Adventures in Babysitting: Introduction to Web Scraping in Python

Julia Piaskowski

2020/02/09

```
## virtualenv: pycas2020
```

https://github.com/jpiaskowski/pycas2020_web_scraping

But, Who Actually Reads These A to Z?

(spoiler: not me)

me and my programming books

The Main Things to Know in a Web Scraping Project:

- Is it worth the trouble?
- Is it ethical?
- Tools available in `BeautifulSoup` and `requests`
- What to look for in html code
- Parsing json objects with `json`
- Rudimentary `pandas` skills
- `<pro-tip> All you need to know about html is how tags work </pro-tip>`

What to Look for in a Scraping Project:

- A sizeable amount of structured data with a regular repeatable format.
- Identical formating is not required, but the more edge cases present, the more complicated the scraping will be.

Ethics in Scraping

Accessing vast troves of information can be intoxicating.

Just because it's possible doesn't mean it should be done

Legal Considerations

(note: I have zero legal training - this is not legal advice!)

- Are you scraping copyrighted material?
- Will your scraping activity compromise individual privacy?
- Are you making a large number of request that may overload or damage a server?
- Is it possible the scraping will expose intellectual property you do not own?
- Are there terms of service governing use of the website and are you following those?
- Will your scraping activities diminish the value of the original data?

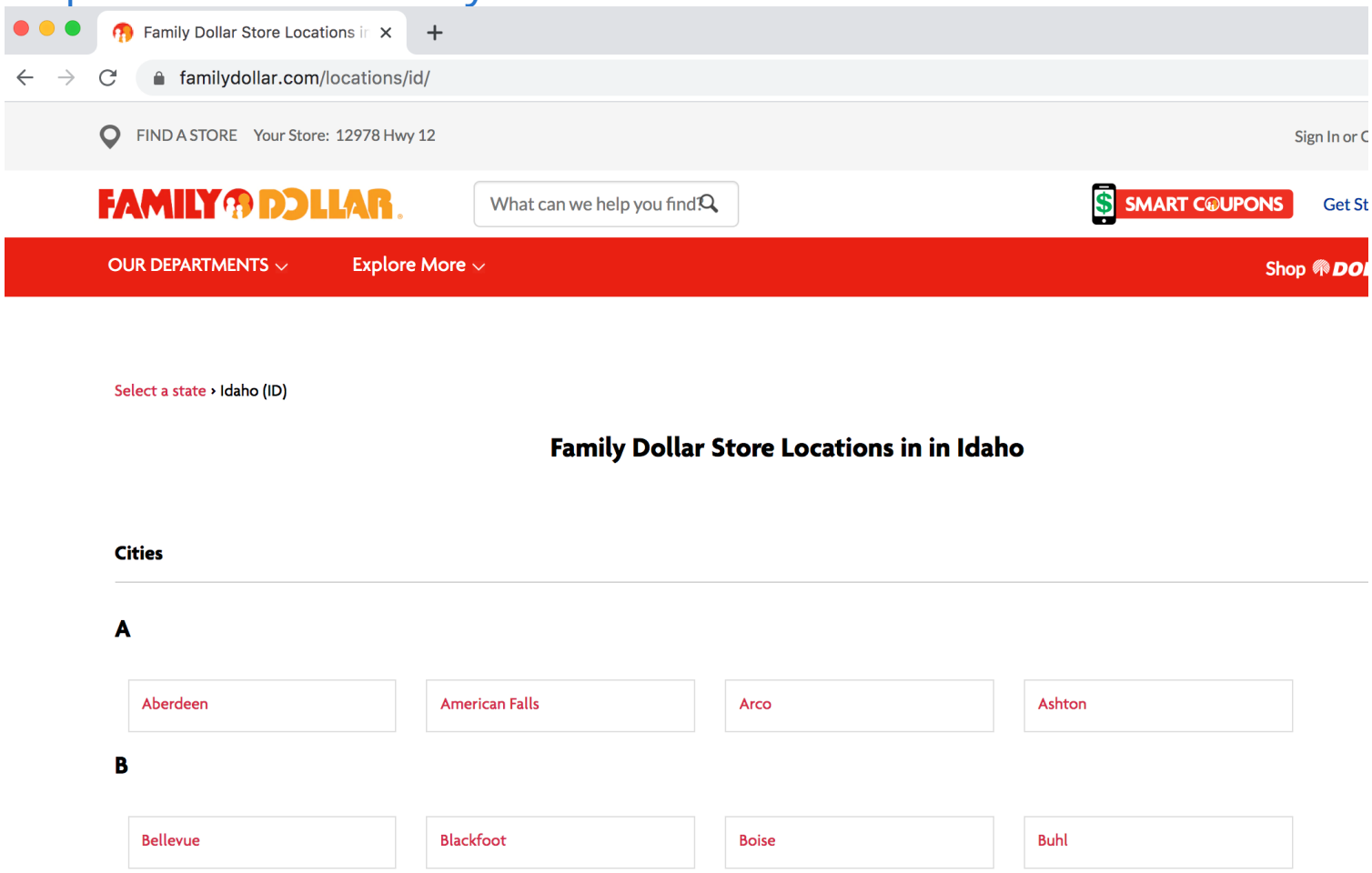
Dollar Stores are Taking Over the World!

