Lab Report

By: Jahmil Ally (501045419)

Observations and Analysis

Q1:

Output:

I'm in method E // is executed first because it has the highest priority (lowest number: `0`).

I'm in method A // has the same priority as D but they are executed in the order they are declared.

I'm in method D

I'm in method B //has no priority like C and are executed last in the order they are declared.

I'm in method C

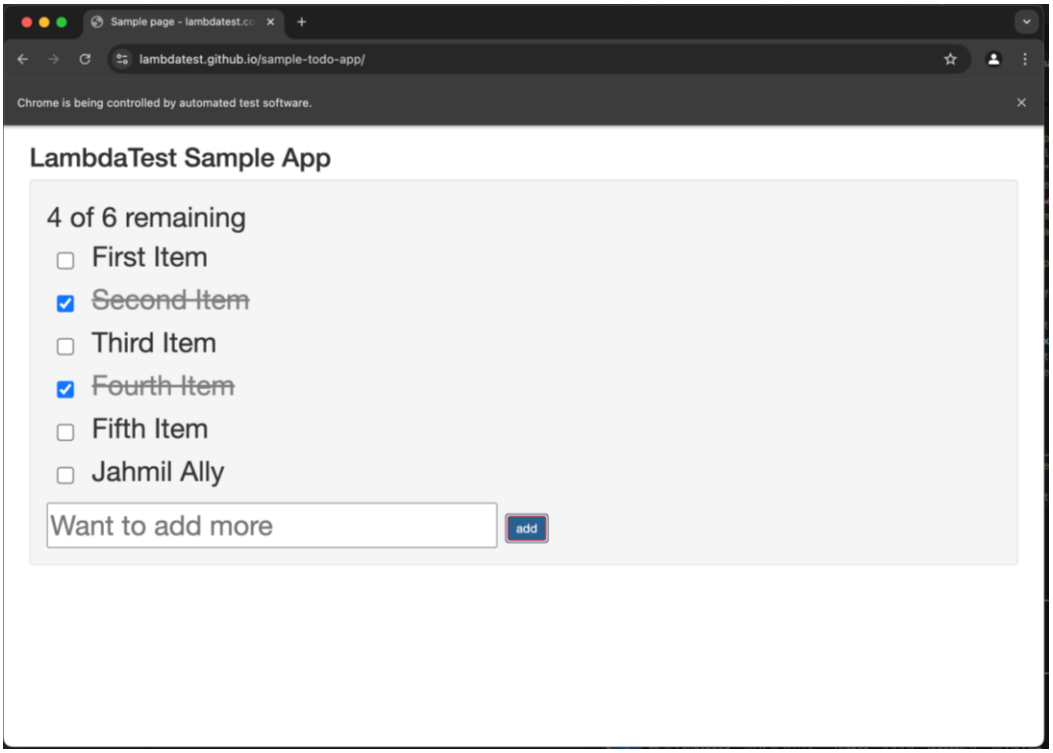
Q2:

```
jah@Jahmils-MacBook-Pro lab3 % python3 Q2.py
Page Title: Welcome: Mercury Tours
Title matches the expected value!
.
----------------------------------------------------------------------
Ran 1 test in 4.525s

OK
jah@Jahmils-MacBook-Pro lab3 %
```

Q3:

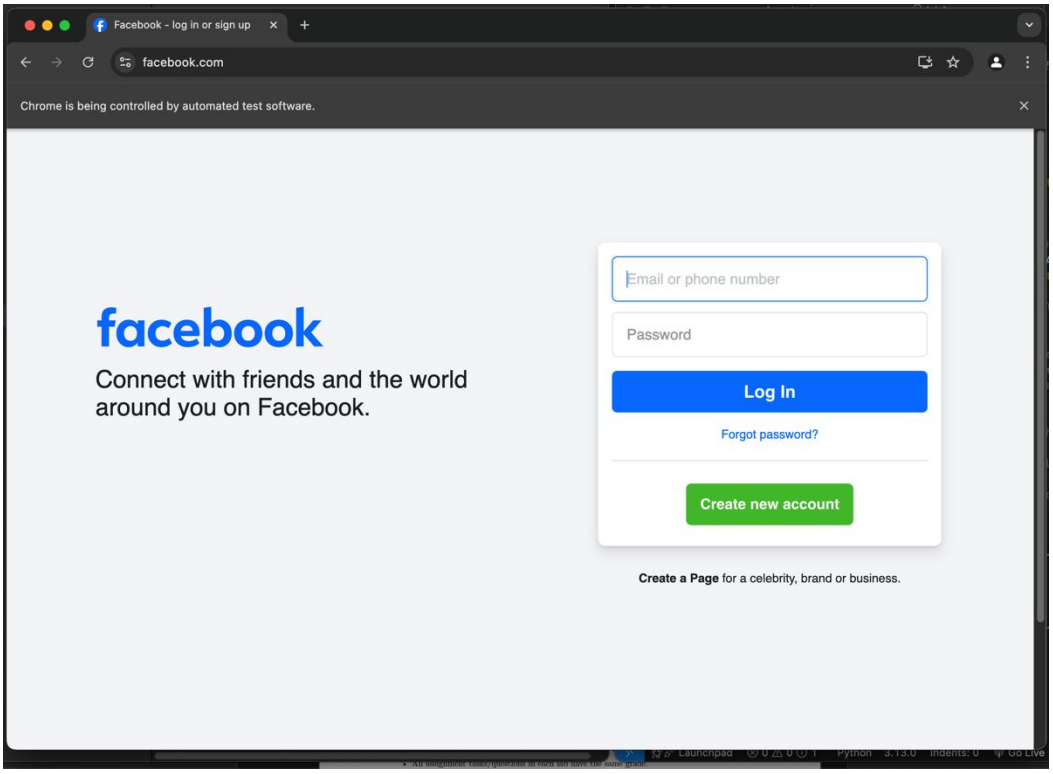
```
jah@Jahmils-MacBook-Pro lab3 % python3 Q3.py
Tag name of the email field: input
.
----------------------------------------------------------------------
Ran 1 test in 2.792s

OK
jah@Jahmils-MacBook-Pro lab3 %
```

Q4:




```
jah@Jahmils-MacBook-Pro lab3 % python3 Q4.py
.
----------------------------------------------------------------------
Ran 1 test in 4.293s

OK
jah@Jahmils-MacBook-Pro lab3 %
```

Q5:

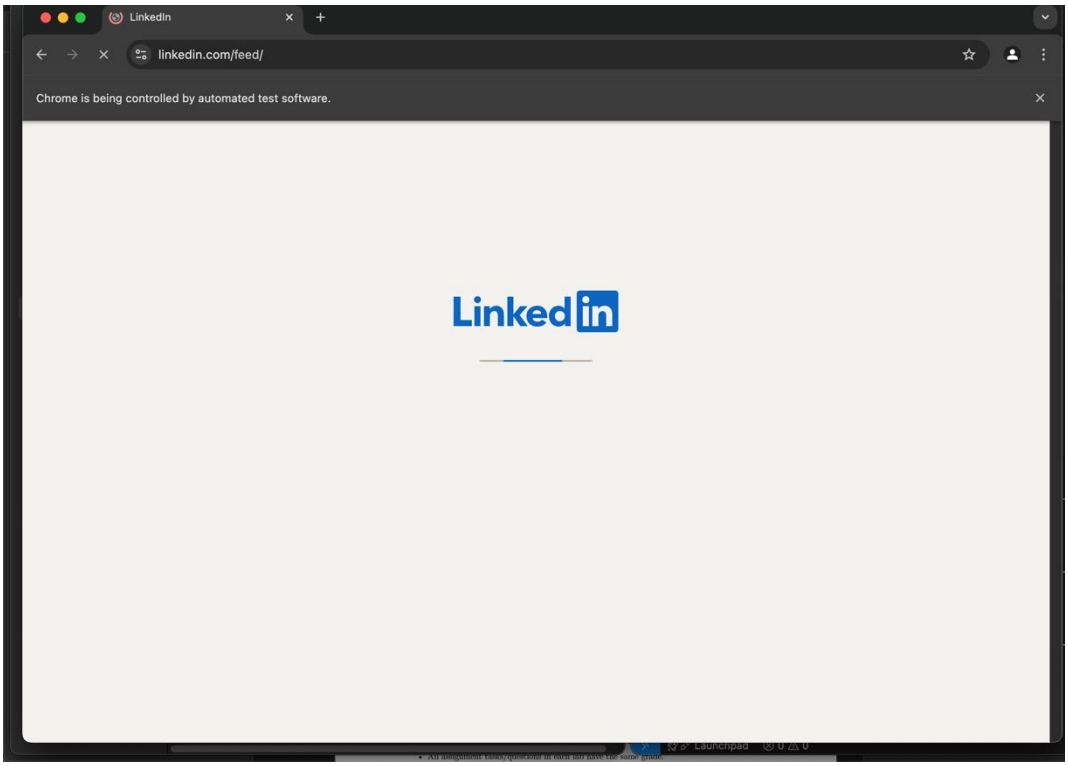



```
jah@Jahmils-MacBook-Pro lab3 % python3 Q5.py
Browser opened successfully.
Google launched.
Searched for Facebook.
Clicked the first search result.
Page Title: Facebook - log in or sign up
Current URL: https://www.facebook.com/
Page title verified.
.Browser closed.

----------------------------------------------------------------------
Ran 1 test in 31.663s

OK
jah@Jahmils-MacBook-Pro lab3 %
```

