**Selenium Interview Question**

1. What is selenium?

Ans - Selenium is a UI automation tool which will help us to automate the GUI. It will not support desktop application.

1. Does Selenium support automation of pages make in PHP and HTML?

Ans- It is independent of language, it can automate any pages made in any language.

1. Does selenium support automation of application made in only java?

Ans- No, it can only automate web applications.

1. Does selenium have execution engine?

Ans- No. It means selenium can not execute its own script by itself. Because for the execution of selenium scripts, we need to rely on 3rd party tool like TestNG or JUnit sometime Java main() method. We usually don’t prefer main() method for their execution as we cannot avail many testing feature via it and also will not get report for their execution.

1. Does selenium has excel reading methods?

Ans- No, Selenium API doesn’t have any such methods. If you want to read data from external source like property files or excel file, you need to use apache-poi or JExcel (apache poi is easy to use than JExcel) for excel and properties class for property file.

1. What are components of selenium?

Ans- It mainly has 4 components- Selenium IDE, Selenium RC, Selenium Grid and Selenium WebDriver.

1. Selenium IDE is a simple record and playback kind of tool which comes as an add-on for Mozilla Firefox and Chrome. It is used for prototype testing. Test cases written in IDE can be exported in many programming languages like Ruby, Java, C#, etc. Edit and Debug options along with record are also available. It is an excellent tool for beginners to understand the syntax of Selenium WebDriver. Selenium IDE records multiple locators for each element it interacts with. If one locator fails during playback, the others will be tried until one is successful. Selenium IDE requires no additional setup other than installing the extension on your browser.
2. Selenium RC: Selenium RC (Remote Control) was the first tool of Selenium Suite. Earlier it was known as JavaScript Executor. RC was the tool which made Selenium famous in the market.

It was the first tool which provided the support for multiple programming languages (JAVA, Ruby, Perl, PHP, Python, and C#).

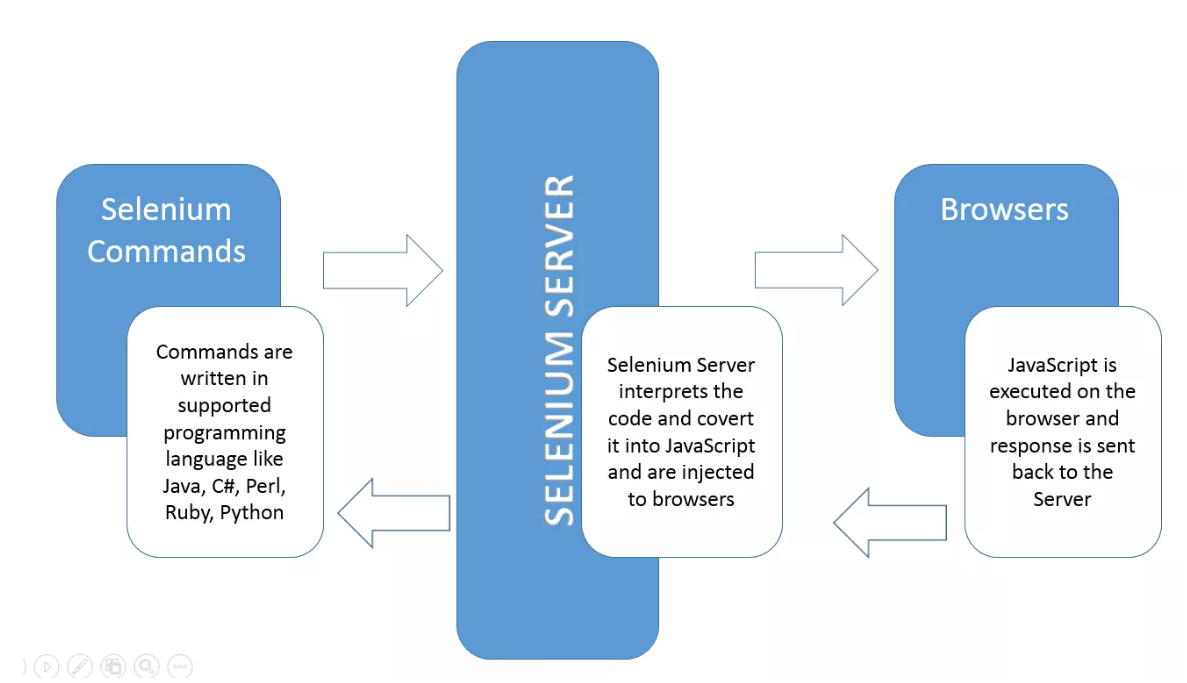
It also supported almost all the major vendors of Browsers like Mozilla Firefox, Google Chrome, Internet Explorer. All the browsers which support JavaScript can be automated using this tool.

