

FULL STACK

## Automation with Selenium WebDriver



# You Already Know

## Course(s):

Software Testing: Get a high paying Job In Technology



- Explain software testing and software tester
  - Software testing
  - Software tester
- Describe methodologies used in software testing
  - Software Development Life Cycle (SDLC)
  - Waterfall and Agile methodologies
  - Software Testing Life Cycle (STLC)
- List the differences between manual testing and automation testing
  - Manual vs. Automation



## Recap

- Demonstrate bugs in a software
  - Bugs
  - Bug tool





# A Day in the Life of a Test Engineer

Joe, a Test Engineer for Abq Inc., has been working hard. His company is facing employee bandwidth problem in testing a project given by an e-learning company. The company has decided to use automation testing to solve this problem.

Joe has been asked to design a functionality to automate testing using Selenium.

In this lesson, we will learn how to solve this real-world scenario to help Joe effectively and quickly complete his task.



## Learning Objectives

By the end of this lesson, you will be able to:

- 👁 Work with Selenium IDE and WebDriver
- 👁 Develop automation scripts for application
- 👁 Work with radio buttons, drop-down list, multi-select, and checkboxes
- 👁 Work with external elements like frames, alerts, pop-ups, and sub-windows



# FULL STACK

## Testing Basics

# Software Testing

Software testing is the process of evaluating a system or its components to ensure that the software system is defect free.

Testing: Checking if the tangible result matches the projected or the expected output.

High analytical skills required to test an application for all possible use cases with minimum test cases.



# Why Software Testing

Software testing checks the quality of a software application for its:

Functionality

Efficiency

Reliability

Usability

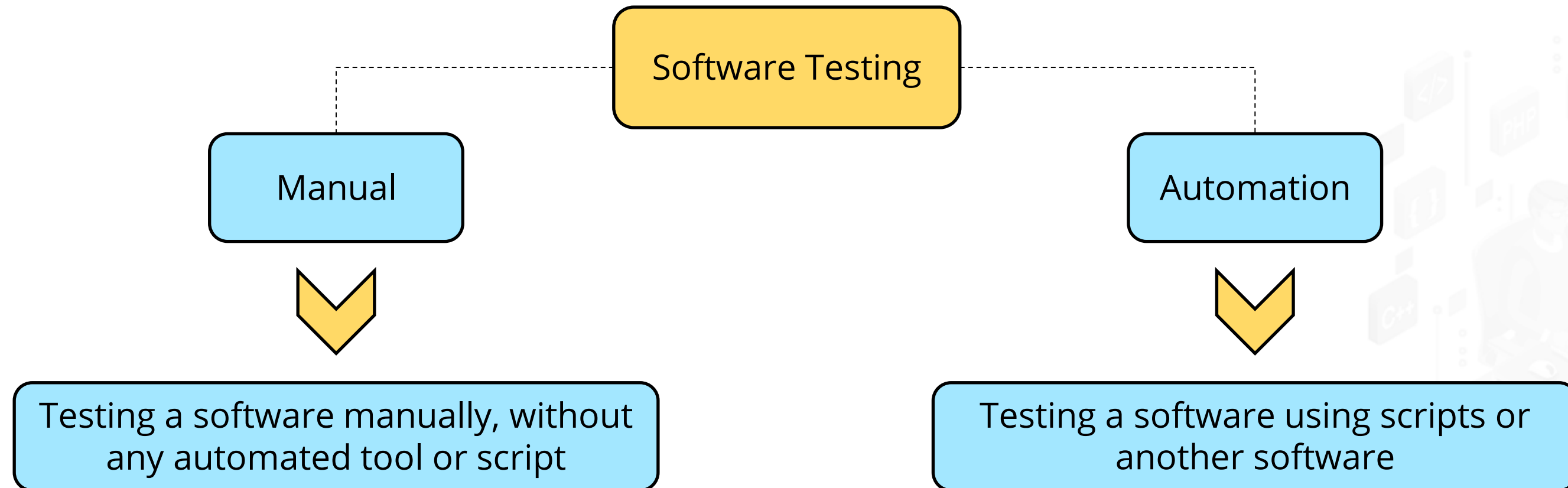
Maintainability

Portability



# Types of Testing

Software testing can be broadly classified as manual testing and automation testing.



# Manual Testing

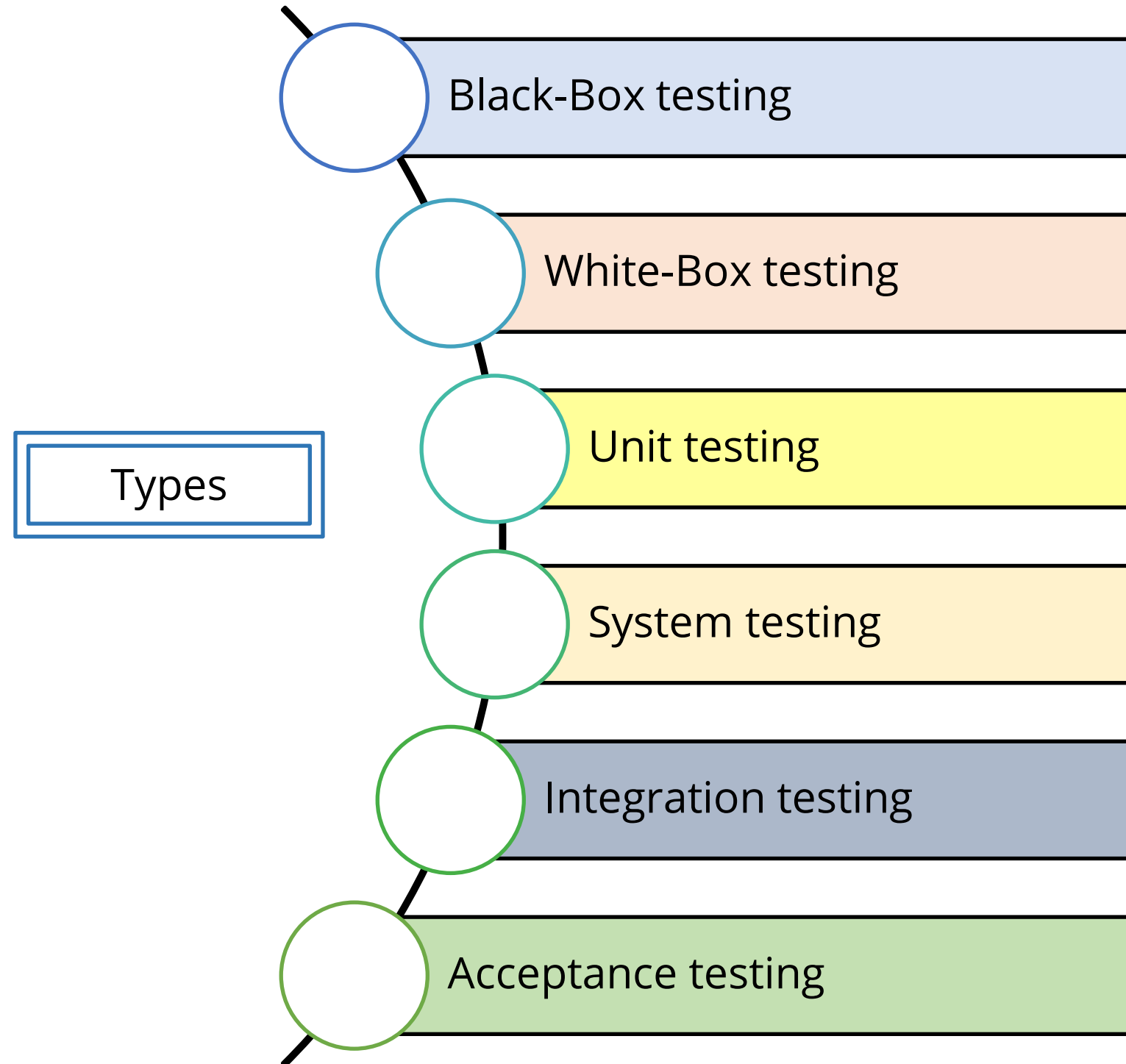
Includes the most primitive types of testing

A new application must be tested manually before automating the testing process

Test cases are manually executed



# Manual Testing: Types



# Automation Testing

Uses a third-party software to test the application or product

Involves automation of a manual process

Used mainly in load, performance, and stress testing

Saves time, improves accuracy, and increases test coverage





# Manual Testing vs. Automation Testing

Manual Testing	Automation Testing
Requires human intervention for test execution	Uses tools to execute test cases
Time consuming and labour intensive	Saves time, cost, and manpower
Any application can be tested manually	Recommended only for stable systems
Not as accurate as automation testing	Reliable as testing is performed by scripts and tools
Not cost effective for high-volume regression	Not cost effective for low-volume regression
Suitable when the test cases need to be executed for a limited number of times	Suitable for frequent execution of same set of test cases

# Real-Time Implementation of Testing Techniques: HR Payroll System

---

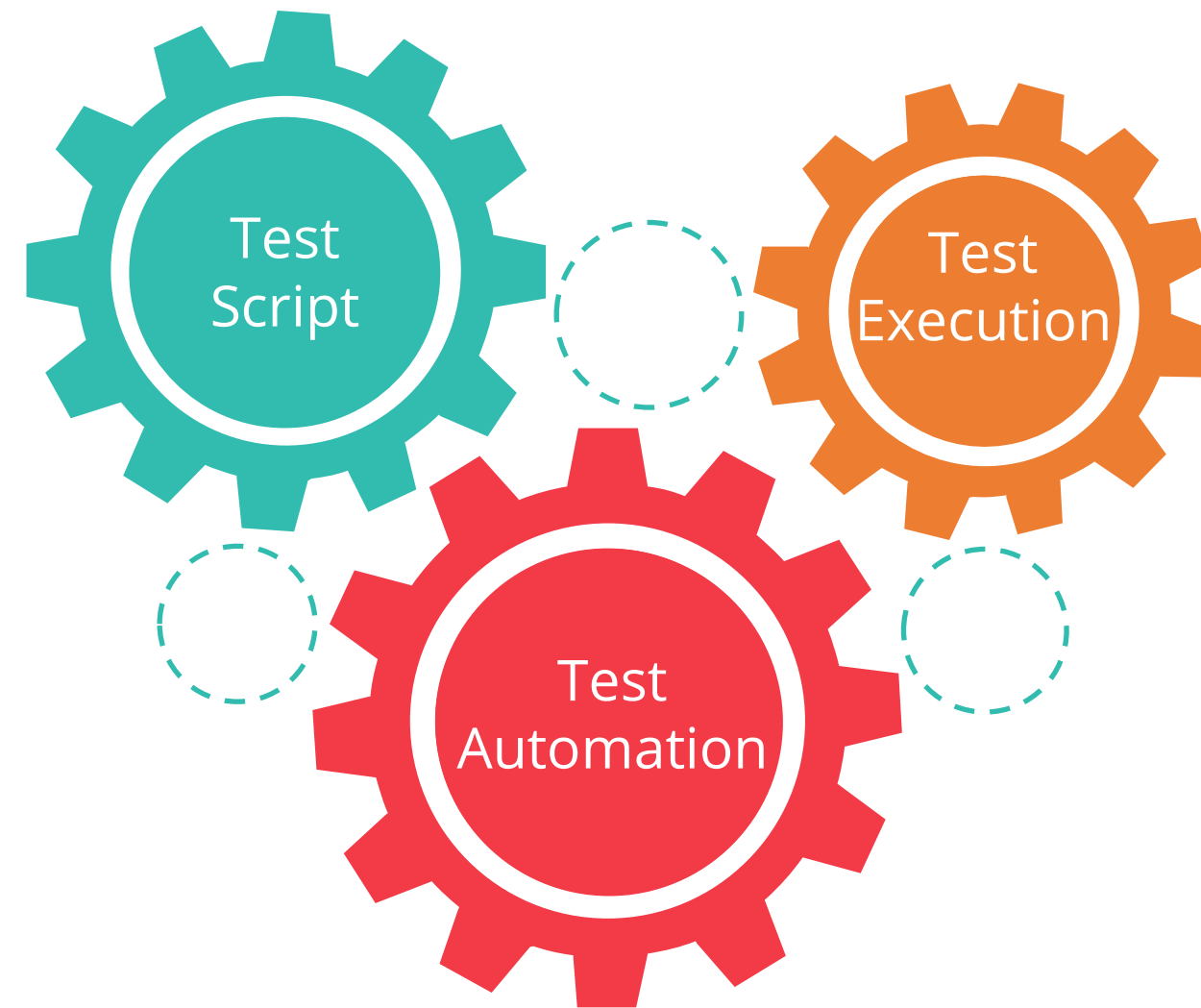


# FULL STACK

## What is Selenium and How it is Used in the Industry

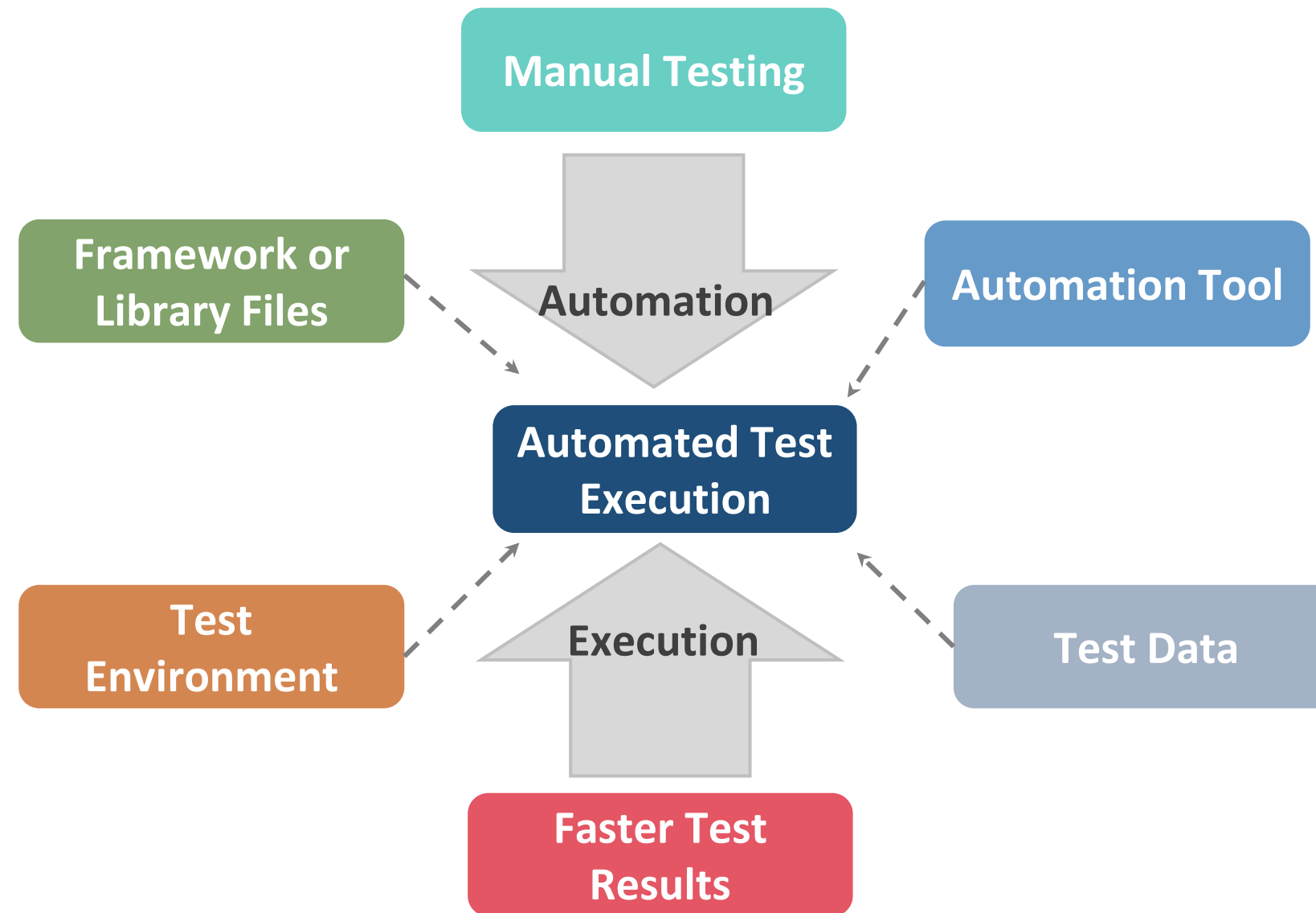
# What Is Automation?

