Automate Your Website with Selenium

A Day in the Life of a Full Stack Developer

Joe, a Full Stack Developer for Abq Inc., has been working hard and delivered most of the functions of a fully dynamic web application. They were also approved by the Quality Assurance team. However, the company is still facing bandwidth issues in basic automation testing due to a resource crunch.

During sprint planning, Joe has been assigned to perform high-level testing of the features that are incorporated in the application with the help of TestNG and enhance it with Jenkins.

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

Learning Objectives

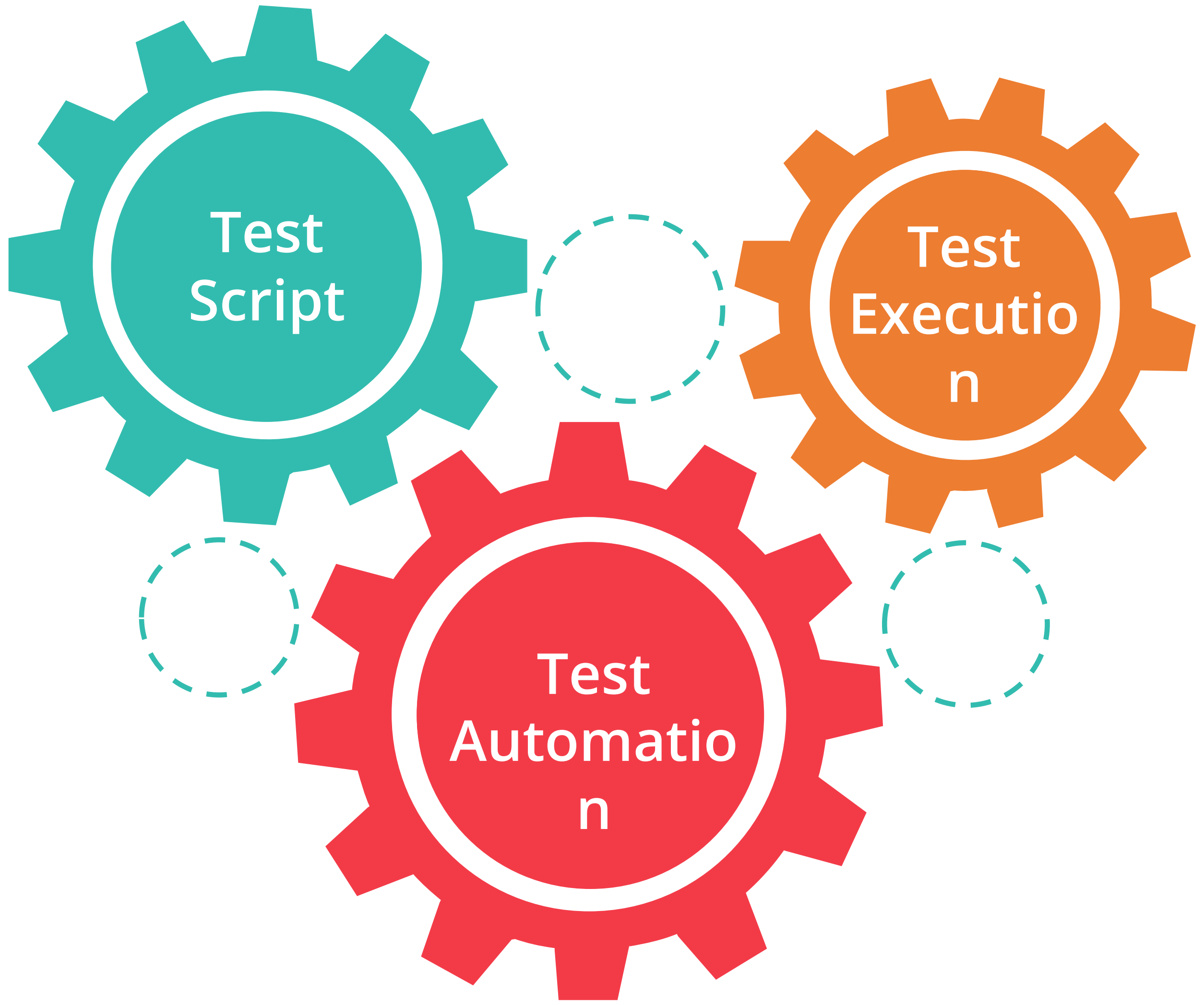
By the end of this lesson, you will be able to:

- 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 iframes, alerts, pop-ups, and sub-windows

What Is Selenium and How it Is Used in the Industry

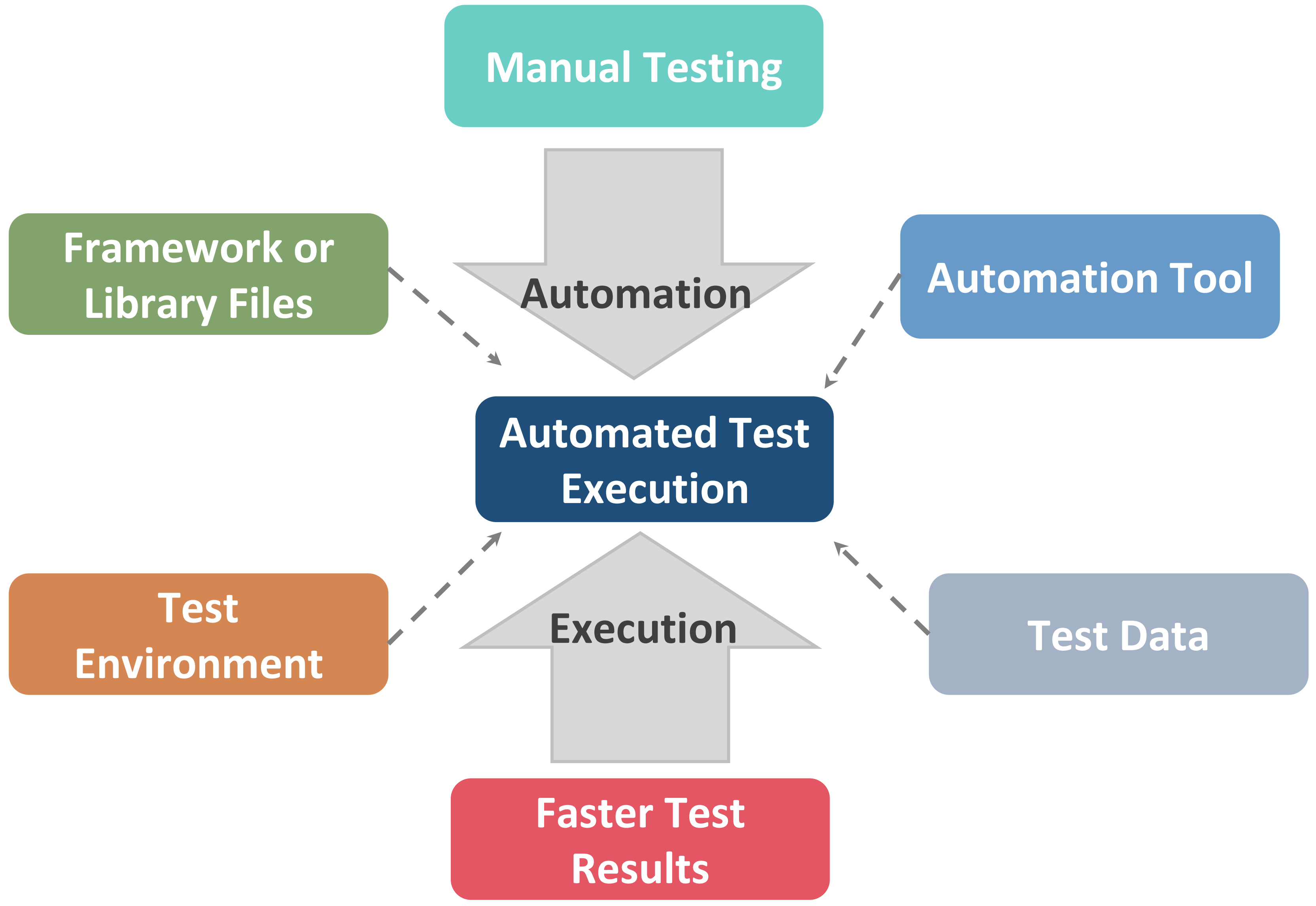
What Is Automation?

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.

