LECTURE 13

Intro to Web Development

WEB DEVELOPMENT IN PYTHON

In the next few lectures, we'll be discussing web development in Python.

Python can be used to create a full-stack web application or as a scripting language used in conjunction with other web technologies. Well-known websites which utilize Python include

- Reddit (major codebase in Python)
- Dropbox (uses the Twisted networking engine)
- Instagram (uses Django)
- Google (Python-based crawlers)
- Youtube

WEB DEVELOPMENT IN PYTHON

In order to cover all of the aspects of front- and back-end web development, we will build and serve a website from scratch over the next few lectures. We will be covering

- Crawling and Scraping
- Templating
- Frameworks
- Databases
- WSGI Servers

GETTING STARTED

First, let’s introduce the website concept. We will be building a website that allows travelers to quickly look up some information, statistics and important advisories for countries that they may be visiting.

Our data source will be the CIA World Factbook, a rather large repository of information. Much of it is not useful for the average traveler, so we will pull out the relevant parts. However, the Factbook is updated constantly so we need to be able to grab the most recent information.

Before we can build our website, we need to understand how to grab the data. We will put together a Python module to grab a few pieces of data, but we can always extend it later as we add more features to our website.

CRAWLING AND SCRAPING

The process of gathering information from web pages is known as crawling and scraping.

- Crawling: the process of recursively finding links within a page and fetching the corresponding linked page, given some set of root pages to start from.

- Scraping: the process of extracting information from a web page or document.

Note: crawling and scraping is a legal grey issue. “Good” crawlers obey the robots.txt file accessible from the root of any website, which dictates the allowable actions that may be taken by a web robot on that website. It’s not clear whether disobeying the directives of robots.txt is grounds for legal action.

CRAWLING AND SCRAPING

Before we begin, since we're interested in gathering information from the CIA World Factbook, we should check out https://www.cia.gov/robots.txt -- but it doesn't exist! Let's check out craigslist.org/robots.txt as an example:

```
User-agent: *
Disallow: /reply
Disallow: /fb/
Disallow: /suggest
Disallow: /flag
Disallow: /mf
Disallow: /eaf
```

Applies to all robots

Directories that are prohibited

SCRAPING

Let’s talk about the objectives of our scraper.

- Input: Country Name (e.g. “Norway”) or Keyword (e.g. “Population”).

- Returns: Dictionary. If the input was a country name, dictionary contains relevant information about that country. If the input was a keyword, dictionary contains results for every country.

To start, we will support every country supported by the CIA World Factbook and we will support the keywords Area, Population, Capital, Languages, and Currency.

While we’re not technically doing any web development today, we’re creating a script which we can call from our website when a user requests information.

SCRAPING

Let’s write some basic code to get started.

We’ll be calling `get_info()` from our website, but we include some code for initializing `search_term` when the module is run by itself.

The very first thing we need to do is access the main page and figure out whether we’re looking for country or keyword data.