Software test automation refers to the activities and efforts that automate manual tasks and operations in a software test process using well-defined strategies and systematic solutions.



# Automation for Agility

Test automation is the only way to achieve agility.





# Automation Lifecycle

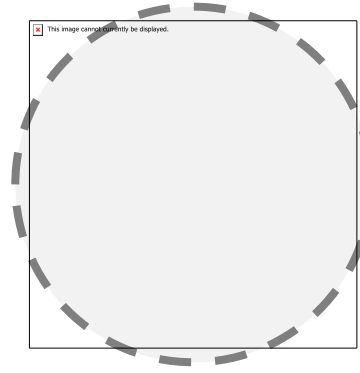


# Automation Tools

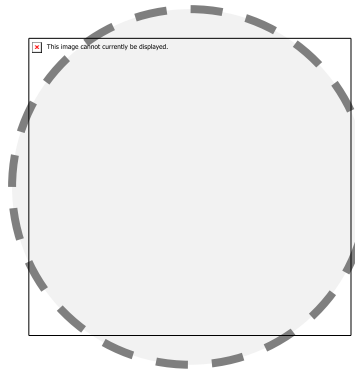
There are various Test Automation tools available and each has its own advantages and limitations. Some of the tools are:



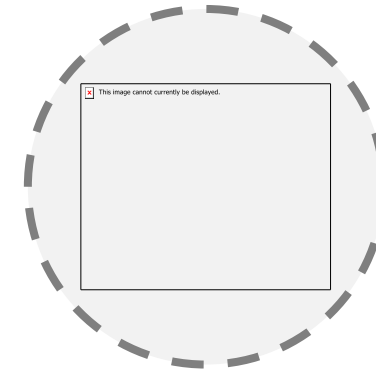
Selenium  
Testing



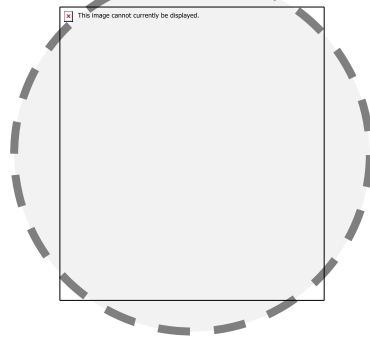
Quick Test  
Professional



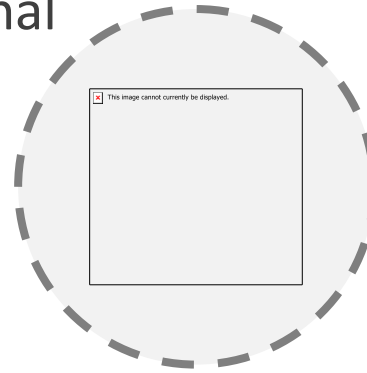
Appium



FitNesse



Ranorex



Silk Test



Eggplant



TestComplete



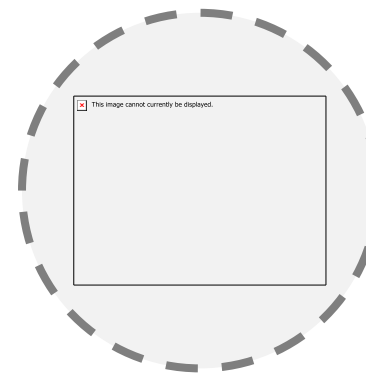
Rational  
Functional Test



Watir

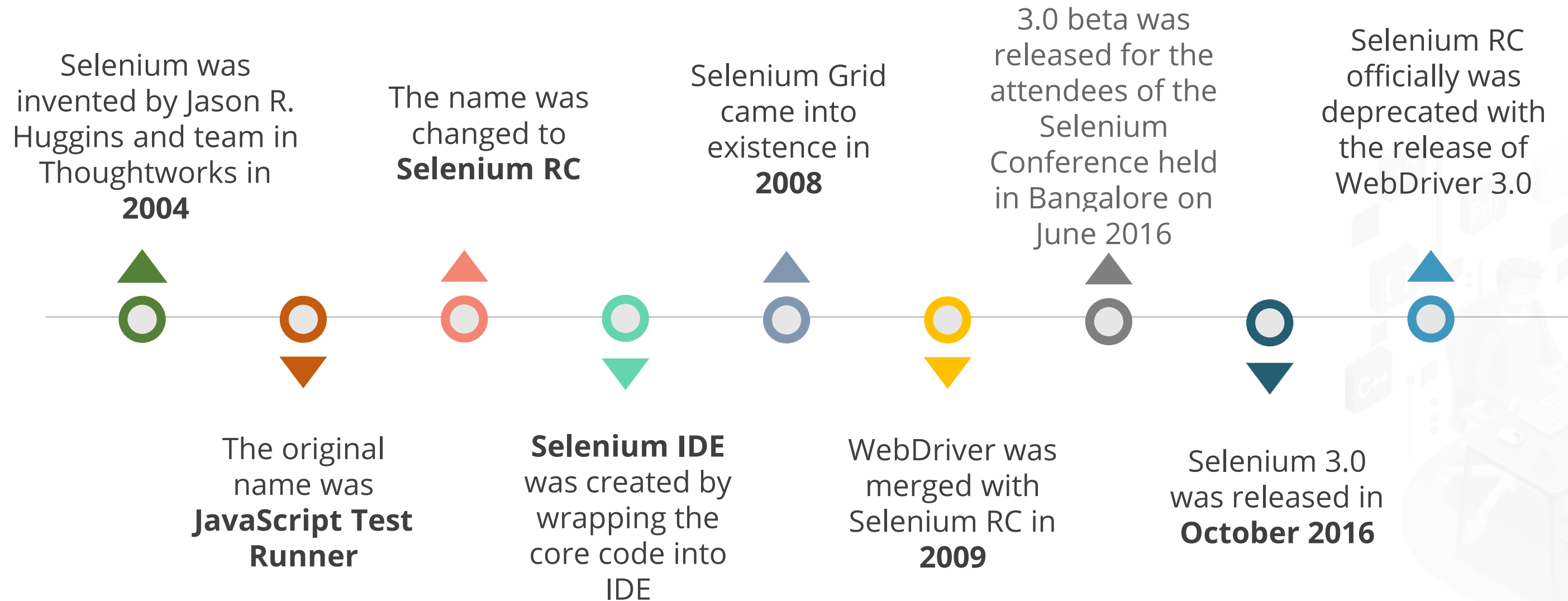


Tricentis Tosca



SOATest

# Brief History of Selenium



# Uses of Selenium in the Industry



# FULL STACK

## Features of Selenium



# Features of Selenium IDE

Record and playback

Autofilling locators: ID, name, link, XPath, CSS, and DOM

Dropdown selection of Selenium commands

Features of debugging tests

Capability to save tests as HTML and convert to WebDriver Java, Ruby, Python, and C#

Support for Selenium user-extensions.js file

Scheduler for scheduling your tests

# Adding Selenium IDE to the Browser

- Installing IDE ([from addons.mozilla.org](http://addons.mozilla.org))

- Various menu options

- Test Cases and Test Suites

- Commands or Targets or Value

- Assertions, Accessors, and Actions

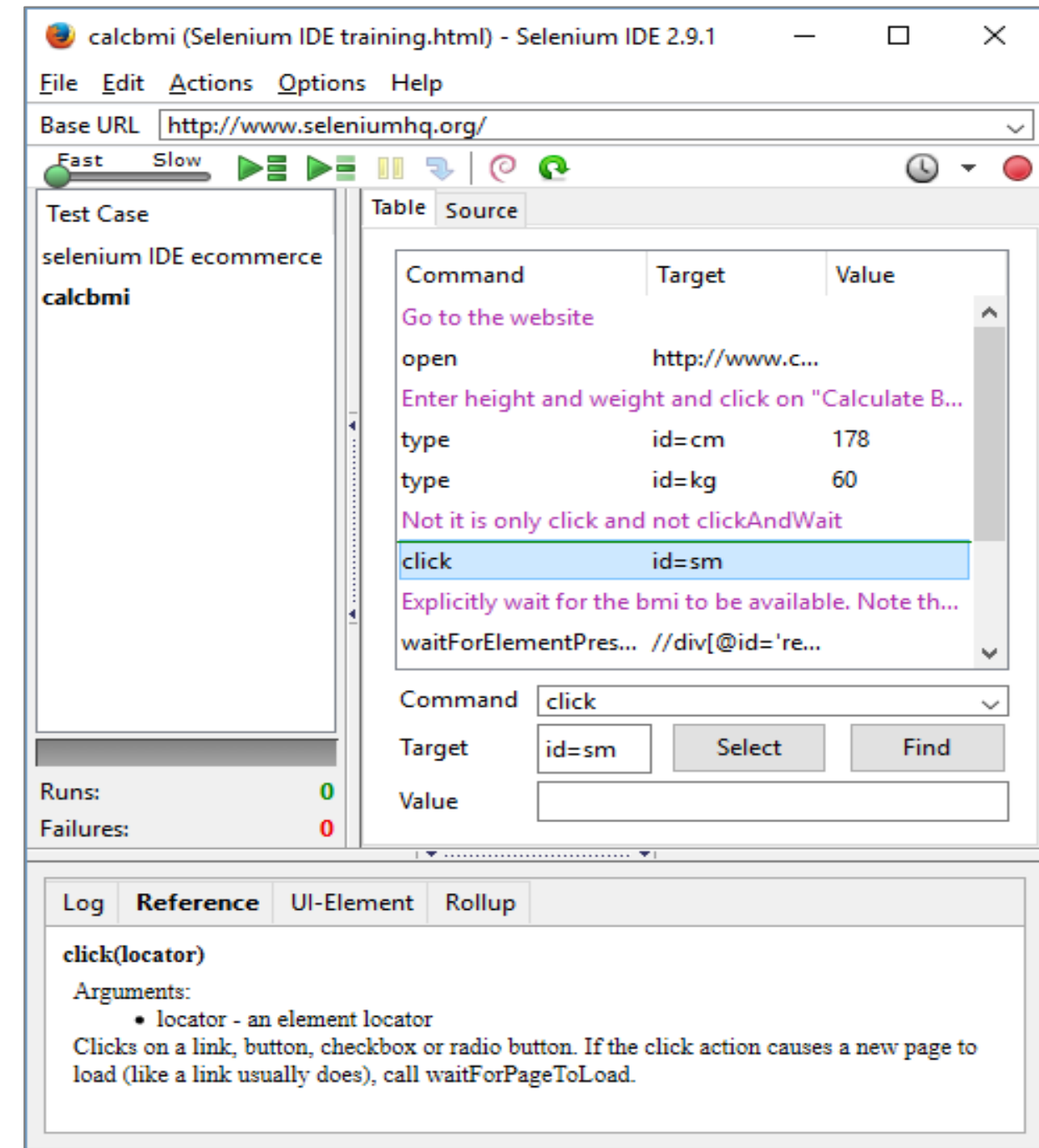
- Getting locators: Select

- Locating elements on page: Find

- Testing Results: shades of green and red

- Options settings

- User Extensions



# Features of Selenium



**Duration: 30 min.**

## **Problem Statement:**

Demonstrate the features of Selenium.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to demonstrate the features of Selenium:

1. Describe the features of Selenium.



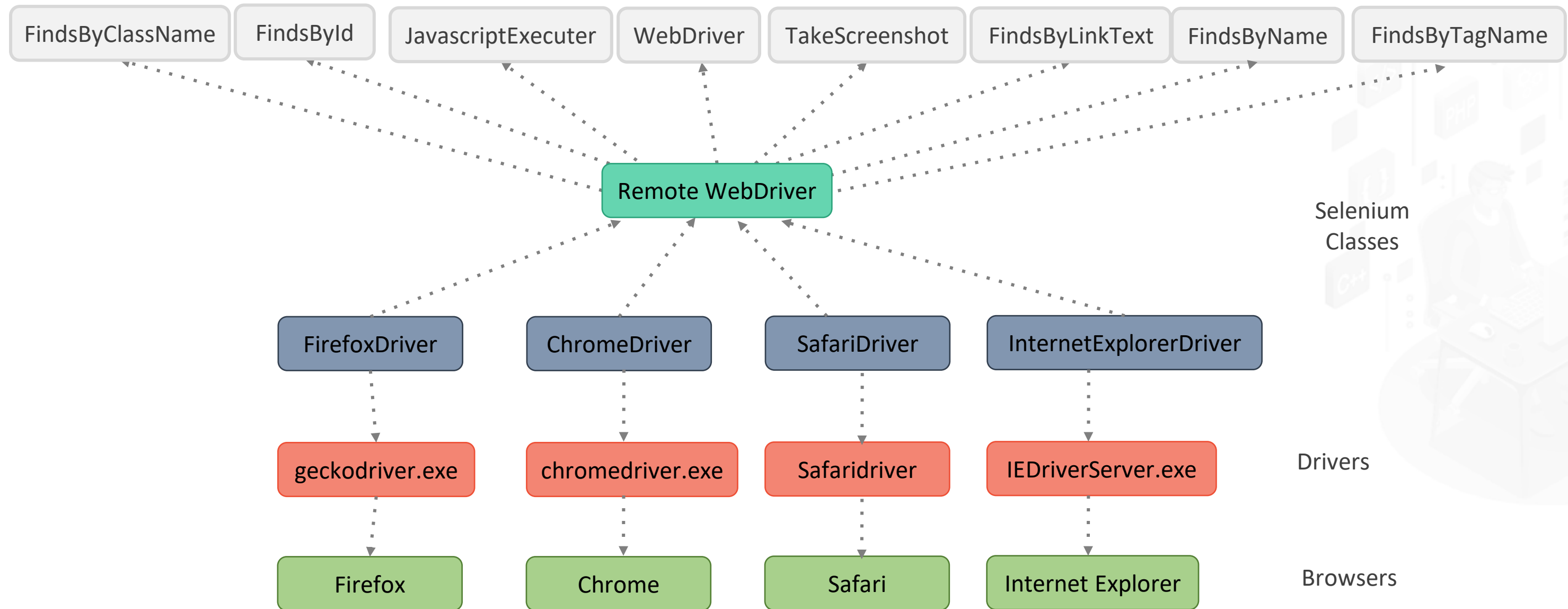
# FULL STACK

## WebDriver Architecture

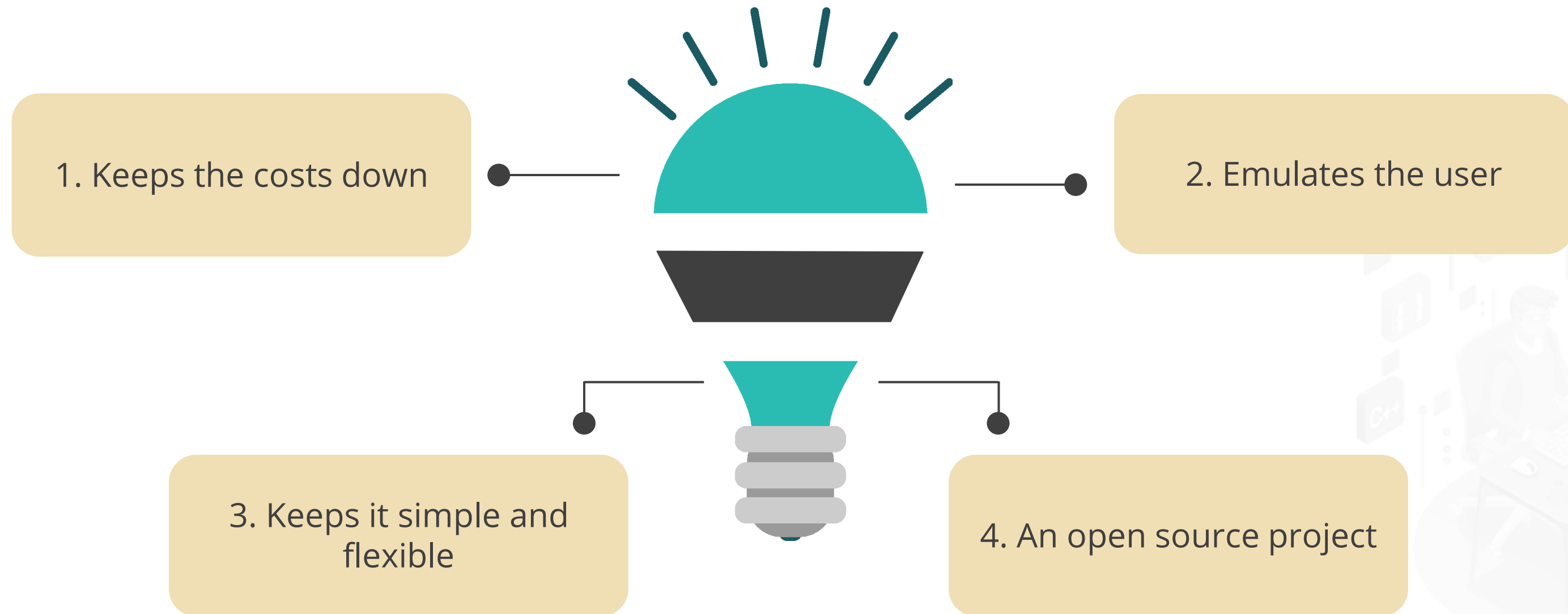


# WebDriver Architecture

This is the Java class hierarchy of WebDriver and third-party driver executable required to execute tests.



# WebDriver Advantages



Reference - <http://www.aosabook.org/en/selenium.html> by Simon Stewart - Creator of WebDriver

# WebDriver Installation and Integration in Eclipse



**Duration: 30 min.**

## **Problem Statement:**

Demonstrate how Selenium web driver is installed and integrated in Eclipse.

ASSISTED PRACTICE

# **Assisted Practice: Guidelines**

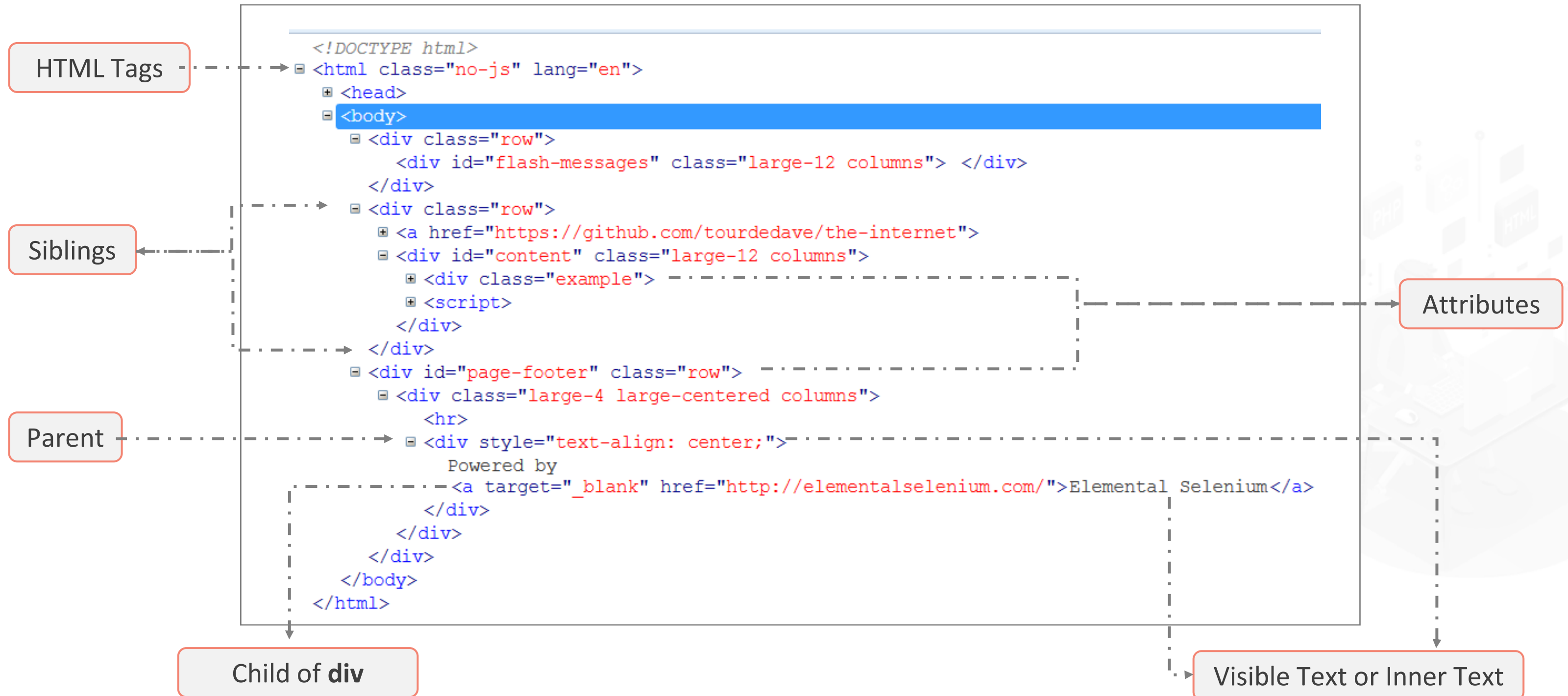
Steps to demonstrate WebDriver installation and configuration in Eclipse:

1. Download Selenium Standalone Server jar.
2. Launch Eclipse and create a Java project.
3. Configure WebDriver with Eclipse.



## Multiple Ways to Locate Elements

# Typical HTML Page Elements



# Typical HTML Page Elements

Element	HTML Tag	Key attributes
Text Field	input	type=text
Radio Button	input	type = radio
Checkbox	input	type=checkbox
Button	input	type=submit
Text	div, span, p	-
WebTable	table, thead, tbody, th, tr, td	-
Frame	frame, iframe	-
Images	img	alt, src
DropDown, Multiple Select	select	multiple
Link(Anchor)	a	href



You will see ID, name, and class attributes in many of the elements but, they are optional.



# findElement() Method

When do you need to deal with multiple elements on a webpage?

When dynamic elements that do not have unique ids are displayed on a page, such as:

- HTML tables displaying data from database
- Dropdowns and multiple selects displaying dynamic data

When there are groups of radio buttons or checkboxes that you need to select based on input data

When there are specific types of elements that you want to find inside a particular element, e.g., searching all links inside the “div”

# findElement() Method

## Syntax:

```
List<WebElement> allInputElements = driver.findElements(By.tagName("input"));  
for (WebElement oneInputElementAtATime : allInputElements )  
{  
    System.out.println(oneInputElementAtATime.getAttribute("value"));  
}
```



What happens if findElement() and findElements() do not find any such element or elements?

- You will receive an error in your console specifying the same and you should use a different locating technique to locate elements.

# WebDriver Locators

Selenium WebDriver uses eight locators to find the elements on a webpage.

The object identifiers or locators supported by Selenium are:

<b>Id</b> (unique, non-dynamic)	<code>driver.findElement(By.id("Email"));</code>
<b>Name</b> (unique, non-dynamic)	<code>driver.findElement(By.name("EmailID"));</code>
<b>Class</b> (unique, non-dynamic)	<code>driver.findElement(By.className("mandatory"));</code>
<b>CSSSelector</b>	<code>driver.findElement(By.cssSelector("input.login"));</code>
<b>XPath</b>	<code>driver.findElement(By.xpath("//input[@class='login']"));</code>
<b>linkText</b>	<code>driver.findElement(By.linkText("Gmail"));</code>
<b>Partial LinkText</b>	<code>driver.findElement(By.partialLinkText("Inbox"));</code>
<b>TagName</b>	<code>driver.findElement(By.tagName("input"));</code>

# WebElement Class Methods

Working with elements on a web page can be further enhanced and made easy by WebElement class methods.

The following methods help in acting on elements, retrieving information about elements, and getting positions, size, and visible texts:

- |   |   |  |  |   |
|---|---|--|--|---|
|    |   |                                 |   |    |
| <ul style="list-style-type: none"><li>• <code>clear()</code></li><li>• <code>click()</code></li><li>• <code>Submit()</code></li><li>• <code>sendKeys()</code></li></ul> | <ul style="list-style-type: none"><li>• <code>isEnabled()</code></li><li>• <code>isDisplayed()</code></li><li>• <code>isSelected()</code></li></ul> | <ul style="list-style-type: none"><li>• <code>findElement()</code></li><li>• <code>findElements()</code></li></ul> | <ul style="list-style-type: none"><li>• <code>getCssValue()</code></li><li>• <code>getAttribute()</code></li><li>• <code>getText()</code></li><li>• <code>getTagName()</code></li><li>• <code>getScreenshotAs()</code></li></ul> | <ul style="list-style-type: none"><li>• <code>getLocation()</code></li><li>• <code>getRectangle()</code></li><li>• <code>getSize()</code></li></ul> |

# Locating Guidelines

Locators are key to robust tests, and if one can select good locators, a test becomes more resilient.

- Locators should be unique, descriptive, and unlikely to change.
- Best options are ID, name, and class if they match the above condition.
- **LinkText** is also subject to change and will not work if your app supports internationalization (i18n).
- If an element does not have a unique ID, name, and class, search for parents that have a unique id.
- If that is does not work, request developers to provide unique locators.

# Timeout Methods

**Timeouts** are an interface for managing timeout behavior for **WebDriver** instances.

Method	What it does
<ul style="list-style-type: none"><li>implicitlyWait()</li></ul>	<ul style="list-style-type: none"><li>Amount of time the driver should wait while searching for an element (call to findElement) if it is not immediately available</li></ul>
<ul style="list-style-type: none"><li>pageLoadTimeout()</li></ul>	<ul style="list-style-type: none"><li>Amount of time to wait for a page to load completely</li></ul>
<ul style="list-style-type: none"><li>setScriptTimeout()</li></ul>	<ul style="list-style-type: none"><li>Amount of time to wait for an asynchronous script to finish execution</li></ul>



Implicitly waiting vs. putting **Thread.sleep()**:

**Implicit wait** polls the DOM to wait for a certain amount of time until an element is found. **Thread.sleep()** causes executing thread to sleep for specified time.

# Synchronizing Automation

Handling synchronization is critical to create resilient tests.

## **Unconditional Synchronization:**

Make the tool wait for a certain amount of time by specifying timeout value before the tool proceeds with the execution.

## **Synchronization Categories**

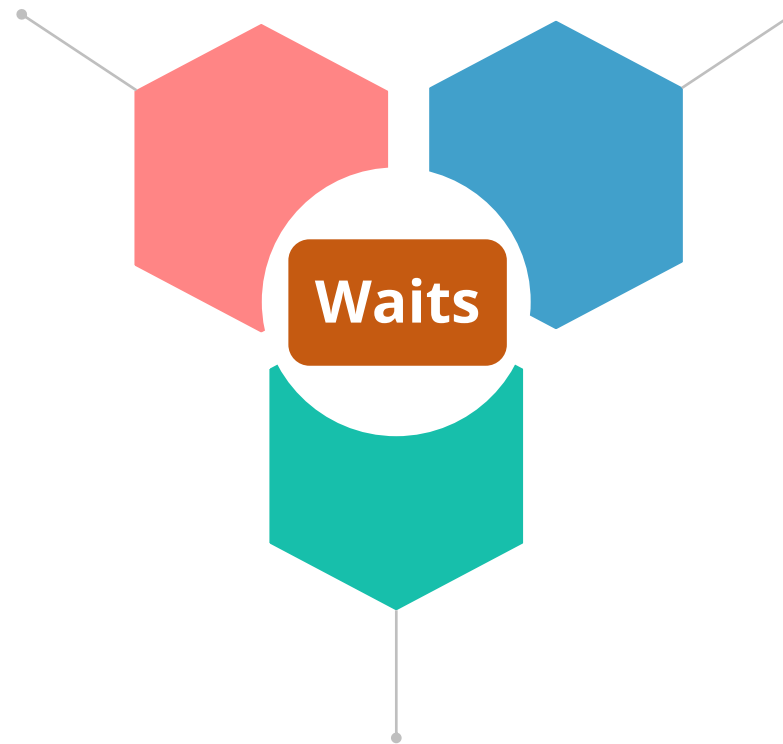
## **Conditional Synchronization:**

Specify conditions along with timeout value to make the tool wait, check for the condition, and then proceed with the execution.



# Why to Use Waits?

A website or webpage comprises various elements other than HTML like images, pop-ups, alerts, and animations. Automation scripts must run once all elements are loaded.



As all elements load at different intervals, it becomes difficult to automate an element if it is not loaded on the webpage and throws **ElementNotVisibleException**.

