Data 2.0 Hackathon
Web Scraping Workshop

Sophie Janaskie
Yale School of Forestry and Environmental Studies
March 17, 2018

What is web scraping?

- A method of extracting information from websites programmatically
- A way to convert semi-structured web data into usable, structured data that we can analyze

	Date	Mean Temp (C)	Max Temp (C)	Min Temp (C)
2	1/1/17	29	32	26
3	1/2/17	28	31	26
4	1/3/17	28	30	26
5	1/4/17	28	31	25
6	1/5/17	29	33	25
7	1/6/17	30	34	26
8	1/7/17	31	35	27
9	1/8/17	31	35	27

Why is it useful, and when to use it?

- No API or readily available data
- Programmatically pull data from the web to save time from manual collection
- Useful for projects and capstones!

“The Internet is one giant API – with a really terrible interface.”

- Ryan Mitchell, author of Web Scraping with Python

Agenda

1. Scraping workflow
2. HTML Basics
3. Pulling the web page with Requests
4. Parsing the HTML with BeautifulSoup
5. Storing your data
6. Other considerations

Web Scraping Workflow

Web Scraping Workflow

Web Scraping Workflow

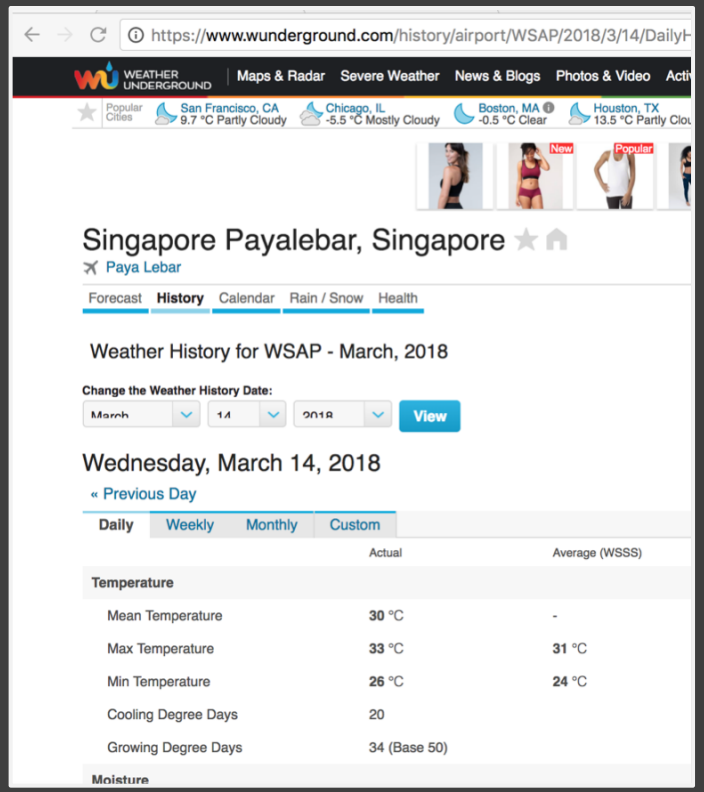
Requests ***BeautifulSoup*** ***csv***

```
import requests
from bs4 import BeautifulSoup
import csv
```

