A black background with red text

AI-generated content may be incorrect.

Computer Systems Engineering

Verification and Validation of Software

Project: Install the **Selenium**

Date: 25/6/2025

Name: Sama Haitham Sammar

ID: 202110795

Class: Verification and Validation – Section1

Instructor: Sami Awad

**Part 1 - Theoretical**

**Introduction to Selenium:**

Selenium is a free and open-source tool used to automate web application testing. It allows developers to write scripts that simulate real user actions, such as clicking buttons, entering text and navigating pages. Selenium supports multiple programming languages such as Python, Java, and JavaScript, and works with most modern web browsers. It is widely used in software testing to help find errors and ensure that applications work correctly before they are released.

**Objective:**

The goal of this part is to learn about Selenium as a testing tool, and explore how it can be installed and used in real scenarios.

**Video Watched:**

**Title:** How to install Selenium on Windows 10 | Complete Installation Guide 2021.

**Link:** [**https://youtu.be/0oJsDVPX-LA?si=V0mTQGBX8jW5fvIj**](https://youtu.be/0oJsDVPX-LA?si=V0mTQGBX8jW5fvIj)

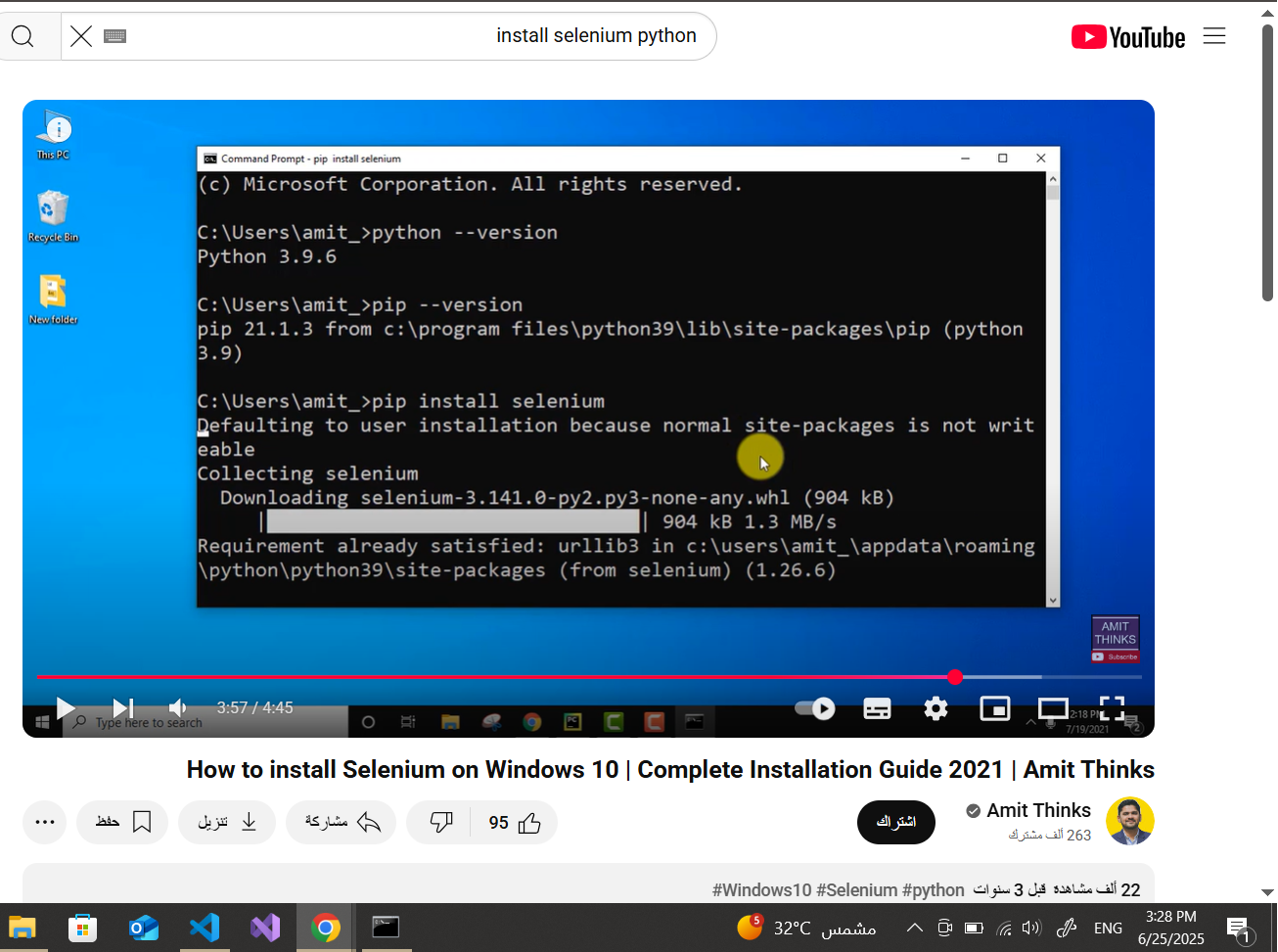


Figure 1 Screenshot while watching the video and installing selenium

**Summary of the video:**

The video provides a complete guide on how to:

* Install Python and Pip.
* Install Selenium and WebDriver Manager.

In addition to watching the video, I also searched online to understand more about Selenium.

This helped me understand that Selenium is not just about writing code, but it also offers tools like Selenium IDE for beginners and Selenium WebDriver for developers. It’s widely used in regression testing to repeatedly test functionalities after updates.

**Link:** <https://en.wikipedia.org/wiki/Selenium_(software)>

**Part 2 – Practical**

**Objective:**

To apply knowledge of Selenium by writing a script to test the login form using multiple test cases. The test simulates user input for different scenarios.

**Tools Used:**

|  |  |
| --- | --- |
| Python | Programming Language |
| Selenium | Web automation/testing library |
| WebDriver-Manager | Auto-downloads WebDriver (ChromeDriver) |
| |  | | --- | | Visual Studio Code |  |  | | --- | |  | | Code editor |
| Google Chrome | Browser for testing |

**Web Page Used:**

A simple HTML login form named login.html with two fields:

* Username
* Password

**Screenshots:**



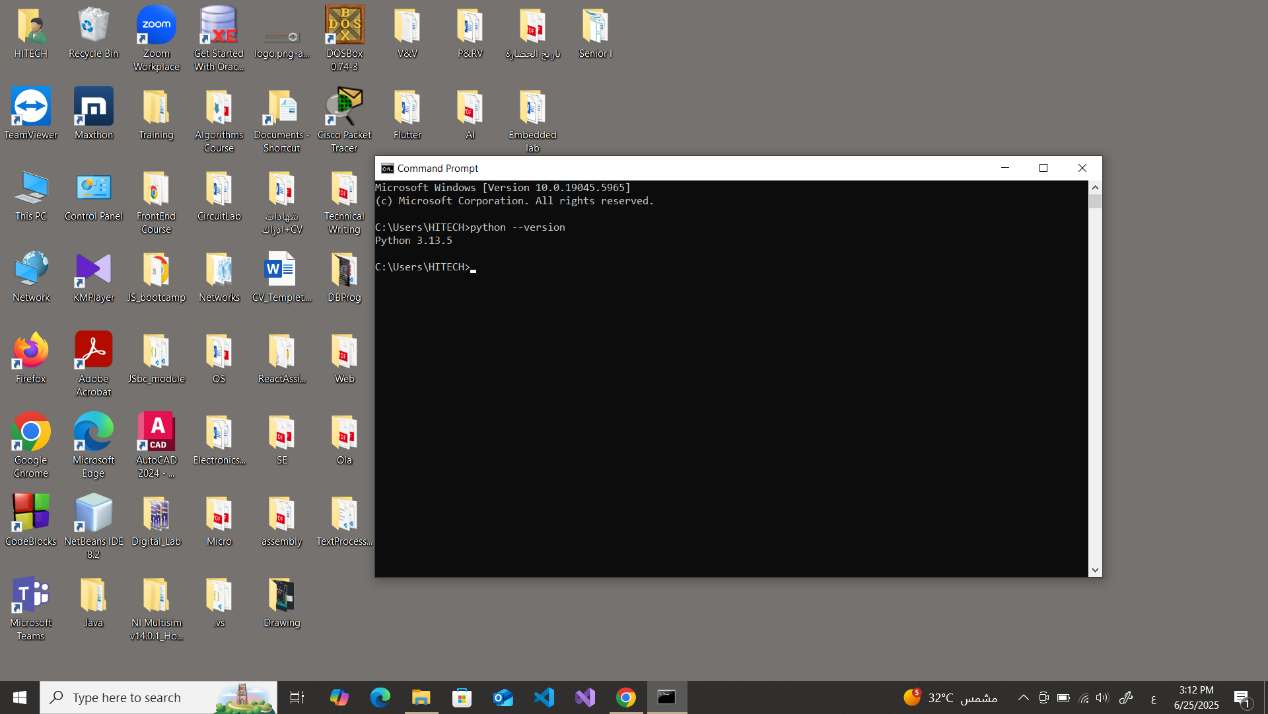
****



Figure 2 Python is installed

To check: python --version

