SDJ INTERNATIONAL COLLEGE

Selenium Python
Unit 2

– By Tushar Sir

– By Tushar Sir

Selenium Python Introduction

- Selenium's Python Module is built to perform automated testing with Python.
- Selenium Python bindings provides a simple API to write functional/acceptance tests using Selenium WebDriver.
- Selenium Python bindings provide a convenient API to access Selenium WebDrivers like Firefox, Ie, Chrome, Remote etc.
- **Open Source and Portable :** Selenium is an open source and portable Web testing Framework.
- **Combination of tool and DSL :** Selenium is combination of tools and DSL (Domain Specific Language) in order to carry out various types of tests.
- **Easier to understand and implement :** Selenium commands are categorized in terms of different classes which make it easier to understand and implement.

– By Tushar Sir

Selenium Python Introduction

- **Reduce test execution time :** Selenium supports parallel test execution that reduce the time taken in executing parallel tests.
- **Lesser resources required :** Selenium requires lesser resources when compared to its competitors like UFT, RFT, etc.
- **Supports Multiple Operating Systems :** Android, iOS, Windows, Linux, Mac, Solaris.
- **Supports Multiple Browsers :** Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari, etc.
- **Parallel Test Execution :** It also supports parallel test execution which reduces time and increases the efficiency of tests.

– By Tushar Sir

Why is Python useful for automation testing?

- Python is very useful for automation testing because it supports multiple programming patterns.
- Python has many built-in testing frameworks such as Pytest and Robot, which covers the debugging and faster workflow.
- It is an interpreted language means the interpreter implements the code line by line at a time that's makes debugging easy.
- Python is Cross-platform Language; that's why it can run on different platforms like Windows, Linux, UNIX, and Macintosh,
- Python can be easily implemented with other programming languages such as C, C++, JAVA, etc.

– By Tushar Sir

Selenium Python Installation

- For any operating system selenium can be installed after you have installed python on your operating system. (We assume that latest version of Python is already downloaded).
- Once we successfully install the Python in our operation system, we will install the Selenium libraries.
- For this, we will execute the following command in our command prompt:

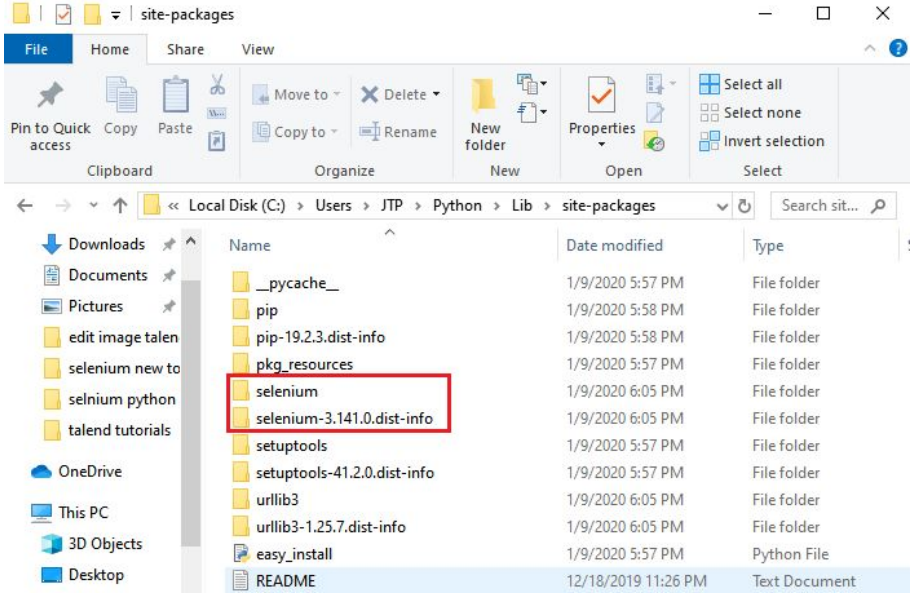
  Python -m pip install -U Selenium

- And, this command will successfully install the latest Selenium package.

– By Tushar Sir

Selenium Python Installation

- After that executing the above command, it will create the Selenium folder automatically having all the Selenium libraries as we can see in the below screenshot:



– By Tushar Sir

Selenium Python Installation

Installing Webdrivers

- One Can Install Firefox, Chromium, PhantomJs(Deprecated Now), etc.
- for using Firefox you may need to install **GeckoDriver**
- for using Chrome you may need to install **Chromium**

Why Use a ChromeDriver?

- As Google Chrome dominates the browser market, the use of a ChromeDriver becomes a must.
- Selenium WebDriver uses the ChromeDriver to communicate test scripts with Google Chrome.
- It is used to navigate between web pages and provide input to the same.

