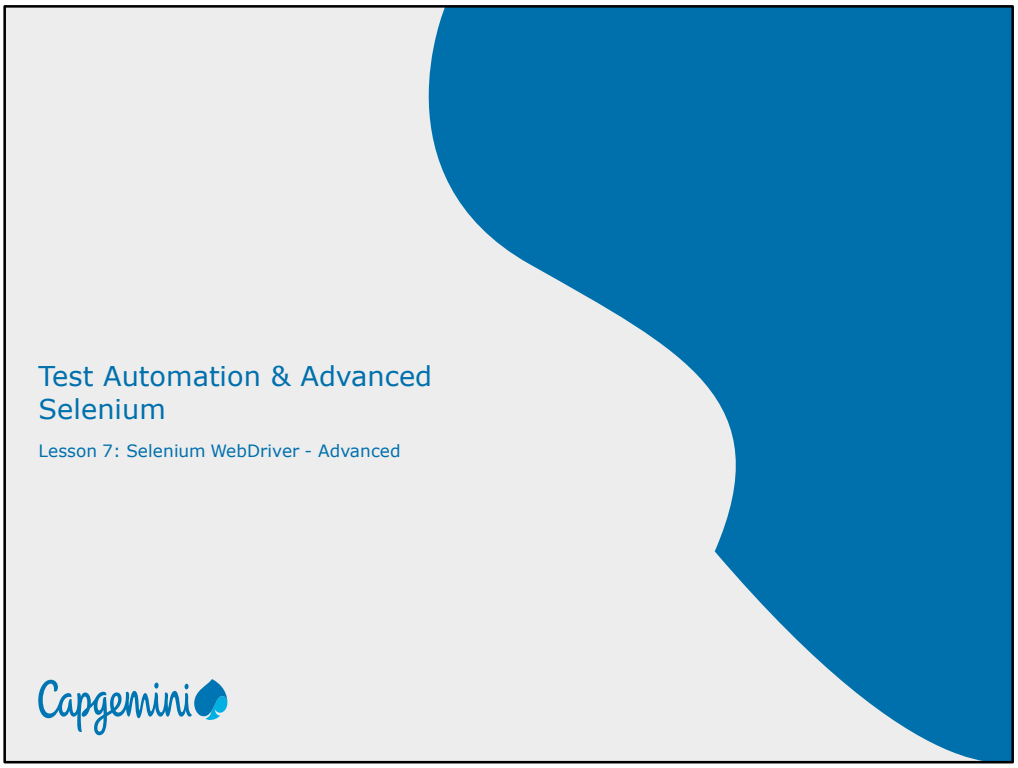

Test Automation & Advanced Selenium
Lesson 7: Selenium WebDriver - Advanced
Capgemini

Lesson Objectives

- Overview of Cross Browser Testing
- Cross Browser Testing in Selenium WebDriver
- Launching Firefox Browser With Selenium 3 & GeckoDriver
- Launching Edge Browser using Microsoft Edge Driver with Selenium 3
- Introduction to Headless Browsers
- Other Important Browsers
- Introduction to Selenium Grid
- What is Selenium Grid?
- Selenium Grid Architecture – The Hub & The Node
- Selenium Grid Architecture – Configuring Hub and Node
- Selenium Grid Architecture – RemoteWebDriver
- DesiredCapabilities and Profile Setting in Selenium WebDriver

7.1: Selenium WebDriver - Advanced

Overview of Cross Browser Testing

- Cross Browser Testing is a process to test web applications across multiple browsers
- Cross browser testing involves checking compatibility of your application across multiple web browsers and ensures that your web application works correctly across different web browsers
- Cross-browser testing has very little to do with what a web site looks like, and a lot more to do with how it functions
- With wide range of web browsers available, just testing your web application on single web browser is not enough
- Selenium WebDriver has support for different web browsers and for each web browser the API provides browser driver classes
- Selenium WebDriver supports HtmlUnitDriver, Firefox Driver, Chrome Driver, Opera Driver, MS Edge Driver, MS Internet Explorer Driver, Safari Driver, GhostDriver (PhamtomJS) and so on.

Overview of Cross Browser Testing

With wide range of web browsers available, end users using different web browsers to access your web applications, it has now become crucial to test web applications on multiple browsers. So just testing your web application on single web browser is not enough. You need to make sure that your web application works fine across multiple browser. Cross Browser Testing is a process to test web applications across multiple browsers. Cross browser testing involves checking compatibility of your application across multiple web browsers and ensures that your web application works correctly across different web browsers. Cross-browser testing has very little to do with what a web site looks like, and a lot more to do with how it functions.

