# The Start:

Web scraping is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications. There are many different ways to perform web scraping to obtain data from websites. These include using online services, particular API’s or even creating your code for web scraping from scratch.

# Three major Libraries:

1. BeautifulSoup: Slow but beginner friendly.
2. Selenium: slow but better than BeautifulSoup. Can be used for small projects
3. Scrapy: written in python and very fast. Go used to large projects

# Python Installation and Pycharm installation:

1. Python: <https://www.python.org/downloads/>
2. Pycharm: <https://www.jetbrains.com/help/pycharm/installation-guide.html> (Bonus tip: once installed, to import packages you can press ctrl+alt+s in windows or Mac: cmd+semicolon to open a window from where you can install the different packages)

# Python Basics for Web scrapping:

1. Learn about list, opening file, dictionary, operations on file, loop statements, pandas dataframe

# HTML for web scrapping:

1. There is one new tag in HTML5 i.e., article tag and is used to add codes, books etc

# How to find element in Web scrapping:

1. General rule is depicted below;
   1. ID
   2. Class name
   3. Tag name, CSS selector
   4. Xpath

# Implementing web scrapping using beautiful soup

1. Using bs4 to import beautifulsoup
2. Then import requests.
3. We will use a parser lxml for this basic parser

# Scrapping single page using beautifulsoup:

1. Fetching the page using using request .get method.
2. Get the page context using .text
3. Create soup using object of a BeautifulSoup class.
4. Now you hve soup as an object for the beautifulsoup, you can use different functions for finding different tags or classes etc.
5. For ex Soup.find(id=’specific\_id’), or by using the class name soup.find(‘tagname’, class\_=’classname’)
6. To locate multiple records by tag name, use Soup.find\_all(‘h2’) ///PS: find\_all will return a list and find will return an element
7. **Complete code in pycharm**
8. Just to keep up you can use inspect element in chrome and look out for arrow sign on top left and locate the box or part from which you want to extract

# Scrapping multiple pages using beautiful soup:

### codes works fine in jupyter notebook  
  
from bs4 import BeautifulSoup  
import requests  
  
root\_url='https://subslikescript.com'  
website\_hp=f'{root\_url}/movies'  
response=requests.get(website\_hp)  
content=response.text  
soup=BeautifulSoup(content,'lxml')  
box=soup.find('article',class\_='main-article')  
links=[]  
for link in box.find\_all('a',href=True):  
 links.append(link['href'])  
print(links)  
  
for link\_all in links:  
 website\_single\_page=f'{root\_url}/{link\_all}'  
 response1=requests.get(website\_single\_page)  
 content\_all=response1.text  
 soup\_all=BeautifulSoup(content\_all,'lxml')  
 box\_title=soup\_all.find('h1').get\_text()  
 trans=soup\_all.find('div',class\_='full-script').get\_text(strip=True,separator='')  
 with open(f'{box\_title}.txt','w',encoding="utf-8") as file:  
 print(box\_title)  
 file.write(trans)

# Handling Pagination in beautifulsoup:

### codes works fine in jupyter notebook  
  
from bs4 import BeautifulSoup  
import requests  
  
root\_url='https://subslikescript.com'  
website\_hp=f'{root\_url}/movies\_letter-A'  
response=requests.get(website\_hp)  
content=response.text  
soup=BeautifulSoup(content,'lxml')  
### locate the box where page number is written  
box\_page=soup.find('ul', class\_='pagination')  
pages=box\_page.find\_all('li', class\_='page-item')  
last\_page=pages[-2].text ## this is the last page number on the website  
  
for page in range(1,int(last\_page)+1):  
 result = requests.get(f'{website\_hp}?page={page}') ## structuring according to page name  
 content=result.text  
 soup=BeautifulSoup(content, 'lxml')  
 box=soup.find('article',class\_='main-article')  
 links=[]  
 for link in box.find\_all('a',href=True):  
 links.append(link['href'])  
  
 for link\_all in links:  
 try:  
 website\_single\_page=f'{root\_url}/{link\_all}'  
 response1=requests.get(website\_single\_page)  
 content\_all=response1.text  
 soup\_all=BeautifulSoup(content\_all,'lxml')  
 box\_title=soup\_all.find('h1').get\_text()  
 trans=soup\_all.find('div',class\_='full-script').get\_text(strip=True,separator='')  
 with open(f'{box\_title}.txt','w',encoding="utf-8") as file:  
 print(box\_title)  
 file.write(trans)  
 except:  
 print("this link doesn't work ",link\_all)

# Xpath Guid:

1. Xpath is used to scrap web data using selenium or scrapy.
2. Xpath syntax contains tagname, attribute name, attribute value.
3. Normal syntax: //tagname, for example selecting H1 is via //h1, putting a square bracket and specifying the number will give which numbered tag should be used suppose it has 2 H1 tag then //H1[1] or //H1[2]
4. Other examples: //tagname[@attributename=’Value’] for ex: //article[@class=”main-article”]
5. There are different functions which you can use for Xpath listed below:
   1. Look out for cheat sheet in the document
6. <https://scrapinghub.github.io/xpath-playground/> (website to practice xpath)

# Special Expression to build Xpath:

1. //article/h1/text(): here we will see the use of single and double slash :: hierarchy is first put double slash to get the root element and then use single slash to get the nearest next tag then the next one will be, in this case we have some text under h1 tag so using text() gives us required text
2. So basicaaly // will help us to locate any level in the document; suppose using //article//text() will gives us text at any level in the article tag
3. We have single dot(.) and double dot(..) character; so basically in single dot will give the current node but using double dot will give the parent node
4. \* this character is wildcard character and is used to get al the elements in the current parent node.

# Introduction to selenium

## To identify Javascript driven website

1. To identify first load the website 🡪 got to inspect element🡪 go to setting🡪 in debugger section disable javascript🡪 reload the website (if it doesn’t comes up it is javascript driven)

## Installing selenium and ChromeDriver

1. Need to install chromedriver of same version as of chrome

## Working around selenium

1. We use driver object here to find the element, id, class, etc.
2. driver.find\_element\_by\_tag\_name(‘h1’), driver.find\_element\_by\_tag\_name(‘p’), etc.
3. driver.find\_element\_by\_class\_name(‘full-script’),etc.
4. driver.find\_element\_by\_xpath(‘//tag\_name[@attributename=”value”])
5. there are other way too driver.find\_element\_by\_css\_selector(),driver.find\_element\_by name(),driver.find\_element\_by\_link\_text()
6. if you wants to get the multiple elements so in above code just put plural form of element i.e., elements and you are good to go; just remember this will return a list of element not an element.

### Clicking on a button using Selenium:

from selenium import webdriver  
website='https://www.adamchoi.co.uk/overs/detailed'  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path)  
driver.get(website)  
all\_matched\_button=driver.find\_element\_by\_xpath('//label[@analytics-event="All matches"]') ## this is to locate the button  
all\_matched\_button.click() ### this is to click on the button

### Getting contents of the table

from selenium import webdriver  
website='https://www.adamchoi.co.uk/overs/detailed'  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path)  
driver.get(website)  
all\_matched\_button=driver.find\_element\_by\_xpath('//label[@analytics-event="All matches"]') ## this is to locate the button  
all\_matched\_button.click() ### this is to click on the button  
### now we will look into how to extract content of a table using selenium  
table\_row=driver.find\_elements\_by\_tag\_name('tr') # this will be a list containing sleeium elements onces you write .text in front of that object you will get what exactly si there in that object  
date=[]  
home\_team=[]  
score=[]  
away\_team=[]  
for table\_row\_cont in table\_row:  
 ## in python the indexes stat from 0 but in table elements it starts with 1  
 date.append(table\_row\_cont.find\_element\_by\_xpath('./td[1]').text) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
 home=table\_row\_cont.find\_element\_by\_xpath('./td[2]').text  
 print(home)  
 home\_team.append(home) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
 score.append(table\_row\_cont.find\_element\_by\_xpath('./td[3]').text) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
 away\_team.append(table\_row\_cont.find\_element\_by\_xpath('./td[4]').text) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td

to download in a csv you can just use the list formed to generate the dataframe using pnadas

### Selecting a dropdown using Selenium

# to select the dropdown using selenium lets do it in this manner  
from selenium import webdriver  
from selenium.webdriver.support.ui import Select  
import time  
import pandas as pd  
  
website='https://www.adamchoi.co.uk/overs/detailed'  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path)  
driver.get(website)  
all\_matches=driver.find\_element\_by\_xpath('//label[@analytics-event="All matches"]')  
  
all\_matches.click()  
  
dropdown=Select(driver.find\_element\_by\_id('country'))  
dropdown.select\_by\_visible\_text('Spain')  
  
time.sleep(**3**)  
  
tabel\_cont=driver.find\_elements\_by\_tag\_name('tr')  
date=[]  
home\_team=[]  
score=[]  
away\_team=[]  
for table\_row\_cont in tabel\_cont:  
 ## in python the indexes stat from 0 but in table elements it starts with 1  
 date.append(table\_row\_cont.find\_element\_by\_xpath('./td[1]').text) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
 home=table\_row\_cont.find\_element\_by\_xpath('./td[2]').text  
 print(home)  
 home\_team.append(home) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
 score.append(table\_row\_cont.find\_element\_by\_xpath('./td[3]').text) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
 away\_team.append(table\_row\_cont.find\_element\_by\_xpath('./td[4]').text) ## since table\_row\_cont contains rows context just wants to extract the lower value of tr i.e. td  
  
driver.quit()  
  
df=pd.DataFrame({'date':date**,**'home\_team':home\_team**,**'score':score**,**'away\_team':away\_team})  
df.to\_csv('football\_allscore\_spain.csv'**,**index=False)

