Week2_9_DataCollection

Outline

1. Introduction to Selenium
2. Control Web Browser Using Selenium
3. Web Scraping in a Loop

1. Introduction to Selenium

Selenium is a Python module (library/package) for web automation and testing. It allows you to control a web browser through your scripts, enabling you to automate tasks such as filling out forms, clicking buttons, navigating pages, and extracting data. Selenium is commonly used for web scraping, automated testing of web applications, and performing repetitive web-based tasks.

1.1 Install selenium

We are using selenium to obtain data from webpages. Now, launch the Jupyter Notebook and import selenium.

In [38]:
```
import selenium
```

In [1]:
```
import selenium
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-abb2a9e03f2a> in <module>
----> 1 import selenium

ModuleNotFoundError: No module named 'selenium'
```

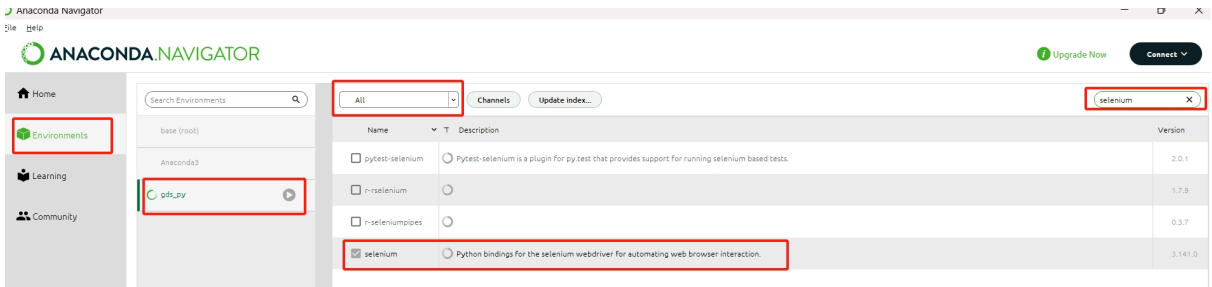
This error indicates that the selenium is not installed in your computer. To install a package into its own environment, please follow these steps:

In [11]:
```
# the first way:
!pip install selenium
```

```
Requirement already satisfied: selenium in c:\users\lwz12\anaconda3\envs\gds_py\lib\site-packages (3.141.0)
Requirement already satisfied: urllib3 in c:\users\lwz12\anaconda3\envs\gds_py\lib\site-packages (from seleni
um) (1.26.18)
```

An alternative way:

- (1) Open Anaconda Navigator.
- (2) Click Connect, then click SIGN IN next to Anaconda.org.
- (3) Select Environments from the left-hand navigation, then look for your package by name using the Search Packages field (**make sure you select "All" module, not merely those "Installed"**). Filter packages further using the dropdown above the Name column.
- (4) Select the checkbox of the selenium you want to install, then click the Apply.



Now `import selenium` again. There should be no errors.

In [39]:
```
import selenium
```

Then we create a brower representing the web browser.

1.2 Download webdriver

Using *selenium* to control web browser requires a webdriver application, which is available from the selenium website.

1. Go to the website (https://selenium.dev/) and click "Download";
2. Scroll down and you can see "Platforms Supported by Selenium". Under "Browser", there is a list of web browsers. Click your daily choice and download it accordingly.
3. Unzip the downloaded file and save the application file in the Notebook folder.

2. Control Web Browser Using Selenium

2.1 Import selenium and launch web brower

First, we import selenium in the following way. In this way, we import the module webdriver from the big library selenium so that we can call webdriver directly.

In [40]:
```
from selenium import webdriver
```

Then we create a brower representing the web browser.

In [41]:
```
# ask the webdriver to open a google chrome page for us
browser = webdriver.Chrome('chromedriver')
```

You may see some safety warning message from your system, just allow it. Now, you should see a new web browser launched.

For a different web browser, you need to change the function name (after the webdriver.) as well as the driver application name. For example,

- browser = webdriver.Chrome('chromedriver')
- browser = webdriver.Firefox('geckodriver')
- browser = webdriver.Edge('msedgedriver')

Now, we can surf the internet using selenium and this browser ! Let's try Google first.

