Web Scraping

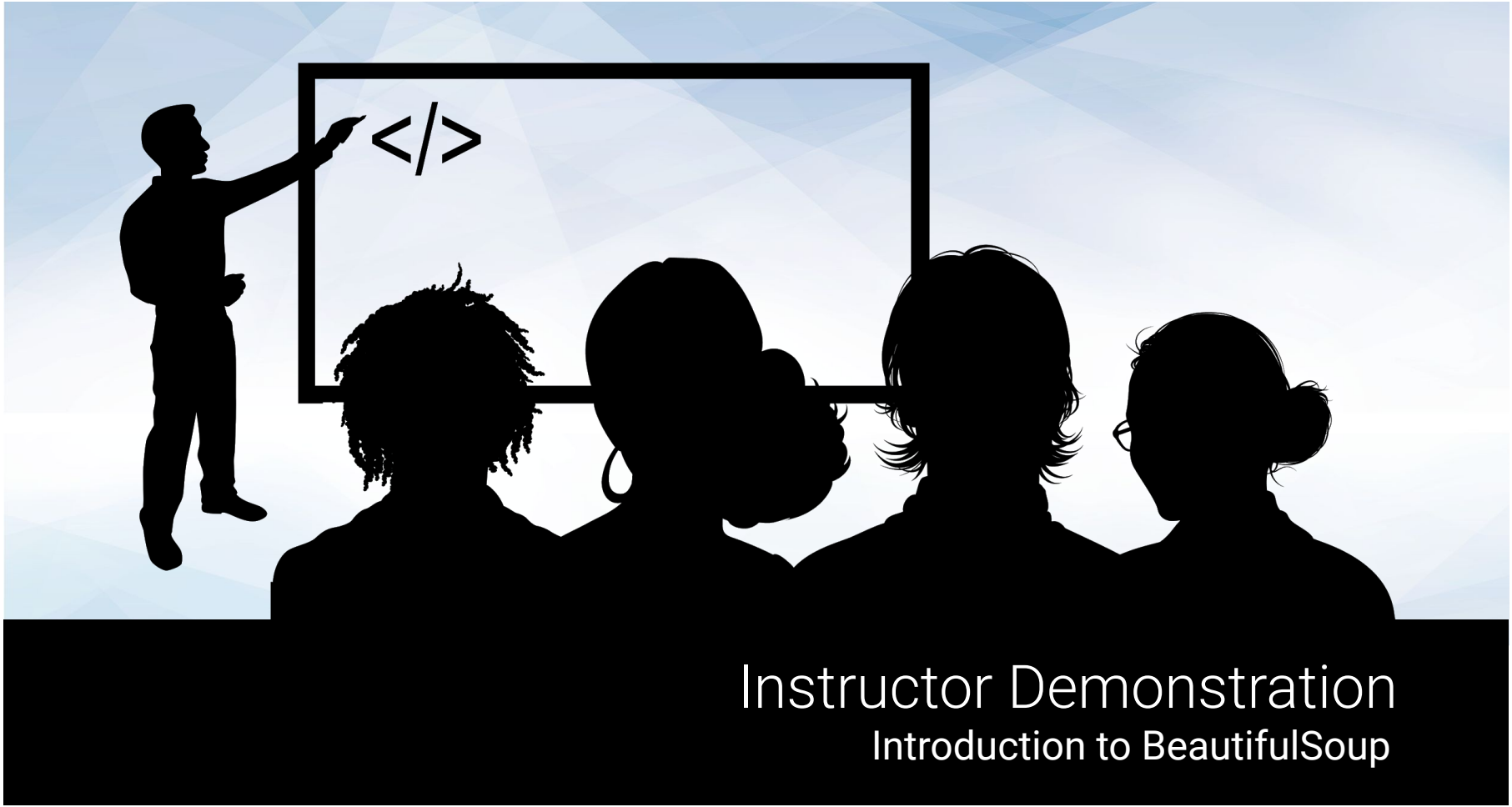
Data Boot Camp

Lesson 12.2

Class Objectives

By the end of today's class you will be able to:

- Use Beautifulsoup to scrape their own data from the web.
- Learn to save the results of web scraping into MongoDB.



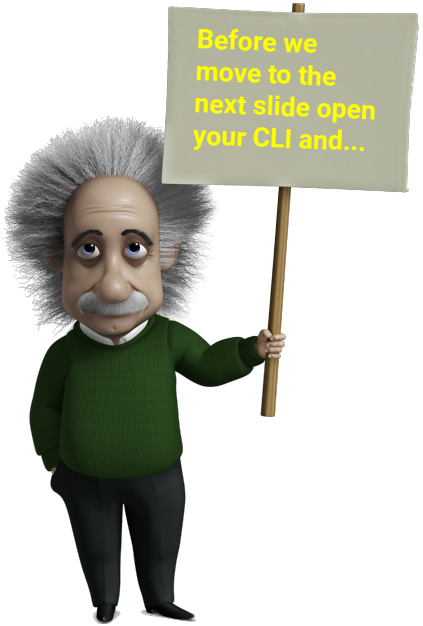
Instructor Demonstration

Introduction to BeautifulSoup

Introduction to BeautifulSoup

What is **BeautifulSoup**?

➔ A Python library created to pull data out from HTML and XML files. It works with your favorite parser resulting in idiomatic ways of navigating, searching, and modifying the parse tree.




```
conda activate PythonData
conda install bs4
conda install lxml
```

Introduction to BeautifulSoup

BeautifulSoup - The Basics

➔ `from bs4 import BeautifulSoup as bs`

```
In [1]: # import dependency
        from bs4 import BeautifulSoup as bs
```

➔ `bs(html_string, 'html.parser')` - This line of code is to create a BeautifulSoup object. The object is assigned to a variable. In this case `soup`.

```
In [2]: html_string = """
        <html>
        <head>
        <title>
        A Simple HTML Document
        </title>
        </head>
        <body>
        <p>This is a very simple HTML document</p>
        <p>It only has two paragraphs</p>
        </body>
        </html>
        """
```

```
In [3]: # create a BS object
        soup = bs(html_string, 'html.parser')
```

Introduction to BeautifulSoup

BeautifulSoup - The Basics

- ➔ `type()` - This method will return the type of the parsed HTML
- ➔ `prettify()` - This method returns a more readable version of the HTML

```
In [4]: type(soup)

Out[4]: bs4.BeautifulSoup

In [5]: print(soup.prettify())

<html>
 <head>
  <title>
   A Simple HTML Document
  </title>
 </head>
 <body>
  <p>
   This is a very simple HTML document
  </p>
  <p>
   It only has two paragraphs
  </p>
 </body>
</html>
```

Introduction to BeautifulSoup

BeautifulSoup - The Basics

- ➔ `soup.title` - You can extract any element from a BS object. This in particular will return the whole 'title' element including the tags.
- ➔ `.text` - Adding this to the end of the call returns the text containing within the title element. Note that will return still surrounded by white space.
- ➔ `.strip()` - Python String method to remove leading and trailing whitespace

```
In [7]: soup.title

Out[7]: <title>
        A Simple HTML Document
        </title>

In [8]: soup.title.text

Out[8]: '\nA Simple HTML Document\n'

In [9]: soup.title.text.strip()

Out[9]: 'A Simple HTML Document'
```

Introduction to BeautifulSoup

BeautifulSoup - The Basics

- ➔ `soup.body` - This will return the entire body of the HTML file and adding the below will return
- ➔ `.text`
- ➔ `.strip()`
- ➔ `.p.text`

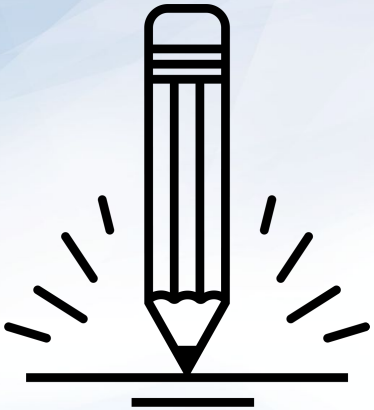
```
In [10]: soup.body

Out[10]: <body>
         <p>This is a very simple HTML document</p>
         <p>It only has two paragraphs</p>
         </body>

In [11]: soup.body.text

Out[11]: '\nThis is a very simple HTML document\nIt only has two paragraphs\n'

In [14]: soup.body.text.strip()

Out[14]: 'This is a very simple HTML document\nIt only has two paragraphs'

In [15]: soup.body.p.text

Out[15]: 'This is a very simple HTML document'
```

Introduction to BeautifulSoup

BeautifulSoup - The Basics

- ➔ `find_all(<element>)` - method returns a list containing all of the HTML elements of a specific type.
- ➔ `('p')` - this element will return all the paragraphs.
- ➔ `[an index number]` - you can also return an specific paragraph using the index number.

```
In [17]: soup.body.find_all('p')

Out[17]: [<p>This is a very simple HTML document</p>, <p>It only has two paragraph
         s</p>]

In [18]: soup.body.find_all('p')[0]

Out[18]: <p>This is a very simple HTML document</p>

In [19]: soup.body.find_all('p')[1]

Out[19]: <p>It only has two paragraphs</p>
```



Activity: CNN Soup

In this activity, you will take your first step in web scraping by taking an external HTML file, parsing it, and then printing out specific elements to the console.