7.1: Selenium WebDriver - Advanced

Cross Browser Testing in Selenium WebDriver

- ChromeDriver
  - ChromeDriver is a standalone server which implements WebDriver's wire protocol for Chromium, the open source project that Google Chrome is based on
  - ChromeDriver is available for Chrome on Android and Chrome on Desktop (Mac, Linux, Windows, and ChromeOS)
  - In order to use ChromeDriver, you should specify its location using the webdriver.chrome.driver system property

```
//Setting System property for ChromeDriver
System.setProperty("webdriver.chrome.driver","/path/to/chromedriver”);

//Create a new instance of a ChromeDriver
WebDriver driver = New ChromeDriver();

//Launching Chrome browser and loading Google home page
driver.get("https://www.google.co.in");

//Closing browser
driver.close();
```

7.1: Selenium WebDriver - Advanced

Cross Browser Testing in Selenium WebDriver (Cont.)

- InternetExplorerDriver
  - Internet Explorer driver is a standalone server which implements WebDriver's wire protocol
  - InternetExplorerDriver works only with Windows system and the execution speed is slow Comparatively to other browsers.
  - In order to use InternetExplorerDriver, you should specify its location using the webdriver.ie.driver system property

```
//Setting System property for InternetExplorerDriver
System.setProperty("webdriver.ie.driver","/path/to/IEDriver”);

//Create a new instance of a InternetExplorerDriver
WebDriver driver = New InternetExplorerDriver();

//Launching Internet Explorer browser and loading Google home page
driver.get("https://www.google.co.in");

//Closing browser
driver.close();
```

7.1: Selenium WebDriver - Advanced

Cross Browser Testing in Selenium WebDriver (Cont.)

- FirefoxDriver
  - Selenium WebDriver does not require any property to be set before launching Firefox browser
  - Selenium WebDriver provides default support for Firefox browser & it automatically references the default location of the Firefox.exe

```
//Create a new instance of a FirefoxDriver
WebDriver driver = New FirefoxDriver();

//Launching Firefox browser and loading Google home page
driver.get("https://www.google.co.in");

//Closing browser
driver.close();
```

7.1: Selenium WebDriver - Advanced

Cross Browser Testing in Selenium WebDriver (Cont.)

- SafariDriver
  - SafariDriver comes bundled with the Selenium server's package
  - It is readily available to execute the test scripts on Safari
  - Thus, user can directly launch the Safari Browser similar to that of Mozilla Firefox

```
//Create a new instance of a SafariDriver
WebDriver driver = New SafariDriver();

//Launching Safari browser and loading Google home page
driver.get("https://www.google.co.in");

//Closing browser
driver.close();
```

7.1: Selenium WebDriver – Advanced

Launching Firefox Browser With Selenium 3 & GeckoDriver

- GeckoDriver
- It is the latest Firefox driver by Selenium that is supported by Marionette protocol of Firefox
- Gecko is the browser engine for Firefox
- When you develop your tests, geckodriver acts as a proxy between your tests and the Firefox browser
- As selenium 3 does not have any native implementation of Firefox, all the driver commands will be directed through geckodriver
- geckodriver is an executable that starts the server on your system. All your tests communicate to this server in translating your browser commands to required actions on the Firefox browser
- You must be using Selenium 3.3.1 or later & Firefox 55 and greater
- You must download geckodriver for different platforms

Launching Firefox Browser With Selenium 3 & GeckoDriver

GeckoDriver: It is the latest Firefox driver by Selenium that is supported by Marionette protocol of Firefox. gecko is the browser engine for Firefox. The gecko browser engine is opensource and it can be accessed by developers in their applications to render webpages. When you develop your tests, geckodriver acts as a proxy between your tests and the Firefox browser. As selenium 3 does not have any native implementation of Firefox, all the driver commands will be directed through geckodriver. geckodriver is an executable that starts the server on your system. All your tests communicate to this server in translating your browser commands to required actions on the Firefox browser. geckodriver is not yet feature complete. Support is best in Firefox 54 and greater, generally the more recent the Firefox version, the better the experience as they have more bug fixes and features. Some features will only be available in the most recent Firefox versions, and we strongly advise using the latest Firefox Nightly with geckodriver.