# Extracting data from audible site

## Building the bot

from selenium import webdriver  
import pandas as pd  
website="https://www.audible.com/search"  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path)  
driver.get(website)  
driver.maximize\_window()  
  
container=driver.find\_element\_by\_class\_name("abdl-impression-container ")  
products=container.find\_elements\_by\_xpath('.//li[contains(@class, "productListItem")]')  
  
book\_title=[]  
book\_author=[]  
book\_runtime=[]  
for product in products:  
 book\_title.append(product.find\_element\_by\_xpath('.//h3[contains(@class,"bc-heading")]').text)  
 book\_author.append(product.find\_element\_by\_xpath('.//li[contains(@class,"authorLabel")]').text)  
 book\_runtime.append(product.find\_element\_by\_xpath('.//li[contains(@class,"runtimeLabel")]').text)  
  
driver.quit()  
  
book\_dataframe=pd.DataFrame({"book\_title":book\_title**,**"book\_author":book\_author**,**"book\_runtime":book\_runtime})  
book\_dataframe.head()

## Headless Mode:

So selenium has an ‘option’ library where it does not open a separate window for getting the data and whole process run in the background

from selenium import webdriver  
from selenium.webdriver.chrome.options import Options  
import pandas as pd  
  
option=Options()  
option.headless=True  
option.add\_argument('window-size=1920x1080')  
  
website="https://www.audible.com/search"  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path**,**options=option)  
driver.get(website)  
driver.maximize\_window()  
  
container=driver.find\_element\_by\_class\_name("abdl-impression-container ")  
products=container.find\_elements\_by\_xpath('.//li[contains(@class, "productListItem")]')  
  
book\_title=[]  
book\_author=[]  
book\_runtime=[]  
for product in products:  
 book\_title.append(product.find\_element\_by\_xpath('.//h3[contains(@class,"bc-heading")]').text)  
 book\_author.append(product.find\_element\_by\_xpath('.//li[contains(@class,"authorLabel")]').text)  
 book\_runtime.append(product.find\_element\_by\_xpath('.//li[contains(@class,"runtimeLabel")]').text)  
  
driver.quit()  
  
book\_dataframe=pd.DataFrame({"book\_title":book\_title**,**"book\_author":book\_author**,**"book\_runtime":book\_runtime})  
book\_dataframe.head()

## Handling Pagination:

from selenium import webdriver  
from selenium.webdriver.chrome.options import Options  
import pandas as pd  
import time  
  
option=Options()  
option.headless=False  
# option.add\_argument('window-size=1920x1080')  
  
website="https://www.audible.com/search"  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path**,**options=option)  
driver.get(website)  
driver.maximize\_window()  
  
### pagination  
pagination=driver.find\_element\_by\_xpath('//ul[contains(@class,"pagingElements")]')  
pages=pagination.find\_elements\_by\_tag\_name('li')  
  
last\_page=int(pages[-**2**].text)  
current\_page=**1**book\_title=[]  
book\_author=[]  
book\_runtime=[]  
while current\_page<=last\_page:  
 time.sleep(**3**)  
 container = driver.find\_element\_by\_class\_name("adbl-impression-container ")  
 products = container.find\_elements\_by\_xpath('.//li[contains(@class, "productListItem")]')  
 for product in products:  
 book\_title.append(product.find\_element\_by\_xpath('.//h3[contains(@class,"bc-heading")]').text)  
 book\_author.append(product.find\_element\_by\_xpath('.//li[contains(@class,"authorLabel")]').text)  
 book\_runtime.append(product.find\_element\_by\_xpath('.//li[contains(@class,"runtimeLabel")]').text)  
  
 current\_page+=**1** try:  
 next\_page = driver.find\_element\_by\_xpath('//span[contains(@class,"nextButton")]') ## this is for moving to next page  
 next\_page.click()  
 except:  
 pass  
driver.quit()  
  
book\_dataframe=pd.DataFrame({"book\_title":book\_title**,**"book\_author":book\_author**,**"book\_runtime":book\_runtime})  
book\_dataframe.to\_csv("book\_data.csv"**,**index=False)

## Implicit or Explicit waits

In javascript the content to the website is loaded dynamically so it can take some time to load fully so in python we have a functionality where we can let the driver to sleep for few seconds by usning time library and this is hence called **implicit waits**

For Explicit waits we have to load few more libraries:

1. from selenium.webdriver.common.by import By
2. from selenium.webdriver.support.ui import WebdriverWait
3. from selenium.webdriver.support import expected\_conditions as Ed

condition is described as WebdriverWait(driver,5).until(EC.presence\_of\_element\_located((By.CLASS\_NAME,’value’))

here in place of class name it can be xpath etc

from selenium import webdriver  
from selenium.webdriver.chrome.options import Options  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC  
import pandas as pd  
import time  
  
option=Options()  
option.headless=False  
# option.add\_argument('window-size=1920x1080')  
  
website="https://www.audible.com/search"  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
driver=webdriver.Chrome(path**,**options=option)  
driver.get(website)  
driver.maximize\_window()  
  
### pagination  
pagination=driver.find\_element\_by\_xpath('//ul[contains(@class,"pagingElements")]')  
pages=pagination.find\_elements\_by\_tag\_name('li')  
  
last\_page=int(pages[-**2**].text)  
current\_page=**1**book\_title=[]  
book\_author=[]  
book\_runtime=[]  
while current\_page<=last\_page:  
 time.sleep(**3**) ### implicit waits  
 ## explicit wait  
 container = WebDriverWait(driver**,5**).until(EC.presence\_of\_element\_located((By.CLASS\_NAME**,**'adbl-impression-container ')))  
 products = WebDriverWait(container**,5**).until(EC.presence\_of\_all\_elements\_located((By.XPATH**,**'.//li[contains(@class, "productListItem")]')))  
  
 # container = driver.find\_element\_by\_class\_name("adbl-impression-container ")  
 # products = container.find\_elements\_by\_xpath('.//li[contains(@class, "productListItem")]')  
 for product in products:  
 book\_title.append(product.find\_element\_by\_xpath('.//h3[contains(@class,"bc-heading")]').text)  
 book\_author.append(product.find\_element\_by\_xpath('.//li[contains(@class,"authorLabel")]').text)  
 book\_runtime.append(product.find\_element\_by\_xpath('.//li[contains(@class,"runtimeLabel")]').text)  
  
 current\_page+=**1** try:  
 next\_page = driver.find\_element\_by\_xpath('//span[contains(@class,"nextButton")]') ## this is for moving to next page  
 next\_page.click()  
 except:  
 pass  
driver.quit()  
  
book\_dataframe=pd.DataFrame({"book\_title":book\_title**,**"book\_author":book\_author**,**"book\_runtime":book\_runtime})  
book\_dataframe.to\_csv("book\_data.csv"**,**index=False)

when should you use implicit or explicit waits?

Use implicit wait when testing out code but explicit wit when code is final and its need to be executed

# Extracting data from Twitter website

## Login to the website

from selenium import webdriver  
  
web = "https://twitter.com/"  
path=r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
  
driver=webdriver.Chrome(path)  
driver.get(web)  
driver.maximize\_window()  
  
login=driver.find\_element\_by\_xpath("//a[@href='/login']")  
login.click()  
  
time.sleep(**2**)  
username=driver.find\_element\_by\_xpath('//input[@name="text"]')  
username.send\_keys("youremail")  
  
### click on next button  
next\_b=driver.find\_element\_by\_xpath("//div[@role='button']//span[text()='Next']")  
next\_b.click()  
### finding password area  
password=driver.find\_element\_by\_xpath('//input[@name="password"]')  
username.send\_keys("yourpassword”)  
  
## loging in  
login=driver.find\_element\_by\_xpath("//div[@role='button']//span[text()='Log in']")  
login.click()

## Building a function for scrapping infinite scroll

from selenium import webdriver  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC  
import pandas as pd  
import time  
  
web = "https://twitter.com/i/flow/login"  
path = r"C:\Users\Admin\Desktop\web scrapping\chromedriver"  
  
driver = webdriver.Chrome(path)  
driver.get(web)  
driver.maximize\_window()  
time.sleep(**5**)  
  