Suggested Time:
15 Minutes

Activity: CNN Soup

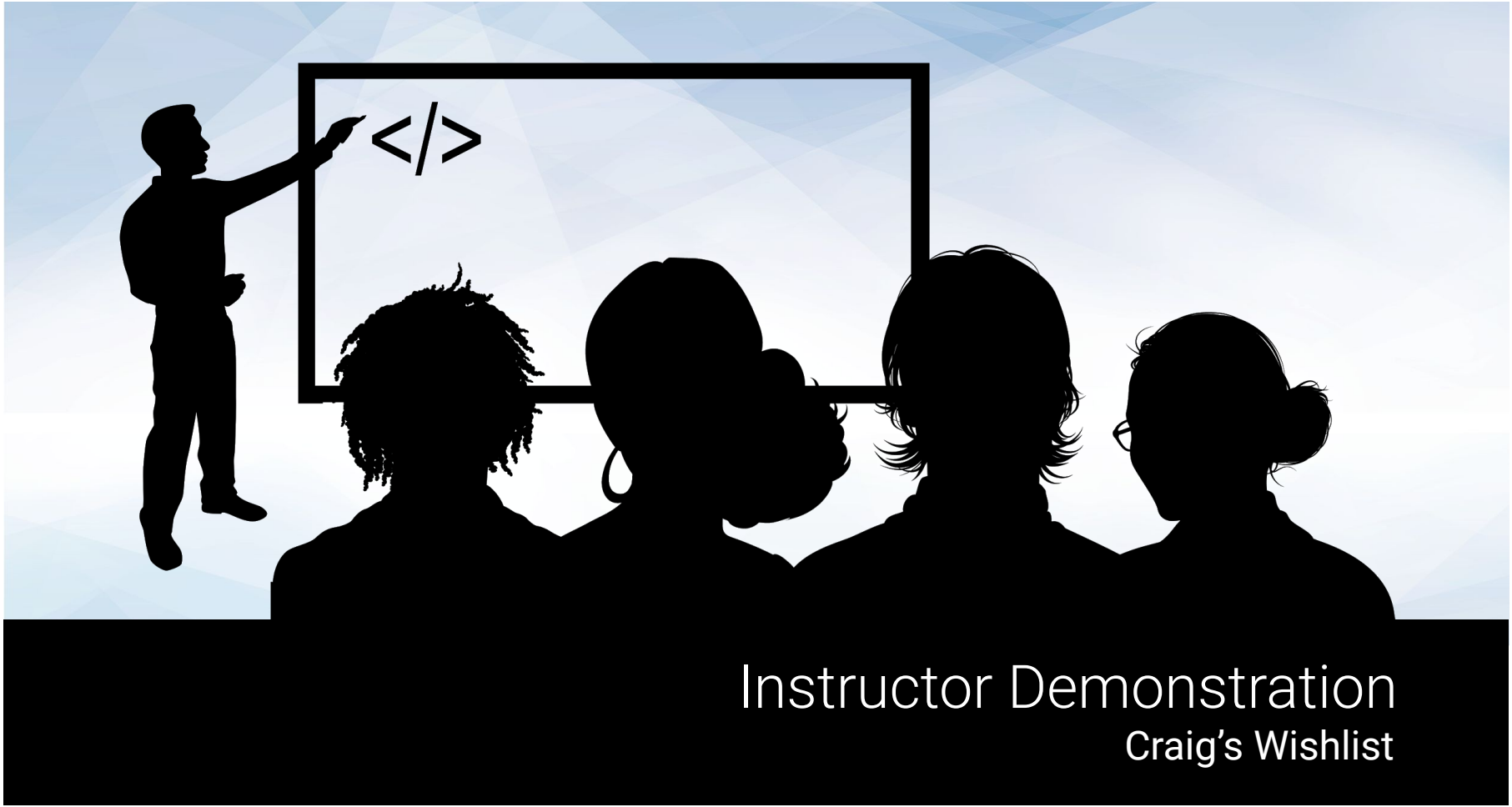
Instructions:

- Believe it or not, CNN's website for 1996: Year in Review is still alive on the web! We have, however, stored the HTML document as a string in your starter file.
- Your task, should you accept it (and you should), is to use BeautifulSoup to scrape and print the following pieces of information:
  - The title of the webpage
  - All paragraph texts on the page
  - The top 10 headlines for the year. This last one is a bit tricky and may not come out perfectly!

- **Hint:**
  - For the third task in this activity you will need a means of filtering the data… Perhaps over multiple iterations… With a loop… HINT HINT!

- **Bonus:**
  - If you finish early, head over to the BeautifulSoup documentation to read up on accessing attributes and navigating the DOM.

Time’s Up! Let’s Review.



Instructor Demonstration

Craig's Wishlist

Craig's Wishlist

BeautifulSoup - Live Website!

- So far you have only parsed HTML strings with BeautifulSoup. Now it is time to scrape a live website!

➔ Look at the code below. Notice anything different from what we have worked on so far today?

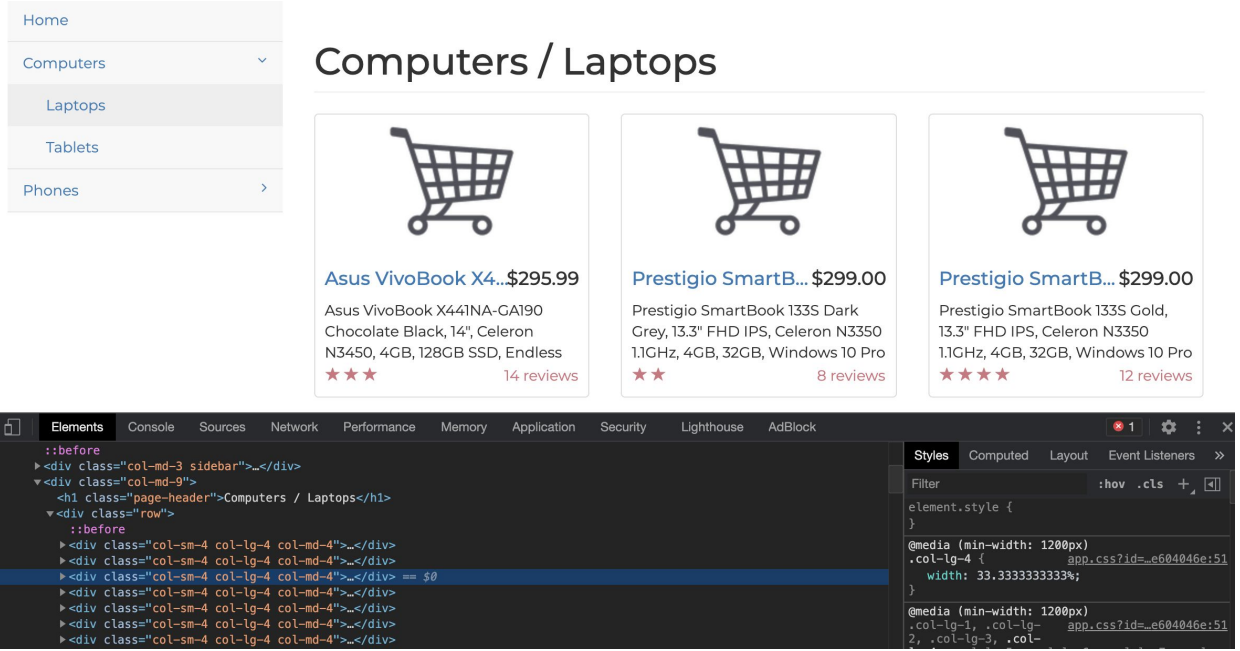
```
[1]: # Dependencies
     from bs4 import BeautifulSoup
     import requests

[2]: # URL of page to be scraped
     url = 'https://webscraper.io/test-sites/e-commerce/allinone/computers/laptops'

[3]: # Retrieve page with the requests module
     response = requests.get(url)

[4]: # Create BeautifulSoup object; parse with 'html.parser'
     soup = BeautifulSoup(response.text, 'html.parser')

[5]: # Examine the results, then determine element that contains sought info
     print(soup.prettify())
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <!-- Anti-flicker snippet (recommended)  -->
  <style>
   .async-hide {
                opacity: 0 !important
        }
  </style>
  <script>
   (function (a, s, y, n, c, h, i, d, e) {
                s.className += ' ' + y;
                h.start = 1 * new Date;
```

Craig's Wishlist

BeautifulSoup - Live Website!

➔ Another way to look into the HTML is to open it in a browser and open up the page's source with the inspector.



Craig's Wishlist

BeautifulSoup - Live Website!

