CS 021 –Computer Programming I: Python
Use Cases – Web Scraping

Overview

The following procedure outlines the process through which information can be extracted from websites and condensed into a useable format using a technique known as Web Scraping. This technique is widely used in industry in all kinds of applications. Some of its most popular uses include data collection for training Machine Learning models, market research, and business development. Due to its high customizability and fairly low implementation cost, nearly all companies use this form of data mining.

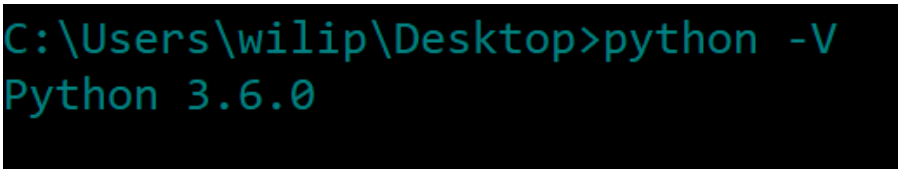
Note: not all websites allow scrapers. Please read their guidelines to ensure that all rules are being followed. These guidelines can potentially lead to legal ramifications if not followed on a case by case basis.

Procedure

Note: this tutorial is for Windows, if you are on a Mac, simply add the 'sudo' markup before each command

1. Open Command Prompt (Terminal on Mac), navigate to your Desktop, create a new directory for this project and navigate into it
    a. Commands
        i. **'cd desktop'**
            1. cd: change directory
        ii. **'mkdir <filename>'**
            1. mkdir: make directory

2. Ensure that Python is installed on your machine by typing 'python -V' or 'python3 -V' if you have not added it to your system PATH
    a. Commands
        i. **'python 3 -V'**



3. Next, we want to install three packages via PIP, Pythons package manager. The first is called Selenium, which is a headless browser used to retrieve the raw HTML from remote webpages. To install this package, run **'pip install selenium'** and wait for it to complete. The second is an HTML processing package called Beautiful Soup. This will allow us to parse the raw HTML that is returned from the Selenium request. To install, run **'pip install bs4'.** The third is a console output formatting package called pprint. To install, run **'pip install pprint'.**
    a. Commands
        i. **'pip install selenium'**
        ii. **'pip install bs4'**
        iii. **'pip install pprint'**

4. Once these packages are installed, we need to open our text editor. For this project, we are using **IDLE**, the editor that comes bundled with Python. Open IDLE, the use the command Control + N to create a new file, we won't be using the Python Shell for this example.

5. Now, we can get down to business. To get started, we need to import Selenium and Beautiful Soup and a few other dependencies into our program. Add all of the lines below to the top of your program.

    i. **Selenium Imports**

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import
WebDriverWait
from selenium.webdriver.support import
expected_conditions as EC
from selenium.common.exceptions import
TimeoutException
```

    ii. **Beautiful Soup Imports**

```
import urllib.request, urllib.parse, requests, re
from bs4 import BeautifulSoup as Soup
```

    iii. **Pprint Import**

```
import pprint
```

    iv. **System Imports**

```
from time import sleep
import sys
```

6. After we import all of the necessary packages, we can begin writing the scraper. First, we need to initialize a Client object, which will create a Selenium instance and fetch the website that we give it. Start by creating a class called 'Client' that takes an object as a parameter.

```
class Client(object):
```

7. Now, in Client class add the following code.

```
def __init__(self, url):
        option = webdriver.ChromeOptions()
        option.add_argument('-incognito')
        browser = webdriver.Chrome(

executable_path='C:/Users/wilip/Downloads/chromedriver_win3
2/chromedriver.exe',
            chrome_options=option
        )
        browser.get(url)
        sleep(5)
        self.source = browser.execute_script("return
document.getElementsByTagName('html')[0].innerHTML")

    def __repr__(self):
        return repr(self.source)
```

8. Once the code above is in your Client class, you will need to find the location of your copy of 'chromedriver.exe'. This is the executable that Selenium uses to fetch websites and is the backbone of the Chrome web browser. If you don't have Chrome installed, please do that now. It is usually located in your downloads folder as shown above.

```
'C:/Users/wilip/Downloads/chromedriver_win32/chromedriver.exe'
```

When you find the location of this program, change the path above to your path. After this is completed, the Selenium setup is finished.

9. Let's create a function that runs our scraper. For this example, the function will be called 'scrape'.

```
def scrape():
```

10. Now the fun begins. Pick your favorite scraper-friendly website, grab its URL and put it into your **scrape** function as a URL variable. Don't forget to wrap it in quotes!

```
url = 'https://www.catchafire.org/jobs'
```

11. After you grab the URL, add the following code below your URL variable.

