Big Data - Foundations and Applications

Lesson #11 - Importing Data from the Web

Ivanovitch Silva

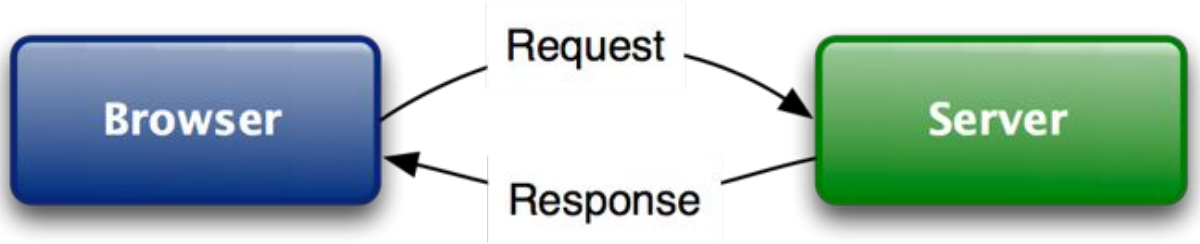
October, 2017

Agenda

- Use the urllib and requests packages
- Make HTTP requests (GET requests)
- Scrape web data such as HTML
- Parse HTML into useful data (BeautifulSoup)

HyperText Transfer Protocol (HTTP)

- Foundation of data communication for the web
- HTTP is the protocol that is used by web servers and browsers to communicate.
- HTTP is based on a request and a response.



GET requests using *urllib*

```
from urllib.request import urlopen, Request

url = "https://www.wikipedia.org/"
request = Request(url)
response = urlopen(request)
html = response.read()
response.close()
```

#print the type of response
#change the url to http://portal.imd.ufrn.br

GET requests using a higher-level request lib.

- One of the most downloaded Python packages

Used by: Twitter, Spotify, Microsoft, Amazon, Lyft, BuzzFeed, Reddit, The NSA, Her Majesty's Government, Google, Twilio, Runscope, Mozilla, Heroku, PayPal, NPR, Obama for America, Transifex, Native Instruments, The Washington Post, SoundCloud, Kippt, Sony, and Federal U.S. Institutions that prefer to be unnamed claim to use Requests internally.

http://docs.python-requests.org/

Get requests using "requests"

```
# Import package
import requests

# Specify the url: url
url = "https://www.wikipedia.org/"

# Packages the request, send the request
# and catch the response:
response = requests.get(url)

# Extract the response: text
text = response.text
```

Understanding status code

```
response.status_code
```

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

- 200 - Everything went okay, and the server returned a result (if any).
- 301 - The server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint's name has changed.
- 401 - The server thinks you're not authenticated. This happens when you don't send the right credentials to access an API (we'll talk about this in a later mission).
- 400 - The server thinks you made a bad request. This can happen when you don't send the information the API requires to process your request, among other things.
- 403 - The resource you're trying to access is forbidden; you don't have the right permissions to see it.
- 404 - The server didn't find the resource you tried to access.

Headers

- The server sends more than a status code and the data when it generates a response.
- It also sends metadata containing information on how it generated the data and how to decode it.
- This information appears in the response headers.

```
response.headers
```