- ➔ The listing price can also be found to exist within an `h4` element whose class is `price`.
- ➔ To retrieve the content of these elements you can call the `find_all('div', class_='caption'` method on the `soup` object.

```
<img class="img-responsive" alt="item" src="/images/test-sites/e-commerce/items/cart2.png">
<div class="caption">
  <h4 class="pull-right price">$295.99</h4>
▼<h4>
   <a href="/test-sites/e-commerce/allinone/product/545" class="title" title="Asus VivoBook X441NA-GA190">Asus
   VivoBook X4...</a>
 </h4>
▼<p class="description">
   "Asus VivoBook X441NA-GA190 Chocolate Black, 14", Celeron N3450, 4GB, 128GB SSD, Endless OS, ENG kbd"
 </p>
```

```
# results are returned as an iterable list
results = soup.find_all('div', class_='caption')
```

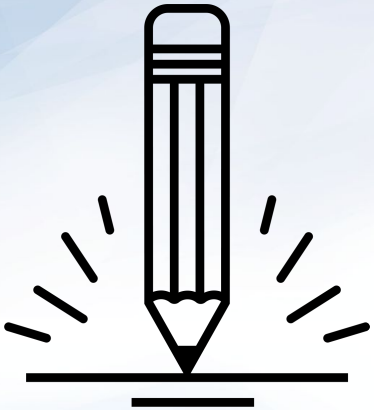
Craig's Wishlist

BeautifulSoup - Live Website!

- ➔ By iterating through the listings, specific information can then be pulled from the BeautifulSoup object.
- ➔ Each listing's title can therefore be gathered using `result.find('a', class_='title').text`, their prices collected with `result.find('h4', class_='price').text`, and their links retrieved by accessing the "href" attribute of each listing using `result.a['href']`.

[7]:
```
# Loop through returned results
for result in results:
    # Error handling
    try:
        # Identify and return title of listing
        title = result.find('a', class_='title').text
        # Identify and return price of listing
        price = result.find('h4', class_='price').text
        # Identify and return link to listing
        link = result.a['href']

        # Print results only if title, price, and link are available
        if (title and price and link):
            print('-------------')
            print(title)
            print(price)
            print(link)
    except AttributeError as e:
        print(e)
```

```
Aspire E1-510
$306.99
/test-sites/e-commerce/allinone/product/517
-------------
Lenovo V110-15IA...
$321.94
/test-sites/e-commerce/allinone/product/548
-------------
Lenovo V110-15IA...
$356.49
/test-sites/e-commerce/allinone/product/549
-------------
```



Activity: Reddit Scraper

In this activity, you will scrape the Python Reddit for potentially interesting content. You will also have to filter for threads with twenty or more comments in them.

Suggested Time:
15 Minutes

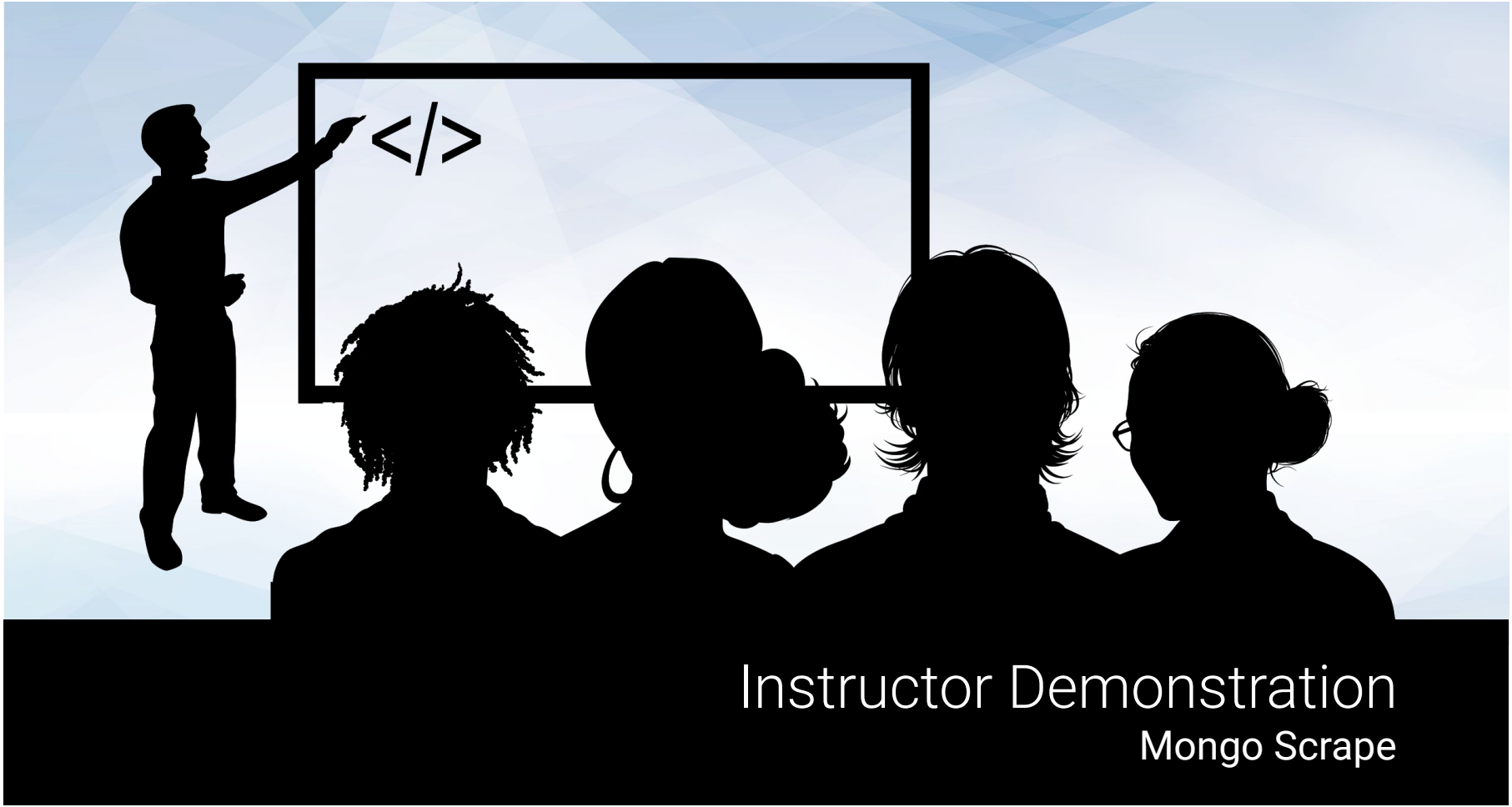
Reddit Scraper

Instructions:

- In this activity, you will scrape a cached version of the Programmer Humor subreddit using BeautifulSoup. Scrape only threads that have twenty or more comments and then print the thread's title, number of comments, as well as the URL for the thread.

```
----------------

Doing conditionals
Comments: 258
https://www.reddit.com/r/ProgrammerHumor/comments/7pw5qk/doing_conditionals/

----------------

Perfect date
Comments: 58
https://www.reddit.com/r/ProgrammerHumor/comments/7pyyl2/perfect_date/

----------------

The truth about java.
Comments: 61
https://www.reddit.com/r/ProgrammerHumor/comments/7pxod4/the_truth_about_java/

----------------
```

Time’s Up! Let’s Review.



Instructor Demonstration

Mongo Scrape

Mongo Scrape

Translate the results of a web scraper to a MongoDB database

➔ For this particular application we are going to use the `lxml` parser instead of `html.parser`. Check out the BeautifulSoup documentation and check an informative table on the various parsers available to BS.

[1]:
```
# Dependencies
from bs4 import BeautifulSoup
import requests
import pymongo
```

[2]:
```
# Initialize PyMongo to work with MongoDBs
conn = 'mongodb://localhost:27017'
client = pymongo.MongoClient(conn)
```

[3]:
```
# Define database and collection
db = client.commerce_db
collection = db.items
```

[4]:
```
# URL of page to be scraped
url = 'https://webscraper.io/test-sites/e-commerce/allinone/computers/laptops'

# Retrieve page with the requests module
response = requests.get(url)
# Create BeautifulSoup object; parse with 'lxml'
soup = BeautifulSoup(response.text, 'lxml')
```

Mongo Scrape

Translate the results of a web scraper to a MongoDB database

➔

[5]:
```
# Examine the results, then determine element that contains sought info
# results are returned as an iterable list
results = soup.find_all('div', class_='caption')

# Loop through returned results
for result in results:
    # Error handling
    try:
        # Identify and return title of listing
        title = result.find('a', class_='title').text
        # Identify and return price of listing
        price = result.find('h4', class_='price').text
        # Identify and return link to listing
        link = result.a['href']

        # Run only if title, price, and link are available
        if (title and price and link):
            # Print results
            print('-------------')
            print(title)
            print(price)
            print(link)

            # Dictionary to be inserted as a MongoDB document
            post = {
                'title': title,
                'price': price,
                'url': link
            }

            collection.insert_one(post)

    except Exception as e:
        print(e)
```

```
-------------
Asus VivoBook X4...
$295.99
/test-sites/e-commerce/allinone/product/545
-------------
Prestigio SmartB...
$299.00
/test-sites/e-commerce/allinone/product/546
-------------
Prestigio SmartB...
$299.00
/test-sites/e-commerce/allinone/product/547
-------------
Aspire E1-510
```

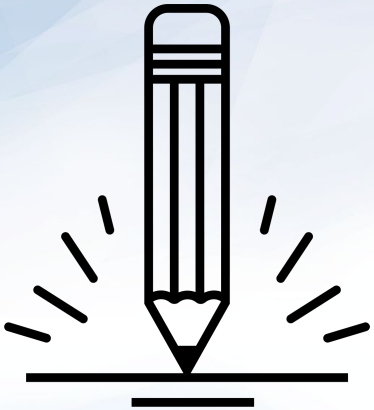
Mongo Scrape

Translate the results of a web scraper to a MongoDB database

➔ After the application has pushed all of the scraped data into the Mongo database, this can be verified by querying the database and printing out all of the results to the console.