```
all_opportunities = []
source = Client(url).source
soup = Soup(source,'lxml')
```

12. Now, let's test it out! Below the two lines above and inside your **scrape**, add a print statement to verify that this code is executing correctly.

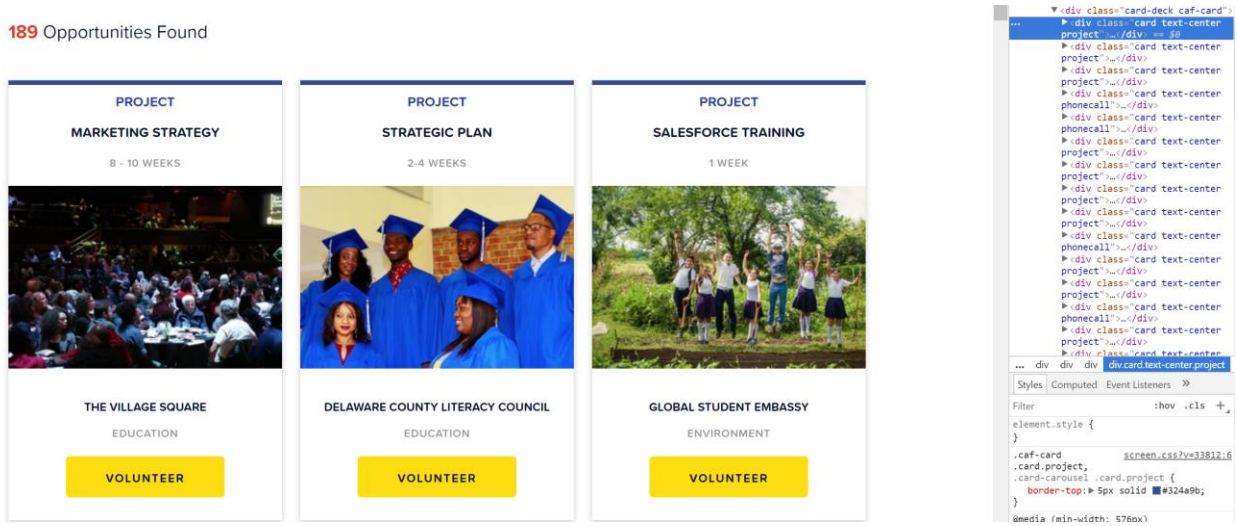
```
print(soup)
```

13. At the bottom of the file, below the scrape function, call **scrape()** and run the script by either clicking, **Run** followed by **Run Module** or by using the shortcut **Fn + F5**. The result will be the raw HTML (after Javascript rendering thanks to Selenium) for the website you chose.

```
ty&lt;/h1&gt;
      &lt;p&gt;Please follow these simple instructions to &lt;a href="http://www
.activatejavascript.org" target="_blank"&gt;enable JavaScript in your web browse
r&lt;/a&gt;.&lt;/p&gt;
    &lt;/div&gt;
  </noscript>
<!-- REDESIGN: Bootstrap JS: we need this locally, no idea where it is right now
. Note also jquery version -->
<script src="//cdnjs.cloudflare.com/ajax/libs/tether/1.4.0/js/tether.min.js"></s
cript>
<script src="//v4-alpha.getbootstrap.com/dist/js/bootstrap.min.js"></script>
<script src="https://det2iec3jodwn.cloudfront.net/responsive/javascripts/bootstr
ap-hacks.js?v=33812"></script>
<!-- REDESIGN: custom JS scripts, leave alone for now! -->
<script src="https://det2iec3jodwn.cloudfront.net/gen/site.vendor.js?v=33812"></
script>
<script src="https://det2iec3jodwn.cloudfront.net/responsive/javascripts/jquery.
block-ellipsis.js?v=33812"></script>
<script src="https://det2iec3jodwn.cloudfront.net/responsive/javascripts/site.js
?v=33812"></script>
<script src="https://det2iec3jodwn.cloudfront.net/responsive/javascripts/project
s/search.js?v=33812"></script>
<script type="text/javascript">
    // show browser outdated modal
    $(function() {
      initUnsupported();
    })
    var initUnsupported = function() {
      if ($('#modalBrowserUnsupported').length) {
        var $browserModal = $('#modalBrowserUnsupported');
        $browserModal.modal('show');
      }
    }
  </script>
<iframe id="intercom-frame" style="display: none;"></iframe><div id="intercom-co
ntainer" style="position: fixed; width: 0px; height: 0px; bottom: 0px; right: 0p
x; z-index: 2147483647;"><div class="intercom-app" data-reactroot=""><span></spa
n><span></span><!-- react-empty: 4 --><span></span><!-- react-empty: 6 --></div>
</div></body></html>
```

Now, at this point you might be thinking that this isn't very useful, just a bunch of jumbled HTML. That is why we are using Beautiful Soup. It provides us with a set of tools for parsing through this mess and finding the data that we actually want.

14. Let's parse the data. To start, navigate to the website you chose to scrape, right click, and inspect the page. We're looking for a pattern in the HTML attributes of the data that we are looking for. For this example, the website we are scraping is a volunteering platform and we want to get the specifics for each individual opportunity. Below is what the website looks like.



As you can see on the far-right side of the screen, each opportunity has the same class, 'card text-center project'. We can then tell Beautiful Soup that we want to filter the HTML that is retrieved by this class name, thus giving us the set of data that we are looking for.

15. To achieve this result, we need to add this line of code to our **scrape** function directly under 'soup = Soup(source, 'lxml')'. This code tells Beautiful Soup to parse the raw HTML that Selenium retrieved into an array of elements that contain this specific class name.

