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

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

GETTING STARTED

First, let's introduce the website concept. We will be building a website which allows a user to compare ticket prices for various local events.

When you search for event tickets on Craigslist, you can sort by price. However, sometimes the price is per ticket and sometimes it is for a set – rendering sorting useless.

On our website, the user can enter search for an event and location and they will be given a list of ticket postings on Craigslist, ordered by price per ticket.

Today, we will be implementing a script which gathers data from craigslist and extracts the required information.

CRAWLING AND SCRAPING

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

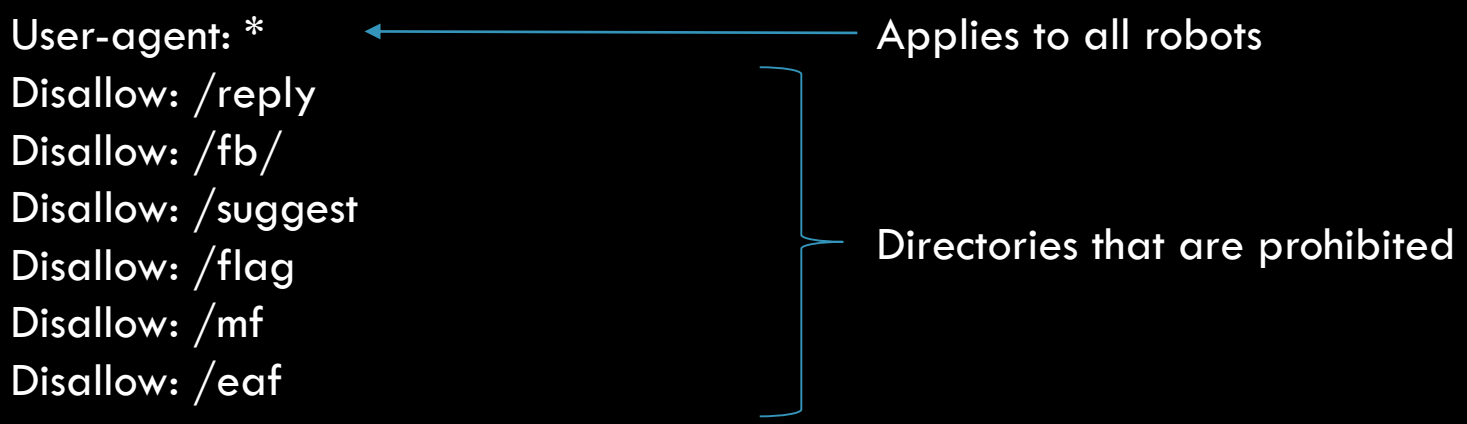
- Crawling: the process of iteratively finding links within a page and fetching the corresponding linked page, given some set of root pages to start from.

- 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 ticket information from Craigslist, let's check out craigslist.org/robots.txt:



SCRAPING CRAIGSLIST

Let's talk about the objectives of our scraper.

- Input: event, location.

- Returns: sorted list of post information as (date, venue, # tickets, title, price/ticket).

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 ticket information.

SCRAPING CRAIGSLIST

Let's write some basic code to get started.

We'll be calling `get_posts()` from our application, but we include some code for initializing `event_name` and `location` when the module is run by itself.

The very first thing we need to do is access the search results page. We'll be using the requests module to do this.

```
def get_posts(event_name, location):
    pass
if __name__ == "__main__":
    event_name = raw_input("Event to search for: ")
    location = raw_input("Location of event: ")
    get_posts(event_name, location)
```

SCRAPING CRAIGSLIST

ticket_scraper.py

```
import requests

url = ".craigslist.org/search/tia?sort=date&query="

def get_posts(event_name, location):
    ev = '+'.join(event_name.split())
    page = requests.get("http://" + str(location.lower()) + url + ev)

if __name__ == "__main__":
    event_name = raw_input("Event to search for: ")
    location = raw_input("Location of event: ")
    get_posts(event_name, location)
```

When we make a search for tickets on Craigslist, we can see that the URL looks something like this:
http://tallahassee.craigslist.org/search/tia?sort=date&query=taylor+swift

SCRAPING CRAIGSLIST

ticket_scraper.py

```
import requests

url = ".craigslist.org/search/tia?sort=date&query="

def get_posts(event_name, location):
    ev = '+'.join(event_name.split())
    page = requests.get("http://" + str(location.lower()) + url + ev)

if __name__ == "__main__":
    event_name = raw_input("Event to search for: ")
    location = raw_input("Location of event: ")
    get_posts(event_name, location)
```

