Demo 1: Google Scholar Source Searching

In this demo, we are going to automate a Google Scholar search. The search will input whatever text you want to look for, and pull up new tabs of the first n search results, where you can choose n.

We will need to add one import in this case, shown down below.

```
from selenium.webdriver.common.keys import Keys
```

Keys will allow us to send non-string keys to text input boxes.

Now, let's create a new function called googleScholarSearch. It will take in a string argument of the text to be searched, and an integer argument that is the number of articles/links to be pulled up.

```
def googleScholarSearch(searchText, numberOfArticles):
```

In this demo, we need to (a) pull up Google Scholar, (b) input a string into the search bar and submit it using the enter key, (c) find the first <numberOfArticle> links in the page and parse through them to make sure they are articles, and (d) open new tabs and iterate through the article links to open all of them.

Pulling up Google Scholar is easy, and we'll maximize the Selenium window while we're at it.

```
driver.maximize_window()
driver.get("https://scholar.google.com/")
```

Now, we need to find the search bar element. We'll pull up the inspect tool for Google Scholar. We know that the search bar requires user input and that it is a text input, so we can CTRL + F search for an input tag. As we look through the <input> elements, remember that the type has to be "text" and that hovering over the element in the inspect tool will show which element it corresponds to on the page. We eventually find that the search bar element contains the following code:

```
<input type="text" class="gs_in_txt gs_in_ac" name="q" value=""
id="gs_hdr_tsi" size="50" maxlength="2048" autocapitalize="off"
autocomplete="off" aria-label="Search" dir="ltr">
```

The name of the element is “q” so we will use that to find the search bar element using Selenium.

```
searchBar = driver.find_element(By.NAME, “q”)
```

We now want to send and enter the text we want to search. Similar to the previous demo, we can do this by sending searchText as keys, then sending the “enter” key. However, the enter key cannot be sent using strings, which is where we will use Keys.

```
searchBar.send_keys(searchText)
searchBar.send_keys(Keys.RETURN)
```

Now, we need to find the first n articles that Google Scholar pulls up. We can do this by creating a list, then filling it with links. The <a> tag in HTML corresponds to hyperlink elements, so we can search by tag name to find the article hyperlink elements. The element itself will not be the link, so using get_attribute(“href”) we can find the link attached to the element. Additionally, not every hyperlink will lead to an article. Some may just lead back to Google Scholar and some may not include links, so we need to make sure that these are proper links before including them in our searchResults list. Using this code, we were able to loop through all links to find the actual informative ones.

```
for element in results:
     link = element.get_attribute("href")
    if (link != None and ("google" not in link or
    "books.google" in
        link and "https://" in link):
         articleLinks.append(link)
```

However, we can run into a problem. What if the number of links on the first search result page is less than the number of articles we want? To fix this, we’ll add more to the list searchResults. We have to automate moving to the next page of Google Scholar, so we will find the element that does this and click it whenever we need more articles. The type of this element in HTML should be <a>, but that doesn’t assist in looking for it. However, the button should have a link, so we can CTRL + F the inspect tool and search “href”. Again, hovering over each element will highlight the part of the webpage it corresponds to, so by going through each instance of “href” we eventually find the link to the “next” button. We will have to go into this subsection and copy the XPath of the <span> element. With this, we can now go to the next page of search results. Here’s the complete code for this section:

```
results= []
articleLinks = [ ]

nextPagePath =
'/html/body/div/div[10]/div[2]/div[3]/div[3]/div[2]/center/tabl
e/tbody/tr/td[12]/a/span'

while len(articleLinks) < numberOfArticles:
     results = driver.find_elements(By.TAG_NAME, "a")
     for element in results:
          link = element.get_attribute("href")
          if (link != None and("google" not in link or
          "books.google" in
              link) and "https://" in link):
                    articleLinks.append(link)
     nextButton = driver.find_element(By.XPATH, nextPagePath)
     nextButton.click()
```

Note: nextButton has to be called on every iteration because once the page changes, the element is no longer the same. The XPath for button still works because it’s in the same location, but if the location were to change with the new page, this code would NOT WORK.

Now, using what we have, we can start to open the search results. We will do this with a for loop, using numberOfArticles as a range. Each result will be opened in a new tab, so there will be new code here! driver.execute_script() will execute the JavaScript code that is put in as an argument, and driver.switch_to_windows(handle) and driver.window_handles(index) are described in the Basic Selenium Syntax section of this document. The following code is what we used:

```
for i in range(numberOfArticles):
     driver.execute_script("window.open('');")
     driver.switch_to.window(driver.window_handles[i + 1])
     driver.get(articleLinks[i])
```

The complete code for this demo is below.

```
def googleScholarSearch(searchText, numberOfArticles):
     driver.maximize_window()
     driver.get("https://scholar.google.com/")
```

```
          searchBar = driver.find_element(By.NAME, "q")

          searchBar.send_keys(searchText)
          searchBar.send_keys(Keys.RETURN)

          results = []
          articleLinks = [ ]

          nextPagePath =
          '/html/body/div/div[10]/div[2]/div[3]/div[3]/div[2]/center
          /table/tbody/tr/td[12]/a/span'

          while len(articleLinks) < numberOfArticles:
               results = driver.find_elements(By.TAG_NAME, "a")
               for element in results:
                    link = element.get_attribute("href")
                    if (link != None and
                         ("google" not in link or "books.google" in
link)
                         and "https://" in link):
                         articleLinks.append(link)
               nextButton = driver.find_element(By.XPATH,
nextPagePath)
               nextButton.click()

          for i in range(len(articleLinks)):
               driver.execute_script("window.open('');")
               driver.switch_to.window(driver.window_handles[i + 1])
               driver.get(articleLinks[i])

          return

     googleScholarSearch("your-text", <integer input>)
```

NOTE: if a link prompts downloading of a file, running this code will automatically download the file.

By running this code and modifying “your-text” and <integer input>, you can find scholarly articles on a topic of your choice!