

Selenium Basics to Mid Level

Selenium is the most asked automation tool in fresher interviews. Examples use Java, the most common language with Selenium.

1. What is automation testing?

In manual testing a person clicks and types. In **automation testing** we write a program (a script) that does the clicking and typing for us, and checks the result by itself.

- **Good for:** tests that repeat again and again (regression), tests with lots of data, tests that must run at night.
- **Not good for:** a feature that changes every day, tests that run only once, checking how the design "feels".

2. What is Selenium?

Selenium is a free tool that controls a web browser (Chrome, Firefox, Edge) through code. It works only for **web** apps. Main parts:

PART	USE
WebDriver	The main part. Code you write to control the browser.
Grid	Runs tests on many browsers and machines at the same time.
IDE	Browser plugin to record and play tests. Good for learning, not for real projects.

How it works



Your code sends a command like "click this button". The driver passes it to the browser. The browser does it and sends back the result.

3. Your first script

You add the Selenium library using Maven (a tool that downloads libraries). In `pom.xml`:

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>4.x.x</version>
</dependency>
```

What it does: tells Maven to download Selenium so you can use it. Selenium 4 also downloads the browser driver automatically.

```
WebDriver driver = new ChromeDriver();
driver.get("https://example.com/login");
driver.findElement(By.id("username")).sendKeys("testuser");
driver.findElement(By.id("password")).sendKeys("Pass@123");
driver.findElement(By.cssSelector("button[type='submit']")).click();
Assert.assertEquals(driver.getTitle(), "Dashboard");
driver.quit();
```

Line by line: open Chrome → go to the login page → type in the username box → type in the password box → click the Login button → check the page title is "Dashboard" (test passes or fails here) → close the browser.

4. Locators: how Selenium finds things on a page

A web page is made of **elements** (buttons, text boxes, links). To click a button, Selenium must first find it. The way of finding is called a **locator**.

LOCATOR	EXAMPLE	NOTE
id	<code>By.id("email")</code>	Best choice. Fast and usually unique.
name	<code>By.name("q")</code>	Good if id is missing.
className	<code>By.className("btn")</code>	Many elements may share it.
tagName	<code>By.tagName("a")</code>	Used to get lists, like all links.

linkText	<code>By.linkText("Forgot password?")</code>	For links, full visible text. Also <code>partialLinkText</code> .
cssSelector	<code>By.cssSelector("input#email")</code>	Fast and flexible.
xpath	<code>By.xpath("//button[text()='Login']")</code>	Most powerful. Can find by text and parent/child.

XPath and CSS cheat sheet

```
//input[@id='email']           // XPath: input whose id is email
//button[text()='Login']       // XPath: button with exact text Login
//div[contains(@class,'error')] // XPath: class contains the word error
input#email                    // CSS: input with id email
input.form-ctrl                // CSS: input with class form-ctrl
button[type='submit']          // CSS: button with attribute type=submit
```

Absolute XPath (`/html/body/div[1]/...`) breaks when the page changes a little. **Relative XPath** (starts with `//`) is safer. Prefer id, then CSS, then relative XPath.

To get a locator: right-click the element in Chrome → Inspect → look at its id/class, or right-click the HTML → Copy → Copy selector / XPath.

5. Common actions on elements

```
WebElement box = driver.findElement(By.id("search"));
box.sendKeys("laptop");
box.clear();
driver.findElement(By.id("go")).click();
String msg = driver.findElement(By.id("msg")).getText();
boolean shown = driver.findElement(By.id("msg")).isDisplayed();
```

What it does: find the search box and store it → type "laptop" → clear the text → click the Go button → read the visible text of the message → check whether the message is visible (true/false). Also available: `isEnabled()`, `isSelected()` (for checkbox/radio), `getAttribute("value")`.

```
List<WebElement> rows = driver.findElements(By.tagName("tr"));
System.out.println(rows.size());
```

What it does: `findElement` returns **one** element and throws an error if not found.

`findElements` (with s) returns a **list**, and an empty list if nothing is found. Here we count the table rows.

Browser commands

```
driver.navigate().back();
driver.navigate().refresh();
driver.manage().window().maximize();
driver.getCurrentUrl();
```

What it does: go back one page, reload the page, make the window full size, and read the current URL.

6. Waits (very important, always asked)

Pages take time to load. If Selenium looks for a button before it appears, you get

`NoSuchElementException`. A **wait** tells Selenium to wait for the element.

WAIT	HOW IT WORKS
Implicit	One global setting. For every find, wait up to N seconds before failing.
Explicit	Wait for one specific condition on one element (visible, clickable). Most recommended.
Fluent	Like explicit, but you also set how often to check (polling) and which errors to ignore.
Thread.sleep	Fixed pause. Avoid it: it always waits the full time even if the page is ready.

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));
```