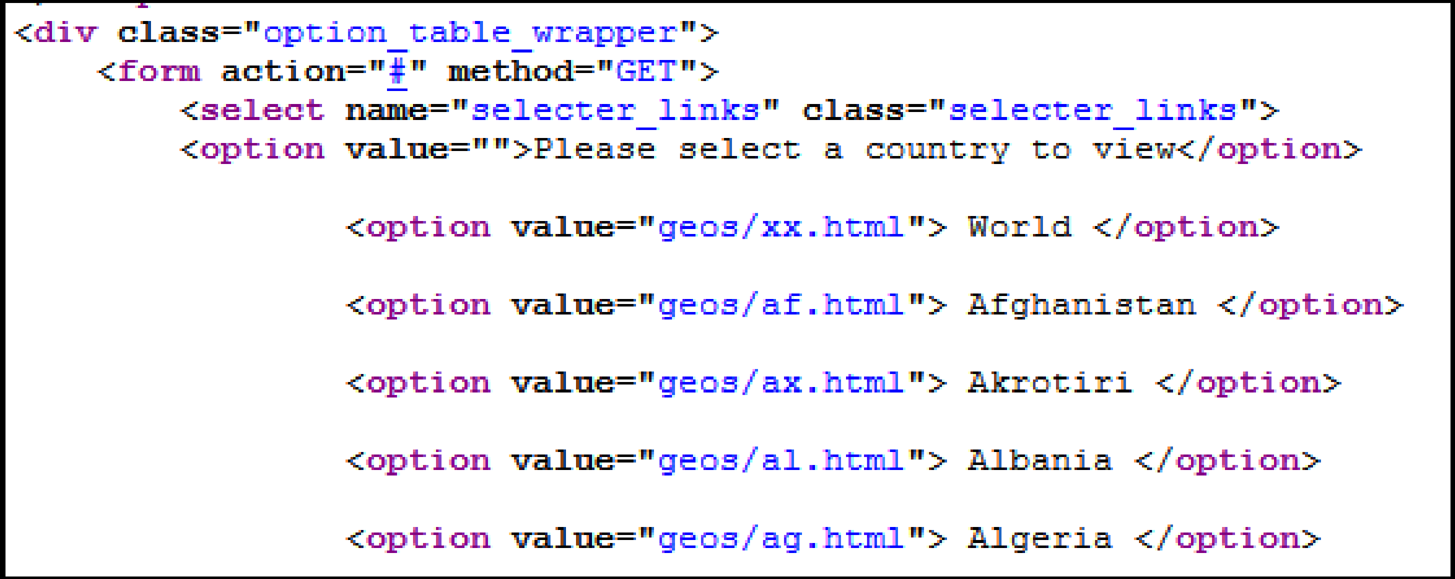
```python
def get_info(search_term):
    pass

if __name__ == "__main__":
    search_term = raw_input("Please enter a country or keyword: ")
    results = get_info(search_term)
```

SCRAPING

Take a look at the main page of the Factbook and take a minute to tour the entries of a handful of countries: https://www.cia.gov/library/publications/the-world-factbook/

Now view the source of these pages.

Notice that the country names are available from a drop-down list, which links to the country's corresponding page.



```
<div class="option_table_wrapper">
    <form action="#" method="GET">
        <select name="selecter_links" class="selecter_links">
        <option value="">Please select a country to view</option>

                <option value="geos/xx.html"> World </option>

                <option value="geos/af.html"> Afghanistan </option>

                <option value="geos/ax.html"> Akrotiri </option>

                <option value="geos/al.html"> Albania </option>

                <option value="geos/ag.html"> Algeria </option>
```

SCRAPING

If the search term provided is found in this drop-down list, we only need to navigate to its corresponding page and pull the information on that country.

If the search term is one of the approved keywords, we need to navigate to every one of these pages and pull the corresponding information. Note the approved keywords must be mapped to the actual keywords used on the website.

Otherwise, we have to report an error.

```
keywords = {"Area": "Area", "Population": "Population",
"Capital": "Capital", "Languages": "Languages", "Currency":
"Exchange rates"}
```

SCRAPING

`requests.get(url)` returns a Response object with all of the important info about the page including its source.

factbook_scraper.py

```
from __future__ import print_function
import requests

url = "https://www.cia.gov/library/publications/the-world-factbook/"
keywords = {"Area": "Area", "Population": "Population", "Capital":
"Capital", "Languages": "Languages", "Currency": "Exchange rates"}

def get_info(search_term):
    main_page = requests.get(url)
    if search_term in keywords:
        return get_keyword(search_term, main_page)
    else:
        # Look for country name on main page!
```

SCRAPING CRAIGSLIST

To search the country names, we'll need to know a bit about the source. After viewing the source on the page, we find the following:

```
<div class="option_table_wrapper">
   <form action="#" method="GET">
      <select name="selecter_links" class="selecter_links">
      <option value="">Please select a country to view</option>
            <option value="geos/xx.html"> World </option>
            <option value="geos/af.html"> Afghanistan </option>
            <option value="geos/ax.html"> Akrotiri </option>
            <option value="geos/al.html"> Albania </option>
            <option value="geos/ag.html"> Algeria </option>
            …
```

SCRAPING CRAIGSLIST

The structure of a link to a country page is very systematic. We can take advantage of this to pull out the information we need.

```
<div class="option_table_wrapper">
   <form action="#" method="GET">
      <select name="selecter_links" class="selecter_links">
      <option value="">Please select a country to view</option>
            <option value="geos/xx.html"> World </option>
            <option value="geos/af.html"> Afghanistan </option>
            <option value="geos/ax.html"> Akrotiri </option>
            <option value="geos/al.html"> Albania </option>
            <option value="geos/ag.html"> Algeria </option>
            …
```