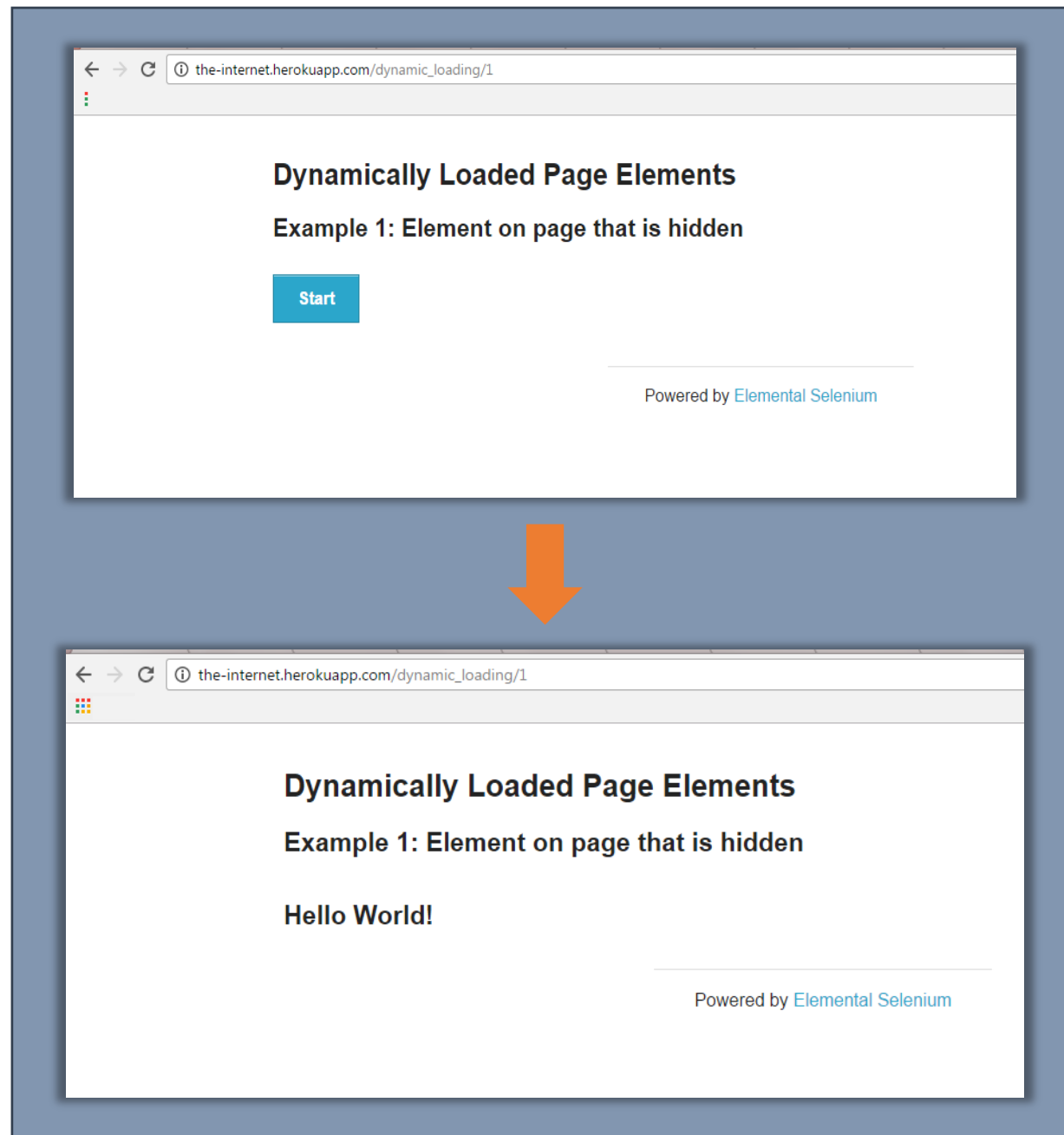
Both implicit and explicit waits are used to prevent such a situation. Both have different implementations. This helps developers to determine the designing of web pages for quick loading and reduce the wait time as much as possible.

# Limitations of Implicit Wait

## Test Case:

- Go to [http://the-internet.herokuapp.com/dynamic\\_loading/1](http://the-internet.herokuapp.com/dynamic_loading/1)
- Click on **Start** button
- Verify that **Hello World!** comes after clicking on **Start**

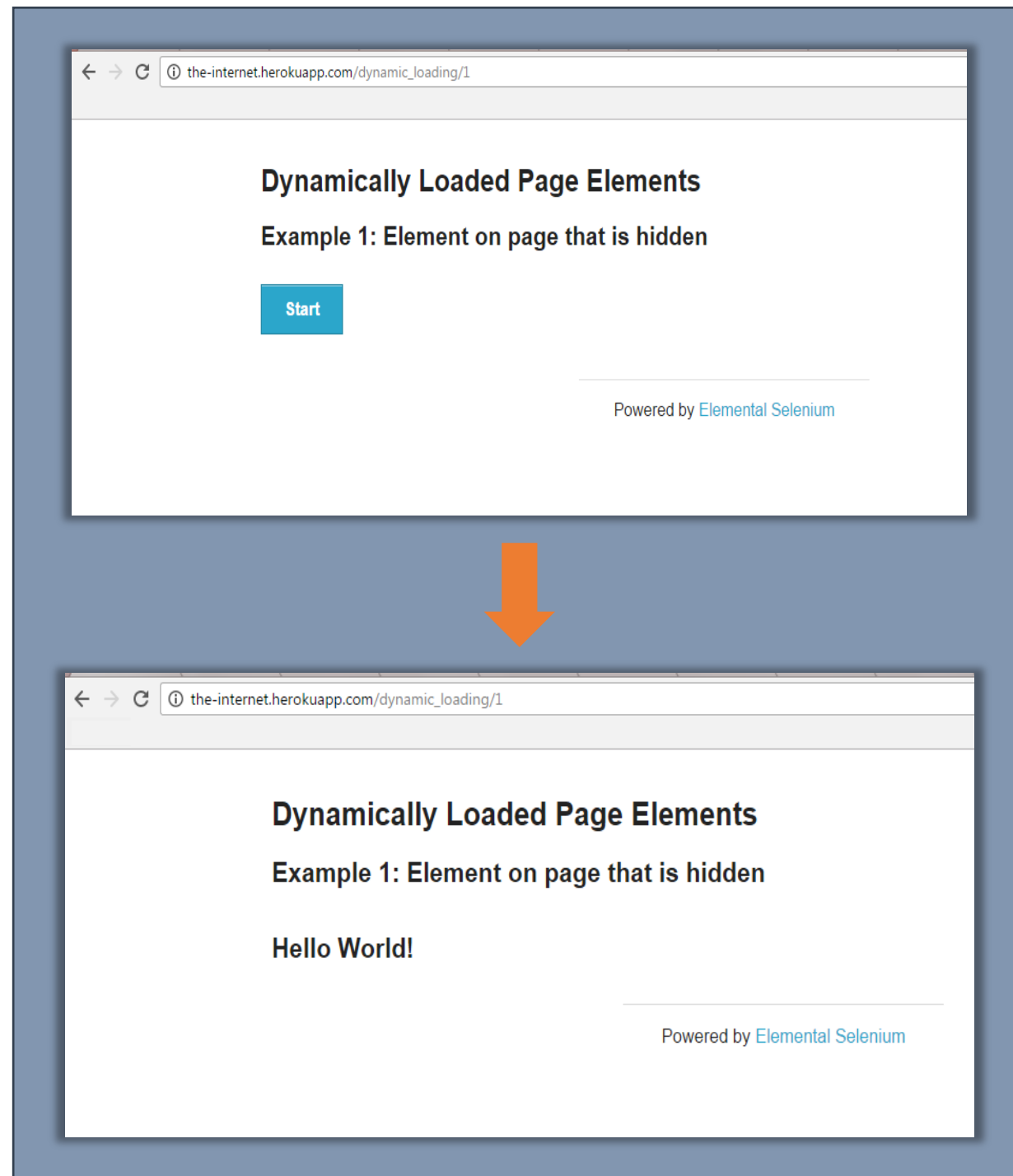
When you put **implicit wait** in this test case and use TestNG **assertEquals()** method to compare the output with **Hello World!**, the test fails.



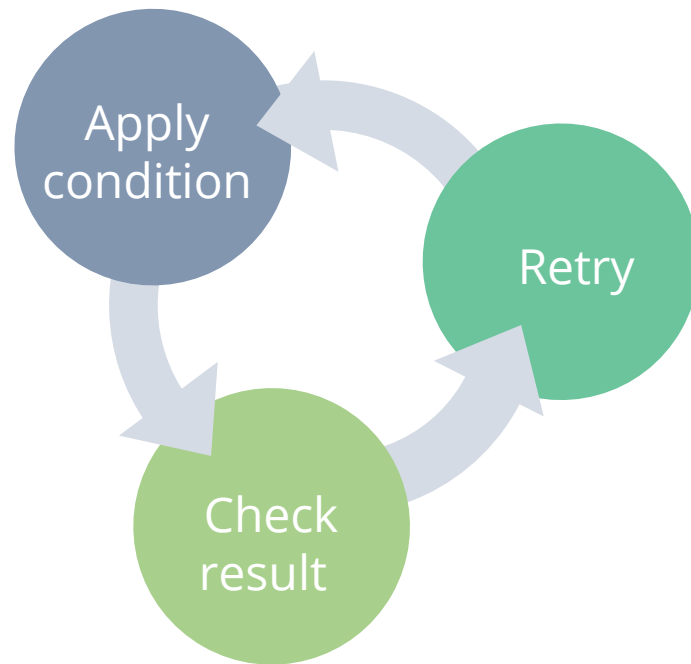
# Limitations of Implicit Wait

## Test Case Explanation:

- **Implicit Wait** only waits for the element to be present.
- In this test case, element (id=finish) is present on the page but is not visible.
- When you click on **Start** button, it takes time for the element (id=finish) to become visible.
- Implicit wait finds the element (even though it's invisible) and gives the control to the next line of code in your test script.
- Therefore, **assertEquals()** executes even before the element is made visible.
- **getText()** method does not work on invisible elements.
- This is why the test fails.



# Explicit Wait



Using **Explicit Wait**, you can wait for certain conditions:

- Element to be invisible or visible
- Alert, frame, or window to be present
- Title of the page to have specific value

- **Explicit Wait** pings the webpage every 500 milliseconds (configurable) to check the condition.
- It returns the controls to the test script once the condition is satisfied.

## Syntax:

```
WebDriverWait wait = new WebDriverWait(driver, 30);  
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("login-passwd")));
```

# Fluent Wait

**FluentWait** is a parent class of **WebDriverWait** class. Using this class, you can:

- Make the polling time configurable
- Specify the exceptions to be ignored
- Create custom conditions with your application under test

## Syntax:

Program waits for 30 seconds for an element to be present. It checks for the element every five seconds.

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver) //  
.withTimeout (30, TimeUnit.SECONDS)  
.pollingEvery(5, TimeUnit.SECONDS)  
.ignoring(NoSuchElementException.class);  
  