What it does: sets an implicit wait of 5 seconds for the whole test.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("msg")));
```

What it does: waits up to 10 seconds, but moves on **as soon as** the message appears. If it never appears, the test fails with `TimeoutException`. Another common one: `elementToBeClickable`.

7. Special situations

Dropdown

```
Select country = new Select(driver.findElement(By.id("country")));
country.selectByVisibleText("India");
```

What it does: wraps the dropdown in the `Select` class and picks the option by the text you see. Other ways: `selectByValue`, `selectByIndex`. Works only for dropdowns made with the `<select>` tag.

Alert popup

```
Alert alert = driver.switchTo().alert();
alert.accept();
```

What it does: moves control to the browser popup and clicks OK. Use `dismiss()` for Cancel, `getText()` to read the message.

Frame (a page inside a page)

```
driver.switchTo().frame("payFrame");
driver.switchTo().defaultContent();
```

What it does: go inside the frame to work with its elements, then come back to the main page.

New tab or window

```
String main = driver.getWindowHandle();
for (String h : driver.getWindowHandles()) driver.switchTo().window(h);
```

What it does: each window has an ID called a handle. We save the main window's ID, then loop through all open windows and switch to them. Switch back with `driver.switchTo().window(main)`.

Mouse actions

```
new Actions(driver).moveToElement(menu).perform();
```

What it does: moves the mouse over an element (hover), for example to open a menu.

Actions also does double-click, right-click and drag-and-drop. Always end with **perform()**.

Screenshot

```
File shot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
```

What it does: takes a screenshot, normally saved when a test fails as proof.

8. TestNG: running and checking tests

Selenium only controls the browser. A test framework like **TestNG** (or JUnit) organizes tests and reports pass/fail.

```
public class LoginTest {
    WebDriver driver;

    @BeforeMethod
    public void setUp() { driver = new ChromeDriver(); }

    @Test
    public void validLogin() {
        driver.get("https://example.com/login");
        // ... steps ...
        Assert.assertEquals(driver.getTitle(), "Dashboard");
    }

    @AfterMethod
    public void tearDown() { driver.quit(); }
}
```

What it does: **@BeforeMethod** runs before every test (open browser). **@Test** marks a test.

@AfterMethod runs after every test (close browser). **Assert** compares actual and expected; if they differ, the test fails.

ANNOTATION / TERM	MEANING

@BeforeClass / @AfterClass	Run once before / after all tests in the class.
@DataProvider	Runs the same test with many sets of data.
testng.xml	File to choose and group which tests to run.
Assert vs verify	Hard assert stops the test at first failure. Soft assert (<code>SoftAssert</code>) continues and reports all failures at the end.

9. Page Object Model (POM)

If every test has locators written inside it, one UI change means editing many tests. In **POM** each page gets its own class that holds its locators and actions. Tests only call these actions.

```
public class LoginPage {
    WebDriver driver;
    By user = By.id("username");
    By pass = By.id("password");
    By btn = By.cssSelector("button[type='submit']");

    public LoginPage(WebDriver driver) { this.driver = driver; }

    public void login(String u, String p) {
        driver.findElement(user).sendKeys(u);
        driver.findElement(pass).sendKeys(p);
        driver.findElement(btn).click();
    }
}
```

What it does: the test now just writes `new LoginPage(driver).login("testuser", "Pass@123");`. If the username id changes, you fix it in **one** place. Benefits: less repeated code, easy maintenance, tests are easy to read.

Typical project structure