– By Tushar Sir

Selenium Python Installation

Chromium

- Chromium is the engine behind Google's Chrome web browser. It is very common now to use Chromium to drive the user-interface for desktop applications.
- The Chromium projects include Chromium and Chromium OS, the open-source projects behind the Google Chrome browser and Google Chrome OS, respectively.
- This site houses the documentation and code related to the Chromium projects and is intended for developers interested in learning about and contributing to the open-source projects.
- Chromium is an open-source browser project that aims to build a safer, faster, and more stable way for all users to experience the web.
- This site contains design documents, architecture overviews, testing information, and more to help you learn to build and work with the Chromium source code.

– By Tushar Sir

Selenium Python Installation

Can I use selenium with Chromium?

- To execute tests in the Chrome browser using Selenium WebDriver, you need to use ChromeDriver.
- ChromeDriver is a separate executable used by Selenium WebDriver to control the Chrome browser.
- Chrome tracks user browsing data for personalized ads. Users can use “Incognito Mode” to prevent saving browsing history, cookies, and form data.
- Chromium does not include user tracking, making it more privacy-centric. However, it lacks personalized features based on browsing data.

– By Tushar Sir

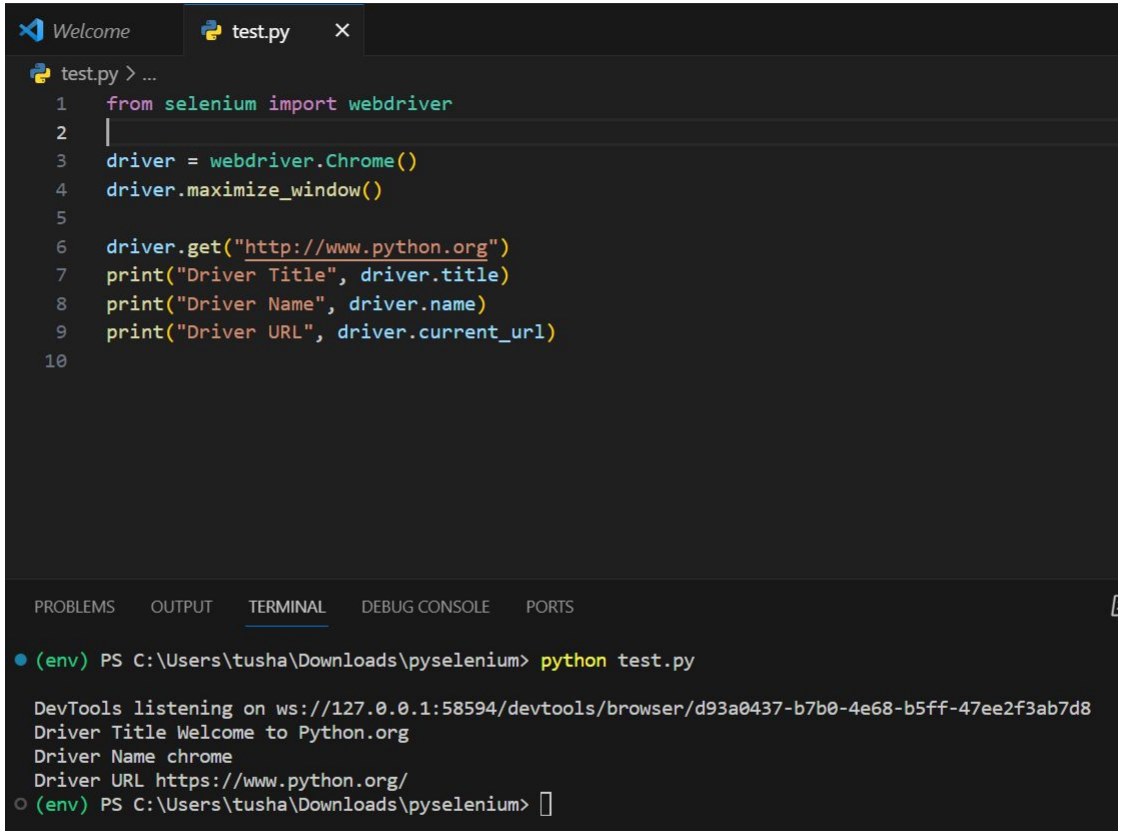
Selenium Python Installation

- **Chrome:** https://sites.google.com/chromium.org/driver/
- **Edge:** https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
- **Firefox:** https://github.com/mozilla/geckodriver/releases
- **Safari:** https://webkit.org/blog/6900/webdriver-support-in-safari-10/

– By Tushar Sir

Getting Started with Selenium Python

Simple Program (Test Script)



– By Tushar Sir

Navigating links using get method – Selenium Python

How to navigate links using Python Selenium?

- The first thing you'll want to do with WebDriver is navigate to a link. The normal way to do this is by calling get method:
- **Syntax:** *driver.get(url)*
- **Example:** *driver.get("http://www.google.com")*
- WebDriver will wait until the page has fully loaded (that is, the onload event has fired) before returning control to your test or script.
- It’s worth noting that if your page uses a lot of AJAX on load then WebDriver may not know when it has completely loaded.
- If you need to ensure such pages are fully loaded then you can use waits.