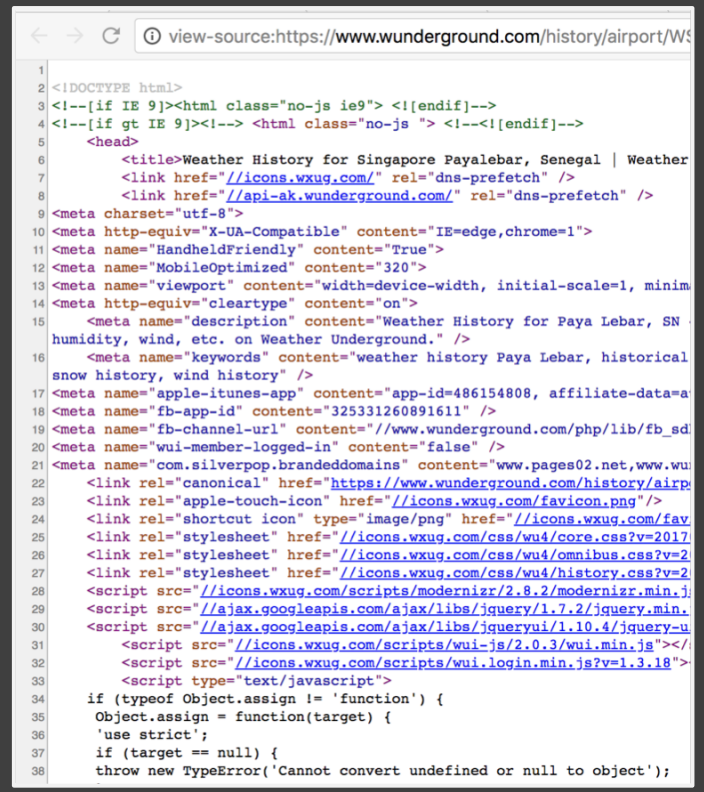
HTML Basics

Web Page Structure

Rendered HTML



Source Code



HTML/CSS

Hyper Text Markup Language provides structure for web pages.
CSS provides styling and design.

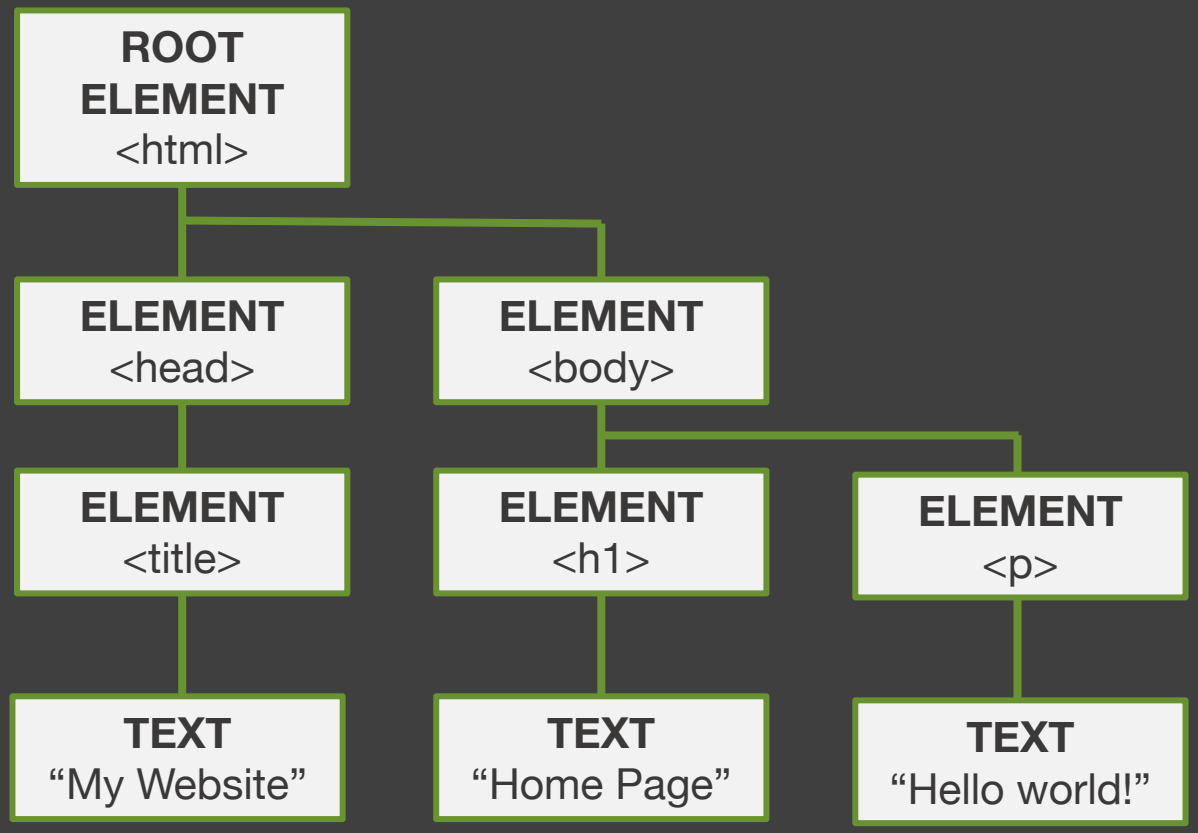
```
<!DOCTYPE html>
<html>
<body>

<h1>This is a heading</h1>
<h2>This is a smaller heading</h2>
<a href="https://www.google.com">This is a link</a>
<img src="picture.jpg" alt="this is a picture" width="100" height="150">

</body>
</html>
```

Document Object Model

The DOM creates an object-oriented model of the information to enable easier manipulation.



