
# Automation Testing Tutorial

## Introduction

It overwhelms many beginners with the sheer variety of tools and the intricacies of installing their initial framework. We will simplify automation, demonstrating how to transition from time-consuming manual regression testing to efficient, repeatable tests. Master the core concepts that apply across any toolset through this Automation Testing Tutorial for Beginners. Want to have a look at the roadmap? Download our entire **Automation Testing course syllabus**.

## Why Students or Freshers Learn Automation Testing

Students and freshers must master Automation Testing because it offers a clear route to hot tech jobs:

- **Strong Job Prospects & Salary:** Automation is unavoidable in today's agile development (DevOps/CI/CD), ensuring solid job opportunities and good pay.

- **Critical Skill:** It shifts testing from lengthy manual verification to effective, reproducible script running, making you an invaluable part of any development team.
- **Gateway to Coding:** A simple entry point to code in (e.g., Python/Java) with the immediate value of enhancing project quality.
- **Career Advancement:** Expertise results in niche positions such as SDET (Software Development Engineer in Test) and Test Architect.
- **Speed and Accuracy:** Automated tests give instant feedback, enabling development teams to deliver high-quality code much quicker.

Explore **automation testing interview questions and answers** for reviewing your skills.



## Step-by-Step Automation Testing Tutorial for Beginners

Automation testing is an important practice in contemporary software development. This step-by-step automation testing tutorial will walk beginners through the basics and hands-on steps of installing and running your first automation test using Selenium WebDriver and Python, a very popular and beginner-friendly combination.

## Understanding Automation Testing Fundamentals

### What is Automation Testing?

Automation testing is the utilization of software tools to run scripted tests against a software application, compare actual versus expected results, and report the results. It replaces human-based slow, error-prone manual testing.

## Why Automate?

| Benefit | Description |
| --- | --- |
| **Speed** | Tests run much faster, providing quick feedback to developers. |
| **Accuracy** | Eliminates human error and ensures the exact same steps are repeated every time. |
| **Coverage** | Allows for broader test coverage, especially for complex scenarios and regression testing. |
| **Reusability** | Scripts can be reused across different releases and environments. |

## The Automation Testing Pyramid


Automation Testing Tutorial

The Test Automation Pyramid recommends prioritizing various categories of tests:

- **Unit Tests (Base):** Test individual pieces/functions; most numerous and fastest.
- **Service/API Tests (Middle):** Test business logic and integration points without going through the UI.
- **UI/End-to-End Tests (Top):** Test the application using the User Interface; slowest and most brittle.

## Selecting Your Tools (The Stack)

We will use the below technology stack throughout this tutorial, which is most commonly used in the industry:

| Component | Tool/Technology | Role |
|---|---|---|
| **Programming Language** | Python | The language used to write the test scripts. |
| **Testing Framework** | Pytest | Simplifies test execution, reporting, and structure. |
| **Automation Tool** | Selenium WebDriver | The library that controls the web browser. |
| **Web Browser** | Google Chrome | The application under test. |

## Installation and Setup

### Step 1: Install Python

Make sure you have Python 3.8+ installed on your system. You can download it from the official Python website. Confirm the installation by opening your terminal or command prompt and typing:

*python –version*

*# Output should show your installed version, e.g., Python 3.11.4*

### Step 2: Install Needed Libraries (Selenium and Pytest)

We install the needed libraries using pip (Python package installer).

*# Install Selenium WebDriver for browser control*

*pip install selenium*

*# Install Pytest for test execution framework*

*pip install pytest*

## Step 3: Install the WebDriver

Selenium requires a WebDriver executable to interact with the browser. We will use the Chrome driver.

1. **Check your version of Chrome:** Open up Chrome, navigate to *Settings (⋮)* > *Help* > *About Google Chrome*.
2. **Download WebDriver:** Visit the [official ChromeDriver website] (or its equivalent website for Firefox/Edge). Download the same version as your Chrome browser.
3. **Setup:** Put the downloaded *chromedriver.exe (or chromedriver)* file into a directory that is in your system's **PATH**. An easy method for starters is usually to put it right into the same directory as your Python script for local testing.

**Note:** More recent Selenium (4.6+) versions usually resolve the WebDriver installation in an automated way via a feature named Selenium Manager. If you happen to be using a newer version, you might not need to manually download it!

## Creating Your First Automation Script

We are going to write a straightforward **End-to-End (E2E)** test that tests a search feature on a website (e.g., Google).

## Step 1: Set Up a Project Structure

Make a new folder for your project and a Python file within it. Name your test file, following Pytest conventions, by starting with test_.

*/AutomationProject*

*└── test_search_function.py*

## Step 2: Find the Web Elements (Locators)

In order to interact with a webpage, you require a means of uniquely addressing elements (buttons, input fields, links). These are referred to as locators.