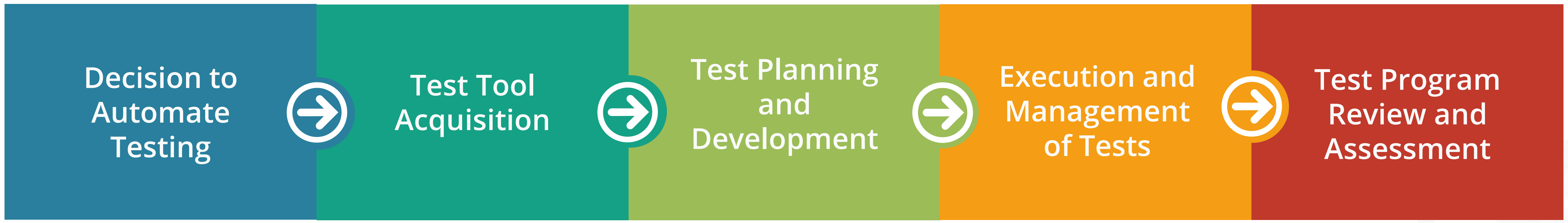


Automation for Agility

Test automation is the only way to achieve agility.



Automation Lifecycle

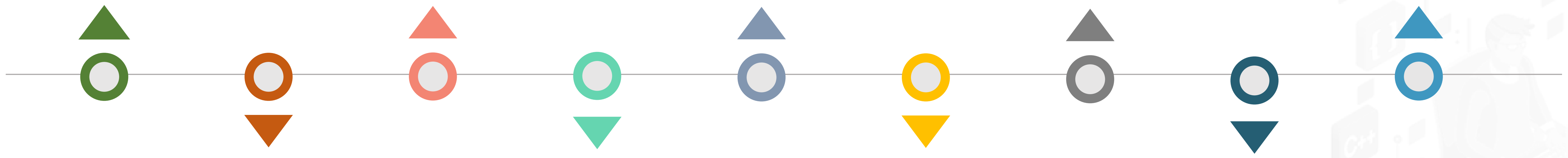

Decision to Automate Testing
Test Tool Acquisition
Test Planning and Development
Execution and Management of Tests
Test Program Review and Assessment

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



Selenium was invented by Jason R. Huggins and team in Thoughtworks in **2004**

The original name was **JavaScript Test Runner**

The name was changed to **Selenium RC**

Selenium IDE was created by wrapping the core code into IDE

Selenium Grid came into existence in **2008**

WebDriver was merged with Selenium RC in **2009**

In the Selenium Conference in Bangalore in **June 2016**, 3.0 beta was released only for attendees

Selenium 3.0 was released in **October 2016**

Selenium RC officially deprecated with the release of WebDriver 3.0

How Is Selenium Being Used in the Industry?

Plays a key role in DevOps

Provides faster feedback and agility

Provides improved coverage across platforms, browsers, and mobile devices in terms of quality

Helps in parallel execution of a large number of tests

Provides behavior-driven development with Cucumber and JBehave

Supports Mobile Test Automation along with Appium

FULL STACK

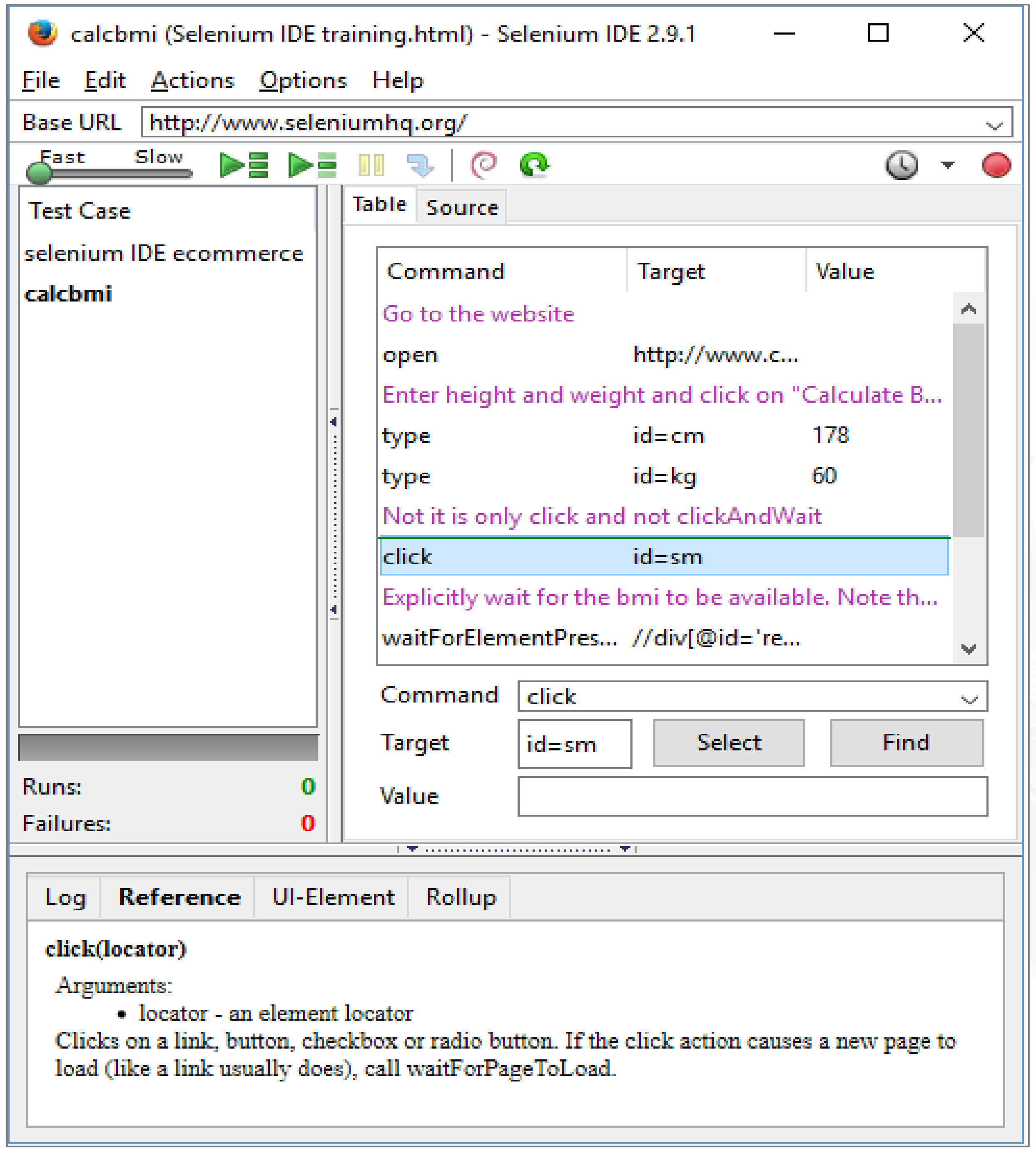
Features of Selenium

Features of Selenium IDE

- Record and playback
- Autofilling locators: ID, name, link, XPath, CSS, and DOM
- Dropdown selection of Selenium commands
- Feature 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]())
- 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



Base URL http://www.seleniumhq.org/

Test Case: selenium IDE ecommerce, **calcbmi**

Command	Target	Value
Go to the website		
open	http://www.c...	
Enter height and weight and click on "Calculate B...		
type	id=cm	178
type	id=kg	60
Not it is only click and not clickAndWait		
click	id=sm	
Explicitly wait for the bmi to be available. Note th...		
waitForElementPres...	//div[@id='re...	

Command: click
Target: id=sm
Value:

Runs: 0
Failures: 0

click(locator)
Arguments:
- locator - an element locator

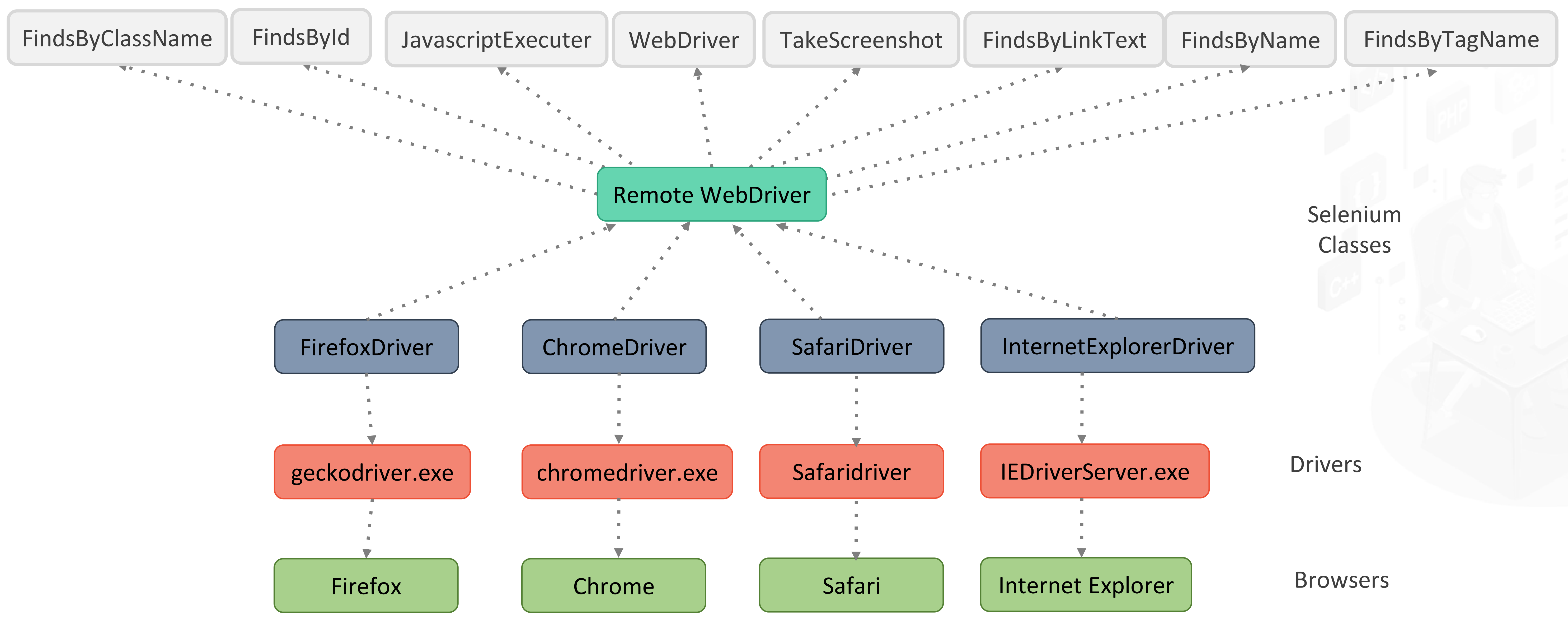
Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.