WebElement searchID = wait.until(new Function<WebDriver, WebElement>()  
    { public WebElement apply(WebDriver driver)  
        { return driver.findElement(By.id("login-passwd"));}  
    });
```

# Timeouts Guidelines



## Best practices:

- Do not use **Thread.sleep()**, except when **Explicit Wait** fails.
- Do not use **Implicit Wait**.
- Avoid using **Implicit Wait** and **Explicit Wait** in combination.
- Test your code in all the browsers (expected as per requirement) and optimize your wait time.
- Do not give hard-coded waits in your scripts. Make the script configurable.

# Multiple Ways to Locate Elements



**Duration: 40 min.**

## **Problem Statement:**

Demonstrate how elements are located using Selenium WebDriver.

ASSISTED PRACTICE



## Assisted Practice: Guidelines

Steps to locate elements in multiple ways using Selenium WebDriver:

1. Demonstrate ID as a locator.
2. Demonstrate class name as locator.
3. Demonstrate name as locator..
4. Demonstrate link text as locator.
5. Demonstrate Xpath as locator.
6. Demonstrate CSS selector as locator.
7. Demonstrate Xpath handling complex and dynamic elements as locator.



## Locating Elements through CSS and Xpath

# Locating with Xpath

XPath is the XML path and query language for selecting nodes from an XML document which is a W3C recommendation.

<b>Absolute XPath</b>	html/body/div/h2/a - absolute path to get the anchor element
<b>Relative XPath</b>	//input - get all "input" elements on the page
<b>Using Index</b>	//input[1] - getting all first "input" elements of a parent on the page
<b>Using Attribute Values</b>	//input[@value='Login'] - "input" with "value" attribute "Login"
<b>Two Attributes - AND</b>	//input[@value='Login'][@type='submit'] - "input" matching two attribute values
<b>Two Attributes - OR</b>	//input[@value='Login' or @type='submit'] - "input" matching one of the attributes
<b>Partial Match - contains</b>	//input[contains(@id, 'admin')] - "input" having "id" attribute containing "admin"

 Do not copy attribute values from this slide to your FirePath field as it may lead to error.

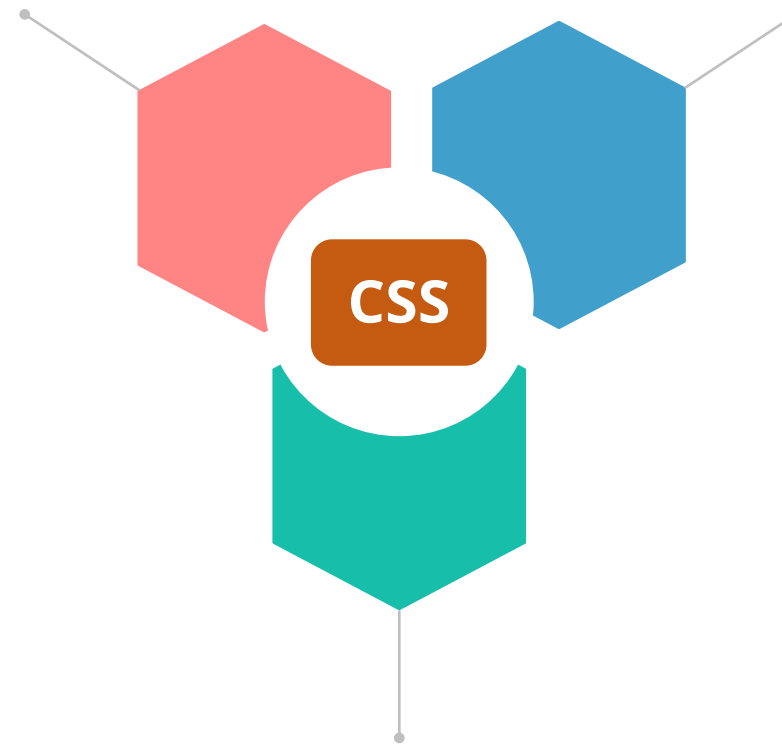
# Xpath vs. CSS

Below is the key difference between Xpath and CSS locating techniques:

Objective	In XPath	In CSS
Find an element inside a node, not necessarily immediate child	//	[space]
Find an element inside a node at an immediate level (immediate child)	/	>
Indicating attribute	@	Nothing

# Locating Elements with CSS

Selenium does not change the style of elements. It interacts with elements by clicking, getting values, typing, sending keys, etc.



CSS can change styling by declaring which bits of the markup it wants to alter through the use of selectors.

In web design, CSS (Cascading Style Sheets) are used to apply presentation in terms of colors, fonts, and layout to the markup (HTML) on a page. These visual properties are stored in a separate file with .CSS extension.

# Locating Elements with CSS

Ways to locate elements with CSS:

<b>Absolute CSS</b>	Html>body>div>h2>a - absolute path to get the anchor element
<b>Relative CSS</b>	input – get all <b>input</b> elements on the page
<b>Class selector</b>	p.step-third – <b>p</b> element that has “class” attribute <b>step-third</b>
<b>Using Attribute Values</b>	input[value='Login'] – <b>input</b> with <b>value</b> attribute <b>Login</b>
<b>Two Attributes - AND</b>	input[value='Login'][type='submit'] – <b>input</b> matching two attribute values
<b>id Selector</b>	table#SiteLinks – <b>table</b> that has <b>id</b> attribute <b>SiteLinks</b>
<b>Starts with - ^=</b>	p[id^='step'] – <b>p</b> that has <b>id</b> attribute starting with <b>step</b>
<b>Ends with - \$=</b>	div[id\$='erce'] – <b>div</b> that has <b>id</b> attribute ending with <b>erce</b>
<b>Contains - *=</b>	input[id*='admin'] – <b>input</b> that has <b>id</b> attribute containing <b>admin</b>
<b>Following sibling</b>	form ~ h3 - following sibling <b>h3</b> of <b>form</b> element
<b>First following sibling</b>	div + h3 – first following sibling <b>h3</b> of <b>div</b> element
<b>Position in hierarchy</b>	div[id='eCommerce'] h4:nth-child(2) – <b>h4</b> which is second child of <b>div</b> that has <b>id</b> – <b>eCommerce</b>

# Locating Elements Through CSS and XPath



**Duration: 60 min.**

## **Problem Statement:**

Demonstrate how elements are located through CSS and XPath.

ASSISTED PRACTICE



# Assisted Practice: Guidelines

Steps to locate elements through CSS and XPath:

1. Demonstrate how to find the element present on the page by using CSS selector.
2. Demonstrate how to find the element present on the page by using Xpath.
3. Push the code to your GitHub repositories.



# FULL STACK

## Handling Various Web Elements

# Radio Button and Checkboxes

Radio buttons and checkboxes can be handled by below methods of **WebElement** class:

Element	What can you do?	WebElement Methods
Radio button	<ul style="list-style-type: none"><li>• Select a radio button</li><li>• Check if a radio button is selected</li></ul>	<ul style="list-style-type: none"><li>• Click()</li><li>• isSelected()</li></ul>
Checkbox	<ul style="list-style-type: none"><li>• Select a checkbox</li><li>• Deselect a checkbox</li><li>• Check if a checkbox is selected</li></ul>	<ul style="list-style-type: none"><li>• Click()</li><li>• Click()</li><li>• isSelected()</li></ul>

# Drop-Down List

The following methods are available in **Select** class provided by Selenium. These are applicable for drop-down as well as multi-select components.

What can you do ?	Select class methods
<ul style="list-style-type: none"><li>Select an option</li></ul>	<ul style="list-style-type: none"><li><code>selectByVisibleText()</code></li><li><code>selectByValue()</code></li><li><code>selectByIndex()</code></li></ul>
<ul style="list-style-type: none"><li>Check what options are available</li></ul>	<ul style="list-style-type: none"><li><code>getOptions()</code></li></ul>
<ul style="list-style-type: none"><li>Check which option is selected first</li></ul>	<ul style="list-style-type: none"><li><code>getFirstSelectedOption()</code></li></ul>
<ul style="list-style-type: none"><li>Check if it is a drop-down or multi-select</li></ul>	<ul style="list-style-type: none"><li><code>isMultiple()</code></li></ul>

# Multi-Select Options

The following methods are only applicable to multi-select elements:

What can you do ?	Select class methods
<ul style="list-style-type: none"><li>• Deselect an option</li></ul>	<ul style="list-style-type: none"><li>• <code>deselectByVisibleText()</code></li><li>• <code>deselectByValue()</code></li><li>• <code>deselectByIndex()</code></li></ul>
<ul style="list-style-type: none"><li>• Get all selected options</li></ul>	<ul style="list-style-type: none"><li>• <code>getAllSelectedOptions()</code></li></ul>
<ul style="list-style-type: none"><li>• Deselect all</li></ul>	<ul style="list-style-type: none"><li>• <code>deselectAll()</code></li></ul>

# HTML Tags for Tables

Common tags for HTML Table:

- `<table>` : Defines an HTML Table
- `<tr>` : Defines a row
- `<th>` : Defines a header cell
- `<td>` : Defines standard (not header) cells in HTML table.
- `<thead>` : Groups table header content in HTML table
- `<tbody>` : Groups the table body content

## HTML Table Structure

The following image shows how a complex HTML table would appear on a webpage:

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	\$50.00	http://www.jsmith.com	<a href="#">edit</a> <a href="#">delete</a>
Bach	Frank	fbach@yahoo.com	\$51.00	http://www.frank.com	<a href="#">edit</a> <a href="#">delete</a>
Doe	Jason	jdoe@hotmail.com	\$100.00	http://www.jdoe.com	<a href="#">edit</a> <a href="#">delete</a>
Conway	Tim	tconway@earthlink.net	\$50.00	http://www.timconway.com	<a href="#">edit</a> <a href="#">delete</a>

# HTML Table Methods

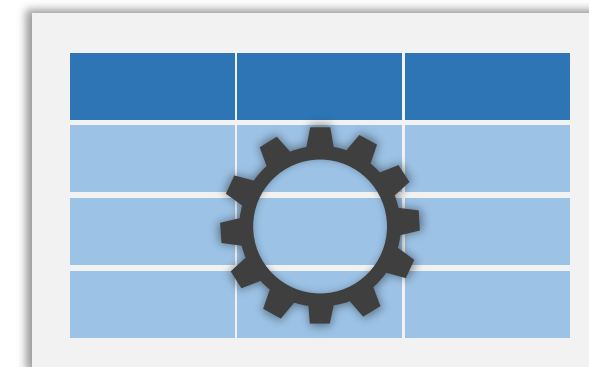


## Working with HTML tables in WebDriver:

- No standard methods are provided by WebDriver to work with HTML tables
- We need to create reusable methods as part of our framework.

## Examples of re-usable methods:

- Finding the number of rows and columns in a table.
- Finding a particular cell from the table.
- Taking action on a particular cell based on the content of some other cell.





# Handling Various Web Elements



**Duration: 60 min.**

## **Problem Statement:**

Demonstrate how web elements are handled in Selenium.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to handle web elements:

1. Handle various web elements present on the web page.



# Date Picker



**Duration: 60 min.**

## **Problem Statement:**

Demonstrate how to automate calendars on the web page

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to perform:

1. Create a selenium project.
2. Write code for calendar automation



# FULL STACK

## Working with External Elements

# External Elements

So far, we have focused on a single HTML page. However, the real web application includes:



JavaScript alerts



Browser Pop-up windows



Pop-up windows. For example: file upload, print dialog



Frames and iframes

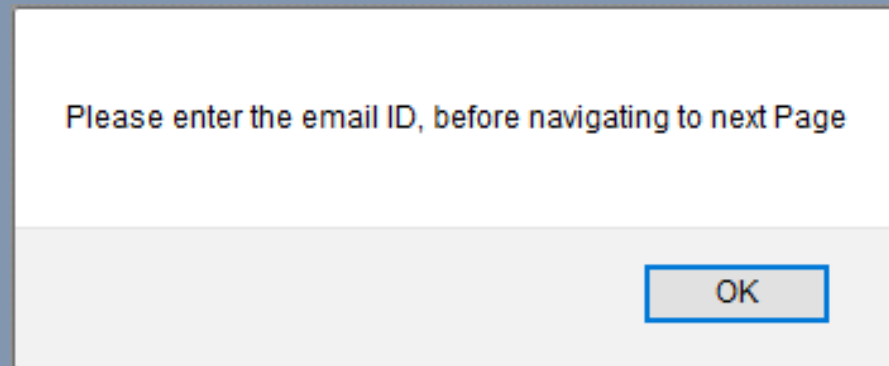


# JavaScript Alerts

WebDriver provides APIs in **Alert** class to handle JavaScript popups.

**JavaScript has three types of pop-ups:**

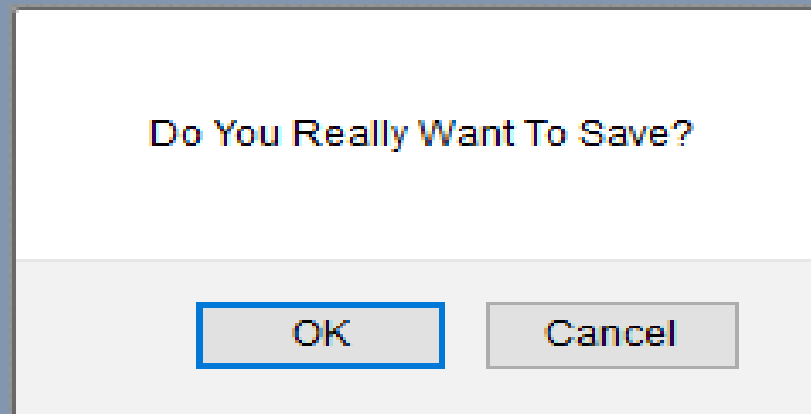
**Simple Alert Box:** Contains a message and OK button



Please enter the email ID, before navigating to next Page

OK

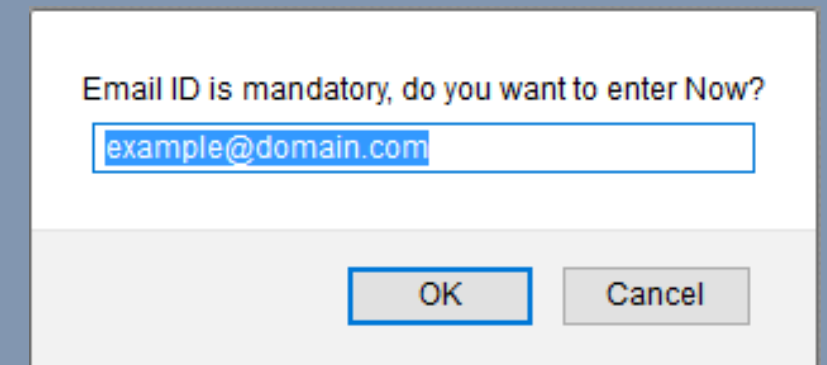
**Confirm Box:** Contains a message along with OK and Cancel button



Do You Really Want To Save?

OK Cancel

**Prompt Box:** Contains a message, a text box to enter a value along with OK and Cancel button



Email ID is mandatory, do you want to enter Now?

example@domain.com

OK Cancel

# JavaScript Alerts

## Syntax for Handling Alerts

**Alert = driver.switchTo().alert()** // Tells the driver to switch focus on alert pop-up window.

## Key Operations that can be performed on JavaScript popups

**alert.getText()** // Gets the text on alert window.

**alert.accept()** // Accepts the alert (for example, clicking **Ok** button).

**alert.dismiss()** // Dismisses the alert (for example, clicking **Cancel** button ).

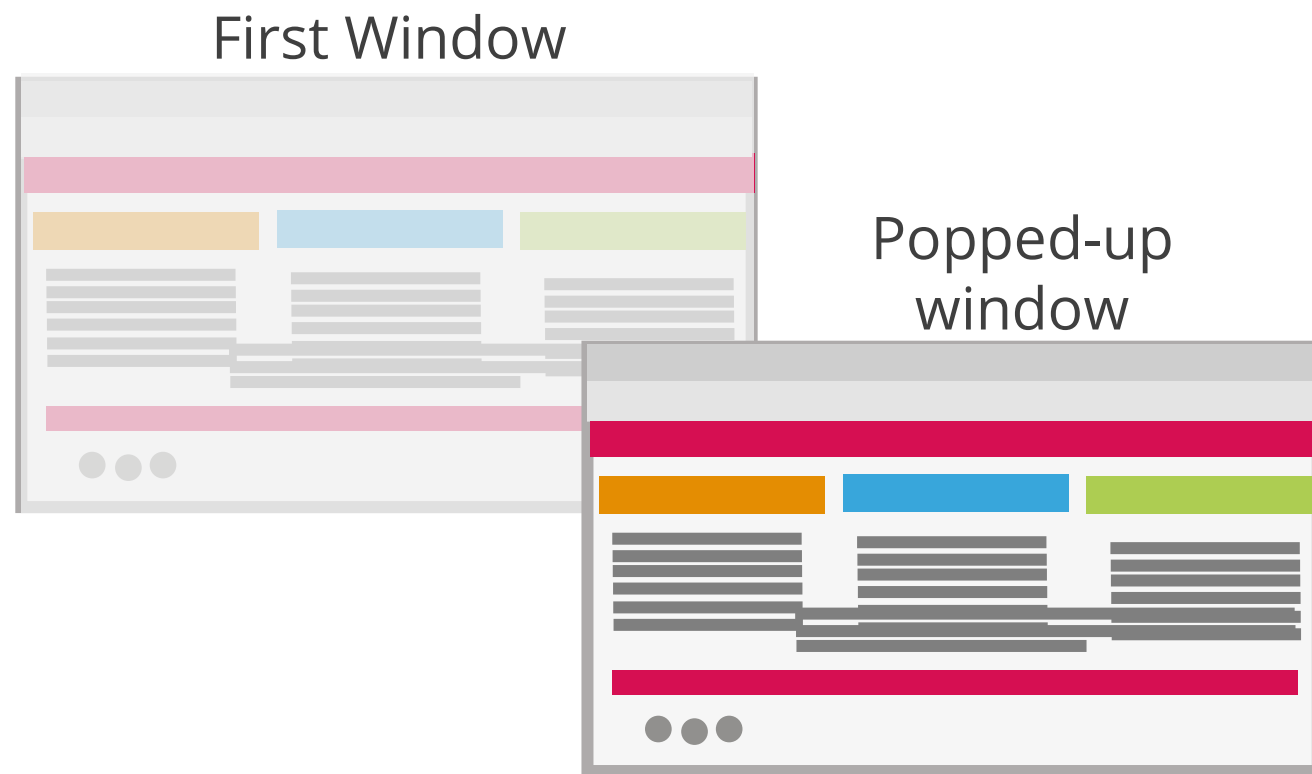
**alert.sendKeys()** // Passes text to be entered in any field on the pop-up window.

## Exception to be handled

**NoAlertPresentException** // This exception triggers when there is no alert, but system is trying to switch to an alert.



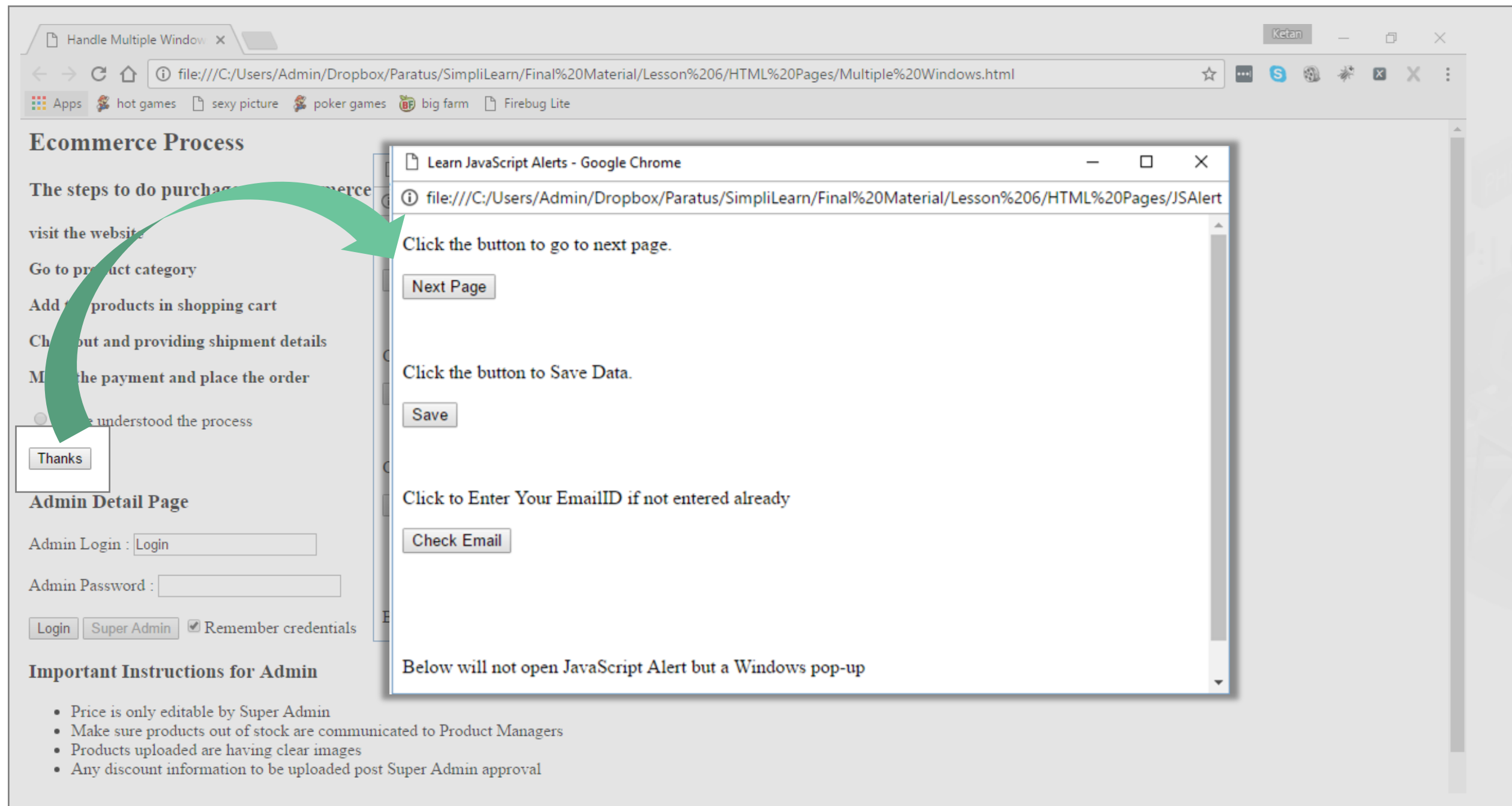
# Browser Pop-Ups



- Some application designs generate additional browser window if a button or link is clicked.
- In some cases, though the focus is on a new window, WebDriver object focus remains on the previous window.
- This means that executing `driver.findElement()` or `driver.getTitle()` will get results from the previous window.
- WebDriver provides API to switch the WebDriver instance focus from previous window to current window.
- You should use `driver.close()`, instead of `driver.quit()`, to close the current window.

# Browser Pop-Ups

The screen below shows that clicking on **Thanks** button invokes another browser window. Now, the focus is visibly on new window, but WebDriver instance is still on the previous window.



# Browser Pop-Ups

**WebDriver provides APIs to handle multiple windows.**

**driver.switchTo().window()** //Tells the driver to switch focus to a window by window name or window handle.

**driver.getWindowHandle()** //Gets the window handle of the current window.

**driver.getWindowHandles()** //Gets window handles of all the windows opened by the current driver.

**The two types of parameters the window() method can take are:**

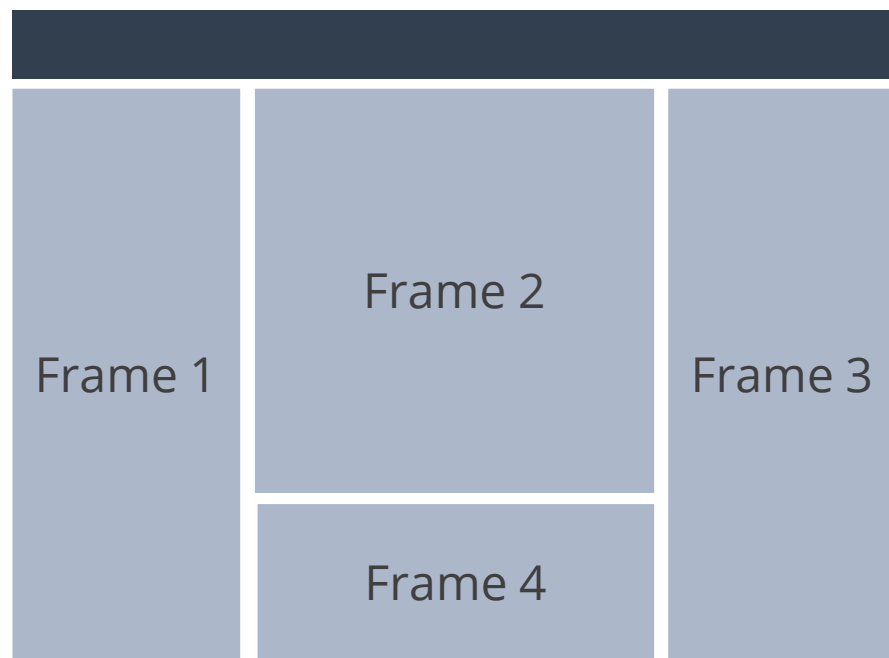
**driver.switchTo().window(windowName)** // Where “windowName” is the name of the new window opened by the browser.

**driver.switchTo().window(windowHandle)** // Where “windowHandle” is the window handle returned by driver.getWindowHandle(s)() method.

## Exception to be handled

**NoSuchWindowException** // This exception triggers when the system tries to switch focus on a window that does not exist.

# HTML Frames



- HTML frames enable you to display documents in different sections.
- Frames/iFrames can be used to show content from another website. For example, ads
- Frames are like HTML pages embedded in another HTML page.
- Generally, `driver.findElement()` or other WebDriver commands do not search inside frames.

# Handling Frames

**WebDriver provides APIs to handle Frames**

**driver.switchTo().frame()** //Tells driver to switch focus to a frame by ID, name, index, or **WebElement**  
**Driver.switchTo().defaultContent()** //Tells driver to switch focus back to default page  
**Driver.switchTo().parentFrame()** //Tells driver to switch focus to parent frame.

**The frame() method can take three types of parameters:**

**driver.switchTo().frame(id or name)**      // ID or name attribute of the frame  
**driver.switchTo().frame(index)**              // Position of the frame within the frameset  
**driver.switchTo().frame(WebElement)**      // frame **WebElement**

# User Interactions Automation



The Advanced User Interactions API helps to perform simple to complex actions on a web page, such as:

- Right click
- Double click
- Drag and drop
- Pressing and releasing keys
- Mouse hover



# User Interactions Automation

There are two types of user interactions:

## Keyboard interactions

Pressing a key: `keyDown()`

Releasing a key: `keyUp()`

Typing in a text field: `sendKeys()`

## Mouse interactions

Click on element:  
`Click()/Click(element)`

Click and Hold: `clickAndHold()`

Right click: `contextClick()`

Double click: `doubleClick()`

Drag and Drop: `dragAndDrop()`

Mouse hover: `moveToElement()`

Releasing left mouse button: `release()`

# User Interactions Automation

**Actions class** is used to generate a series of actions.

