Topic	Web Scraping	
Class Description	**Students will scrape the data from NASA's website to analyze and filter the same for future classes.**	
Class	**C127**	
Class time	**45 mins**	
Goal	● Learn about web scraping ● Understand how tools like google inspect, selenium, etc. can be used for our advantage	
Resources Required	● Teacher Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm Up** **Teacher-led Activity** **Student-led Activity** **Wrap up**	**5 mins** **15 min** **15 min** **5 min**

CONTEXT

- **Review the concepts learned in the earlier classes**

Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 mins)	Hi <Student Name>! We have learned a lot of things in this module so far! Can you recall all the things we have learnt?	**ESR:** - Statistics and Charts - Linear Regression - Logistic Regression - Decision Tree - Naive Bayes

		- Flask APIs
	I have an exciting quiz question for you! Are you ready to answer this question? Teacher click on the Quiz Time button on the bottom right corner of your screen to start the In-Class Quiz. A quiz will be visible to both you and the student. Encourage the student to answer the quiz question. The student may choose the wrong option, help the student to think correctly about the question and then answer again. After the student selects the correct option, the End Quiz button will start appearing on your screen. Click the End quiz to close the quiz pop-up and continue the class.	ESR: Yes
	Yes! Now, with everything we have learned, it's time to put our knowledge to use. Up until now, we have provided you with the datasets which you used to perform different statistical and machine learning concepts, but we will not always receive the data.	**ESR:** "Yes!"

	Therefore in today's class, we will learn about Web-Scraping, where we will write a program that can fetch all the data from NASA's website for us. Are you excited?	
	Let's dive into the code.	
Teacher Initiates Screen Share		
CHALLENGE ● **Explanation of why and how web-scraping can be used to gather data from the web for different purposes**		
Step 2: Teacher-led Activity (20 min)	*Teacher shows the website to the student that needs to be scrapped.* *<Teacher opens the link from Teacher activity 1>* https://exoplanets.nasa.gov/exoplanet-catalog/	
	In today's class, we will scrape this website's data. In this website, there is data present for about 4,277 exo-planets in 428 pages. We will later use this data to perform analysis and do deep study using machine learning, etc. Are you excited about it?	**ESR:** Yes
	Let's start by creating a virtual environment in a new directory - **python3.8 -m venv venv**	

	Let's source the virtual environment - MACOS/UBUNTU - **source venv/bin/activate** WINDOWS - **venv\Scripts\activate.bat**	
	Great, now let's pip install a few modules - **pip install bs4** **pip install selenium** Great! Let's understand these modules before we proceed.	
	bs4 (BeautifulSoup) is a python module, which is famously used for parsing text as HTML and then performing actions in it, such as finding specific HTML tags with a particular class/id, or listing out all the li tags inside the ul tags. Selenium, on the other hand, is used to interact with the webpage. It is famously used for automation testing, such as testing the functionality of a website (Login/Logout/etc.) but can be also used to interact with the page such as clicking a button, etc. Since we have to scrape data from 428 pages, clicking on the button to	-

	go to the next page would come in handy. Selenium opens up the webpage in a browser. Now that we know the purpose of these modules, let's start coding.	

	Create a new file in your directory where you created the virtual environment, and name it as scraper.py.
	
	Now import the desired modules into this file:
	
	`from selenium import webdriver`
	`from bs4 import BeautifulSoup`
	`import time`
	`import csv`
	
	We are importing time to make our code sleep for some time, so that the web-page could load properly before we start scraping. We are importing csv so that we can export the data that we scrape into CSV.	

	Now, let's open the link we want to scrape with Chrome browser using Selenium - `START_URL = "https://exoplanets.nasa.gov/exoplanet-catalog/"`	

```
browser =
webdriver.Chrome("/path/to/chromedriv
er")
browser.get(START_URL)
time.sleep(10)
```

Now, what is chromedriver here? It's a driver that will help us open chrome browser with Selenium. For that, we will have to download it from the following link -

https://chromedriver.chromium.org/downloads

Once downloaded, your /path/to/chromedriver should be the path where it is downloaded. For ease, you can have it in the same folder as your code.