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

SCRAPING CRAIGSLIST

ticket_scraper.py

```
import requests
from lxml import html

url = ".craigslist.org/search/tia?sort=date&query="

def get_posts(event_name, location):
    ev = '+'.join(event_name.split())
    page = requests.get("http://" + str(location.lower()) + url + ev)
    tree = html.fromstring(page.text)

if __name__ == "__main__":
    event_name = raw_input("Event to search for: ")
    location = raw_input("Location of event: ")
    get_posts(event_name, location)
```

The lxml module comes with a dedicated html parser.

html.fromstring(html_source) will create a nice tree structure of the html, which we can traverse to find elements of interest.

SCRAPING CRAIGSLIST

ticket_scraper.py

```
import requests
from lxml import html

url = ".craigslist.org/search/tia?sort=date&query="

def get_posts(event_name, location):
    ev = '+'.join(event_name.split())
    page = requests.get("http://" + str(location.lower()) + url + ev)
    tree = html.fromstring(page.text)
    post_pages = get_pages(tree, location)

if __name__ == "__main__":
    event_name = raw_input("Event to search for: ")
    location = raw_input("Location of event: ")
    get_posts(event_name, location)
```

Now that we have our search results in a nice tree structure, we can find links to the posts, request those pages and create trees out of their source.

SCRAPING CRAIGSLIST

To extract the post URLs, we'll need to know a bit about the source. After viewing the source on the page, we find the following embedded:

```
<p class="row" data-pid="4852285511"> <a href="/tix/4852285511.html"
class="i"><span class="price">&#x0024;800</span></a> <span
class="txt"> <span class="star"></span> <span class="pl"> <time
datetime="2015-01-18 10:56" title="Sun 18 Jan 10:56:26 AM (13 days
ago)">Jan 18</time> <a href="/tix/4852285511.html" data-
id="4852285511" class="hdrlnk">TAYLOR SWIFT TICKETS!</a> </span> <span
class="l2"> <span class="price">&#x0024;800</span> <span class="pnr">
<small> (Midtown)</small> <span class="px"> <span class="p">
</span></span> </span> <a class="gc" href="/tix/" data-
cat="tix">tickets - by owner</a> </span> </span>
```

SCRAPING CRAIGSLIST

All of the post links are embedded in this way. So we just need to figure out how to systematically grab the links.

```
<p class="row" data-pid="4852285511"> <a href="/tix/4852285511.html"
class="i"><span class="price">&#x0024;800</span></a> <span
class="txt"> <span class="star"></span> <span class="pl"> <time
datetime="2015-01-18 10:56" title="Sun 18 Jan 10:56:26 AM (13 days
ago)">Jan 18</time> <a href="/tix/4852285511.html" data-
id="4852285511" class="hdrlnk">TAYLOR SWIFT TICKETS!</a> </span> <span
class="l2"> <span class="price">&#x0024;800</span> <span class="pnr">
<small> (Midtown)</small> <span class="px"> <span class="p">
</span></span> </span> <a class="gc" href="/tix/" data-
cat="tix">tickets - by owner</a> </span> </span>
```

We'll use an xpath query, which is used to specify elements of an html document.

SCRAPING CRAIGSLIST

Here's our path expression: `//p[@class='row']/a[@class='i']/@href`

- Navigate to any p tag with class = row.
- Find any child a tags with class = i.
- Grab the href attribute.

```
<p class="row" data-pid="4852285511"> <a href="/tix/4852285511.html"
class="i"><span class="price">&#x0024;800</span></a> <span
class="txt"> <span class="star"></span> <span class="pl"> <time
datetime="2015-01-18 10:56" title="Sun 18 Jan 10:56:26 AM (13 days
ago)">Jan 18</time> <a href="/tix/4852285511.html" data-
id="4852285511" class="hdrlnk">TAYLOR SWIFT TICKETS!</a> </span> <span
class="12"> <span class="price">&#x0024;800</span> <span class="pnr">
<small> (Midtown)</small> <span class="px"> <span class="p">
</span></span> </span> <a class="gc" href="/tix/" data-
cat="tix">tickets - by owner</a> </span> </span>
```