– By Tushar Sir

Navigating links using get method – Selenium Python

test.py > ...

```
1  from selenium import webdriver
2
3  driver = webdriver.Chrome()
4
5  # get google.co.in
6  driver.get("https://google.com/")
7
```

test.py > ...

```
1  from selenium import webdriver
2
3  driver = webdriver.Chrome()
4
5  # get google.co.in
6  driver.get("https://google.com/search?q=sdjic")
7
```

– By Tushar Sir

Navigating links using get method – Selenium Python

test.py > ...

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

driver = webdriver.Chrome()
driver.get("https://www.google.com/")
#identify the Google search text box and enter the value
driver.find_element('name',"q").send_keys("sdjic")
time.sleep(3)
#click on the Google search button
driver.find_element('name',"btnK").send_keys(Keys.ENTER)
time.sleep(3)
#close the browser
driver.close()
print("sample test case successfully completed")
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

```
(env) PS C:\Users\tusha\Downloads\pyselenium> python test.py

DevTools listening on ws://127.0.0.1:56563/devtools/browser/2e54e
[8452:2764:1003/194344.727:ERROR:ssl_client_socket_impl.cc(968)]
sample test case successfully completed
```

– By Tushar Sir

Navigating links using get method – Selenium Python

- **Step 1: -** In the first step, we will type the following statement to import the web driver.
- **Step 2: -** After that, we will open the Google Chrome browser. As we can see in the below screenshot, we have multiple types of browsers options available, and we can select any browser from the list like Chrome, Edge, firefox, Internet Explorer, opera, safari, etc. Following are the sample code for opening the Google Chrome browser:

  driver = webdriver.Chrome()

- **Step 3: -** In the next step, we will be maximizing our browser window size, and the sample code is as below:

  driver.maximize_window()

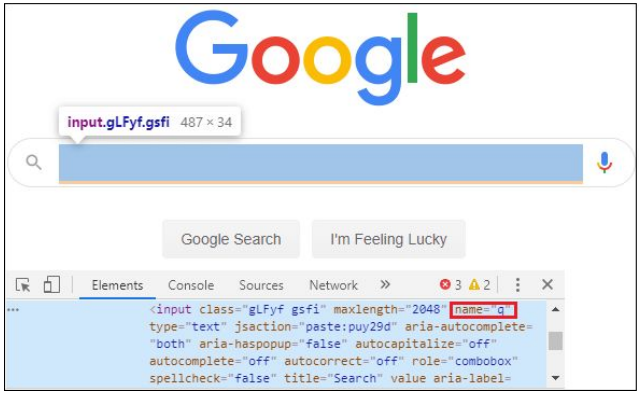
- **Step 4: -** Then, we will navigate to the given URL. The sample code is as below:

  driver.get("https://www.google.com/")

– By Tushar Sir

Navigating links using get method – Selenium Python

- **Step 5: -** In this step, we are trying to locate the Google search text box with the help of its Name attribute value. Right-click on the Google search text box, and select the Inspect option in the pop-up menu.
- The developer tool window will be launched with all the specific codes used in the development of the Google search text box. And, copy the value of its Name attribute, that is "q" as we can see in the below image:



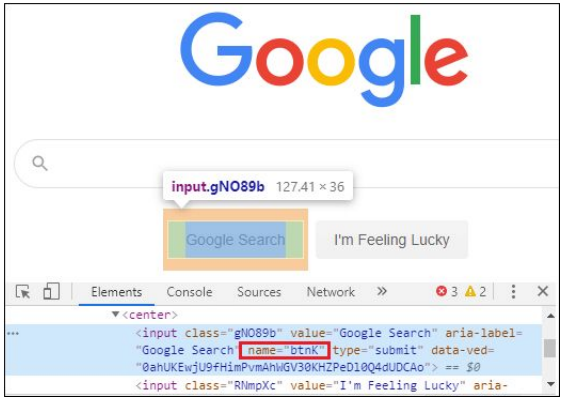
- Here the sample code:

driver.find_element_by_name("q").send_keys("sdjic")

– By Tushar Sir

Navigating links using get method – Selenium Python

- **Step 6: -** Once we identify the Google search text box, and we will identify the Google Search button. So for this, follow the below process:
- Right-click on the Google searchbutton, and select the Inspect option from the given pop-up menu. The developer tool window will be launched with having all the specific codes used in the development of the Google search button. Then, copy the value of its name attribute that is "btnK" as we can see in the below image:



- And, the sample code is as following:

driver.find_element_by_name("btnK").send_keys(Keys.ENTER)

– By Tushar Sir

Navigating links using get method – Selenium Python

- **Step 7: -** In the last step, we are closing the browser. And, the sample code for closing the browser is as follows:

  driver.close()

Note: -

Old API	New API
find_element_by_id(‘id’)	find_element(By.ID, ‘id’)
find_element_by_name(‘name’)	find_element(By.NAME, ‘name’)
find_element_by_xpath(‘xpath’)	find_element(By.XPATH, ‘xpath’)

– By Tushar Sir

Navigating links using get method – Selenium Python

test.py > ...