Get the website

Using the Requests library

Requests library

- Sending an HTTP request

```
url = "https://www.wunderground.com/history/airport/WSAP/2017/1/1/DailyHistory.html"
response = requests.get(url)
```

- Access the response's body of content

```
content = response.content
```

- Check to make sure the request was successful (200 code)

```
status = response.status_code
```

Parse the HTML

Using the BeautifulSoup library

BeautifulSoup

Make the Soup

```
soup = BeautifulSoup(content, "lxml")
```

Navigate the Tree

- Using .title, .contents, .children, .parent, .next_sibling, .text etc.

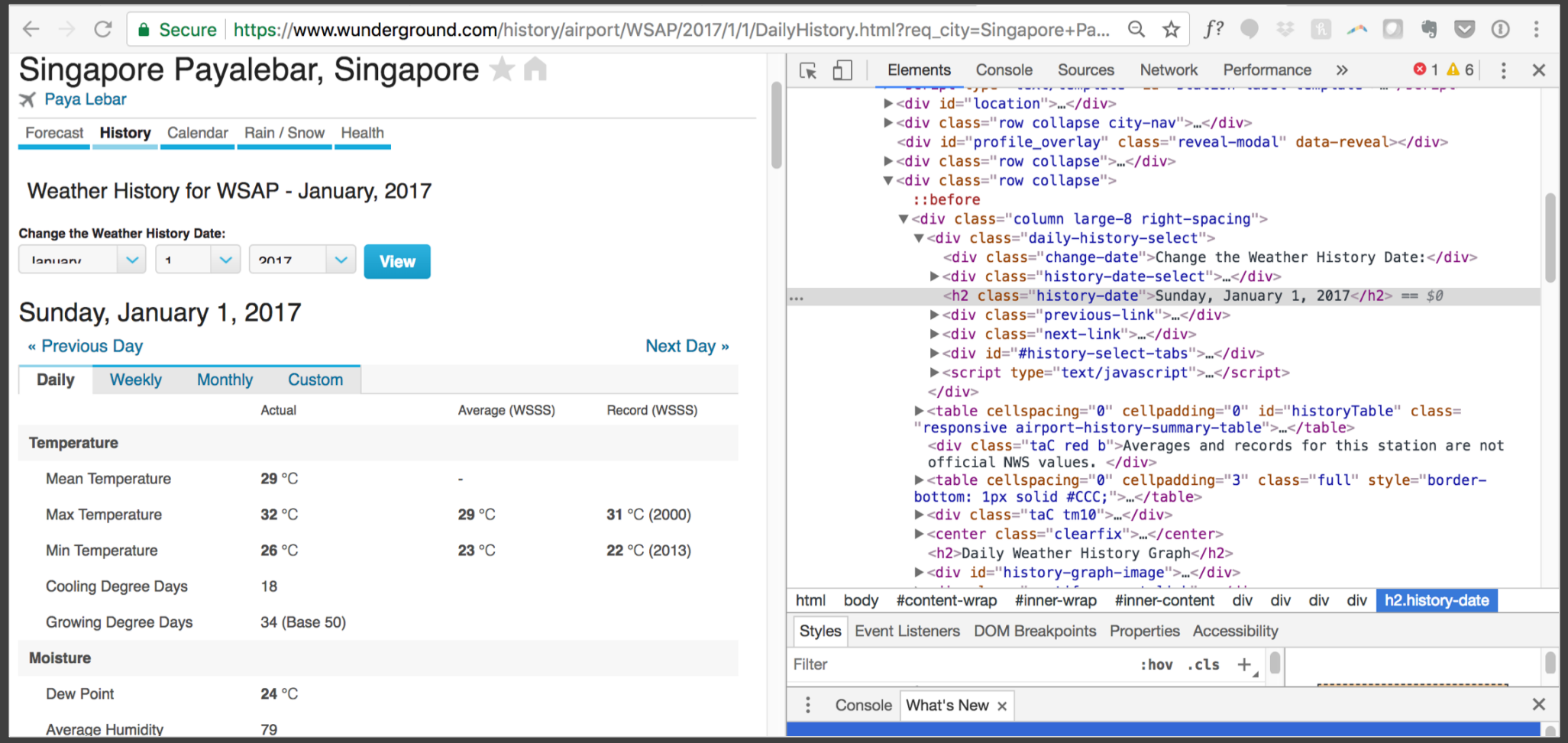
Search the Tree

- Using find( ), find_all( ), select( ), etc.