SCRAPING

```
from __future__ import print_function
import requests, re, sys

def get_info(search_term):
    main_page = requests.get(url)
    if search_term in keywords:
         return get_keyword(search_term, main_page)
    else:
        link = re.search(r'<option value="([^"]+)"> ' +
                              search_term.capitalize() + ' </option>', main_page.text)
        if link:
            # Country link found!
            pass
        else:
            return None
```

We will use regular expressions to pull out the link associated with the search term, but only if it can be found as a country name.

SCRAPING

```python
from __future__ import print_function
import requests, re, sys

def get_info(search_term):
    main_page = requests.get(url)
    if search_term in keywords:
         return get_keyword(search_term, main_page)
    else:
        link = re.search(r'<option value="([^>]+)"> ' +
                         search_term.capitalize() + ' </option>', main_page.text)
        if link:
            country_page = requests.get(url + link.group(1))
            return get_country(country_page)
        else:
            return None
```

Now, we build the URL of the country's page, if it exists, and request that url to be passed into our get_country function.

SCRAPING

Now, how does the data appear on a country's page?

```
<div id='field' class='category sas_light' style='padding-left:5px;'>
  <a href='../docs/notesanddefs.html?fieldkey=2119&term=Population'>Population:</a>
  <a href='../fields/2119.html#af'><img src='../graphics/field_listing_on.gif'></a>
</div>
<div class=category_data>32,564,342 (July 2015 est.)</div>
```

Essentially, the keyword appears within a div tag with class "category sas_light" while the data associated with the keyword appears in the subsequent div tag with class "category_data".

This is a little inconvenient because the data is not nested *within* some identifying header or attribute. It simply appears *after* the corresponding keyword identifier.

SCRAPING

Now, how does the data appear on a country’s page?

```
<div id='field' class='category sas_light' style='padding-left:5px;'>
  <a href='../docs/notesanddefs.html?fieldkey=2119&term=Population'>Population:</a>
  <a href='../fields/2119.html#af'><img src='../graphics/field_listing_on.gif'></a>
</div>
<div class=category_data>32,564,342 (July 2015 est.)</div>
```

We can try to use regular expressions here to describe the data pattern.

```
keyword + r“:.*?<div class=category_data>(.+?)</div>”
```

SCRAPING

```
def get_country(country_page):
    results = []
    for keyword in keywords.values():
        data = get_country_by_keyword(country_page, keyword)
        if data:
            results.append((keyword, data))
    return results
```

Now, unfortunately, not all of the data appears in a uniform fashion. We'll have to uniquely access some of it based on what we're looking for.

SCRAPING

```
def get_country_by_keyword(country_page, keyword):
    data = re.search(keyword + r":.*?<div class=category_data>(.+?)</div>",
                     country_page.text, re.S)
    if data:
        return data.group(1)
    return None
```

Take a look at the factbook_scraper.py to see what we *really* have to do to grab this data. It's a bit more complicated!

SCRAPING

We've taken care of scraping data for a single country. Now what about scraping all of the data for a specific keyword?

```python
def get_keyword(search_term, main_page):
    results = []
    keyword = keywords[search_term]
    link_pairs = re.findall(r'<option value="([^>]+)"> (.*?) </option>',
                            main_page.text)
    for link, country in link_pairs:
        country_page = requests.get(url + link)
        data = get_country_by_keyword(country_page, keyword)
        results.append((country, data))
    return results
```

SCRAPING FACTBOOK

The entire contents of factbook_scraper.py is posted next to the lecture slides. Let's give it a try!

Now, we have a script which will take in a single search term and return to us some information about either a single country, or about all of the countries in the world.

Next lecture, we'll start building a website around this script.

CRAWLING AND SCRAPING

Note that while we used regular expressions and requests in this example, there are a number of ways to perform crawling and scraping since it is such a common use of Python. Some of the modules below are more powerful than the tools we learned this lecture but have a steeper learning curve.

- Scrapy
- BeautifulSoup
- RoboBrowser
- lxml