DATA SCIENCE 102:
HTML Web Scraping

AGENDA

- What is HTML?
- HTML Elements
- Tables
- Web Scraping
    - Requests
    - BeautifulSoup

WHAT IS HTML

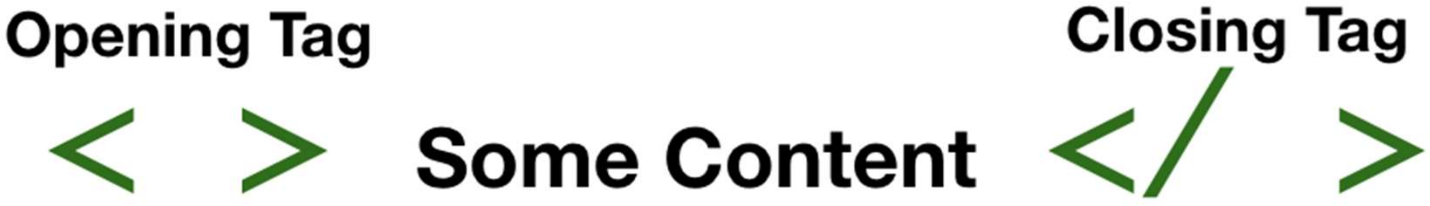
- HTML stands for **Hyper Text Markup Language**.

- It is the standard language which your browser (Google Chrome, Safari, Microsoft Edge) interprets from websites in order to display nice webpages for you.

- It consists of different elements that are used for different purposes
.
- These elements are represented as tags, used to tell browser how to display the content

HTML TAG

- In HTML, codes are enclosed in tags like these.

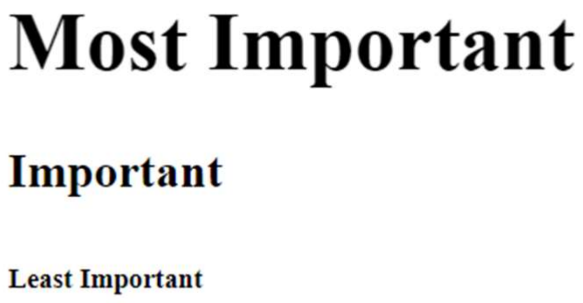


HTML TAG - HEADERS

- Headers are defined using <h1> to <h6>.

- The smaller the number, the bigger the header. When writing html, we use <h1> for most important header and <h6> for least important header.

```html
<h1>Most Important</h1>
<h3>Important</h3>
<h6>Least Important</h6>
```



Most Important

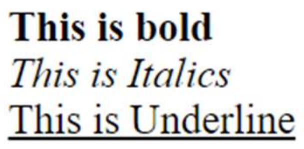
Important

Least Important

HTML TAG - TEXT FORMATTING

- We can edit texts with some simple formatting like.
  - Bold
  - Italics
  - Underline

```
<b>This is Bold</b>
<i>This is Italics</i>
<u>This is underline</u>
```


This is bold
This is Italics
<u>This is Underline</u>

HTML TAG - PARAGRAPH

- For long texts or paragraph, we will put into <p> tag.

```
<p>This is a long paragraph.
I need to study hard to
score well.</p>
```

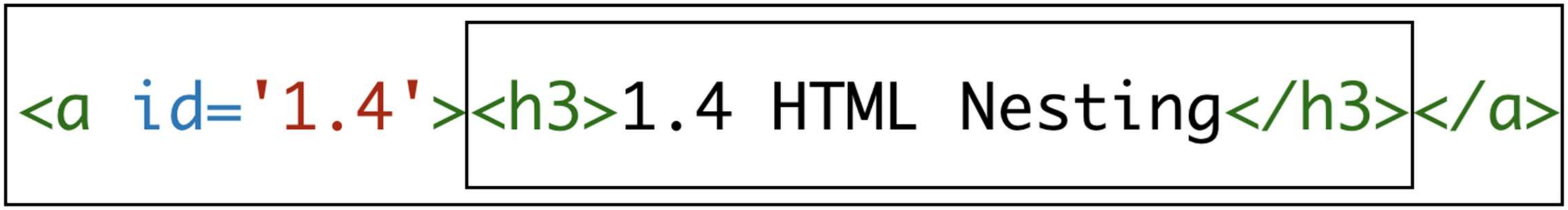
```
This is a long paragraph. I need to study hard to score well.
```

HTML TAG - OTHERS

- There are also other tags available in HTML, namely <div>, <a>, <img> etc for different purposes.
- Some tags such as <br> or <hr> do not need closing tag.
- You can refer to here for more information.

HTML TAG - NESTING

- HTML Tags can be nested within each other, and they must be closed appropriately.



```
<a id='1.4'><h3>1.4 HTML Nesting</h3></a>
```

ATTRIBUTES

- Attributes are settings that changes the styling, dimension, identifier or action of the tag

```
< attribute_name= >
```

COMMON ATTRIBUTES

Attribute	Description
alt	Specifies an alternative text for an image, when the image cannot be displayed
class	Point to a class in a style sheet. Other languages can make changes to HTML element with a specified class
href	Specifies the URL (web address) for a link
id	Specifies a unique id for an element
src	Specifies the URL (web address) for an image
title	Specifies extra information about an element (displayed as a tool tip)

HTML TAG - <a>

- Attributes are most commonly seen in <div> and <a> tag.
- In <a> tag, the attribute **href** will be used to specify the hyperlink of the content.

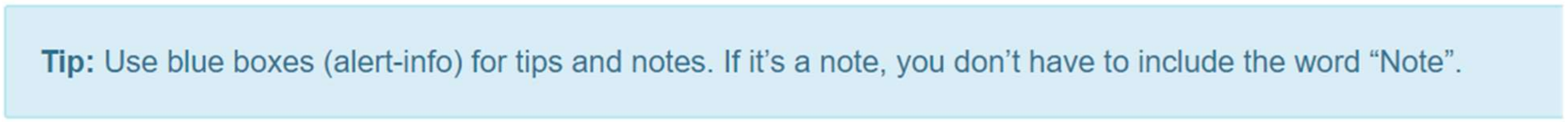
```
<a href="www.google.com">Click here</a>
```

Click here

HTML TAG - <div>

- In <div> tag, the attribute **class** will be used so that all elements within it can be changed by other languages (Javascript, CSS etc).