FULL STACK

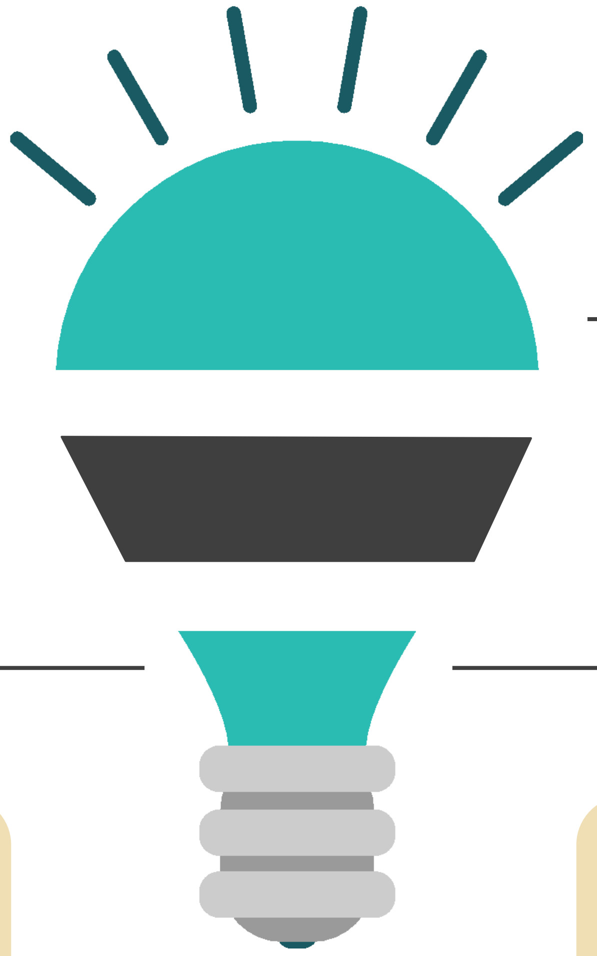
WebDriver Architecture

WebDriver Architecture

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



WebDriver Advantages



1. Keeps the cost down

2. Emulates the user

3. Keeps it simple and flexible

4. An open-source project

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 WebDriver is installed and integrated in Eclipse.

ASSISTED PRACTICE

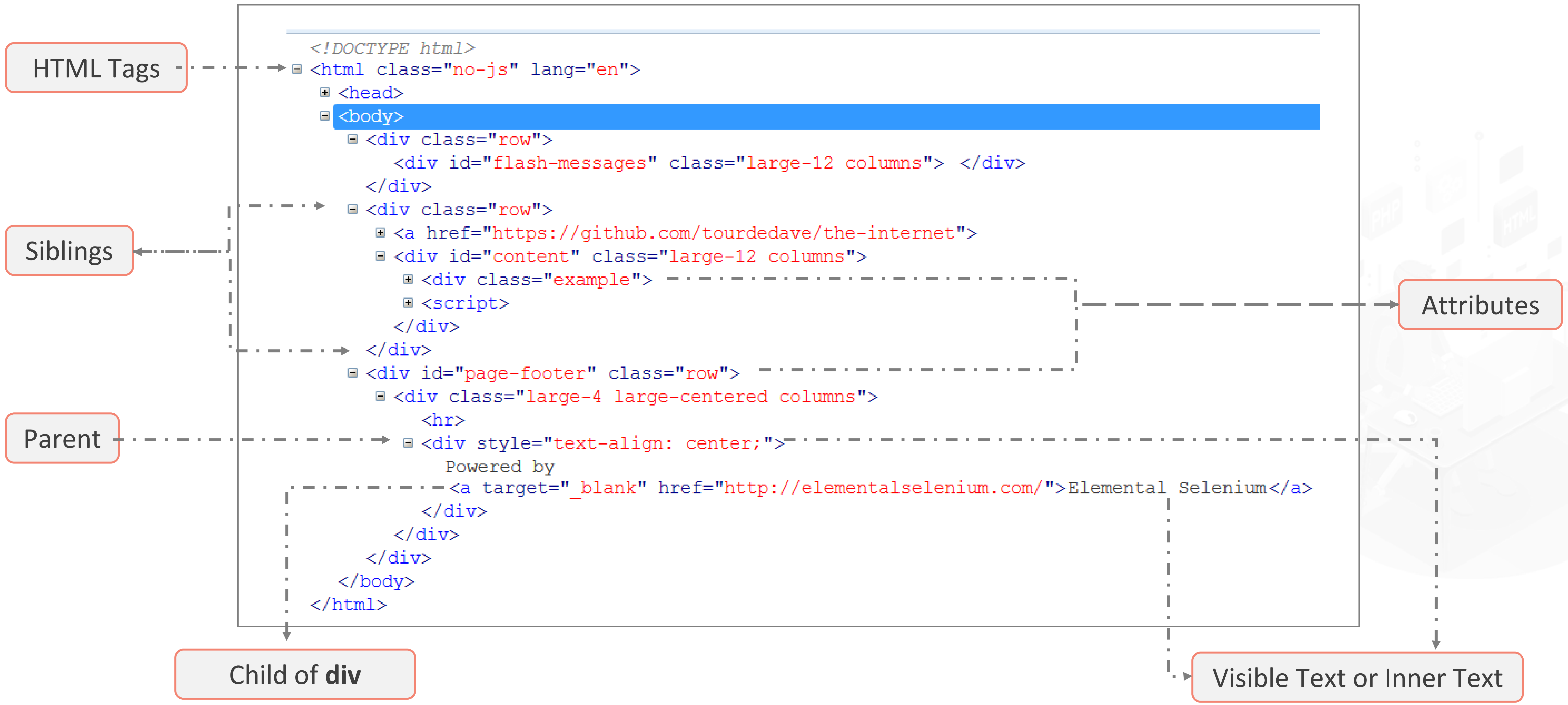
Assisted Practice: Guidelines

Steps to set up a webdriver:

1. Download Selenium Standalone Server jar
2. Launch Eclipse and create a Java project
3. Configure WebDriver with Eclipse
4. Push the code to GitHub repositories

Locating Web Page Elements

Typical HTML Page Elements



```
<!DOCTYPE html>
⊟ <html class="no-js" lang="en">
   ⊞ <head>
   ⊟ <body>
      ⊟ <div class="row">
            <div id="flash-messages" class="large-12 columns"> </div>
        </div>
      ⊟ <div class="row">
         ⊞ <a href="https://github.com/tourdedave/the-internet">
         ⊟ <div id="content" class="large-12 columns">
            ⊞ <div class="example">
            ⊞ <script>
           </div>
        </div>
      ⊟ <div id="page-footer" class="row">
         ⊟ <div class="large-4 large-centered columns">
              <hr>
            ⊟ <div style="text-align: center;">
                 Powered by
                 <a target="_blank" href="http://elementalselenium.com/">Elemental Selenium</a>
              </div>
           </div>
        </div>
     </body>
</html>
```

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 the 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 like searching all links inside the “div”.