Open the page that you want to target (e.g., google.com) and use your browser's Developer Tools (F12) in order to inspect the elements:

| Element | Tag/Attribute | Locator Type | Locator Value |
|---|---|---|---|
| **Search Box** | name="q" | Name | q |
| **Search Button** | (Often complex) | XPath/CSS | (We'll rely on the simple Enter key press) |

## Step 3: Write the Pytest Script

Open test_search_function.py and input the following code.

```
from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

import time

# — Pytest function must start with 'test_' —

def test_google_search():

    # 1. Initialization: Setup the WebDriver

    # Note: Modern Selenium often handles the driver path automatically.
```

```
    # If not, you might need: driver =
webdriver.Chrome(executable_path='/path/to/chromedriver')

    # Initialize the Chrome browser

    driver = webdriver.Chrome()

    print("\nBrowser is successfully launched.")

    try:

        # 2. Navigate to the URL

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

        print(f"Navigated to: {driver.current_url}")

        # Maximize the window for better viewing

        driver.maximize_window()

        # 3. Locate the Search Box (Input Element)

        # Using By.NAME with the value 'q' (the name attribute of the search box)

        search_box = driver.find_element(By.NAME, "q")

        # 4. Perform Action: Type the search query

        search_term = "Automation Testing Tutorial"

        search_box.send_keys(search_term)

        print(f"Typed search term: '")
```

```
        # 5. Perform Action: Press the Enter key
        search_box.send_keys(Keys.ENTER)
        # Wait a few seconds for the results page to load
        time.sleep(3)
        # 6. Verification (Assertion)
        # Check if the title of the results page contains the search term
        expected_title_part = search_term
        actual_title = driver.title
        assert expected_title_part in actual_title
        print(f"SUCCESS: Title verified. Expected part '{expected_title_part}' found in title: '{actual_title}'")
        # Optional: Check if the search term is still visible in the search box
        updated_search_box = driver.find_element(By.NAME, "q")
        assert updated_search_box.get_attribute("value") == search_term
    except Exception as e:
        print(f"TEST FAILED: An error occurred: {e}")
        # Capture a screenshot on failure (helpful in real-world scripts)
        driver.save_screenshot("failure_screenshot.png")
```

*raise # Re-raise the exception to make the test fail in Pytest*

*finally:*

*# 7. Cleanup: Close the browser after test execution*

*driver.quit()*

*print("Browser closed. Test execution finished.")*

## Running the Automation Test

### Step 1: Open the Terminal

Go to your project directory *(/AutomationProject)* in your terminal or command prompt.

### Step 2: Execute Pytest

Pytest automatically finds any files beginning with test_ and functions within them beginning with test_.

*pytest test_search_function.py*

**Expected Output (Success)**

Pytest will run the script, and you will observe the Chrome browser open, navigate, type, and close.

*========== test session starts ==========*

*platform win32 — Python 3.11.4, pytest-8.3.2, pluggy-1.5.0*

*rootdir: C:\AutomationProject*

*collected 1 item*

*test_search_function.py .  [100%]*

*Browser is successfully launched.*

*Navigated to: https://www.google.com/*

*Typed search term: 'Automation Testing Tutorial'*

*SUCCESS: Title verified. Expected part 'Automation Testing Tutorial' found in title: 'Automation Testing Tutorial – Google Search'*

*Browser closed. Test execution finished.*

*============================== 1 passed in 10.51s ==============================*

## Essential Shell Scripting Concepts Defined

This part outlines the essential concepts employed in the code above.

### A. Initializing WebDriver

*driver = webdriver.Chrome()*

This line is the central part of Selenium. It initializes a new Chrome browser instance and creates the driver object, which is your entry point for giving commands (such as get, click, send_keys) to the browser.

### B. Finding Elements (find_element)

In order to interact with any element, you need to first locate it. The By class assists in defining the locator strategy:

*search_box = driver.find_element(By.NAME, “q”)*

| Locator Strategy | Explanation | When to Use |
|---|---|---|
| **By.ID** | Finds element by unique ID attribute. | Best practice; most reliable. |
| **By.NAME** | Finds element by name attribute. | Good for input fields. |
| **By.XPATH** | Uses XPath expression (XML Path Language). | When ID/Name are unavailable; complex but flexible. |
| **By.CSS_SELECTOR** | Uses CSS Selector syntax. | Fast and often preferred over XPath. |
| **By.CLASS_NAME** | Finds element by class attribute. | Useful if the class is unique. |

## C. Actions and Interactions

After being found, you can interact with the element:

| Command | Purpose | Example |
|---|---|---|
| .send_keys() | Types input into a field. | search_box.send_keys(“text”) |
| .click() | Clicks a button or link. | login_button.click() |
| .get_attribute(“value”) | Retrieves the value of an attribute. | input_field.get_attribute(“value”) |

## D. Assertions

Assertions are tests that decide whether a test succeeds or not. They compare an Actual Result (what the script finds) with an Expected Result (what is supposed to happen).