Web Scraping

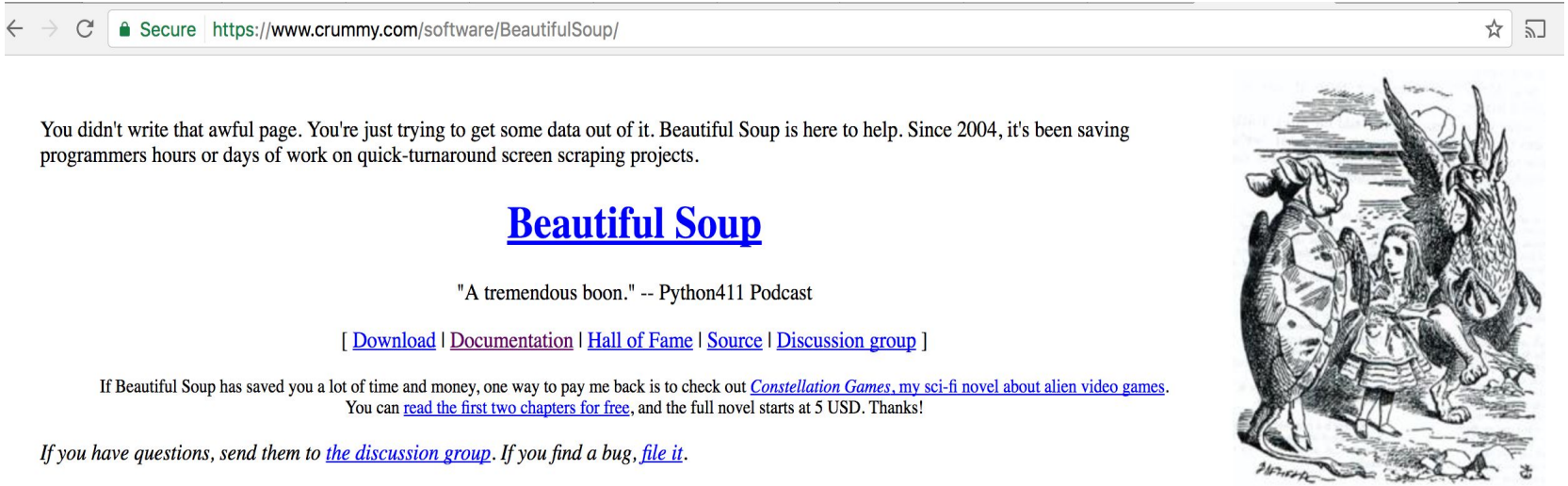
- A lot of data aren't accessible through datasets or APIs.
- One way to access the data without waiting for the provider to create an API is to use a technique called **Web scraping**

Scraping the Web: what might someone do with this kind of data?

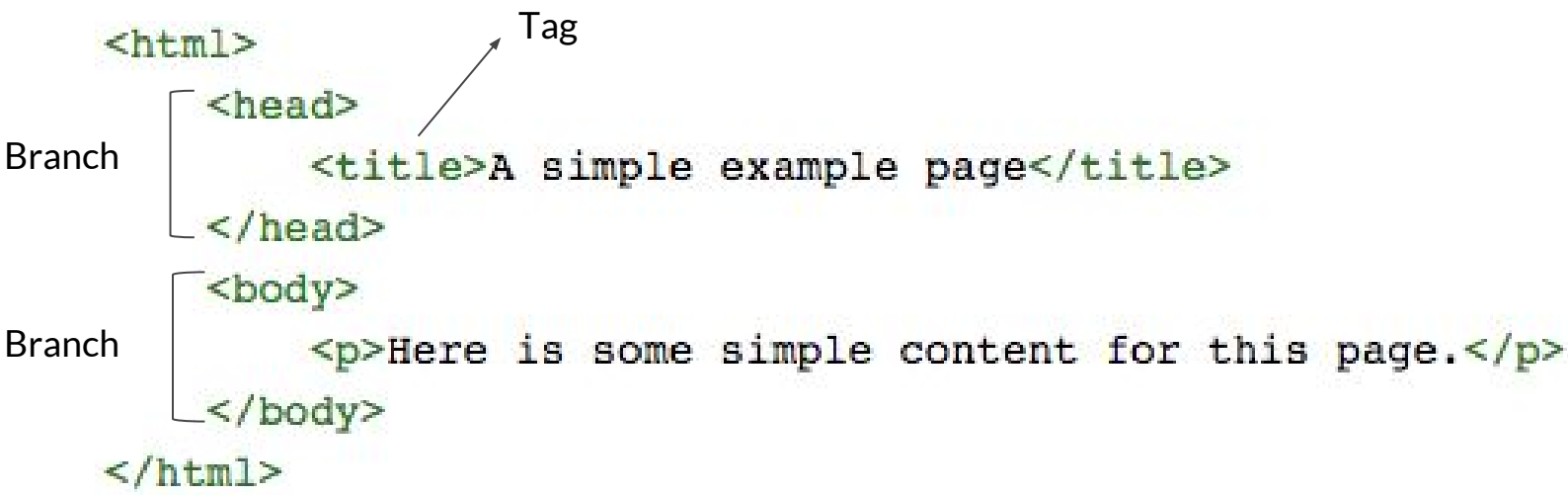
- Find email addresses proximate to certain keywords for spamming purposes
- Imagine, for example, a single site that aggregates (illicitly, probably) raw content from a dozen other websites
- Harvest stats from government websites
- Scan listings from multiple job sites for search strings
- Perform sentiment analysis on blog sites from a variety of platforms
- Monitor price fluctuations among many different web retailers for a specific product
- There's really no end to it ...

