Untitled

January 10, 2018

1 Web Scraping with python

1.1 Intro

web scrapers can access data from databases which expand to millions of pages unlike browser which are generally good at executing js scripts. A Google search for "cheapest flights to Boston" will result in a slew of advertisements and popular flight search sites. Google only knows what these websites say on their content pages, not the exact results of various queries entered into a flight search application. However, a well-developed web scraper can chart the cost of a flight to Boston over time, across a variety of websites, and tell you the best time to buy your ticket.

if it hits you "Isn't data gathering what APIs are for?" (If you're unfamiliar with APIs, see Chapter 4.) Well, APIs can be fantastic, if you find one that suits your purposes. They can provide a convenient stream of well-formatted data from one server to another. You are gathering data across a collection of sites that do not have a cohesive API. The data you want is a fairly small, finite set that the webmaster did not think warranted an API. The source does not have the infrastructure or technical ability to create an API.

A web browser can tell the processor to send some data to the application that handles your wireless (or wired) interface, but many languages have libraries that can do that just as well.

```
In [11]: import requests
         html = requests.get("http://pythonscraping.com/pages/page1.html")
         type(html)

Out[11]: requests.models.Response

In [12]: string_html=html.text
         type(string_html)

Out[12]: str
```

An Introduction to BeautifulSoup "Beautiful Soup, so rich and green, Waiting in a hot tureen! Who for such dainties would not stoop? Soup of the evening, beautiful Soup!"

```
In [14]: from bs4 import BeautifulSoup

In [20]: bsObj = BeautifulSoup(string_html,'html.parser')

In [21]: bsObj
```

```
Out[21]: <html>
         <head>
         <title>A Useful Page</title>
         </head>
         <body>
         <h1>An Interesting Title</h1>
         <div>
         Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incidid
         </div>
         </body>
         </html>
```

```
In [22]: bsObj.h1
```

```
Out[22]: <h1>An Interesting Title</h1>
```

we extracted from the page was nested two layers deep into our BeautifulSoup object structure (html body h1). However, when we actually fetched it from the object, we called the h1 tag directly: bsObj.h1 In fact, any of the following function calls would produce the same output: bsObj.html.body.h1 bsObj.body.h1 bsObj.html.h1

Virtually any information can be extracted from any HTML (or XML) file, as long as it has some identifying tag surrounding it, or near it. In chapter 3, we'll delve more deeply into some more-complex BeautifulSoup function calls, as well as take a look at regular expressions and how they can be used with Beau tifulSoup in order to extract information from websites.

```
In [ ]: html = requests.get("http://www.pythonscraping.com/pages/page1.html")
```

Here, to anticipate the server error or the page not found error we include try and except and othe r conditions to filter the ways the scraping can go wrong.

```
In [ ]: try:
            html = urlopen("http://www.pythonscraping.com/pages/page1.html")
        except HTTPError as e:
            print(e)
        #return null, break, or do some other "Plan B"
        else:
            #program continues. Note: If you return or break in the
            #exception catch, you do not need to use the "else" statement
```

```
In [ ]: if html is None:
            print("URL is not found")
        else:
            #program continues
```

```
In [26]: import requests
         from urllib.error import HTTPError
         from bs4 import BeautifulSoup

         def getTitle(url):
```

```
        try:
            html = requests.get(url)
            string_html = html.text
        except HTTPError as e:              #if page not found for above request
                                             #an HTTP error will be returned.
            return None
        try:
            bsObj = BeautifulSoup(string_html,'html.parser')
            title = bsObj.body.h1
        except AttributeError as e:       #Server not found
            return None
        return title
    title = getTitle("http://www.pythonscraping.com/pages/page1.html")
    if title == None:
        print("Title can't be found")
    else:
        print(title)
```

```
<h1>An Interesting Title</h1>
```

You'll also likely want to heavily reuse code. Having generic functions such as getSiteHTML and getTitle (complete with thorough exception handling) makes it easy to quickly—and reliably—scrape the web.

1.2 Chap 2

1.2.1 Advanced HTML parsing

There are many techniques to chip away the content that doesn't look like the content that we're searching for, until we arrive at the information we're seeking. In this chapter, we'll take look at parsing complicated HTML pages in order to extract only the information we're looking for.

We need better formatted HTML: so what we do is find the print this page link or mobile version link(sometimes by presenting yourself as a mobile device) Look for the information hidden in a JavaScript file. Remember, you might need to examine the imported JavaScript files in order to do this. For example, I once collected street addresses (along with latitude and longitude) off a website in a neatly formatted array by looking at the JavaScript for the embedded Google Map that displayed a pinpoint over each address. This is more common for page titles, but the information might be available in the URL of the page itself. If the information you are looking for is unique to this website for some reason, you're out of luck. If not, try to think of other sources you could get this informa tion from. Is there another website with the same data? Is this website displaying data that it scraped or aggregated from another website?