SCRAPING CRAIGSLIST

Here's our path expression: `//p[@class='row']/a[@class='i']/@href`

You can find more information about creating XPath expressions on the w3schools site, or you can use a tool like Firebug to automatically generate the XPath of an element.

```
<p class="row" data-pid="4852285511"> <a href="/tix/4852285511.html"
class="i"><span class="price">&#x0024;800</span></a> <span
class="txt"> <span class="star"></span> <span class="pl"> <time
datetime="2015-01-18 10:56" title="Sun 18 Jan 10:56:26 AM (13 days
ago)">Jan 18</time> <a href="/tix/4852285511.html" data-
id="4852285511" class="hdrlnk">TAYLOR SWIFT TICKETS!</a> </span> <span
class="l2"> <span class="price">&#x0024;800</span> <span class="pnr">
<small> (Midtown)</small> <span class="px"> <span class="p">
</span></span> </span> <a class="gc" href="/tix/" data-
cat="tix">tickets - by owner</a> </span> </span>
```

SCRAPING CRAIGSLIST

```
def get_pages(root, location):
    post_urls = root.xpath("//p[@class='row']/a[@class='i']/@href")
    trees = []
    for i in range(len(post_urls)):
        if not post_urls[i].startswith('http'):
            post_urls[i] = "http://" + str(location) +
                                ".craigslist.org" + post_urls[i]
        page = requests.get(post_urls[i])
        tr = html.fromstring(page.text)
        trees.append(tr)
    return trees
```

Calling the `xpath(path_expr)` method on our tree will return a list of data that matched the expression.

SCRAPING CRAIGSLIST

```
def get_pages(root, location):
    post_urls = root.xpath("//p[@class='row']/a[@class='i']/@href")
    trees = []
    for i in range(len(post_urls)):
        if not post_urls[i].startswith('http'):
            post_urls[i] = "http://" + str(location) +
                           ".craigslist.org" + post_urls[i]
        page = requests.get(post_urls[i])
        tr = html.fromstring(page.text)
        trees.append(tr)
    return trees
```

Some URLs are full URLs, but local results only return the suffix of the URL so we need to construct a full URL.

SCRAPING CRAIGSLIST

```
def get_pages(root, location):
    post_urls = root.xpath("//p[@class='row']/a[@class='i']/@href")
    trees = []
    for i in range(len(post_urls)):
        if not post_urls[i].startswith('http'):
            post_urls[i] = "http://" + str(location) +
                                ".craigslist.org" + post_urls[i]
        page = requests.get(post_urls[i])
        tr = html.fromstring(page.text)
        trees.append(tr)
    return trees
```

After we've scraped all of the post URLs, we grab the page and make a tree from it.

SCRAPING CRAIGSLIST

```
def get_pages(root, location):
    post_urls = root.xpath("//p[@class='row']/a[@class='i']/@href")
    trees = []
    for i in range(len(post_urls)):
        if not post_urls[i].startswith('http'):
            post_urls[i] = "http://" + str(location) +
                                ".craigslist.org" + post_urls[i]
        page = requests.get(post_urls[i])
        tr = html.fromstring(page.text)
        trees.append(tr)
    return trees
```

Now we return a list of all of the trees constructed from the post links.

SCRAPING CRAIGSLIST

ticket_scraper.py

```
import requests
from lxml import html

url = ".craigslist.org/search/tia?sort=date&query="

def get_posts(event_name, location):
    ev = '+'.join(event_name.split())
    page = requests.get("http://" + str(location.lower()) + url + ev)
    tree = html.fromstring(page.text)
    post_pages = get_pages(tree, location)
    post_info = get_post_info(posts_pages)

if __name__ == "__main__":
    event_name = raw_input("Event to search for: ")
    location = raw_input("Location of event: ")
    get_posts(event_name, location)
```

Now we'll gather the post information for each of the trees we constructed.