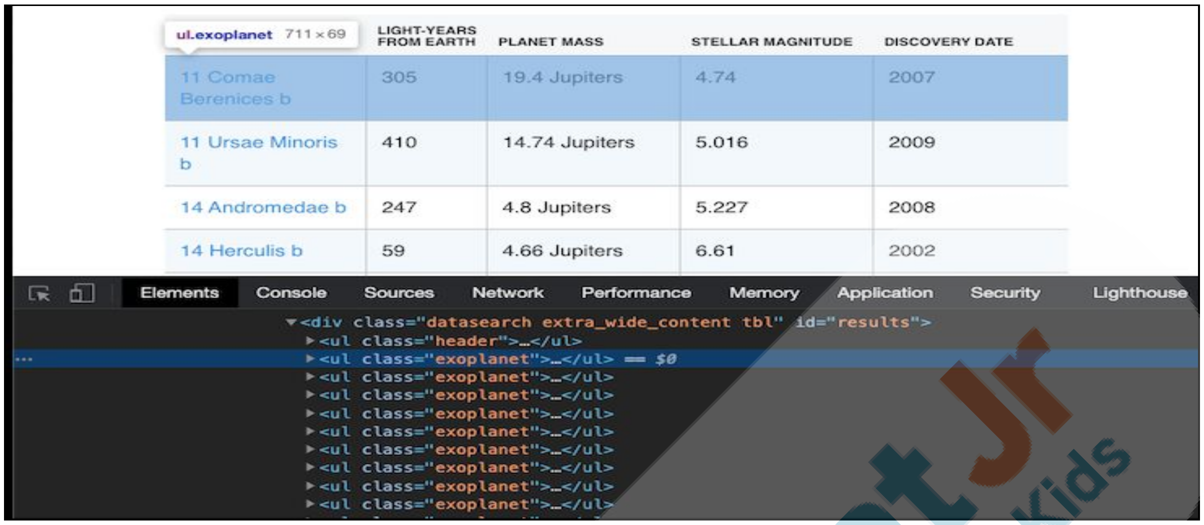
If you are a mac user, there is one additional step involved after the chromedriver is installed. Run the following command in your terminal -

```
xattr -d com.apple.quarantine
/path/to/chromedriver
```

MacOS in general is heavily protected from spam or risky softwares, and it does not trust the drivers downloaded from the web. By doing the step above, we are letting Apple know to not quarantine the driver we just downloaded and trust it.

	Okay, now we are all set to code. I will create a function called scrape() and inside that function, I will add a list called headers and planet data - `def scrape():` `headers = ["name", "light_years_from_earth", "planet_mass", "stellar_magnitude", "discovery_date"]` `planet_data = []` Inside the headers, I have the name of the columns I can see on the table mentioned in the web-page we are scraping, and the planet_data would be to save all the details of the planet. We will create a csv from these lists.	
	Now, let's just try to scrape the first page only. Before we do that, let's inspect the page. We can see that all the rows in the table are ul tags with class as exoplanet.	

```python
def scrape():
    headers = ["name",
"light_years_from_earth",
"planet_mass", "stellar_magnitude",
"discovery_date"]
    planet_data = []
```



	LIGHT-YEARS FROM EARTH	PLANET MASS	STELLAR MAGNITUDE	DISCOVERY DATE
11 Comae Berenices b	305	19.4 Jupiters	4.74	2007
11 Ursae Minoris b	410	14.74 Jupiters	5.016	2009
14 Andromedae b	247	4.8 Jupiters	5.227	2008
14 Herculis b	59	4.66 Jupiters	6.61	2002

```
▼<div class="datasearch extra_wide_content tbl" id="results">
  ▶<ul class="header">…</ul>
  ▶<ul class="exoplanet">…</ul> == $0
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
  ▶<ul class="exoplanet">…</ul>
```

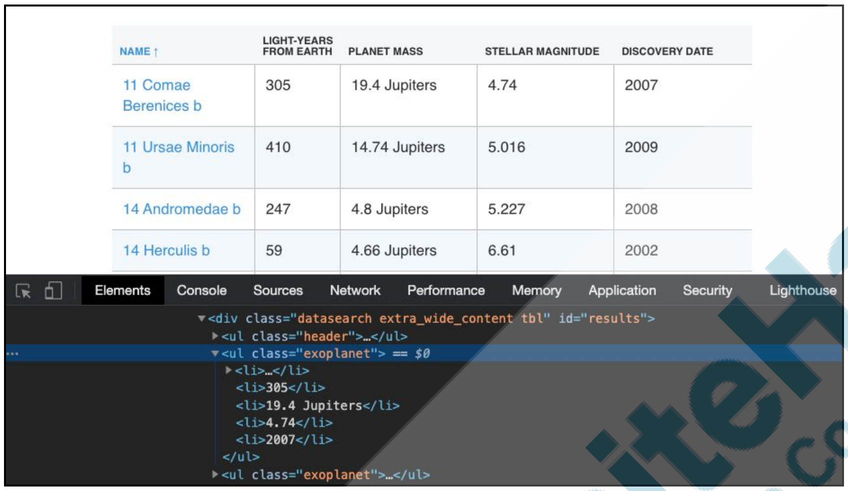
Therefore, we need to find all the ul tags with class exoplanet in order to scrape the data. We can do this with the following code:

```
soup = 
BeautifulSoup(browser.page_source, 
"html.parser")
        for ul_tag in 
soup.find_all("ul", attrs={"class", 
"exoplanet"}):
```