findElement() Method

Syntax:

```
List<WebElement> allInputElements = driver.findElements(By.tagName(“input”));
    for (WebElement oneInputElemenetAtATime : allInputElements )
{
           System.out.println(oneInputElemenetAtATime.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 the elements.

WebDriver Locators

Selenium WebDriver uses eight locators to find the elements on a webpage. The object identifiers or locators supported by Selenium are:

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

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:

- clear()
- click()
- Submit()
- sendKeys()

- isEnabled()
- isDisplayed()
- isSelected()

- findElement()
- findElements()

- getCssValue()
- getAttribute()
- getText()
- getTagName()
- getScreenshotAs( )

- getLocation()
- getRectangle()
- getSize()

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 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?
• implicitlyWait()	• Amount of time the driver should wait while searching for an element (call to findElement) if it is not immediately available
• pageLoadTimeout()	• Amount of time to wait for a page to load completely
• setScriptTimeout()	• Amount of time to wait for an asynchronous script to finish execution

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 a specified time.

Synchronizing Automation

Handling synchronization is critical to create resilient tests.

Synchronization Categories

Unconditional Synchronization:

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

Conditional Synchronization:

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

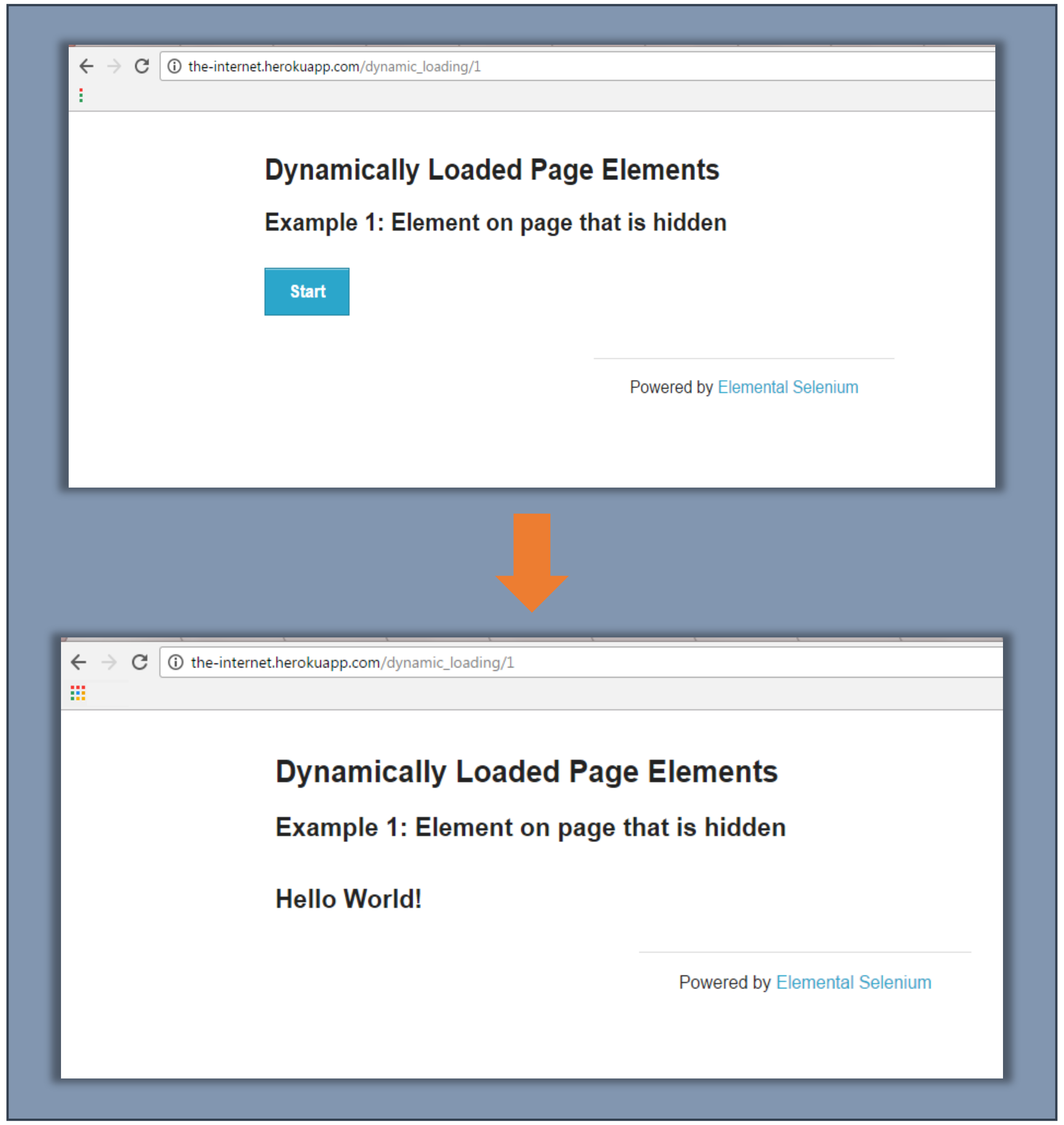
Why Use Waits?

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

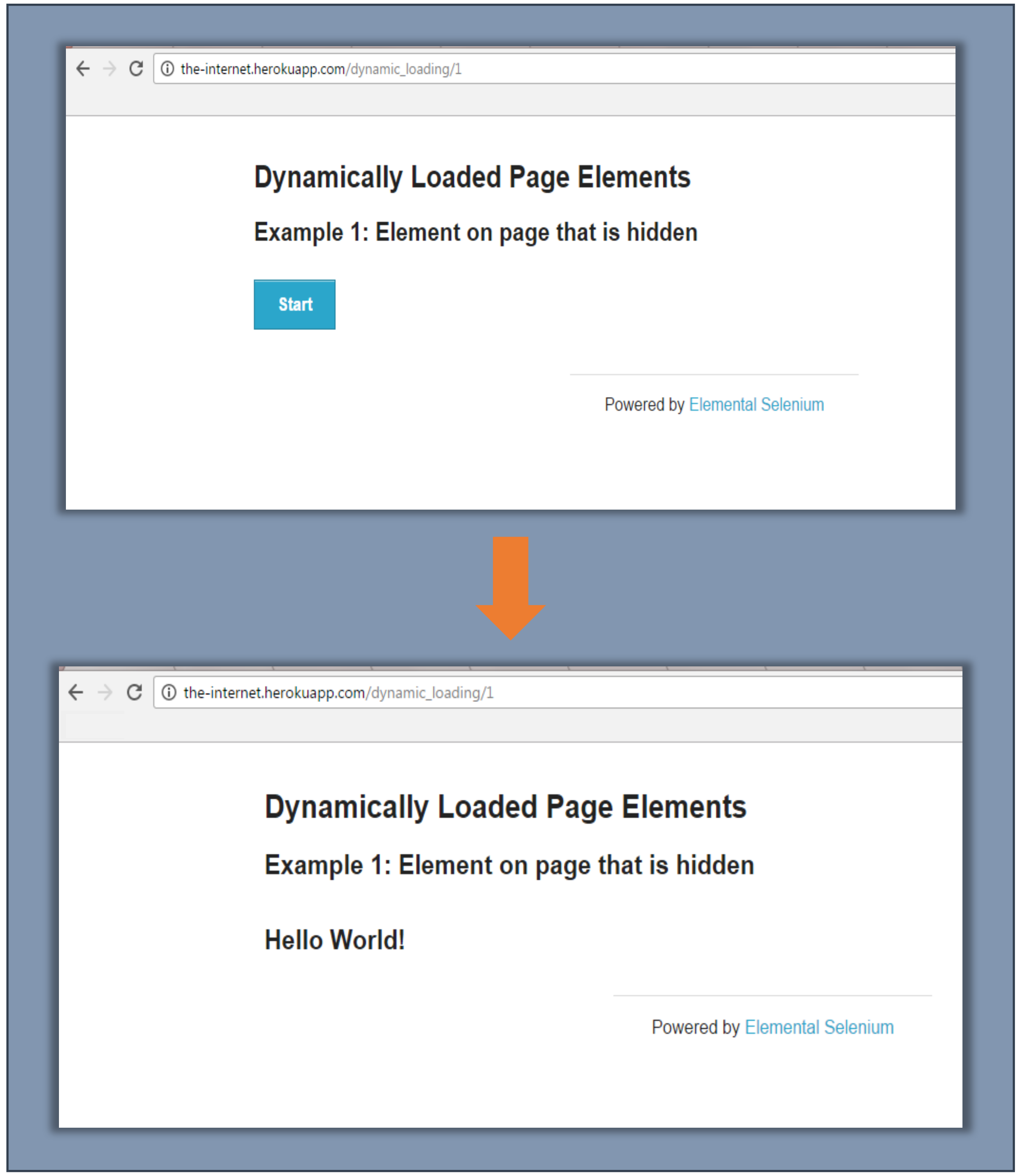

the-internet.herokuapp.com/dynamic_loading/1
Dynamically Loaded Page Elements
Example 1: Element on page that is hidden
Start
Powered by Elemental Selenium
the-internet.herokuapp.com/dynamic_loading/1
Dynamically Loaded Page Elements
Example 1: Element on page that is hidden
Hello World!
Powered by Elemental Selenium

Test Case:

- Go to 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


the-internet.herokuapp.com/dynamic_loading/1

Dynamically Loaded Page Elements

Example 1: Element on page that is hidden

Powered by Elemental Selenium

the-internet.herokuapp.com/dynamic_loading/1

Dynamically Loaded Page Elements

Example 1: Element on page that is hidden

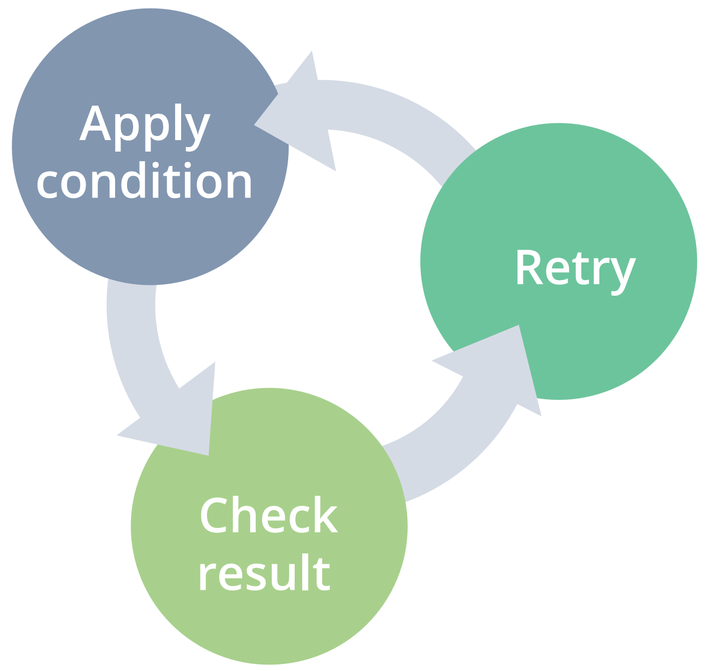
Hello World!

Powered by Elemental Selenium

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.

Locating Web Page Elements

Duration: 40 min.

Problem Statement:
Demonstrate multiple ways to locate web page elements.

Assisted Practice: Guidelines

Steps to locate web page elements:

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
8. Push the code to GitHub repositories

FULL STACK

Locating Elements with 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.

Absolute XPath	html/body/div/h2/a - absolute path to get the anchor element
Relative XPath	//input – get all “input” elements on the page