SCRAPING CRAIGSLIST

Again, to gather information from the posts, we'll have to examine the source of the post pages. The first piece of interest:


```
<p class="attrgroup">
<span class="otherpostings">
<a href="/search/tix?catAbb=tix&sale_date=2014-12-21">sunday 2014-
12-21</a>
</span>
<span>venue : <b>Raymond James Stadium</b></span>
<span>number available: <b>2</b></span>
</p>
```

SCRAPING CRAIGSLIST


```
<p class="attrgroup">
<span class="otherpostings">
<a href="/search/tix?catAbb=tix&sale_date=2014-12-21">sunday 2014-
12-21</a>
</span>
<span>venue : <b>Raymond James Stadium</b></span>
<span>number available: <b>2</b></span>
</p>
```

Our path expression: `//p[@class='attrgroup']/span//text()`

SCRAPING CRAIGSLIST


```
<p class="attrgroup">
<span class="otherpostings">
<a href="/search/tix?catAbb=tix&sale_date=2014-12-21">sunday 2014-
12-21</a>
</span>
<span>venue : <b>Raymond James Stadium</b></span>
<span>number available: <b>2</b></span>
</p>
```

```
info = post.xpath("//p[@class='attrgroup']/span//text()")
```

Will return [‘sunday 2014-12-21’, ‘venue : ‘, ‘Raymond James Stadium’, ‘number available : ‘, ‘2’]

SCRAPING CRAIGSLIST


```
<p class="attrgroup">
<span class="otherpostings">
<a href="/search/tix?catAbb=tix&sale_date=2014-12-21">sunday 2014-
12-21</a>
</span>
<span>venue : <b>Raymond James Stadium</b></span>
<span>number available: <b>2</b></span>
</p>
```

```
info = post.xpath("//p[@class='attrgroup']/span//text()")
date = info[0]
for i in range(len(info)):
      if info[i].startswith('venue'):
            venue = info[i+1]
      elif info[i].startswith('number available:'):
            tix = info[i+1]
```

SCRAPING CRAIGSLIST

So now we have the date of the post, the venue (if applicable), and number of tickets. Now we need to get the post title and price.


```
<h2 class="postingtitle">
  <span class="star"></span>
  <span class="postingtitletext">Taylor Swift FLOOR Tix
      - Best Offer Wins - $900 -
      <span class="price">&#x0024;900</span>…
```

Our path expression for title:

```
//h2[@class='postingtitle']/span[@class='postingtitletext']/text()
```

SCRAPING CRAIGSLIST


```
<h2 class="postingtitle">
  <span class="star"></span>
  <span class="postingtitletext”>Taylor Swift FLOOR Tix
      - Best Offer Wins - $900 -
      <span class=“price”>&#x0024;900</span>…
```

```
title = post.xpath("//h2[@class='postingtitle']
                   /span[@class='postingtitletext']/text()")
title = title[0]
```

Will return [‘Taylor Swift FLOOR Tix - Best Offer Wins - $900 - ’]

SCRAPING CRAIGSLIST


```
<h2 class="postingtitle">
  <span class="star"></span>
  <span class=“postingtitletext”>Taylor Swift FLOOR Tix
      - Best Offer Wins - $900 -
      <span class=“price”>&#x0024;900</span>…
```

To find the price, we can use the following xpath:

```
//h2[@class='postingtitle']/span[@class='postingtitletext']/
span[@class='price']/text()
```

SCRAPING CRAIGSLIST


```
<h2 class="postingtitle">
  <span class="star"></span>
  <span class="postingtitletext”>Taylor Swift FLOOR Tix
      - Best Offer Wins - $900 -
      <span class="price”>&#x0024;900</span>…
```

To find the price, we can use the following xpath:

```
price = post.xpath("//h2[@class='postingtitle']/span[@class=
                    'postingtitletext']/span[@class='price']/text()")
price = price[0].strip('$')
```

SCRAPING CRAIGSLIST

Now we have the date, venue, # of tickets, title, and listed price. What makes our idea useful however is that we will try to sort by price/ticket rather than the listed price. Here's how we'll do it:

- Search the post body for common phrases that indicate price per ticket (e.g. $50 each ticket, $50 per ticket, $50/each, etc.). When found, use the price indicated as the price per ticket.

- Else, search the post body for common phrases the indicate the price for all of the tickets (e.g. "pair for $50", "all 3 for $50", etc). When found, divide the price indicated by number of tickets listed.

- Otherwise, divide the listed price by number of tickets listed.

