

# 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

```
11<meta name="handheldfriendly" content="true" />
12<meta name="Mobileoptimized" content="320" />
13<meta name="viewport" content="width=device-width, initial-scale=1, minimal-ui">
14<meta http-equiv="cleartype" content="on">
15  <meta name="description" content="Weather History for Paya Lebar, SN - Get Weather History, humidity, wind, etc. on Weather Underground." />
16  <meta name="keywords" content="weather history Paya Lebar, historical weather, weather & snow history, wind history" />
17<meta name="apple-itunes-app" content="app-id=486154808, affiliate-data=at=1010lrYB&ct=websi" />
18<meta name="fb-app-id" content="325331260891611" />
19<meta name="fb-channel-url" content="//www.wunderground.com/php/lib/fb_sdk/channel.php" />
20<meta name="wui-member-logged-in" content="false" />
21<meta name="com.silverpop.brandeddomains" content="www.pages02.net, www.wunderground.com" />
22  <link rel="canonical" href="https://www.wunderground.com/history/airport/WSAP/2018/3/14/"/>
23  <link rel="apple-touch-icon" href="//icons.wxug.com/favicon.png" />
24  <link rel="shortcut icon" type="image/png" href="//icons.wxug.com/favicon.png" />
25  <link rel="stylesheet" href="//icons.wxug.com/css/wu4/core.css?v=2017041301" />
26  <link rel="stylesheet" href="//icons.wxug.com/css/wu4/omnibus.css?v=2018022701" />
27  <link rel="stylesheet" href="//icons.wxug.com/css/history.css?v=2015100201" />
28  <script src="//icons.wxug.com/scripts/modernizr/2.8.2/modernizr.min.js"></script>
29  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
30  <script src="//ajax.googleapis.com/ajax/libs/jqueryui/1.10.4/jquery-ui.min.js"></script>
31  <script src="//icons.wxug.com/scripts/wui-2.0.3/wui.min.js"></script>
32  <script src="//icons.wxug.com/scripts/wui.login.min.js?v=1.3.18"></script>
33  <script type="text/javascript">
34  if (typeof Object.assign != 'function') {
35    Object.assign = function(target) {
36      'use strict';
37      let s = arguments.length;
38      let i = 1;
39      let source = Array.prototype.slice.call(arguments);
40      while (i < s) {
41        target = Object.assign(target, source[i]);
42        i++;
43      }
44      return target;
45    };
46  }
47
```



1	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!

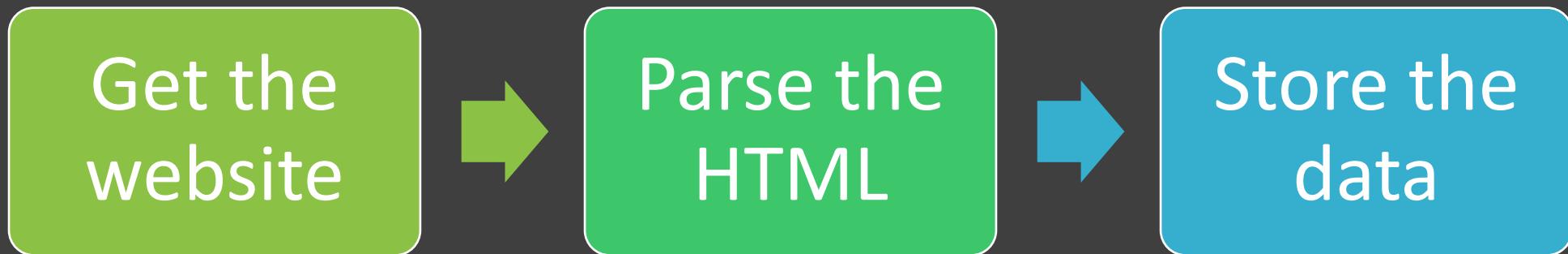
*“The Internet is one giant API – with  
a really terrible interface.”*

- 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

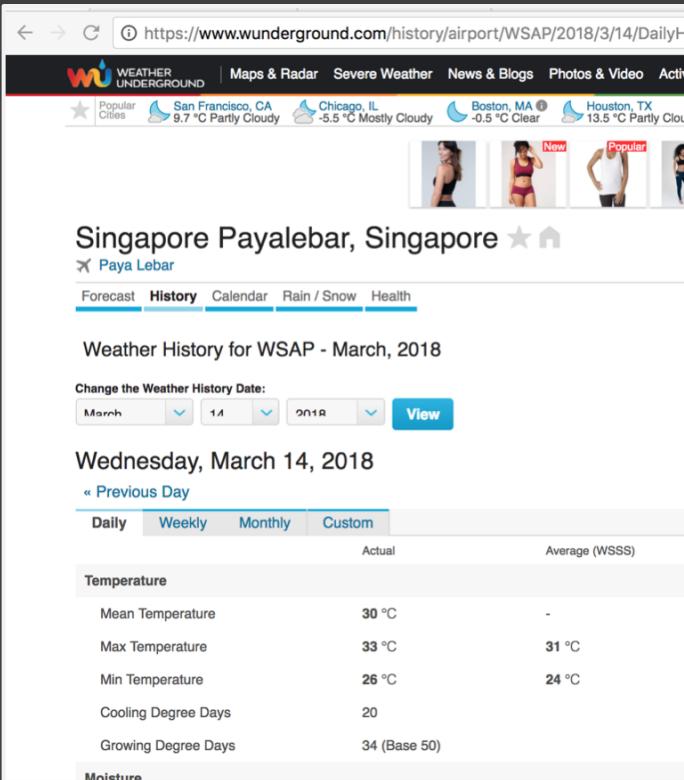


```
import requests  
from bs4 import BeautifulSoup  
import csv
```