# login=driver.find\_element\_by\_xpath("//a[@href='/login']")  
# login.click()  
  
# time.sleep(2)  
username = driver.find\_element\_by\_xpath('//input[@name="text"]')  
username.send\_keys("")  
  
### click on next button  
next\_b = driver.find\_element\_by\_xpath("//div[@role='button']//span[text()='Next']")  
next\_b.click()  
  
time.sleep(**5**)  
#### enter username  
username\_secondwindow = driver.find\_element\_by\_xpath('//input[@autocapitalize="none"]')  
username\_secondwindow.send\_keys("")  
  
### click on next button  
next\_b = driver.find\_element\_by\_xpath("//div[@role='button']//span[text()='Next']")  
next\_b.click()  
  
time.sleep(**5**)  
### finding password area  
password = driver.find\_element\_by\_xpath('//input[@name="password"]')  
password.send\_keys("")  
  
## loging in  
login = driver.find\_element\_by\_xpath("//div[@role='button']//span[text()='Log in']")  
login.click()  
  
time.sleep(**5**)  
  
### clicking on search box  
search\_icon = driver.find\_element\_by\_xpath("//a[@href='/explore']")  
search\_icon.click()  
  
time.sleep(**10**)  
#### writing content to search  
search\_area = driver.find\_element\_by\_xpath("//form[@role='search']//input[@placeholder='Search']")  
search\_area.click()  
search\_area.send\_keys("python")  
search\_area.submit() ### to use search option in a web  
  
time.sleep(**20**)  
  
tweet\_box = WebDriverWait(driver**, 20**).until(  
 EC.presence\_of\_all\_elements\_located((By.XPATH**,** "//article[@role='article']")))  
user\_name = []  
written\_tweet = []  
  
  
def get\_tweet(element):  
 try:  
 user = element.find\_element\_by\_xpath(".//span[contains(text(),'@')]").text  
 wtweet = element.find\_element\_by\_xpath(".//div[@lang]").text  
 list\_tweet = [user**,** wtweet]  
 except:  
 list\_tweet=['user'**,** 'tweet']  
  
 return list\_tweet  
  
  
### we are just finding username, etc  
for tweet in tweet\_box:  
 list\_tweet\_actual = get\_tweet(tweet)  
 user\_name.append(list\_tweet\_actual[**0**])  
 written\_tweet.append(" ".join(list\_tweet\_actual[**1**].split()))  
  
driver.quit()  
  
df = pd.DataFrame({"user\_name": user\_name**,** "written\_tweet": written\_tweet})  
df.to\_csv("tweets.csv"**,** index=False)

## Building function for infinite scrolling

Trick to do infinite scrolling:

last\_height=driver.execute\_script("return document.body.scrollHeight")  
  
while True:  
 driver.execute\_script("window.scrollTo(0,document.body.scrollHeight);")  
 time.sleep(**5**)  
 new\_height=driver.execute\_script("return document.body.scrollHeight")  
 if new\_height==last\_height:  
 break  
 else:  
 last\_height=new\_height

## For infinite scrolling

from selenium import webdriver  
import time  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC  
import pandas as pd  
  
web = "https://twitter.com/TwitterSupport/status/1415364740583395328"  
# web = "https://twitter.com/TwitterSupport"  
path = '/Users/frank/Downloads/chromedriver'  
driver = webdriver.Chrome(path)  
driver.get(web)  
driver.maximize\_window()  
  
def get\_tweet(element):  
 try:  
 user = element.find\_element\_by\_xpath(".//span[contains(text(), '@')]").text  
 text = element.find\_element\_by\_xpath(".//div[@lang]").text  
 tweet\_data = [user**,** text]  
 except:  
 tweet\_data = ['user'**,** 'text']  
 return tweet\_data  
  
user\_data = []  
text\_data = []  
tweet\_ids = set() ### for getting only unique tweets  
scrolling=True  
while scrolling:  
 tweets = WebDriverWait(driver**, 5**).until(  
 EC.presence\_of\_all\_elements\_located((By.XPATH**,** "//article[@role='article']")))  
 print(len(tweets))  
 for tweet in tweets[-**15**:]:  
 tweet\_list = get\_tweet(tweet)  
 tweet\_id = ''.join(tweet\_list)  
   
 if tweet\_id not in tweet\_ids:  
 tweet\_ids.add(tweet\_id)  
 user\_data.append(tweet\_list[**0**])  
 text\_data.append(" ".join(tweet\_list[**1**].split()))  
  
 # Get the initial scroll height  
 last\_height = driver.execute\_script("return document.body.scrollHeight")  
 while True:  
 # Scroll down to bottom  
 driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")  
 # Wait to load page  
 time.sleep(**2**)  
 # Calculate new scroll height and compare it with last scroll height  
 new\_height = driver.execute\_script("return document.body.scrollHeight")  
 # condition 1  
 if new\_height == last\_height: # if the new and last height are equal, it means that there isn't any new page to load, so we stop scrolling  
 scrolling = False  
 break  
 # condition 2  
 # if len(data) > 60: ### use this statement as break one from above if you want certain number of elements  
 # scrolling = False  
 # break  
 else:  
 last\_height = new\_height  
 break

# Scrapy

1. So scrapy can be used with anaconda much easily
2. Scrapy command can be put on anaconda terminal and its easy to use there.
3. Scrapy bench ## to get how scrapy works on the machine
4. Scrapy fetch <https://google.com/> ## to fetch the html url markup of the given website

In order to see what all commands are available for scrapy just put the word scrapy in anaconda terminal and get the command that exist but few commands are already listed above.

## How to start a project in scrapy

1. Write scrapy startproject pojectname on anaconda prompt
2. It will ask to change directory
3. Change the directory through anaconda prompt
4. See the folder path and go to the folder and click on spider folder and look out for the project name
5. Open that in pycharm if required.

## Various aspect of scrapy:

1. Scrapy has two thing : 1. Spider (extensively use for web scrapping) other is crawler (used for indexing web pages not much use in scraping)
2. In scrapy we have response as a function similar to driver or soup in selenium or beautiful soup respectively.
3. In scrapy the scrapping can be done only through xpath
4. response.xpath().get() & response.xpath().getall() //will give list
5. we will use yield as one of the other function to get the data.

## Scrapy Shell command

1. Write scrapy shell
2. Will get the window which tell you are in terminalmode
3. Write r=scrapy.Request(url=”valueof url”)
4. fetch( r )
5. response.body // to get complete html markup
6. response.xpath(“xpathhere/text()”].get // give text if to get the value of the cell
7. so use shell command when ever you want to test the code

## Building first spider

1. Open up the environment created for in pycharm
2. Change the parse functions
3. import scrapy  
     
     
   class WorldometersSpider(scrapy.Spider):  
    name = "worldometers"  
    allowed\_domains = ["www.worldometers.info"]  
    start\_urls = ["https://www.worldometers.info/world-population/population-by-country/"]  
     
    def parse(self**,** response):  
    title=response.xpath("//h1/text()").get()  
    countries=response.xpath("//td/a/text()").getall()  
     
    yield {'title':title**,** 'countries':countries}
4. In terminal write tis command scrapy crawl “spider name created”

“scrapy crawl worldometers”

## Getting links listed in the website

import scrapy  
  
  
class WorldometersSpider(scrapy.Spider):  
 name = "worldometers"  
 allowed\_domains = ["www.worldometers.info"]  
 start\_urls = ["https://www.worldometers.info/world-population/population-by-country/"]  
  
 def parse(self**,** response):  
 # title=response.xpath("//h1/text()").get()  
 countries=response.xpath("//td/a")  
  
 for country in countries:  
 country\_name=country.xpath(".//text()").get()  
 link=country.xpath(".//@href").get()  
  
 yield {'country\_name':country\_name**,**'link':link}

## Relative vs absolute link handling

1. import scrapy  
     
     
   class WorldometersSpider(scrapy.Spider):  
    name = "worldometers"  
    allowed\_domains = ["www.worldometers.info"]  
    start\_urls = ["https://www.worldometers.info/world-population/population-by-country/"]  
     
    def parse(self**,** response):  
    # title=response.xpath("//h1/text()").get()  
    countries=response.xpath("//td/a")  
     
    for country in countries:  
    country\_name=country.xpath(".//text()").get()  
    link=country.xpath(".//@href").get()  
     
     
     
    ### 1. way to jugaad way  
    # absolute\_url=f'https://www.worldometers.info/{link}'  
    # yield scrapy.Request(url=absolute\_url)  
     
    # ## 2. other way to make absolute url  
    # absolute\_url=response.urljoin(link)  
    # yield scrapy.Request(url=absolute\_url)  
     
    ## 3. way direct with no bt  
    # yield response.follow(url=link)  
     
    # yield {'country\_name':country\_name,'link':link}

## Scrapping data from multiple value

import scrapy  
import time  
  
class WorldometersSpider(scrapy.Spider):  
 name = "worldometers"  
 allowed\_domains = ["www.worldometers.info"]  
 start\_urls = ["https://www.worldometers.info/world-population/population-by-country/"]  
  
 def parse(self**,** response):  
 # title=response.xpath("//h1/text()").get()  
 countries=response.xpath("//td/a")  
  
 for country in countries:  
 country\_name=country.xpath(".//text()").get()  
 link=country.xpath(".//@href").get()  
  
