Day 2: Web Scraping w/ Selenium and Web Drivers

Requirements:
python 3.7+
pip

pip install selenium
geckodriver (instructions)

References:
Selenium Documentation
CSS2XPath
Selenium Practice
Code for today

Activity 1: Finding and Interacting with Elements on a Page

```
from selenium import webdriver
from selenium.webdriver.common.by import By
browser = webdriver.Firefox()
browser.get('https://hoopshype.com/salaries/players/')
```

Let's say we want to get the names of all of the players.

Selenium drivers have an easy way to find elements using find_elements()

We can select all the players on the page using:

```
players  = browser.find_elements(By.CLASS_NAME, "name")
```

We can then iterate through the players and print their names.

```
for player in players:
     print(player.text)
```

Let's say we want to traverse each page one by one using clicking from the dropdown menu.

```
dropdown_parent = browser.find_element(By.CLASS_NAME, "all")
dropdown_parent.click()
```

This will open the dropdown menu for us.

We can get all of the options for the dropdown menu with

```
dropdown_options = browser.find_elements(By.CLASS_NAME,
"team-list-ul")
```

We can verify we have the correct options selected by printing them:

```
for option in dropdown_options:
      print(option.text)
```

Now, the dropdown_options is a list of one. We can confirm that by printing

```
print(dropdown_options)
```

Meaning, we can't click on the text of the teams-list-ul in order to get to new page.
We have to find the specific **<a>** tag to click on.

To get the first entry, which is the page shown by default (2023/2024):

```
team_link = browser.find_element(By.CLASS_NAME,
'team-list-ul').find_elements(By.TAG_NAME, 'a')[0]
```

To get the second entry (2022/2023):

```
team_link = browser.find_element(By.CLASS_NAME,
'team-list-ul').find_elements(By.TAG_NAME, 'a')[1]
```

and so on, replacing the digit in [ ] with the desired one.
Click on the link you want to with `team_link.click()`

We can of course grab the players again from the new page (2022/2023) as well.

```
players  = browser.find_elements(By.CLASS_NAME, "name")
for player in players:
    print(player.text)
```

Once we click on a team_link with `team_link.click()`
the drop-down menu disappears. So we need to do this over again to bring the drop-down menu up again, which shows us the dropdown_options:

```
dropdown_parent = browser.find_element(By.CLASS_NAME, "all")
dropdown_parent.click()
```

It would be nice if we made a method to click on the dropdown button and automatically click the next page for us given an **n** value.

```
def next_page(browser, n):
    dropdown_parent=browser.find_element(By.CLASS_NAME, "all")
    dropdown_parent.click()
    team_link = browser.find_element(By.CLASS_NAME,
'team-list-ul').find_elements(By.TAG_NAME, 'a')[n]
    team_link.click()
```

Calling `next_page(browser, 2)` points us to the 2020-2021 page.

We can also write a method to get the players off a page for us and append it to a list, `players_list`

```
def get_players(browser, players_list):
    players  = browser.find_elements(By.CLASS_NAME, "name")
    for player in players:
      players_list.append(player.text)
```

We have the building blocks to be able to get to the next page of the site and scrape the players from this site.
The only thing missing is a for loop that has us go through each page until we reach the last page. I will leave this for you to try.

Activity 2: [Form Authentication]

We will move onto filling out a form for authentication, which will allow us to practice finding form fields and inputting data, which are all tasks we will need to do for scraping content from authenticated sites like forums or portals.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
browser = webdriver.Firefox()
browser.get('https://the-internet.herokuapp.com/login')
```

By inspection we can see the username field is an <input> with id username, which makes it easy. We can find the element by ID directly.

```
user_field= browser.find_element(By.ID, 'username')
user_field.click()
```

We can input text into the field using send_keys()

```
user_field.send_keys('tomsmith')
```

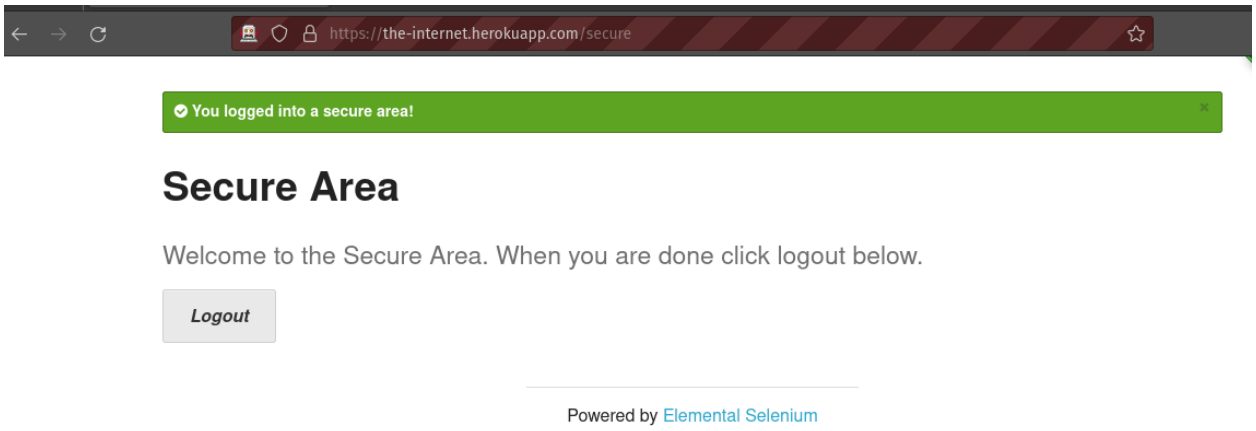
We can find the password field similarly by ID.