Q6:

```
jah@Jahmils-MacBook-Pro lab3 % python3 Q6.py
Session One executed
.
----------------------------------------------------------------------
Ran 1 test in 4.304s

OK
Session Two executed
.
----------------------------------------------------------------------
Ran 1 test in 6.844s

OK
Session Three executed
.
----------------------------------------------------------------------
Ran 1 test in 7.170s

OK
jah@Jahmils-MacBook-Pro lab3 %
```

Q7:



```
jah@Jahmils-MacBook-Pro lab3 % python3 Q7.py
Browser opened successfully.
LinkedIn login page opened.
Login form submitted.
Actual URL: https://www.linkedin.com/feed/
Login successful: URL matches expected URL.
.Browser closed.

----------------------------------------------------------------------
Ran 1 test in 32.328s

OK
jah@Jahmils-MacBook-Pro lab3 %
```

Appendix:

Q2:

Q2.py > ...

```python
import unittest
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager

class Guru99TitleTest(unittest.TestCase):
    # This class inherits from unittest.TestCase, which is a built-in class in the unittest module.
    # It provides various methods to create and run tests.

    @classmethod
    def setUpClass(cls):
        """Setup WebDriver before running tests"""
        # This method is a class method that sets up the WebDriver before any tests are run.
        # It is executed once for the entire class.
        cls.driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
        # The above line initializes the Chrome WebDriver using the ChromeDriverManager to handle the driver installation.
        cls.driver.get("http://demo.guru99.com/test/newtours/")
        # The above line navigates the WebDriver to the specified URL.

    def test_title(self):
        """Test to verify the title of the webpage"""
        # This is a test method that will be run by the unittest framework.
        actual_title = self.driver.title
        # The above line retrieves the title of the current webpage.
        print(f"Page Title: {actual_title}")
        # The above line prints the actual title of the webpage.

        expected_title = "Welcome: Mercury Tours"
        # The above line sets the expected title of the webpage.

        # Assertion to verify the title
        self.assertEqual(actual_title, expected_title, f"Error: Title mismatch! Expected: {expected_title}, Got: {actual_title}")
        # The above line asserts that the actual title matches the expected title.
        # If the titles do not match, it will raise an AssertionError with the specified message.
        print("Title matches the expected value!")
        # The above line prints a success message if the titles match.

```

```python
    @classmethod
    def tearDownClass(cls):
        """Quit WebDriver after all tests"""
        # This method is a class method that quits the WebDriver after all tests are run.
        # It is executed once for the entire class.
        cls.driver.quit()
        # The above line quits the WebDriver, closing all associated windows.

if __name__ == "__main__":
    unittest.main()
    # The above line runs the test case when the script is executed directly.
```

Q3:

Q3.py > FacebookEmailFieldTest

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

class FacebookEmailFieldTest(unittest.TestCase):
    # This class inherits from unittest.TestCase, which is a built-in class in the unittest module.
    # It provides various methods to create and run tests.