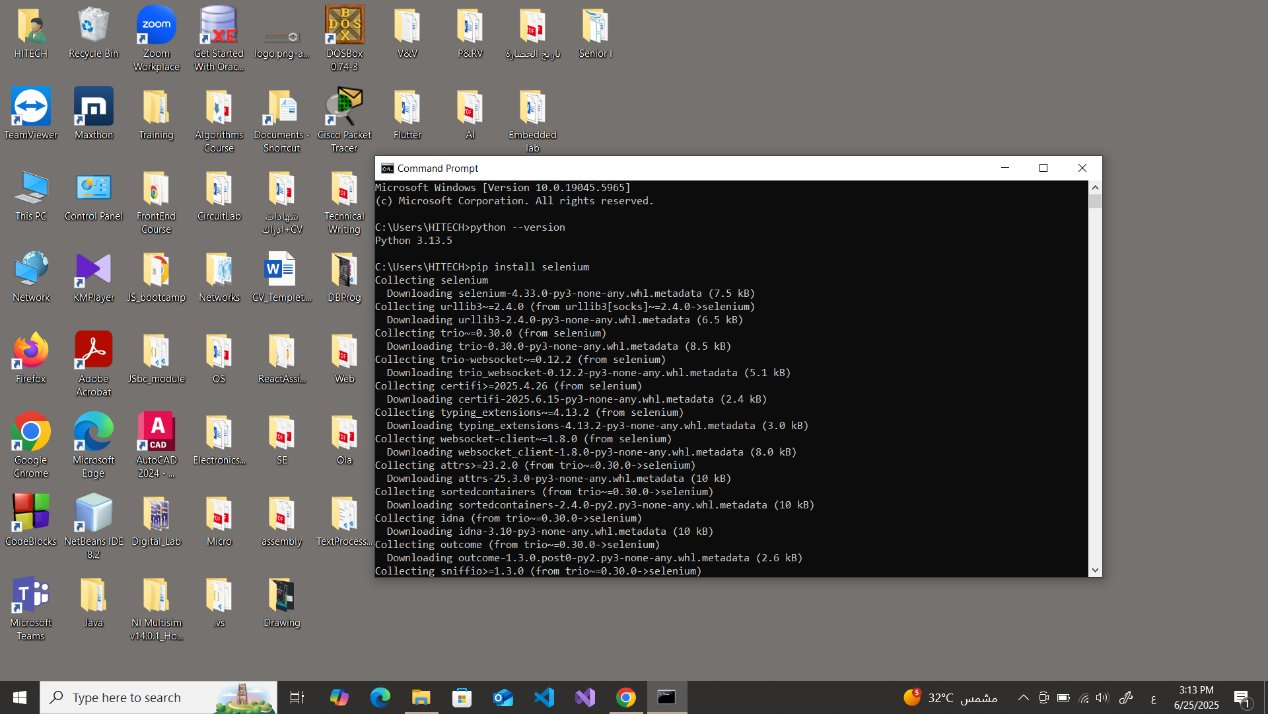




Figure 3 Installing Selenium

Pip install selenium

A screenshot of a computer

AI-generated content may be incorrect.