```
project/
├─ pom.xml           (libraries)
├─ testng.xml        (which tests to run)
├─ src/
│   ├─ main/java/pages/  LoginPage.java, HomePage.java
│   ├─ main/java/utils/  BrowserSetup, WaitUtils, ExcelReader
│   └─ test/java/tests/  LoginTest.java, SearchTest.java
```

10. Java you must know for Selenium interviews

Interviewers ask Java along with Selenium. These topics come up most, shown with where you use them in automation.

CONCEPT	SIMPLE MEANING	WHERE IN AUTOMATION
Class and object	Class is a blueprint, object is a real thing made from it.	<code>LoginPage</code> is a class, <code>new LoginPage(driver)</code> is an object.
Encapsulation	Hide data, give access through methods.	Locators are <code>private</code> in a page class. Tests call public methods.
Inheritance	A child class reuses the parent's code (<code>extends</code>).	All tests <code>extend BaseTest</code> to reuse browser setup and close.
Polymorphism	One thing, many forms. Overloading: same method name, different parameters. Overriding: child changes the parent's method.	<code>WebDriver driver = new ChromeDriver();</code> can later be Firefox with no other change.
Abstraction	Show what it does, hide how.	<code>WebDriver</code> is an interface. You call <code>click()</code> without knowing the browser internals.

Interface vs abstract class	Interface: only the list of methods (a contract). Abstract class: some methods ready, some left to the child.	<code>WebDriver</code> , <code>WebElement</code> are interfaces.
= vs <code>equals()</code>	= compares memory address. <code>equals()</code> compares the content of strings.	Comparing page text: use <code>equals()</code> .
static, final	static: belongs to the class, not an object. final: cannot be changed.	A shared <code>static WebDriver</code> ; constants like timeout values.
Collections	List (ordered, duplicates allowed), Set (no duplicates), Map (key-value).	<code>List<WebElement></code> for many elements; Map for test data.
Exceptions	<code>try / catch / finally</code> handles errors. Checked: forced by compiler. Unchecked: at run time.	Handling <code>NoSuchElementException</code> ; <code>finally</code> to close the browser.
String methods	<code>length</code> , <code>substring</code> , <code>contains</code> , <code>equals</code> , <code>equalsIgnoreCase</code> , <code>split</code> , <code>trim</code>	Cleaning and checking page text.

```

public class BaseTest {
    protected WebDriver driver;
    @BeforeMethod public void setUp() { driver = new ChromeDriver(); }
    @AfterMethod public void tearDown() { driver.quit(); }
}

public class LoginTest extends BaseTest {
    @Test public void validLogin() { driver.get("https://example.com"); }
}

```

What it does: `BaseTest` holds the common start and close code once. `LoginTest` extends `BaseTest` gets it for free (inheritance), so each new test class stays short.

11. More Selenium topics

Checkbox and radio button

```
WebElement box = driver.findElement(By.id("agree"));
if (!box.isSelected()) box.click();
```

What it does: ticks the checkbox only if it is not already ticked, so the test works every time.

Tables

```
String cell =
driver.findElement(By.xpath("//table[@id='orders']//tr[2]/td[3]")).getText();
```

What it does: reads the text from row 2, column 3 of the table. To read all rows, use `findElements` on `//table[@id='orders']//tr` and loop.

File upload

```
driver.findElement(By.id("resume")).sendKeys("/Users/me/resume.pdf");
```

What it does: for an `<input type="file">`, you do not click. You just send the file path as text.

JavaScript Executor

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("arguments[0].scrollIntoView(true);", element);
js.executeScript("arguments[0].click();", element);
```

What it does: runs JavaScript inside the page. First line scrolls to the element. Second line clicks it using JavaScript, a last option when the normal click is blocked by another element.

Headless browser

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless=new");
WebDriver driver = new ChromeDriver(options);
```

What it does: runs Chrome without showing a window. It is faster and is used on servers and CI machines.

New tab, cookies, relative locators (Selenium 4)

```
driver.switchTo().newWindow(WindowType.TAB);
driver.manage().deleteAllCookies();
driver.findElement(with(By.tagName("input")).below(By.id("email")));
```

What it does: opens a new tab and moves to it, clears all cookies (useful to start fresh), and finds an input that is **below** the email field. Relative locators find elements by their position.

Dynamic elements

- If an id changes on each load (like `btn_8374`), use `//button[starts-with(@id,'btn_')]` or `contains()`.
- Prefer stable attributes: `data-testid`, `name`, visible text, or relationship to a nearby element.
- Elements that load late: use an explicit wait.

12. Common exceptions and fixes

EXCEPTION	WHY IT HAPPENS	FIX
NoSuchElementException	Element not found: wrong locator, or not loaded yet, or inside a frame.	Check the locator, add explicit wait, switch to frame.
StaleElementReferenceException	The page refreshed and the saved element is old.	Find the element again before using it.
ElementClickInterceptedException	Another element (popup, banner) covers the button.	Close the cover, wait for it to go, scroll, or use JS click.
ElementNotInteractableException	Element is present but hidden or disabled.	Wait until it is visible or clickable.

TimeoutException	Explicit wait condition never became true.	Check the condition and locator, increase time if needed.
NoSuchFrameException / NoAlertPresentException	Wrong frame name, or no popup is open.	Check the frame, wait for the alert.
SessionNotCreatedException	Browser and driver versions do not match.	Update the browser or driver (Selenium Manager usually handles it).

13. Building a real framework

Data-driven testing with TestNG