[6]:
```
# Display items in MongoDB collection
listings = db.items.find()

for listing in listings:
    print(listing)
```

```
{'_id': ObjectId('60490ef581aa420dd7a2057f'), 'title': 'Asus VivoBook X4...', 'price': '$295.99', 'url': '/test-sites/e-commerce/all
inone/product/545'}
{'_id': ObjectId('60490ef581aa420dd7a20580'), 'title': 'Prestigio SmartB...', 'price': '$299.00', 'url': '/test-sites/e-commerce/all
inone/product/546'}
{'_id': ObjectId('60490ef581aa420dd7a20581'), 'title': 'Prestigio SmartB...', 'price': '$299.00', 'url': '/test-sites/e-commerce/all
inone/product/547'}
{'_id': ObjectId('60490ef581aa420dd7a20582'), 'title': 'Aspire E1-510', 'price': '$306.99', 'url': '/test-sites/e-commerce/allinone/
product/517'}
{'_id': ObjectId('60490ef581aa420dd7a20583'), 'title': 'Lenovo V110-15IA...', 'price': '$321.94', 'url': '/test-sites/e-commerce/all
inone/product/548'}
{'_id': ObjectId('60490ef581aa420dd7a20584'), 'title': 'Lenovo V110-15IA...', 'price': '$356.49', 'url': '/test-sites/e-commerce/all
inone/product/549'}
{'_id': ObjectId('60490ef581aa420dd7a20585'), 'title': 'Hewlett Packard...', 'price': '$364.46', 'url': '/test-sites/e-commerce/alli
none/product/550'}
```



Activity: Hockey Headers

In this activity, you will scrape the news page of the NHL website for the articles and then post the title/header of each code block to the best of your ability.

Suggested Time:
15 Minutes

Hockey Headers

Instructions:

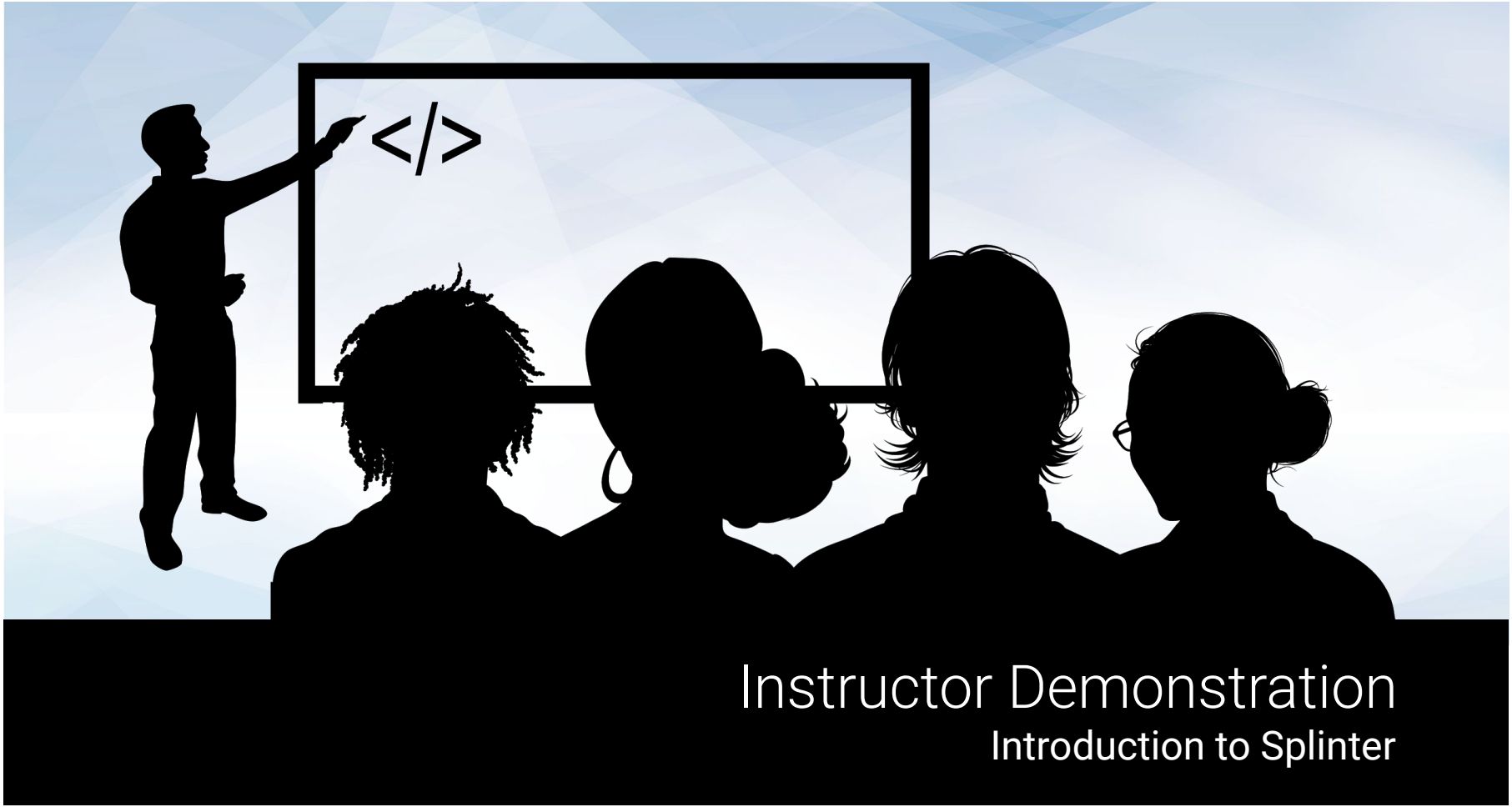
- Teamwork! Speed! Mental and physical toughness! Passion! Excitement! Unpredictable matchups down to the wire! What could be better? While these terms could easily be applied to a data science hackathon, we're talking about the magnificent sport of hockey.
- Your assignment is to scrape the articles on the news page of the NHL website (https://www.nhl.com/news) - which is frequently updated - and then post the results of your scraping to MongoDB.
- Use BeautifulSoup and requests to scrape the header and subheader of each article on the front page.
- Post the above information as a MongoDB document and then print all of the documents on the database to the console.
- In addition to the above, post the date of the article publication as well.

Time’s Up! Let’s Review.

Countdown timer

15:00

(with alarm)



Instructor Demonstration

Introduction to Splinter

Introduction to Splinter

- **Important:** You must have the webdriver-manager installed for this activity.

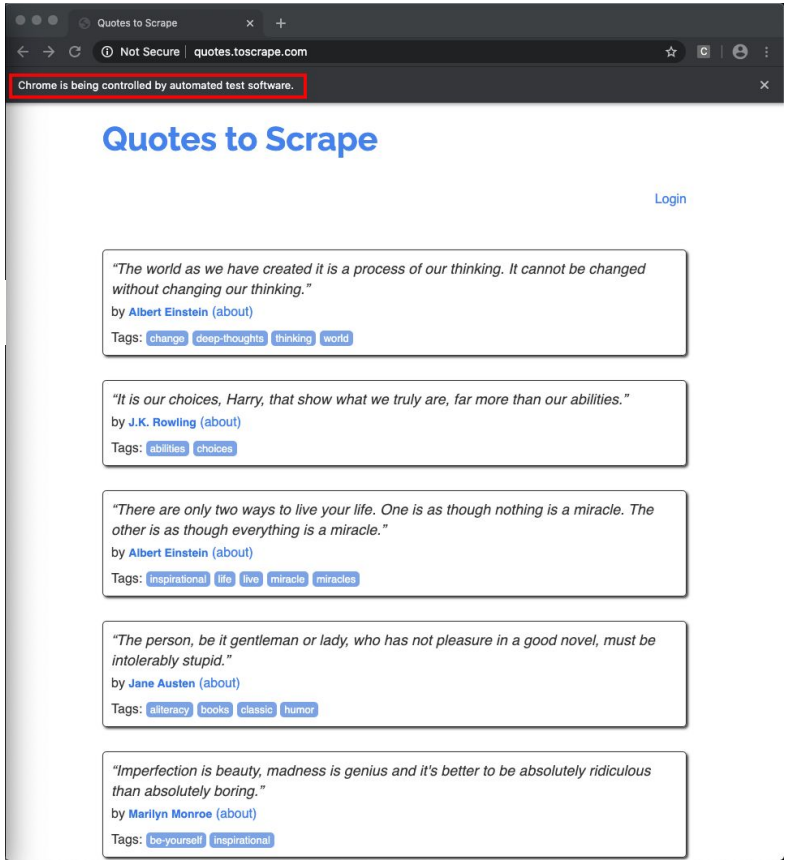
  ➔ **Installation**

```
pip install webdriver_manager
pip install splinter
```

Introduction to Splinter

```
from splinter import Browser
from bs4 import BeautifulSoup
from webdriver_manager.chrome import ChromeDriverManager
```