Selenium RC is also known as Selenium 1.

The architecture of Selenium RC:

In Selenium RC, there is a manual process called **Selenium Server** is mandatory to start before execution,which acts as a middleman between the code and the browser.

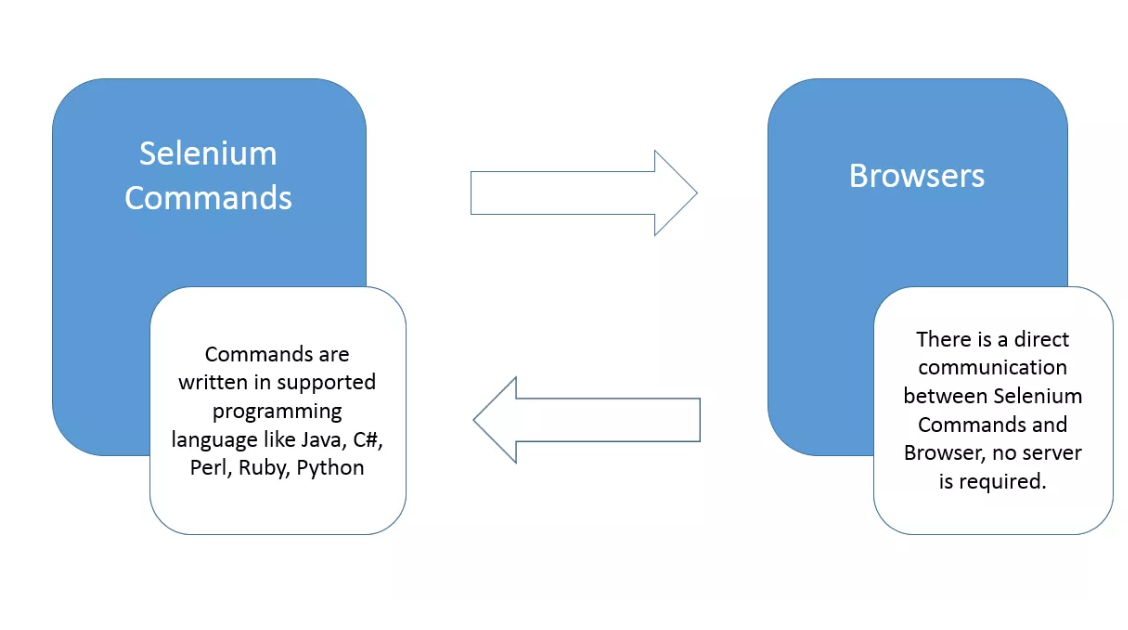
The commands (API’s) are sent to Server. It interprets the command and converts it into JavaScript and then JavaScript is injected to the browser. Now the browser executes the javascript and responds to a server, which again interprets the command and returns to code in the respective language.



Selenium RC is deprecated now.

1. Selenium WebDriver: It is very powerful. Because of many limitations with RC, WebDriver was developed. It does not require any manual process like Selenium Server. There is a direct communication between code and browser.

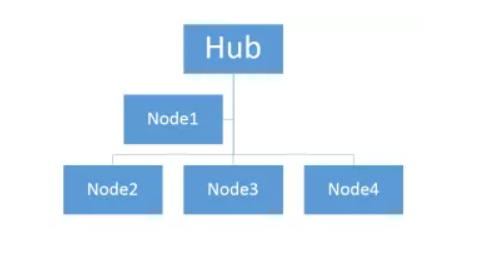
* Features of Selenium WebDriver:
* Open source
* Supports all the key vendors of the browser like Mozilla Firefox, Internet Explorer, Google Chrome, Safari, etc.
* Support Multiple languages like C#, JAVA, Ruby, Perl, Python, and PHP.
* Supports multiple platforms like Linux, Windows, MAC, etc.
* No middleman like Selenium RC server is required.
* Easy to remember API’s.
* Easy to integrate with Testing frameworks.
* Framework Development.
* Parallel Testing capabilities.