    @classmethod
    def setUpClass(cls):
        """Setup WebDriver before running tests"""
        # This method is a class method that sets up the WebDriver before any tests are run.
        # It is executed once for the entire class.
        cls.driver = webdriver.Chrome()  # Change to webdriver.Firefox() if using Firefox
        # The above line initializes the Chrome WebDriver. You can change it to Firefox if needed.
        cls.driver.implicitly_wait(10)
        # The above line sets an implicit wait of 10 seconds for the WebDriver. This means the WebDriver will wait up to 10 seconds
        # for elements to appear before throwing an exception.

    def test_get_email_field_tag_name(self):
        """Test to find the email field and retrieve its tag name"""
        # This is a test method that will be run by the unittest framework.
        self.driver.get("http://www.facebook.com")
        # The above line navigates the WebDriver to the specified URL.

        # Locate the email field
        email_field = self.driver.find_element(By.NAME, "email")
        # The above line finds the email field on the webpage by its name attribute.

        # Retrieve and print the tag name
        tag_name = email_field.tag_name
        # The above line retrieves the tag name of the email field element.
        print(f"Tag name of the email field: {tag_name}")
        # The above line prints the tag name of the email field.

        # Verify if the tag name is 'input'
        self.assertEqual(tag_name, "input", "Tag name does not match!")
        # The above line asserts that the tag name of the email field is 'input'.
        # If the tag name does not match, it will raise an AssertionError with the specified message.
```

```python
    @classmethod
    def tearDownClass(cls):
        """Close the WebDriver after running tests"""
        # This method is a class method that quits the WebDriver after all tests are run.
        # It is executed once for the entire class.
        cls.driver.quit()
        # The above line quits the WebDriver, closing all associated windows.

if __name__ == "__main__":
    unittest.main()
    # The above line runs the test case when the script is executed directly.

```

Q4:

Q4.py > LambdaTestTodoApp > setUpClass

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

class LambdaTestTodoApp(unittest.TestCase):
    # This class inherits from unittest.TestCase, which is a built-in class in the unittest module.
    # It provides various methods to create and run tests.

    @classmethod
    def setUpClass(cls):
        """Setup WebDriver before running tests"""
        # This method is a class method that sets up the WebDriver before any tests are run.
        # It is executed once for the entire class.
        cls.driver = webdriver.Chrome()  # Change to webdriver.Firefox() if using Firefox
        # The above line initializes the Chrome WebDriver. You can change it to Firefox if needed.
        cls.driver.implicitly_wait(10)
        cls.driver.get("https://lambdatest.github.io/sample-todo-app/")
        # The above line navigates the WebDriver to the specified URL.