```
results = soup.findAll('div', attrs={'class':'card text-center project'})
```

16. Let's test it now. Remove the print statement that we added in step 12 and add print(results) under the line above. Then run the script. The output is shown below.

```
[<div class="card text-center project">
<div class="card-block">
<h4 class="card-title">
<a href="/volunteer/20406/">

          Project

      </a>
</h4>
<h5 class="card-sub-title">
<a href="/volunteer/20406/">Photography</a>
</h5>
<p class="card-extra">

        1 week

    </p>
</div>
<a href="/volunteer/20406/">
<figure class="hasimage" style="background-image: url('https://d20xup02wxfuga.cl
oudfront.net/33812/images/np_covers/099e58f01d544e4f9748d874550338fe_300x180_cro
pped.png')">
<figcaption>
<span class="quote quote-open"></span>
<p>

        This project will save us $1,500, allowing us to <strong>use that funds
towards increasing and expanding our work and impact in the lives of adolescent
girls in Kenya.</strong>
</p>
<span class="quote quote-close"></span>
</figcaption>
</figure>
</a>
<div class="card-footer">
```

As you can see, the **results** array is a set of div elements with the class name that we specified, so we are getting closer to the end result. That being said, we don't want any HTML element in our final dataset, so we need to filter this array even further until we are left with the raw data that is contained in these elements.

17. To achieve this, we need to loop through each element in the **results** array. First lets create a for loop that does just that. Add the following code into your **scrape** function below the print statement from the previous step.

```
for result in results:
        d = {'source':'catchafire', 'is_company': False}

        try:
             body = result.find('div', attrs={'class':'card-block'})

            d['type'] = body.find('h4').get_text().strip()
             d['offsiteURL'] = "https://www.catchafire.org" +
             body.find('h4').find('a').attrs['href']

             d['title'] = body.find('h5').find('a').get_text().strip()

            d['duration'] = body.find('p').get_text().strip()
             footer = result.find('div', attrs={'class':'card-footer'})

             d['company'] = footer.find('h6').find('a').get_text().strip()

             d['companyLink'] = "https://www.catchafire.org" +
             footer.find('h6').find('a').attrs['href']

             d['category'] = footer.find('p').find('a').get_text().strip()

        except Exception as e:
            print(e)

        if 'title' in d.keys():
            all_opportunities.append(d)

return all_companies
```

This block of code iterates through the **results** array and extracts relevant data from each element inside the parent div. Each data field is then added to the **d** object which constructs a single data object with all necessary data. In this example, we are finding the opportunity **type, URL, title, duration, company, companyLink,** and **category**. Once all fields are found, and the **d** object is completed, it will be added to the all_opportunities array. When the loop finishes, the all_opportunities array that contains all of the data we want will be returned. Simply remove all print statements and call the **scrape** function by adding `pprint.pprint(scrape('35',2))` at the end of your file. The output is shown below.

```
[{'category': 'Human Services',
  'company': 'Pulse Center for Patient Safety Education & Advocacy',
  'companyLink': 'https://www.catchafire.org/organizations/pulse-center-for-patient-safety-education---advocacy_13917',
  'duration': '6-8 weeks',
  'is_company': False,
  'offsiteURL': 'https://www.catchafire.org/volunteer/18385/',
  'source': 'catchafire',
  'title': 'Public Relations Strategy',
  'type': 'Project'},
 {'category': 'Human Services',
  'company': 'Pulse Center for Patient Safety Education & Advocacy',
  'companyLink': 'https://www.catchafire.org/organizations/pulse-center-for-patient-safety-education---advocacy_13917',
  'duration': '1-2 months',
  'is_company': False,
  'offsiteURL': 'https://www.catchafire.org/volunteer/18155/',
  'source': 'catchafire',
  'title': 'Earned Income Plan',
  'type': 'Project'},
 {'category': 'Human Services',
  'company': 'Easterseals New Jersey',
  'companyLink': 'https://www.catchafire.org/organizations/easterseals-new-jersey_13626',
  'duration': '8 - 10 weeks',
  'is_company': False,
  'offsiteURL': 'https://www.catchafire.org/volunteer/16701/',
  'source': 'catchafire',
  'title': 'Marketing Strategy',
  'type': 'Project'}]
```


Now that we have the data we want, we have a plethora of application that we can use it for. In the next lesson, we will learn how to use a package called Seaborn to turn this data into beautiful graphs and charts.