Store in Cascade, Idaho

Goal: Extract addresses for all Family Dollar stores in Idaho.

The Starting Point:

https://locations.familydollar.com/id/



Step 1: Load the Libraries

```
import requests # for making standard html requests
from bs4 import BeautifulSoup # magical tool for parsing html data
import json # for parsing data
from pandas import DataFrame as df # data organization
```

Step 2: Grab Some Data from Target Web Address

```
page = requests.get("https://locations.familydollar.com/id/")
soup = BeautifulSoup(page.text, 'html.parser')
```

Beautiful Soup will take html or xml content and transform it into a complex tree of objects. Here are several common types:

- `BeautifulSoup` - the soup (the parsed content)
- `Tag` - main type of bs4 element you will encounter
- `NavigableString` - string within a tag
- `Comment` - special type of NavigableString

Step 3: Determine How to Extract Relevant Content from bs4 Soup

This process can be frustrating.

Step 3: Finding Content…

- Start with one representative example and then scale up
- Viewing the page's html source code is essential
    - Run at your own risk:

```
print(soup.prettify())
```

Step 3: Finding Content…

- It is usually easiest to browse via “View Page Source”:

```
9352
9353
9354                    <H3 style="font-size:16px; font-weight:bold; padding-bottom:10px; margin-bottom:20px; border-bottom:1px solid #cdcdcd;">Cities</H3>
9355                    <h4 id="aAnchor">A</h4>
9356                    <div class="group-wrapper">
9357
9358
9359                            <div class="itemlist"><a dta-linktrack="City index page - Aberdeen" href="https://locations.familydollar.com/id/aberdeen/">Aberdeen</a></div>
9360
9361
9362
9363                            <div class="itemlist"><a dta-linktrack="City index page - American Falls" href="https://locations.familydollar.com/id/american-falls/">Americ
9364
9365
9366
9367                            <div class="itemlist"><a dta-linktrack="City index page - Arco" href="https://locations.familydollar.com/id/arco/">Arco</a></div>
9368
9369
9370
9371                            <div class="itemlist"><a dta-linktrack="City index page - Ashton" href="https://locations.familydollar.com/id/ashton/">Ashton</a></div>
9372
9373
```

- What attribute or tag sets your content apart from the rest?

Step 3: Finding Content by Searching

Searching for ‘href’ does not work.

```
dollar_tree_list = soup.find_all('href')
dollar_tree_list
```

```
## []
```

But searching on a specific class is often successful:

```
dollar_tree_list = soup.find_all(class_ = 'itemlist')
for i in dollar_tree_list[:2]:
  print(i)
```

```
## <div class="itemlist"><a dta-linktrack="City index page - Aberdeen" href="https://locations
## <div class="itemlist"><a dta-linktrack="City index page - American Falls" href="https://loca
```

Step 3: Finding Target Content by Using 'contents'

```
type(dollar_tree_list)
## <class 'bs4.element.ResultSet'>
len(dollar_tree_list)
## 48
```

Next, extract contents from this BeautifulSoup "ResultSet".

```
example = dollar_tree_list[2] # Arco, ID (single representative example)
example_content = example.contents
print(example_content)
```

```
## [<a dta-linktrack="City index page - Arco" href="https://locations.familydollar.com/id/arco,
```

Step 3: Finding Content in Attributes

Find out what attributes are present in the contents:

Note: `contents` usually return a list of exactly one item, so the first step is to index that item.

```
example_content = example.contents[0]
example_content.attrs
```

```
## {'dta-linktrack': 'City index page - Arco', 'href': 'https://locations.familydollar.com/id/a
```