Earlier, the chrome window we opened with Selenium, we named it as browser. Now, we are creating a BeautifulSoup object where we are passing the browser's page source and asking our bs4 to use html.parser in it, which means it will read the page as an HTML.

Next, we are creating a for loop to iterate over all the ul_tags inside

	soup.find_all(“ul”, attrs={“class”: “exoplanet”}), meaning that it will final all the ul_tags with class exoplanet.	
	Next, let’s again check the HTML with google inspect to see what’s inside the ul tag.	

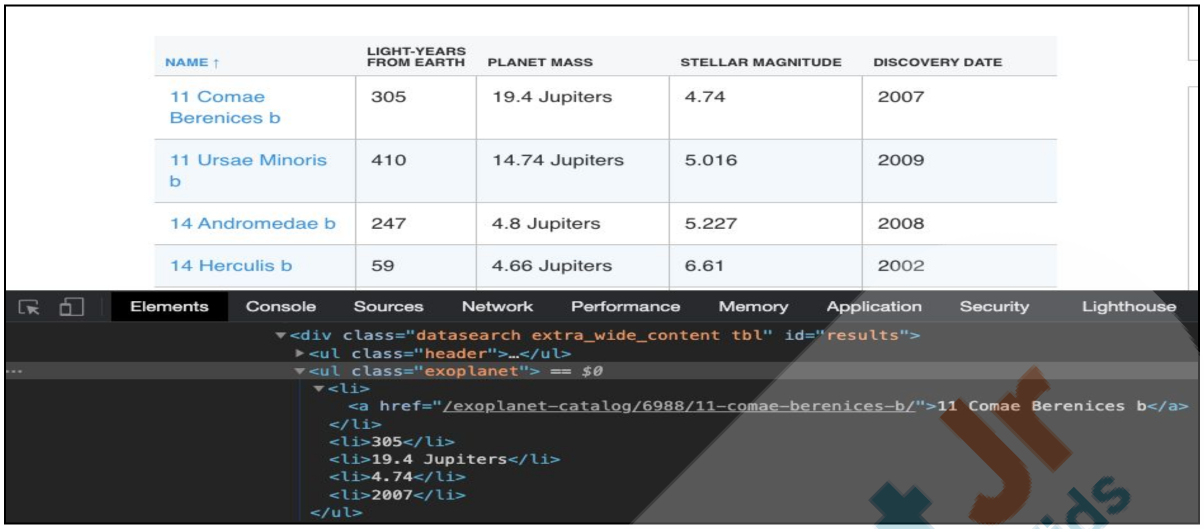


NAME ↑	LIGHT-YEARS FROM EARTH	PLANET MASS	STELLAR MAGNITUDE	DISCOVERY DATE
11 Comae Berenices b	305	19.4 Jupiters	4.74	2007
11 Ursae Minoris b	410	14.74 Jupiters	5.016	2009
14 Andromedae b	247	4.8 Jupiters	5.227	2008
14 Herculis b	59	4.66 Jupiters	6.61	2002

```
▼<div class="datasearch extra_wide_content tbl" id="results">
  ▶<ul class="header">…</ul>
  ▼<ul class="exoplanet"> == $0
    ▶<li>…</li>
     <li>305</li>
     <li>19.4 Jupiters</li>
     <li>4.74</li>
     <li>2007</li>
   </ul>
  ▶<ul class="exoplanet">…</ul>
```

	Here, we can see that it consists of li tags inside which we can get data. Again, we will need to iterate over all the li tags. For this, we will find all the li tags. Can you tell me how I can find all the li tags inside the ul_tag?	**ESR:** `li_tags = ul_tag.find_all("li")`
	Great, now all we have to do is to iterate over these li tags and fetch the data, create a temporary list and then finally append that list into the planet_data list that we created earlier. Let’s inspect the li tags a bit deeper.	