Beautiful Soup

- Parse and extract structured data from HTML
- To the Internet, a webpage is just a SOUP of text, symbols, and whitespace



Web page structure



- We can think of HTML documents as "trees," and the nested tags as "branches" (similar to a family tree).
- BeautifulSoup works the same way.

Beautiful Soup #1 (simple.html)

```
# import package
import requests
from bs4 import BeautifulSoup

# specify the url
url = 'https://nbviewer.jupyter.org/github/ivanovitchm/\
EEC2006/blob/master/Lesson%2011/html/simple.html'

# packages the request, send the request and catch the response
response = requests.get(url)

# extract the content
content = response.content
```

conda install -c anaconda beautifulsoup4

Beautiful Soup #2
(Retrieving Elements From A Page)

```
# Initialize the parser, and pass in the content we grabbed earlier.
parser = BeautifulSoup(content, 'html.parser')

# Get the body tag from the document.
# Since we passed in the top level of the document to the parser,
# we need to pick a branch off of the root.
# With BeautifulSoup, we can access branches by using tag types as attributes.
body = parser.body

# Get the p tag from the body.
p = body.p

# Print the text inside the p tag.
print(p.text)
```

Using **Find All**

- While it's nice to use the tag type as a property, it's not always a very robust way to parse a document.
- It's usually better to be more explicit by using the find_all method.
- This method will find all occurrences of a tag in the current element, and return a list.
- E.g: print all hyperlinks from a page

Beautiful Soup #3 (find_all)

```
# Get a list of all occurrences
# of the body tag in the element.
body = soup.find_all("body")

# Get the paragraph tag.
p = body[0].find_all("p")

# Get the text.
print(p[0].text)
```

Beautiful Soup #4 (find_all)

```
# Get the page content and set up a new parser.
response = requests.get("http://portal.imd.ufrn.br/")
content = response.content
soup = BeautifulSoup(content, 'html.parser')

# Find all 'a' tags (which define hyperlinks): a_tags
a_tags  = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```

Element ID

```
<html>
    <head>
        <title>A simple example page</title>
    </head>
    <body>
        <div>
            <p id="first">
                First paragraph.
            </p>
        </div>
        <p id="second">
            <b>
                Second paragraph.
            </b>
        </p>
    </body>
</html>
```

- HTML allows elements to have IDs. Because **they are unique**, we can use an ID to refer to a specific element.
- HTML uses the div tag to create a divider that splits the page into logical units.
- For example, different dividers hold a Web page's footer, sidebar, and horizontal menu.

Beautiful Soup #5 (element ID)

```
# Get the page content and set up a new parser.
url = 'https://nbviewer.jupyter.org/github/ivanovitchm/\
EEC2006/blob/master/Lesson%2011/html/simple_ids.html'

response = requests.get(url)

# extract the content
content = response.content

# Initialize the parser, and pass in the content we grabbed earlier.
parser = BeautifulSoup(content, 'html.parser')

# Pass in the ID attribute to only get the element with that specific ID.
first_paragraph = parser.find_all("p", id="first")[0]
print(first_paragraph.text)
```

Element Classes

```
<html>
    <head>
        <title>A simple example page</title>
    </head>
    <body>
        <div>
            <p class="inner-text">
                First inner paragraph.
            </p>
            <p class="inner-text">
                Second inner paragraph.
            </p>
        </div>
        <p class="outer-text">
            <b>
                First outer paragraph.
            </b>
        </p>
        <p class="outer-text">
            <b>
                Second outer paragraph.
            </b>
        </p>
    </body>
</html>
```

- In HTML, elements can also have classes.
- Classes aren't globally unique.
- In other words, many different elements belong to the same class, usually because they share a common purpose or characteristic.

Beautiful Soup #6 (classes)