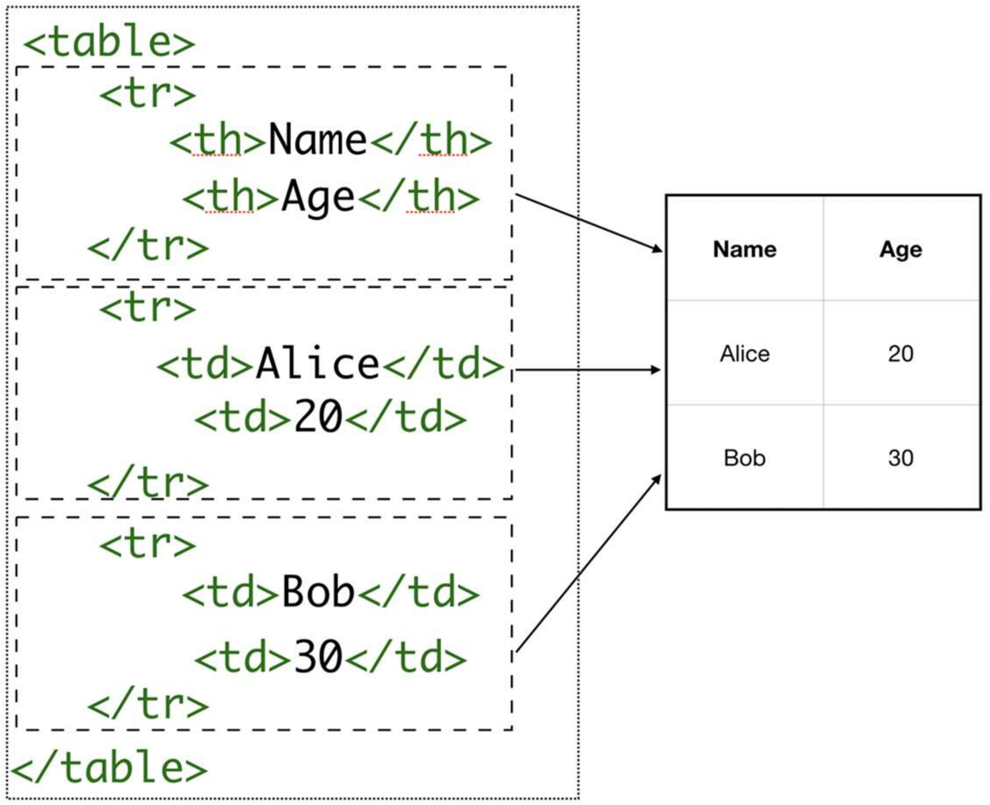
```html
<div class="alert alert-block alert-info">
<b>Tip:</b> Use blue boxes (alert-info) for tips
and notes.
If it’s a note, you don’t have to include the word
"Note".
</div>
```



Tip: Use blue boxes (alert-info) for tips and notes. If it’s a note, you don’t have to include the word “Note”.

TABLES

- In most of the websites, many useful datasets are stored within tables
- We use the following tags to determine a table :
  - <table>
  - <tr>
  - <th>
  - <td>



```
<table>
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <td>Alice</td>
        <td>20</td>
    </tr>
    <tr>
        <td>Bob</td>
        <td>30</td>
    </tr>
</table>
```

Name	Age
Alice	20
Bob	30

TABLES

- Few things to note when writing a table with HTML :
  - Opening and closing <table> tags
  - Each row is represented by <tr> tags
  - Header cells are represented by <th> tags
  - Data cells are represented by <td> tags
- We can also add attribute like *id* or *class* for table tags

WEB SCRAPING

- Data Science Project Stages
- What is Web Scraping?
- Web Scraping Mindset
- Robots.txt

DATA SCIENCE PROJECT STAGES

Problem Specification

- Understand business scenario
- Define the project problem and scope
- Define limitations of the project

Data Gathering & Preprocessing

- **Develop a system to gather data**
- **Clean and prepare raw data for processing**
- **Usually the most time consuming stage in a data science project**

Descriptive Analytics

- Exploratory Data Analysis
- Basic understanding of the dataset
- Answer the initial assumptions you may have of the data

Machine Learning

- Depending on the scope/nature of your project, apply necessary machine learning models to tackle the problem
- Train the machine learning model and assess its' performance

Deployment

- Consult with project stakeholders on suitability of model
- Deploy model for live usage

WEB SCRAPING

- Data do not usually come in a usable format. There are situation where people are hired just to copy and paste manually over hundred of webpages.
- Web scraping is a process of extracting data from web automatically, thus increased the efficiency of data analysis.
- For companies such as Facebook / Instagram, data is part of their assets. Please make sure you scrape and use it responsibly.

WEB SCRAPING MINDSET

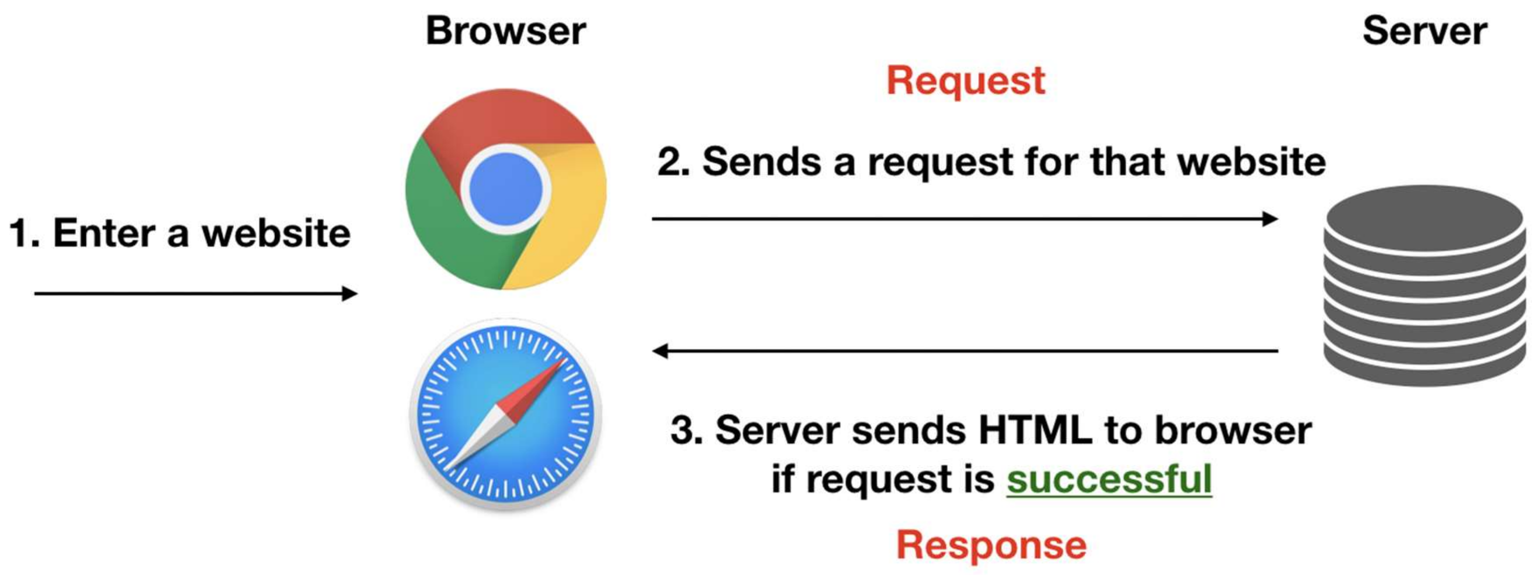
- Life is tough. You usually won't get 100% of the data you want, only getting about 80% of the data.
- Before scraping, look at the website and the content you want before you start, otherwise it will be a hassle to keep staring at the code.
- Sometimes, it could be illegal to scrape certain parts of the website, so do look out for the robots.txt hyperlink to know what you can or cannot scrape.
- Don't ever use your home wifi to scrape unless you want to risk getting it banned. Try public wifi or VPNs.