WhiteHat Jr
Live online Coding for Kids



NAME ↑	LIGHT-YEARS FROM EARTH	PLANET MASS	STELLAR MAGNITUDE	DISCOVERY DATE
11 Comae Berenices b	305	19.4 Jupiters	4.74	2007
11 Ursae Minoris b	410	14.74 Jupiters	5.016	2009
14 Andromedae b	247	4.8 Jupiters	5.227	2008
14 Herculis b	59	4.66 Jupiters	6.61	2002

```
▼<div class="datasearch extra_wide_content tbl" id="results">
  ▶<ul class="header">…</ul>
  ▼<ul class="exoplanet"> == $0
    ▼<li>
        <a href="/exoplanet-catalog/6988/11-comae-berenices-b/">11 Comae Berenices b</a>
      </li>
      <li>305</li>
      <li>19.4 Jupiters</li>
      <li>4.74</li>
      <li>2007</li>
    </ul>
```

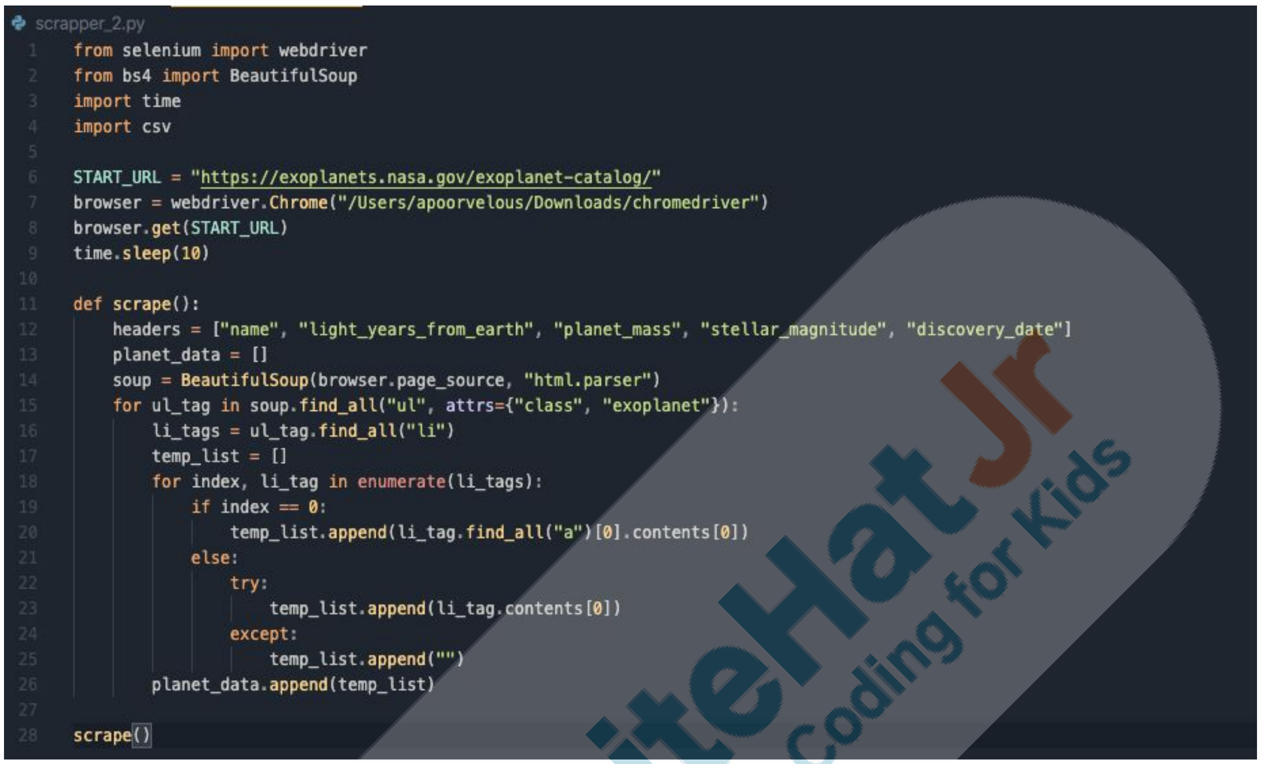
Here, we can see that the li tags have the name of the planet inside an anchor tag, and other details directly as HTML. For this, we need to make sure that we treat the first li tag differently and others differently. For this, we will write the following code:

```
temp_list = []
            for index, li_tag in
enumerate(li_tags):
                if index == 0:

temp_list.append(li_tag.find_all("a")
[0].contents[0])
                else:
                    try:

temp_list.append(li_tag.contents[0])
                    except:

temp_list.append("")

planet_data.append(temp_list)
```

