

Welcome to the world of Data Scraping (Web Scraping):

Web scraping also known as web harvesting or web data extraction, is the process of automatically gathering or mining information and content from websites. To allow for future manipulation and analysis, this data is typically saved in a local file. Applications for web scraping include data analysis, market research, price comparison, content aggregation, and more.

It involves utilizing specialized tools and software to extract required data from WebPages which can then be stored in a central local database or spreadsheet for later retrieval or analysis as mentioned above. Utilizing a variety of tools and libraries, including requests, BeautifulSoup, Scrapy, Selenium, ParseHub, etc., one can perform web scraping. Depending on the size and complexity of the web scraping project, every tool offers its own advantages and downsides.

The following steps are involved in web scraping:

Step 1: Find the URLs you wish to scrape.

The very first step of web scraping is typically to identify the website or web page from which you want to extract data. This could be a single page or multiple pages, depending on the scope of your project. Once you have identified the page(s), you will need to inspect the HTML code of the page(s) to identify the specific elements or data points you want to scrape. This could include text, images, links, tables, and other elements that are relevant to your needs. After identifying the relevant elements, you can use a web scraping tool or write your own code to extract the data from the page(s) and save it in a usable format such as a CSV or JSON file. In short, the first step of web scraping is to identify the target page(s) and analyze their structure to determine the best approach for extracting the relevant data.

Web scraping can also be done manually, using a bot or web crawler. Web scraping is used for a variety of purposes, including contact scraping, web indexing, data mining, online price monitoring, and product review scraping. However, it is important to note that the legality of web scraping varies across different countries and jurisdictions. In some cases, web scraping may be considered legal if the data being scraped is publicly available and there are no laws or terms of service agreements that prohibit the scraping. However, scraping certain types of data, such as copyrighted material or personal information that is protected by privacy laws, may be illegal. Hence, it is imperative for businesses and individuals engaged in web scraping to be aware of legal issues involved and to ensure they are complying with applicable laws and regulations. It may also be a good practice to seek legal advice before engaging in any web scraping activities.

Here we will work with the following URL:

<https://parsehub.com/sandbox/showtimes>

Step 2: Inspect the webpage to determine what information you wish to extract.

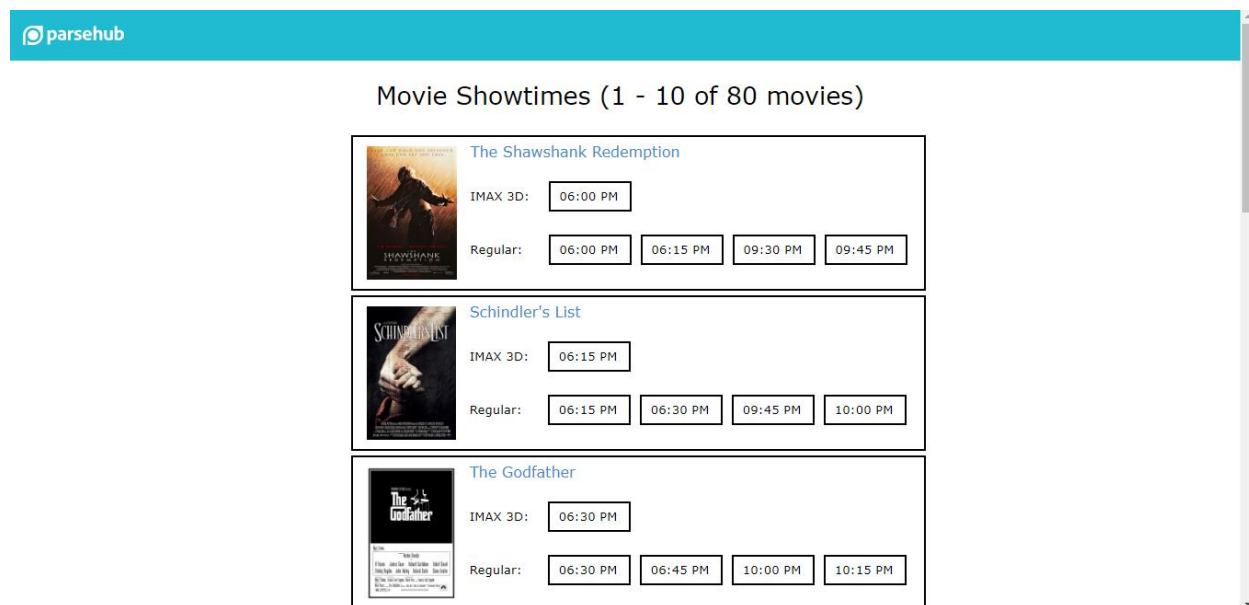
The HTML code of the page can be examined using the developer tools built into your web browser to pinpoint the precise areas you wish to scrape. The majority of today's browsers, including Chrome and Firefox, feature powerful developer tools that let you inspect and edit a page's source code in real-time. After launching the developer tools, you can navigate the page's HTML structure using the "Elements" tab and run JavaScript code and view the results

using the "Console" tab. You may find the precise DOM components that contain the data you want to scrape by navigating the website in this way and looking at the HTML code.