ROBOTS.TXT

- Web site owners use the /robots.txt file to give instructions about their site to web robots; this is called The Robots Exclusion Protocol.
- Before these robots can crawl their webpage they will visit /robots.txt to see what is allowed or disallowed.
- Here is an example of a robots.txt file from Google: https://www.google.com/robots.txt
- You can learn more about robots.txt here: https://www.robotstxt.org/robotstxt.html

REQUESTS

- Whenever we visit a website, our browsers will send a request to retrieve that website's HTML codes to display on our browser. The server of that website will respond by returning a response "object".


Browser
Server
Request
2. Sends a request for that website
1. Enter a website
3. Server sends HTML to browser if request is successful
Response

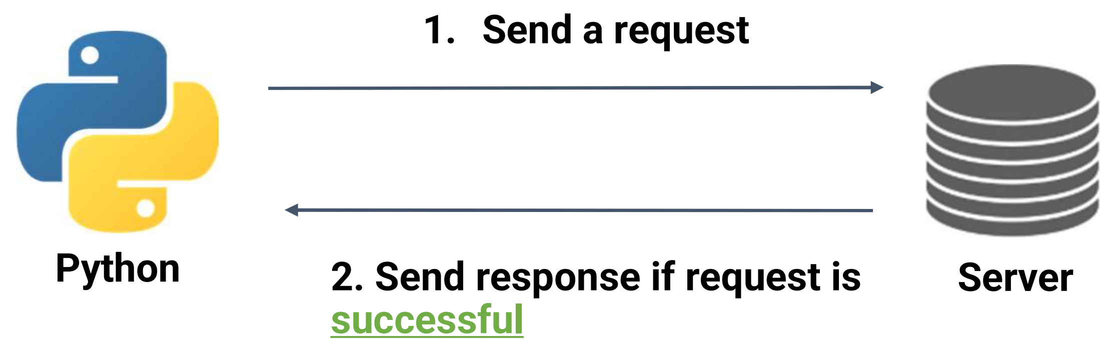
REQUESTS

- To access a website with Python, we will use the requests library to access it.

```
import requests

# Source 1: Credits at the end of the notebook
url1 = "https://www.google.com"

# Use requests.get(url) to send a request to the server and
get the contents. Store it in resp
resp = requests.get(url1)
```

REQUESTS

- When writing code with requests, python will use **requests** to send a request straight to the server, thus we can get the same response without opening the url via browser.



REQUESTS

- There are situation where the request is not successful. Thus, we can get the **status code** of the response using the following method.

```
resp.status_code
```

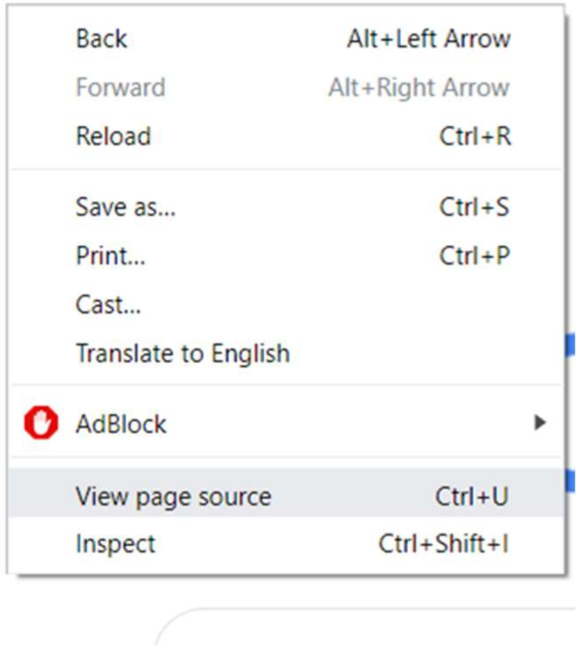
- Here are some of the common status codes :
    - 200 - Success
    - 404 - Not Found
    - 500 - Internal Server Error
    - 503 - Service Unavailable
    - etc.

REQUESTS

- We can get the html code of a webpage in the form of text, using **.text** method

```
resp.text
```

- We can also get the same result as **.text** by going to the website via browser, then click on view page source.



IDENTIFYING PATTERN

- When working on web scraping project, it is important to identify the behaviour/pattern of the webpage.
- For example, some websites share the same url across multiple pages :

```
"https://www.random-page.com/1"
"https://www.random-page.com/2"
"https://www.random-page.com/3"
"https://www.random-page.com/4"
```

IDENTIFYING PATTERN

- For websites that share the same url, we can make use of **for loop** to increase the efficiency of web scraping.

```
url = "https://bulma-low.now.sh/gaming/"
pages = [1,2]
for i in pages:
    # Create new URL
    new_url = url+ str(i)
    print(new_url)

    resp = requests.get(new_url)
```

Base URL

Concatenating page number to the URL

BEAUTIFUL SOUP

- By using **.text** from request library, this is what we will see (HTML Output):

```
<!doctype html><html itemscope=""
itemtype="http://schema.org/WebPage" lang="en-SG"><head><meta
content="text/html; charset=UTF-8" http-equiv="Content-
Type"><meta content="/logos/doodles/2019/50th-anniversary-of-
the-moon-landing-6524862532157440.2-l.png"
itemprop="image"><meta content="50th Anniversary of the Moon
Landing" property="twitter:title"><meta content="Celebrate 50
years since the moon landing by experiencing the journey in
today\'s out-of-this-world #GoogleDoodle!
…………………………………………………………………………………
```