	Let’s break it down line by line. We are first creating a temp_list to store all the data of this row. Then, we are iterating over the li_tags we fetched earlier but this time, in a different way. What is enumerate?	**ESR:** Enumerate is a function that returns the index along with the element.
	That’s right! Enumerate will give us both the index and the element on that index, instead of just the index that a traditional loop gives. Using this index, if the index is 0 (first element), we are first finding the anchor tag inside the li_tag, and then copying the inner HTML of it. Else, we want to directly copy the inner HTML of the li_tag. To facilitate this, in case a column is empty and we get an error, we are going to use the try except block. Lastly, we will append this temp_list into the planet_data.	

	Our code so far looks like this:
	
	`from selenium import webdriver`
	`from bs4 import BeautifulSoup`
	`import time`
	`import csv`
	
	`START_URL =`
	`"https://exoplanets.nasa.gov/exoplane`
	`t-catalog/"`	

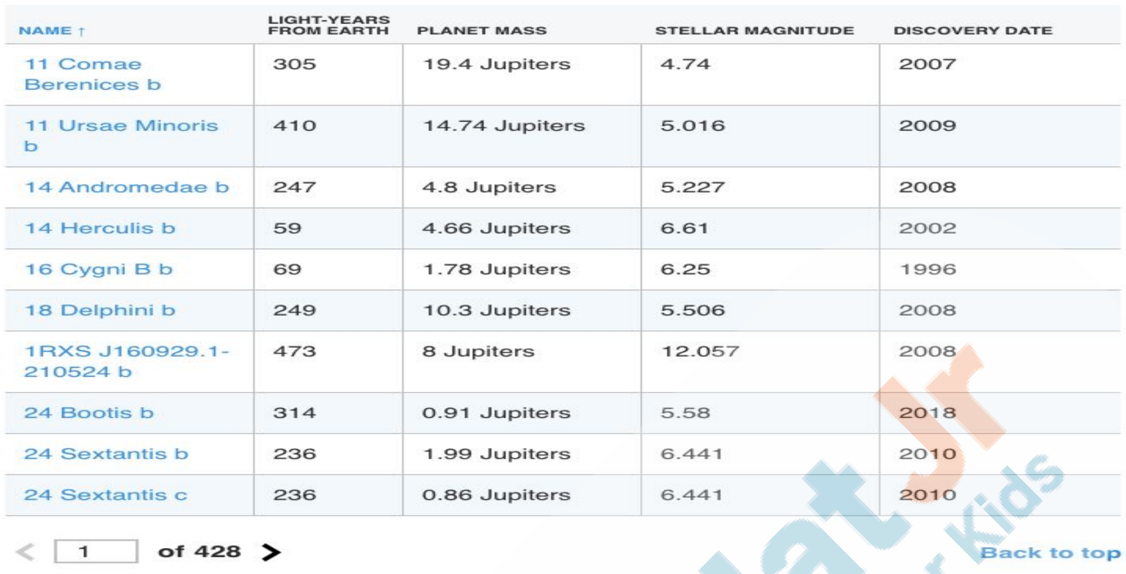
```
browser = webdriver.Chrome("/Users/apoorvelous/Downloads/chromedriver")
browser.get(START_URL)
time.sleep(10)

def scrape():
    headers = ["name", "light_years_from_earth", "planet_mass", "stellar_magnitude", "discovery_date"]
    planet_data = []
    soup = BeautifulSoup(browser.page_source, "html.parser")
    for ul_tag in soup.find_all("ul", attrs={"class", "exoplanet"}):
        li_tags = ul_tag.find_all("li")
        temp_list = []
        for index, li_tag in enumerate(li_tags):
            if index == 0:
                temp_list.append(li_tag.find_all("a")[0].contents[0])
            else:
                try:
                    temp_list.append(li_tag.contents[0])
                except:
                    temp_list.append("")
        planet_data.append(temp_list)
```