Using Index	//input[1] – getting all first “input” elements of a parent on the page
Using Attribute Values	//input[@value='Login'] – “input” with “value” attribute “Login”
Two Attributes – AND	//input[@value='Login'][@type='submit'] – “input” matching two attribute values
Two Attributes – OR	//input[@value='Login‘ or @type='submit'] – “input” matching one of the attributes
Partial Match - contains	//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 an error.

Xpath vs. CSS

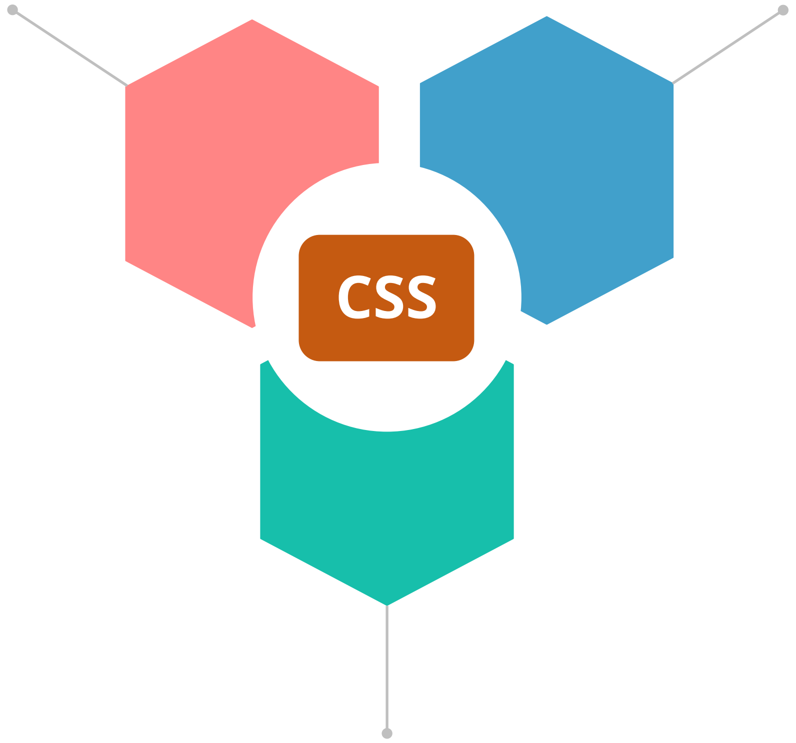
Below is the key difference between Xpath and CSS locating techniques:

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 the elements by clicking, getting values, typing, sending keys, etc.

CSS can change styling by declaring which bits of the markup it wants to alter with 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:

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

Locating Elements through Xpath and CSS

Duration: 30 min.

Problem Statement:
Using Selenium WebDriver, write a program to log into Facebook by locating elements using Xpath or CSS.

Assisted Practice: Guidelines

Steps to locate elements through Xpath and CSS:

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

simplilearn

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	• Select a radio button • Check if a radio button is selected	• Click() • isSelected()
Checkbox	• Select a checkbox • Deselect a checkbox • Check if a checkbox is selected	• Click() • Click() • isSelected()

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
• Select an option	• selectByVisibleText() • selectByValue() • selectByIndex()
• Check what options are available	• getOptions()
• Check which option is selected first	• getFirstSelectedOption()
• Check if it is a drop-down or multi-select	• isMultiple()

Multi-Select Options

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

What can you do ?	Select class methods
• Deselect an option	• deselectByVisibleText() • deselectByValue() • deselectByIndex()
• Get all selected options	• getAllSelectedOptions()
• Deselect all	• deselectAll()

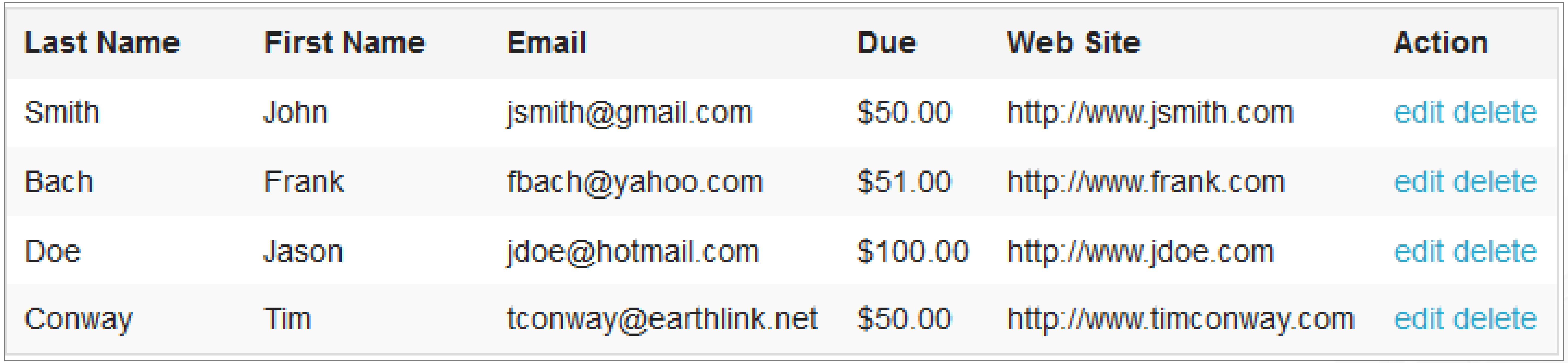
HTML Tags for Tables

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	edit delete
Bach	Frank	fbach@yahoo.com	$51.00	http://www.frank.com	edit delete
Doe	Jason	jdoe@hotmail.com	$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	$50.00	http://www.timconway.com	edit delete

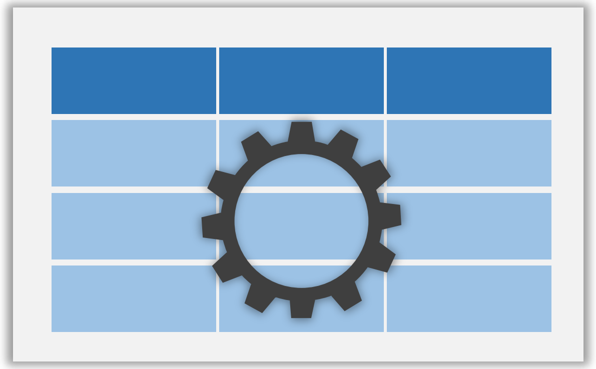
HTML Table Methods

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:
Using Selenium WebDriver, write a program to fill up a registration form with different HTML components.

Assisted Practice: Guidelines

Steps to handle web elements:

1. Navigate to the registration page
2. Automate all the fields of the form
3. Push the code to GitHub repositories

Working with External Elements

External Elements

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

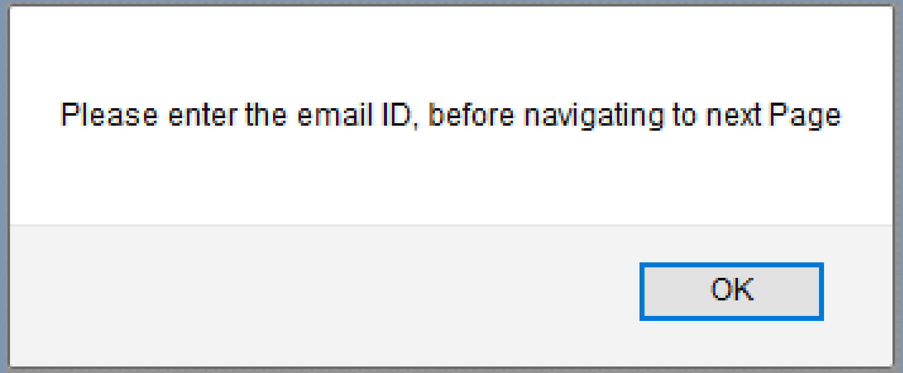
- JavaScript alerts
- Browser Pop-up windows
- Pop-up windows. For example: file upload, print dialog