Extract the relevant attribute:

```
example_href = example_content['href']
print(example_href)
```

```
## https://locations.familydollar.com/id/arco/
```

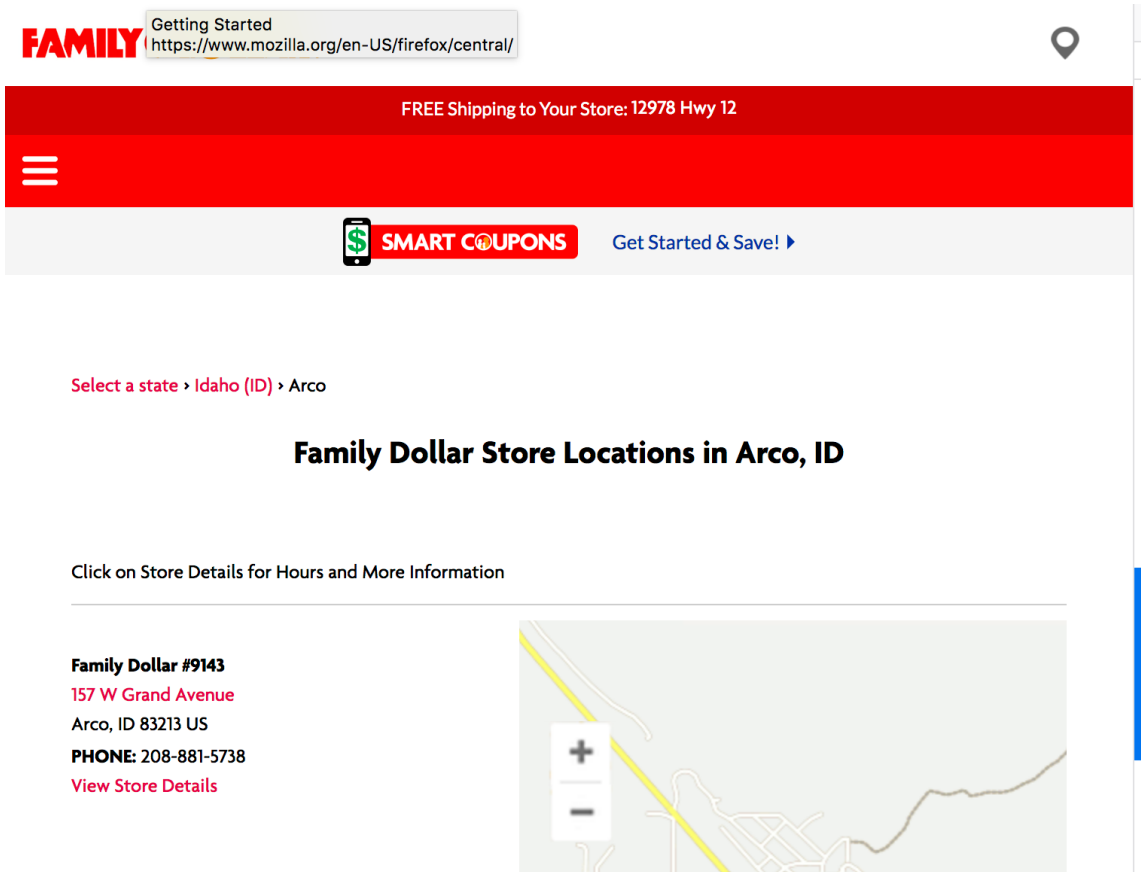
Step 4: Extract the Relevant Content

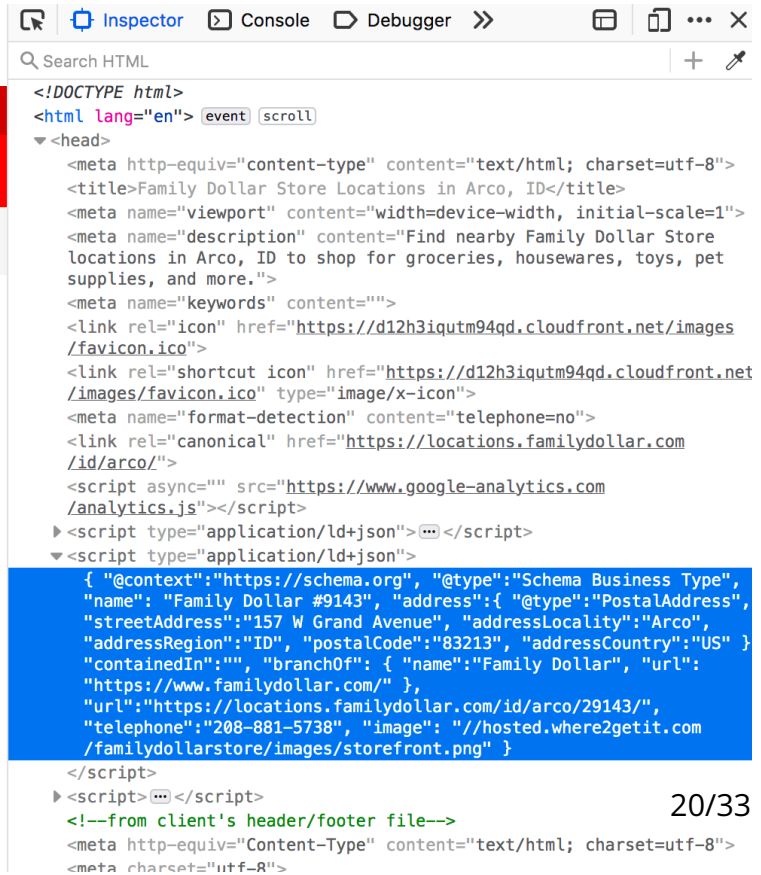
```
city_hrefs = [] # initialise empty list

for i in dollar_tree_list:
    cont = i.contents[0]
    href = cont['href']
    city_hrefs.append(href)

#  check to be sure all went well
for i in city_hrefs[:2]:
  print(i)


## https://locations.familydollar.com/id/aberdeen/
## https://locations.familydollar.com/id/american-falls/
```

Result: a list of URL's of Family Dollar stores in Idaho to scrape

Repeat Steps 1-4 for the City URLs

```
page2 = requests.get(city_hrefs[2]) # representative example
soup2 = BeautifulSoup(page2.text, 'html.parser')
```





Extract Address Information

from `type="application/ld+json"`

```
arco = soup2.find_all(type="application/ld+json")
print(arco[1])
```

```
## <script type="application/ld+json">
##  {
##    "@context":"https://schema.org",
##    "@type":"Schema Business Type",
##    "name": "Family Dollar #9143",
##    "address":{
##      "@type":"PostalAddress",
##      "streetAddress":"157 W Grand Avenue",
##      "addressLocality":"Arco",
##      "addressRegion":"ID",
##      "postalCode":"83213",
##      "addressCountry":"US"
##    },
##    "containedIn":"",
```

Use ‘contents’ to Find Address Information

Extract the contents (from the second list item) and index the first (and only) list item:

```
arco_contents = arco[1].contents[0]
arco_contents
```

```
## '\n\t{\n\t  "@context":"https://schema.org",\n\t  "@type":"Schema Business Type",\n\t  "name
```