1. Selenium Grid: It is the last component of the selenium suite and is used for parallel testing or distributive testing. It allows us to execute test scripts on different machines at same time on different browser from your machine. There is a Hub which controls the execution on various machines, and there are multiple nodes on which actual implementation is done.

It is the last component of the selenium suite and is used for parallel testing or distributive testing. It allows us to execute test scripts on different machines at same time. There is a Hub which controls the execution on various machines, and there are multiple nodes on which actual implementation is done. This is master- slave relationship.

The Architecture of Selenium Grid: In Grid one of the systems is created as Hub. Hub works as a central point controlling all the nodes. Nodes are an actual machine on which execution is done.



1. What are the limitations of selenium?

Ans:

a) It supports testing of only web-based applications.

b) We cannot directly test mobile applications using it, we need to use 3rd party tool like Appium integration with selenium. Appium is “cross-platform”: It facilitates cross platform testing for both iOS and Android using the same API. Appium is open source. You don’t need source code of app to automate it. It is server written in NodeJS. Through Appium you can test automatically your mobile application either on emulator or on real device. [Appium](http://appium.io/) server reads the command coming in from test Java code and executes that command on real device as well as on emulator.

Basic configuration for it-

Install at least JAVA 8, Microsoft .Net framework 4.5, Android SDK (API level 17 or higher), Developer options enabled and Install Appium on your machine.

Jar needed-

Selenium java, Appium Java client and Gson jar.

1. We created a new Java project in Eclipse and added/build above mentioned JAR files in your project.
2. Add the Android apk file path in our code.
3. Set the above-mentioned device capabilities.
4. Initialize WebDriver instance with AndroidDriver as the path for Appium server through which they can connect to each other.
5. Now find elements and perform action on those elements. You can find an element’s id by launching the uiautomationviewer tool available in Android SDK tools folder. You need to take a screenshot of the device and hover mouse on that particular element.
6. Quit your driver.

File app= **new** File("apk-file-path");

DesiredCapabilities capabilities= **new** DesiredCapabilities();

capabilities.setCapability("deviceName", "your-device-name");

capabilities.setCapability("platformVersion", "platform-version");

capabilities.setCapability("platformName", "platform-name");

capabilities.setCapability("app", app.getAbsolutePath());

Webdriver driver = **new** AndroidDriver(**new** URL("http://127.0.0.1:4723/wd/hub"),capabilities);

driver.findElement(By.id("username-element")).sendKeys("username");

driver.findElement(By.id("password-element")).sendKeys("password");

driver.findElement(By.id("password-element ")).click();

driver.quit();

}

Steps to execute automation script:

1. Launch Appium server.
2. Execute the above java code from Eclipse.
3. Now it will start executing script on real device or on emulator

C) Captcha and Barcode is not supported by selenium. At the time of registration in the application, they have captcha or Barcode which can not be read by selenium.

D) For report generation, we need to use 3rd party tools like TestNG and Junit or extents report.

E) As selenium is a free tool, you get very less support whenever you get any error or exception or configuration issue as there is no ready vendor support though the user can find numerous helping communities.

F) User is expected to possess some prior language experience.

1. What are the different types of locators?

Ans- ID, ClassName, Name, TagName, Linktext, Partial linktext, XPath and CSS selector.

1. What does mean by implicit wait?

Ans- If webdriver cannot find an element in the document object model(DOM), then it will wait for defined amount of time for the element to appear in the DOM. An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0.

Once set, the implicit wait is set for the life of the WebDriver object instance.

It is applicable for the entire page. Driver will wait till 60sec to make sure that all element on page get loaded. If the element is not available within the specified Time an NoSuchElementException will be raised.

***driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS)***

1. What does mean by explicit wait?

Ans- Unlike implicit wait, you can write custom code or conditions for wait before proceeding further in the code. WebDriverWait by default calls the ExpectedCondition every 500 milliseconds until it returns successfully.

**When to use:** If element takes a long time to load. Also, used to check property of an element (presence, clickability. etc).

•There can be instance when a particular element takes more than a minute to load.

•In that case you definitely not like to set a huge time to Implicit wait, as if you do this your browser will going to wait for the same time for every element.