Figure 4 Installing WebDriver-manager

Pip install webdriver-manager

The code of web page:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Login Page</title>  </head>  <body>  <h2>Login Form</h2>  <form>  <label for="username">Username:</label>  <input type="text" id="username" name="username"><br><br>  <label for="password">Password:</label>  <input type="password" id="password" name="password"><br><br>  <input type="submit" value="Login">  </form>  </body>  </html> |

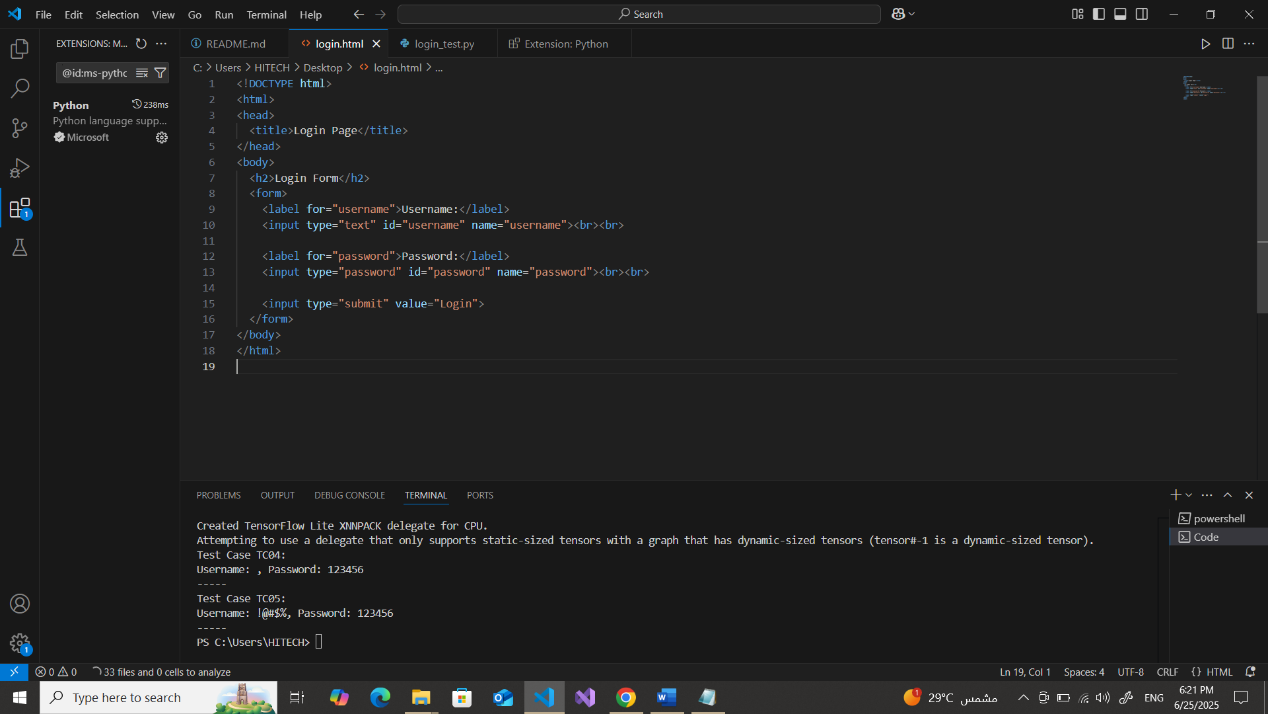


Figure 5 login.html

Python Test:

|  |
| --- |
| **from** **selenium** **import** webdriver  **from** **selenium.webdriver.common.by** **import** By  **from** **selenium.webdriver.chrome.service** **import** Service  **from** **webdriver\_manager.chrome** **import** ChromeDriverManager  **import** **time**  test\_cases = [  {"username": "", "password": ""}, # TC#1  {"username": "Sama", "password": "123456"}, # TC#2  {"username": "Sama", "password": ""}, # TC#3  {"username": "", "password": "123456"}, # TC#4  {"username": "!@#$%", "password": "123456"}, # TC#5  ]  driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))  **for** i, case **in** enumerate(test\_cases):  driver.get("file:///C:/Users/HITECH/Desktop/login.html")  time.sleep(**1**)  username\_field = driver.find\_element(By.ID, "username")  password\_field = driver.find\_element(By.ID, "password")  username\_field.clear()  password\_field.clear()  username\_field.send\_keys(case["username"])  password\_field.send\_keys(case["password"])  **print**(f"Test Case TC0{i+1}:")  **print**(f"Username: {case['username']}, Password: {case['password']}")  **print**("-----")  time.sleep(**2**)    driver.quit() |

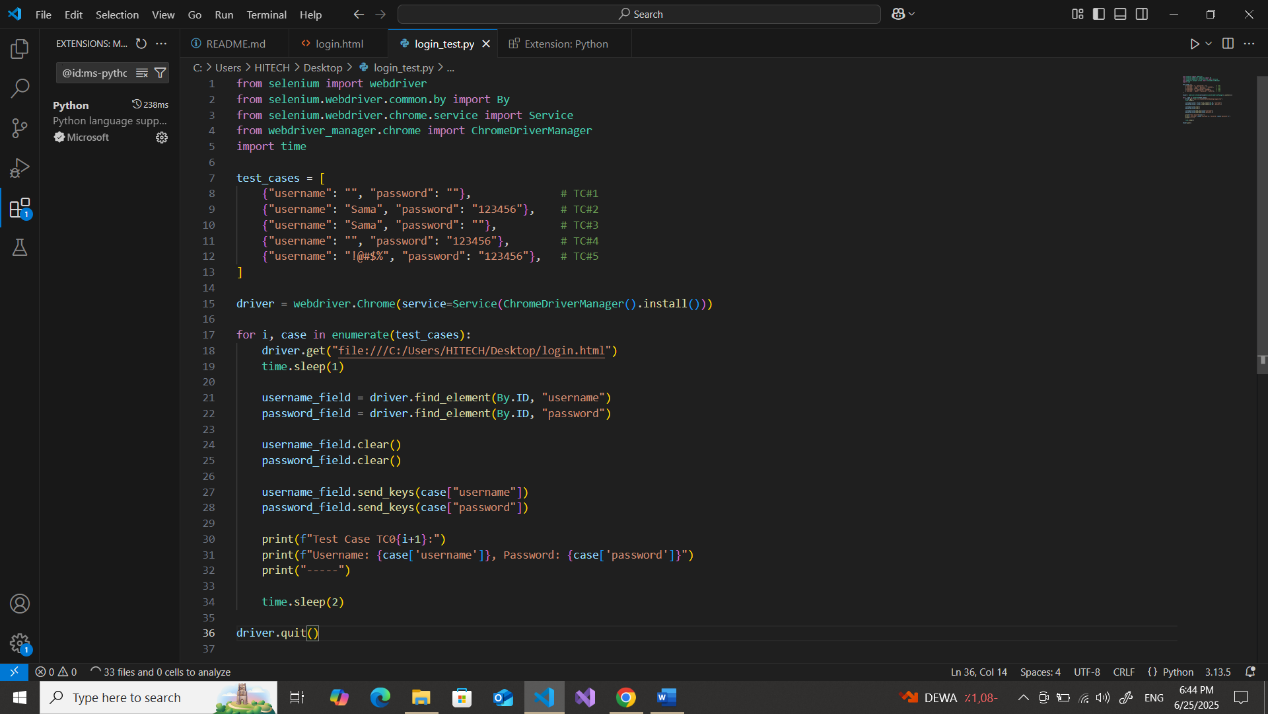


Figure 6 Python Test

Test Steps:

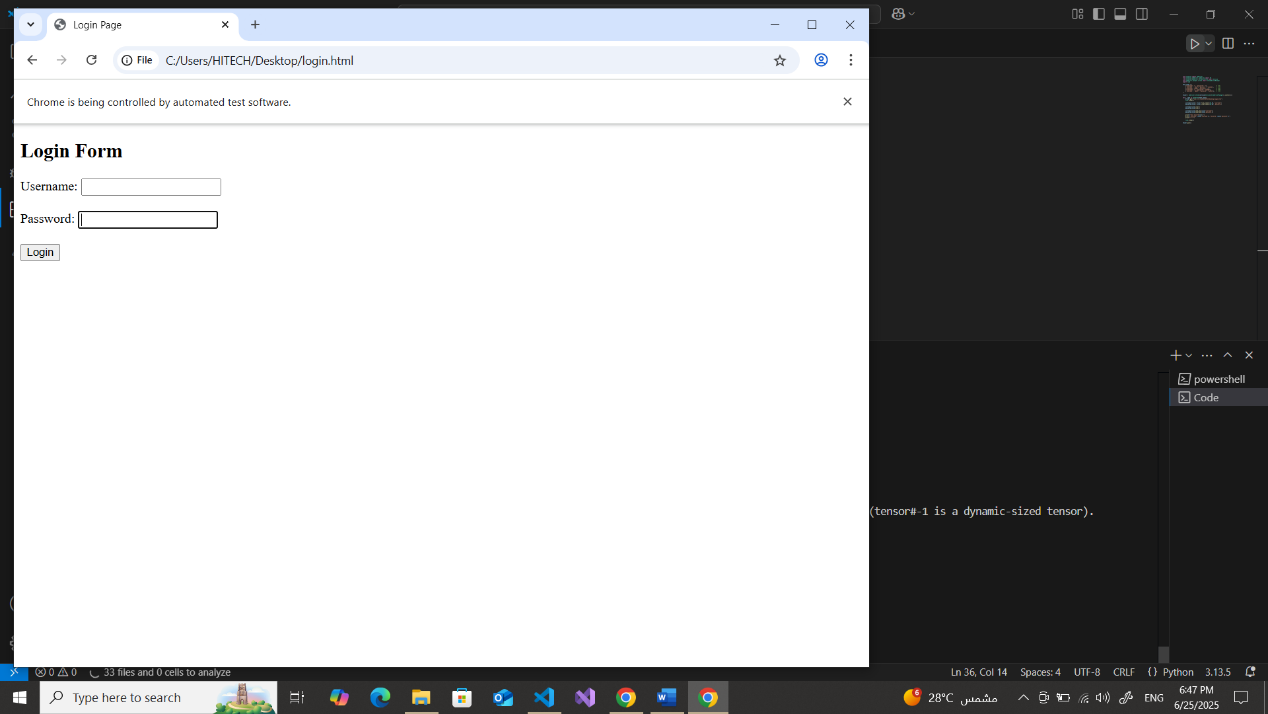
1. Run the Python script using Selenium
2. Automatically launch the Chrome browser
3. Load the local login.html page from the Desktop
4. Loop through 5 test scenarios
5. Fill in username and password fields with different values
6. Wait briefly to observe results
7. Refresh the page before each new test case
8. Close the browser automatically after completing all tests

Test Cases Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Description | Username | Password | Expected Behavior |
| TC#1 | Both fields empty |  |  | No action or error shown |
| TC#2 | Both fields valid | Sama | 123456 | Successful input simulation |
| TC#3 | Username only | Sama |  | Incomplete input, no login |
| TC#4 | Password only |  | 123456 | Incomplete input, no login |
| TC#5 | Special characters in username | !@#$% | 123456 | Input accepted or handled appropriately |