SCRAPING CRAIGSLIST

```
post_body = post.xpath("//section[@id='postingbody']/text()")

m = re.search(r"\$([\d]+)(/| each| per| a)", ''.join(post_body))
m2 = re.search(r"(both|pair|all|all [\d]+) for \$([\d]+)", ''.join(post_body))

if not m is None:
    if int(m.group(1)) < int(price):
        price = m.group(1)
elif not m2 is None:
    if int(m2.group(2)) > int(price):
        price = m2.group(2)
        price = str(int(price)/int(tix))
elif not price == '1':
    price = str(int(price)/int(tix))

post_info.append((date, venue, tix, title, price))
```

Grab the text of the post body to analyze for ticket prices.

SCRAPING CRAIGSLIST

```
post_body = post.xpath("//section[@id='postingbody']/text()")

m = re.search(r"\$([\d]+)(/| each| per| a)", ''.join(post_body))
m2 = re.search(r"(both|pair|all|all [\d]+) for \$([\d]+)", ''.join(post_body))

if not m is None:
    if int(m.group(1)) < int(price):
        price = m.group(1)
elif not m2 is None:
    if int(m2.group(2)) > int(price):
        price = m2.group(2)
        price = str(int(price)/int(tix))
elif not price == '1':
    price = str(int(price)/int(tix))

post_info.append((date, venue, tix, title, price))
```

Search the text for phrases indicating the price per ticket and create a MatchObject m.

Also search for phrases indicating the price for the set and create a MatchObject m2.

SCRAPING CRAIGSLIST

```
post_body = post.xpath("//section[@id='postingbody']/text()")

m = re.search(r"\$([\d]+)(/| each| per| a)", ''.join(post_body))
m2 = re.search(r"(both|pair|all|all [\d]+) for \$([\d]+)", ''.join(post_body))

if not m is None:
    if int(m.group(1)) < int(price):
        price = m.group(1)
elif not m2 is None:
    if int(m2.group(2)) > int(price):
        price = m2.group(2)
        price = str(int(price)/int(tix))
elif not price == '1':
    price = str(int(price)/int(tix))

post_info.append((date, venue, tix, title, price))
```

Replace the price if we match a price per ticket phrase.

SCRAPING CRAIGSLIST

```
post_body = post.xpath("//section[@id='postingbody']/text()")

m = re.search(r"\$([\d]+)(/| each| per| a)", ''.join(post_body))
m2 = re.search(r"(both|pair|all|all [\d]+) for \$([\d]+)", ''.join(post_body))

if not m is None:
    if int(m.group(1)) < int(price):
        price = m.group(1)
elif not m2 is None:
    if int(m2.group(2)) > int(price):
        price = m2.group(2)
        price = str(int(price)/int(tix))
elif not price == '1':
    price = str(int(price)/int(tix))

post_info.append((date, venue, tix, title, price))
```

Replace the price if we saw a phrase indicating the price for the set. Then divide the price by the number of tickets listed.

SCRAPING CRAIGSLIST

```
post_body = post.xpath("//section[@id='postingbody']/text()")

m = re.search(r"\$([\d]+)(/| each| per| a)", ''.join(post_body))
m2 = re.search(r"(both|pair|all|all [\d]+) for \$([\d]+)", ''.join(post_body))

if not m is None:
    if int(m.group(1)) < int(price):
        price = m.group(1)
elif not m2 is None:
    if int(m2.group(2)) > int(price):
        price = m2.group(2)
        price = str(int(price)/int(tix))
elif not price == '1':
    price = str(int(price)/int(tix))

post_info.append((date, venue, tix, title, price))
```

If we couldn't find any information in the post body, just divide the listed price by the listed number of tickets.

SCRAPING CRAIGSLIST

```
post_body = post.xpath("//section[@id='postingbody']/text()")

m = re.search(r"\$([\d]+)(/| each| per| a)", ''.join(post_body))
m2 = re.search(r"(both|pair|all|all [\d]+) for \$([\d]+)", ''.join(post_body))

if not m is None:
    if int(m.group(1)) < int(price):
        price = m.group(1)
elif not m2 is None:
    if int(m2.group(2)) > int(price):
        price = m2.group(2)
        price = str(int(price)/int(tix))
elif not price == '1':
    price = str(int(price)/int(tix))

post_info.append((date, venue, tix, title, price))
```

Append a tuple containing all of the information from the post.

SCRAPING CRAIGSLIST

The entire contents of ticket_scraper.py is posted next to the lecture slides.

Now, we have a script which will take in an event and location and return a list of post information, sorted by price per ticket.

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

CRAWLING AND SCRAPING

Note that while we used lxml 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