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

driver = webdriver.Chrome()
driver.get("https://www.python.org")
print(driver.title)
search_bar = driver.find_element('name',"q")
search_bar.clear()
search_bar.send_keys("getting started with python")
search_bar.send_keys(Keys.RETURN)
print(driver.current_url)
driver.close()
```

– By Tushar Sir

Navigating links using get method – Selenium Python

test.py > ...

```
1   from selenium import webdriver
2   from selenium.webdriver.common.keys import Keys
3   from selenium.webdriver.common.by import By
4
5   driver = webdriver.Chrome()
6   driver.get("http://www.python.org")
7   assert "Python" in driver.title
8   elem = driver.find_element(By.NAME, "q")
9   elem.clear()
10  elem.send_keys("pycon")
11  elem.send_keys(Keys.RETURN)
12  assert "No results found." not in driver.page_source
13  driver.close()
```

– By Tushar Sir

Using Selenium to write tests

- Selenium is mostly used for writing test cases. The selenium package itself doesn't provide a testing tool/framework.
- You can write test cases using Python unittest module.
- The other options for a tool/framework are **pytest** and **nose**. We use unittest as the framework of choice.

– By Tushar Sir

Using Selenium to write tests

test.py > ...

```python
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By

class PythonOrgSearch(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()

    def test_search_in_python_org(self):
        driver = self.driver
        driver.get("http://www.python.org")
        self.assertIn("Python", driver.title)
        elem = driver.find_element(By.NAME, "q")
        elem.send_keys("pycon")
        elem.send_keys(Keys.RETURN)
        self.assertNotIn("No results found.", driver.page_source)

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    unittest.main()
```

– By Tushar Sir

Interacting with Webpage – Selenium Python

- Just being able to go to places isn't terribly useful. What we'd really like to do is to interact with the pages, or, more specifically, the HTML elements within a page.
- First of all, we need to find one. WebDriver offers a number of ways to find elements.
- For example, given an element defined as:

  <input type="text" name="passwd" id="passwd-id" />

- To find an element one needs to use one of the locating strategies, For example: -

  element = driver.find_element(By.ID,"passwd-id")

  element = driver.find_element(By.NAME, "passwd")

  element = driver.find_element(By.XPATH, "//input[@id='passwd-id']")

– By Tushar Sir

Interacting with Webpage – Selenium Python

- Also, to find multiple elements, we can use :

  elements = driver.find_elements(By.NAME, "passwd")

- One can also look for a link by its text, but be careful! The text must be an exact match! One should also be careful when using XPATH in WebDriver.
- If there's more than one element that matches the query, then only the first will be returned. If nothing can be found, a NoSuchElementException will be raised.
- WebDriver has an "Object-based" API, we represent all types of elements using the same interface.
- This means that although one may see a lot of possible methods one could invoke when one hits IDE's auto-complete key combination, not all of them will make sense or be valid.

– By Tushar Sir

Interacting with Webpage – Selenium Python

- So after getting an element what next? One might want to enter text into a field, for example:

  element.send_keys("some text")

- One can simulate pressing the arrow keys by using the “Keys” class:

  element.send_keys(" and some", Keys.ARROW_DOWN)

- Also note, that it is possible to call send_keys on any element, which makes it possible to test keyboard shortcuts such as those used on Gmail. One can easily clear the contents of a text field or textarea with the clear method:

  element.clear()

– By Tushar Sir

Interacting with Webpage – Selenium Python

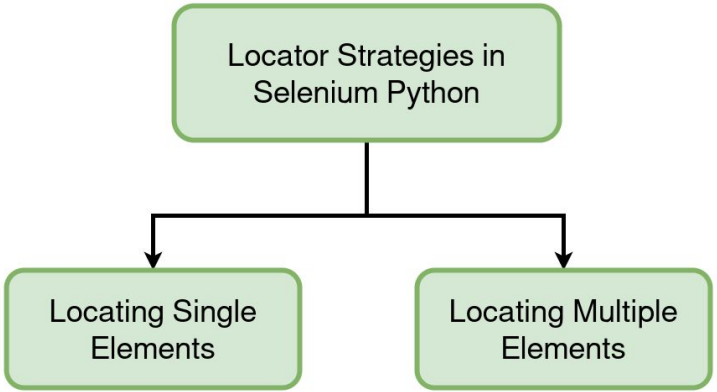
test.py > ...