- Frames and iframes

JavaScript Alerts

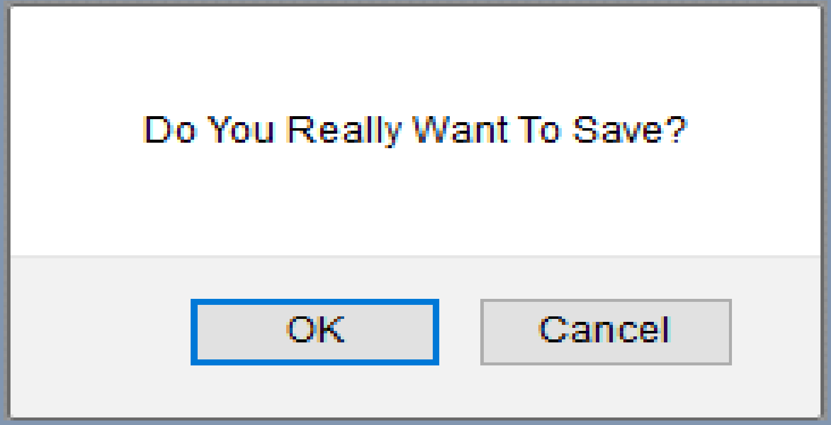
WebDriver provides APIs in **Alert** class to handle JavaScript popups.

JavaScript has three types of pop-ups:

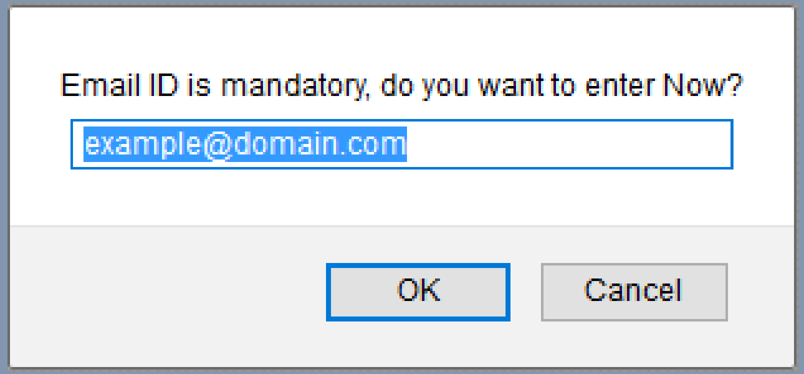
Simple Alert Box: Contains a message and OK button


Please enter the email ID, before navigating to next Page

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


Do You Really Want To Save?

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

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).

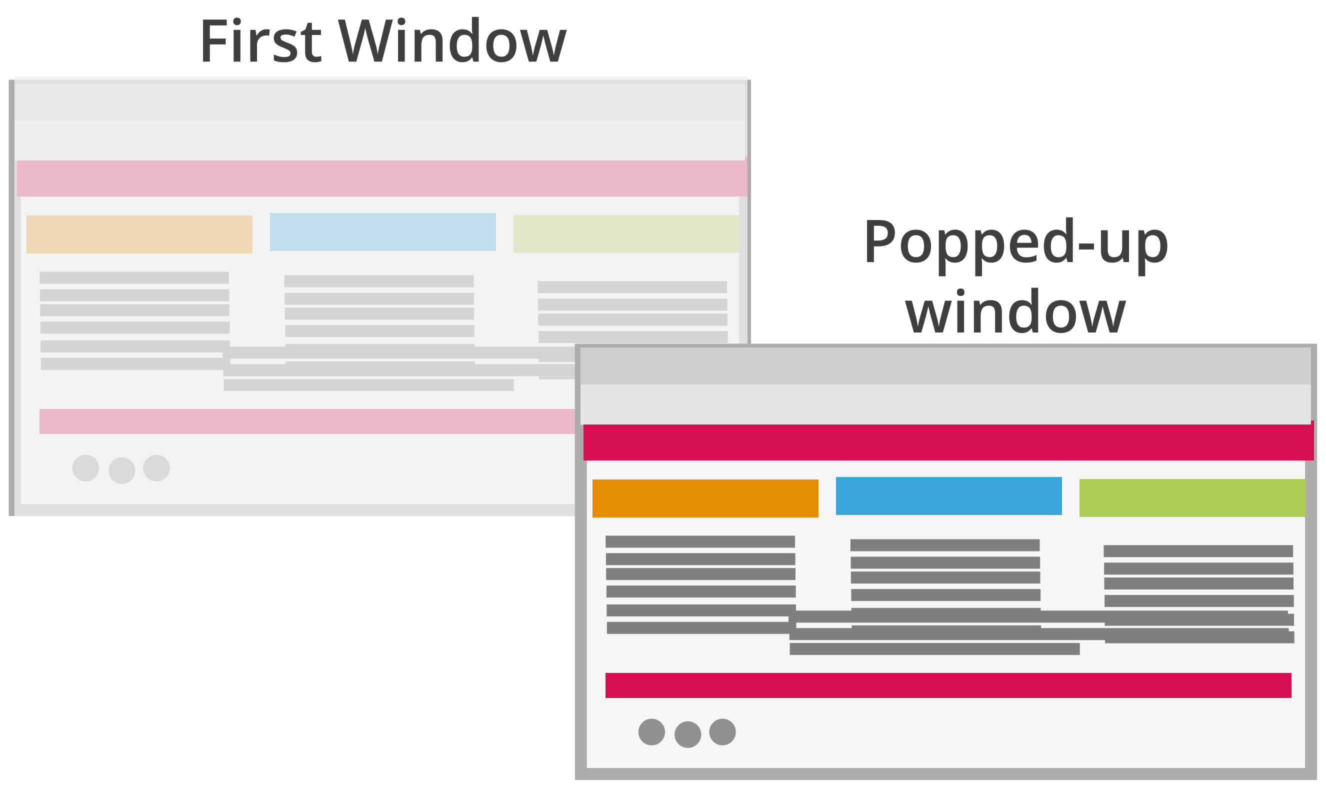
alert.dismiss() // Dismisses the alert (for example, clicking **Cancel** button).

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.

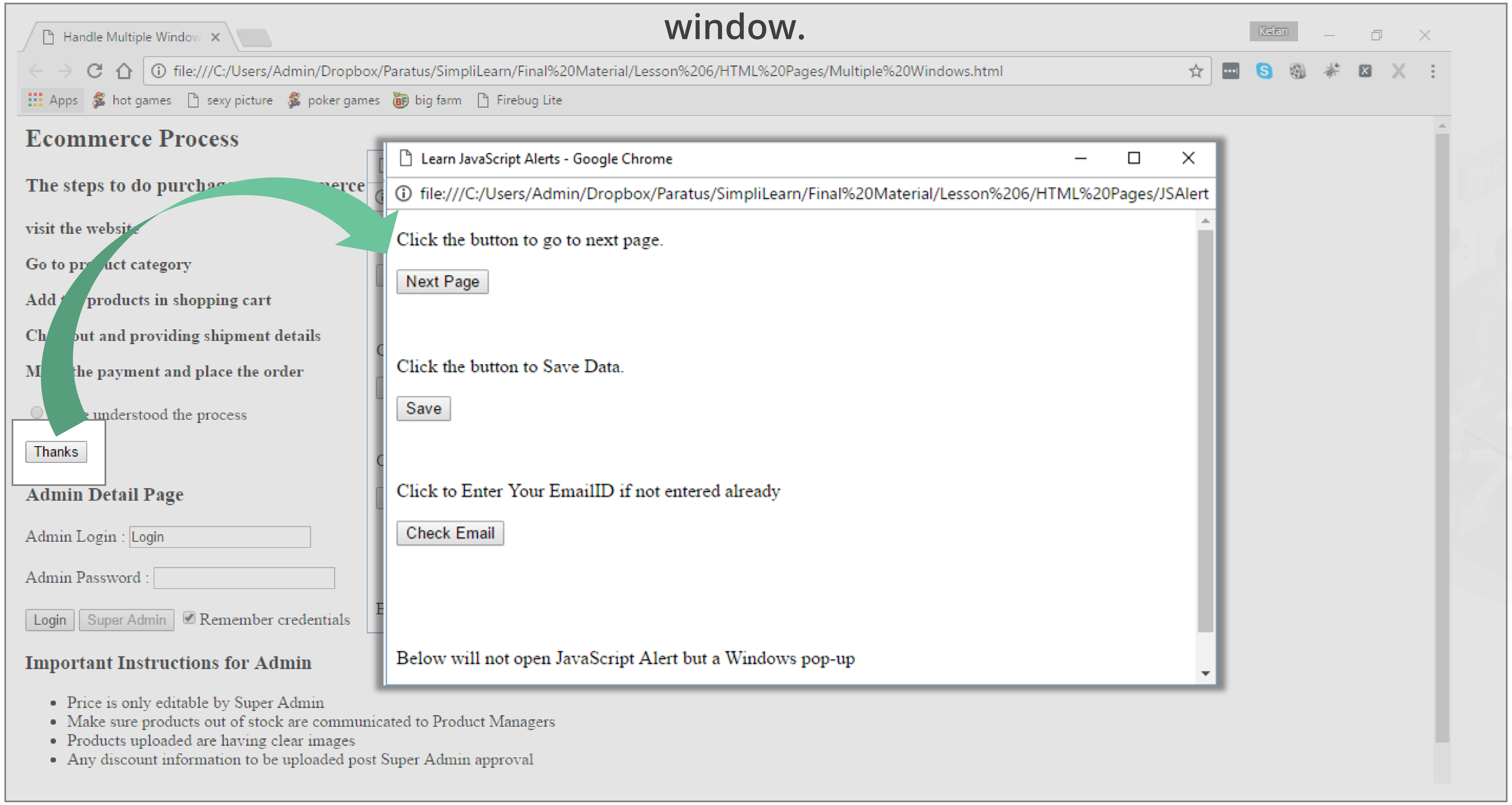
Browser Pop-Ups



- 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 fetch 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 the new window, but WebDriver instance is still on the previous window.


Handle Multiple Window ×

file:///C:/Users/Admin/Dropbox/Paratus/SimpliLearn/Final%20Material/Lesson%206/HTML%20Pages/Multiple%20Windows.html

Apps hot games sexy picture poker games big farm Firebug Lite

Ecommerce Process

The steps to do purcha... ...merce

visit the websit...

Go to pr...uct category

Add ...products in shopping cart

Ch...out and providing shipment details

M... the payment and place the order

...e understood the process

Thanks

Admin Detail Page

Admin Login : Login

Admin Password :

Login Super Admin ☑ Remember credentials

Important Instructions for Admin

- Price is only editable by Super Admin
- Make sure products out of stock are communicated to Product Managers
- Products uploaded are having clear images
- Any discount information to be uploaded post Super Admin approval

Learn JavaScript Alerts - Google Chrome

file:///C:/Users/Admin/Dropbox/Paratus/SimpliLearn/Final%20Material/Lesson%206/HTML%20Pages/JSAlert

Click the button to go to next page.

Next Page

Click the button to Save Data.

Save

Click to Enter Your EmailID if not entered already

Check Email

Below will not open JavaScript Alert but a Windows pop-up

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.

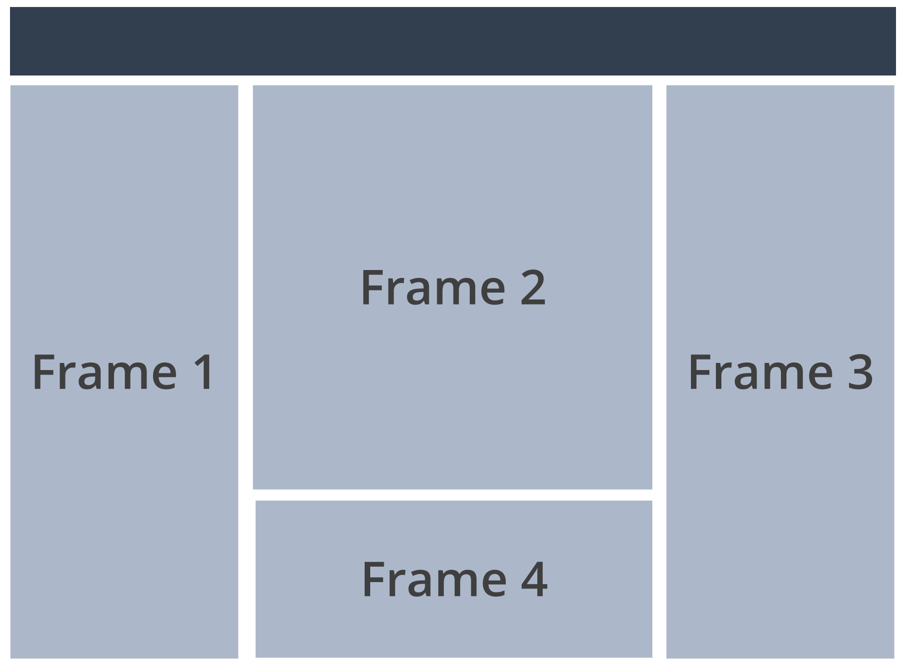
The two types of parameters the window() method can take are:

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,

- 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 work with external elements.

ASSISTED PRACTICE

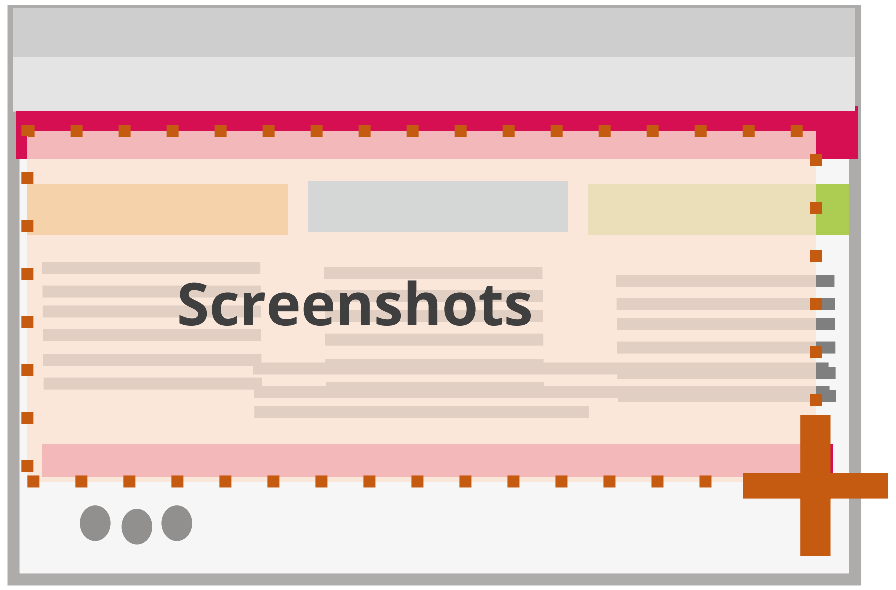
Assisted Practice: Guidelines

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

Screenshots

Taking Screenshots



You may need screenshots in Test Automation for:

- Client expectations
- Regulatory requirements
- Developers aid in debugging application defects
- Testers to debug Test Script defects

Taking Screenshots

TakesScreenshot is an interface that provides **getScreenshotAs()** method to capture the screenshot.

It supports the following three output formats:

Screenshot as file

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

Screenshot as base64 data