```

```python

    def test_check_items_and_add_name(self):
        """Test to check 'Second Item' and 'Fourth Item', then add a name and submit"""
        # This is a test method that will be run by the unittest framework.
        driver = self.driver

        try:
            # Locate and check 'Second Item'
            second_item = driver.find_element(By.NAME, "li2")
            # The above line finds the 'Second Item' checkbox on the webpage by its name attribute.
            second_item.click()
            # The above line clicks the 'Second Item' checkbox to check it.
            self.assertTrue(second_item.is_selected(), "Second Item is not checked!")
            # The above line asserts that the 'Second Item' checkbox is checked.
            # If it is not checked, it will raise an AssertionError with the specified message.

            # Locate and check 'Fourth Item'
            fourth_item = driver.find_element(By.NAME, "li4")
            # The above line finds the 'Fourth Item' checkbox on the webpage by its name attribute.
            fourth_item.click()
            # The above line clicks the 'Fourth Item' checkbox to check it.
            self.assertTrue(fourth_item.is_selected(), "Fourth Item is not checked!")
            # The above line asserts that the 'Fourth Item' checkbox is checked.
            # If it is not checked, it will raise an AssertionError with the specified message.

            # Locate text field, clear it, and add a name
            name_field = driver.find_element(By.ID, "sampletodotext")
            # The above line finds the text field on the webpage by its ID attribute.
            name_field.clear()
            # The above line clears any existing text in the text field.
            name_field.send_keys("Jahmil Ally")  # Replace with your own name
            # The above line enters the specified name into the text field.

            # Submit the form
            add_button = driver.find_element(By.ID, "addbutton")
            # The above line finds the 'Add' button on the webpage by its ID attribute.
            add_button.click()
            # The above line clicks the 'Add' button to submit the form.
```

```
59
60              # Verify new item is added
61              time.sleep(2)  # Allow time for item to appear
62              # The above line waits for 2 seconds to allow the new item to appear on the webpage.
63              new_item = driver.find_element(By.XPATH, "//span[text()='Jahmil Ally']")
64              # The above line finds the new item on the webpage by its text content.
65              self.assertIsNotNone(new_item, "New item was not added!")
66              # The above line asserts that the new item is not None.
67              # If the new item is not found, it will raise an AssertionError with the specified message.
68
69              # Additional check: Compare actual vs. expected text
70              expected_text = "Jahmil Ally"
71              # The above line sets the expected text of the new item.
72              actual_text = new_item.text
73              # The above line retrieves the actual text of the new item.
74              self.assertEqual(actual_text, expected_text, "New item text does not match!")
75              # The above line asserts that the actual text of the new item matches the expected text.
76              # If the texts do not match, it will raise an AssertionError with the specified message.
77
78          except Exception as e:
79              print(f"Test failed: {str(e)}")
80              # The above line prints the error message if an exception occurs during the test.
81              driver.save_screenshot(f"error_screenshot_{int(time.time())}.png")  # Save screenshot for debugging
82              # The above line saves a screenshot of the webpage for debugging purposes.
83              raise
84              # The above line re-raises the exception to indicate test failure.
85
86      @classmethod
87      def tearDownClass(cls):
88          """Close the WebDriver after tests"""
89          # This method is a class method that quits the WebDriver after all tests are run.
90          # It is executed once for the entire class.
91          cls.driver.quit()
92          # The above line quits the WebDriver, closing all associated windows.
93
94  if __name__ == "__main__":
95      unittest.main()
96      # The above line runs the test case when the script is executed directly.
97
```

Q5:

Q5.py > GoogleSearchTest > setUpClass

```python
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import random

class GoogleSearchTest(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """ Setup WebDriver before running tests """
        # This method is a class method that sets up the WebDriver before any tests are run.
        # It is executed once for the entire class.
        chrome_options = Options()
        # The above line creates an instance of Chrome options.
        chrome_options.add_argument("--disable-blink-features=AutomationControlled")
        # The above line adds an argument to disable automation control features.
        chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")
        # The above line sets a custom user-agent string.
        cls.driver = webdriver.Chrome(options=chrome_options)
        # The above line initializes the Chrome WebDriver with the specified options.
        cls.driver.implicitly_wait(10)
        print("Browser opened successfully.")
```

```python

    def test_search_facebook_and_verify_title(self):
        """ Test to search for 'Facebook', click the first result, and verify the page title """
        # This is a test method that will be run by the unittest framework.

        # Step 1: Launch Google
        self.driver.get("http://www.google.com")
        # The above line navigates the WebDriver to the specified URL.
        print("Google launched.")

        # Step 2: Perform search for 'Facebook'
        time.sleep(random.uniform(2, 5))  # Wait for Google to load
        # The above line waits for a random time between 2 to 5 seconds to simulate human behavior.
        search_box = self.driver.find_element(By.NAME, "q")
        # The above line finds the search box on the Google homepage by its name attribute.
        search_box.send_keys("Facebook")
        # The above line enters the search term 'Facebook' into the search box.
        search_box.submit()
        # The above line submits the search form.
        print("Searched for Facebook.")

        # Step 3: Wait for results to load and click the first search result
        WebDriverWait(self.driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//h3"))
        )
        # The above lines wait up to 20 seconds for the search results to load and locate the first result by its XPath.
        first_result = self.driver.find_element(By.XPATH, "//h3")
        # The above line finds the first search result on the webpage by its XPath.
        first_result.click()
        # The above line clicks the first search result.
        print("Clicked the first search result.")
```

```
58
59          # Step 4: Verify the page title contains 'Facebook'
60          time.sleep(random.uniform(15, 25))  # Wait for page to load
61          # The above line waits for a random time between 15 to 25 seconds to allow the page to load.
62          for _ in range(5):
63              actual_title = self.driver.title
64              # The above line retrieves the title of the current webpage.
65              current_url = self.driver.current_url
66              # The above line retrieves the current URL of the webpage.
67              print(f"Page Title: {actual_title}")
68              # The above line prints the actual title of the webpage.
69              print(f"Current URL: {current_url}")
70              # The above line prints the current URL of the webpage.
71              if "Facebook" in actual_title:
72                  break
73              # The above line breaks the loop if the title contains 'Facebook'.
74              time.sleep(5)  # Wait a bit longer if the title is not correct
75              # The above line waits for 5 seconds before checking the title again.
76
77          self.assertTrue("Facebook" in actual_title, "Error: Title does not contain 'Facebook'.")
78          # The above line asserts that the actual title contains 'Facebook'.
79          # If the title does not contain 'Facebook', it will raise an AssertionError with the specified message.
80          print("Page title verified.")
81
82      @classmethod
83      def tearDownClass(cls):
84          """ Close the WebDriver after tests """
85          # This method is a class method that quits the WebDriver after all tests are run.
86          # It is executed once for the entire class.
87          cls.driver.quit()
88          # The above line quits the WebDriver, closing all associated windows.
89          print("Browser closed.")
90
91  if __name__ == "__main__":
92      unittest.main()
93      # The above line runs the test case when the script is executed directly.
94
```