A different approach to automate the process of obtaining data from a webpage is by using a web scraping application like BeautifulSoup or Scrapy. With the help of these tools, you may create a code that searches through a page's HTML structure and retrieves particular data without having to manually recognize each element. These tools allow you to directly store the scraped data into a useable format, like a CSV or JSON file.

Regardless of the approach you take, it's imperative to be familiar with HTML and CSS so you can understand the structure and layout of a webpage, and how to identify specific elements within it. Additionally, as mentioned above, it's important to be aware of any legal or ethical considerations when scraping data from websites, and to obtain the necessary permissions or follow best practices to avoid any potential problems.

The webpage should look like this when you open the website

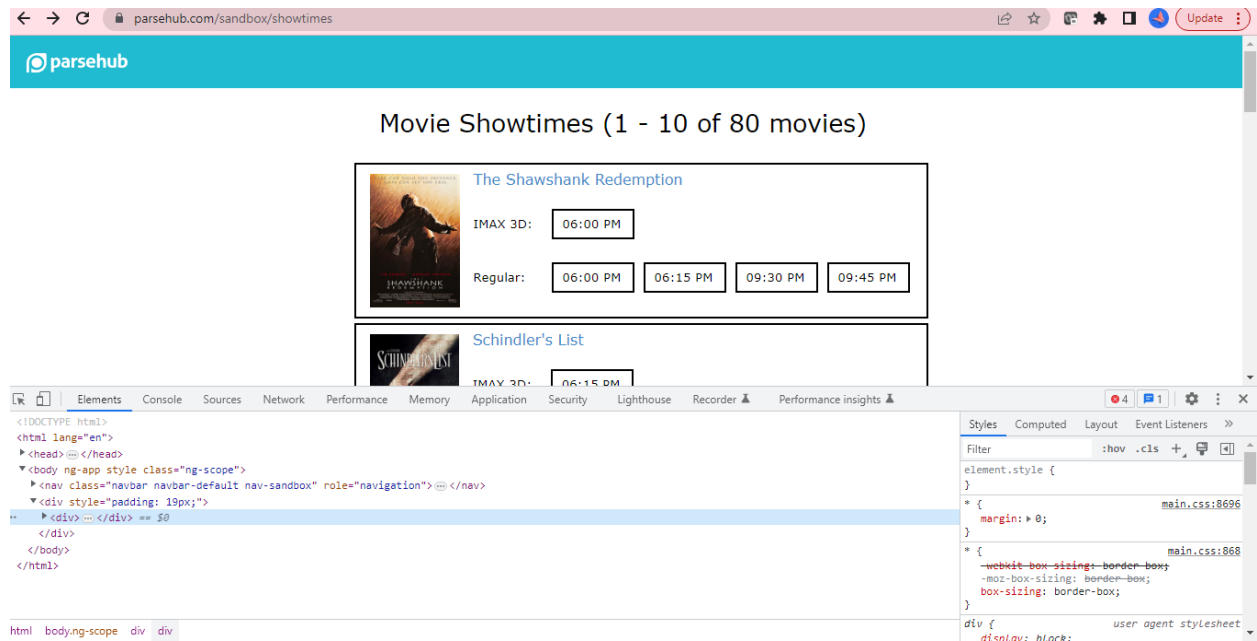


There is a list of movies on the webpage. Each entry on the page has a movie title, a link to a page with information about the movie, an image, and a Showtime. Only the movie title, link, and movie time will be scraped in this specific case.

We can analyse this web page using developer tools. If you use Chrome, click the three dots in the top-right corner of the page (**see the screenshot below**), then choose **More Tools**. Click Developer Tools (**More Tools > Developer Tools**) after that.

Next, click on the **Elements** tab in the Developer Tools page.

You'll note that every element has an ID or class name when you browse through the HTML on the developer tools page. Some class names may include **image** and **title**. You'll need to specify these class names later in your Python code, so you can identify and make a note of them now.



Step 3: Make the coding environment ready

To set up your coding environment for web scraping with Python, you can follow these steps:

1. **Install Python:** Download and install the latest version of Python from the official website (<https://www.python.org/downloads/>).
2. **Install a code editor or IDE:** Choose a code editor or IDE (Integrated Development Environment) to write and run your Python code. Some popular choices include Visual Studio Code, PyCharm, Sublime Text, and Atom.
3. **Install the required packages:** Two common libraries for web scraping with Python are **BeautifulSoup** and **Requests**. We will also use pandas to store our data in a CSV file. Install these packages using pip, the package installer for Python. Open a terminal or command prompt and run the following command:

pip install requests pandas beautifulsoup4

4. **Start coding:** After the installation is complete, you are ready to start coding. Open your code editor or IDE, create a new Python file, and start writing your web scraping code using the **requests** and **BeautifulSoup** libraries.