In [42]:
```
# enter URL of google:
browser.get('http://www.google.com')
```

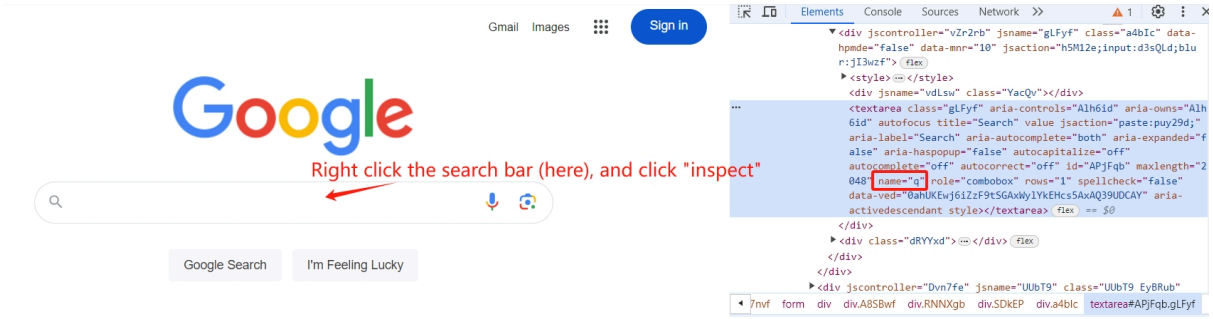
Notice that for some browers, you must keep "http://" in the web address.

2.2 Find Elements and Send Keys

Let's search something.

First, you need to find the search input window element by right clicking the search window and choose "Inspect".

Find in the source codes the specific **"id" or "name"** of the input window and then use it in the function `find_element_by_id()` or `find_element_by_name()`. Here we find the name is "q", so we use this name and define an object (variable) search for the input window.



In [16]:
```
# specify the "id" or "name" of the input window
search = browser.find_element_by_name('q')

# An alternative way:
# search = browser.find_element_by_id("APjFqb")
```

To type in texts in the window, you can call the search and use the function `send_keys()`. For example, we search "selenium" here.

In [17]:
```
# type something in the search bar using .send_keys
search.send_keys('selenium')
```

Next, we want to run the search through pressing ENTER. Selenium can do that through the function `search.send_keys(Keys.ENTER)`. But before running it, you need to import Keys from the big library Selenium.

In [18]:
```
# import Keys from selenium.
# the Keys class simulates keyboard key presses in a web browser
from selenium.webdriver.common.keys import Keys

# press ENTER
search.send_keys(Keys.ENTER)
```

Congratulations! You just controlled the web browser to search selenium.

Let's try a different website, our class roster website https://classes.cornell.edu/browse/roster/SP23). We want to search for available econometric courses.

In [19]:
```
# Enter the website:
browser.get('https://classes.cornell.edu/browse/roster/SP23')

# find the element by id
search = browser.find_element_by_id('q') # how could you find the name "q"?

# type "econometrics" in the search bar
search.send_keys('econometrics')
```

Another way to run the search

- we can click the "Go" button. First, we need to find the button name ("search-go"), and assign it to a variable search_button.

In [20]:
```
# navigate to the "Go" button
search_button = browser.find_element_by_id('search-go')
```

To click it, you call this search_button and use the function click() .

In [21]:
```
# click the go button
search_button.click()
```

2.3 Find Text using xpath

Now, if we want to save the information of all search results, how to do that?

First of all, how can you identify the course title "Econometrics I" in the first result? If you right click the title and check the source code, you may find no id or name available for the text. An alternative way is to use `find_element_by_xpath()`, a function that navigates to the target text based on Xpath.

XPath is the language used for locating nodes in an XML document (also works for HTML). To get the xpath, we right click the source code you want to locate, and go to "Copy" and then "Copy XPath". Then paste it into the function `find_element_by_xpath()`.

Note that there are many functions for locating element(s), for more of the family `find_element(s)_by_`, find this click: https://selenium-python.readthedocs.io/locating-elements.html

In [23]:
```
# try the XPath method: .find_element_by_xpath:
title = browser.find_element_by_xpath('//*[@id="dtitle-AEM7100"]')
```

To get the text inside source code, we call the title and use the function .text() .

In [24]:
```
# get the text
title.text
```

Out[24]: 'Econometrics I'

3. Test yourself

3.1 Can you do the following steps using Selenium in Python?

1. Launch your web browser and open the class roster website https://classes.cornell.edu/browse/roster/SP23;
2. Search "Econometrics";
3. Obtain and print (in Python) the class number, class title, class weekdays, class time, and class instructor for the first course. Hint: you can use `.get_attribute('any attribute name')` to get any attribute value of the element.