BEAUTIFUL SOUP

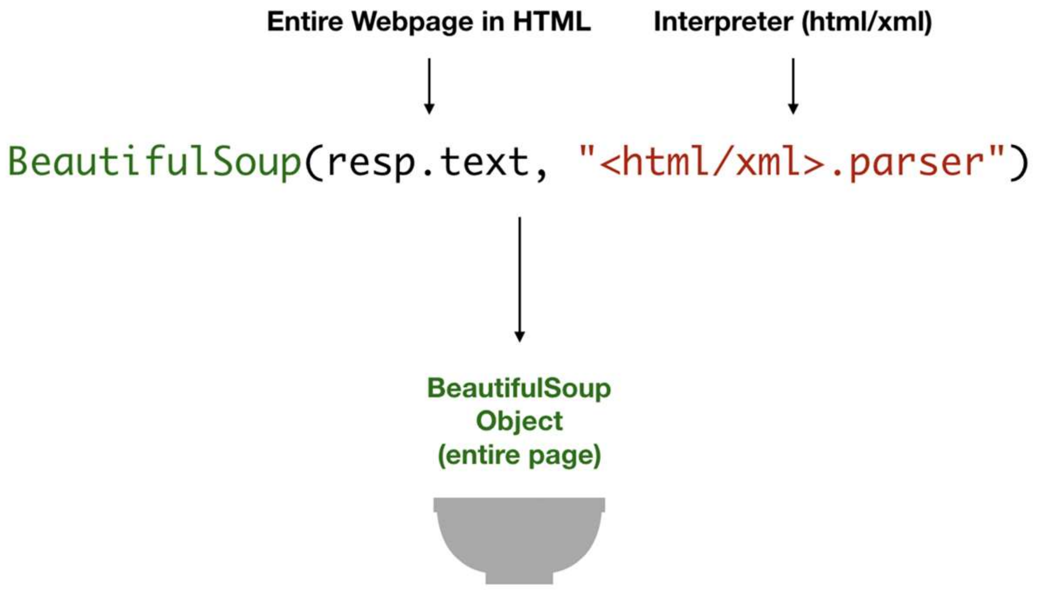
- As we see from the output, using **.text** will not be getting data that is usable for our analysis.

- We will need **BeautifulSoup** to process this HTML output, and help us to find relevant information on the website that is useful for the analysis

- **BeautifulSoup** is a powerful library from python, that helps to parse HTML/XML so that data can be extracted easily.

BEAUTIFUL SOUP

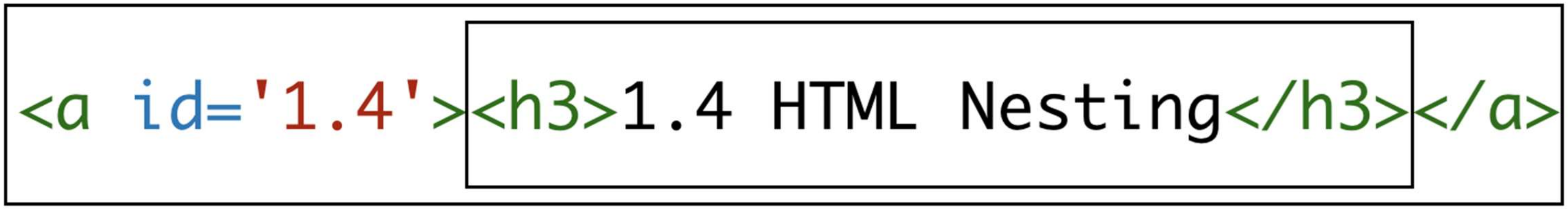
- As the name suggests, BeautifulSoup is an object that contains all of your data (soup), and just like your Alphabets soup, where search for favourite A-B-Cs, you can search for the data you want.



```
BeautifulSoup(resp.text, "<html/xml>.parser")
```

BEAUTIFUL SOUP

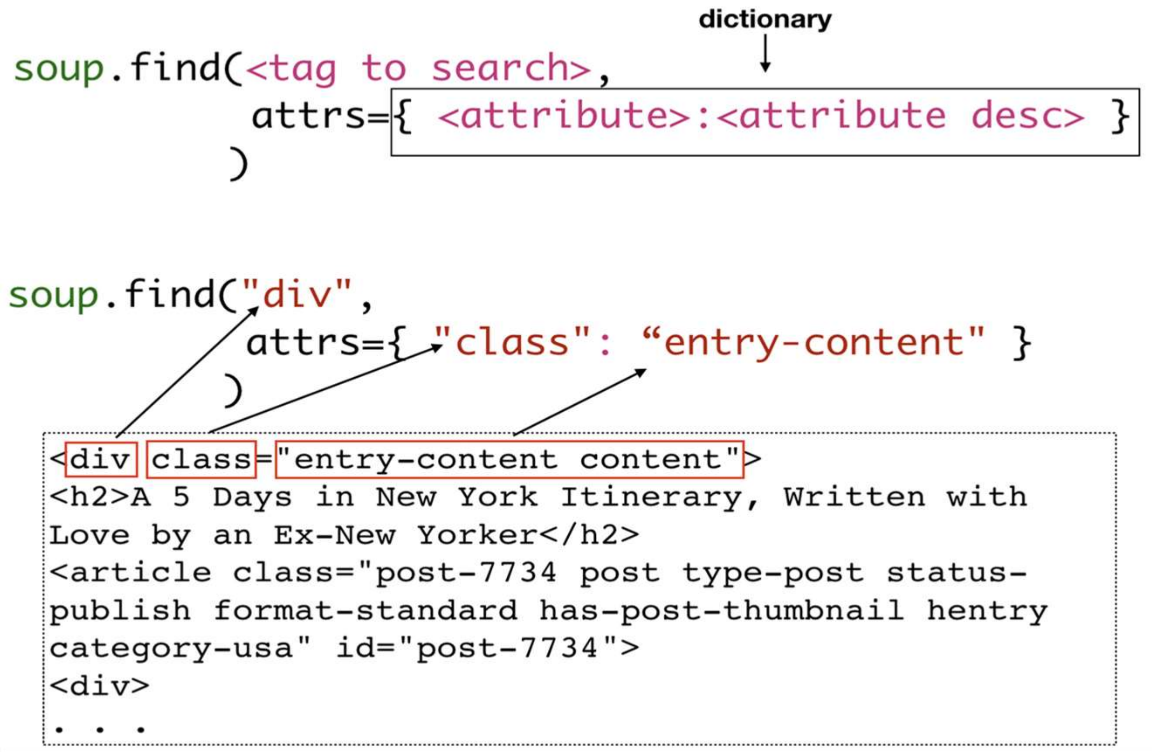
- We can use .**find()** to search within a soup object. To find these data, we need to recall the nesting of HTML codes. A web page is a large HTML nest, which within it nests smaller HTML nests.



```
<a id='1.4'><h3>1.4 HTML Nesting</h3></a>
```

BEAUTIFUL SOUP - FIND ONE

- To find a particular text/data, use the **.find()** method and search based on its attributes, where the attrs= parameter takes in a dictionary.



```
soup.find(<tag to search>,
          attrs={ <attribute>:<attribute desc> }
         )
```

```
soup.find("div",
          attrs={ "class": “entry-content" }
         )
```

```
<div class="entry-content content">
<h2>A 5 Days in New York Itinerary, Written with
Love by an Ex-New Yorker</h2>
<article class="post-7734 post type-post status-
publish format-standard has-post-thumbnail hentry
category-usa" id="post-7734">
<div>
. . .
```

BEAUTIFUL SOUP - FIND ONE

- In the following code, we are searching for tag where attribute is equal to “class” and value of the attribute should be “entry-content”.

```
new_york_entry = new_york_soup.find('div', attrs={'class' : 
'entry-content'})
```