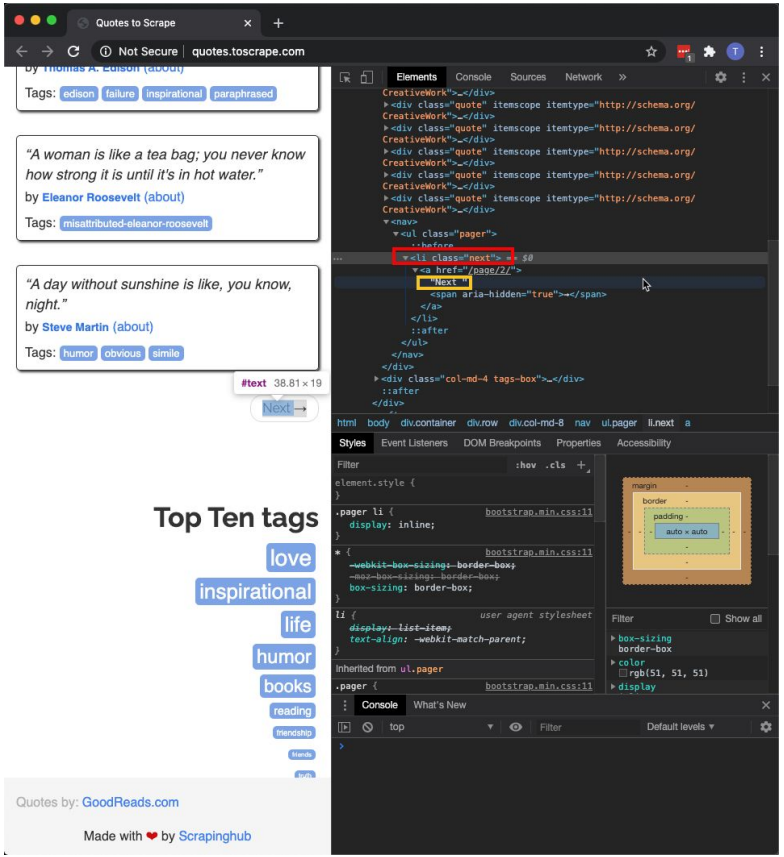
```
# Setup splinter
executable_path = {'executable_path': ChromeDriverManager().install()}
browser = Browser('chrome', **executable_path, headless=False)
```



Note: You might need to install splinter using `pip install splinter`

Introduction to Splinter

- ➔ Open the Chrome inspector to identify the element that the application will need to click.

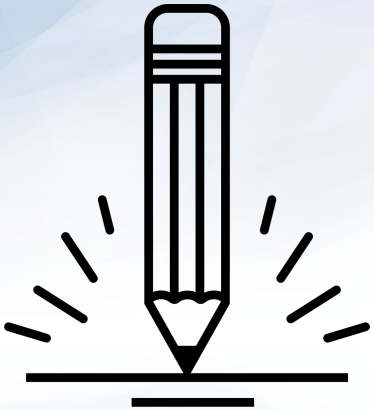


Introduction to Splinter

➔ The `click_link_by_partial_text()` method.

```
In [18]: for x in range(1, 6):

             html = browser.html
             soup = BeautifulSoup(html, 'html.parser')

             quotes = soup.find_all('span', class_='text')

             for quote in quotes:
                 print('page:', x, '-------------')
                 print(quote.text)

             browser.links.find_by_partial_text('Next')
```

```
page: 1 -------------
“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”
page: 1 -------------
“It is our choices, Harry, that show what we truly are, far more than our abilities.”
page: 1 -------------
“There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything
is a miracle.”
page: 1 -------------
“The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid.”
page: 1 -------------
“Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring.”
page: 1 -------------
“Try not to become a man of success. Rather become a man of value.”
page: 1 -------------
“It is better to be hated for what you are than to be loved for what you are not.”
page: 1 -------------
“I have not failed. I've just found 10,000 ways that won't work.”
page: 1 -------------
“A woman is like a tea bag; you never know how strong it is until it's in hot water.”
page: 1 -------------
```



Activity: Bookscraper

In this activity, you will practice your webscraping skills on a site similar to the one shown in the instructor demonstration.

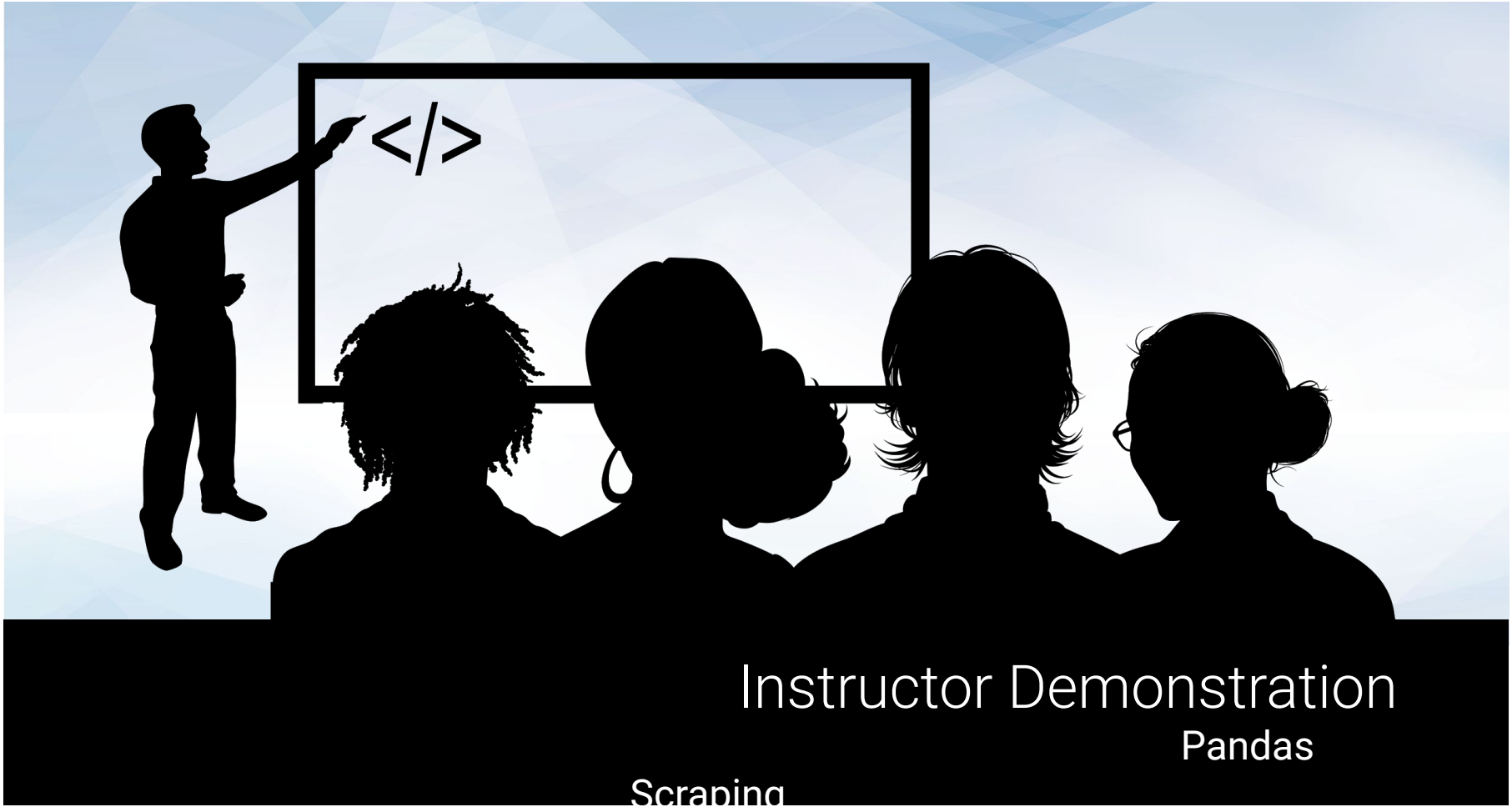
Suggested Time:
15 Minutes

Bookscraper

Instructions:

- Go to http://books.toscrape.com/
- Scrape the titles and the URLs to all books on this fictional online bookstore. Display the results in console.
- That's it!
- If you're craving extra challenge, try scraping all books by category. Good luck!

Time’s Up! Let’s Review.