In this project we will work with Jupyter Notebook. To install **requests**, **pandas** and **beautifulsoup4** in Jupyter Notebook, you can follow these steps:

1. **Open Jupyter Notebook:** Launch Jupyter Notebook by opening a terminal or command prompt, navigating to the directory where you want to create your notebook, and then running the command **jupyter notebook**.
2. **Create a new Python notebook:** Click on the "New" button in the top right corner of the Jupyter Notebook web interface and select "Python 3".

3. **Install the required packages:** In a new code cell, enter the following code to install the required packages:
!pip install requests pandas beautifulsoup4

This will install the requests, pandas, and beautifulsoup4 libraries in your Jupyter Notebook environment.

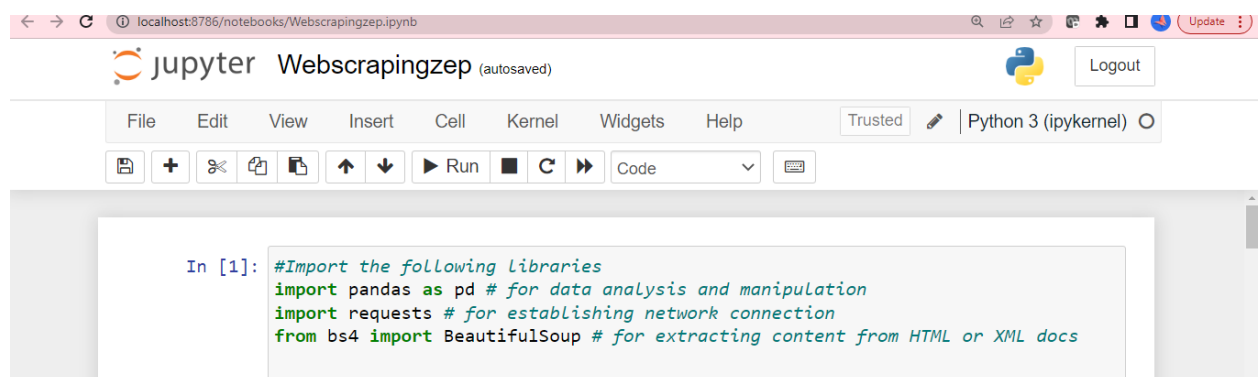
4. **Verify installation:** You can verify that the libraries have successfully installed by importing them in a new code cell. For example, run the following code to import requests and get the HTML content of a webpage:

```
import requests
response = requests.get('https://www.abcxyz.com/')
print(response.content)
```

If you see the HTML content of abcxyz's homepage printed out, then `requests` has been successfully installed and imported. Similarly, you can import `pandas` and `beautifulsoup4` in new code cells to verify that they have been successfully installed. Note that when installing packages in a Jupyter Notebook, it's important to include the exclamation mark! at the beginning of the `pip install` command to run it as a system command instead of a Python command

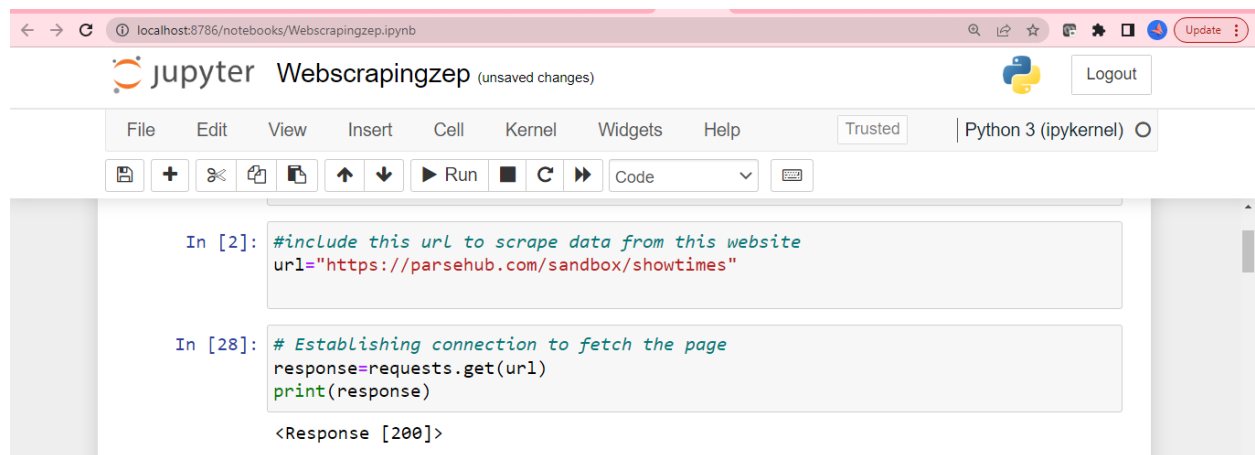
Note: *As mentioned earlier, while scraping data, it's important to always respect website owner's robots.txt files and not bring down the website by making too many requests. Additionally, it's important to always check the legality of using data obtained from web scraping. In general, it's best to scrape data only if it's allowed by the website owner and if the Terms of Service allow for it.*

Step 4: The beautifulsoup4, requests, and pandas libraries must be imported next



Step 5: Put in a URL and run a fetch request. Here we will work with the following URL:

<https://parsehub.com/sandbox/showtimes>



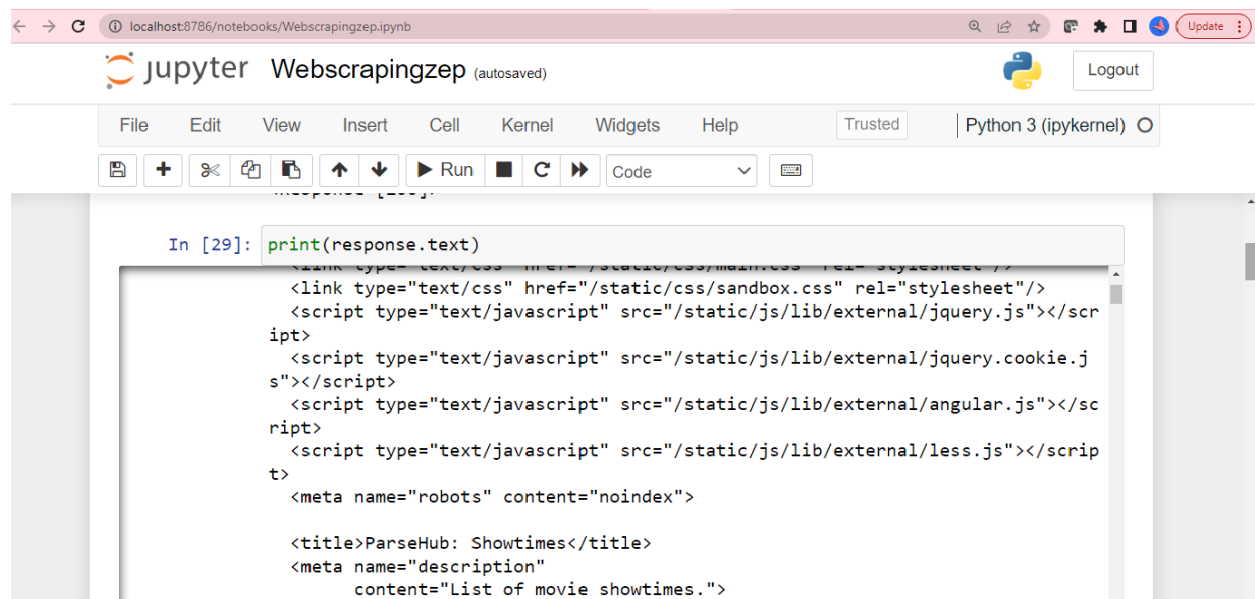
```
In [2]: #include this url to scrape data from this website
url="https://parsehub.com/sandbox/showtimes"

In [28]: # Establishing connection to fetch the page
response=requests.get(url)
print(response)

<Response [200]>
```

In web scraping, we often use the `get ()` method of the `requests` library in Python to retrieve the HTML content of a webpage. This is because the `get ()` method sends an HTTP GET request to the specified URL and returns the response content as a string. A 200 response code indicates that the login was successful and that you can now access the content that requires authentication on the website. Just execute the `print (response)` statement once. Since the application is working properly, comment out the print statement by adding '#' before it so that next time it should not get executed

To view the content of the requested page in web scraping using Python's `requests` library, you can use the `.text` attribute of the response object returned by the `requests.get()` function. The content will be printed to the console .You should see the following output



```
In [29]: #print(response.text)

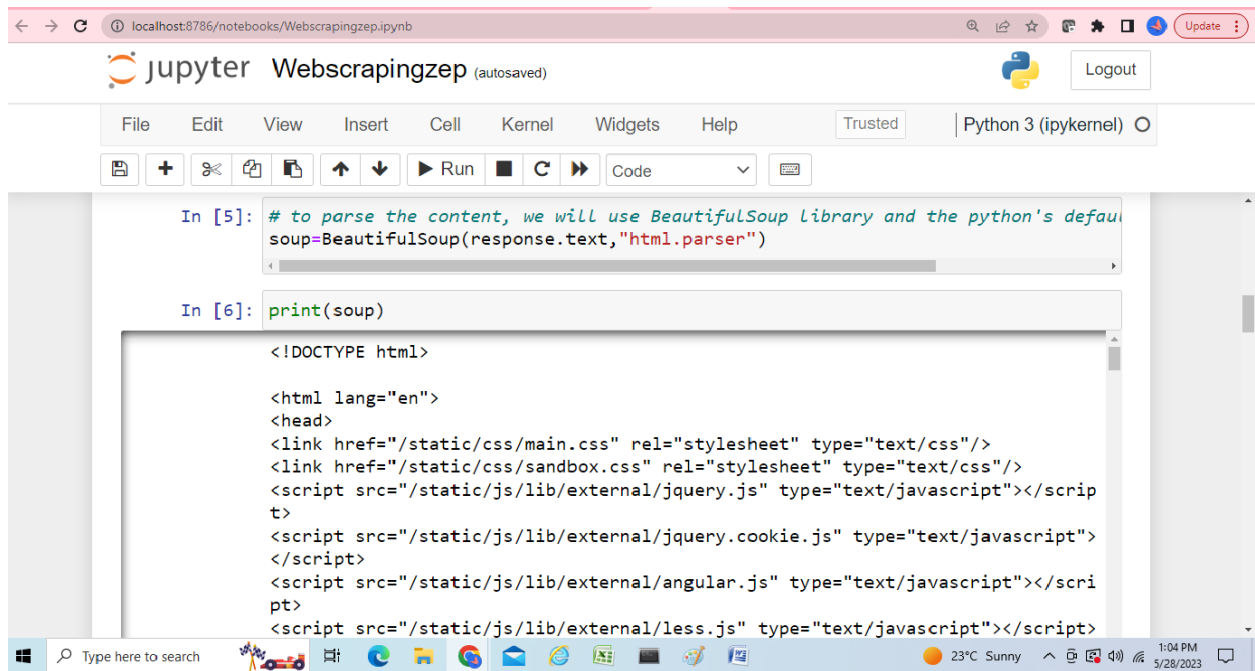
<link type="text/css" href="/static/css/sandbox.css" rel="stylesheet"/>
<script type="text/javascript" src="/static/js/lib/external/jquery.js"></scr
ipt>
<script type="text/javascript" src="/static/js/lib/external/jquery.cookie.j
s"></script>
<script type="text/javascript" src="/static/js/lib/external/angular.js"></sc
ript>
<script type="text/javascript" src="/static/js/lib/external/less.js"></scrip
t>
<meta name="robots" content="noindex">

<title>ParseHub: Showtimes</title>
<meta name="description"
content="List of movie showtimes.">
```