•To avoid that situation you can simply put a separate time on the required element only.

•By following this your browser implicit wait time would be short for every element and it would be large for specific element.

We can wait for particular element on the page to get displayed.

***WebDriverWait wait= new WebDriverWait(driver, 60);***

***Wait.until(ExpectedConditions.elementToBeSelected(driver.findElement(By.xpath(“”))));***

Let’s say you have situation you have page and that has been loaded, there is an 3rd party advertisement which can comes on your page after 5ses, with this explicit wait we can wait for that advt. but can’t be possible with implicit wait.

ExpectedConditions class has following methods-

elementToBeClickable(By locator),

elementToBeSelected(Webelement element),

textToBePresentInElement(By locator, String text)

titleContains(String title)

Note: Explicit wait will not wait for 60 sec, the moment when that advt. appears on that page it will come out from wait time and start executing the script onwards. Same is the case with implicit wait, it will come out once the page gets loaded.

1. What is pooling interval?

Ans- Another kind of WebDriverWait. Where we can say that what is the frequency of pooling. In case of explicit wait, your driver will check for web element for every 500 milliseconds. But in the pooling interval-

***WebDriverWait wait= new WebDriverWait(driver, 60);***

***Wait.pollingEvery(5, TimeUnit.SECONDS)***

***Wait.until(ExpectedConditions.elementToBeSelected(driver.findElement(By.xpath(“”))));***

It will check for webelement for every 5 seconds. Check at 5 sec, 10 sec ,15sec…60sec.

1. What is the fluent wait?

Ans- •Let’s say you have an element which sometime appears in just 1 second and some time it takes minutes to appear.

•In that case it is better to use fluent wait with polling mechanism, as this will try to find element again and again until it finds it or until the final timer runs out.

•It tries to find the web element repeatedly at regular intervals (as specified in polling period) of time until the timeout or till the object gets found. It can define the maximum amount of time to wait for a specific condition and frequency with which to check the condition before throwing an “ElementNotVisibleException” exception.

There is another explicit wait mechanism. It can be achieved in 2 ways-

1. WebDriverWait 2) Fluent wait.

It will find the web element repeatedly at regular intervals of time until the timeout or till the objects get found.

***Wait wait = new FluentWait(WebDriver reference)***

***.withTimeout(Duration.ofSeconds(SECONDS))***

***.pollingEvery(Duration.ofSeconds(SECONDS))***

***.ignoring(Exception.class);***

/below one is deprecated code. Use the above one

***Boolean wait= new FlunetWait<WebDriver>(driver)***

***.withTimout(60, TimeUnit.SECONDS)***

***.pollingEvery(5, TimeUnit.SECONDS)***

***.ignoring(NoSuchElementException.class)***

***.until(ExpectedConditions.elementToBeSelected(driver.findElement(By.xpath(“”))))***

To poll for every 5 seconds upto 60 seconds to find the web element. When ever we poll, we will get exception till the element is not present and we should ignore such exception.

1. Why synchronization in a test is required?

Ans- When tests are run, the application may not always respond with the same speed. For example- it might take s few seconds for progress bar to reach 100percent, a status message to appear, a button to become enabled and a window or pop-up message to open.

You can handle these anticipated timing problems by synchronizing your test to ensure that selenium webdriver will wait until your application is ready before performing a step.

1. Disadvantage of using implicit wait?

Ans- Suppose you need to test app where ”getting the list of hotels” in city take 5 seconds but “getting list of hotels in country” takes 15 seconds. Due to this we will increase our implicit wait time from 5 sec to 15 seconds. Now in future, if due to some technical glitches or some performance coding issues, your wait time for scenario- ”getting the list of hotels” increases from 5 sec to 10-13 sec, then we are not able to catch this bugs via our test scripts. Later on it will be questioned by our QA manager. And if we reduce the wait time, “getting list of hotels in country” will get fail. So this issue can be resolved using explicit wait. Implicit wait define wait time globally. Using explicit wait, we will be maintaining wait time as per the specification given by developer. And if a wait time for any page increase, then it can be easily identified.

However implicit wait will slow down your tests if application responds normally as it will wait for each element appearing in the DOM and increase the overall execution time. It is recommended to avoid or minimize the use of it. Try to handle synchronization issues with an explicit wait which provides more control when compared with implicit wait.