Instructor Demonstration

Pandas

Scraping

Pandas Scraping

- In Python the Pandas library includes a simple web scraper capability that pulls HTML table data into a dataframe.
- Inserting the URL to the method `read_html()` will return the data in a list format.

```
In [1]: import pandas as pd

In [2]: url = 'https://en.wikipedia.org/wiki/List_of_capitals_in_the_United_States'

In [3]: tables = pd.read_html(url)
        tables
```

	Albany Congress	Albany Congress
0	Albany, New York	Stadt Huys
1	Stamp Act Congress	Stamp Act Congress
2	New York, New York	City Hall
3	First Continental Congress	First Continental Congress
4	Philadelphia, Pennsylvania	Carpenters' Hall
5	Second Continental Congress	Second Continental Congress
6	Philadelphia, Pennsylvania	Independence Hall
7	Baltimore, Maryland	Henry Fite House
8	Philadelphia, Pennsylvania	Independence Hall
9	Lancaster, Pennsylvania	Court House
10	York, Pennsylvania	Court House
11	Philadelphia, Pennsylvania	College Hall
12	Congress of the Confederation	Congress of the Confederation
13	Philadelphia, Pennsylvania	Independence Hall
14	Princeton, New Jersey	Nassau Hall
15	Annapolis, Maryland	Maryland State House
16	Trenton, New Jersey	French Arms Tavern
17	New York, New York	City Hall

```
In [4]: type(tables)

Out[4]: list
```

Pandas Scraping

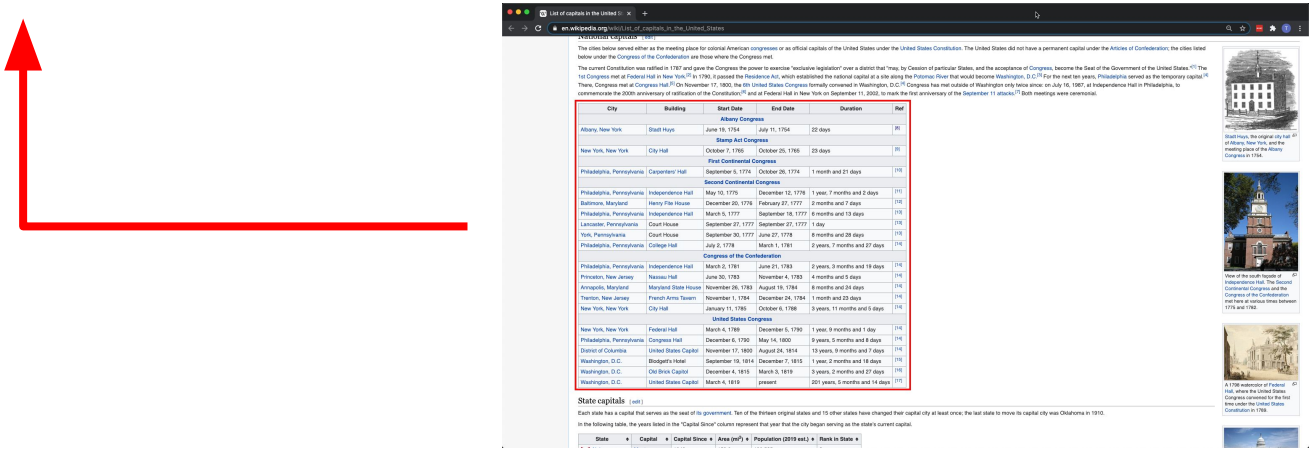
- The wikipedia page scraped contains four HTML table data. Hence, the returned list contains four tables within the list. In order to specify what table we want to grab we can use list indexing.

We can slice off any of those dataframes that we want using normal indexing.

```
In [5]: df = tables[0]
        df.head()
```

Out[5]:

	City	Building	Start Date	End Date	Duration	Ref
	Albany Congress	Albany Congress	Albany Congress	Albany Congress	Albany Congress	Albany Congress
0	Albany, New York	Stadt Huys	June 19, 1754	July 11, 1754	22 days	[8]
1	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress
2	New York, New York	City Hall	October 7, 1765	October 25, 1765	23 days	[9]
3	First Continental Congress	First Continental Congress	First Continental Congress	First Continental Congress	First Continental Congress	First Continental Congress
4	Philadelphia, Pennsylvania	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	[10]



Pandas Scraping

- Often time we need to perform a great deal of data cleaning. Lets go over few instances we might come across while performing data cleaning.

➔ **Dropping entire unnecessary rows**. If we close analyze the table we scraped from wikipedia, you will note there are six header rows that are irrelevant for our use. Let's see how we can get rid of them.

```
df.columns = df.columns.get_level_values(0)
df = df.loc[df.Ref.str.startswith("[")]
```

```
In [7]: df = tables[0]
        df
```

Out[7]:

	City	Building	Start Date	End Date	Duration	Ref
	Albany Congress	Albany Congress	Albany Congress	Albany Congress	Albany Congress	Albany Congress
0	Albany, New York	Stadt Huys	June 19, 1754	July 11, 1754	22 days	[8]
1	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress	Stamp Act Congress
2	New York, New York	City Hall	October 7, 1765	October 25, 1765	23 days	[9]
3	First Continental Congress	First Continental Congress	First Continental Congress	First Continental Congress	First Continental Congress	First Continental Congress
4	Philadelphia, Pennsylvania	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	[10]
5	Second Continental Congress	Second Continental Congress	Second Continental Congress	Second Continental Congress	Second Continental Congress	Second Continental Congress
6	Philadelphia, Pennsylvania	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	[11]
7	Baltimore, Maryland	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	[12]
8	Philadelphia, Pennsylvania	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	[13]
9	Lancaster, Pennsylvania	Court House	September 27, 1777	September 27, 1777	1 day	[13]
10	York, Pennsylvania	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	[13]
11	Philadelphia, Pennsylvania	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	[14]
12	Congress of the Confederation	Congress of the Confederation	Congress of the Confederation	Congress of the Confederation	Congress of the Confederation	Congress of the Confederation
13	Philadelphia, Pennsylvania	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	[14]
14	Princeton, New Jersey	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	[14]
15	Annapolis, Maryland	Maryland State House	November 26, 1783	August 19, 1784	8 months and 24 days	[14]
16	Trenton, New Jersey	French Arms Tavern	November 1, 1784	December 24, 1784	1 month and 23 days	[14]
17	New York, New York	City Hall	January 11, 1785	October 6, 1788	3 years, 11 months and 5 days	[14]
18	United States Congress	United States Congress	United States Congress	United States Congress	United States Congress	United States Congress
19	New York, New York	Federal Hall	March 4, 1789	December 5, 1790	1 year, 9 months and 1 day	[14]
20	Philadelphia, Pennsylvania	Congress Hall	December 6, 1790	May 14, 1800	9 years, 5 months and 8 days	[14]

	City	Building	Start Date	End Date	Duration	Ref
0	Albany, New York	Stadt Huys	June 19, 1754	July 11, 1754	22 days	[8]
2	New York, New York	City Hall	October 7, 1765	October 25, 1765	23 days	[9]
4	Philadelphia, Pennsylvania	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	[10]
6	Philadelphia, Pennsylvania	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	[11]
7	Baltimore, Maryland	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	[12]
8	Philadelphia, Pennsylvania	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	[13]
9	Lancaster, Pennsylvania	Court House	September 27, 1777	September 27, 1777	1 day	[13]
10	York, Pennsylvania	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	[13]
11	Philadelphia, Pennsylvania	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	[14]
13	Philadelphia, Pennsylvania	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	[14]
14	Princeton, New Jersey	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	[14]
15	Annapolis, Maryland	Maryland State House	November 26, 1783	August 19, 1784	8 months and 24 days	[14]
16	Trenton, New Jersey	French Arms Tavern	November 1, 1784	December 24, 1784	1 month and 23 days	[14]
17	New York, New York	City Hall	January 11, 1785	October 6, 1788	3 years, 11 months and 5 days	[14]
19	New York, New York	Federal Hall	March 4, 1789	December 5, 1790	1 year, 9 months and 1 day	[14]
20	Philadelphia, Pennsylvania	Congress Hall	December 6, 1790	May 14, 1800	9 years, 5 months and 8 days	[14]
21	District of Columbia	United States Capitol	November 17, 1800	August 24, 1814	13 years, 9 months and 7 days	[14]
22	Washington, D.C.	Blodgett's Hotel	September 19, 1814	December 7, 1815	1 year, 2 months and 18 days	[15]
23	Washington, D.C.	Old Brick Capitol	December 4, 1815	March 3, 1819	3 years, 2 months and 27 days	[16]
24	Washington, D.C.	United States Capitol	March 4, 1819	present	201 years, 5 months and 14 days	[17]

Pandas Scraping

➔ **Splitting values** from one column to two separate columns.

```
columnsplit = df['City'].str.split(", ", expand=True)
df = df.assign(City=columnsplit[0], State=columnsplit[1])
df
```

	City	Building	Start Date	End Date	Duration	Ref
0	Albany, New York	Stadt Huys	June 19, 1754	July 11, 1754	22 days	[8]
2	New York, New York	City Hall	October 7, 1765	October 25, 1765	23 days	[9]
4	Philadelphia, Pennsylvania	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	[10]
6	Philadelphia, Pennsylvania	ndependence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	[11]
7	Baltimore, Maryland	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	[12]
8	Philadelphia, Pennsylvania	ndependence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	[13]
9	Lancaster, Pennsylvania	Court House	September 27, 1777	September 27, 1777	1 day	[13]

Out[25]:

	City	Building	Start Date	End Date	Duration	Ref	State
0	Albany	Stadt Huys	June 19, 1754	July 11, 1754	22 days	[8]	New York
2	New York	City Hall	October 7, 1765	October 25, 1765	23 days	[9]	New York
4	Philadelphia	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	[10]	Pennsylvania
6	Philadelphia	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	[11]	Pennsylvania
7	Baltimore	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	[12]	Maryland
8	Philadelphia	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	[13]	Pennsylvania
9	Lancaster	Court House	September 27, 1777	September 27, 1777	1 day	[13]	Pennsylvania
10	York	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	[13]	Pennsylvania
11	Philadelphia	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	[14]	Pennsylvania
13	Philadelphia	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	[14]	Pennsylvania
14	Princeton	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	[14]	New Jersey
15	Annapolis	Maryland State House	November 26, 1783	August 19, 1784	8 months and 24 days	[14]	Maryland
16	Trenton	French Arms Tavern	November 1, 1784	December 24, 1784	1 month and 23 days	[14]	New Jersey
17	New York	City Hall	January 11, 1785	October 6, 1788	3 years, 11 months and 5 days	[14]	New York
19	New York	Federal Hall	March 4, 1789	December 5, 1790	1 year, 9 months and 1 day	[14]	New York
20	Philadelphia	Congress Hall	December 6, 1790	May 14, 1800	9 years, 5 months and 8 days	[14]	Pennsylvania
21	District of Columbia	United States Capitol	November 17, 1800	August 24, 1814	13 years, 9 months and 7 days	[14]	None
22	Washington	Blodgett's Hotel	September 19, 1814	December 7, 1815	1 year, 2 months and 18 days	[15]	D.C.
23	Washington	Old Brick Capitol	December 4, 1815	March 3, 1819	3 years, 2 months and 27 days	[16]	D.C.
24	Washington	United States Capitol	March 4, 1819	present	201 years, 5 months and 14 days	[17]	D.C.

Pandas Scraping

➔ **Dropping unnecessary column(s).**

```
df = df.drop(['Ref'], axis=1)
```

Out[25]:

	City	Building	Start Date	End Date	Duration	Ref	State
0	Albany	Stadt Huys	June 19, 1754	July 11, 1754	22 days	[8]	New York
2	New York	City Hall	October 7, 1765	October 25, 1765	23 days	[9]	New York
4	Philadelphia	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	[10]	Pennsylvania
6	Philadelphia	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	[11]	Pennsylvania
7	Baltimore	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	[12]	Maryland
8	Philadelphia	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	[13]	Pennsylvania
9	Lancaster	Court House	September 27, 1777	September 27, 1777	1 day	[13]	Pennsylvania
10	York	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	[13]	Pennsylvania
11	Philadelphia	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	[14]	Pennsylvania
13	Philadelphia	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	[14]	Pennsylvania
14	Princeton	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	[14]	New Jersey
15	Annapolis	Maryland State House	November 26, 1783	August 19, 1784	8 months and 24 days	[14]	Maryland
16	Trenton	French Arms Tavern	November 1, 1784	December 24, 1784	1 month and 23 days	[14]	New Jersey
17	New York	City Hall	January 11, 1785	October 6, 1788	3 years, 11 months and 5 days	[14]	New York
19	New York	Federal Hall	March 4, 1789	December 5, 1790	1 year, 9 months and 1 day	[14]	New York
20	Philadelphia	Congress Hall	December 6, 1790	May 14, 1800	9 years, 5 months and 8 days	[14]	Pennsylvania
21	District of Columbia	United States Capitol	November 17, 1800	August 24, 1814	13 years, 9 months and 7 days	[14]	None
22	Washington	Blodgett's Hotel	September 19, 1814	December 7, 1815	1 year, 2 months and 18 days	[15]	D.C.
23	Washington	Old Brick Capitol	December 4, 1815	March 3, 1819	3 years, 2 months and 27 days	[16]	D.C.
24	Washington	United States Capitol	March 4, 1819	present	201 years, 5 months and 14 days	[17]	D.C.

Out[44]:

	City	Building	Start Date	End Date	Duration	State
0	Albany	Stadt Huys	June 19, 1754	July 11, 1754	22 days	New York
2	New York	City Hall	October 7, 1765	October 25, 1765	23 days	New York
4	Philadelphia	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	Pennsylvania
6	Philadelphia	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	Pennsylvania
7	Baltimore	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	Maryland
8	Philadelphia	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	Pennsylvania
9	Lancaster	Court House	September 27, 1777	September 27, 1777	1 day	Pennsylvania
10	York	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	Pennsylvania
11	Philadelphia	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	Pennsylvania
13	Philadelphia	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	Pennsylvania
14	Princeton	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	New Jersey
15	Annapolis	Maryland State House	November 26, 1783	August 19, 1784	8 months and 24 days	Maryland
16	Trenton	French Arms Tavern	November 1, 1784	December 24, 1784	1 month and 23 days	New Jersey
17	New York	City Hall	January 11, 1785	October 6, 1788	3 years, 11 months and 5 days	New York
19	New York	Federal Hall	March 4, 1789	December 5, 1790	1 year, 9 months and 1 day	New York
20	Philadelphia	Congress Hall	December 6, 1790	May 14, 1800	9 years, 5 months and 8 days	Pennsylvania
21	District of Columbia	United States Capitol	November 17, 1800	August 24, 1814	13 years, 9 months and 7 days	None
22	Washington	Blodgett's Hotel	September 19, 1814	December 7, 1815	1 year, 2 months and 18 days	D.C.
23	Washington	Old Brick Capitol	December 4, 1815	March 3, 1819	3 years, 2 months and 27 days	D.C.
24	Washington	United States Capitol	March 4, 1819	present	201 years, 5 months and 14 days	D.C.

Pandas Scraping

➔ **Reset index.**

```
df = df.reset_index(drop=True)
```

	City	Building	Start Date	End Date	Duration	State
0	Albany	Stadt Huys	June 19, 1754	July 11, 1754	22 days	New York
2	New York	City Hall	October 7, 1765	October 25, 1765	23 days	New York
4	Philadelphia	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	Pennsylvania
6	Philadelphia	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	Pennsylvania
7	Baltimore	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	Maryland
8	Philadelphia	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	Pennsylvania
9	Lancaster	Court House	September 27, 1777	September 27, 1777	1 day	Pennsylvania
10	York	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	Pennsylvania
11	Philadelphia	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	Pennsylvania
13	Philadelphia	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	Pennsylvania
14	Princeton	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	New Jersey

Out[9]:

	City	Building	Start Date	End Date	Duration	State
0	Albany	Stadt Huys	June 19, 1754	July 11, 1754	22 days	New York
1	New York	City Hall	October 7, 1765	October 25, 1765	23 days	New York
2	Philadelphia	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	Pennsylvania
3	Philadelphia	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	Pennsylvania
4	Baltimore	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	Maryland

Pandas Scraping

- Pandas also provides a function called `to_html` function to revert DataFrames back to HTML.

Pandas also had a `to_html` method that we can use to generate HTML tables from DataFrames.

```
In [9]: html_table = df.to_html()
        html_table
```

```
Out[9]: '<table border="1" class="dataframe">\n  <thead>\n    <tr style="text-align: right;">\n      <th></th>\n      <th>Capital.</th>\n      <th>Capital Since</th>\n      <th>Area (mi2)</th>\n      <th>Population (2019 est.)</th>\n      <th>Rank in State</th>\n    </tr>\n    <tr>\n      <th>State</th>\n      <th></th>\n      <th></th>\n      <th></th>\n      <th></th>\n    </tr>\n  </thead>\n  <tbody>\n    <tr>\n      <th>California</th>\n      <td>Sacramento</td>\n      <td>1854</td>\n      <td>97.9</td>\n      <td>513624</td>\n      <td>6</td>\n    </tr>\n    <tr>\n      <th>Colorado</th>\n      <td>Denver</td>\n      <td>1867</td>\n      <td>153.3</td>\n      <td>727211</td>\n      <td>1</td>\n    </tr>\n    <tr>\n      <th>Connecticut</th>\n      <td>Hartford</td>\n      <td>1875</td>\n      <td>17.3</td>\n      <td>122105</td>\n      <td>3</td>\n    </tr>\n    <tr>\n      <th>Delaware</th>\n      <td>Dover</td>\n      <td>1777</td>\n      <td>22.4</td>\n      <td>38079</td>\n      <td>2</td>\n    </tr>\n    <tr>\n      <th>Florida</th>\n      <td>Tallahassee</td>\n      <td>1824</td>\n      <td>95.7</td>\n      <td>194500</td>\n      <td>7</td>\n    </tr>\n    <tr>\n      <th>Georgia</th>\n      <td>Atlanta</td>\n      <td>1868</td>\n      <td>133.5</td>\n      <td>506811</td>\n      <td>1</td>\n    </tr>\n    <tr>\n      <th>Hawaii</th>\n      <td>Honolulu</td>\n      <td>1845</td>\n      <td>68.4</td>\n      <td>345064</td>\n      <td>1</td>\n    </tr>\n    <tr>\n      <th>Idaho</th>\n      <td>Boise</td>\n      <td>1865</td>\n      <td>63.8</td>\n      <td>228959</td>\n      <td>1</td>\n    </tr>\n    <tr>\n      <th>Illinois</th>\n      <td>Springfield</td>\n      <td>1837</td>\n      <td>54.0</td>\n      <td>114230</td>\n      <td>6</td>\n    </tr>\n    <tr>\n      <th>Indiana</th>\n      <td>Indianapolis</td>\n      <td>1825</td>\n      <td>361.5</td>\n      <td>876384</td>\n      <td>1</td>\n    </tr>\n    <tr>\n      <th>Iowa</th>\n      <td>Des Moines</td>\n      <td>1857</td>\n      <td>75.8</td>\n      <td>214237</td>\n      <td>1</td>\n    </tr>\n    <tr>\n      <th>Kansas</th>\n      <td>Topeka</td>\n      <td>1856</td>\n      <td>56.0</td>\n      <td>125310</td>\n      <td>4</td>\n    </tr>\n    <tr>\n      <th>Kentucky</th>\n      <td>Frankfort</td>\n      <td>1792</td>\n      <td>14.7</t
```

- Following we can use `replace` in order to clean up the table.