scrapper_2.py

```python
from selenium import webdriver
from bs4 import BeautifulSoup
import time
import csv

START_URL = "https://exoplanets.nasa.gov/exoplanet-catalog/"
browser = webdriver.Chrome("/Users/apoorvelous/Downloads/chromedriver")
browser.get(START_URL)
time.sleep(10)

def scrape():
    headers = ["name", "light_years_from_earth", "planet_mass", "stellar_magnitude", "discovery_date"]
    planet_data = []
    soup = BeautifulSoup(browser.page_source, "html.parser")
    for ul_tag in soup.find_all("ul", attrs={"class", "exoplanet"}):
        li_tags = ul_tag.find_all("li")
        temp_list = []
        for index, li_tag in enumerate(li_tags):
            if index == 0:
                temp_list.append(li_tag.find_all("a")[0].contents[0])
            else:
                try:
                    temp_list.append(li_tag.contents[0])
                except:
                    temp_list.append("")
        planet_data.append(temp_list)

scrape()
```

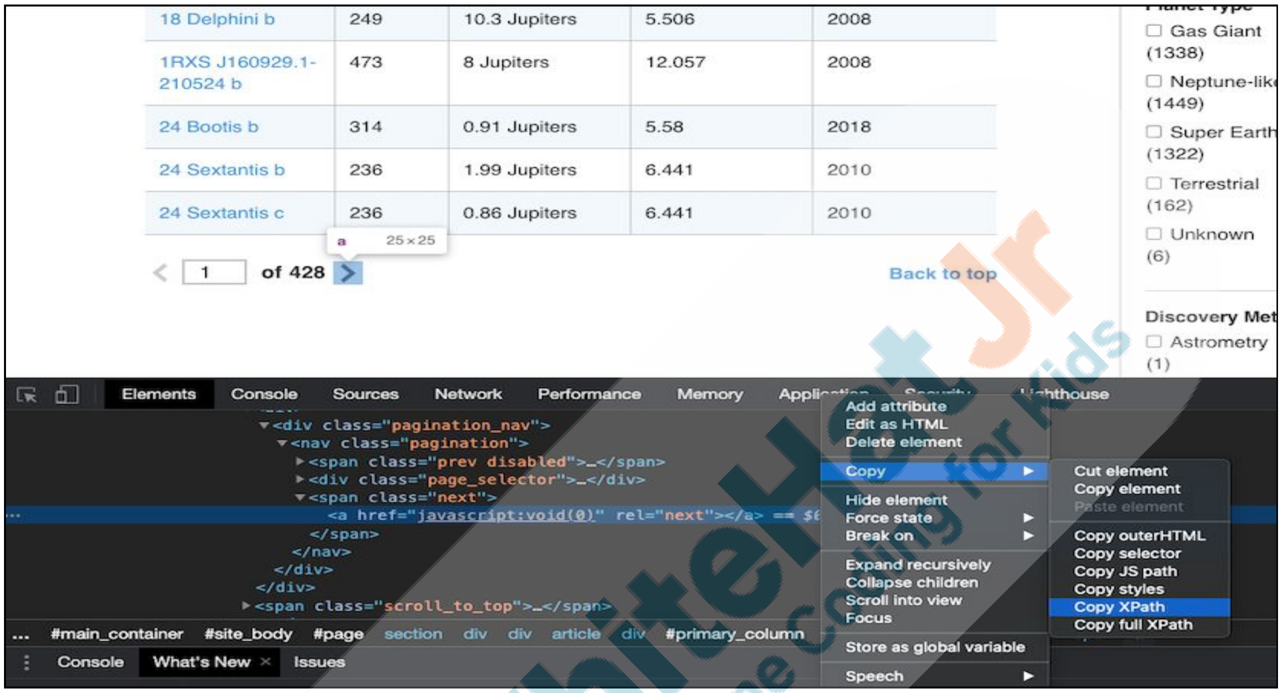
Now, one final thing that we need to still figure out is, how to change the page by clicking on the next button. Let's check on the browser first -