Once the code runs successfully, you may include `#` before the `print (response.text)` command to stop it from running in the future. `#` is the symbol for commenting in Python. The comments are not executed by the Python compiler.

Step 6: Scrape required data from the HTML file

To scrape data from an HTML file using Python, you can use the BeautifulSoup library. We use BeautifulSoup in web scraping because it is a Python library that allows us to parse HTML and XML documents, which is useful for extracting data from websites. With BeautifulSoup, you can search for elements within an HTML or XML document based on their tag name, attributes, or contents. You can also navigate the document's tree structure to find nested elements. Once you have located the elements you are interested in, you can extract their data and use it for analysis or processing.



The screenshot shows a Jupyter Notebook titled 'Webscrapingzep' running on a local host. The interface includes a top bar with navigation icons, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a toolbar with icons for file operations and code execution. The notebook content shows two input cells. The first cell, labeled 'In [5]:', contains a comment and a line of code: `# to parse the content, we will use BeautifulSoup Library and the python's default library` and `soup=BeautifulSoup(response.text,"html.parser")`. The second cell, labeled 'In [6]:', contains the code `print(soup)`. The output of the second cell is displayed in a scrollable area, showing the HTML document structure: `<!DOCTYPE html>`, `<html lang="en">`, `<head>`, `<link href="/static/css/main.css" rel="stylesheet" type="text/css"/>`, `<link href="/static/css/sandbox.css" rel="stylesheet" type="text/css"/>`, `<script src="/static/js/lib/external/jquery.js" type="text/javascript"></script>`, `<script src="/static/js/lib/external/jquery.cookie.js" type="text/javascript"></script>`, `<script src="/static/js/lib/external/angular.js" type="text/javascript"></script>`, and `<script src="/static/js/lib/external/less.js" type="text/javascript"></script>`. The bottom of the image shows a Windows taskbar with various application icons and system information like '23°C Sunny' and '1:04 PM 5/28/2023'.

```
In [5]: # to parse the content, we will use BeautifulSoup Library and the python's default library
soup=BeautifulSoup(response.text,"html.parser")

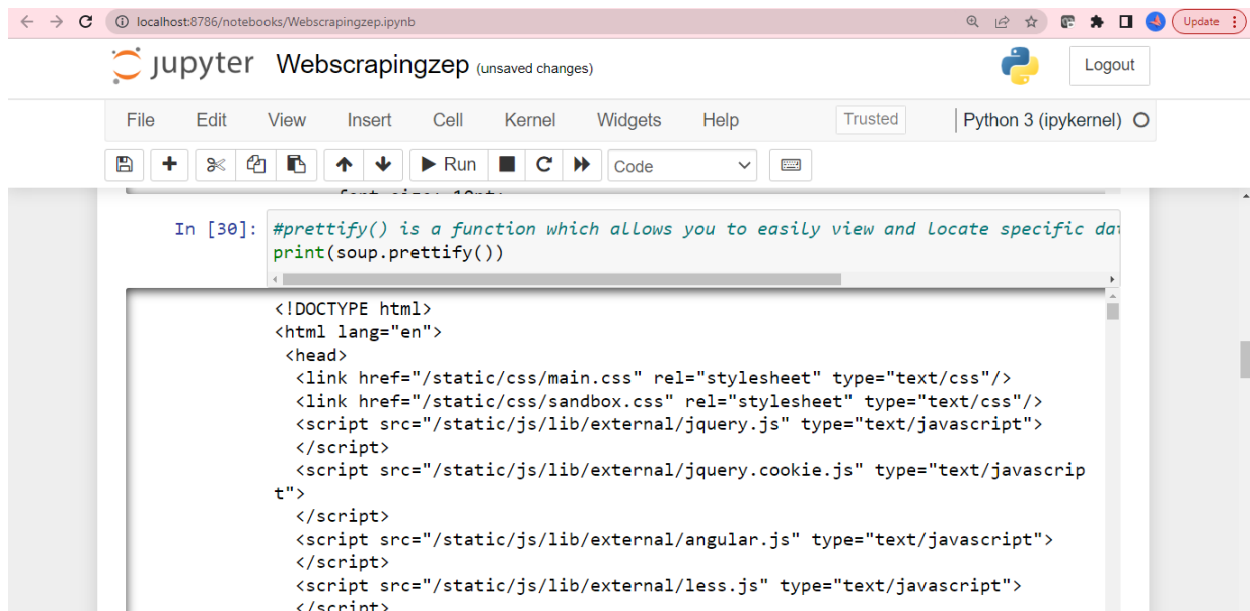
In [6]: print(soup)

<!DOCTYPE html>

<html lang="en">
<head>
<link href="/static/css/main.css" rel="stylesheet" type="text/css"/>
<link href="/static/css/sandbox.css" rel="stylesheet" type="text/css"/>
<script src="/static/js/lib/external/jquery.js" type="text/javascript"></script>
<script src="/static/js/lib/external/jquery.cookie.js" type="text/javascript"></script>
<script src="/static/js/lib/external/angular.js" type="text/javascript"></script>
<script src="/static/js/lib/external/less.js" type="text/javascript"></script>
```

