Selenium Tutorial for Beginners

Introduction

Are you tired of doing repetitive tasks that may include link checking, ad verification on different browsers, or competitor data scraping? Most beginners get overwhelmed with automation setup, such as drivers and dependencies. Selenium gives the power to automate these testing tasks efficiently. This Selenium tutorial simplifies the setup and shows how to use it for data gathering and QA. Click here to view the complete **Selenium Automation Course Syllabus**!

Why Students or Freshers Learn Selenium?

Learning Selenium will definitely be very beneficial for students and freshers entering the tech industry as it offers a direct path to in-demand skills:

- **High Demand for Automation:** due to the fact that companies in all sectors need test automation engineers to provide faster release cycles and high quality, Selenium skills are in demand.

- **Gateway to SDET Roles:** It's the main tool for an SDET, that is Software Development Engineer in Test, as this is the role that combines coding with quality assurance and thus is highly paid.
- **Cross-Browser Testing:** Selenium can run on all major web browsers (Chrome, Firefox, Edge, and Safari), hence is capable of executing web applications under any environment.
- **Free and Open-Source:** it boasts an enormous community support base and zero licensing cost, thereby making it the industry standard.
- **Flexibility in Language:** It supports several programming languages, like Java, Python, and C#. A learner can, therefore, use his or her already acquired knowledge in coding.

Ready to showcase your skills in an interview? Click here for a list of essential **Selenium Interview Questions and Answers**!

Step-by-Step Selenium Tutorial for Beginners

Selenium WebDriver is an open-source automation framework for web applications. It provides APIs that interact with the browser drivers to send commands to the browsers directly for performing any action. In this Selenium tutorial for beginners, we will set up a simple project and write our first automated test in Java using Selenium WebDriver.

Part 1. Installation and Setup: The Automation Environment

We will be using Java as the programming language, Maven as the build tool to manage dependencies, and TestNG as the testing framework.

Step 1: Install JDK and IDE

- **Prerequisite:** Selenium 4 needs Java 11 or later.
- **Action:** Install a stable JDK (e.g., Java 21).
- **IDE:** An Integrated Development Environment, such as IntelliJ IDEA or Eclipse with Maven support, should be installed.

Step 2: Create a Maven Project

Maven simplifies adding required libraries (dependencies) and structuring the project.

- Create a new Maven Project in your IDE.
- Provide a Group ID like *com.mycompany* and an Artifact ID like *selenium-starter*.
- Maven creates a default pom.xml file.

Step 3: Configure Dependencies in pom.xml

pom.xml is considered the heart of your project configuration. Dependencies to be added include Selenium WebDriver, TestNG-a testing framework, and WebDriverManager for handling browser drivers automatically. Provide the following within the <depencies> tag in your *pom.xml*:

```
<dependencies>

    <dependency>

        <groupId>org.seleniumhq.selenium</groupId>

        <artifactId>selenium-java</artifactId>

        <version>4.16.1</version>

    </dependency>

    <dependency>
```

```
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.8.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.github.bonigarcia</groupId>
        <artifactId>webdrivermanager</artifactId>
        <version>5.6.3</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Maven Sync: Save the file. Your IDE should automatically download these libraries and their transitive dependencies.

Part 2. Writing the First Test Case

We will write a simple test that navigates to a search engine, performs a search, and then verifies the title of the results page.

Step 4: Create the Test Class

Create a new Java class called *GoogleSearchTest.java* in the src/test/java directory of your standard Maven structure.

Step 5: Implement the Basic Structure with TestNG Annotations

We define setup, test execution, and clean-up steps using TestNG annotations: *@BeforeMethod, @Test, @AfterMethod.*

```
package com.mycompany.seleniumstarter;

import io.github.bonigarcia.wdm.WebDriverManager;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.Test;

import java.time.Duration;

public class GoogleSearchTest {

    // WebDriver is the main interface for testing
```

```
WebDriver driver;

// — SETUP METHOD —

@BeforeMethod

public void setup() {

    // Automatically sets up the driver (no manual download needed)

    WebDriverManager.chromedriver().setup();

    driver = new ChromeDriver();

    // Setting Implicit Wait: Global wait time for element to appear

    driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

    driver.manage().window().maximize();

}

// — TEST METHOD —

@Test

public void verifySeleniumSearchTitle() {

    String searchTerm = “Selenium WebDriver”;

    // 1. Navigate to the page

    driver.get(“https://www.google.com”);
```

```
    // 2. Locate the search input field
    // 'q' is the name attribute of the search box on Google
    WebElement searchBox = driver.findElement(By.name("q"));
    // 3. Perform the search action
    searchBox.sendKeys(searchTerm);
    searchBox.submit(); // Submits the form, equivalent to pressing Enter
    String actualTitle = driver.getTitle();
    System.out.println("Page Title: " + actualTitle);
    Assert.assertTrue(actualTitle.contains(searchTerm),
                "Assertion Failed: Page title does not contain the search term.");
}
@AfterMethod
public void tearDown() {
    if (driver != null) {
        driver.quit(); // Closes all browser windows opened by the WebDriver
    }
}
```