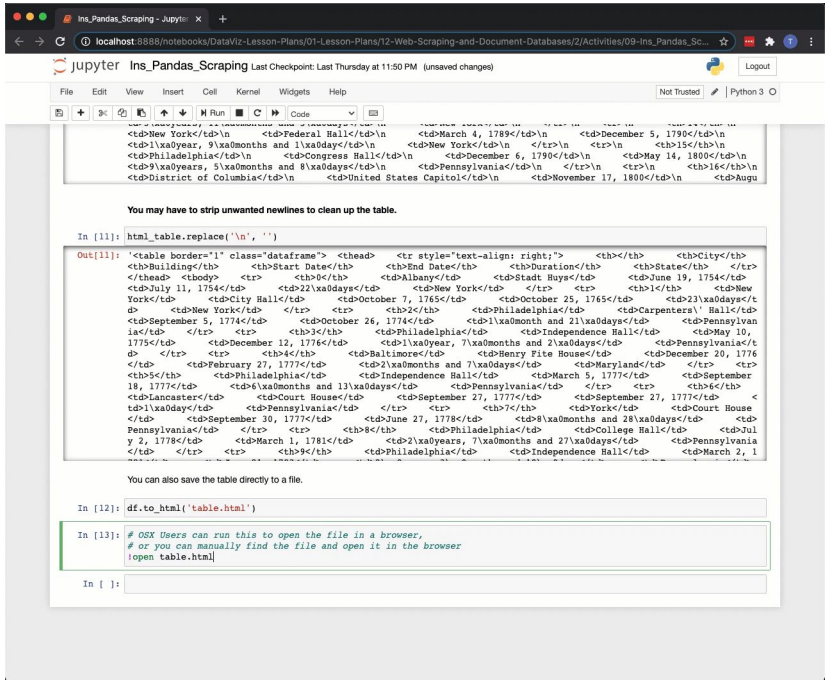
You may have to strip unwanted newlines to clean up the table.

```
In [10]: html_table.replace('\n', '')
```

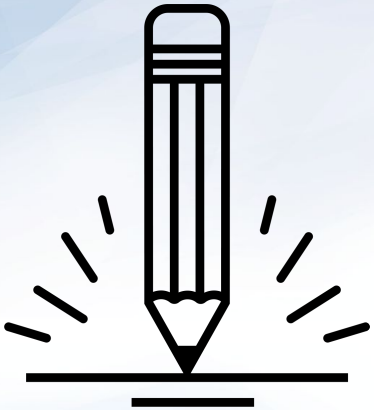
```
Out[10]: '<table border="1" class="dataframe">  <thead>    <tr style="text-align: right;">      <th></th>      <th>Capital.</th>      <th>Capital Since</th>      <th>Area (mi2)</th>      <th>Population (2019 est.)</th>      <th>Rank in State</th>    </tr>    <tr>      <th>State</th>      <th></th>      <th></th>      <th></th>      <th></th>    </tr>  </thead>  <tbody>    <tr>      <th>California</th>      <td>Sacramento</td>      <td>1854</td>      <td>97.9</td>      <td>513624</td>      <td>6</td>    </tr>    <tr>      <th>Colorado</th>      <td>Denver</td>      <td>1867</td>      <td>153.3</td>      <td>727211</td>      <td>1</td>    </tr>    <tr>      <th>Connecticut</th>      <td>Hartford</td>      <td>1875</td>      <td>17.3</td>      <td>122105</td>      <td>3</td>    </tr>    <tr>      <th>Delaware</th>      <td>Dover</td>      <td>1777</td>      <td>22.4</td>      <td>38079</td>      <td>2</td>    </tr>    <tr>      <th>Florida</th>      <td>Tallahassee</td>      <td>1824</td>      <td>95.7</td>      <td>194500</td>      <td>7</td>    </tr>    <tr>      <th>Georgia</th>      <td>Atlanta</td>      <td>1868</td>      <td>133.5</td>      <td>506811</td>      <td>1</td>    </tr>    <tr>      <th>Hawaii</th>      <td>Honolulu</td>      <td>1845</td>      <td>68.4</td>      <td>345064</td>      <td>1</td>    </tr>    <tr>      <th>Idaho</th>      <td>Boise</td>      <td>1865</td>      <td>63.8</td>      <td>228959</td>      <td>1</td>    </tr>    <tr>      <th>Illinois</th>      <td>Springfield</td>      <td>1837</td>      <td>54.0</td>      <td>114230</td>      <td>6</td>    </tr>    <tr>      <th>Indiana</th>      <td>Indianapolis</td>      <td>1825</td>      <td>361.5</td>      <td>876384</td>      <td>1</td>    </tr>    <tr>      <th>Iowa</th>      <td>Des Moines</td>      <td>1857</td>      <td>75.8</td>      <td>214237</td>      <td>1</td>    </tr>    <tr>      <th>Kansas</th>      <td>Topeka</td>      <td>1856</td>      <td>56.0</td>      <td>125310</td>      <td>4</td>    </tr>    <tr>      <th>Kentucky</th>      <td>Frankfort</td>      <td>1792</td>      <
td>14.7</td>      <td>27679</td>      <td>14</td>    </tr>    <tr>      <th>Louisiana</th>      <td>Baton Rouge</td>      <td>1880</td>      <td>76.8</td>      <td>220236</td>      <td>2</td>    </tr>    <tr>      <th>Maine</th>      <td>A
```

Pandas Scraping

- Last but not least, we are able to save the HTML table utilizing `df.to_html('table.html')`. Note that Mac OS X users are able to run `!open table.html` to open the file directly to a browser.



	City	Building	Start Date	End Date	Duration	State
0	Albany	Stadt Huys	June 19, 1754	July 11, 1754	22 days	New York
1	New York	City Hall	October 7, 1765	October 25, 1765	23 days	New York
2	Philadelphia	Carpenters' Hall	September 5, 1774	October 26, 1774	1 month and 21 days	Pennsylvania
3	Philadelphia	Independence Hall	May 10, 1775	December 12, 1776	1 year, 7 months and 2 days	Pennsylvania
4	Baltimore	Henry Fite House	December 20, 1776	February 27, 1777	2 months and 7 days	Maryland
5	Philadelphia	Independence Hall	March 5, 1777	September 18, 1777	6 months and 13 days	Pennsylvania
6	Lancaster	Court House	September 27, 1777	September 27, 1777	1 day	Pennsylvania
7	York	Court House	September 30, 1777	June 27, 1778	8 months and 28 days	Pennsylvania
8	Philadelphia	College Hall	July 2, 1778	March 1, 1781	2 years, 7 months and 27 days	Pennsylvania
9	Philadelphia	Independence Hall	March 2, 1781	June 21, 1783	2 years, 3 months and 19 days	Pennsylvania
10	Princeton	Nassau Hall	June 30, 1783	November 4, 1783	4 months and 5 days	New Jersey
11	Annapolis	Maryland State House	November 26, 1783	August 19, 1784	8 months and 24 days	Maryland
12	Trenton	French Arms Tavern	November 1, 1784	December 24, 1784	1 month and 23 days	New Jersey
13	New York	City Hall	January 11, 1785	October 6, 1788	3 years, 11 months and 5 days	New York
14	New York	Federal Hall	March 4, 1789	December 5, 1790	1 year, 9 months and 1 day	New York
15	Philadelphia	Congress Hall	December 6, 1790	May 14, 1800	9 years, 5 months and 8 days	Pennsylvania
16	District of Columbia	United States Capitol	November 17, 1800	August 24, 1814	13 years, 9 months and 7 days	None
17	Washington	Blodgett's Hotel	September 19, 1814	December 7, 1815	1 year, 2 months and 18 days	D.C.
18	Washington	Old Brick Capitol	December 4, 1815	March 3, 1819	3 years, 2 months and 27 days	D.C.
19	Washington	United States Capitol	March 4, 1819	present	201 years, 5 months and 14 days	D.C.



Activity: Doctor Decoder

In this activity, you will use `read_html` from Pandas to scrape a Wikipedia article. You will then use the resulting DataFrame to convert a list of medical abbreviations to their full description.

Suggested Time:
15 Minutes

Hockey Headers

Instructions:

- Use Panda's `read_html` to parse the URL.
- Find the medical abbreviations DataFrame in the list of DataFrames and assign it to `df`.
  - Keep the columns `['abb', 'full_name', 'other']`
- Drop the `other` column from the DataFrame.
- Drop the header row (the first row) and set the index to the `abb` column.
- Loop through the list of medical abbreviations and print the abbreviation along with the full description.
  - Use the DataFrame to perform the lookup.

Time’s Up! Let’s Review.


The End