 # yield {'country\_name':country\_name,'link':link}  
  
  
  
 ### 1. way to jugaad way  
 # absolute\_url=f'https://www.worldometers.info/{link}'  
 # yield scrapy.Request(url=absolute\_url)  
  
 # ## 2. other way to make absolute url  
 # absolute\_url=response.urljoin(link)  
 # yield scrapy.Request(url=absolute\_url)  
  
 ## 3. way direct with no bt  
 yield response.follow(url=link**,** callback=self.parse\_country**,**meta={'country\_name':country})  
  
 time.sleep(**3**)  
  
  
 def parse\_country(self**,**response):  
 rows=response.xpath("(table[contains(@class, 'table')])[1]/tbody/tr")  
 country=response.request.meta['country\_name']  
  
 for row in rows:  
 year=row.xpath('.//td[1]/text()').get()  
 population = row.xpath('.//td[2]/strong/text()').get()  
 yield {  
 'country':country**,** 'year': year**,** 'population':population  
 }

## Project: Extracting single page of audible .com

import scrapy  
  
  
class AudibleSpider(scrapy.Spider):  
 name = "audible" ## name of the spider  
 allowed\_domains = ["www.audible.com"] ## this should be the unique root domain of the website  
 start\_urls = ["https://www.audible.com/search/"] ## this should be the url from which the data is to be extracted  
  
 def parse(self**,** response):  
 product\_container=response.xpath('//div[@class="adbl-impression-container "]/div/span/ul/li')  
 # product\_container = response.xpath('//div[@class="adbl-impression-container "]/li')  
 for product in product\_container:  
 book\_title=product.xpath('.//h3[contains(@class,"bc-heading")]/a/text()').get()  
 book\_author = product.xpath('.//li[contains(@class,"authorLabel")]/span/a/text()').getall()  
 book\_length = product.xpath('.//li[contains(@class,"runtimeLabel")]/span/text()').get()  
  
 yield{  
 'title':book\_title**,** 'author':book\_author**,** 'length':book\_length  
 }

and in terminal write scrapy crawl filename

## Extracting multiple pages in Scrapy by handling pagination

import scrapy  
  
class AudibleSpider(scrapy.Spider):  
 name = "audible" ## name of the spider  
 allowed\_domains = ["www.audible.com"] ## this should be the unique root domain of the website  
 start\_urls = ["https://www.audible.com/search/"] ## this should be the url from which the data is to be extracted  
  
 def parse(self**,** response):  
 product\_container=response.xpath('//div[@class="adbl-impression-container "]/div/span/ul/li')  
 # product\_container = response.xpath('//div[@class="adbl-impression-container "]/li')  
 for product in product\_container:  
 book\_title=product.xpath('.//h3[contains(@class,"bc-heading")]/a/text()').get()  
 book\_author = product.xpath('.//li[contains(@class,"authorLabel")]/span/a/text()').getall()  
 book\_length = product.xpath('.//li[contains(@class,"runtimeLabel")]/span/text()').get()  
  
 yield{  
 'title':book\_title**,** 'author':book\_author**,** 'length':book\_length  
 }  
 ### handling pagination using scrappy  
 pagination=response.xpath('//ul[contains(@class,"pagingElements")]')  
 next\_page\_url=pagination.xpath('.//span[contains(@class,"nextButton")]/a/@href').get()  
  
 if next\_page\_url:  
 yield response.follow(url=next\_page\_url**,**callback=self.parse)

here it is necessary to know that we don’t need additional while loop to evaluate this only using how we have use next\_page\_url can do the trick

## Changing user agent in Scrapy

So when we scrape data using scrapy we have a user agent that gives what agent is used to scrape the data but we have a capability to change it and can decisive the websites.

### Steps to do so:

1. Open anaconda prompt write scrapy shell website name
   1. Scrapy shell <https://www.audible.com/search>
   2. Request.headers
   3. It contains one dictionary by the name: User-agent which contains which agent it has been used .
2. To change the user agent we have few methods:
   1. First get what user agent is used by your browser:
      1. Go to the website
      2. Inspect the elements
      3. You can just type in chrome which user agent is currently being used in browser and it will give it to you (our: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36)
   2. Method-1: to open the pycharm
      1. Open Settings.py file:
      2. See there is one variable name

(user agent variable change it to the user agent that you have)

#USER\_AGENT = "firstscrapy (+http://www.yourdomain.com)"

* + 1. There is another method by changing through the Default request header

#DEFAULT\_REQUEST\_HEADERS = {

# "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8",

# "Accept-Language": "en"}

It can change to

DEFAULT\_REQUEST\_HEADERS = { Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36)

* 1. Method-2 : change in the spider file only
     1. It uses a new function start\_request function

1. import scrapy  
     
     
   class AudibleSpider(scrapy.Spider):  
    name = "audible" ## name of the spider  
    allowed\_domains = ["www.audible.com"] ## this should be the unique root domain of the website  
    # start\_urls = ["https://www.audible.com/search/"] ## this should be the url from which the data is to be extracted  
     
    def start\_requests(self):  
    yield scrapy.Request(url="https://www.audible.com/search/"**,**callback=self.parse**,** headers={"User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"})  
     
     
     
     
    def parse(self**,** response):  
    product\_container=response.xpath('//div[@class="adbl-impression-container "]/div/span/ul/li')  
    # product\_container = response.xpath('//div[@class="adbl-impression-container "]/li')  
    for product in product\_container:  
    book\_title=product.xpath('.//h3[contains(@class,"bc-heading")]/a/text()').get()  
    book\_author = product.xpath('.//li[contains(@class,"authorLabel")]/span/a/text()').getall()  
    book\_length = product.xpath('.//li[contains(@class,"runtimeLabel")]/span/text()').get()  
     
    yield{  
    'title':book\_title**,** 'author':book\_author**,** 'length':book\_length**,** 'agent': response.request.headers['User-Agent'] ## to get which user agent we are using  
    }  
    ### handling pagination using scrappy  
    pagination=response.xpath('//ul[contains(@class,"pagingElements")]')  
    next\_page\_url=pagination.xpath('.//span[contains(@class,"nextButton")]/a/@href').get()  
     
    if next\_page\_url:  
    yield response.follow(url=next\_page\_url**,**callback=self.parse**,**headers={"User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"})

**ONE CRAZY THING ABOUT USER AGENT IS YOU CAN USE ANY USER AGENT TO DO YOUR STUFF:**

**HERE IS THE WEBSITE WHERE ACCORDING TO THE OS YOU CAN SELECT THE USER AGENT (**[**https://explore.whatismybrowser.com/useragents/explore/**](https://explore.whatismybrowser.com/useragents/explore/)**)**

## Building Crawler through Scrapy

Okay till now we were using basic template to get our spider generated but using scrapy there are different template you can use.

1. Command to know what all template you can create
   1. Scrapy genspider -l
   2. There are usually 4 (basic, crawl, csvfeed, xmlfeed) we will be using craw for our use.
   3. Scrapy genspider -t name of the template file name and website name
      1. scrapy genspider -t crawl transcipt subslikescript.com/movies
   4. open the spider in the pycharm
   5. When you open that file a kind of some new functions you will see:
2. import scrapy  
   from scrapy.linkextractors import LinkExtractor  
   from scrapy.spiders import CrawlSpider**,** Rule  
     
     
   class TransciptSpider(CrawlSpider):  
    name = "transcipt"  
    allowed\_domains = ["subslikescript.com"]  
    start\_urls = ["http://subslikescript.com/"]  
     
    rules = (Rule(LinkExtractor(allow=r"Items/")**,** callback="parse\_item"**,** follow=True)**,**) ### this is something new you get options like link extractor which have attributes like allow which basically tells which links to follow, inplace of allow it can also have deny or restrict\_xpath. The restrict xpath tell which xpath to follow to get the result. Then it has callback and follow functons too  
     
    def parse\_item(self**,** response):  
    item = {}  
    #item["domain\_id"] = response.xpath('//input[@id="sid"]/@value').get()  
    #item["name"] = response.xpath('//div[@id="name"]').get()  
    #item["description"] = response.xpath('//div[@id="description"]').get()  
    return item

to extract the different elements

import scrapy  
from scrapy.linkextractors import LinkExtractor  
from scrapy.spiders import CrawlSpider**,** Rule  
  
  
class TranscriptsSpider(CrawlSpider):  
 name = 'transcripts'  
 allowed\_domains = ['subslikescript.com']  
 start\_urls = ['https://subslikescript.com/movies']  
  
 # Setting rules for the crawler  
 rules = (  
 Rule(LinkExtractor(restrict\_xpaths=("//ul[@class='scripts-list']/a"))**,** callback='parse\_item'**,** follow=True)  
 )  
  
 def parse\_item(self**,** response):  
 # Getting the article box that contains the data we want (title, plot, etc)  
 article = response.xpath("//article[@class='main-article']")  
  