# 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.*

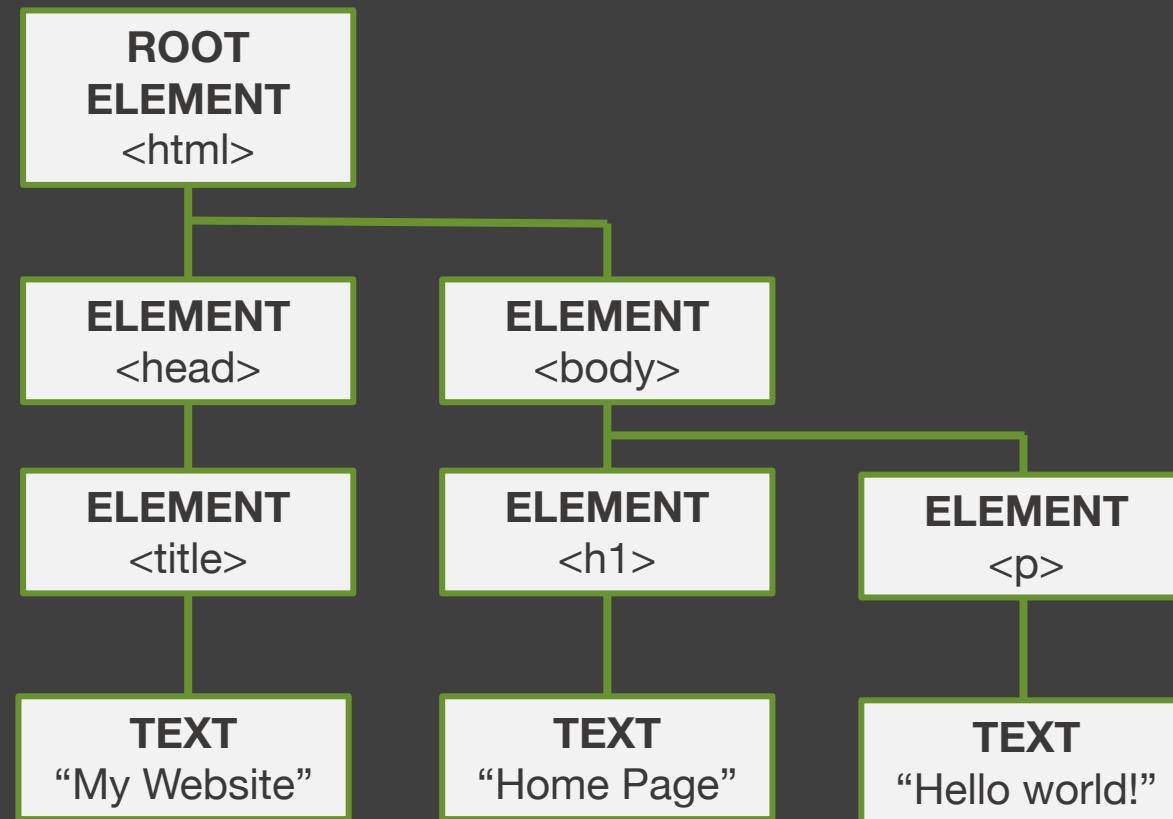
```
<!DOCTYPE html>
<html>
<body>

<h1>This is a heading</h1>
<h2>This is a smaller heading</h2>
<a href="https://www.google.com">This is a link</a>


</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

The screenshot shows a web browser window displaying the weather history for WSAP in Singapore Payalebar for January 2017. The browser's developer tools are open, specifically the 'Elements' tab of the 'Inspector' panel. The 'Elements' tab shows the DOM structure of the page, with the element `<h2 class="history-date">Sunday, January 1, 2017</h2>` selected. The 'Styles' tab is also visible at the bottom of the developer tools.

**Singapore Payalebar, Singapore ★**

Paya Lebar

Forecast History Calendar Rain / Snow Health

Weather History for WSAP - January, 2017

Change the Weather History Date:

January 1, 2017 View

Sunday, January 1, 2017

« Previous Day Next Day »

Daily Weekly Monthly Custom

	Actual	Average (WSSS)	Record (WSSS)
Temperature			
Mean Temperature	29 °C	-	
Max Temperature	32 °C	29 °C	31 °C (2000)
Min Temperature	26 °C	23 °C	22 °C (2013)
Cooling Degree Days	18		
Growing Degree Days	34 (Base 50)		
Moisture			
Dew Point	24 °C		
Average Humidity	79		

Elements Console Sources Network Performance

h2.history-date

Styles Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

Console What's New

# 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 results
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 results
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](#)