7.1: Selenium WebDriver - Advanced

Launching Firefox Browser With Selenium 3 & GeckoDriver (Cont.)

```
//Create a new instance of a FirefoxDriver
WebDriver driver = new FirefoxDriver();

//Setting System property for geckodriver
System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");

//Launching Firefox browser and loading Google home page
driver.get("https://www.google.co.in");

//Closing browser
driver.close();
```

7.1: Selenium WebDriver - Advanced

Launching Edge Browser using Microsoft Edge Driver with Selenium 3

- To start using WebDriver with Microsoft Edge browser, make sure you have Windows 10 on your machine and download the specified Microsoft WebDriver server version for your build
- In order to launch Edge Browser, we need to specify the system property with the path of the MicrosoftWebDriver.exe file

```
//Create a new instance of a EdgeDriver
WebDriver driver = new EdgeDriver();

//Setting System property for MicrosoftWebDriver
System.setProperty("webdriver.edge.driver",
"/path/to/microsoftwebdriver");

//Launching Microsoft Edge browser and loading Google home page
driver.get("https://www.google.co.in");

//Closing browser
driver.close();
```

Demo – Cross Browser Testing in Selenium WebDriver

- Running tests in :
  - ChromeDriver
  - InternetExplorerDriver
  - FirefoxDriver
- Launching Firefox Browser With Selenium 3 & GeckoDriver
- Launching Edge Browser using Microsoft Edge Driver with Selenium 3

7.1: Selenium WebDriver - Advanced

Introduction to Headless Browsers

Headless Browser

- Web browser without a graphical user interface
- Normally, interaction with a website are done with mouse and keyboard using a browser with a GUI
- While most headless browser provides an API to manipulate the page/DOM, download resources etc.
- So instead of, for example, actually clicking an element with the mouse, a headless browser allows you to click an element by code
- Headers, Local storage and Cookies work the same way
- List of Headless Browsers
  - PhanthomJS
  - HtmlUnit
  - TrifleJS
  - Splash

Headless browsers are typically used in following situations:
1. You have a central build tool which does not have any browser installed on it. So to do the basic level of sanity tests after every build you may use the headless browser to run your tests.
2. You want to write a crawler program that goes through different pages and collects data, headless browser will be your choice. Because you really don't care about opening a browser. All you need is to access the webpages.
3. You would like to simulate multiple browser versions on the same machine. In that case you would want to use a headless browser, because most of them support simulation of different versions of browsers. We will come to this point soon.

Things to pay attention to before using headless browser
Headless browsers are simulation programs, they are not your real browsers. Most of these headless browsers have evolved enough to simulate, to a pretty close approximation, like a real browser. Still you would not want to run all your tests in a headless browser. JavaScript is one area where you would want to be really careful before using a Headless browser. JavaScript are implemented differently by different browsers. Although JavaScript is a standard but each browser has its own little differences in the way that they have implemented JavaScript. This is also true in case of headless browsers also. For example HtmlUnit headless browser uses the Rihno JavaScript engine which not being used by any other browser.

Selenium support for headless browser
Selenium supports headless testing using its class called HtmlUnitDriver. This class internally uses HtmlUnit headless browser. HtmlUnit is a pure Java implementation. HtmlUnitWebDriver can be created as mentioned below:

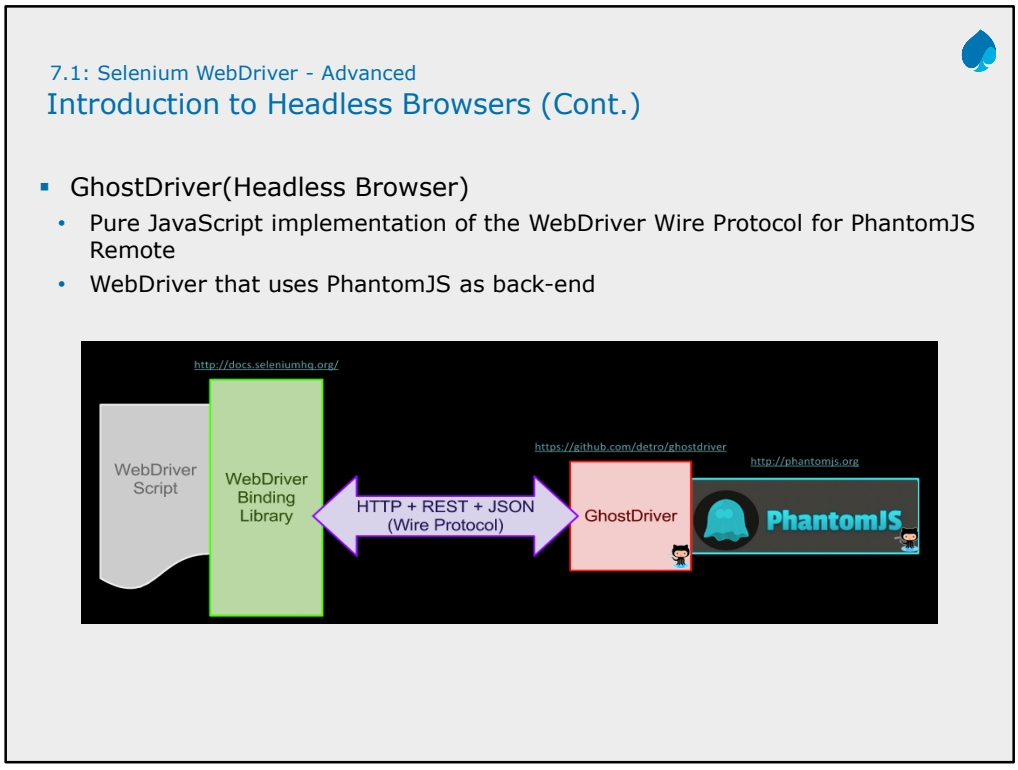
```
HtmlUnitDriver unitDriver = new HtmlUnitDriver();
```

7.1: Selenium WebDriver - Advanced

Introduction to Headless Browsers (Cont.)

- PhanthomJS (Headless Browser)
  - Headless website testing
- Headless Web Kit with JavaScript API
  - Screen capture
- Programmatically capture web contents, including SVG and Canvas
  - Page automation
- Access and manipulate webpages with the standard DOM API, or with usual libraries like jQuery

```
page.evaluate(function()
{
  //Fill in form on page
  document.getElementById('Name').value = 'John Doe';
  document.getElementById('Email').value = 'john.doe@john.doe';
  //Submit
  $('#SubmitButton').click();
}
```



GhostDriver is the project that provides the code that exposes the WebDriver API for use within PhantomJS.
PhantomJS bakes the GhostDriver code into itself, and ships it as part of its downloadable executable.
Thus, to use "GhostDriver" with PhantomJS, only PhantomJS is needed.

Demo – Test Execution in Headless Browser

- Running tests in headless browsers:
  - HTMLUnitDriver
  - PhantomJS

7.1: Selenium WebDriver - Advanced

Other Important Browsers

- Mobile Browsers:
  - Mobile web browsers differ greatly in terms of features offered an operating systems supported
  - Best can display most websites and offer page zoom and keyboard shortcuts, while others can only display websites optimizes for mobile devices
  - Appium and Selendroid are the two cross browser mobile automation tool
- Appium(Mobile Browser)
  - Open-source tool for automating native, mobile web, and hybrid applications on iOS and Android platforms, which is handled by a Appium node.js server.
  - Cross-platform which allows to write tests against multiple platforms (iOS, Android), using the same API
  - Enables code reuse between iOS and Android test suites

Native apps are those written using the iOS or Android SDKs.
Mobile web apps are web apps accessed using a mobile browser (Appium supports Safari on iOS and Chrome or the built-in 'Browser' app on Android). Hybrid apps have a wrapper around a "webview" -- a native control that enables interaction with web content. Projects like Phonegap, make it easy to build apps using web technologies that are then bundled into a native wrapper, creating a hybrid app.

Appium was designed to meet mobile automation needs according to a philosophy outlined by the following four tenets:

You shouldn't have to recompile your app or modify it in any way in order to automate it.
You shouldn't be locked into a specific language or framework to write and run your tests.
A mobile automation framework shouldn't reinvent the wheel when it comes to automation APIs.
A mobile automation framework should be open source, in spirit and practice as well as in name.

7.1: Selenium WebDriver - Advanced

Other Important Browsers (Cont.)