NAME ↑	LIGHT-YEARS FROM EARTH	PLANET MASS	STELLAR MAGNITUDE	DISCOVERY DATE
11 Comae Berenices b	305	19.4 Jupiters	4.74	2007
11 Ursae Minoris b	410	14.74 Jupiters	5.016	2009
14 Andromedae b	247	4.8 Jupiters	5.227	2008
14 Herculis b	59	4.66 Jupiters	6.61	2002
16 Cygni B b	69	1.78 Jupiters	6.25	1996
18 Delphini b	249	10.3 Jupiters	5.506	2008
1RXS J160929.1-210524 b	473	8 Jupiters	12.057	2008
24 Bootis b	314	0.91 Jupiters	5.58	2018
24 Sextantis b	236	1.99 Jupiters	6.441	2010
24 Sextantis c	236	0.86 Jupiters	6.441	2010

	There, we have a button at the bottom of the page. We want to go to the next page after our code is done with all the ul_tags of the current page. We can go to the next page with the following code: `browser.find_element_by_xpath('//*[@id="primary_column"]/footer/div/div/div/nav/span[2]/a').click()` Here, we are finding an element with xpath, and then clicking it. But, what is xpath?	**ESR:** Varied
	XPath can be used to navigate through elements and attributes in an XML document. XPath is a syntax for defining parts of an XML document.	

Let’s see how to get the xpath of an element -



18 Delphini b	249	10.3 Jupiters	5.506	2008
1RXS J160929.1-210524 b	473	8 Jupiters	12.057	2008
24 Bootis b	314	0.91 Jupiters	5.58	2018
24 Sextantis b	236	1.99 Jupiters	6.441	2010
24 Sextantis c	236	0.86 Jupiters	6.441	2010

```
▼<div class="pagination_nav">
  ▼<nav class="pagination">
    ▶<span class="prev disabled">…</span>
    ▶<div class="page_selector">…</div>
    ▼<span class="next">
        <a href="javascript:void(0)" rel="next"></a> == $0
      </span>
    </nav>
  </div>
</div>
▶<span class="scroll_to_top">…</span>
```

That’s right! We just need to find the element in google inspect, right click on it and the click on **copy xpath**.

Now, we have successfully scraped the first page and we have also seen how we can go to the next page.

We need to scrape 428 pages in total. Can you tell me how we can do that?

ESR:
By looping over 428 times.

Teacher Stops Screen Share

Now it's your turn. Please share your screen with me.

- **Ask Student to press ESC key to come back to panel**
- **Guide Student to start Screen Share**