- If multiple tags matched the search criteria, it will return the first text.

```
new_york_first_p = new_york_entry.find('p')
```

BEAUTIFUL SOUP - FIND ONE

- The result of the search will be known as “element tag”

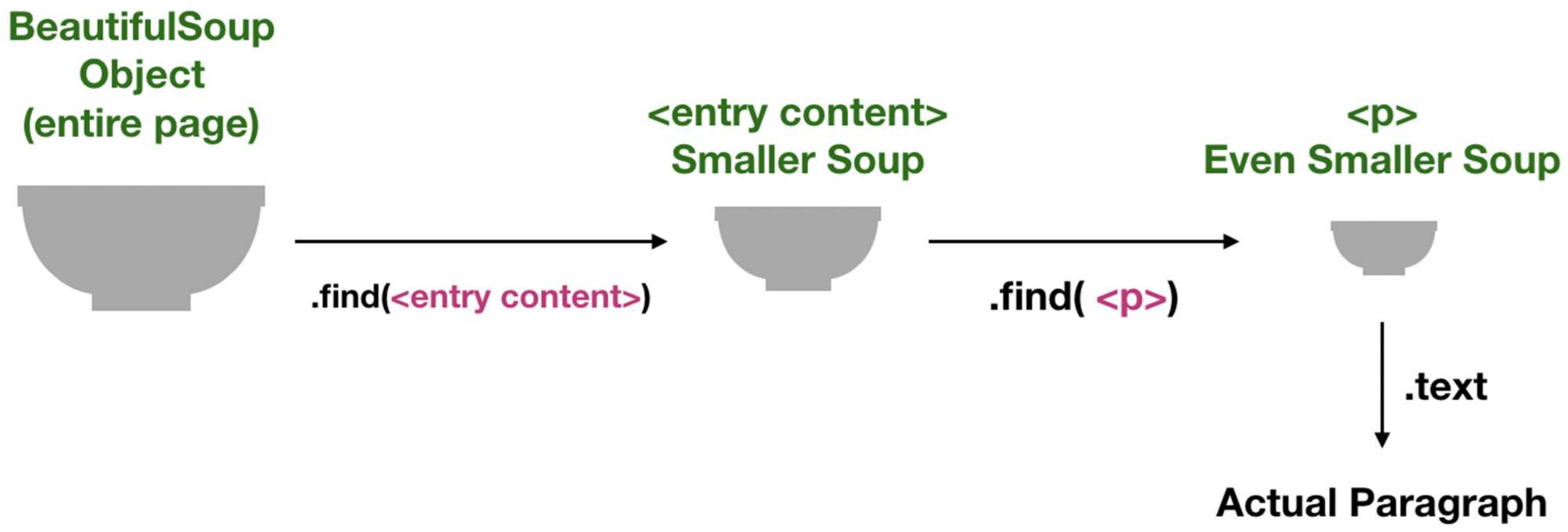
```
print(type(new_york_first_p)) # <class 'bs4.element.Tag'>
```

- We can use **.text** on element tag to get the text element of the search.

```
new_york_first_p.text
```

BEAUTIFUL SOUP - FIND ONE

- In a nutshell, this is how the search is done by BeautifulSoup.



BEAUTIFUL SOUP - FIND ALL

- We can also use **find_all()** if we wish to search for all matched results.

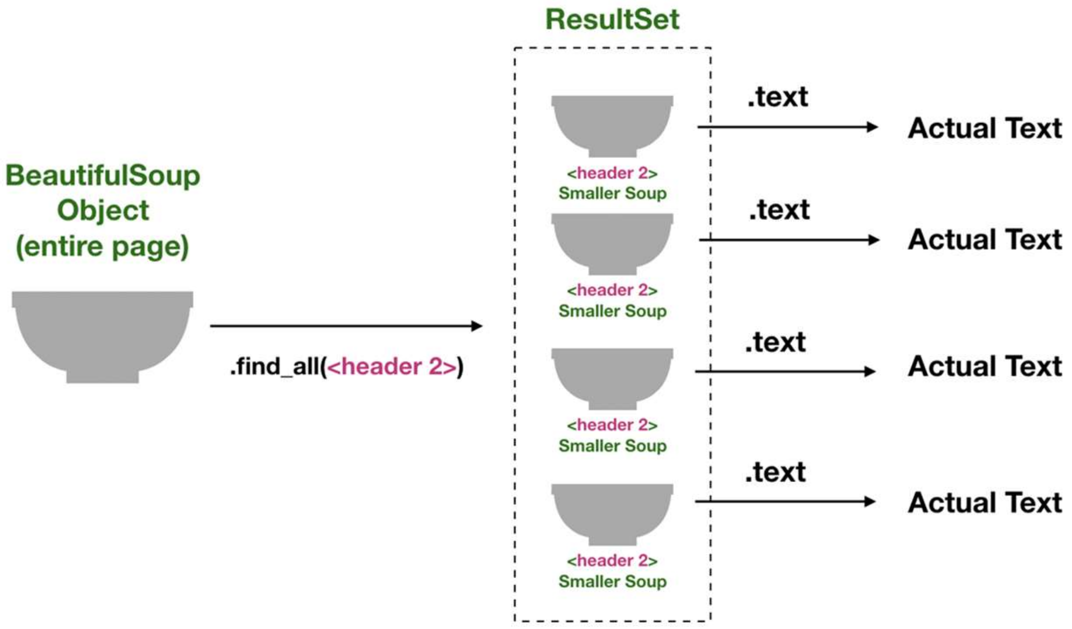
- **find_all()** will return a ResultSet, and we **cannot** use **.text** method on a result set.

```
title2_tags = new_york_soup.find_all('h2')
print(type(title2_tags)) # <class 'bs4.element.ResultSet'>
```

BEAUTIFUL SOUP - FIND ALL

- We need to use **for loop** to get the text of each element in the result set.

```
for t2 in title2_tags:
    print(t2.text)
```



SCRAPING TABLE - PD.READ HTML

- When scraping table on the web page, we can use **pd.read_html()** from **pandas** library.

- **read_html()** will search for ALL **<table>** on the web page, and return **a list of DataFrames**. Thus, we need to use index to access table from the result.

```
result = pd.read_html("https://bulma-
low.now.sh/gaming/2",attrs={"class":"evenrowsgray wikitable
sortable jquery-tablesorter"})
dota2_df = result[0] # first table in the web page
```

DATA SCIENCE 102:
Selenium

AGENDA

- Static vs Dynamic Web Pages
- Selenium
- Understanding Web Behaviour
- Web Driver
- Locating Elements
- Select
- Keys
- Limitations

STATIC VS DYNAMIC WEB PAGES

- A static web page is a web page that is delivered to the user's web browser exactly as stored.
- In other words, web page written with HTML codes only are all static. The content on these pages does not change, thus, we can simply scrape the content via *requests & BeautifulSoup*
- A dynamic web page is a web page where content is generated by web application.
- Most of the popular web pages are dynamic. We can interact with these webpages and the content will be then delivered by *Javascript* as we click/scroll.