 # Extract the data we want and then yield it  
 yield {  
 'title': article.xpath("./h1/text()").get()**,** 'plot': article.xpath("./p/text()").get()**,** 'transcript': article.xpath("./div[@class='full-script']/text()").getall()**,** 'url': response.url**,** }

to add the utf-8 encoding make a change in setting file by typing FEED\_EXPORT\_ENCODING=’utf-8’

now run the file scrapy craw transcript

## Handling Pagination using scrapy

1. It is to understand that in the Rules tuple the order actually matters if in first rule you defined callback and follow attribute then in the subsequent rule it is fine not to define it again. For ex (this include pagination too )

import scrapy  
from scrapy.linkextractors import LinkExtractor  
from scrapy.spiders import CrawlSpider**,** Rule  
  
  
class TransciptSpider(CrawlSpider):  
 name = "transcipt"  
 allowed\_domains = ["subslikescript.com"]  
 start\_urls = ["https://subslikescript.com/movies\_letter-J"]  
  
 rules = (  
 Rule(LinkExtractor(restrict\_xpaths=("//ul[@class='scripts-list']/a"))**,** callback="parse\_item"**,** follow=True)**,** Rule(LinkExtractor(restrict\_xpaths=("(//a[@rel='next'])[1]"))) ### this is for pagination  
 )  
  
 def parse\_item(self**,** response):  
 # Getting the article box that contains the data we want (title, plot, etc)  
 article = response.xpath("//article[@class='main-article']")  
  
 # Extract the data we want and then yield it  
 yield {  
 'title': article.xpath("./h1/text()").get()**,** 'plot': article.xpath("./p/text()").get()**,** 'transcript': article.xpath("./div[@class='full-script']/text()").getall()**,** 'url': response.url**,** }

## Crawler User Agent

Other steps mentioned in the above sections with same heading remains same. The only thing which we will discuss here is how to change the user agent in the code itself

import scrapy  
from scrapy.linkextractors import LinkExtractor  
from scrapy.spiders import CrawlSpider**,** Rule  
  
  
class TransciptSpider(CrawlSpider):  
 name = "transcipt"  
 allowed\_domains = ["subslikescript.com"]  
 # start\_urls = ["https://subslikescript.com/movies\_letter-J"]  
 user\_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"  
  
 def start\_requests(self):  
 yield scrapy.Request(url='https://subslikescript.com/movies\_letter-J'**,** headers={'user-agent':self.user\_agent})  
  
 rules = (  
 Rule(LinkExtractor(restrict\_xpaths=("//ul[@class='scripts-list']/a"))**,** callback="parse\_item"**,** follow=True**,**process\_request='set\_user\_agent')**,** Rule(LinkExtractor(restrict\_xpaths=("(//a[@rel='next'])[1]"))**,**process\_request='set\_user\_agent') ### this is for pagination and also to change the user agent  
 )  
  
 def set\_user\_agent(self**,**request**,**spider):  
 request.headers['user-Agent']=self.user\_agent  
 return request  
  
 def parse\_item(self**,** response):  
 # Getting the article box that contains the data we want (title, plot, etc)  
 article = response.xpath("//article[@class='main-article']")  
  
 # Extract the data we want and then yield it  
 yield {  
 'title': article.xpath("./h1/text()").get()**,** 'plot': article.xpath("./p/text()").get()**,** # 'transcript': article.xpath("./div[@class='full-script']/text()").getall(),  
 'url': response.url**,** 'user\_agent':response.request.headers['user-Agent']  
 }

# Exporting data to a database like MongoDB or SQLite

1. First we need to understand what is in the pipeline.py file in the folder that we have created

# Define your item pipelines here  
#  
# Don't forget to add your pipeline to the ITEM\_PIPELINES setting  
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
  
  
# useful for handling different item types with a single interface  
from itemadapter import ItemAdapter  
  
  
class FirstscrapyPipeline:  
 def process\_item(self**,** item**,** spider):  
 return item

1. Here we can see we have one function in the pipeline class named as process\_item, but it has also two other functions which are open\_spider(self,spider) and close\_spider(self, spider)
2. Both the functions can be used to give certain messages.

# Define your item pipelines here  
#  
# Don't forget to add your pipeline to the ITEM\_PIPELINES setting  
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
  
  
# useful for handling different item types with a single interface  
from itemadapter import ItemAdapter  
  
import logging  
class FirstscrapyPipeline:  
 def open\_spider(self,spider):  
 logging.warning('Spider-open-Pipeline')  
 def close\_spider(self,spider):  
 logging.warning('Spider closed-Pipeline')  
 def process\_item(self**,** item**,** spider):  
 return item

1. Remember that you can also add some other class to this pipeline.py file but it is important to add similar class name in the dictionary under the “# Configure item pipelines” comment and in the ITEM\_PIPELINE dictionary.

ITEM\_PIPELINES = {  
 "firstscrapy.pipelines.FirstscrapyPipeline": **300,**}

Here you can see there is one number given to our function of a class this number denotes the which class should class first like in the above case if we add a new class by a name new\_item (syntax to add so: folderneme.piepline.classname)and give it 200 as a number then it will execute first.

1. You can simply run the spider again to see how the message of the open\_spider and close spider comes.

## MongoDB

1. First create a free account on MongoDB

<https://www.mongodb.com/products/platform/cloud#deviceSync>

1. Then install a library in conda

One thing install the libraries where your folder is:

* 1. First library is: conda install pymongo dnspyhton -y

1. Create a class which gonna call Mogodb for inserting the elements in the mongodb database

class MongodbPipeline:  
 collection\_name="movie\_transcript"  
 def open\_spider(self,spider):  
 logging.warning('Spider-open-Pipeline')  
 self.client=pymongo.MongoClient("")  
 self.db = self.client['Movies\_transcript\_database']  
 def close\_spider(self,spider):  
 logging.warning('Spider closed-Pipeline')  
 self.client.close()  
 def process\_item(self**,** item**,** spider):  
 self.db[self.collection\_name].insert(item)  
 return item

1. Update the class name in settings.py file.
2. Go to Mongodb website after your cluster is created and then follow following steps;
   1. Click on database access and add “Add New Database user.
   2. Give name and pass word
   3. Please check whether in the dropdown select “read and write any database” under built in role.
   4. Click on add user
   5. Now click on network access
      1. Set the ip address for the purpose of learning we are seting generic IP address (0.0.0.0/0)
   6. Now go to database 🡪cluster sandbox🡪connect🡪 set up driver🡪 setup connection by changing language to python.
   7. Select the string coming below by the name mongo ….. and paste it in the Mongoclient.

# Define your item pipelines here  
#  
# Don't forget to add your pipeline to the ITEM\_PIPELINES setting  
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
  
  
# useful for handling different item types with a single interface  
from itemadapter import ItemAdapter  
  
import logging  
import pymongo  
# class FirstscrapyPipeline:  
# def open\_spider(self):  
# logging.warning('Spider-open-Pipeline')  
# def close\_spider(self):  
# logging.warning('Spider closed-Pipeline')  
# def process\_item(self, item, spider):  
# return item  
  
class MongodbPipeline:  
 collection\_name="movie\_transcript"  
 def open\_spider(self,spider):  
 logging.warning('Spider-open-Pipeline')  
 self.client=pymongo.MongoClient("mongodb+srv://puru:puru@cluster0.idbtzs6.mongodb.net/?retryWrites=true&w=majority") ### remember here to change the password which you will get as <password> in the above string  
 self.db = self.client['Movies\_transcript\_database']  
 def close\_spider(self,spider):  
 logging.warning('Spider closed-Pipeline')  
 self.client.close()  
 def process\_item(self**,** item**,** spider):  
 self.db[self.collection\_name].insert(item)  
 return item

* 1. To see how your data is stored click on “collections” on mongodb site databse tab and can see your data.