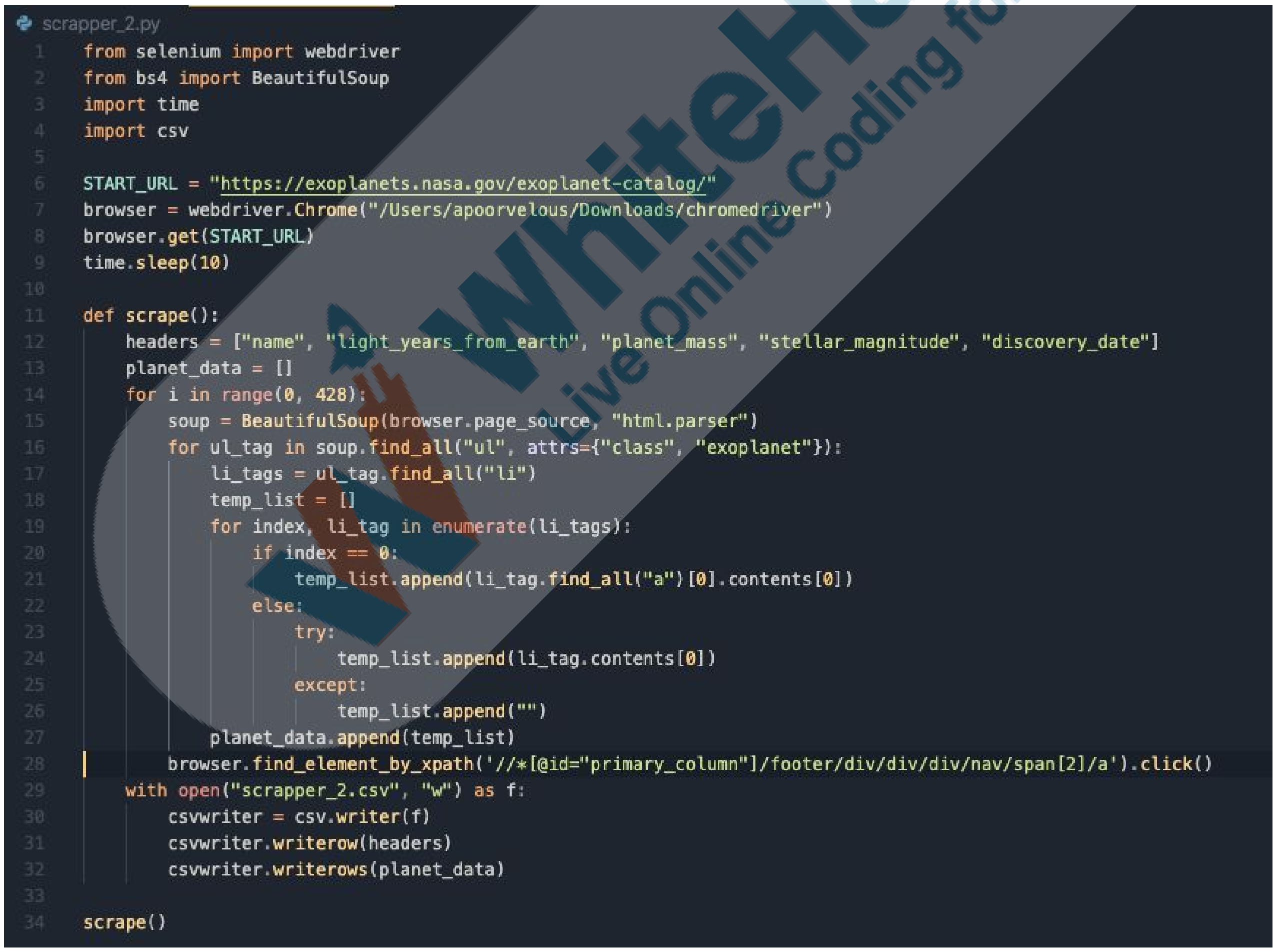
WhiteHat Jr
Live online Coding for Kids

- **Teacher gets into Fullscreen**

ACTIVITY

- **Student will implement the loop to the code**
- **Student will generate a csv of the data scraped**
- **Student will cross verify the data of the csv with the data of the website**

Step 3: Student-Led Activity (10 min)	*Help students implement the loop and create a csv.* Final code would look like this:	*The student writes code to implement the loop 428 times and then create a csv of all the data fetched.*


scrapper_2.py

```python
from selenium import webdriver
from bs4 import BeautifulSoup
import time
import csv

START_URL = "https://exoplanets.nasa.gov/exoplanet-catalog/"
browser = webdriver.Chrome("/Users/apoorvelous/Downloads/chromedriver")
browser.get(START_URL)
time.sleep(10)

def scrape():
    headers = ["name", "light_years_from_earth", "planet_mass", "stellar_magnitude", "discovery_date"]
    planet_data = []
    for i in range(0, 428):
        soup = BeautifulSoup(browser.page_source, "html.parser")
        for ul_tag in soup.find_all("ul", attrs={"class", "exoplanet"}):
            li_tags = ul_tag.find_all("li")
            temp_list = []
            for index, li_tag in enumerate(li_tags):
                if index == 0:
                    temp_list.append(li_tag.find_all("a")[0].contents[0])
                else:
                    try:
                        temp_list.append(li_tag.contents[0])
                    except:
                        temp_list.append("")
            planet_data.append(temp_list)
        browser.find_element_by_xpath('//*[@id="primary_column"]/footer/div/div/div/nav/span[2]/a').click()
    with open("scrapper_2.csv", "w") as f:
        csvwriter = csv.writer(f)
        csvwriter.writerow(headers)
        csvwriter.writerows(planet_data)

scrape()
```

	Awesome! We just scraped our first web-page! Now, let's randomly pick a few planets from the last of the csv and check if they match with the planets mentioned in the last page of the website.	*Student cross checks the data.*
	Bravo! We're done!	
Teacher Guides Student to Stop Screen Share		
FEEDBACK • **Appreciate the student for their efforts** • **Identify 2 strengths and 1 area of progress for the student**		
Step 4: Wrap-Up (5 min)	So, in this project class we learned how we can easily fetch data from a website. You can try to scrape different websites and generate lots of data for machine learning/statistical analysis.	**ESR:** varied
Teacher Clicks ✖ End Class		

Activity	Activity Name	Links
Teacher Activity 1	site link	https://exoplanets.nasa.gov/exoplanet-catalog/

Teacher Activity 2	solution	https://github.com/whitehatjr/Scraper/tree/master/scraper