```python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By

# Provide the path to the Chrome WebDriver executable
driver = webdriver.Chrome()

# Open the Gmail login page
driver.get("https://mail.google.com/")

# Locate the email input field and enter your email address
email_input = driver.find_element(By.NAME,"identifier")
email_input.send_keys("your_email@gmail.com")
email_input.send_keys(Keys.RETURN)
# Wait for the password input field to load
password_input = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.NAME, "password"))
)
# Enter your password and submit the form
password_input.send_keys("your_password")
password_input.send_keys(Keys.RETURN)
driver.quit()
```

– By Tushar Sir

Locator Strategies – Selenium Python

- Locators Strategies in Selenium Python are methods that are used to locate elements from the page and perform an operation on the same.
- Selenium's Python Module is built to perform automated testing with Python.
- Selenium Python bindings provides a simple API to write functional/acceptance tests using Selenium WebDriver.
- After opening a page using selenium such as geeksforgeeks, one might want to click some buttons automatically or fill a form automatically or any such automated task.



– By Tushar Sir

Locator Strategies to locate single first elements

Locators	Description
By.ID	The first element with the id attribute value matching the location will be returned.
By.NAME	The first element with the name attribute value matching the location will be returned.
By.XPATH	The first element with the xpath syntax matching the location will be returned.

– By Tushar Sir

Locator Strategies to locate single first elements

By.LINK_TEXT	The first element with the link text value matching the location will be returned.
By.PARTIAL_LINK_TEXT	The first element with the partial link text value matching the location will be returned.
By.TAG_NAME	The first element with the given tag name will be returned.
By.CLASS_NAME	the first element with the matching class attribute name will be returned.
By.CSS_SELECTOR	The first element with the matching CSS selector will be returned.

– By Tushar Sir

Locator Strategies to locate multiple elements

Locators	Description
find_elements(By.NAME, “name”)	All elements with name attribute value matching the location will be returned.
find_elements(By.XPATH, “xpath”)	All elements with xpath syntax matching the location will be returned.
find_elements(By.LINK_TEXT, “link text”)	All elements with link text value matching the location will be returned.
find_elements(By.PARTIAL_LINK_TEXT, “partial link text”)	All elements with partial link text value matching the location will be returned.

– By Tushar Sir

Locator Strategies to locate multiple elements

find_elements(By.TAG_NAME, “tag name”)	All elements with given tag name will be returned.
find_elements(By.CLASS_NAME, “class name”)	All elements with matching class attribute name will be returned.
find_elements(By.CSS_SELECTOR, “css selector”)	All elements with matching CSS selector will be returned.

– By Tushar Sir

Locating Single and Multiple elements

find_element_by_id (By.ID):

- With this strategy, the first element with the id attribute value matching the location will be returned. If no element has a matching id attribute, a NoSuchElementException will be raised.
- **Syntax:**

  driver.find_element(By.ID, "id_of_element")

```
<html>
<body>
<form id="loginForm">
<input name="username" type="text" />
<input name="password" type="password" />
<input name="continue" type="submit" value="Login" />
</form>
</body>
<html>
```

– By Tushar Sir

Locating Single and Multiple elements

find_element_by_id (By.ID):

- Now after you have created a driver, you can grab an element using –

  login_form = driver.find_element(By.ID, 'loginForm')

Find_element_by_name (By.NAME) :

- With this strategy, the first element with the name attribute value matching the location will be returned. If no element has a matching name attribute, a NoSuchElementException will be raised.
- **Syntax:**

  driver.find_element(By.NAME, "name_of_element")

– By Tushar Sir

Locating Single and Multiple elements

- **Example:**

```
<html>
<body>
<form id="loginForm">
<input name="username" type="text" />
<input name="password" type="password" />
<input name="continue" type="submit" value="Login" />
</form>
</body>
<html>
```

- Now after you have created a driver, you can grab an element using –

  element = driver.find_element(By.NAME, 'username')

– By Tushar Sir

Locating Single and Multiple elements

find_element_by_xpath (By.XPATH) :

- With this strategy, the first element with pattern of xpath matching the location will be returned. If no element has a matching element attribute, a NoSuchElementException will be raised.
- **Syntax:**

  driver.find_element(By.XPATH, "xpath")

- Now after you have created a driver, you can grab an element using:

  login_form = driver.find_element(By.XPATH, "/html/body/form[1]")

  login_form = driver.find_element(By.XPATH, "//form[1]")

– By Tushar Sir

Locating Single and Multiple elements

find_element_by_tag_name (By.TAG_NAME) :

- With this strategy, the first element with the given tag name will be returned. If no element has a matching tag name, a NoSuchElementException will be raised.
- **Syntax:**

  driver.find_element(By.TAG_NAME, "Tag name")

- Now after you have created a driver, you can grab an element using –

  login_form = driver.find_element(By.TAG_NAME, 'h1')

– By Tushar Sir

Locating Single and Multiple elements

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


def indeed_job_search():
    browser = webdriver.Chrome()
    browser.get('https://in.indeed.com/')
    browser.implicitly_wait(5)
    search_bar = browser.find_element('id','text-input-what')
    search_bar.send_keys('Web Developer')
    search_bar.send_keys(Keys.ENTER)

    browser.implicitly_wait(5)

if __name__ == "__main__":
    indeed_job_search()
```