## SQLite

1. For sqlite we don’t have to install any library but we have library already.

# Define your item pipelines here  
#  
# Don't forget to add your pipeline to the ITEM\_PIPELINES setting  
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html  
  
  
# useful for handling different item types with a single interface  
from itemadapter import ItemAdapter  
  
import logging  
import pymongo  
import sqlite3  
# class FirstscrapyPipeline:  
# def open\_spider(self):  
# logging.warning('Spider-open-Pipeline')  
# def close\_spider(self):  
# logging.warning('Spider closed-Pipeline')  
# def process\_item(self, item, spider):  
# return item  
  
class MongodbPipeline:  
 collection\_name="movie\_transcript"  
 def open\_spider(self**,**spider):  
 self.client=pymongo.MongoClient("mongodb+srv://puru:puru@cluster0.idbtzs6.mongodb.net/?retryWrites=true&w=majority")  
 self.db = self.client['Movies\_transcript\_database']  
 def close\_spider(self**,**spider):  
 self.client.close()  
 def process\_item(self**,** item**,** spider):  
 self.db[self.collection\_name].insert(item)  
 return item  
  
class SQLPipeline:  
 def open\_spider(self**,**spider):  
 self.connection=sqlite3.connect("transcript\_movies.db")  
 self.c=self.connection.cursor()  
 ## query  
 try:  
 self.c.execute("""  
 CREATE TABLE transcripts(  
 title TEXT,  
 plot TEXT,  
 transcript TEXT,  
 url TEXT)  
 """)  
 self.connection.commit()  
 except sqlite3.OperationalError:  
 pass  
  
 def close\_spider(self**,**spider):  
 self.connection.close()  
 def process\_item(self**,** item**,** spider):  
 self.c.execute("""  
 Insert INTO transcripts VALUES (?,?,?,?)  
 """**,**(  
 item.get('title')**,**item.get('plot')**,**item.get('transcript')**,**item.get('url')  
 ))  
 self.connection.commit()  
 return item

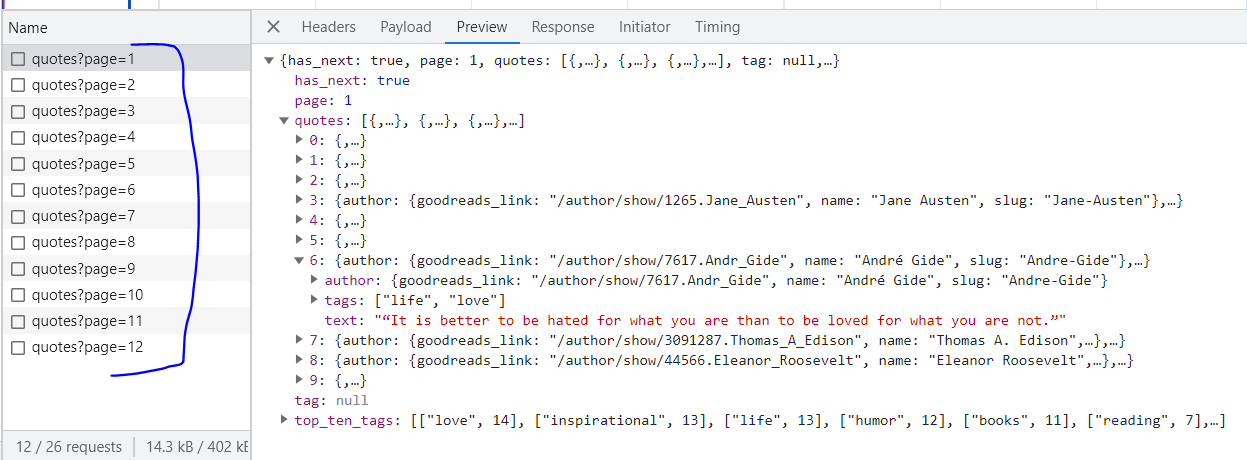
1. You can see we have sqllite as a function few notable changes here are we use connections instead of clients and others are self explanatory.
2. Just remember in sql the content can stored in the form of strings so change the spider code when extracting data from web as shown below:
3. import scrapy  
   from scrapy.linkextractors import LinkExtractor  
   from scrapy.spiders import CrawlSpider**,** Rule  
     
     
   class TransciptSpider(CrawlSpider):  
    name = "transcipt"  
    allowed\_domains = ["subslikescript.com"]  
    # start\_urls = ["https://subslikescript.com/movies\_letter-J"]  
    user\_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"  
     
    def start\_requests(self):  
    yield scrapy.Request(url='https://subslikescript.com/movies\_letter-J'**,** headers={'user-agent':self.user\_agent})  
     
    rules = (  
    Rule(LinkExtractor(restrict\_xpaths=("//ul[@class='scripts-list']/a"))**,** callback="parse\_item"**,** follow=True**,**process\_request='set\_user\_agent')**,** Rule(LinkExtractor(restrict\_xpaths=("(//a[@rel='next'])[1]"))**,**process\_request='set\_user\_agent') ### this is for pagination and also to change the user agent  
    )  
     
    def set\_user\_agent(self**,**request**,**spider):  
    request.headers['user-Agent']=self.user\_agent  
    return request  
     
    def parse\_item(self**,** response):  
    # Getting the article box that contains the data we want (title, plot, etc)  
    article = response.xpath("//article[@class='main-article']")  
    ## below step when trying sqlite  
    transcript\_list=article.xpath("./div[@class='full-script']/text()").getall()  
    transcript\_string=" ".join(transcript\_list)  
     
    # Extract the data we want and then yield it  
    yield {  
    'title': article.xpath("./h1/text()").get()**,** 'plot': article.xpath("./p/text()").get()**,** 'transcript':transcript\_string**,** 'url': response.url**,** # 'user\_agent':response.request.headers['user-Agent']  
    }
4. to view the content of the database created go to <https://inloop.github.io/sqlite-viewer/> and drop the file and one can see the complete data

# How to scrape website API via scrapy

## Handling single page website

1. Here we will try to extract the data via API of a website and we will can see such information in the network tab of inspect elements.
2. Let’s first create a new project for scrapy
3. Once created go to the website and inspect elements and then go to network section in that section select fetch/XHR tab and see the request come🡪 then click on the request to see the request url and all.
4. import scrapy  
   import json  
     
     
   class QuotesSiteSpider(scrapy.Spider):  
    name = "quotes\_site"  
    allowed\_domains = ["quotes.toscrape.com"]  
    start\_urls = ["https://quotes.toscrape.com/api/quotes?page=1"]  
     
    def parse(self**,** response):  
    json\_response=json.loads(response.body) ### by giving .body as a command it will give the complete response of the website with every tag and its value sone can simply just use the print(response.body) command to see the results  
    quotes=json\_response.get('quotes')  
    for quote in quotes:  
    yield{  
    'author':quote.get('author').get('name')**,** 'tags': quote.get('tags')**,** 'quote': quote.get('text')  
     
    }

## Handling multiple page website

In our website is <https://quotes.toscrape.com/scroll> is kind of infinte scrolling type but has limit of 10 pages. 

In the above image you can see when scrolling down the website we got different pages

So in order to extract the next page, you can also see one elemet we have is also known by has\_next in the above image we are going to use that

import scrapy  
import json  
  
  
class QuotesSiteSpider(scrapy.Spider):  
 name = "quotes\_site"  
 allowed\_domains = ["quotes.toscrape.com"]  
 start\_urls = ["https://quotes.toscrape.com/api/quotes?page=1"]  
  
 def parse(self**,** response):  
 json\_response=json.loads(response.body) ### by giving .body as a command it will give the complete response of the website with every tag and its value sone can simply just use the print(response.body) command to see the results  
 quotes=json\_response.get('quotes')  
 for quote in quotes:  
  
 yield{  
 'author':quote.get('author').get('name')**,** 'tags': quote.get('tags')**,** 'quote': quote.get('text')  
  
 }  
 has\_next=json\_response.get('has\_next')  
 if has\_next:  
 next\_page\_number=json\_response.get('page')+**1** yield scrapy.Request(  
 url=f'https://quotes.toscrape.com/api/quotes?page={next\_page\_number}'**,** callback=self.parse  
  
 )

## Login to a website using scrapy

[**Reminder**: this is for educational purpose and should not be used as this may lead to your account band and criminal prosecutions]

Basics to know before going on:

1. <https://quotes.toscrape.com/login>
2. Go to the network tab and select all in the first ribbon
3. When you open above website and click on preserve log In inspect element.
4. Reload the website and then you can see the different requests.
5. Ideally it should let you log in and in when you see one of the request as log in and in that you can see different options like URL, request method and request code,etc.
6. So there is some thing called csrf token and this keeps on changing whenever the website refresh is done.
7. Now for this we will create a new spider in the same folder.
   1. Write scrapy genspider quotes\_login [quotes.toscrape.com/login](https://quotes.toscrape.com/login) and then open in pycharm and see the following code

