Web scraping basics

Scraping Vs crawling

```
Web scraping: Automating the process of extracting information
from web pages
              * for data collection and analysis
              * for incorporating in a web app


Web crawling: Automating the process of traversing links on web
pages
              * for indexing the web
              * for collecting data from multiple web sites or
                pages.
```

Legal and ethical issues

Legal issues

- Check the ‘Terms of Use’ of a web site.
- Regardless, murky and unsettled

- Probably depends upon three things:
  - factual, non-proprietary data is generally ok
  - proprietary data scraping (if available) depends on what you do with it
  - potential or actual damage to the scrapee

Ethical issues

- Public vs. private information
- Purpose
- Try to get the information openly
- Is there a public interest involved

Techniques

String manipulation: Automating the process of extracting information from web pages

* for data collection and analysis
* for incorporating in a web app

Libraries for web scraping

requests: Python library for connecting to a web page, managing the connection and retrieving contents of the page

Beautiful Soup: A library that utilizes the ‘tag structure’ of an html page to quickly parse the contents of a page and retrieve data

Selenium: A library that utilizes the ‘tag structure’ of an html page to execute javascript scripts on the page and retrieve data. Slower than Beautiful Soup but gets around the ‘javascript’ problem

requests

We did that last week

BeautifulSoup4

- HTML (and XML) parser
- Uses ‘tags’

- Creates a parse tree (using lxml/ html5lib or other python parser)
- Can handle incomplete tagging

- tags are organized in hierarchical dictionaries

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

bs4

```
initialize bs4 object: BeautifulSoup(document,parser)
      parser: lxml (fast) or html5lib (slower but more robust)
```

```
import requests
from bs4 import BeautifulSoup
url = "http://www.epicurious.com/search/Tofu%20Chili"
response = requests.get(url)
page_soup = BeautifulSoup(response.content,'lxml')
print(page_soup.prettify())
```

page_soup is the object from which we will extract the data we need

Unique data identifiers

- We want to create a list of recipes and links to the recipes
- We need to figure out how to ‘programmatically’ extract each recipe name and recipe link

- Search for the tag with a unique attribute value that identifies recipes and recipe links
- We’ll look at the <a> tags because clickable links are in them

Unique data identifiers

```
for tag in page_soup.find_all('a'):
    if 'Spicy Lemongrass Tofu' in tag.get_text():
        print(tag)
```

This gets the innermost tags with the recipe name. Unfortunately, the attributes of 'a' do not uniquely identify recipe link tags. Let's look at its parent

```
for tag in page_soup.find_all('a'):
    if 'Spicy Lemongrass Tofu' in tag.get_text():
        print(tag.parent)
        break
```

<h4 class="hed" data-reactid="78" data-truncate="3" itemprop="name"><a data-reactid="79" href="/recipes/food/views/spicy-lemongrass-tofu-233844">Spicy Lemongrass Tofu</h4>

Test uniqueness of itemprop=name

```
for tag in page_soup.find_all('h4',itemprop='name'):
    print(tag.get_text())
```

bs4 functions

`<tag>.find(<tag_name>,attribute=value)`	finds the first matching child tag (recursively)
`<tag>.find_all(<tag_name>,attribute=value)`	finds all matching child tags (recursively)
`<tag>.get_text()`	returns the marked up text
`<tag>.parent`	returns the (immediate) parent
`<tag>.parents`	returns all parents (recursively)
`<tag>.children`	returns the (direct) children
`<tag>.descendants`	returns all children (recursively)
`<tag>.get(attribute)`	returns the value of the specified attribute

Practice 1 - Extract recipes and recipe links

```
Write a function epicurious_recipes(search_string) that
returns the list of recipes associated with search_string
```

Practice 2 - Ingredients and Instructions

Call the function with a search_string, open the link associated with the first recipe, then return the ingredients and preparation instructions associated with that link

Logging in to a site

- Figure out the login url
- https://en.wikipedia.org/w/index.php?title=Special:UserLogin&returnto=Main+Page
- Look for the login form in the html source
- form_tag = page_soup.find('form')
- Look for ALL the inputs in the login form (some may be tricky!)
- input_tags = form_tag.find_all('input')
- Create a Python dict object with key,value pairs for each input
- Use requests.session to create an open session object
- Send the login request (POST)
- Send followup requests keeping the sessions object open

Setting up the inputs

```
payload = {
    'wpName': username,
    'wpPassword': password,
    'wploginattempt': 'Log in',
    'wpEditToken': "+\\",
    'title': "Special:UserLogin",
    'authAction': "login",
    'force': "",
    'wpForceHttps': "1",
    'wpFromhttp': "1",
    #'wpLoginToken': ‘', #We need to read this from the page
    }
```

Extracting token information

wpLoginToken: the value of this attribute is provided by the page. we need to extract it.

```
login_page_response = s.get('https://en.wikipedia.org/w/index.php?
title=Special:UserLogin&returnto=Main+Page')
soup = BeautifulSoup(login_page_response.content,'lxml')
token = soup.find('input',{'name':"wpLoginToken"}).get('value')
```

Finalizing session parameters

```
username='MyWikipediaAccount1'
password='wikiuser'

def get_login_token(response):
    soup = BeautifulSoup(response.text, 'lxml')
    token = soup.find('input',{'name':"wpLoginToken"}).get('value')
    return token

payload = {
        'wpName': username,
        'wpPassword': password,
        'wploginattempt': 'Log in',
        'wpEditToken': "+\\",
        'title': "Special:UserLogin",
        'authAction': "login",
        'force': "",
        'wpForceHttps': "1",
        'wpFromhttp': "1",
        #'wpLoginToken': '',
        }
```

Activating session

```
with requests.session() as s:
    response = s.get('https://en.wikipedia.org/w/index.php?title=Special:UserLogin&returnto=Main+Page')
    payload['wpLoginToken'] = get_login_token(response)
    #Send the login request
    response_post = s.post('https://en.wikipedia.org/w/index.php?title=Special:UserLogin&action=submitlogin&type=login',
                           data=payload)
    #Get another page and check if we're still logged in
    response = s.get('https://en.wikipedia.org/wiki/Special:Watchlist')
```

Selenium

- Mimics a browser
- Designed for server testing

- Used in conjunction with a headless browser for web scraping

- Useful when web pages are using javascript
- Useful when a server bans code…

Setting up Selenium

- Install Selenium
  - `pip install selenium`
- Download and install phantomJS (headless browser)
  - http://phantomjs.org/download.html

Using Selenium

- Import webdriver from selenium
  - `from selenium import webdriver`
- Initialize a browser object
  - `browser = webdriver.Firefox()`
  - `browser = webdriver.PhantomJS(executable_path='/Applications/phantomjs')`
- Launch the browser and get a web page
  - `browser.get(url)`
- Extract data using Selenium Selectors

Selenium selectors

- Find element/elements by css selector
  - `browser.find_element_by_css_selector("nav.wrapper")`
  - `browser.find_elements_by_css_selector("nav.wrapper")`
- Find element/elements by class/id
  - `browser.find_elements_by_class_name("wrapper")`
- Find element/elements by tag
  - `browser.find_elements_by_tag_name("div")`
- Find element/elements by link text
  - `browser.find_elements_by_link_text(“AUD")`
  - `browser.find_elements_by_partial_link_text(“AU”)`
- Find element/elements by XPath
  - `browser.find_elements_by_xpath(“//div/div/table")`

Try this!

url = http://www.xe.com/currencytables/?from=EUR&date=2017-01-30

Use selenium to extract the rates table from the above page