```
Actions builder = new Actions(driver);
builder.click(AllRowsOfTable.get(1)) /* 1st Row selected */
    .keyDown(Keys.CONTROL)
    .click(AllRowsOfTable.get(3)) /* 3rd Row selected */
    .click(AllRowsOfTable.get(6)). /* 6th Row selected */
    .keyUp(Keys.CONTROL);
```

**Build()** method to get action:

```
Action selectMultipleRows = builder.build();
```

Execute the action by calling **perform()**.

```
selectMultipleRows.perform();
```



# Working with External Elements



**Duration: 60 min.**

## **Problem Statement:**

Using Selenium WebDriver, write a program to handle alerts.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to work with external elements:

1. Handle external pop ups.
2. Handle new tabs and new windows.
3. Push the code to your GitHub repositories.



# FULL STACK

## Screenshots and Browser Profiles

# Capturing Screenshots

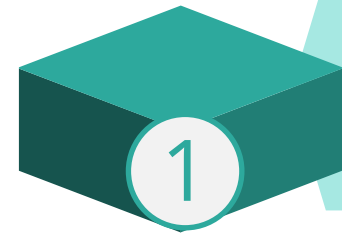
In bug analysis, screenshots are desirable. In automation testing, it is mandatory to take a screenshot for verification of functionality of test cases. Selenium automatically takes screenshots during test execution.

- Selenium provides TakesScreenshot interface to capture screenshots
- WebDriver instance has to be type casted to TakesScreenshot

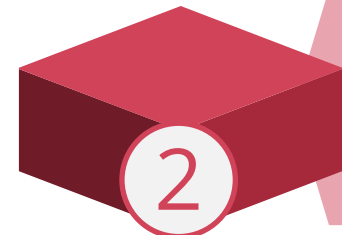
# Capturing Screenshots

Screenshots can be captured in Selenium in 3 simple steps.

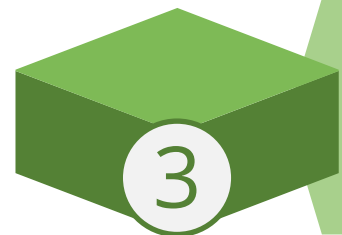
## Steps



Convert WebDriver object to TakesScreenshot object



Call getScreenshotAs method to create an image file



Copy the file to the desired location

## Sample Code