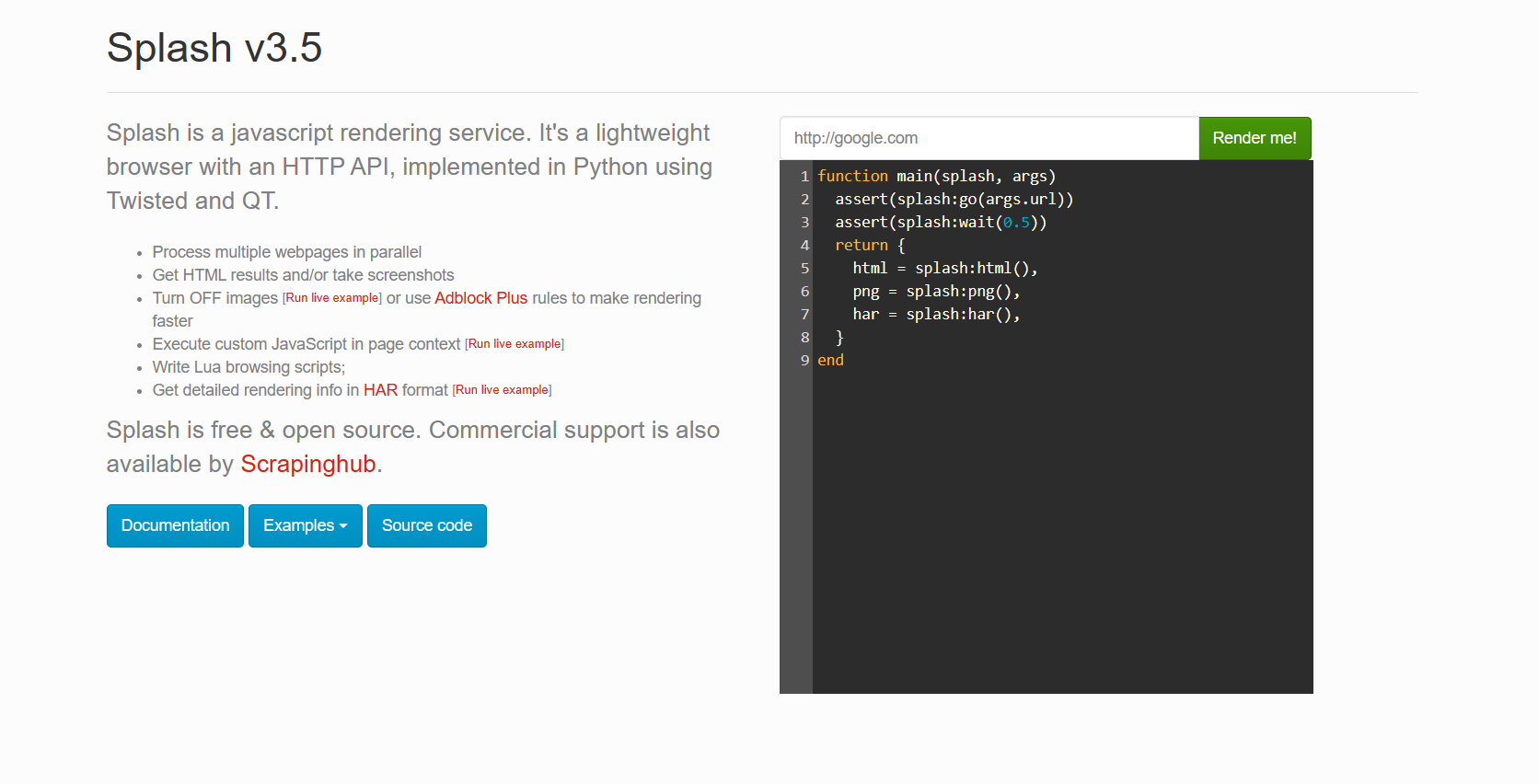
import scrapy  
from scrapy import FormRequest  
  
class QuotesLoginSpider(scrapy.Spider):  
 name = "quotes\_login"  
 allowed\_domains = ["quotes.toscrape.com"]  
 start\_urls = ["https://quotes.toscrape.com/login"]  
  
 def parse(self**,** response):  
 csrf\_token=response.xpath("//input[@name='csrf\_token']/@value").get()  
 yield FormRequest.from\_response(response**,** formxpath='//form'**,** formdata={  
 'csrf\_token':csrf\_token**,** 'username':'admin'**,** 'password':'admin'  
 }**,** callback=self.after\_login  
  
 )  
  
 def after\_login(self**,**response):  
 if response.xpath("//a[@href='/logout']/text()").get():  
 print("logged in finally") ### we will see if logout button prompt up that means we are successfully logged in

# Splash Basics:

In this section, we will learn Splash. Scrapy alone doesn't let us scrape JavaScript-driven websites, so we have to add Splash to it. We will keep scraping websites using Python/Scrapy, but, in addition to that, we will build a small script on Splash to get the HTML code behind the JavaScript website we wish to scrape.

Initial steps before working on this:

1. Install Docker from here: <https://www.docker.com/products/docker-desktop/>
2. Setup the docker and let its engine start
3. To get splash
4. Open up a cmd prompt and write this command docker pull scrapinghub/splash
5. This will download the set of images which is basically a set of instructions for setting up of docker container.
6. Then write second command : docker run -it -p 8050:8050 scrapinghub/splash
7. The above command is used to run splash on browser just copy the http mentioned after server listening on and paste that on browser.
8. There is an another method to get the splash run on browser just by loading up the docker and clicking on the link of the port from docker container.
9. You can start stop and do other function using docker.

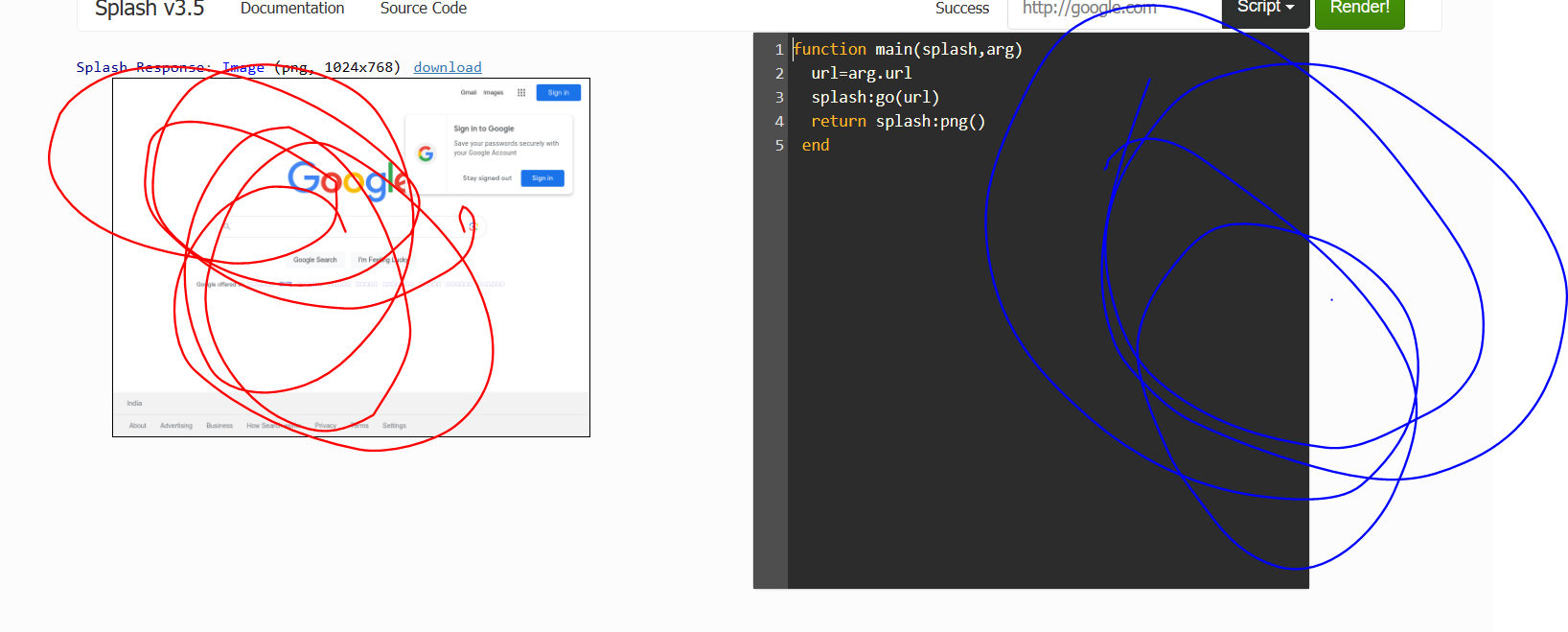


Browser will show this and in this on the right side playground and it uses a new programming language especially for splash called **Lua**

### Some basics of Lua

1. Function main(splash,arg) ### common way to define a function
   1. url=arg.url
   2. splash:go(url)
   3. return splash:png() ## this will get the screenshot of the website that is mentioned in the white box beside render me box . other command can be html, wait etc.

end ### mandatory to add



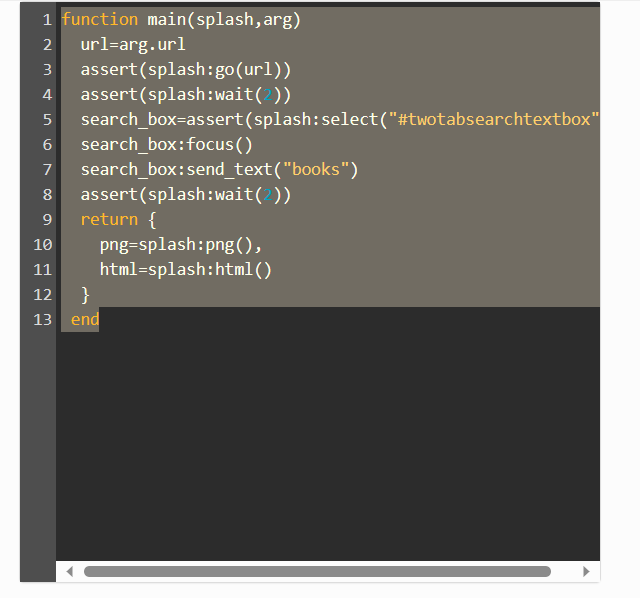
Blue is for playground and red is the output