In [25]:
```
# (1) Launch your web browser and open the class roster website https://classes.cornell.edu/browse/roster/SP

from selenium import webdriver

# launch the web browser
browser = webdriver.Chrome('chromedriver')

# open the website
browser.get('https://classes.cornell.edu/browse/roster/SP23')
```

In [26]:
```
# (2) Search "Econometrics";

# find the search bar
search = browser.find_element_by_id('q')

# type in 'econometrics'
search.send_keys('econometrics')

# find the GO button
search_button = browser.find_element_by_id('search-go')
```

```
# click the button
search_button.click()
```

In [27]:
```
# (3) Obtain and print (in Python) the class number, class title, class weekdays, class time, and class inst

# class number
c_number = browser.find_element_by_xpath('//*[@id="head-AEM-7100"]/div[1]')
print(c_number.text)

# class title
c_title = browser.find_element_by_xpath('//*[@id="dtitle-AEM7100"]')
print(c_title.text)

# class weekday
c_weekday = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[3]/div[2]/div[1]/div[2]/ul[2
print(c_weekday.text)

# class time
c_time = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[3]/div[2]/div[1]/div[2]/ul[2]/l
print(c_time.text)

# class instructor
c_instructor = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[3]/div[2]/div[1]/div[2]/u
print(c_instructor.get_attribute('data-content'))
```

```
AEM 7100
Econometrics I
W
2:40pm - 5:10pm
Shanjun Li (sl2448)
```

3.2 Try getting the same information for the second result

In [28]:
```
# class number
c_number = browser.find_element_by_xpath('//*[@id="head-CEE-6640"]/div[1]')
print(c_number.text)

# class title
c_title = browser.find_element_by_xpath('//*[@id="dtitle-CEE6640"]')
print(c_title.text)

# class weekday
c_weekday = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[4]/div[2]/div[1]/div[2]/ul[2
print(c_weekday.text)

# class time
c_time = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[4]/div[2]/div[1]/div[2]/ul[2]/l
print(c_time.text)

# class instructor
c_instructor = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[4]/div[2]/div[1]/div[2]/u
print(c_instructor.get_attribute('data-content'))
```

```
CEE 6640
Microeconometrics of Discrete Choice
TR
11:25am - 12:40pm
Ricardo Daziano (ra477)
```

4. Web Scraping in a Loop

4.1 Question: Can we loop over all the results in the list?

The current way to obtain each piece of information does not allow us to loop (why?).

The key issue is that the XPathes of the class number and the class title contain course-specific information (e.g., "head-CEE-6640"), which are not loopable. In contrary, the XPathes of the class weekday, time, and instructor are loopable,

because they are in the format of `relative position`. Thus, we need to find relative pathes to the class number and the class title elements.

After analyzing the structure of the source code of the webpage (see lecture slides for details), we find the pathes for both elements.

In [71]:
```
# AEM 7100
# class number (using relative position XPath)
c_number = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[3]/h3[1]/div[1]')
print(c_number.text)

# class title (using relative position XPath)
c_title = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[3]/h3[1]/div[2]')
print(c_title.text)
```

```
AEM 7100
Econometrics I
```

In [72]:
```
# CEE 6640
# class number (using relative position XPath)
c_number = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[4]/h3[1]/div[1]')
print(c_number.text)

# class title (using relative position XPath)
c_title = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[4]/h3[1]/div[2]')
print(c_title.text)
```

```
CEE 6640
Microeconometrics of Discrete Choice
```

We also find that to loop from the first result to the second, we just need to change the index in the second div[] from 3 to 4. Thus, we can print the information of the first two results in a loop in the following way.

In [31]:
```
for i in range(2):  #  note that i will take values of 0 and 1

    index = i + 3    # convert i into the index used for div[]

    # class number
    c_number = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/h3[1]
    print(c_number.text)

    # class title
    c_title = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/h3[1]/
    print(c_title.text)

    # class weekday
    c_weekday = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/div[
    print(c_weekday.text)

    # class time
    c_time = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/div[2]/
    print(c_time.text)

    # class instructor
    c_instructor = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/d
    print(c_instructor.get_attribute('data-content'))
```

```
AEM 7100
Econometrics I
W
2:40pm - 5:10pm
Shanjun Li (sl2448)
CEE 6640
Microeconometrics of Discrete Choice
TR
11:25am - 12:40pm
Ricardo Daziano (ra477)
```

4.2. Loop and save the results

Now, we want to loop and save all results into a DataFrame and then export it into an Excel file. To do this, we first need to save all the results in variable lists.