Next, convert to a json object:
(these are way easier to work with)

```
arco_json =  json.loads(arco_contents)
```

Extract Content from a json Object

A json object is a dictionary:

```
type(arco_json)
```

```
## <class 'dict'>
```

```
print(arco_json)
```

```
## {'@context': 'https://schema.org', '@type': 'Schema Business Type', 'name':
'Family Dollar #9143', 'address': {'@type': 'PostalAddress', 'streetAddress': '157 W
Grand Avenue', 'addressLocality': 'Arco', 'addressRegion': 'ID', 'postalCode':
'83213', 'addressCountry': 'US'}, 'containedIn': '', 'branchOf': {'name': 'Family
Dollar', 'url': 'https://www.familydollar.com/'}, 'url':
'https://locations.familydollar.com/id/arco/29143/', 'telephone': '208-881-5738',
'image': '//hosted.where2getit.com/familydollarstore/images/storefront.png'}
```

Extract Content from a json Object

```
arco_address = arco_json['address']
arco_address
```

```
## {'@type': 'PostalAddress', 'streetAddress': '157 W Grand Avenue',
'addressLocality': 'Arco', 'addressRegion': 'ID', 'postalCode':
'83213', 'addressCountry': 'US'}
```

Step 5: Put It All Together

Iterate over the list store URLs in Idaho:

```
locs_dict = [] # initialise empty list

for link in city_hrefs:
  locpage = requests.get(link)   # request page info
  locsoup = BeautifulSoup(locpage.text, 'html.parser')
      # parse the page's content
  locinfo = locsoup.find_all(type="application/ld+json")
      # extract specific element
  loccont = locinfo[1].contents[0]
      # get contents from the bs4 element set
  locjson = json.loads(loccont)  # convert to json
  locaddr = locjson['address'] # get address
  locs_dict.append(locaddr) # add address to list
```