– By Tushar Sir

Locating Single and Multiple elements

```python
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By

def indeed_job_search():
    browser = webdriver.Chrome()
    browser.get('https://in.indeed.com/')
    browser.implicitly_wait(2)
    search_bar = browser.find_element('id','text-input-what')
    search_bar.send_keys('Web Developer')
    search_bar.send_keys(Keys.RETURN)
    browser.implicitly_wait(2)

    search_results = browser.find_elements(By.XPATH,'//h2/a')

    file = open("job_search.txt", 'a')
    file.write("\n")

    for job_element in search_results:
        job_title = job_element.text
        job_link = job_element.get_attribute('href')
        file.write("%s | link: %s \n" %(job_title, job_link))

    browser.close()

if __name__ == "__main__":
    indeed_job_search()
```

– By Tushar Sir

Locating Single and Multiple elements

```
# Scrape tables
from selenium import webdriver
import time
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
table_url = "https://iqssdss2020.pythonanywhere.com/tutorial/default/dynamic"
driver.get(table_url)
time.sleep(2)

file = open('scrape_tables.csv', "w", encoding = "utf-8")
table_body = driver.find_element(By.XPATH,'//*[@id="result"]/table/tbody')
entries = table_body.find_elements(By.TAG_NAME,'tr')

headers = entries[0].find_elements(By.TAG_NAME,'th')

table_header = ''
for i in range(len(headers)):
    header = headers[i].text
    if i == len(headers) - 1:
        table_header = table_header + header + "\n"
    else:
        table_header = table_header + header + ","
file.write(table_header)
```

```
for i in range(1, len(entries)):
    cols = entries[i].find_elements(By.TAG_NAME,'td')
    table_row = ''
    for j in range(len(cols)):
        col = cols[j].text
        if j == len(cols) - 1:
            table_row = table_row + col + "\n"
        else:
            table_row = table_row + col + ","
    file.write(table_row)


driver.close()
file.close()
```

– By Tushar Sir

Locating Single and Multiple elements

```
# Scrape Text
from selenium import webdriver
import time
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
journalAddress = "https://www.federalregister.gov/documents/2013/09/24/2013-21228/affirmative-action-and
driver.get(journalAddress)
time.sleep(2)

articleObjects = driver.find_elements(By.XPATH,'//div[@id="fulltext_content_area"]/*')

articleDictionary = dict()
myKey = ""
myValue_total = ""
```

– By Tushar Sir

Locating Single and Multiple elements

```
for i in range(len(articleObjects)):
    tagName = articleObjects[i].tag_name
    if tagName.startswith("h"):
        if myKey:
            articleDictionary[myKey] = myValue_total
            myKey = ""
            myValue_total = ""
        myKey = articleObjects[i].get_attribute("innerText")
    if tagName.startswith("p"):
        myValue = articleObjects[i].get_attribute("innerText")
        myValue_total = myValue_total + myValue
    if tagName.startswith("ul"):
        myBullets = articleObjects[i].find_elements(By.XPATH,'li')
        for j in range(len(myBullets)):
            myBullet = myBullets[j].get_attribute("innerText")
            myValue_total = myValue_total + myBullet
driver.close()

article = ''
for key, value in articleDictionary.items():
    article = article + key + '\n\n' + value + '\n\n***************\n\n'
print(article)
```

– By Tushar Sir

Filling the Web Forms

```python
# Filling the Web Form
from selenium import webdriver
from selenium.webdriver.support.select import Select
from selenium.webdriver.common.by import By
import time

driver = webdriver.Chrome()
form_url = "https://iqssdss2020.pythonanywhere.com/tutorial/form/search"
driver.get(form_url)

driver.find_element(By.ID,'search_name').send_keys("A")
time.sleep(2)

Select(driver.find_element(By.ID,"search_grade")).select_by_visible_text("5")
time.sleep(2)

driver.find_element(By.ID,'p5').click()
time.sleep(2)

driver.find_element(By.ID,"privacypolicy").click()
driver.find_element(By.ID,"termsconditions").click()

driver.find_element(By.ID,"search").click()
time.sleep(2)

driver.close()
```

– By Tushar Sir

Create an Action Chain Object

- Selenium's Python Module is built to perform automated testing with Python.
- ActionChains are a way to automate low-level interactions such as mouse movements, mouse button actions, keypress, and context menu interactions.
- This is useful for doing more complex actions like hover over and drag and drop.
- Action chain methods are used by advanced scripts where we need to drag an element, click an element.
- ActionChains are implemented with the help of a action chain object which stores the actions in a queue and when perform() is called, performs the queued operations.
- Action chains in selenium python are the execution of multiple browser actions together in sequence.
- Selenium can chain multiple browser actions together and this chaining of multiple actions is known as Action chains.

– By Tushar Sir

What are Action Chains in Selenium Python?