Modify the Tree

- Using decompose( ), insert( ), NavigableString( ), etc.

Using Inspect Element



Example 1: Wunderground

```
# convert to object model
soup = BeautifulSoup(content, "lxml")

# parse HTML
meantemp = soup.find_all('tr')[2].find_all('td')[1].find_all(attrs={"class":"wx-value"})[0].text
maxtemp = soup.find_all('tr')[3].find_all('td')[1].find_all(attrs={"class":"wx-value"})[0].text
mintemp = soup.find_all('tr')[4].find_all('td')[1].find_all(attrs={"class":"wx-value"})[0].text
precip = soup.find_all('tr')[13].find_all('td')[1].find_all(attrs={"class":"wx-value"})[0].text
wind = soup.find_all('tr')[17].find_all('td')[1].find_all(attrs={"class":"wx-value"})[0].text
```

Example 2: Wunderground

```
nextpagelink = soup.find('div', attrs={'class':'daily-history-select'}).find_all('a')[1].get('href')
```

Store the results

Using the csv library

Example: Wunderground

- *Create a csv file to write your results to.*
- *Create headers.*
- *Write your results in a new row.*

```
# write resutls
with open("WeatherScrape.csv", "w") as file:
    csv_writer = csv.writer(file)
    csv_writer.writerow(['Date', 'Mean Temp (C)', 'Max Temp (C)', 'Min Temp (C)', 'Precipitation (in)', 'Wind Speed (m/s)'])
    csv_writer.writerow([dateString, meantemp, maxtemp, mintemp, precip, wind])
```

```
import requests
from bs4 import BeautifulSoup
import csv, time

# initial GET request
url = "https://www.wunderground.com/history/airport/WSAP/2017/1/1/DailyHistory.html"
response = requests.get(url)
content = response.content

# convert contents to nested data structure
soup = BeautifulSoup(content, "lxml")

# parse HTML
meantemp = soup.find_all('tr')[2].find_all('td')[1].find(attrs={"class":"wx-value"}).text
maxtemp = soup.find_all('tr')[3].find_all('td')[1].find(attrs={"class":"wx-value"}).text
mintemp = soup.find_all('tr')[4].find_all('td')[1].find(attrs={"class":"wx-value"}).text
precip = soup.find_all('tr')[13].find_all('td')[1].find(attrs={"class":"wx-value"}).text
wind = soup.find_all('tr')[17].find_all('td')[1].find(attrs={"class":"wx-value"}).text

# write resutls
with open("WeatherScrape.csv", "w") as file:
    csv_writer = csv.writer(file)
    csv_writer.writerow(['Date', 'Mean Temp (C)', 'Max Temp (C)', 'Min Temp (C)', 'Precipitation (in)', 'Wind Speed (m/s)'])
    csv_writer.writerow([dateString, meantemp, maxtemp, mintemp, precip, wind])

print("Scrape complete!")
```

Scaling up

And other important considerations

Traversing links

- Handling URL generation / pagination

```
# find next link
nextlink = soup.find('div', attrs={'class':'daily-history-select'}).find_all('a')[1].get('href')
url = 'https://www.wunderground.com'+nextlink

# GET request next link
response = requests.get(url)
content = response.content
soup = BeautifulSoup(content, "lxml")
```

- Alternative methods for traversing pages in this scraper

Handling exceptions

- Try/except and if statements
- Filling in N/A's
- Consider special cases

Data Cleaning

- Unnecessary characters may be present in your scraped data
- Can use the .replace( ) function among others to remove these

Review & Results

Thank you!

Resources

- Requests documentation
- BeautifulSoup documentation
- csv documentation