SELENIUM

- *Selenium* is an open source library that automate web browser. It mimics human action on web browser, but at a much faster speed.

- There are two main usages of Selenium :

  - Testing
  - **Scraping**

SELENIUM

- When the content on a web page is hidden by *Javascript*, we cannot scrape information with just *requests & BeautifulSoup.*

- We need to make use the ***Selenium*** library, that mimics human behaviour on the browser.

- By interact with the browser with *Selenium,* we can unhide the content on the browser, then scrape it with *BeautifulSoup.*

UNDERSTAND WEB BEHAVIOUR

- We need to understand how a website react to human behaviour before scraping content from it

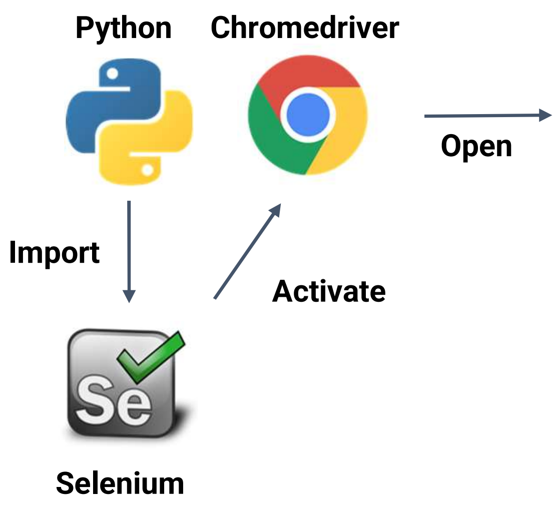
- For example, *Instagram* will only load more photos as we scroll down

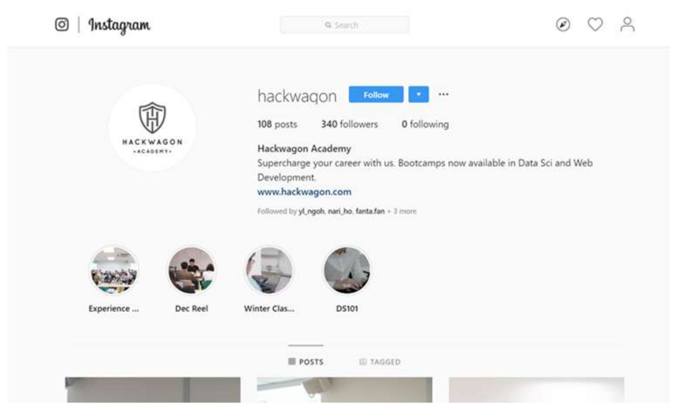
- Another example will be drop down list, where content changed as we select different options

- All these are cases where we will need *Selenium* to help us with Web Scraping

UNDERSTAND WEB BEHAVIOUR

- Below is an example of mindset we need to have when making use of selenium to scrape information from Instagram :




Instagram

Search

hackwagon Follow

108 posts 340 followers 0 following

Hackwagon Academy

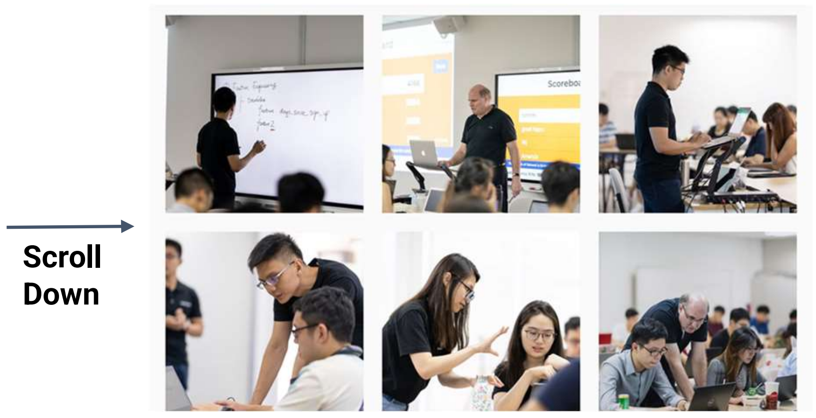
Supercharge your career with us. Bootcamps now available in Data Sci and Web Development.

www.hackwagon.com

Followed by yl_ngoh, nari_ho, fanta.fan + 3 more

Experience ... Dec Reel Winter Clas... DS101

POSTS TAGGED



WEBDRIVER

- We will be automating our web scraping task with the use of webdriver provided by *Selenium*.

- There are multiple web drivers available for different browsers:

    - **Google Chrome (ChromeDriver)**
    - Firefox (Firefox Driver)
    - Internet Explorer (Internet Explorer Driver)
    - etc.

ACTIVATING WEBDRIVER IN PYTHON

- We have downloaded the webdriver in *Lesson 1*, thus we can import *Selenium* then activate and direct it to the target url :

```
from selenium import webdriver

path_to_chromedriver = './chromedriver.exe' # Set your own
path if not in the same folder
driver = webdriver.Chrome(executable_path =
path_to_chromedriver)
url = "https://bulma-low.now.sh/hobby/1"
driver.get(url) # Get driver to open the url
```

LOCATING ELEMENTS

- As we learn in HTML, there are many tags being used to display a browser nicely. These tags are referred to as elements on the webdriver when scraping with *Selenium.*

- We can locate these elements using the following method:

    - find_elements_by_id
    - find_elements_by_class_name
    - find_elements_by_xpath
    - find_elements_by_tag_name

LOCATING ELEMENTS

- For locating single element, we can use the following method:

  - find_element_by_id
  - find_element_by_class_name
  - find_element_by_xpath
  - find_element_by_tag_name

- Note that if there are multiple elements exist, the first element that appeared on the browser will be returned

LOCATING ELEMENTS

- Here is an example of finding a select tag element in webdriver:

HTML :

```
<select id="type" class="form-control
selects_type"></select>
```

Python :

```
dropdownlist = driver.find_element_by_id("type")
```

GET ATTRIBUTE

- After locating the element, we can get the HTML of the located element using *get_attribute()* method