- Action chains are a sequence of actions that are performed in a specific order on a web page to test for a specific outcome.
- These actions can be anything like clicking on an element, keypress, entering text, scrolling, dragging and dropping an object, etc.
- These actions are basically DOM manipulation which is done using Action Chains in Selenium Python.

Example:

- Let’s consider an example of a User who wants to log in to his account and navigate to a specific page on the webpage. For this, the user needs to first enter his credentials into the login page and then click on the “Login” button and then navigate to the desired page. These actions can be automated using Action chains in selenium python by initiating the required action in the correct order and executing the actions in sequence.

– By Tushar Sir

The Syntax for Creating Action Chain Object

- For creating an action chain object you have to **import** the ActionChains class from *selenium.webdriver.common.action_chains* module create an instance of that class.
- **Syntax:**

  from selenium.webdriver.common.action_chains import ActionChain

- The selenium.webdriver.common.action_chains imports the action chain module from the Action chain library of Python.

  driver = webdriver.Chrome()

– By Tushar Sir

The Syntax for Creating Action Chain Object

- The **webdriver.Chrome()** creates a webdriver object which can be used to perform browser actions when required.

  Action = ActionChains(driver)

- The **ActionChains(driver)** creates an action chain object using the driver object which is used to perform action chain execution.

– By Tushar Sir

How to use Action Chains in Selenium Python

- Action chain objects can now be used to perform multiple actions in a sequence to perform a specific task. The perform() function of ActionChains class can be used to perform the actions in a sequence.

```
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Chrome()

driver.get("https://www.browserstack.com/")

element = driver.find_element("link text","Sign in")

action = ActionChains(driver)

action.click(on_element = element)

action.perform()
```

– By Tushar Sir

How to use Action Chains in Selenium Python

```python
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

# import Action chains
from selenium.webdriver.common.action_chains import ActionChains

# create webdriver object
driver = webdriver.Chrome()
driver.get("https://www.geeksforgeeks.org/")

# get element
element = driver.find_element(By.LINK_TEXT,"Data Science")
time.sleep(3)

# create action chain object
action = ActionChains(driver)

# click the item
action.click(on_element = element)

# perform the operation
action.perform()
```

– By Tushar Sir

Action Chain Methods in Selenium Python

Method	Description
click	Clicks an element.
click_and_hold	Holds down the left mouse button on an element.
context_click	Performs a context-click (right click) on an element.
double_click	Double-clicks an element.
drag_and_drop	Holds down the left mouse button on the source element, then moves to the target element and releases the mouse button.

– By Tushar Sir

Action Chain Methods in Selenium Python

drag_and_drop_by_offset	Holds down the left mouse button on the source element, then moves to the target offset and releases the mouse button.
key_down	Sends a key press only, without releasing it.
key_up	Releases a modifier key.
move_by_offset	Moving the mouse to an offset from current mouse position.
move_to_element	Moving the mouse to the middle of an element.
move_to_element_with_offset	Move the mouse by an offset of the specified element, Offsets are relative to the top-left corner of the element.
perform	Performs all stored actions.

– By Tushar Sir

Action Chain Methods in Selenium Python

pause	Pause all inputs for the specified duration in seconds
release	Releasing a held mouse button on an element.
reset_actions	Clears actions that are already stored locally and on the remote end
send_keys	Sends keys to current focused element.

– By Tushar Sir

click method method

- **Syntax –**

  click(on_element=None)

- **Args –**

  on_element – The element to click. If None, clicks on current mouse position.

- **Example –**

  <input type ="text" name ="passwd" id ="passwd-id" />

- To find an element one needs to use one of the locating strategies, For example,

  element = driver.find_element_by_id("passwd-id")

- Now one can use click method as an Action chain as below –

  click(on_element=element)

– By Tushar Sir

click method method

```python
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

# create webdriver object
driver = webdriver.Chrome()
driver.get("https://internshala.com")

# get element
element = driver.find_element("link text","Hire Talent")

# create action chain object
action = ActionChains(driver)

# click the item
action.click(on_element = element)

# perform the operation
action.perform()
```

– By Tushar Sir

click_and_hold method

- **Syntax –**

  click_and_hold(on_element=None)

- **Args –**

  on_element – The element to click. If None, clicks on current mouse position.