Especially when faced with buried or poorly formatted data, it's important not to just start digging. Take a deep breath and think of alternatives. In this section, we'll discuss searching for tags by attributes, working with lists of tags, and parse tree navigation. CSS styling is boon to scrapers. CSS relies on the differentiation of HTML elements that might otherwise have the exact same markup in order to style them differently. tags might look like:

<span class="green"><span>
while others look like this:
<span class="red"></span>
now for scrapers it is really easy to filter tags with only the "red" string.

```
In [27]: html = requests.get("http://www.pythonscraping.com/pages/warandpeace.html")
         string_html=html.text

         bsObj = BeautifulSoup(string_html,'html.parser')
```

we used bsObj.tagName in order to get the first occurrence of that tag on the page. Now, we're calling bsObj.findAll(tagName, tagAttributes) in order to get a list of all of the tags on the page, rather than just the first.

```
In [28]: nameList = bsObj.findAll("span", {"class":"green"})

In [29]: nameList #it should list all the proper nouns in the text, in the order they appear in
                  #War and Peace.

Out[29]: [<span class="green">Anna
          Pavlovna Scherer</span>, <span class="green">Empress Marya
          Fedorovna</span>, <span class="green">Prince Vasili Kuragin</span>, <span class="green
          Funke</span>, <span class="green">The prince</span>, <span class="green">Anna
          Pavlovna</span>, <span class="green">the Empress</span>, <span class="green">The princ
          Pavlovna</span>, <span class="green">Anna Pavlovna</span>]
```

find and findall findAll(tag, attributes, recursive, text, limit, keywords) find(tag, attributes, recursive, text, keywords)

tag

like tag argument (bsObj.body.h1) we can use: you can pass a string name of a tag or even a Python list of string tag names. For example, the following will return a list of all the header tags in a document:

```
In [36]: bsObj.findAll({'h1','h2','h3','h4','h5','h6'})  ## will find every header from the HTML
                                                          ##h6

Out[36]: [<h1>War and Peace</h1>, <h2>Chapter 1</h2>]
```

attributes

The attributes argument takes a Python dictionary of attributes and matches tags that contain any one of those attributes. For example, the following function would return both the green and red span tags in the HTML document:

```
In [ ]: bsObj.findAll("span", {"class":"green", "class":"red"})
```

recursive

The recursive argument is a boolean. How deeply into the document do you want to go? If recursion is set to True , the findAll function looks into children, and child ren's children, for tags that match your parameters. If it is false , it will look only at the top-level tags in your document.

By default, findAll works recursively (recur sive is set to True); it's generally a good idea to leave this as is, unless you really know what you need to do and performance is an issue.

text:

The text argument is unusual in that it matches based on the text content of the tags, rather than properties of the tags themselves.

```
In [39]: namelist=bsObj.findAll(text='the prince')
```

```
In [40]: len(namelist)
```

```
Out[40]: 7
```

limit

limit is used in findAll, find is equivalent to findAll with limit 1.

keyword The keyword argument allows you to select tags that contain a particular attribute. For example:

```
In [ ]: allText = bsObj.findAll(id="text")
        print(allText[0].get_text())      # These give you the whole text of the page that you
                                          #intended to scrap
```

everything that can done with keyword can also be done using normal techniques. above can be accomplished by: bsObj.findall('',{'id':'text'})

Other BeautifulSoup Objects: Till now we have seen two types of BS objects: 1) bsObj 2) tag objects... (find,findAll) other 2 are:

NavigableString objects Used to represent text within tags, rather than the tags themselves (some func tions operate on, and produce, NavigableStrings , rather than tag objects).

The Comment object Used to find HTML comments in comment tags, These four objects are the only objects you will ever encounter (as of the time of this writing) in the BeautifulSoup library.

Navigating Trees:

```
In [ ]: from urllib.request import urlopen
        from bs4 import BeautifulSoup

        html = urlopen("http://www.pythonscraping.com/pages/page3.html")

        bssObj = BeautifulSoup(html)
```

children and descendants bsObj.body.h1 selects the first h1 tag that is a descendant of the body tag. It will not find tags located outside of the body.

bsObj.div.findAll("img") will find the first div tag in the document, then retrieve a list of all img tags that are descendants of that div tag.