- Selendroid:
  - Test automation framework which is used Android native & hybrid applications (apps) and mobile web
- Features:
  - Full compatibility with the JSON Wire Protocol
  - No modification of app under test required in order to automate it
  - Testing the mobile web using built in Android driver webview app
  - UI elements can be found by different locator types
  - Selendroid can interact with multiple Android devices (emulators or hardware devices) at the same time
  - Existing emulators are started automatically
  - Supports hot plugging of hardware devices
  - Full integration as a node into Selenium Grid for scaling and parallel testing

7.2: Working with Selenium Grid

Introduction to Selenium Grid

- Selenium Grid allows you to create a network of connected test machine, also called as nodes
- Each nodes is basically a computer or even a virtual machine with different browsers & operating system combinations
- This network of test machines is controlled by a Hub, using which you can run your tests on different connected nodes
- Using Selenium Grid you can run tests on a variety of Operating System and Browser combinations

Introduction to Selenium Grid

In the current market scenario, technology is going though rapid changes as far as the host of software used by the customers are concerned. They are either getting updated frequently or even new software are getting introduced to the world. For better user experience, it becomes necessary to validate the application with the different combinations of browsers and also different versions of same browser.

Surely, we can have the browsers installed on our box, but then we cannot install more than one version of a browser on a single system. Also, it is not very easy to implement testing of an application under different operating systems & browsers combination as it is definitely a time consuming activity.

Selenium Grid enables us to create a network of test machines with varying combinations of Operating system and browsers. Using Selenium Grid you can run tests on a variety of Operating System and Browser combinations.

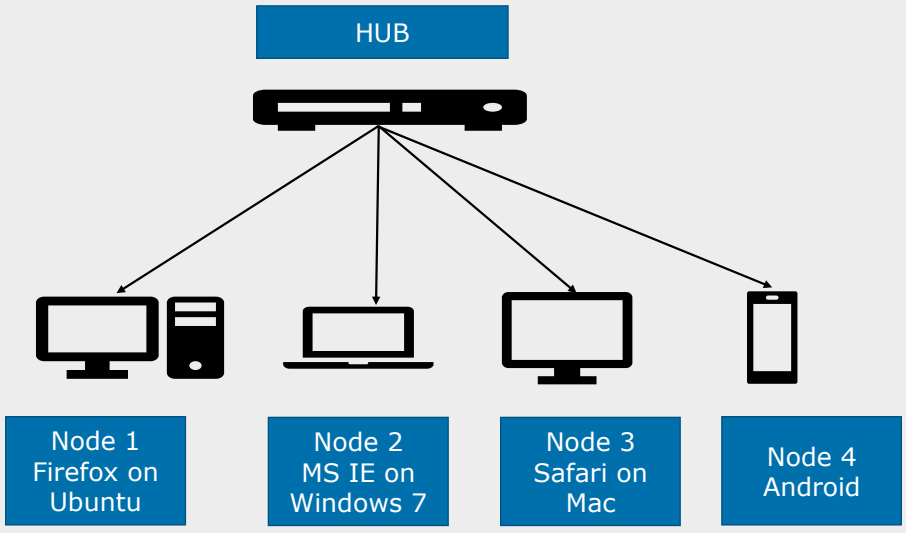
7.2: Working with Selenium Grid

What is Selenium Grid?

- Selenium Grid is a testing tool which allows us to run our tests on different machines against different browsers
- It is a part of the Selenium Suite which specializes in running multiple tests across different browsers, operating system and machines
- It dramatically accelerates the testing process across browsers and across platforms by giving us quick and accurate feedback
- Selenium Grid uses a hub-node concept where you only run the test on a single machine called a hub, but the execution will be done by different machines called nodes
- The selenium-server-standalone package includes Hub, WebDriver, and Selenium RC to execute the scripts in grid
- Selenium Grid has two versions – the older Grid 1 and the newer Grid 2
- We will only focus on Grid 2 because Grid 1 is gradually being deprecated

7.2: Working with Selenium Grid

Selenium Grid Architecture – The Hub & The Node



7.2: Working with Selenium Grid

Selenium Grid Architecture – The Hub & The Node (Cont.)

- Hub
  - The hub is a computer which is the central point where we can load our tests into
  - In Selenium Grid setup, there can be only one hub and that will be the central point in that grid setup
  - We can configure any host machine to be our hub, and it will co-ordinate the activities of test execution
  - When a test with given DesiredCapabilities is given to hub, the hub searches for the node which matches the given configuration
  - Hub will try to find a machine in the grid which matches the criterion and will run the test on that machine

7.2: Working with Selenium Grid

Selenium Grid Architecture – The Hub & The Node (Cont.)