*assert expected_title_part in actual_title*

If the condition in the assert statement is False, the script stop (fails) right away and prints the failure, which is the expected behavior for a test.

## E. Cleanup (driver.quit())

The driver.quit() instruction is crucial. It properly shuts down the whole browser session and kills the ChromeDriver process, releasing system resources. Put this in a finally block or a Pytest fixture so that it will be executed even if the test fails.

## Next Steps in Growth

- **Refactoring with Pytest Fixtures:** Discover how to use @pytest.fixture to automatically arrange the setup (driver = webdriver.Chrome()) and teardown (driver.quit()) for all your tests.
- **Page Object Model (POM):** Keep your code organized by divorcing web elements (locators) from test logic. This makes your scripts more readable and maintainable.
- **Advanced Locators:** Familiarize yourself with more advanced XPath and CSS selectors to manage dynamic web elements.
- **Data-Driven Testing:** Utilize external files (such as CSV or Excel) to supply several sets of data to one test script.

Learn more with our **automation testing challenges and solutions**.

## Real Time Examples for Automation Testing Tutorial for Learners

Following are some real-world examples where Automation Testing is most important and very useful, and therefore great entry points for students:

## E-commerce Checkout Flow (The Money Path):

- **Scenario:** A multi-step transaction process (Login → Add Item to Cart → Enter Shipping Details → Choose Payment → Confirm Order).
- **Why Automate?:** This flow is executed repeatedly and failure means direct loss of revenue. Automation guarantees the “happy path” works in all browsers and devices prior to each deployment.
- **Learner Focus:** Getting expert-level at element identification, dealing with drop-downs, and data input validation.

## API Health and Performance Check:

- **Scenario:** Checking that major backend services (APIs) respond with the right data and status codes (e.g., fetching a user’s profile details or product stock).
- **Why Automate?:** API tests are quick, stable, and trap bugs before they reach the UI layer. They are the basis for microservices architecture.
- **Student Focus:** How to utilize tools (such as Postman or Python’s requests library) to make HTTP requests and check JSON/XML responses.

## Cross-Browser Regression Testing:

- **Scenario:** Executing a set of core tests (such as user login, site navigation) against several web browsers (Chrome, Firefox, Edge).
- **Why Automate?:** It is not possible to execute manually across five browsers for hundreds of tests. Automation promises an identical user experience across browsers.
- **Focus Area:** Configuring WebDriver environments and configuration management to dynamically switch between browsers.

Looking to get started with building your portfolio? Grab our compilation of **Real-World Automation Testing Project Ideas** to implement these ideas right away!

## FAQs About Automation Testing Tutorial for Beginners

**1. What is Automation Testing?**

Executing automated test cases with the help of software tools, comparing actual results with expected results, and reporting results, cutting down manual effort considerably.

**2. What are the skills needed for automation testing?**

Core competencies are knowledge of a programming language (Python/Java), comprehension of testing frameworks (Selenium, Cypress), SQL/APIs knowledge, and solid logical reasoning.

**3. What are the types of automation testing?**

Most common types are Unit Testing, API/Service Testing, UI Testing (End-to-End/E2E), Regression Testing, and Performance Testing.

**4. Can I learn testing in 3 months?**

You definitely learn basics and general tool skills in 3 months. Working as a good, job-ready tester involves hands-on practice and project engagement.

**5. Will AI replace automation testers?**

No, AI will revolutionize the role, not eliminate it. AI will take care of repeated test generation and maintenance so that testers can concentrate more on strategy, intricate scenario crafting, and analytical thinking.

**6. Is test automation easy?**

The fundamentals are simple to grasp, but automation is difficult to master. It involves good coding skills, debugging skills, and an intimate knowledge of software architecture.

**7. Is automation testing a good career in 2025?**

Yes, it is great. With cloud adoption and DevOps increasing, the need for SDETs (Software Development Engineers in Test) to create and handle test infrastructure is extremely high.

**8. What is the salary of automation tester in TCS?**

Salaries differ significantly based on experience and location of the role, but generally range from ₹4.5 lakhs to ₹12 lakhs+ per year for freshers to experienced professionals in India. Click here to know more about **Automation Tester Salary in TCS**.

**9. Which tool is best for automation testing?**

There isn't one "best" tool. Selenium/Appium are optimal for wide UI coverage. Cypress/Playwright are speedy for new web apps. "Best" varies with project technology.

**10. What is the future of Selenium?**

The future is bright and dynamic. Selenium is still the benchmark for cross-browser testing, ongoing improvements with new features (such as Selenium Manager) to keep up with newer frameworks.

## Conclusion

Having completed this automation testing tutorial successfully, you have embarked on the first significant step into software automation testing with valuable, marketable skills.Ready to create a full, scalable system? Join our full **Automation Testing Masterclass** today to make a smooth transition into an in-demand SDET position!