If you want to find only descendants that are children, you can use the .children tag:

```
In [ ]: for child in bssObj.find("table",{"id":"giftList"}).children:
            print(child)
```

This code prints out the list of product rows in the giftList table. If you were to write it using the descendants() function instead of the children() function, about two dozen tags would be found within the table and printed, including img tags, span tags, and individual td tags. It's definitely important to differentiate between children and descendants!

```
In [83]: #siblings
         from urllib.request import urlopen
         from bs4 import BeautifulSoup
         html = urlopen("http://www.pythonscraping.com/pages/page3.html")
         bsObj = BeautifulSoup(html)
         for sibling in bsObj.find("table",{"id":"giftList"}).tr.next_siblings:
             print(sibling)
```

```


<tr class="gift" id="gift1"><td>
Vegetable Basket
</td><td>
This vegetable basket is the perfect gift for your health conscious (or overweight) friends!
<span class="excitingNote">Now with super-colorful bell peppers!</span>
</td><td>
$15.00
</td><td>
<img src="../img/gifts/img1.jpg"/>
</td></tr>


<tr class="gift" id="gift2"><td>
Russian Nesting Dolls
</td><td>
Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mea
</td><td>
$10,000.52
</td><td>
<img src="../img/gifts/img2.jpg"/>
</td></tr>


<tr class="gift" id="gift3"><td>
Fish Painting
</td><td>
If something seems fishy about this painting, it's because it's a fish! <span class="excitingNot
</td><td>
$10,005.00
</td><td>
<img src="../img/gifts/img3.jpg"/>
</td></tr>
```

```
<tr class="gift" id="gift4"><td>
Dead Parrot
</td><td>
This is an ex-parrot! <span class="excitingNote">Or maybe he's only resting?</span>
</td><td>
$0.50
</td><td>
<img src="../img/gifts/img4.jpg"/>
</td></tr>


<tr class="gift" id="gift5"><td>
Mystery Box
</td><td>
If you love suprises, this mystery box is for you! Do not place on light-colored surfaces. May c
</td><td>
$1.50
</td><td>
<img src="../img/gifts/img6.jpg"/>
</td></tr>




/home/prashant/anaconda3/lib/python3.6/site-packages/bs4/__init__.py:181: UserWarning: No parser

The code that caused this warning is on line 193 of the file /home/prashant/anaconda3/lib/python

 BeautifulSoup(YOUR_MARKUP})

to this:

 BeautifulSoup(YOUR_MARKUP, "lxml")

  markup_type=markup_type))
```

```
In [91]: from urllib.request import urlopen
         from bs4 import BeautifulSoup
         import re
         html = urlopen("http://www.pythonscraping.com/pages/page3.html")
         bsObj= BeautifulSoup(html)
         images = bsObj.findAll("img", {"src":re.compile("\.\.\/img\/gifts/img.*\.jpg")})
         #This prints out only the relative image paths that start with ../img/gifts/img and end
         #.jpg
         for image in images:
             print(image["src"])
```

```
../img/gifts/img1.jpg
../img/gifts/img2.jpg
../img/gifts/img3.jpg
../img/gifts/img4.jpg
../img/gifts/img6.jpg
```

```
/home/prashant/anaconda3/lib/python3.6/site-packages/bs4/__init__.py:181: UserWarning: No parser

The code that caused this warning is on line 193 of the file /home/prashant/anaconda3/lib/python

 BeautifulSoup(YOUR_MARKUP})

to this:

 BeautifulSoup(YOUR_MARKUP, "lxml")

  markup_type=markup_type))
```

to get the attributes of the tag objects we use: myTag.attrs The source location for an image, for example, can be found using the following line: myImgTag.attrs['src']

Every tag object that BeautifulSoup encounters is evaluated in this function, and tags that evaluate to "true" are returned while the rest are discarded. For example, the following retrieves all tags that have exactly two attributes:

```
In [92]: bsObj.findAll(lambda tag: len(tag.attrs) == 2)
```

```
Out[92]: [<img src="../img/gifts/logo.jpg" style="float:left;"/>,
          <tr class="gift" id="gift1"><td>
          Vegetable Basket
          </td><td>
          This vegetable basket is the perfect gift for your health conscious (or overweight) fr
          <span class="excitingNote">Now with super-colorful bell peppers!</span>
          </td><td>
          $15.00
          </td><td>
          <img src="../img/gifts/img1.jpg"/>
          </td></tr>,
          <tr class="gift" id="gift2"><td>
          Russian Nesting Dolls
          </td><td>
          Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceles
          </td><td>
          $10,000.52
          </td><td>
          <img src="../img/gifts/img2.jpg"/>
          </td></tr>,
          <tr class="gift" id="gift3"><td>
Fish Painting
</td><td>
If something seems fishy about this painting, it's because it's a fish! <span class="e
</td><td>
$10,005.00
</td><td>
<img src="../img/gifts/img3.jpg"/>
</td></tr>,
<tr class="gift" id="gift4"><td>
Dead Parrot
</td><td>
This is an ex-parrot! <span class="excitingNote">Or maybe he's only resting?</span>
</td><td>
$0.50
</td><td>
<img src="../img/gifts/img4.jpg"/>
</td></tr>,
<tr class="gift" id="gift5"><td>
Mystery Box
</td><td>
If you love suprises, this mystery box is for you! Do not place on light-colored surfa
</td><td>
$1.50
</td><td>
<img src="../img/gifts/img6.jpg"/>
</td></tr>]
```