Step 6: Finalise Data

```
locs_df = df.from_records(locs_dict)
locs_df.drop(['@type', 'addressCountry'], axis = 1, inplace = True)
locs_df.head(n = 5)
```

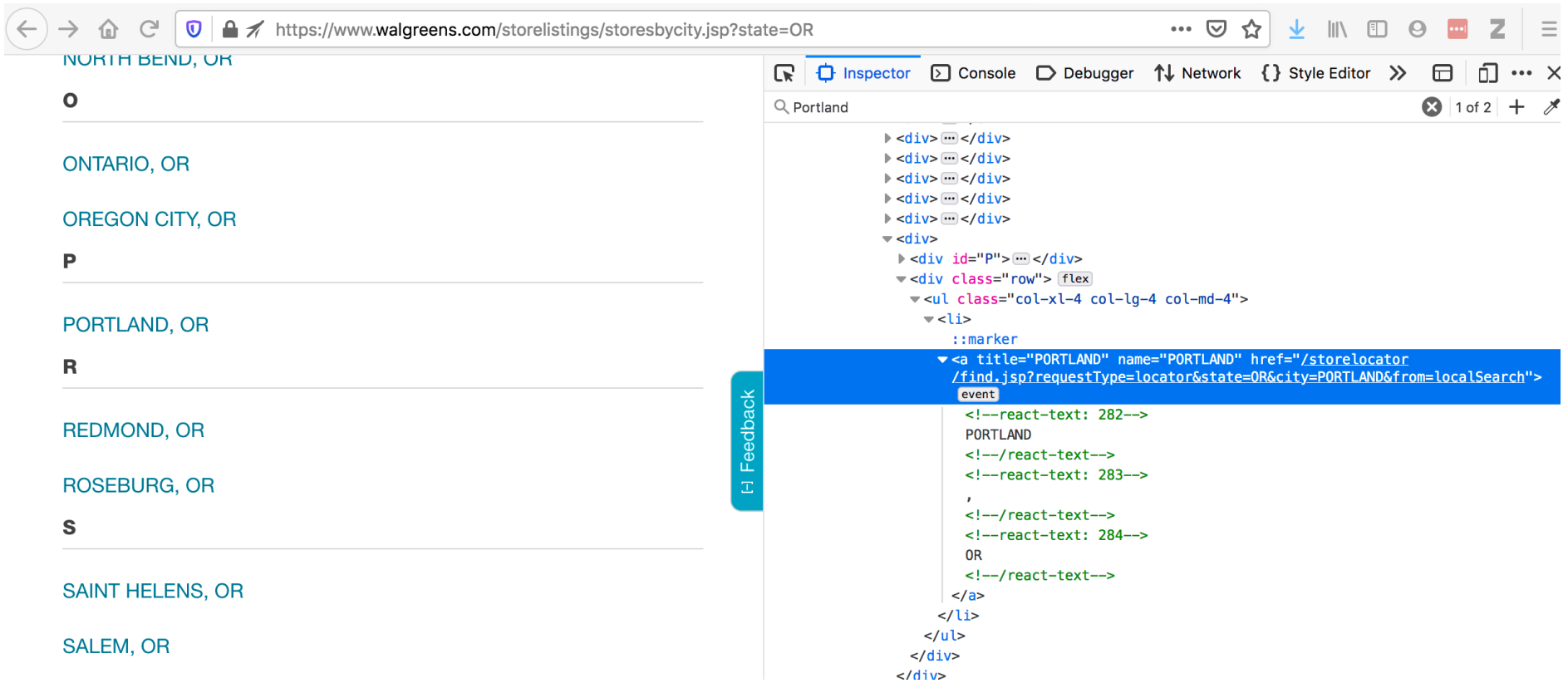
```
##         streetAddress addressLocality addressRegion postalCode
## 0   111 N Main Street        Aberdeen            ID      83210
## 1     253 Harrison St  American Falls            ID      83211
## 2  157 W Grand Avenue            Arco            ID      83213
## 3     177 Main Street          Ashton            ID      83420
## 4     747 N. Main St.        Bellevue            ID      83313
```

Results!!

```
df.to_csv(locs_df, "family_dollar_ID_locations.csv", sep = ",", index = False)
```