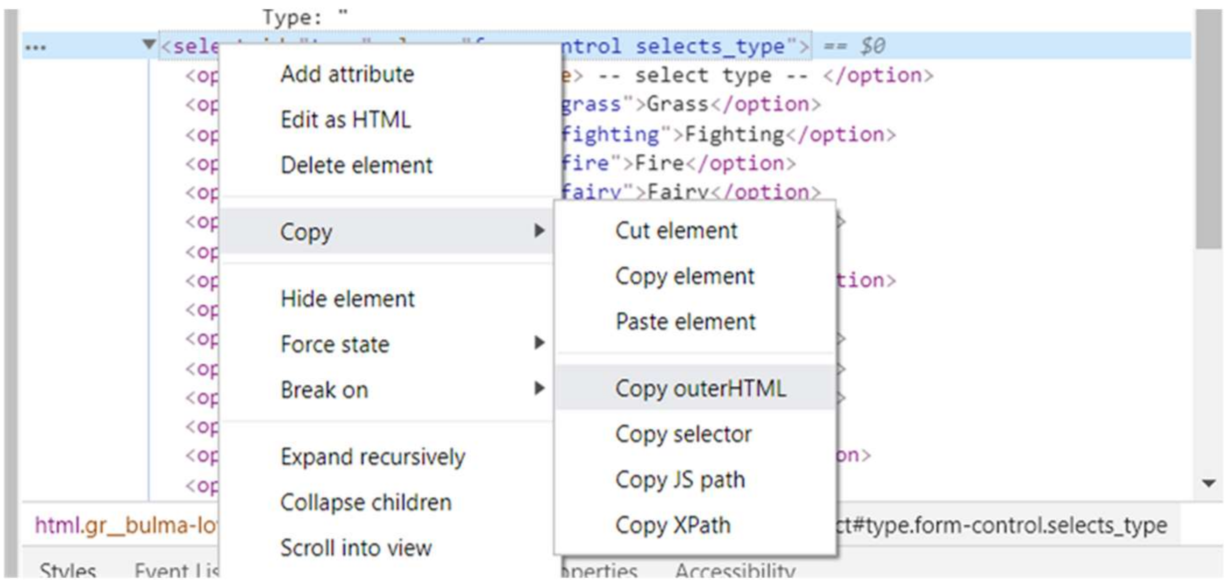
```
html_element = dropdownlist.get_attribute("outerHTML")
```

- Once we got the HTML of the located element, we can then apply what we've learnt earlier by parsing into *BeautifulSoup*

```
soup = BeautifulSoup(html_element)

for option in soup.find_all("option"):
    print(option.text)
```

GET ATTRIBUTE

- Note that we parsed *"outerHTML"* into get attribute. This is the same as going to browser and inspect the element, then right click to copy the outerHTML.



INNER VS OUTER HTML

- We can also parse *"innerHTML"* into *get_attribute()* method. Here is an example to explain the differences between both :

```
<div>
<p>Hello World</p>
</div>
```

- **innerHTML** : if you do innerHTML on div, it will return object containing <p> only.

- **outerHTML**: if you do outerHTML on div you will get an object containing <div> and <p> inside that.

SELECT

- *Selenium* also allowed us to “click” on a drop down list.
- *Select* is a class from *Selenium* that develop to realise this action
- *select_by_index()* method from *Select* allowed us to select from the start of the index until the end, without knowing what option is it
- Just like what we learnt in list, index starts from **0**

SELECT

- We first determine the length of the selection:

```
all_options = driver.find_elements_by_tag_name('option')

total_number_of_options = len(all_options)
```

- Once we determine the length, we will also identify the element we wish to apply *Select* on:

```
select = Select(dropdownlist)
```

SELECT

- *select_by_index()* will do the rest of the jobs for us, together with *for loop* :

```
result= []

for index in range(total_number_of_options):

    select = Select(dropdownlist) # determine the located element

    select.select_by_index(index) # choose the option via index

      # Scraping with pd.read_html() #...
```

KEYS & ACTIONCHAINS

- We mentioned about scrolling behaviour needed in website like Instagram.

- This action can be perfectly perform by *Keys*, together with *ActionChains*

- *Keys* allowed us to input any keys on keyboard onto webdriver

- *ActionChains* allowed us to perform *Keys* action without keeping the windows active (works even with browser is minimized)

KEYS & ACTIONCHAINS

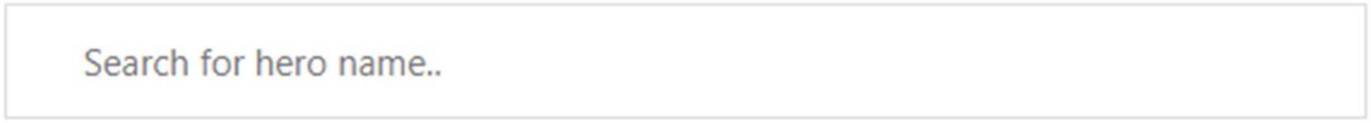
- We can choose almost all keys on a traditional keyboard.
- PAGE_DOWN is an example to perform scrolling down behaviour
- Here are some examples of other possible *Keys*:
    - ARROW_DOWN
    - ESCAPE
    - DELETE
    - ENTER
    - etc.

KEYS & ACTIONCHAINS

- Similar to *Select*, we can locate an element to perform Keys action using *send_keys()*

```
element = driver.find_element_by_id("myInput")

element.send_keys("mage")
```

- This is most commonly used when we want to scrape information from the result of a search bar



KEYS & ACTIONCHAINS

- If it’s a general action that do not need to perform on any element (PAGE_DOWN etc.), we can do it with *ActionChains*

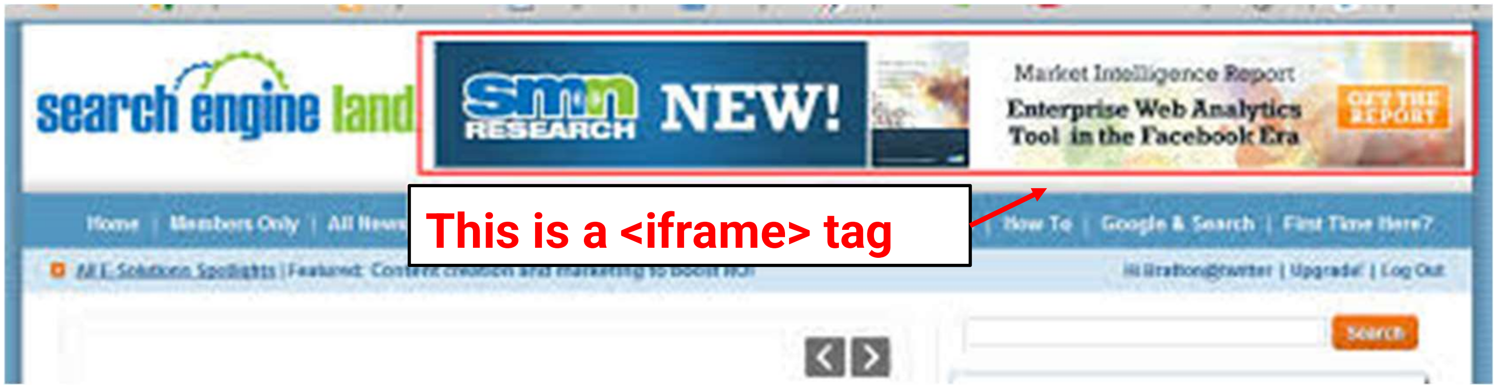
```
actions = ActionChains(driver) # set ActionChains to perform
on our webdriver

actions.send_keys(Keys.PAGE_DOWN) # actions will be sending
a PAGE_DOWN keys

actions.perform() # perform the actions
```