```
# Get the website that contains classes.
url ='https://nbviewer.jupyter.org/github/ivanovitchm/\
EEC2006/blob/master/Lesson%2011/html/simple_classes.html'
response = requests.get(url)
content = response.content
parser = BeautifulSoup(content, 'html.parser')

# Get the first inner paragraph.
# Find all the paragraph tags with the class inner-text.
# Then, take the first element in that list.
first_inner_paragraph = parser.find_all("p", class_="inner-text")[0]
print(first_inner_paragraph.text)
```

Using Selectors

```
<html>
    <head>
        <title>A simple example page</title>
    </head>
    <body>
        <div>
            <p class="inner-text first-item" id="first">
                First paragraph.
            </p>
            <p class="inner-text">
                Second paragraph.
            </p>
        </div>
        <p class="outer-text first-item" id="second">
            <b>
                First outer paragraph.
            </b>
        </p>
        <p class="outer-text">
            <b>
                Second outer paragraph.
            </b>
        </p>
    </body>
</html>
```

- We can use BeautifulSoup's .select method to work with CSS selectors.
- You may have noticed that the same element can have both an ID and a class

Beautiful Soup #7 (selectors)

```
# Get the website that contains classes and IDs.
url = 'https://nbviewer.jupyter.org/github/ivanovitchm/\
EEC2006/blob/master/Lesson%2011/html/ids_and_classes.html'
response = requests.get(url)
content = response.content
parser = BeautifulSoup(content, 'html.parser')

# Select all of the elements that have the first-item class.
first_items = parser.select(".first-item")

# Print the text of the first paragraph (the first element with the first-item class).
print(first_items[0].text)
```

Nesting CSS Selectors

This selector will target any paragraph inside a div tag:

```
div p
```

This selector will target any item inside a div tag that has the class first-item:

```
div .first-item
```

This one is even more specific. It selects any item that's inside a div tag inside a body tag, but only if it also has the ID first:

```
body div #first
```

This selector zeroes in on any items with the ID first that are inside any items with the class first-item:

```
.first-item #first
```

Nesting CSS Selectors

	SEA	NWE
First downs	20	25
Total yards	396	377
Turnovers	1	2
Penalties-yards	7-70	5-36
Total Plays	53	72
Time of Possession	26:14	33:46

```
<table class="stats_table nav_table" id="team_stats">
    <tbody>
        <tr id="teams">
            <th></th>
            <th>SEA</th>
            <th>NWE</th>
        </tr>
        <tr id="first-downs">
            <td>First downs</td>
            <td>20</td>
            <td>25</td>
        </tr>
        <tr id="total-yards">
            <td>Total yards</td>
            <td>396</td>
            <td>377</td>
        </tr>
        <tr id="turnovers">
            <td>Turnovers</td>
            <td>1</td>
            <td>2</td>
        </tr>
        <tr id="penalties">
            <td>Penalties-yards</td>
            <td>7-70</td>
            <td>5-36</td>
        </tr>
        <tr id="total-plays">
            <td>Total Plays</td>
            <td>53</td>
            <td>72</td>
        </tr>
        <tr id="time-of-possession">
            <td>Time of Possession</td>
            <td>26:14</td>
            <td>33:46</td>
        </tr>
    </tbody>
```

Nesting CSS Selectors

```
# Get the Superbowl box score data.
url = 'https://nbviewer.jupyter.org/github/ivanovitchm/\
EEC2006/blob/master/Lesson%2011/html/2014_super_bowl.html'
response = requests.get(url)
content = response.content
parser = BeautifulSoup(content, 'html.parser')

# Find the number of turnovers the Seahawks committed.
turnovers = parser.select("#turnovers")[0]
seahawks_turnovers = turnovers.select("td")[1]
seahawks_turnovers_count = seahawks_turnovers.text
print(seahawks_turnovers_count)
```