Q6:

Q6.py > ...

```python
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from concurrent.futures import ThreadPoolExecutor

class ParallelTest(unittest.TestCase):
    # This class inherits from unittest.TestCase, which is a built-in class in the unittest module.
    # It provides various methods to create and run tests.

    def setUp(self):
        pass  # No setup needed here, WebDriver will be initialized in each test method

    def executSessionOne(self):
        """Test method to execute the first session"""
        chrome_options = Options()
        # The above line creates an instance of Chrome options.
        chrome_options.add_argument("--disable-blink-features=AutomationControlled")
        # The above line adds an argument to disable automation control features.
        chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")
        # The above line sets a custom user-agent string.
        driver = webdriver.Chrome(options=chrome_options)
        # The above line initializes the Chrome WebDriver with the specified options.
        driver.implicitly_wait(10)


        driver.get("http://demo.guru99.com/V4/")
        # The above line navigates the WebDriver to the specified URL.
        user_id = driver.find_element(By.NAME, "uid")
        # The above line finds the user ID field on the webpage by its name attribute.
        user_id.send_keys("Driver 1")
        # The above line enters the text 'Driver 1' into the user ID field.
        print("Session One executed")
        # The above line prints a message indicating that session one has been executed.

        driver.quit()
        # The above line quits the WebDriver, closing all associated windows.
```

```python
    def executSessionTwo(self):
        """Test method to execute the second session"""
        chrome_options = Options()
        # The above line creates an instance of Chrome options.
        chrome_options.add_argument("--disable-blink-features=AutomationControlled")
        # The above line adds an argument to disable automation control features.
        chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")
        # The above line sets a custom user-agent string.
        driver = webdriver.Chrome(options=chrome_options)
        # The above line initializes the Chrome WebDriver with the specified options.
        driver.implicitly_wait(10)


        driver.get("http://demo.guru99.com/V4/")
        # The above line navigates the WebDriver to the specified URL.
        user_id = driver.find_element(By.NAME, "uid")
        # The above line finds the user ID field on the webpage by its name attribute.
        user_id.send_keys("Driver 2")
        # The above line enters the text 'Driver 2' into the user ID field.
        print("Session Two executed")
        # The above line prints a message indicating that session two has been executed.

        driver.quit()
        # The above line quits the WebDriver, closing all associated windows.
```

```
63
64      def executSessionThree(self):
65          """Test method to execute the third session"""
66          chrome_options = Options()
67          # The above line creates an instance of Chrome options.
68          chrome_options.add_argument("--disable-blink-features=AutomationControlled")
69          # The above line adds an argument to disable automation control features.
70          chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")
71          # The above line sets a custom user-agent string.
72          driver = webdriver.Chrome(options=chrome_options)
73          # The above line initializes the Chrome WebDriver with the specified options.
74          driver.implicitly_wait(10)
75
76          driver.get("http://demo.guru99.com/V4/")
77          # The above line navigates the WebDriver to the specified URL.
78          user_id = driver.find_element(By.NAME, "uid")
79          # The above line finds the user ID field on the webpage by its name attribute.
80          user_id.send_keys("Driver 3")
81          # The above line enters the text 'Driver 3' into the user ID field.
82          print("Session Three executed")
83          # The above line prints a message indicating that session three has been executed.
84
85          driver.quit()
86          # The above line quits the WebDriver, closing all associated windows.
87
88      def tearDown(self):
89          pass  # No teardown needed here, WebDriver is closed in each test method
90
91  def run_test(test_method):
92      """Function to run a test method"""
93      suite = unittest.TestSuite()
94      # The above line creates a test suite.
95      suite.addTest(ParallelTest(test_method))
96      # The above line adds the specified test method to the test suite.
97      runner = unittest.TextTestRunner()
98      # The above line creates a test runner.
99      runner.run(suite)
100     # The above line runs the test suite using the test runner.
101
```

```
102 if __name__ == "__main__":
103     test_methods = ['executSessionOne', 'executSessionTwo', 'executSessionThree']
104     # The above line defines a list of test methods to be run in parallel.
105     with ThreadPoolExecutor(max_workers=3) as executor:
106         executor.map(run_test, test_methods)
107         # The above lines create a thread pool executor with a maximum of 3 workers and map the test methods to be run in parallel.
```