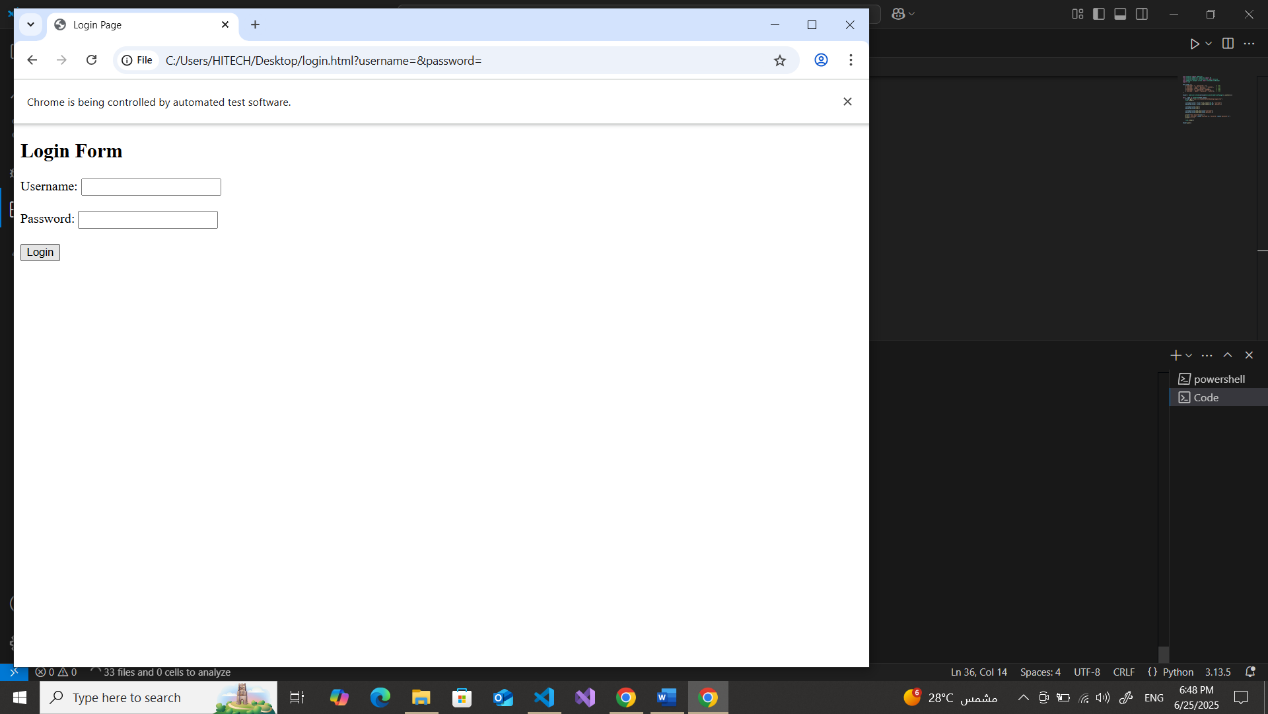
Since the login page is a static HTML file with no backend or validation logic, the result for each test case is that the fields are filled automatically with a specific entry. No actual login or error message is expected. But when press login button the entries will show on the URL.

TC#1:



And this screenshot after press **Login**







TC#2

A screenshot of a computer

AI-generated content may be incorrect.

And this screenshot after press **Login**

A screenshot of a computer

AI-generated content may be incorrect.



TC#3

A computer screen with a white screen

AI-generated content may be incorrect.

And this screenshot after press **Login**

A screenshot of a computer

AI-generated content may be incorrect.



TC#4

A screenshot of a computer

AI-generated content may be incorrect.

And this screenshot after press **Login**

A computer screen with a white screen

AI-generated content may be incorrect.



TC#5

**A screenshot of a computer

AI-generated content may be incorrect.**

And this screenshot after press **Login**

A screenshot of a computer

AI-generated content may be incorrect.



The last screenshot shows **Terminal** that it successfully works

A screenshot of a computer program

AI-generated content may be incorrect.

**Conclusion:**

This report demonstrates the use of selenium to automate test cases on a web application. Through the video, I gained a solid understanding of the setup and usage. The practical implementation involved writing Python code to simulate 5 test scenarios for the login form. All test cases were successfully performed, and the results were observed via browser automation.