We are using Python's default library (html.parser) with BeautifulSoup to parse text.

You can then use `prettify ()` function which allows you to easily view and locate specific data points within the HTML or XML page.



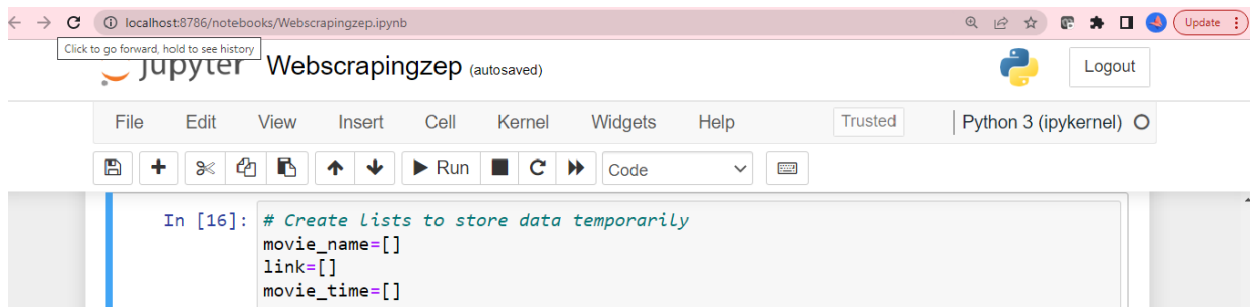
The screenshot shows a Jupyter Notebook titled 'Webscrapingzep' with 'unsaved changes'. The interface includes a top bar with navigation icons, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a toolbar with icons for saving, adding cells, and running code. The code cell contains the following Python code:

```
In [30]: #prettyfy() is a function which allows you to easily view and locate specific data
print(soup.prettify())
```

The output of the code is a formatted HTML document:

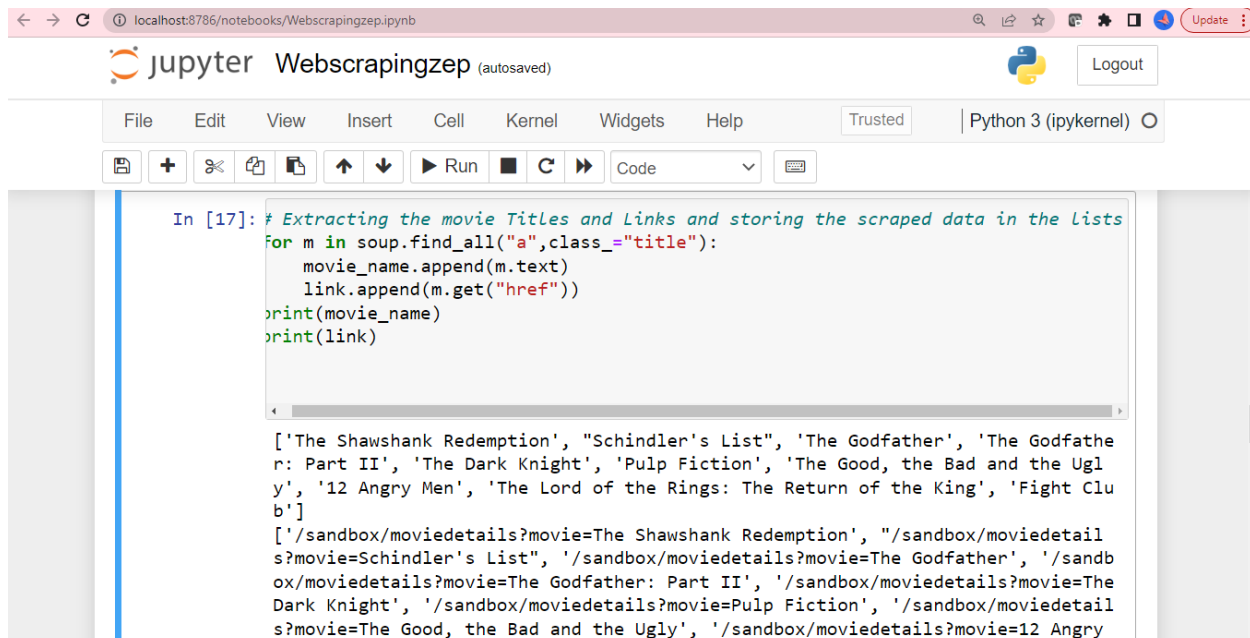
```
<!DOCTYPE html>
<html lang="en">
<head>
  <link href="/static/css/main.css" rel="stylesheet" type="text/css"/>
  <link href="/static/css/sandbox.css" rel="stylesheet" type="text/css"/>
  <script src="/static/js/lib/external/jquery.js" type="text/javascript">
  </script>
  <script src="/static/js/lib/external/jquery.cookie.js" type="text/javascript">
  </script>
  <script src="/static/js/lib/external/angular.js" type="text/javascript">
  </script>
  <script src="/static/js/lib/external/less.js" type="text/javascript">
  </script>
```

