we'll take advantage of the aforementioned patterns in the web pages underlying code.
To start, we'll need the list of all quotes that we'd described above. On this step, however, we'll not be enclosing it in a len() function as we need individual elements.

Afterwards, the inner for loop is to iterate over each quote and extract a specific record.
From the picture above we notice that the quote is enclosed within a span of class text and the author within the small tag with a class name of author.

Finally, we store the quote_text and author names variables in a tuple which we proceed to append to the python list by the name total.

```
    df = pd.DataFrame(total,columns=['quote','author'])
    df.to_csv('quoted.csv')
driver.close()
```

Using the pandas library, we'll initiate a dataframe to store all the records(total list) and specify the column names as quote and author. Finally, export the dataframe to a CSV file which we named quoted.csv in this case.

Don't forget to close the chrome driver using driver.close().

Adittional resources.

1. finding elements

You'll notice that I used the find_elements_by_class method in this walkthrough. This is not the only way to find elements. This tutorial by Klaus explains in detail how to use other selectors.

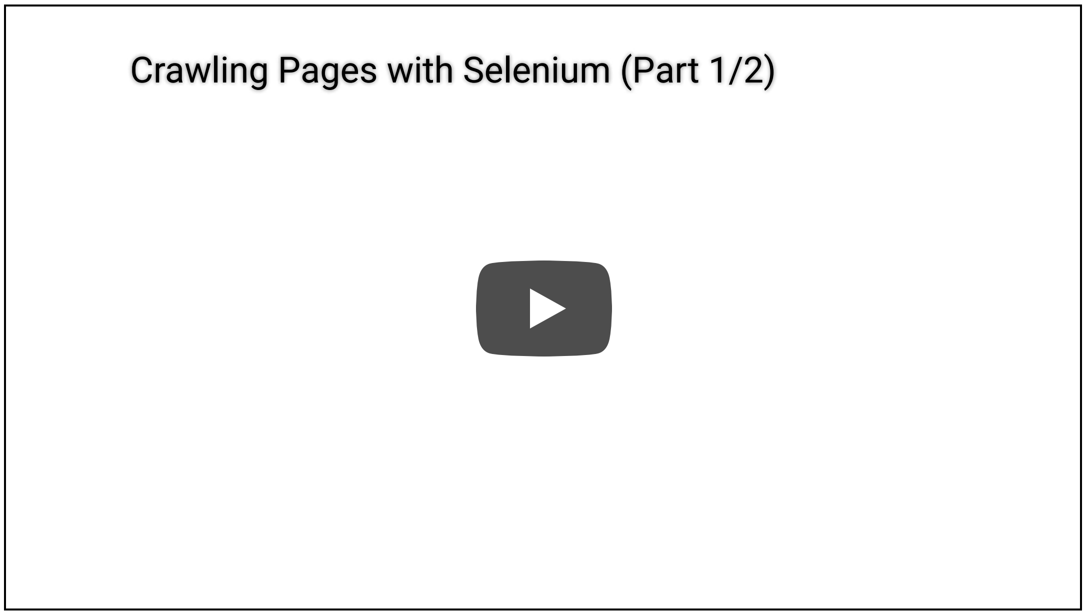
A Practical Guide for Finding Elements with Selenium

Klaus · Jun 12 '19 · 7 min read

#testing #productivity #webdev #javascript

2. Video

If you prefer to learn using videos this series by Lucid programming was very useful to me.



3. Best practises while using selenium

And with that, hopefully, you too can make a simple web scraper using selenium 😎.

If you enjoyed this post subscribe to my newsletter to get notified whenever I write new posts or find me on twitter for a chat.

Thanks.

web scraping techniques with python (3 Part Series)

1) Introduction to web scraping with python

2) Beginner's guide to web scraping with python's selenium

3) web scraping: Managing proxies and Captcha with scrapy and the Scrape...

Lewis kori + FOLLOW

I'm a passionate fitness buff.

@lewiskori lewis_kihiu lewis-kori lewiskori.com

Add to the discussion

PREVIEW SUBMIT

code of conduct - report abuse

Classic DEV Post from Jul 2 '19

How to refill someone’s “cup?”

Cubicle Buddha

Sometimes we rely so much on our friends and coworkers that we often forget to ma...

76 9

Another Post You Might Like

Five things you didn't know you could do with Python

Burke Holland

Python: a large and terrifying snake OR the third most popular programming language. This week, Python (the language, not the snek) aficionado Nina Zakharenko joins us for Five Things that you didn’t know that Python can do

76 2

Another Post You Might Like

Codecademy x Adafruit Launched Hardware Course

Sonny Li

Hello everyone, we recently teamed up with our neighbors at Adafruit and releas...

57 5

How to communicate with non-programmers

Arpit Mohan - Feb 17

A Complete Beginner's Guide to React Router (include Router Hooks)

Ibrahima Ndaw - Feb 17

Add a Table of Contents with Smooth scroll using Gatsby and MDX

Scott Spence - Feb 17

#MakeTheChange: From Java to Kotlin

Tavon - Feb 17