```
public static void takeSnapShot(WebDriver  
webdriver String filePath)  
{  
    //Convert web driver object to  
    TakeScreenShot object  
    TakesScreenshot scrObj -  
    ((TakesScreenshot)webdriver);  
    //Call getScreenshotAs method to create an  
    image file  
    File scrFile =  
    scrObj.getScreenshotAs(outputType.FILE);
```

# Browser Profiles

## Browser profile

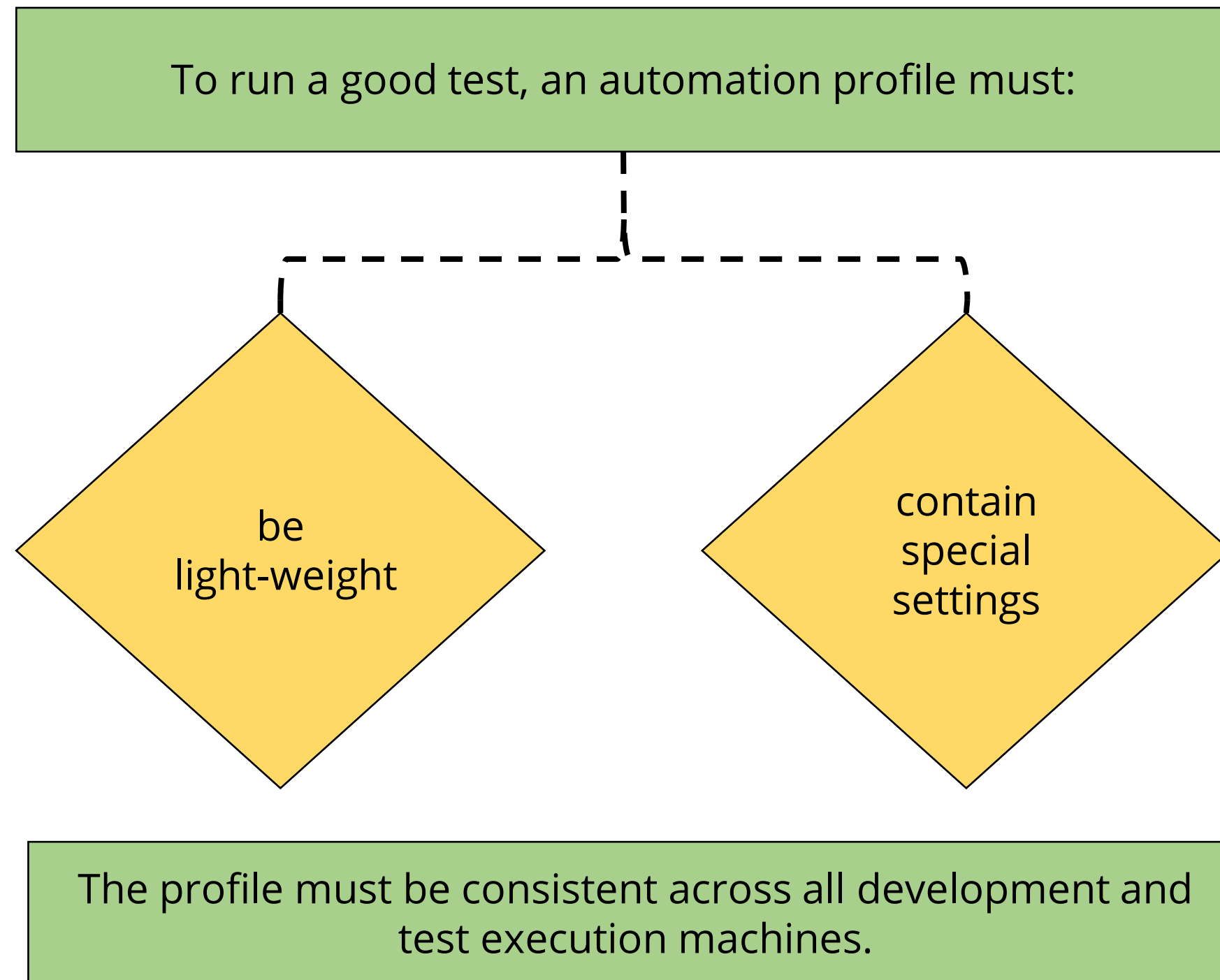
- Set of files maintained by the web browser that contain personal information, such as bookmarks, passwords, and user preferences
- Stored in a separate location from the browser program files
- Multiple profiles, each containing a separate set of user information are possible

Profile Manager allows creation, removal, renaming, and switching of profiles.



# Browser Profiles for Automation

It is advisable to create a separate browser profile while running automation.



# Need for Custom Browser Profiles: Firefox Browser

The default Firefox browser profile is not automation-friendly.

The profile must handle special test needs required to make the test execution reliable.

The profile must handle packaging and deploying the special test settings.

The profile must contain just the required settings and plug-ins needed for test execution.

Selenium copies the entire profile to a temporary directory each time a new session driving a Firefox instance is started.





# Changing Browser Profiles

Steps to change Firefox browser profiles

Close all the open Firefox instances

On the Windows Start Menu select Run; type `firefox.exe -p`

Select the required profile from the dialog box

Click **Start Firefox** button to open a new instance with the selected profile

In Selenium WebDriver software test, the `getProfile` method of **profilesIni**, an inbuilt WebDriver class must be used to access the Firefox profile.



# Screenshots and Browser Profiles



**Duration: 60 min.**

## **Problem Statement:**

Demonstrate how screenshots are captured and browser profiles are changed in Selenium.

ASSISTED PRACTICE

## Assisted Practice: Guidelines

Steps to take screenshots using Selenium WebDriver and set the browser profile:

1. Write a code for screenshots.
2. Run the code.
3. Pushing the code to your GitHub repositories.
4. View browser profile.



# FULL STACK

## AutoIT

# AutoIT: Introduction

AutoIT is an open source tool used to automate processes involving Windows and desktop applications. AutoIT is used as Selenium cannot handle Windows-based activities.

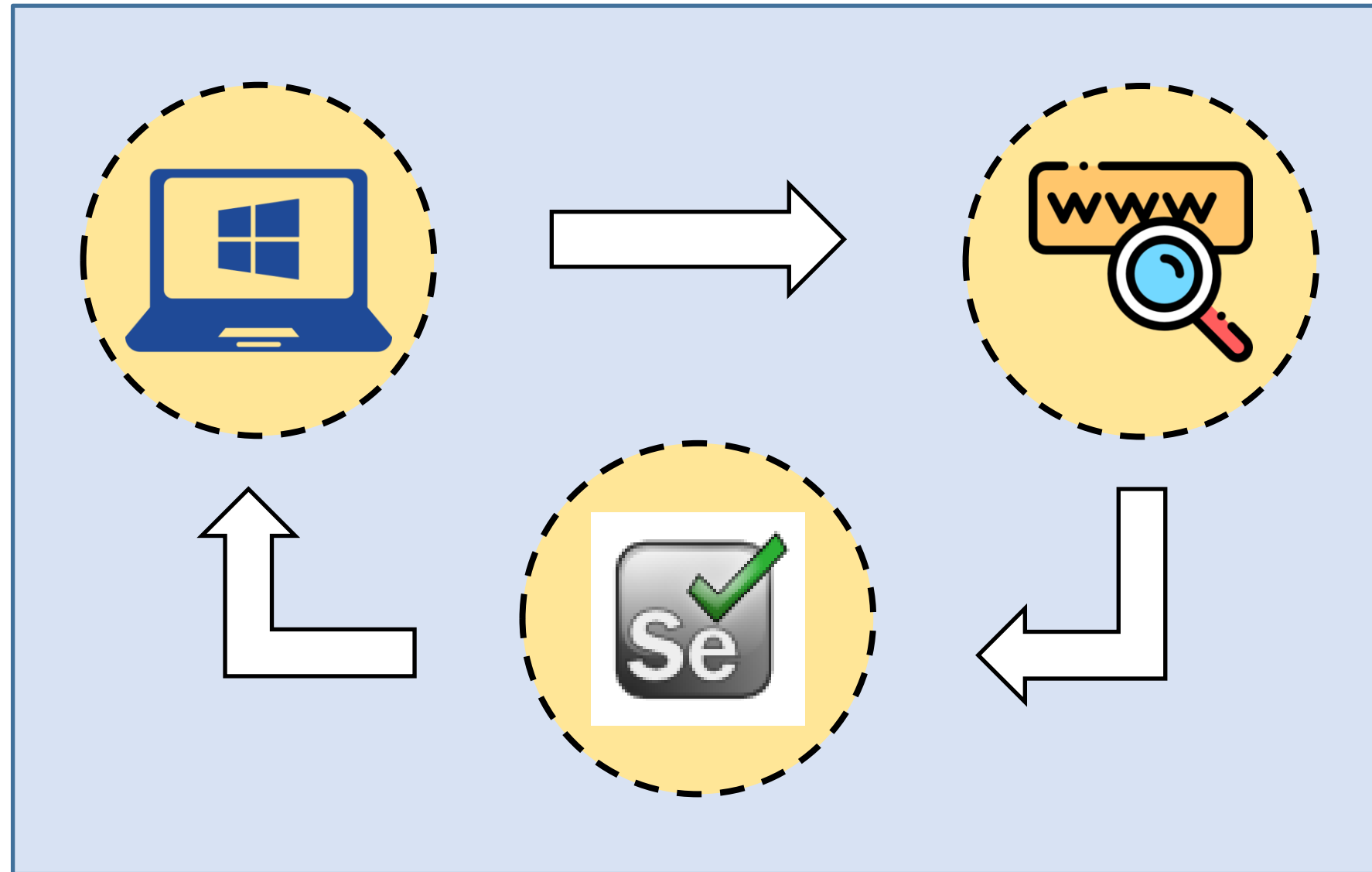


- Third-generation programming language
- Similar to BASIC
- Uses C# and VB for scripting
- A few lines of script are sufficient for automating a task
- A process can be recorded using AutoIT recorder

AutoIT is effective for automating workflows that work between browser, desktop, and Selenium.

# AutoIT

AutoIT is effective for automating workflows that work between browser, desktop and Selenium



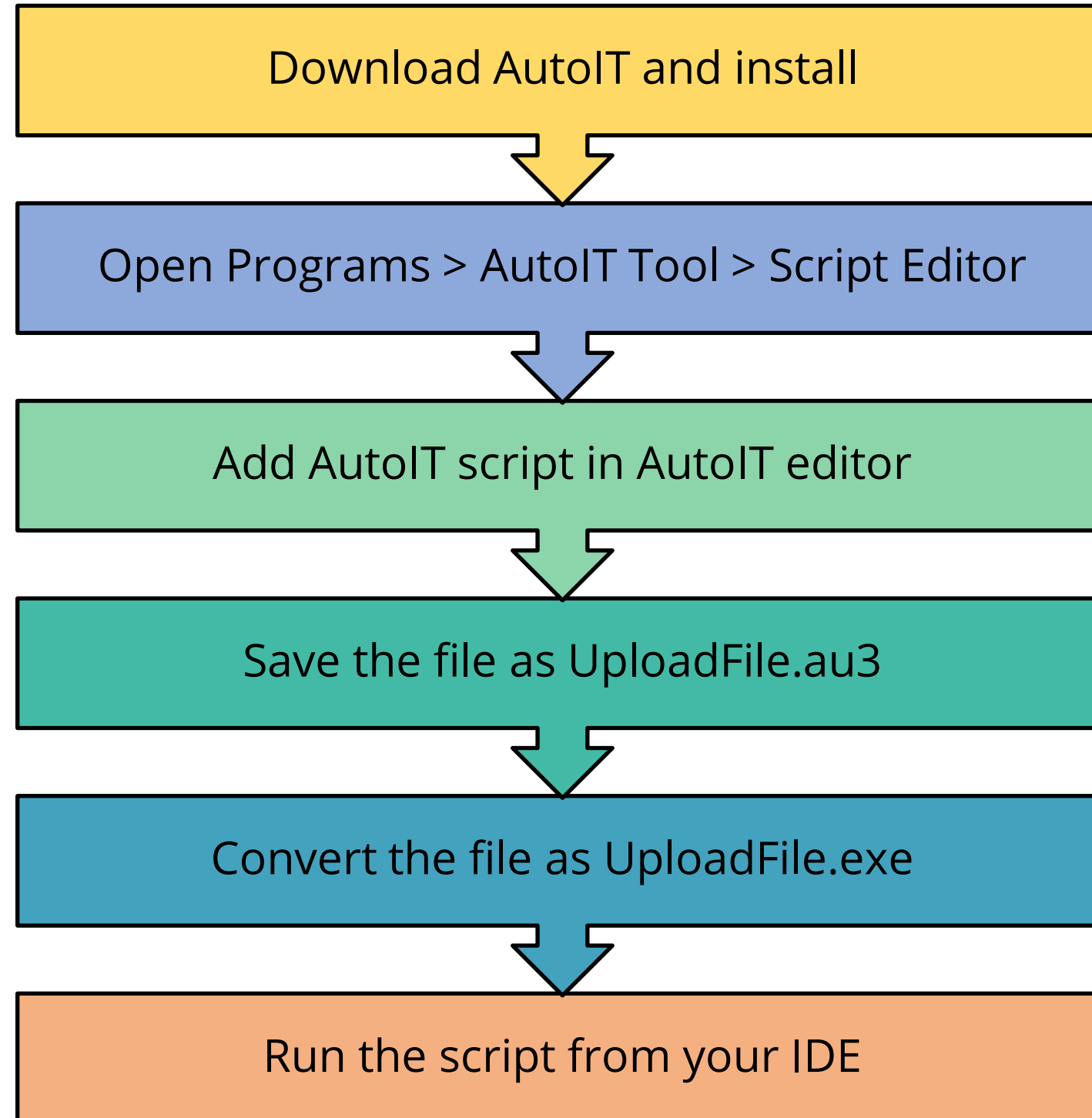
# AutoIT: Features

AutoIT performs keyboard actions and supports web and desktop applications. Some of the other features of AutoIT include:



- Easy-to-learn syntax
- Record and playback
- Compatibility with all Windows OS versions
- Graphical User Interfaces
- Compilation of scripts into standalone executables
- Simulation of keystrokes and mouse movements
- RunAs function

# AutoIT: File Upload





# AutoIT Installation and Configuration



**Duration: 30 min.**

## **Problem Statement:**

Demonstrate installation and configuration of AutoIT.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to install AutoIT and configure it:

1. Install and configure AutoIT.



# Handling File Uploads



**Duration: 60 min.**

## **Problem Statement:**

Demonstrate how file uploads are handled in AutoIT.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to upload files using AutoIT:

1. Handle file upload by SendKeys.
2. Handle file upload by AutoIT Script.



# FULL STACK

## Sikuli for UI Testing

# Sikuli for UI Testing

Sikuli is a GUI-based open source automation tool. This tool is used to handle Windows-based pop-ups and to interact with the elements of a web page. It uses image recognition to interact with the elements of the web page.