Q7:

```python
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
import time
import os
from dotenv import load_dotenv

class LinkedInLoginTest(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """ Setup WebDriver before running tests """
        load_dotenv()  # Load environment variables from .env file
        # The above line loads environment variables from a .env file.
        chrome_options = Options()
        # The above line creates an instance of Chrome options.
        chrome_options.add_argument("--disable-blink-features=AutomationControlled")
        # The above line adds an argument to disable automation control features.
        chrome_options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
        # The above line sets a custom user-agent string.
        cls.driver = webdriver.Chrome(options=chrome_options)
        # The above line initializes the Chrome WebDriver with the specified options.
        cls.driver.implicitly_wait(10)
        print("Browser opened successfully.")
```

```python

    def test_linkedin_login(self):
        """ Test to login to LinkedIn and verify the URL after login """
        # This is a test method that will be run by the unittest framework.
        self.driver.get("https://www.linkedin.com/login")
        # The above line navigates the WebDriver to the LinkedIn login page.
        print("LinkedIn login page opened.")

        # Find the username, password fields and login button
        username_field = self.driver.find_element(By.ID, "username")
        # The above line finds the username field on the LinkedIn login page by its ID attribute.
        password_field = self.driver.find_element(By.ID, "password")
        # The above line finds the password field on the LinkedIn login page by its ID attribute.
        login_button = self.driver.find_element(By.XPATH, "//button[@type='submit']")
        # The above line finds the login button on the LinkedIn login page by its XPath.

        # Fill in the username and password from environment variables
        username_field.send_keys(os.getenv("MY_LK_USER"))
        # The above line enters the username from the environment variable into the username field.
        password_field.send_keys(os.getenv("MY_LK_PASS"))
        # The above line enters the password from the environment variable into the password field.
        login_button.click()
        # The above line clicks the login button to submit the login form.
        print("Login form submitted.")

        # Wait for the page to load
        time.sleep(20)
        # The above line waits for 20 seconds to allow the page to load.

        # Verify the URL after login
        expected_url = "https://www.linkedin.com/feed/"
        # The above line sets the expected URL after a successful login.
        actual_url = self.driver.current_url
        # The above line retrieves the current URL of the webpage.
        print(f"Actual URL: {actual_url}")
        # The above line prints the actual URL of the webpage.
        self.assertEqual(expected_url, actual_url, "Login failed: URL does not match expected URL.")
        # The above line asserts that the actual URL matches the expected URL.
        # If the URLs do not match, it will raise an AssertionError with the specified message.
        print("Login successful: URL matches expected URL.")
```

```
    @classmethod
    def tearDownClass(cls):
        """ Close the WebDriver after tests """
        # This method is a class method that quits the WebDriver after all tests are run.
        # It is executed once for the entire class.
        cls.driver.quit()
        # The above line quits the WebDriver, closing all associated windows.
        print("Browser closed.")

if __name__ == "__main__":
    unittest.main()
    # The above line runs the test case when the script is executed directly.
```