In this project, we are extracting the movie title, movie link, and movie time. Just before the for loops, add the following lists to your Python code.



The screenshot shows a Jupyter Notebook titled 'Webscrapingzep' with 'autosaved' status. The code cell contains the following Python code:

```
In [16]: # Create Lists to store data temporarily
movie_name=[]
link=[]
movie_time=[]
```

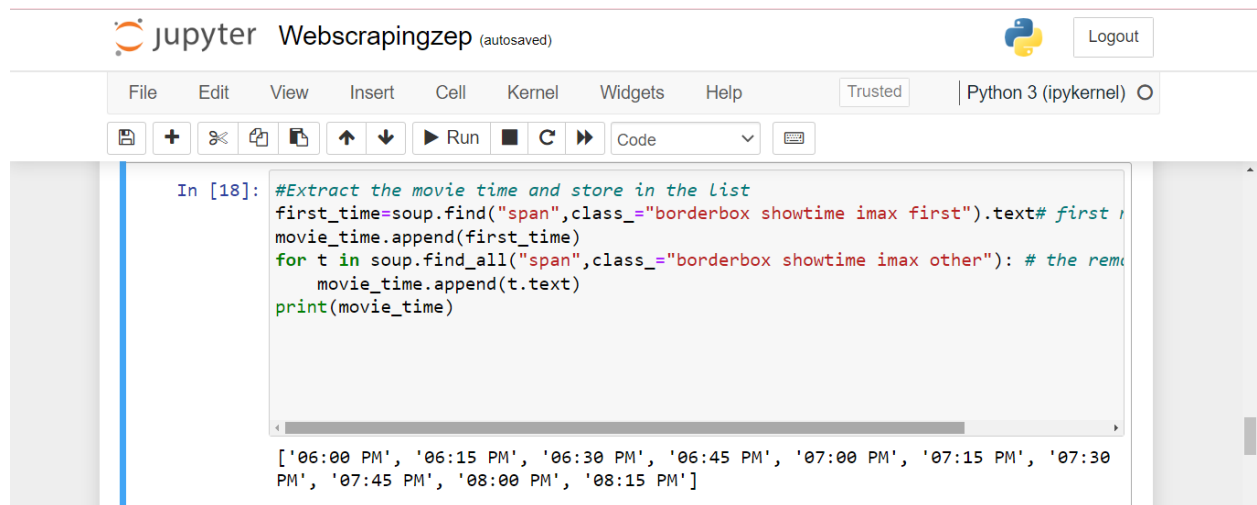


The screenshot shows a Jupyter Notebook titled 'Webscrapingzep' with 'autosaved' status. The code cell contains the following Python code:

```
In [17]: # Extracting the movie Titles and Links and storing the scraped data in the Lists
for m in soup.find_all("a",class_="title"):
    movie_name.append(m.text)
    link.append(m.get("href"))
print(movie_name)
print(link)
```

The output of the code is two lists:

```
['The Shawshank Redemption', 'Schindler's List', 'The Godfather', 'The Godfather: Part II', 'The Dark Knight', 'Pulp Fiction', 'The Good, the Bad and the Ugly', '12 Angry Men', 'The Lord of the Rings: The Return of the King', 'Fight Club']
[/sandbox/moviedetails?movie=The Shawshank Redemption', '/sandbox/moviedetails?movie=Schindler's List', '/sandbox/moviedetails?movie=The Godfather', '/sandbox/moviedetails?movie=The Godfather: Part II', '/sandbox/moviedetails?movie=The Dark Knight', '/sandbox/moviedetails?movie=Pulp Fiction', '/sandbox/moviedetails?movie=The Good, the Bad and the Ugly', '/sandbox/moviedetails?movie=12 Angry
```

