Week 3–4: Advanced Tasks – Automation Basics

Introduction to Automation Testing

Automation testing uses scripts and tools to execute test cases automatically.
It saves time, improves accuracy, and is best for repetitive and regression testing.

What Should Be Automated

- Repetitive test cases executed frequently
- Regression test cases
- Smoke tests and critical user flows
- Stable features with minimal UI changes

What Should NOT Be Automated

- One-time or rarely executed test cases
- Exploratory testing
- Usability and UI look & feel testing
- Frequently changing features

Choose an Automation Stack

JavaScript – Cypress

Used for fast and reliable frontend web automation.
Best for modern web apps with simple setup and quick feedback.

Python – PyTest + Selenium

Easy to learn and beginner-friendly automation stack.
Suitable for web automation and flexible test frameworks.

Java – TestNG + Selenium

Enterprise-level automation with strong framework support.
Best for large, complex projects with long-term automation needs.

Tools & Frameworks (Automation Testing)

Selenium WebDriver

An open-source tool used to automate web browsers.
Supports multiple languages and is widely used for cross-browser testing.

Cypress

A modern JavaScript-based automation tool for web applications.
Provides fast execution, easy setup, and real-time test results.

Postman

A popular tool for testing APIs manually and automatically.
Used to send requests, validate responses, and check API functionality.

PyTest

A Python testing framework used to execute automation test cases.
Supports simple syntax, fixtures, and detailed reporting.

TestNG

A Java-based testing framework inspired by JUnit.
Supports parallel execution, annotations, and test grouping.

JUnit

A widely used Java unit testing framework.
Mainly used by developers to test individual units of code.

Automation Practice – Basics

Locators (Very Important)

ID
Finds an element using its unique ID attribute.
Fast and most reliable locator.

XPath
Finds elements using XML path expressions.
Used when ID is not available (can be absolute or relative).

CSS Selector
Finds elements using CSS rules.
Faster than XPath and widely used in modern apps.

Common Locator Examples

```
ID:       id="username"
XPath:    //input[@name='email']
CSS:      input#username
```

Automation Script: Python + Selenium

What This Script Does

✓ Open browser
✓ Navigate to a website
✓ Fill a form
✓ Submit form
✓ Validate page title / success message

Step 1: Install Required Tools

```
pip install selenium
```

Download **ChromeDriver** and keep it in your system PATH.

Step 2: Sample Automation Script

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

# Open browser
driver = webdriver.Chrome()
driver.maximize_window()

# Navigate to site
driver.get("https://example.com/login")
time.sleep(2)

# Fill the form
driver.find_element(By.ID, "username").send_keys("testuser")
driver.find_element(By.ID, "password").send_keys("password123")

# Submit form
driver.find_element(By.ID, "loginBtn").click()
time.sleep(2)

# Validate title
expected_title = "Dashboard"
actual_title = driver.title

if expected_title == actual_title:
    print("Test Passed: Title matched")
else:
    print("Test Failed: Title not matched")

# Close browser
driver.quit()
```

Validation Using Success Message

```
success_msg = driver.find_element(By.XPATH, "//div[@class='success']").text

if "Login successful" in success_msg:
   print("Test Passed: Success message displayed")
else:
   print("Test Failed: Message not displayed")
```

Same Task in Cypress (Very Short Example)

```
cy.visit('/login')
cy.get('#username').type('testuser')
cy.get('#password').type('password123')
cy.get('#loginBtn').click()
cy.contains('Login successful')
```

TASK 1: First Selenium Script (Login Form)

What this script does

✓ Open browser
✓ Navigate to login page
✓ Enter credentials
✓ Click login
✓ Validate page title or success message

Selenium Script (Python)

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

# Launch browser
driver = webdriver.Chrome()
driver.maximize_window()

# Open website
driver.get("https://example.com/login")
time.sleep(2)

# Enter login details
driver.find_element(By.ID, "username").send_keys("testuser")
driver.find_element(By.ID, "password").send_keys("password123")

# Click login
driver.find_element(By.ID, "loginBtn").click()
time.sleep(2)

# Validation
expected_title = "Dashboard"
actual_title = driver.title

assert expected_title == actual_title
print("Login Test Passed")

driver.quit()
```

Locators Used: ID
Validation: Page title

TASK 2: Automate Search & Validation (Amazon / Flipkart)

Scenario

Search for a product and verify search results appear.

Selenium Script – Search Flow

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

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

driver.get("https://www.amazon.in")
time.sleep(3)

# Search product
driver.find_element(By.ID, "twotabsearchtextbox").send_keys("Laptop")
driver.find_element(By.ID, "nav-search-submit-button").click()
time.sleep(3)

# Validation
results = driver.find_elements(By.XPATH, "//div[@data-component-type='s-search-result']")

assert len(results) > 0
print("Search Test Passed")

driver.quit()
```

Validation: Search results count
Real-world flow: Search → Verify output

TASK 3: API Testing Using Postman (JSON Validation)

Sample API

https://jsonplaceholder.typicode.com/users

Steps in Postman

1. Open **Postman**
2. Select **GET**
3. Paste API URL
4. Click **Send**

Validations to Perform

✓ Status code
✓ Response format (JSON)
✓ Key-value validation

Postman Test Script (Tests Tab)

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response is JSON", function () {
    pm.response.to.be.json;
});

pm.test("User ID exists", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData[0]).to.have.property("id");
});
```

API Type: GET
Validation: Status code + JSON body