```
@DataProvider(name = "logins")
public Object[][] data() {
    return new Object[][] { {"user1","pass1"}, {"user2","wrong"} };
}

@Test(dataProvider = "logins")
public void loginTest(String u, String p) { /* use u and p */ }
```

What it does: the test runs once for each row of data. Real projects read data from Excel (Apache POI library), CSV or JSON files.

Parts of a good framework

PART	TOOL OR IDEA
Build and libraries	Maven or Gradle (<code>pom.xml</code>)
Test runner	TestNG or JUnit
Design pattern	Page Object Model
Config	<code>config.properties</code> file for URL, browser, timeouts. Never hard-code them.

Reports	Extent Reports, Allure, or TestNG's default report. Add a screenshot on failure.
Logging	Log4j, to see what happened step by step.
Cross-browser and parallel	Pass browser name as a parameter. TestNG <code>parallel</code> option or Selenium Grid.
Version control	Git and GitHub
CI/CD	Jenkins or GitHub Actions run the suite automatically on each build or every night.

BDD with Cucumber (just the idea)

BDD means writing tests in plain English that business people can read. Steps are then connected to Selenium code.

```
Feature: Login
  Scenario: Valid user logs in
    Given I am on the login page
    When I enter valid username and password
    Then I should see the dashboard
```

What it does: this is Gherkin syntax: **Given** (starting point), **When** (action), **Then** (expected result). Each line matches a Java method called a step definition.

14. Flaky tests

A **flaky test** passes sometimes and fails sometimes, without any code change. It destroys trust in automation.

- **Common causes:** wrong or missing waits, tests that depend on each other, shared or old test data, unstable locators, slow environment, popups appearing at random.
- **Fixes:** use explicit waits, make every test independent, create fresh data for each test, use stable locators, retry only as a last option.

15. Selenium vs Playwright vs Cypress

SELENIUM	PLAYWRIGHT	CYPRESS
----------	------------	---------

Languages	Java, Python, C#, JS and more	JS/TS, Python, Java, C#	JavaScript / TypeScript only
Browsers	Chrome, Firefox, Edge, Safari	Chromium, Firefox, WebKit	Chrome family, Firefox
Waiting	You add waits yourself	Built-in auto-wait	Built-in auto-retry
Speed and stability	Slower, more setup	Fast, stable	Fast, runs inside the browser
Best known for	Oldest, biggest community, most jobs	Modern, multi-tab, API and UI together	Developer-friendly UI test runner

Tip: learn Selenium well first. The ideas (locators, waits, POM, assertions) are the same in the other two, so they are quick to pick up later.

16. Quick interview answers

- **What is Selenium?** Free open-source tool to automate web browsers. It does not support desktop or mobile apps (mobile uses Appium).
- **findElement vs findElements?** One element (error if missing) vs a list (empty if missing).
- **driver.close() vs driver.quit()?** `close` closes the current window. `quit` closes all windows and ends the session.
- **Which locator do you prefer?** id first, then CSS, then relative XPath.
- **XPath vs CSS?** XPath can find by text and go to parent elements. CSS is usually faster and simpler.
- **Implicit vs explicit wait?** Implicit is global for all elements. Explicit waits for one condition on one element.
- **How do you handle dynamic elements whose id changes?** Use `contains()` or `starts-with()` in XPath, or use another stable attribute.
- **What is POM?** A design pattern: one class per page holding its locators and actions.
- **Challenges of automation?** Changing UI, slow or flaky pages, wrong waits, test data problems.

- **Which tests would you automate first?** Repeated and stable ones: smoke and regression tests of main flows.
- **What are the limitations of Selenium?** Only web apps, no built-in reports, no image or captcha handling, needs waits and extra libraries.
- **How do you handle StaleElementReferenceException?** Find the element again after the page refreshes.
- **What is a flaky test and how do you fix it?** A test that passes and fails randomly. Fix waits, locators and test data.
- **How do you run tests in parallel or on many browsers?** TestNG parallel execution or Selenium Grid.
- **What is headless mode?** Running the browser without a visible window, faster and used in CI.
- **How do you handle a file upload?** `sendKeys` with the file path on the `input type="file"` element.
- **Overloading vs overriding?** Overloading: same method name, different parameters, in one class. Overriding: child class gives a new body to a parent method.
- **Why is WebDriver an interface?** So the same code can work for any browser class (Chrome, Firefox) that implements it.
- **What framework did you use or know?** Data-driven TestNG + Maven + Page Object Model + reports + Git, run through Jenkins.
- **Selenium vs Playwright?** Playwright has built-in waiting and is faster; Selenium has the bigger community and more language support.