1. function main(splash,arg)
   1. url=arg.url
   2. assert(splash:go(url)) ## this will help if the website breaks or not available
   3. assert(splash:wait(2)) ## wait is similar to time in python
   4. return {
      1. png=splash:png(),
      2. html=splash:html()

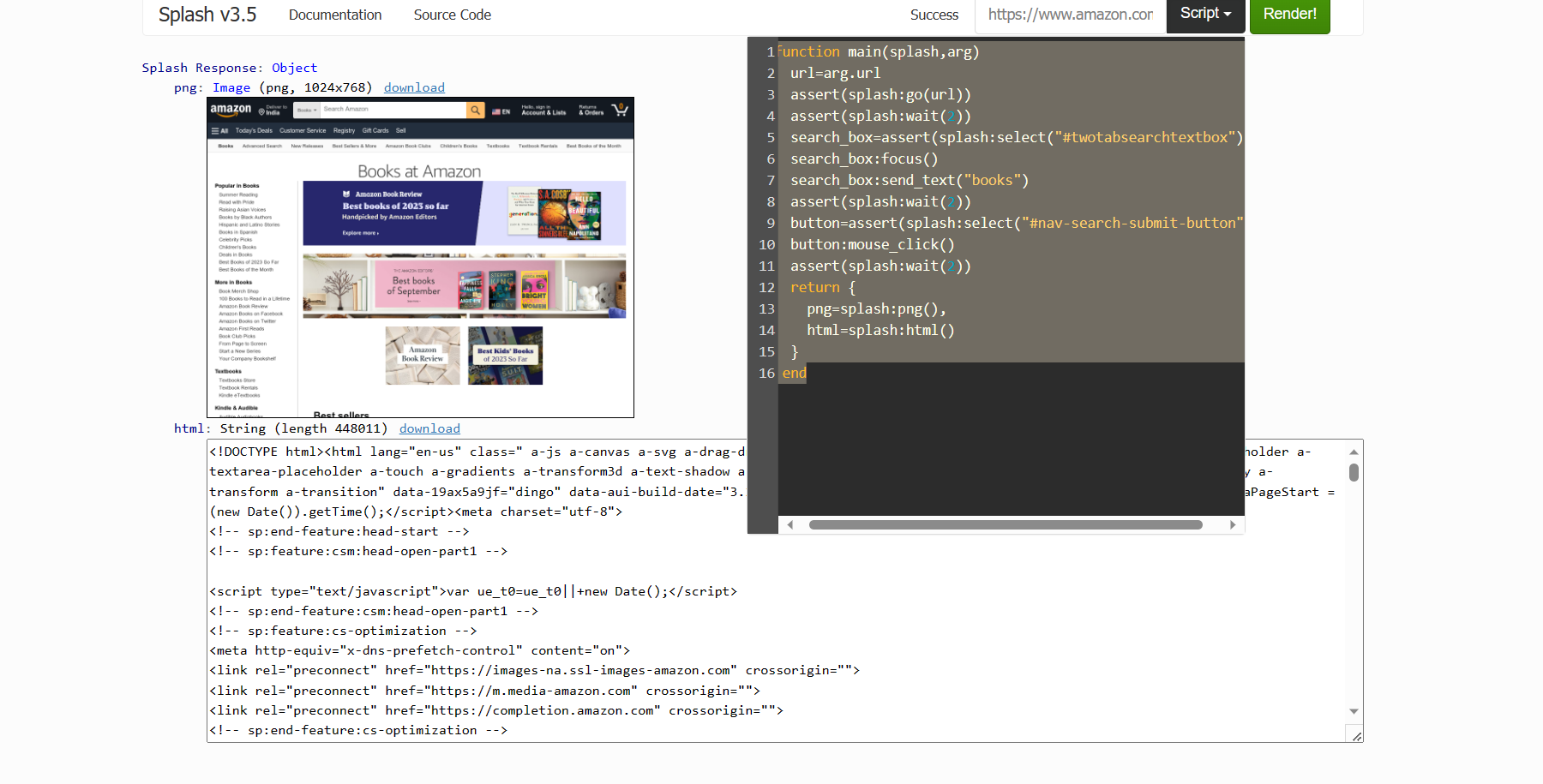
} # by giving two values we said to the console to give two outputs

End

## How to find elements using splash



In the above code it is important to understand line 5 in the search box we have use command splash:select(“#.....”) so select is just like find\_element\_by and we do have select\_all which is returning list. In splash it take css id indicators rather than xpath or class and it started with ‘#’ the other text is the id we found in that element of the inspect element.

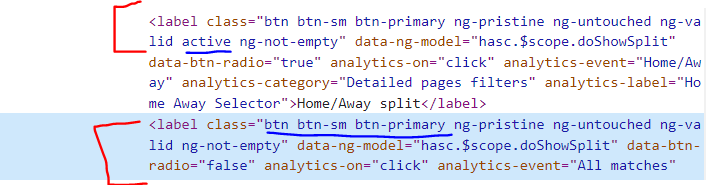


Above Code to click on a button

## Project: Scraping Javascript website using splash



1. in the above code you can see we have use select\_all, see we don’t have to use the # for it
2. Since it uses css id selector so it is required to find the css id but when inspecting elements it can be difficult to get so 🡪 so commonpractice is in generally class attribute the name if the tag and class combining is used



In the above inspect of the elemet we see two lables which we have also captured in the splash code. So we see the two label tag has almost exact same class but one is different to other in terms of active word so we choose first 3 words to make our css id and in splash we have to remove the white spaces with the dot(.) as mentioned.

And also allmatches[2] variable will catch the second label.

1. splash.private\_mode\_disabled=false by adding this line it become similar to google incognito mode and by putting false we get out of it and it is advised to do so because some website may not render correctly.

### Using Scrapy with splash

1. Create a project for scrapy
2. Then install a library in pycharm terminal for scrapy -splash (<https://github.com/scrapy-plugins/scrapy-splash>)
   1. pip install scrapy-splash
   2. in the above github page you will find more add on to the settings file just add that

SPLASH\_URL = 'http://localhost:8050/'  
DOWNLOADER\_MIDDLEWARES = {  
 'scrapy\_splash.SplashCookiesMiddleware': **723,** 'scrapy\_splash.SplashMiddleware': **725,** 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': **810,**}  
SPIDER\_MIDDLEWARES = {  
 'scrapy\_splash.SplashDeduplicateArgsMiddleware': **100,**}

* 1. in the splas\_url: just put your localhost port

1. code in scrapy to get the results as we have achieved in splash

import scrapy  
from scrapy\_splash import SplashRequest  
  
class AdamcoiSpider(scrapy.Spider):  
 name = "adamcoi"  
 allowed\_domains = ["www.adamchoi.co.uk"]  
 # start\_urls = ["http://www.adamchoi.co.uk/"]  
 script="""   
 function main(splash,args)  
 splash.private\_mode\_disabled=false  
 assert(splash:go(args.url))  
 assert(splash:wait(5))  
 all\_matches=assert(splash:select\_all("label.btn.btn-sm.btn-primary"))  
 all\_matches[2]:mouse\_click()  
 assert(splash:wait(5))  
 splash:set\_viewport\_full()  
 return {  
 splash:png(),  
 splash:html()  
 }  
 end   
 """  
  
 def start\_requests(self):  
 yield SplashRequest(url="https://www.adamchoi.co.uk/overs/detailed"**,**callback=self.parse**,** endpoint='execute'**,**args={'lua\_source':self.script})  
 def parse(self**,** response):  
 print(response.body)

1. a modular code to extract data from adamcoi website
2. import scrapy  
   from scrapy\_splash import SplashRequest  
     
   class AdamcoiSpider(scrapy.Spider):  
    name = "adamcoi"  
    allowed\_domains = ["www.adamchoi.co.uk"]  
    # start\_urls = ["http://www.adamchoi.co.uk/"]  
    script="""   
    function main(splash,args)  
    splash.private\_mode\_disabled=false  
    assert(splash:go(args.url))  
    assert(splash:wait(5))  
    all\_matches=assert(splash:select\_all("label.btn.btn-sm.btn-primary"))  
    all\_matches[2]:mouse\_click()  
    assert(splash:wait(5))  
    splash:set\_viewport\_full()  
    return {  
    splash:png(),  
    splash:html()  
    }  
    end   
    """  
     
    def start\_requests(self):  
    yield SplashRequest(url="https://www.adamchoi.co.uk/overs/detailed"**,**callback=self.parse**,** endpoint='execute'**,**args={'lua\_source':self.script})  
    def parse(self**,** response):  
    rows=response.xpath('//tr')  
    for row in rows:  
    date=row.xpath('./td[1]/text()').get()  
    home\_team = row.xpath('./td[2]/text()').get()  
    score = row.xpath('./td[3]/text()').get()  
    away\_team= row.xpath('./td[4]/text()').get()  
    yield{  
    'date':date**,** 'home\_team': home\_team**,** 'score': score**,** 'away\_team': away\_team**,** }

NOTE: multiline comments in splash is given by –[[all text--]]

### Changing the user agent