```
String screenshotBase64 = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.BASE64);
```

Screenshot as raw bytes

```
byte[]
screenshotAsArrayOfBytesFormPNG=((TakesScreenshot)driver).getScreenshotAs(OutputType.BYTES);
```

Screenshots

Duration: 15 min.

Problem Statement:
Using Selenium WebDriver, write a program to take screenshot of your automation screen, while the script is running in background.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to take screenshots:

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

AutoIT

AutoIT

- AutoIT is a freeware for automating Windows GUI.
- It is typically used to automate routine tasks in Microsoft Windows.
- It has a built-in editor with syntax highlighting feature.
- AutoIT scripts can be made into standalone executables.

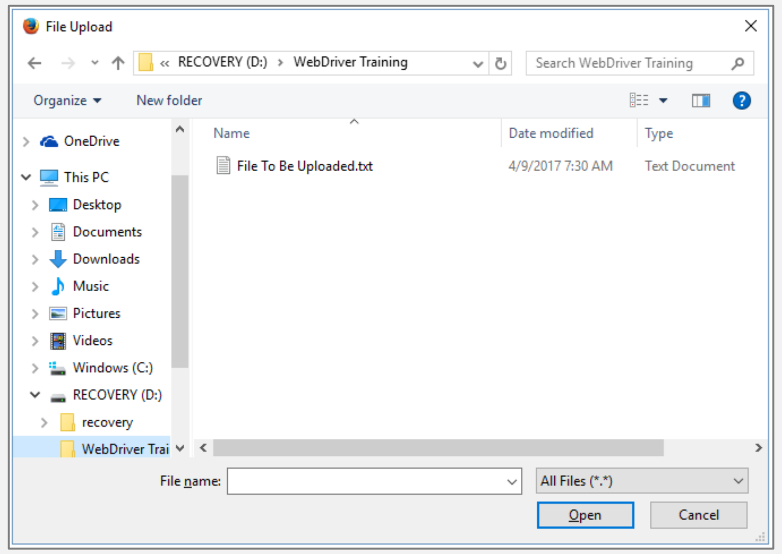
Steps To Upload File Using AutoIT

Identify Locators(controls) on the FileUpload dialog

Write a script in AutoIT script editor to type file along with path under **FileName** field

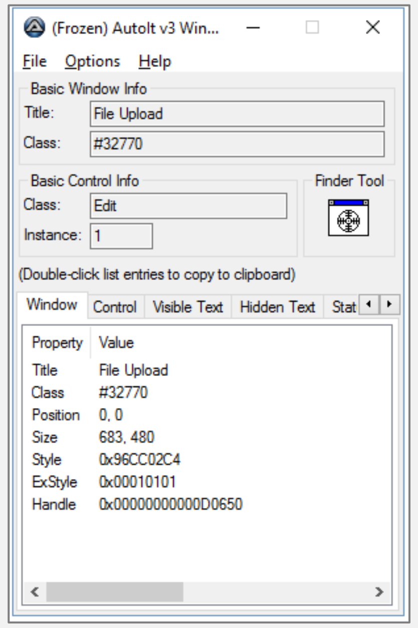
Invoke the executable file from WebDriver tests

- Like WebDriver, to locate elements on a page, you must identify Windows elements on **FileUpload** window.



File To Be Uploaded.txt 4/9/2017 7:30 AM Text Document

- For this, AutoIT provides a finder tool called **AutoIT Window Info.**



Title: File Upload
Class: #32770

Class: Edit
Instance: 1

(Double-click list entries to copy to clipboard)

Property	Value
Title	File Upload
Class	#32770
Position	0, 0
Size	683, 480
Style	0x96CC02C4
ExStyle	0x00010101
Handle	0x00000000000D0650

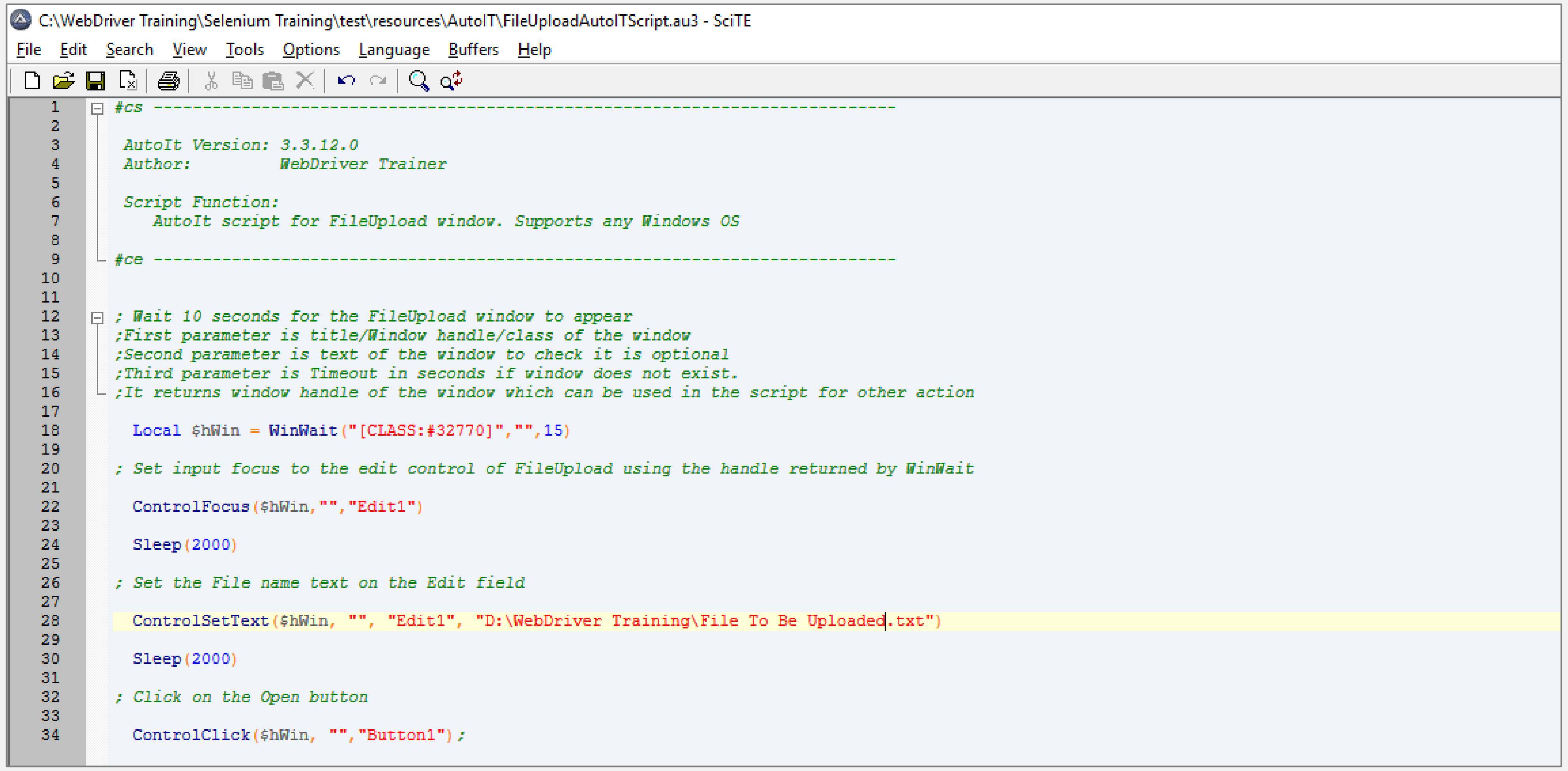
Steps To Upload File Using AutoIT

Identify Locators(controls) on the FileUpload dialog

Write a script in AutoIT script editor to type file along with path under FileName field

Invoke the executable file from WebDriver tests

- Create the script.
- Compile it to create the executable file.


C:\WebDriver Training\Selenium Training\test\resources\AutoIT\FileUploadAutoITScript.au3 - SciTE
File Edit Search View Tools Options Language Buffers Help