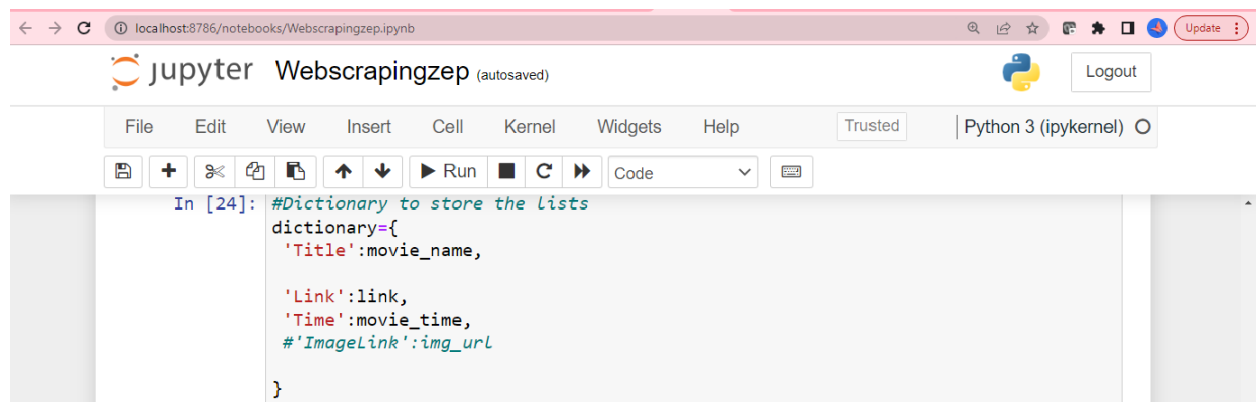


```
In [18]: #Extract the movie time and store in the list
first_time=soup.find("span",class_="borderbox showtime imax first").text# first
movie_time.append(first_time)
for t in soup.find_all("span",class_="borderbox showtime imax other"): # the rem
    movie_time.append(t.text)
print(movie_time)

['06:00 PM', '06:15 PM', '06:30 PM', '06:45 PM', '07:00 PM', '07:15 PM', '07:30 PM', '07:45 PM', '08:00 PM', '08:15 PM']
```

Step 7: Save extracted data to a file

To keep all of our lists together, we must define a dictionary. The procedure is as follows.



```
In [24]: #Dictionary to store the lists
dictionary={
    'Title':movie_name,

    'Link':link,
    'Time':movie_time,
    #'ImageLink':img_url

}
```

Specify the column names and use the pandas library to store the dictionary in a dataframe.

localhost:8786/notebooks/Webscrappingzep.ipynb

jupyter Webscrappingzep (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [26]: # store the dictionary in a dataframe using the pandas library and specify the columns
df=pd.DataFrame(dictionary,columns=['Title','Link','Time'])

In [27]: df #print the dataframe

Out[27]:

	Title	Link	Time
0	The Shawshank Redemption	/sandbox/moviedetails?movie=The Shawshank Rede...	06:00 PM
1	Schindler's List	/sandbox/moviedetails?movie=Schindler's List	06:15 PM
2	The Godfather	/sandbox/moviedetails?movie=The Godfather	06:30 PM
3	The Godfather: Part II	/sandbox/moviedetails?movie=The Godfather: Par...	06:45 PM

Finally save the data to a CSV file.

localhost:8786/notebooks/Webscrappingzep.ipynb

jupyter Webscrappingzep (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [22]: df.to_csv("Final_Data1.csv",index=False)# save the data to a CSV file

Voila!!!!!! This is how the final output should look 😊

localhost:8832/edit/Final_Data1.csv

jupyter Final_Data1.csv Yesterday at 00:30 Logout

File Edit View Language current mode

```

1 Title,Link,Time
2 The Shawshank Redemption,/sandbox/moviedetails?movie=The Shawshank Redemption,06:00 PM
3 Schindler's List,/sandbox/moviedetails?movie=Schindler's List,06:15 PM
4 The Godfather,/sandbox/moviedetails?movie=The Godfather,06:30 PM
5 The Godfather: Part II,/sandbox/moviedetails?movie=The Godfather: Part II,06:45 PM
6 The Dark Knight,/sandbox/moviedetails?movie=The Dark Knight,07:00 PM
7 Pulp Fiction,/sandbox/moviedetails?movie=Pulp Fiction,07:15 PM
8 "The Good, the Bad and the Ugly",/sandbox/moviedetails?movie=The Good, the Bad and the
  Ugly",07:30 PM
9 12 Angry Men,/sandbox/moviedetails?movie=12 Angry Men,07:45 PM
10 The Lord of the Rings: The Return of the King,/sandbox/moviedetails?movie=The Lord of the
  Rings: The Return of the King,08:00 PM
11 Fight Club,/sandbox/moviedetails?movie=Fight Club,08:15 PM
12

```