```
pass_field = browser.find_element(By.ID, 'password')
pass_field.click()
pass_field.send_keys("SuperSecretPassword!")
```

Then we need to find the login button.

```
submit_button = browser.find_element(By.CLASS_NAME, 'radius')
submit_button.click()
```

You'll be logged into the secure area!



Close the browser.

```
browser.close()
```

Activity 3: Dynamically Loaded Content

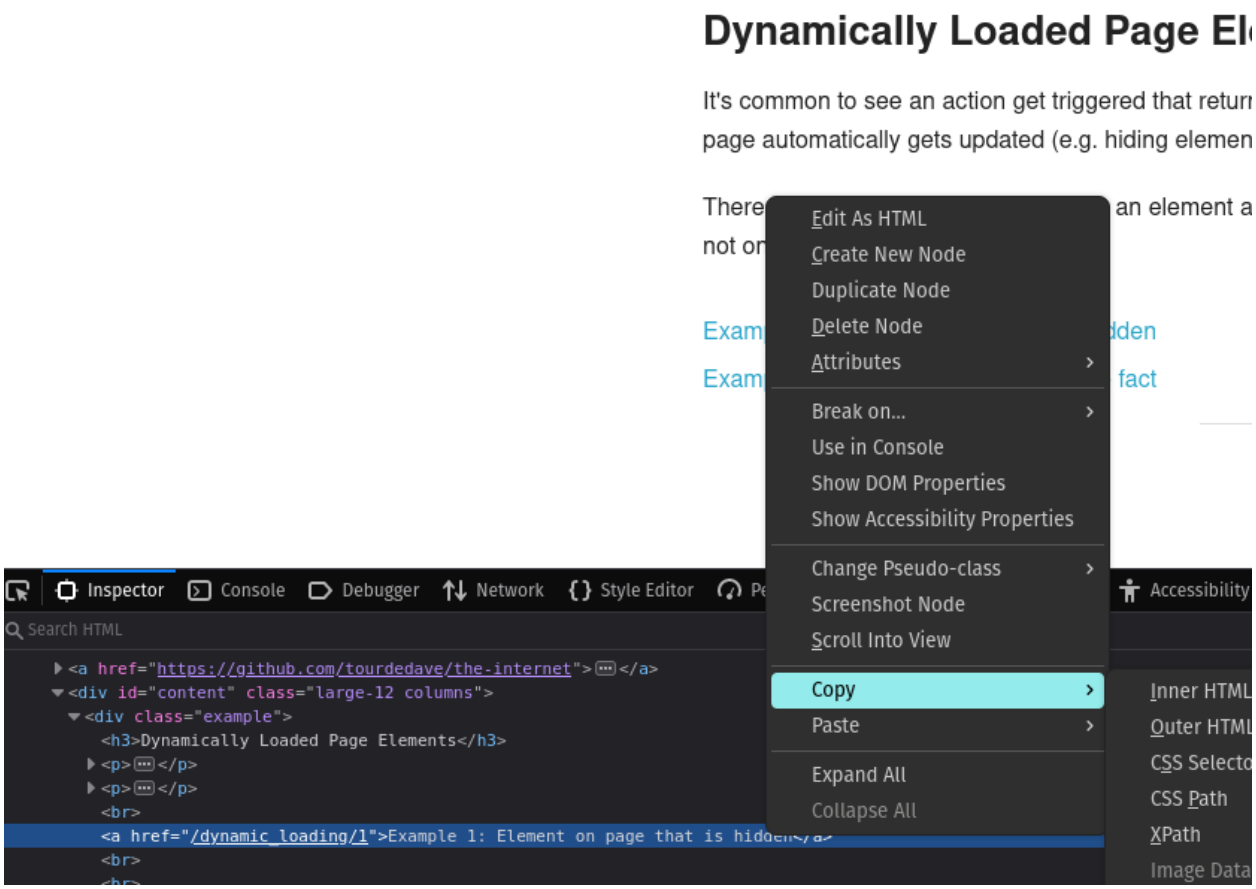
Many pages dynamically load content, meaning they aren't immediately available when you click on them. Think of when you scroll down on a Facebook feed and there is a short buffer while new content is served. It isn't instant. In this case we need to use waits.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```
browser = webdriver.Firefox()
browser.get('https://the-internet.herokuapp.com/dynamic_loading')
```