```
#cs ----------------------------------------------------------------------------

 AutoIt Version: 3.3.12.0
 Author:         WebDriver Trainer

 Script Function:
    AutoIt script for FileUpload window. Supports any Windows OS

#ce ----------------------------------------------------------------------------


; Wait 10 seconds for the FileUpload window to appear
;First parameter is title/Window handle/class of the window
;Second parameter is text of the window to check it is optional
;Third parameter is Timeout in seconds if window does not exist.
;It returns window handle of the window which can be used in the script for other action

  Local $hWin = WinWait("[CLASS:#32770]","",15)

; Set input focus to the edit control of FileUpload using the handle returned by WinWait

  ControlFocus($hWin,"","Edit1")

  Sleep(2000)

; Set the File name text on the Edit field

  ControlSetText($hWin, "", "Edit1", "D:\WebDriver Training\File To Be Uploaded.txt")

  Sleep(2000)

; Click on the Open button

  ControlClick($hWin, "","Button1");
```

Steps To Upload File Using AutoIT

Identify Locators(controls) on the FileUpload dialog

Write a script in AutoIT script editor to type file along with path under FileName field

Invoke the executable file from WebDriver tests

- In your test, invoke the FileUpload dialog using WebDriver.
- Call the AutoIT executable file created in the previous step.

Handling File Uploads

Duration: 60 min.

Problem Statement:
Using Selenium WebDriver, write a program to upload a file using AutoIT.

Assisted Practice: Guidelines

Steps to demonstrate Auto IT:

1. Install and configure Auto IT
2. Handle file upload by SendKeys
3. Handle file upload by AutoIT Script
4. Push the code to GitHub repositories

Key Takeaways

- Selenium is an open-source, portable, and automated testing suite for web applications.
- WebDriver keeps the cost down and emulates the user.
- Selenium WebDriver uses eight locators to find elements on a web page: 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.

Automate a Web Application

Duration: 60 min.

Problem Statement:

Develop a project where you have to:

- Write automation script to automate registration and login operations of a website.

LESSON-END PROJECT