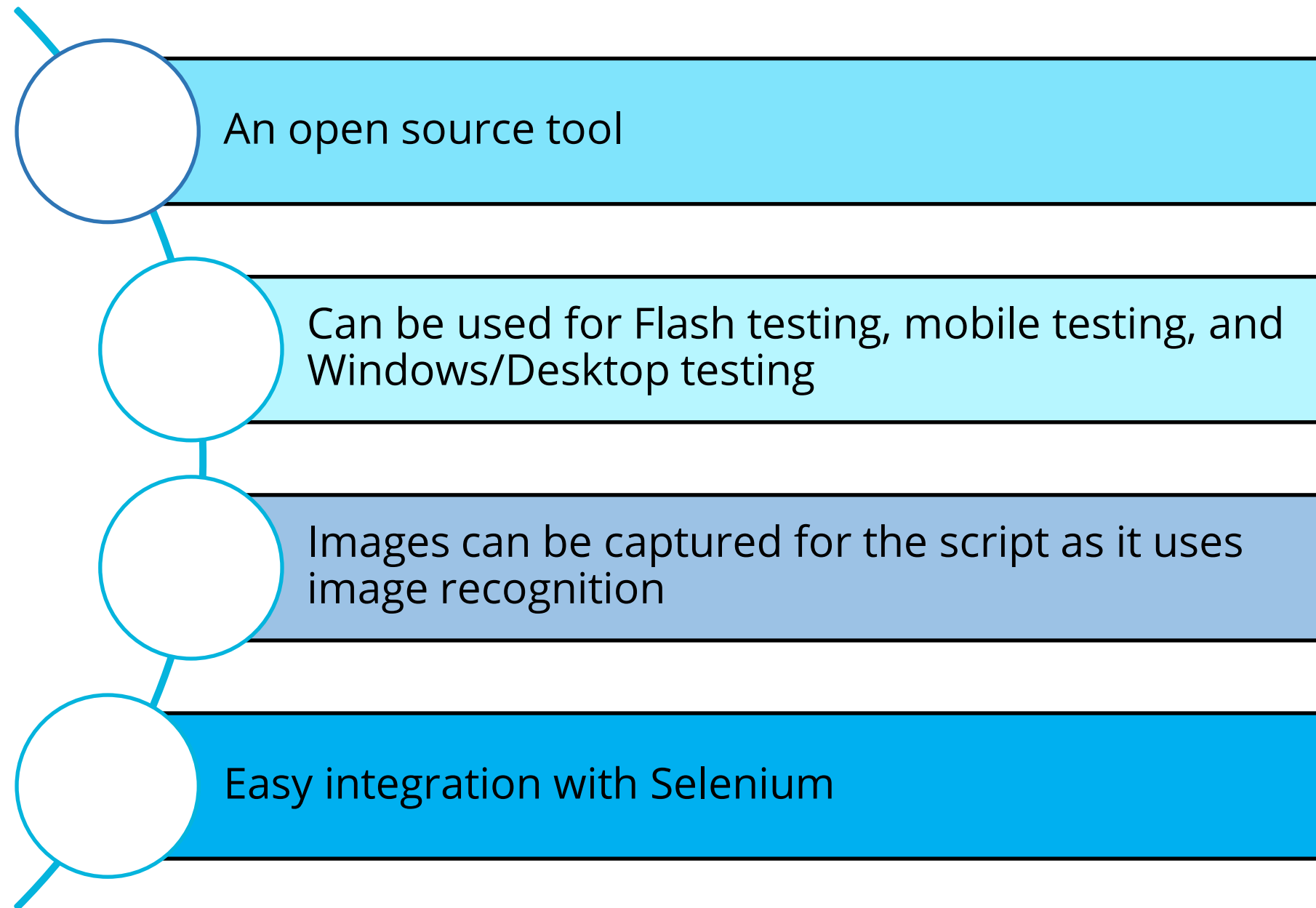


## Features

- Easy integration with Selenium
- Automation of anything seen on the screen
- Platform independent
- Easy automation of Flash objects

Sikuli is preferred when the UI elements are not constantly changing and are stable.

# Sikuli: Advantages

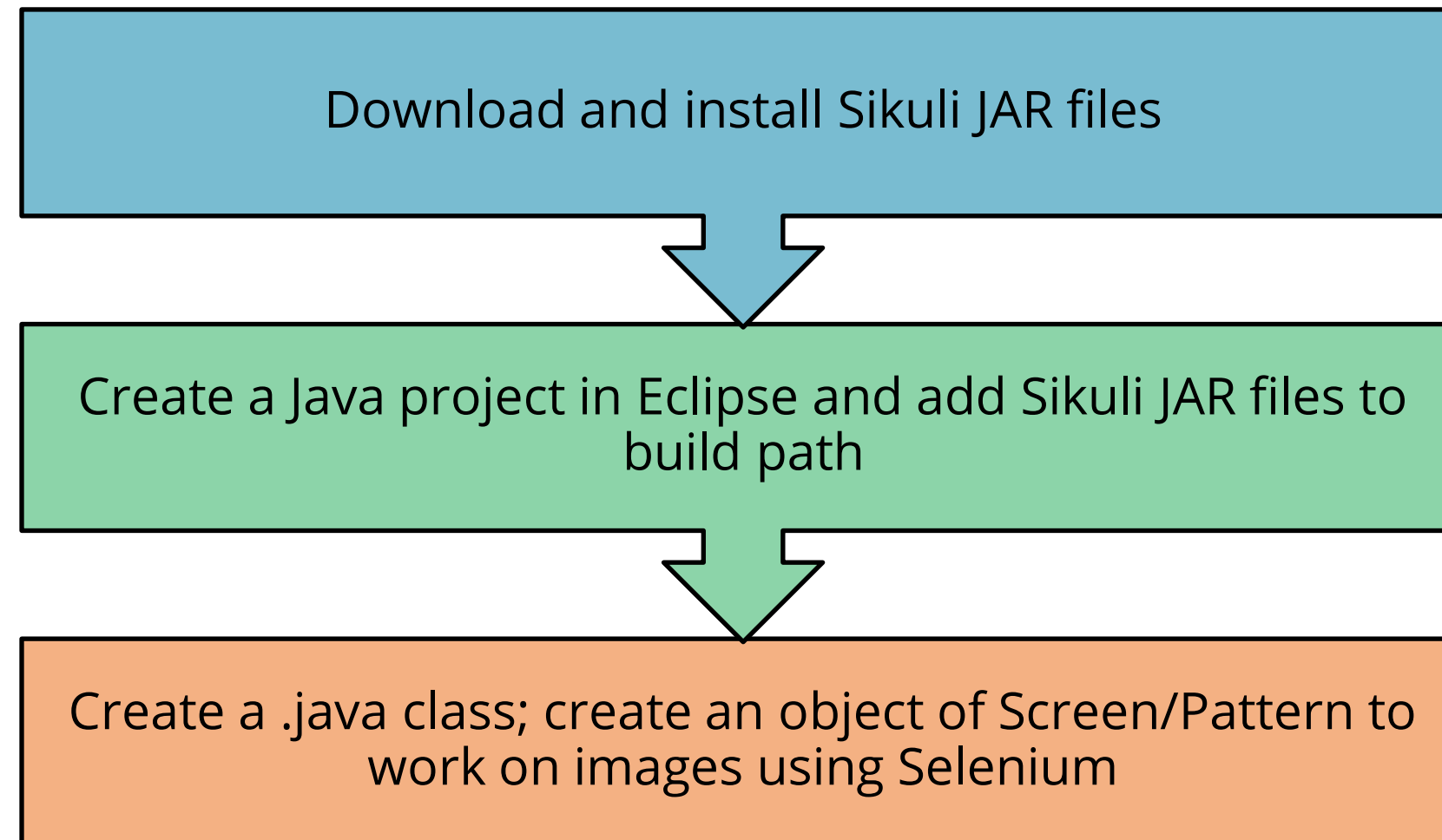


# Sikuli: Classes

Class	Description	Methods
Screen	Used for screen operations; contains predefined methods for all the commonly performed operations on screen elements	click, doubleClick, type, hover, find
Pattern	Used for pattern-based operations; associates the image file with additional attributes to uniquely identify the element	getFileName, similar, exact



# Sikuli: Integration With Selenium



# Sikuli for UI Testing



**Duration: 50 min.**

## **Problem Statement:**

Demonstrate how Sikuli is used for UI testing in Selenium.

ASSISTED PRACTICE

## Assisted Practice: Guidelines

Steps to integrate Sikuli with Selenium WebDriver and interact with web elements:

1. Integrate Sikuli with Selenium WebDriver.
2. Create Screen class and Pattern class.
3. Push the code to your GitHub repositories.

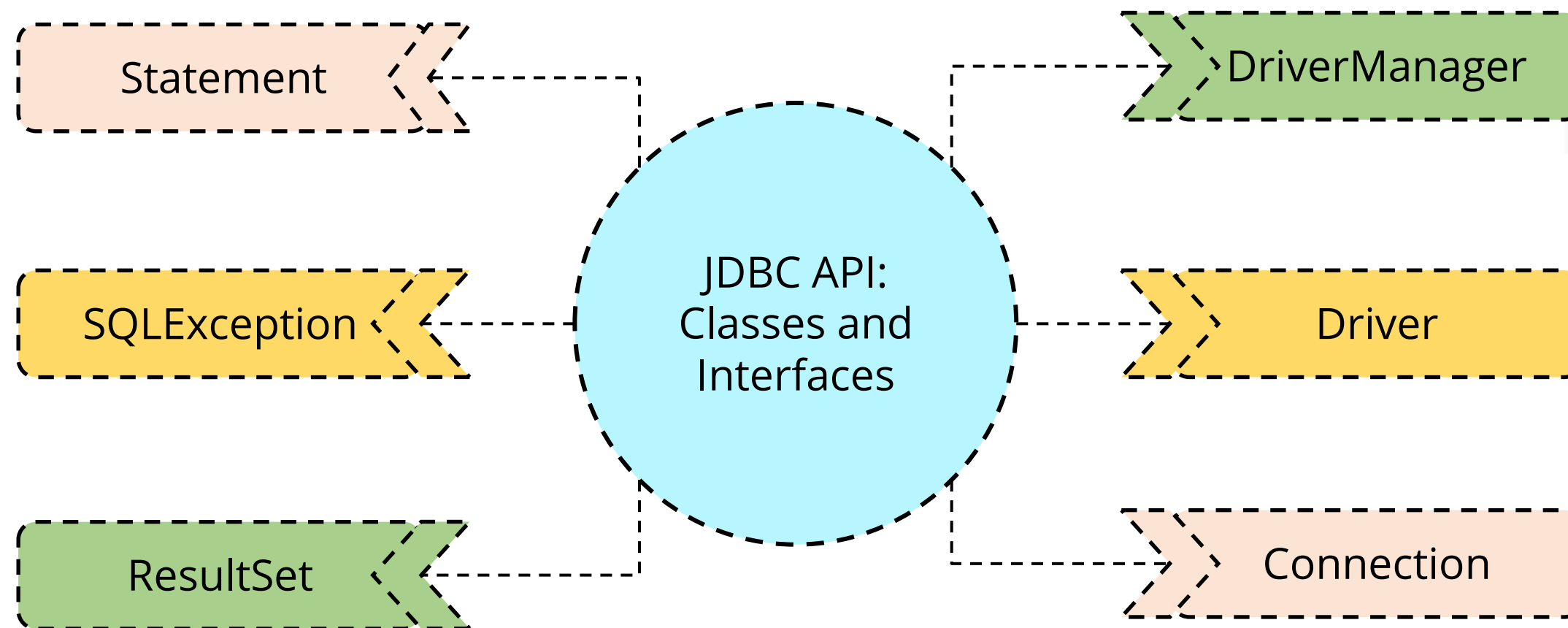


# FULL STACK

## Selenium and JDBC

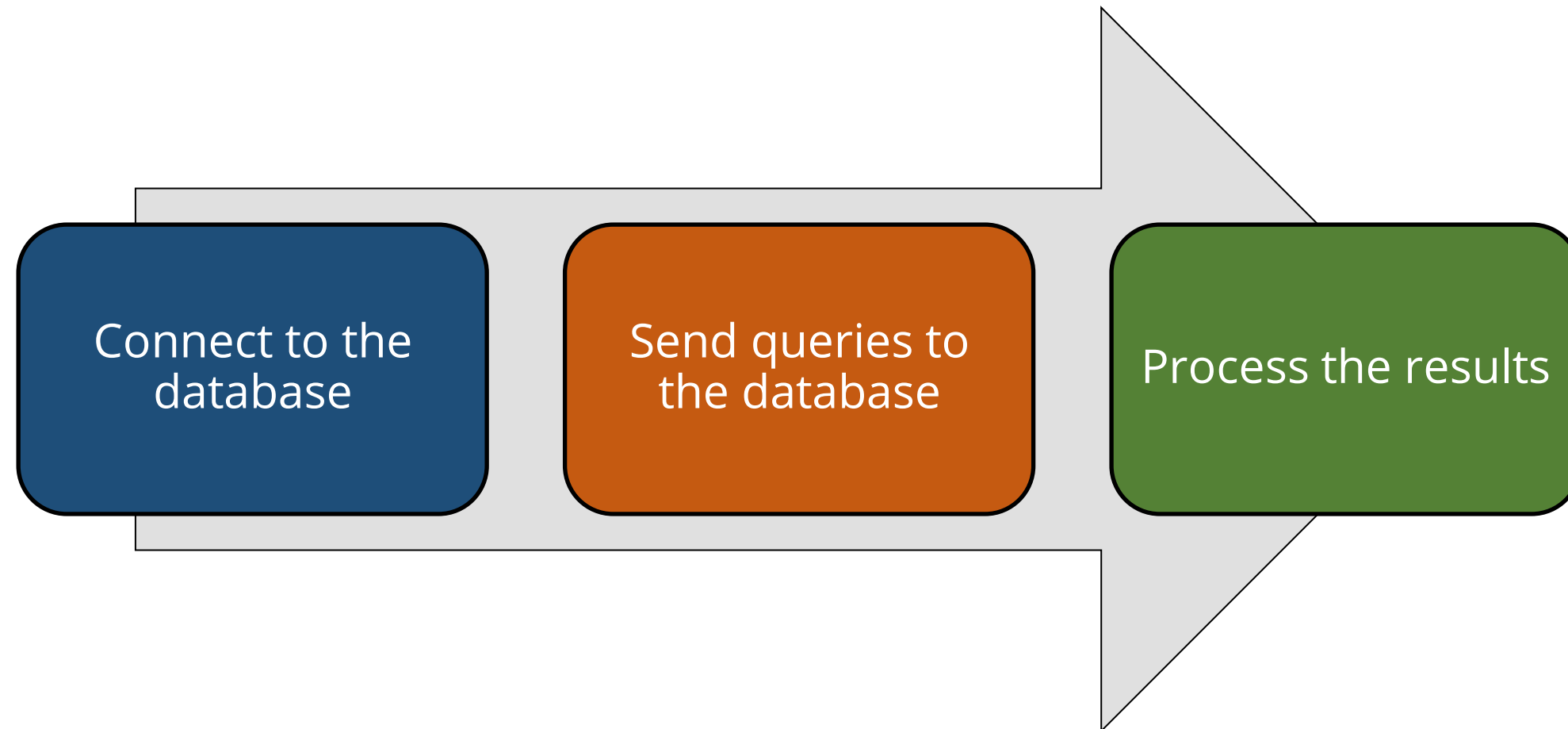
# Selenium and JDBC

JDBC is required to use Selenium WebDriver for database verification. JDBC is a Java-based data-access API used to connect to a wide range of databases.



# Database Testing Using Selenium

Three steps must be followed for database testing using Selenium.



# Selenium and JDBC



**Duration: 20 min.**

## **Problem Statement:**

Demonstrate the usage of JDBC in Selenium.

ASSISTED PRACTICE

# Assisted Practice: Guidelines

Steps to use JDBC in Selenium:

1. Create a table in database.
2. Write the JDBC code to integrate with Selenium.
3. Push the code to your GitHub repositories.





## Key Takeaways

- Software test automation refers to the activities and efforts that automate manual tasks and operations in a software test process using well-defined strategies and systematic solutions.
- Selenium WebDriver uses eight locators to find the elements on a webpage: ID, Name, Class, CSSSelector, Xpath, LineText, Partial Link Text, and TagName.
- AutoIT is an open source tool used to automate Windows and desktop application processes.
- Sikuli is a visual technology used to automate and test graphical user interfaces using images.



# Upload File and Handle Various Web Elements

Duration: 90 min.

## Problem Statement:

You are asked to develop a functionality using Selenium webdriver to handle various web elements.



LESSON-END PROJECT

# Before the Next Class

## You should know:

- Agile and scrum core concepts
- Fundamental concepts of Git and GitHub
- Basics of Java programming
- Implementation of OOPs and Collections
- Database handling
- Automation using Selenium webdriver





# Automating the Amazon Application Using Selenium WebDriver

Duration: 240 min.

## Project Objective:

As a Test Engineer, you are asked to automate the Amazon application using Selenium webdriver.



PHASE-END PROJECT

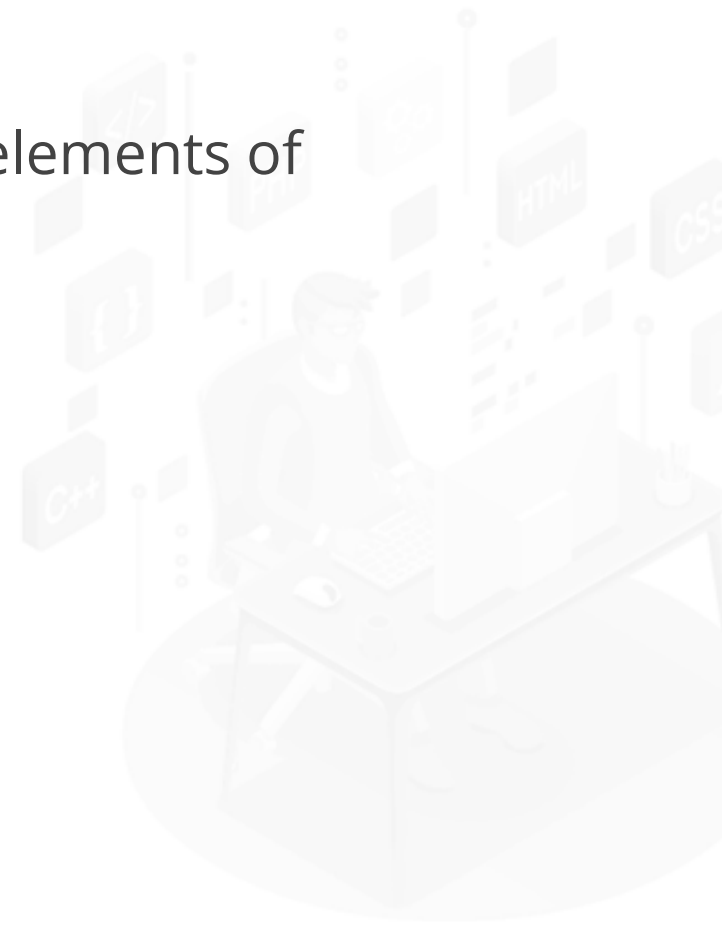
# Background of the Project Statement

The Amazon application has a huge database. The customers are taking more time to find a particular product. As you are a Test Engineer, your manager has asked you to write an automation script to get the list of different products in a particular category. Based on the list, it will be easy for the customers to select the product.



## You Are Asked to Do

- Create a Java class for Amazon application
- Open the browser and locate web elements using locators.
- Write automation script using page object design pattern class to store the web elements of web page.
- Perform transaction management using Selenium and JDBC.



# You Must Use the Following



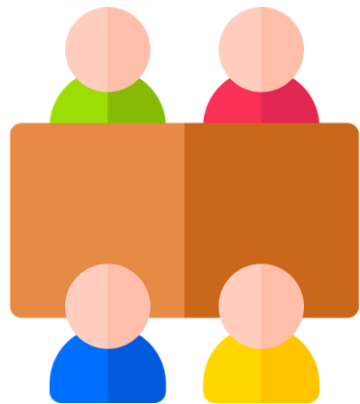
IDE: Eclipse or IntelliJ



Programming language: Java



Git and GitHub



Scrum



Selenium Standalone Server  
3.141.59



Specification Document