```
}
```

Part 3. Core Selenium Concepts (Locators and Commands)

Step 6: Understanding Locators

A Locator is the command to inform Selenium how to find an element (button, text box, link) on a web page. This is the most important part of reliable automation.

Locator Strategy	Method	When to Use	Example
ID	By.id(“username”)	Fastest, most reliable if a unique ID exists.	driver.findElement(By.id(“login-btn”))
Name	By.name(“q”)	Used when an ID is unavailable; often used for form fields.	driver.findElement(By.name(“password”))
ClassName	By.className(“error-msg”)	Used for elements sharing a common style/class (less specific).	driver.findElement(By.className(“header-logo”))
LinkText/PartialLinkText	By.linkText(“Click Here”)	Used to find hyperlink elements based on their visible text.	driver.findElement(By.linkText(“About Us”))

CSS Selector	By.cssSelector(“div.main input#id”)	Powerful and fast; uses CSS rules to locate elements.	driver.findElement(By.cssSelector(“#search-input”))
XPath	By.xpath(“//input[@type=’submit’]”)	Most flexible but slowest; used when other locators fail.	driver.findElement(By.xpath(“//div[2]/button”))

Step 7: Key WebDriver Commands

These are the basic language of interaction between your script and the browser.

Command Type	Example	Description
Browser Actions	driver.get(url)	Opens the specified URL.
	driver.getTitle()	Retrieves the current page title (used in our assertion).
	driver.quit()	Closes the browser and terminates the WebDriver session.
Element Input	element.sendKeys(“text”)	Types data into an input field (used for the search term).
	element.clear()	Clears the text from an input field.

Element Interaction	element.click()	Clicks on a button, link, or radio button.
	element.submit()	Submits the form containing the element.
Element Status	element.isDisplayed()	Returns true if the element is visible on the page.
	element.isEnabled()	Returns true if the element is interactable.

Part 4. Handling Synchronization (Waits)

Web applications load their elements at different speeds, which causes scripts of automation to fail with the **“Element Not Found”** error. **Waits** solve this problem.

Step 8: Implement Synchronization Strategies

Implicit Wait (Global Setting): It sets a default timeout for *driver.findElement()* calls. If the element is not instantly available, then the driver will wait for the defined time before throwing an exception.

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

Explicit Wait (Conditioned): This command tells the WebDriver to wait for a specific condition. Most of the time, it’s used to handle the AJAX elements or dynamic content.

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));

WebElement element =

wait.until(ExpectedConditions.elementToBeClickable(By.id(“dynamic-btn”)));

element.click();

Note: You need to import *org.openqa.selenium.support.ui.WebDriverWait* and *org.openqa.selenium.support.ui.ExpectedConditions* to use Explicit Waits.

Part 5. Running and Analyzing the Test

Step 9: Execute the Test

Since we are using TestNG, execution is straightforward:

- **IDE:** In your IDE, right-click the *GoogleSearchTest.java* file and select **"Run."**
- This script will open the Chrome browser, perform the actions, and then close the browser.
- **Result:** The console in the IDE will display the TestNG report of whether the method *verifySeleniumSearchTitle,* **PASSED or FAILED.**

Step 10: Troubleshooting Failures

If the test fails, then the primary reason is almost always related to the Locator or Synchronization.

- **Locator Failure:** The By strategy you used is wrong or the element id has changed. Re-inspect the element in the Browser developer tools.
- **Synchronization Failure:** The script tried to interact with an element before it was fully loaded – for example, trying to click a button that hasn't finished rendering. Use a more specific Explicit Wait instead of just relying on the Implicit Wait.

The next steps will involve mastering the design patterns such as the POM, which allows code reusability, handling complex web elements like frames and alerts, and integration of your tests into Continuous Integration/Continuous Deployment pipelines.

Is your team struggling with test execution instability, multiple test data sets to manage, or advanced Page Object Model implementations? Click here to view our **Selenium Challenges and Solutions for Beginners**!

Real Time Examples for Selenium Testing Tutorial for Learners

Following are three practical and real-time automation projects using Selenium WebDriver to create a portfolio and showcase in-demand skills:

E-commerce End-to-End Purchase Flow Test (Page Object Model)

Objective: Automate the end-to-end user journey on a dummy e-commerce site, searching for a product, adding it to the cart, filling in shipping details, and confirming that the final order confirmation page is displayed.

Skills Demonstrated:

- **POM:** Using the POM design pattern in order to create reusable code for pages such as HomePage, CartPage, and CheckoutPage.
- **Synchronization:** Handling dynamic elements, such as loading spinners and pop-up modals, using Explicit Waits to maintain test stability.
- **Assertions:** Checking if the total price computed in the cart is the same as the price on the confirmation page via TestNG/JUnit Assertions.

Real-World Application: It's essential for QA teams to ensure that the core revenue-generating path of any online business is always functional before every release.