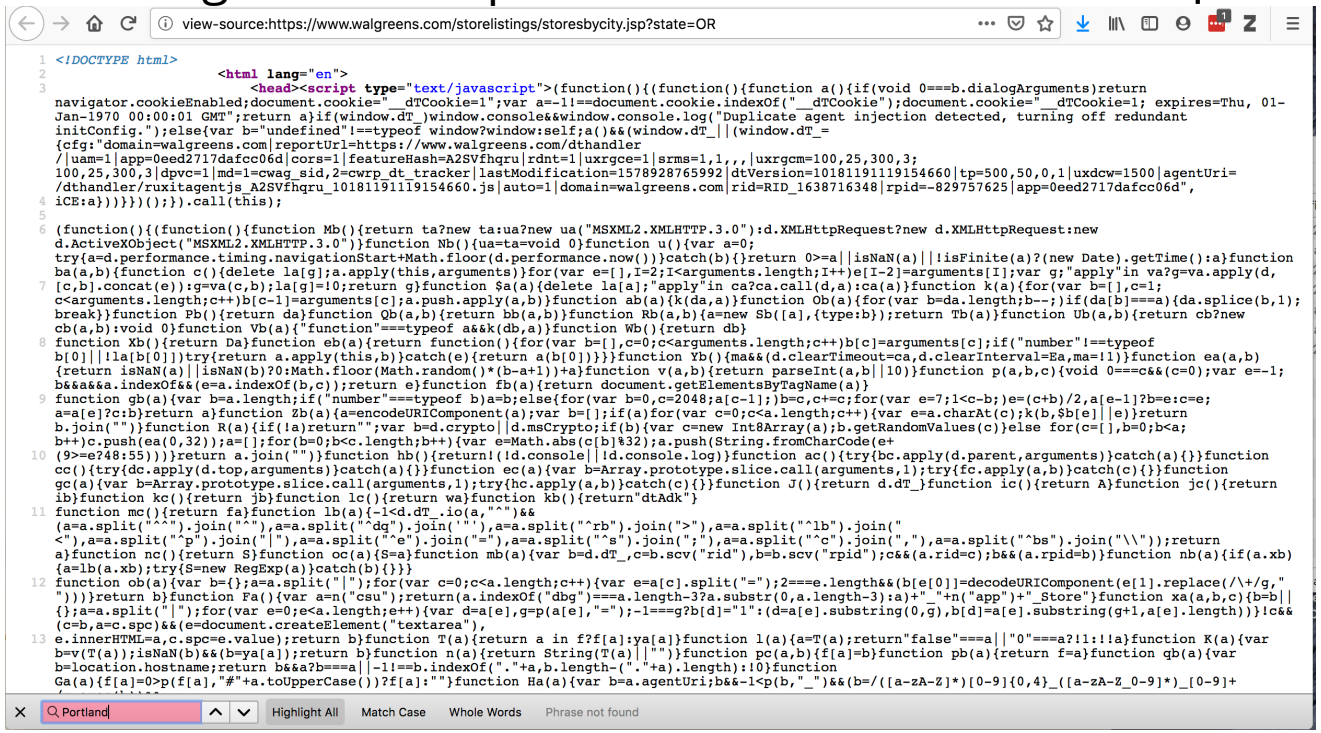
A Few Words on Selenium

“Inspect Element” provides the code for what is displayed in a browser.



A Few Words on Selenium

"View Page Source" - provides the code for what `requests` will obtain



There are plugins modifying the source code - so, it should be accessed *after* the page has loaded in a browser.

A Few Words on Selenium

- Requires a webdriver to retrieve the content
- It actually opens a web browser, and this info is collected
- Selenium is powerful - it can interact with loaded content in many ways
- After getting data, continue to use `BeautifulSoup` as before

```
url = "https://www.walgreens.com/storelistings/storesbycity.jsp?requestType=locator&state=ID"
driver = webdriver.Firefox(executable_path = 'mypath/geckodriver.exe')
driver.get(url)
soup_ID = BeautifulSoup(driver.page_source, 'html.parser')
store_link_soup = soup_ID.find_all(class_ = 'col-xl-4 col-lg-4 col-md-4')
```

The Penultimate Slide

Read the Manuals

- https://beautiful-soup-4.readthedocs.io/en/latest/
- https://selenium.dev/

This talk available at:

https://github.com/jpiaskowski/pycas2020_web_scraping

Perservere

~ After Becoming a Web Scraping Master ~

https://github.com/jpiaskowski/pycas2020_web_scraping

Bonus Slide!