4.2.1 Save results into variable lists

In [33]:
```
# create empty lists for each variable to save results
c_number_list = []
c_title_list = []
c_weekday_list = []
c_time_list = []
c_instructor_list = []

for i in range(8):  #  note that i will take values from 0

    index = i + 3    # convert i into the index used for div[]

    # class number
    c_number = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/h3[1]
    print(c_number.text)
    c_number_list.append(c_number.text)   # use .append() function to add the result to the list

    # class title
    c_title = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/h3[1]/
    print(c_title.text)
    c_title_list.append(c_title.text)

    # class weekday
    c_weekday = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/div[
    print(c_weekday.text)
    c_weekday_list.append(c_weekday.text)

    # class time
    c_time = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/div[2]/
    print(c_time.text)
    c_time_list.append(c_time.text)

    # class instructor
    c_instructor = browser.find_element_by_xpath('//*[@id="search-refresh"]/div[2]/div[' + str(index) + ']/d
    print(c_instructor.get_attribute('data-content'))
    c_instructor_list.append(c_instructor.get_attribute('data-content'))

    # print a blank line between two classes' results
    print("\n")
```

```
AEM 7100
Econometrics I
W
2:40pm - 5:10pm
Shanjun Li (sl2448)


CEE 6640
Microeconometrics of Discrete Choice
TR
11:25am - 12:40pm
Ricardo Daziano (ra477)


ECON 3120
Applied Econometrics
TR
1:00pm - 2:15pm
Douglas McKee (dmm399)


ECON 3140
Econometrics
TR
1:00pm - 2:15pm
Joerg Stoye (js2434)


ECON 6200
Econometrics II
TR
9:40am - 10:55am
Joerg Stoye (js2434)


ECON 7230
Semi and Non Parametric Econometrics
R
2:40pm - 5:10pm
Francesca Molinari (fm72)


ECON 7245
Topics in Econometrics and Machine Learning
T
2:40pm - 5:10pm
Jose Luis Montiel Olea (jlo67)


ECON 7841
Econometrics Workshop
T
11:15am - 12:45pm
Joerg Stoye (js2434)
```

4.2.2 Combine the multiple lists into a DataFrame

In [35]:
```python
import pandas as pd
```

In [36]:
```python
# combine all lists into a Dataframe
allresult = pd.DataFrame(
    {'number': c_number_list,
     'title': c_title_list,
     'weekday': c_weekday_list,
     'time': c_time_list,
     'instructor':c_instructor_list
    })
```

In [ ]:
```
# save the result DataFrame to an Excel file
allresult.to_excel('allresult.xlsx')
```

5. In-class Exercise (assignment 4 - Part 1)

Part 1 (30 Points)

Option 1: Pick one course you are taking this semester and use Selenium to obtain the course information from the Class Roster (https://classes.cornell.edu/browse/roster/SP23). Your code should show the full process (15 Points) from launching the browser to printing all the information. The course information includes

- a. (3 Points) Course number (e.g., CRP5850)
- b. (3 Points) Course title
- c. (3 Points) Course credits
- d. (3 Points) Course instructor
- e. (3 Points) Course time (weekday and time)

Option 2: If your final project needs to collect data from a website, this is the chance you can develop a web-scraping algorithm to collect your own data and show in this assignment.

(Optional 10 pts) North Amiercan Food System Network (NAFSN) is a professional association of people working to strengthen local & regional food systems. They chronicle jobs that pay a living wage in food systems development. They glean employment opportunities daily from LinkedIn, other social media platforms, regional nonprofit networks, and post positions emailed to them directly. The job list is available at:
https://members.foodsystemsnetwork.org/members/classifieds5.php?org_id=NAFS

By 6/12/2014, there are 538 jobs posted on the NAFSN website (notice that your number may different from mine). Design a web-scraping algorithm to automatically collect all posted jobs, and save the data into a DataFrame.