Until now, we have been referencing assets the page either by their **class name** or their **ID**. You can also reference items directly by their CSS Selector or something known as an XPATH. Both can be copied over right from the inspect element. You find the relavent element you want to click on and it's corresponding HTML and you right click -> Copy -> XPATH or CSS Selector. We can then use these to reference the objects on the page.

Here is a screenshot on how to do so for the Example 1 page.



The resulting XPATH is: **/html/body/div[2]/div/div/a[1]**

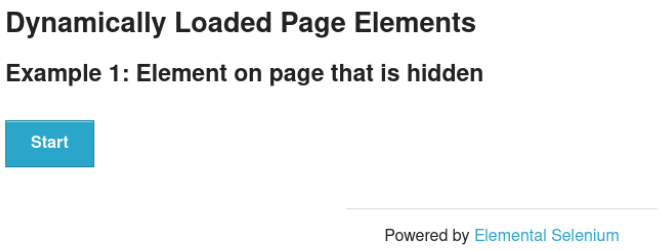
We can then use find_element with the XPATH like so:

```
example_1=browser.find_element(By.XPATH,
"/html/body/div[2]/div/div/a[1]")
```

and click on the element:

```
example_1.click()
```

which brings us to a new page:



Now we are ready to find the start button.
Let's find it by CSS Selector this time.
#start > button:nth-child(1) should be the selector.

Thus the following will access the element.

```
start_button = browser.find_element(By.CSS_SELECTOR, "#start > button:nth-child(1)")
start_button.click()
```

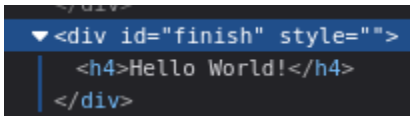
You'll notice there is a loading bar and delay until "Hello World" appears.
Let's navigate to the previous page.

```
 browser.back()
```

Go back to Example 1.

```
example_1=browser.find_element(By.XPATH, "/html/body/div[2]/div/div/a[1]")
example_1.click()
```

Now I want to click on the start button, wait until the element appears, and then retrieve the Hello World text as soon as it is visible. We can see the Hello World text is wrapped in a with an ID of "finish", which will help us isolate it.



This is where we introduce **WebDriverWait()** - which tells the browser - wait for a set period of time until some condition is met. If the condition isn't ever met, the program jumps into the **except** or **finally** branch depending on your program's logic. In this case,

we want to wait until the Hello World! text is visible. So our condition will be **visibility_of_element_located**

There are many other conditions, depending on what you are working on and what you need the browser to wait to do until it moves onto the next thing. By default, it will keep trying to find the visibility of the element ever 500 milliseconds until it is successful- so as soon as it finds Hello World! in the browser, our script will complete. It won't wait the entire 10 seconds. It also guarantees that the program will finish.

This is the **advantage** of using **WebDriverWait** over using a regular sleep function like *time.sleep()***.** For sleep, you'd have to time exactly to the page load speed, and if you sleep too little, the element won't be visible and your script will fail. If you load late, you've wasted time!

You can run this entire script and run it, which will print the Hello World! text that it captured from the browser to your console:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

browser = webdriver.Firefox()
browser.get('https://the-internet.herokuapp.com/dynamic_loading')

example_1=browser.find_element(By.XPATH, "/html/body/div[2]/div/div/a[1]")
example_1.click()

try:
     start_button = browser.find_element(By.CSS_SELECTOR, "#start >
button:nth-child(1)")
     start_button.click()
     element = WebDriverWait(browser,
10).until(EC.visibility_of_element_located((By.ID, "finish")))
     print(element.text)
     browser.close()
except:
     print("failed")
```

Activity 4: Scraping FOMB w/ Selenium

I've outlined some code to change the rows/per page to 100, grab the links to the 100 rows, and then click on the next page in a Python script.

We can probably **wrap the evens and odd code into one method- grab_links()** which combines the two for loops and that we can call the method whenever. If you're clever, you can probably find a way to make it one for loop to cut down on the amount of lines.

We also need to write a for loop for the program to keep clicking next until we reach the last page.

You could hard code how many times to click next (45 times), but if you are truly trying to automate the program, you will want to handle the case where more content is added to the site, and there is another page that gets added.

You will probably want something to capture the **text** from the last pagination link here, which tells you the final page.

46

You can base your upper limit of your range using that number.