- Node
  - Nodes are the host machines on which the tests would be executed
  - These test machine will be used by hub to run tests on
  - Hub can launch one or more nodes either on remote machines or the same machine where the hub itself is located
  - Selenium Grid network can have multiple nodes
  - A node is supposed to have different platforms i.e. different operating system and browsers
  - The node does not need the same platform for running as that of hub

7.2: Working with Selenium Grid

Selenium Grid Architecture – Configuring Hub and Node

- You need to configure the hub and the node, you need to first of all download the Selenium Server JAR file from Seleniumhq's website
- Configuring Hub

```
Command to configure the hub:
java -jar selenium-server-standalone-3.4.0.jar -role hub
```

- Explanation of the command:
  1. The selenium-server-standalone-3.4.0.jar is the name of the jar file
  2. The -role flag is used to set that particular host machine as the Hub

7.2: Working with Selenium Grid

Selenium Grid Architecture – Configuring Hub and Node (Cont.)

- Configuring Node

Command to configure the node:

```
java -Dwebdriver.chrome.driver=C:\chromedriver.exe -jar selenium-server-standalone-3.4.0.jar -role node -hub http://192.168.0.11:4444/grid/register
```

- Explanation of the command:
  1. -Dwebdriver.chrome.driver=C:\chromedriver.exe is used to set the path of the browser driver
  2. -role node flag is used to set that particular host machine as the Node
  3. -hub http://192.168.0.11:4444/grid/register informs the Node to connect to Hub by using the Hub's IP address
  4. -hub http://localhost:4444/grid/register - In case, you need to launch Node on the same machine as your Hub (Optional)

7.2: Working with Selenium Grid

Selenium Grid Architecture – RemoteWebDriver

- Implementation class of the WebDriver interface that a test script developer can use to execute their test scripts via the RemoteWebDriver server on a remote machine
- Needs to be configured so that it can run your tests on a separate machine
- If driver is not Remote WebDriver, communication to the web browser is local

Example:

```
WebDriver driver = new RemoteWebDriver(new
URL("http://localhost:4444/wd/hub"),  DesiredCapabilities.firefox());
```

This example assumes that Selenium Server is running on localhost with the default port of 4444. The nice thing about this is you can run Selenium Server on any machine, change the URL to point to the new machine and run the tests. For example, you can run the test code from your Mac OS X computer but you can run Selenium Server on a Window XP machine. This way you can launch Internet Explorer tests from my Mac OS X computer.

7.2: Working with Selenium Grid

DesiredCapabilities and Profile Setting in Selenium WebDriver

- During automation, we may need to work on a particular session of browser or work with a browser having some specific configurable properties set or unset
- Selenium WebDriver provides certain browser specific properties known as Desired Capabilities
- Desired Capabilities describes a series of key/value pairs that encapsulate aspects of a browser
- DesiredCapabilities class provides a setCapabilityMethod() to set the different capabilities in a browser

Example to use Firefox profile with desired capabilities:

```
DesiredCapabilities dc=DesiredCapabilities.firefox();
FirefoxProfile profile = new FirefoxProfile();
dc.setCapability(FirefoxDriver.PROFILE, profile);
Webdriver driver =  new FirefoxDriver(dc);
```

Demo – Implementing Selenium Grid

- Configuring Hub
- Configuring Node
- Running tests in Selenium Grid Environment

Summary

- In this lesson, you have learnt
  - How the selenium works and interacts with client server
  - Different drivers that are available for selenium-Chrome, IE, Firefox, headless browsers and mobile browsers
  - In Remote WebDriver the server will always run on the machine with the browser you want to test
  - And, there are two ways to user the server: command line or configured in code
  - Capabilities gives facility to set the properties of browser, such as to set BrowserName, Platform, Version of Browser
  - There are two reasons why you might want to use Selenium-Grid
  - To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems
  - To reduce the time it takes for the test suite to complete a test pass

Add the notes here.

Review Question

Question 1:

- PhanthomJS is
- Chrome Driver
- Mobile Browser
- Headless Browser
- Firefox Driver

Question 2: True/False

- Firefox saves your information such as cookies and browser history in a file called your profile.

Question 3: Fill in the Blanks

- Client driver will send the program that is written in eclipse IDE as _____ and send it to Selenium server