Broken Link and Image Verification Script (Data Extraction)

Objective: Write a script to crawl all links in a specific page or section of the website as quickly as possible and check their status.

Skills Demonstrated:

- **Finding Multiple Elements:** Using driver.findElements(By.tagName(“a”)) to retrieve the list of all hyperlink elements.
- **HTTP Status Check:** Iterating over the list of links, extracting the href attribute, and using Java networking libraries-like HttpURLConnection-to verify whether the link returns a healthy HTTP Status Code 200 or a broken code like 404 (Not Found).
- **Reporting:** Log the results (working/broken) to a simple CSV file or console output.

Real-World Application: This is a critical maintenance task of SEO and QA teams to prevent dead ends and maintain website health across large sites.

Competitive Price Monitoring/Data Scraping – Cross-Browser

Objective: Automate the process of logging in to a mock competitor website, extract the price of a particular product, and report it every day.

Skills Demonstrated:

- **Data Extraction:** Using specific locators like XPath or CSS Selectors to precisely target dynamic data on a web page, such as the product price.
- **Cross-Browser Testing:** The script should be configured to run the same test on different browsers, such as Chrome and Firefox, by utilizing browser-specific drivers that maintain consistent behavior.
- **Data Handling:** Store the extracted price data in a list or array, to be analyzed later or stored in a database.

Real-World Application: Heavy usage in marketing, business intelligence, and finance for competitive intelligence analysis and market surveillance.

Ready to turn automation concepts into high-value portfolio projects? Click here for a list of detailed **Selenium Project Ideas** along with setup guides!

FAQs About Selenium Tutorial for Beginners

1. How to learn Selenium testing tool for beginners?

Master a programming language and understand locators: Java or Python. Learn about XPath, CSS, then move on to learning WebDriver commands. Install dependencies using Maven/Gradle. Practice Synchronization: Explicit waits and Implicit Waits. Also, implement the Page Object Model.

2. Is Selenium TDD or BDD?

Selenium in itself is not TDD or BDD; it is simply an automation tool. It can, however, be used within both frameworks. BDD frameworks will often use Selenium in conjunction with other tools like Cucumber or SpecFlow to run your feature files.

3. What are Selenium basics?

Selenium basics would be environment setup: JDK, IDE, Maven; an idea of the WebDriver API; basic browser navigation (driver.get()); finding elements with Locators (By.id, By.xpath); performing actions upon them (.click(),.sendKeys()).

4. What are the 4 parameters of Selenium?

This would more often refer to the core components of the older Selenium 1.0 (Selenium RC) architecture. In the modern Selenium WebDriver (Selenium 4) context, the four key concepts are the WebDriver API, Browser Drivers (Chrome / Firefox), Locators, and Synchronization (Waits).

5. Can I learn Selenium in 1 month?

Well, you can learn the basics about Selenium WebDriver-including set up, locators, basic commands-in a month or so, assuming you already know some programming knowledge. But advanced concepts like POM, framework design, and CI/CD integrations take more time to learn.

6. What is the salary of a Selenium tester?

In general, Selenium Testers, who are often titled as Automation Engineers or SDET, have high salaries. Average salaries in the US range from $75,000 to over $120,000 USD per year depending on experience, location, and specific programming language expertise. Explore more with this **Selenium Tester Salary for Freshers**.

7. Is Selenium still in demand?

Yes, Selenium is still in high demand and is considered the industry standard for web automation testing. Although Cypress and Playwright are new tools that show significant growth, the huge existing infrastructure and wide integration support keep Selenium relevant and in demand.

8. Which language is fastest for Selenium?

The speed of an automation script is usually governed by browser performance and network latency and not the language itself. But yes, Java and Python are two of the most used languages with Selenium, and in enterprise environments, Java is usually preferred.

9. Is Selenium an AI tool?

No, Selenium is not an AI tool. It's a traditional automation framework that acts based on explicit instructions provided in your code. It requires a human to program it for identifying elements and defining test logic. Sometimes AI tools help with Selenium for smart locator generation.

10. How many months to learn Selenium?

In this respect, it takes 3-6 months of continuous learning and project practice to reach a strong professional level in framework development, POM implementation, and CI/CD. Proficiency often boils down to hands-on experience in solving real-world challenges.

Conclusion

Congrats! You have performed the environment setup, and ran your first test using the Selenium WebDriver. Now you know how Locators, Synchronization (Waits), and basic WebDriver API-the foundation skills of any automation profile-fit together. You can now write fast, reliable, and reusable test scripts. Are you ready to take a lead in advancing your career by mastering Page Object Model, Framework Design, and advanced Data-driven Testing? Enroll in our **Selenium Testing Course in Chennai** now and become a certified Selenium Automation Engineer!