- **Example –**

  <input type ="text" name ="passwd" id ="passwd-id" />

- To find an element one needs to use one of the locating strategies, For example,

  element = driver.find_element_by_id("passwd-id")

- Now one can use click method as an Action chain as below –

  click_and_hold(on_element=element)

– By Tushar Sir

Click_and_hold method

```
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

# create webdriver object
driver = webdriver.Chrome()
driver.get("https://www.geeksforgeeks.org/")

# get element
element = driver.find_element("link text","Data Science")

# create action chain object
action = ActionChains(driver)

# click and hold  the item
action.click_and_hold(on_element = element)

# perform the operation
action.perform()
```

– By Tushar Sir

double_click method

- **double_click** method is used to double click on an element or current position.
- **Syntax –**

  double_click(on_element=None)

- **Example –**

  <input type ="text" name ="passwd" id ="passwd-id" />

- To find an element one needs to use one of the locating strategies, For example,

  element = driver.find_element_by_id("passwd-id")

- Now one can use click method as an Action chain as below –

  double_click(on_element=element)

– By Tushar Sir

double_click method

```
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

# create webdriver object
driver = webdriver.Chrome()
driver.get("https://www.geeksforgeeks.org/")

# get element
element = driver.find_element("link text","GATE: DS & AI")

# create action chain object
action = ActionChains(driver)

# click and hold  the item
action.double_click(on_element = element)

# perform the operation
action.perform()
```

– By Tushar Sir

drag_and_drop method

- **drag_and_drop** method holds down the left mouse button on the source element, then moves to the target element and releases the mouse button.
- **Syntax –**

  drag_and_drop(source, target)

- **Args –**

  source: The element to mouse down.

  target: The element to mouse up.

- Now one can use click method as an Action chain as below –

  drag_and_drop(source, target)

– By Tushar Sir

drag_and_drop method

```
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

# create webdriver object
driver = webdriver.Chrome()
driver.get("https://www.geeksforgeeks.org/")

# get source element
source_element = driver.find_element("link text","Python")

# get target element
target_element = driver.find_element("link text","Algorithms")

# create action chain object
action = ActionChains(driver)

# drag and drop the item
action.drag_and_drop(source_element, target_element)

# perform the operation
action.perform()
```

– By Tushar Sir

key_down method

- **key_down** method is used to send a key press, without releasing it. this method is used in case one wants to press, ctrl+c, or ctrl+v. For this purpose one needs to first hold the ctrl key down and then press c. This method automates this work. It should only be used with modifier keys (Control, Alt and Shift).
- **Syntax –**

  key_down(value, element=None)

- **Args –**

  value: The modifier key to send. Values are defined in Keys class.

  element: The element to send keys. If None, sends a key to current focused element.

– By Tushar Sir

key_down method

- **Example –**
- One can use key_down method as an Action chain as below. This example clicks Ctrl+C after opening the webpage

  ActionChains(driver).key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).perform()

– By Tushar Sir

key_up method

- **key_up** method is used to release a pressed key using key_down method.
- **Syntax –**

  key_up(value, element=None)

- **Args –**

  value: The modifier key to send. Values are defined in Keys class.

  element: The element to send keys. If None, sends a key to current focused element.

- **Example –**
- One can use key_up method as an Action chain as below. This example clicks Ctrl+C after opening the webpage and key_up method releases the pressed key later.

– By Tushar Sir

key_down method

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

driver = webdriver.Chrome()
driver.maximize_window()

driver.get("https://www.geeksforgeeks.org")
#to refresh the browser
driver.refresh()

# action chain object creation
a = ActionChains(driver)

# perform the ctrl+c pressing action
a.key_down(Keys.CONTROL).send_keys('C').key_up(Keys.CONTROL).perform()
#to close the browser
driver.close()
```

– By Tushar Sir

perform method

- **perform** method is used to perform all stored operations in action instance of ActionChains class.
- **Syntax–**

  perform()

- **Example -**

  action.click(on_element=element)

  action.perform()

– By Tushar Sir

Pause method

- **pause** method is used to pause all inputs for the specified duration in seconds.
- Pause method is highly important and useful in case one is executing some command that takes some javascript to load or a similar situation where there is a time gap between two operations.
- **Syntax–**

  pause(seconds)

- **Args –**

  seconds – Number of seconds to pause for.

- **Example –**
- One can use pause method as an Action chain as below –

  pause(1024)

– By Tushar Sir

Pause method

```
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.by import By
# import Action chains
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Firefox()

# get geeksforgeeks.org
driver.get("https://www.geeksforgeeks.org/")

# get element
element = driver.find_element(By.LINK_TEXT,"Python")

# create action chain object
action = ActionChains(driver)

# click the item
action.click(on_element = element)

action.pause(1000)

# click the item
action.click(on_element = element)

# perform the operation
action.perform()
```

– By Tushar Sir

release method

- **release** method is used for releasing a held mouse button on an element.
- **Syntax–**

  release(on_element=None)

- **Args –**

  on_element – The element to mouse up. If None, releases on current mouse position.

- **Example –**

  action.click(on_element=element)

  action.release(element)

– By Tushar Sir

release method

```
# import webdriver
from selenium import webdriver
from selenium.webdriver.common.by import By
# import Action chains
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Firefox()

# get geeksforgeeks.org
driver.get("https://www.geeksforgeeks.org/")

# get element
element = driver.find_element(By.LINK_TEXT,"Python")

# create action chain object
action = ActionChains(driver)

# click the item
action.click(on_element = element)

# release the item
action.release(on_element = element)

# perform the operation
action.perform()
```

– By Tushar Sir

Thank You

- **By Tushar Sir**