IFRAME

- Iframe is a HTML tag that is commonly used to place advertisement on the website.

```
<iframe src="www.google.com"></iframe>
```

- Example of <iframe> on a website:

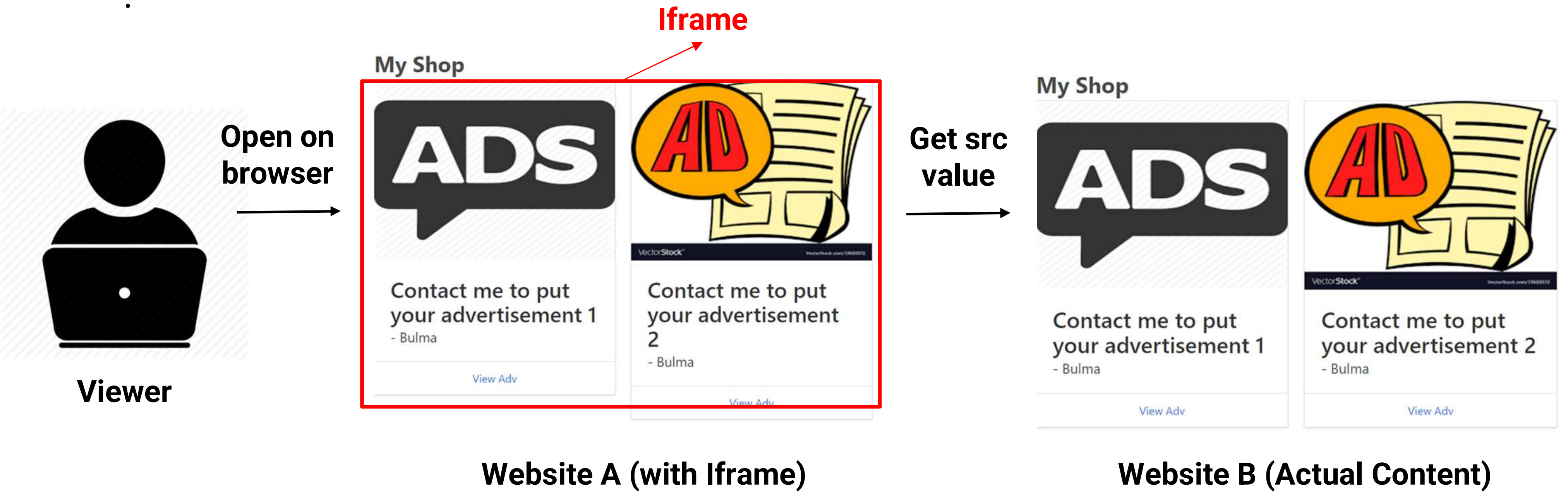

search engine land
SMN RESEARCH NEW!
Market Intelligence Report
Enterprise Web Analytics Tool in the Facebook Era
GET THE REPORT
This is a <iframe> tag
Home | Members Only | All News
How To | Google & Search | First Time Here?
Search

IFRAME

- When scraping information on <iframe>, we cannot simply do it with *BeautifulSoup or get_attribute()*
- Information on iframe that we see on the web page is actually not part of the HTML content
- It is actually stored in another web page, where the link is stated in the attribute “src”
- Thus, we need to get the url by getting the value of “src”

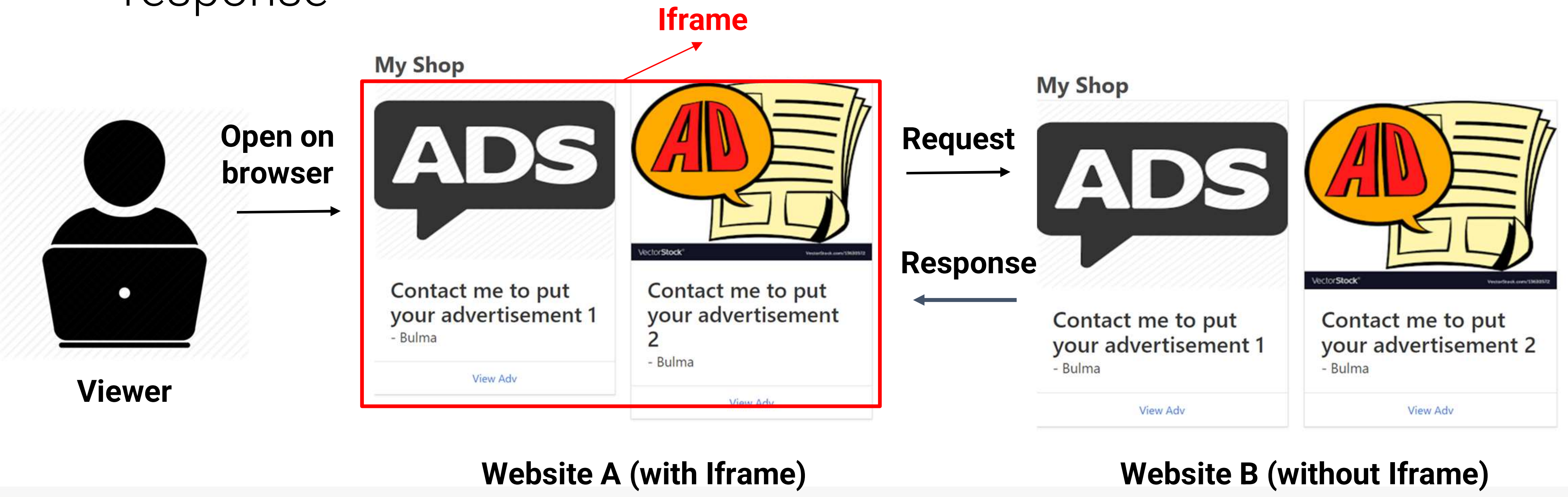
IFRAME

- The mindset needed to scrape information from <iframe> :



IFRAME

- Though the contents looks the same, the HTML of website A does not stored any of it. Website A simply make request to Website B on “src” value and get content from the response


Iframe
My Shop
Open on browser
Viewer
ADS
AD
Contact me to put your advertisement 1
- Bulma
View Adv
Contact me to put your advertisement 2
- Bulma
View Adv
Request
Response
My Shop
ADS
AD
Contact me to put your advertisement 1
- Bulma
View Adv
Contact me to put your advertisement 2
- Bulma
View Adv
Website A (with Iframe)
Website B (without Iframe)

LIMITATIONS

- *Selenium* seems to be a great solution for web scraping. However, it is time consuming as we need to keep python running through many webpages to get the desire content.

- Thus, when working on web scraping with *Selenium*, please make sure that the data scraped are written into file, instead of just storing in DataFrame or list.

- This will save up our laptop memory and ensure that data won't be lost once it's scraped even when terminal crashed.

THANK YOU!

HACKWAGON

• ACADEMY •