In big framework, combination of implicit and explicit wait is best solution for synchronization issues.

1. How to implement custom wait code?

Ans***- WebElement message= (new WebDriverWait(driver, 5))***

***.until(new ExpectedCondtion<WebElement>(){***

***public WebElement apply(WebDriver d){***

***Return d.findElement(By.id(“page4”)***

***}})***

Here we created a custom conditions, which returns the webelement object once the inner findElement() method locates the element within a specified timeout.

Useful in some scenario like

1. based on events and action performed, the value of element attribute might change at runtime like a disabled textbox gets enabled based on user’s rights.

***Return d.findElement(By.id(“username”)).getAttribute(“readOnly”).contains(“true”);***

***Here ExpectedConditions will wait for Boolean return value based on the attribute value of an element.***

1. Waiting for DOM events. The web application may be using a JavaScript framework such as jQuery for AJAX and content manipulation. For ex- jQuery is used to load the big JSON file from server asynchronously on the page. While jQuery is reading and processing this file,

A test can check its status using ACTIVE attribute. Here custom wait is implemented.

JavaScriptExecutor js= (JavaScriptExecutor) d;

Return (Boolean) js.executeScript(“return jQuery.active== 0”)

1. When we have to use the fluent wait?

Ans- Now in some case, HTML is same for

1. Thread.sleep?

Ans- your sleeping your thread for particular time. Thread.sleep(2000) will pause the execution for complete 2 sec even though the page is ready or will load in 1sec. It is not part of a webdriver, part of java. Advised not to use in the framework.

1. What is page load timeout?

Ans- When you are doing driver.get(), we load the URL. Here we can supply the timeout. It can achieved

•The pageLoadTimeout limits the time that the script allots for a web page to be displayed.

•If the page loads within the time then the script continues.

•If the page does not load within the timeout the script will be stopped by a TimeoutException.

1. What is JavaScript wait?

Ans- *ExpectedConditions<Boolean> javaScriptWait= new ExpectedConditions<Boolean>(){*

*Public Boolean apply(Webdriver driver){*

*Return ((JavaScriptExecutor)driver).executeScript(“return document.readyState”).equals(“complete”);*

*}*

*}*

*Try{*

*System.out.printlin(“Waiting for page to get load..”);*

*WebDriverWait wait= new WebDriverWait(driver,60);*

*Wait.until(javaScriptWait);*

*}Catch(Throwable error){*

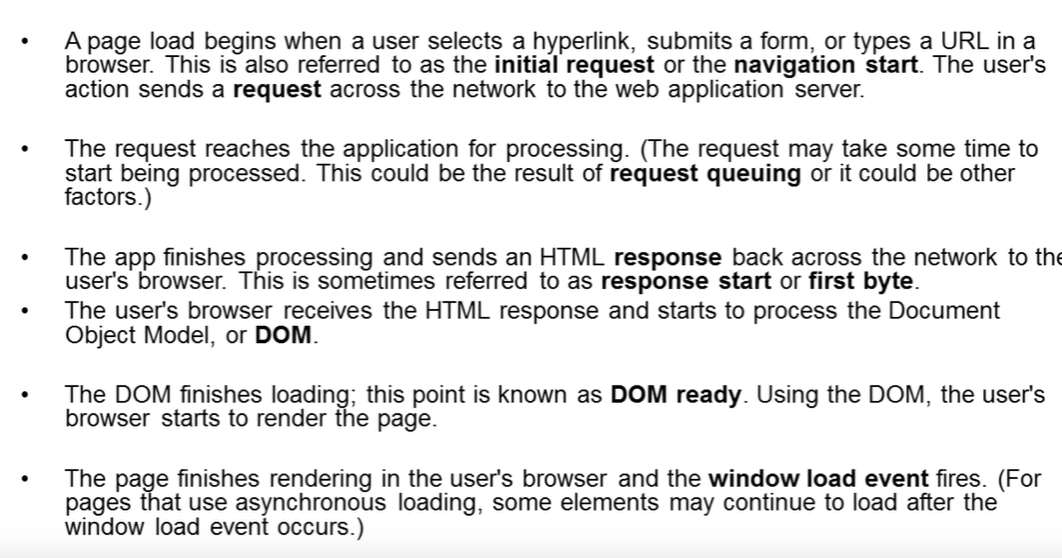
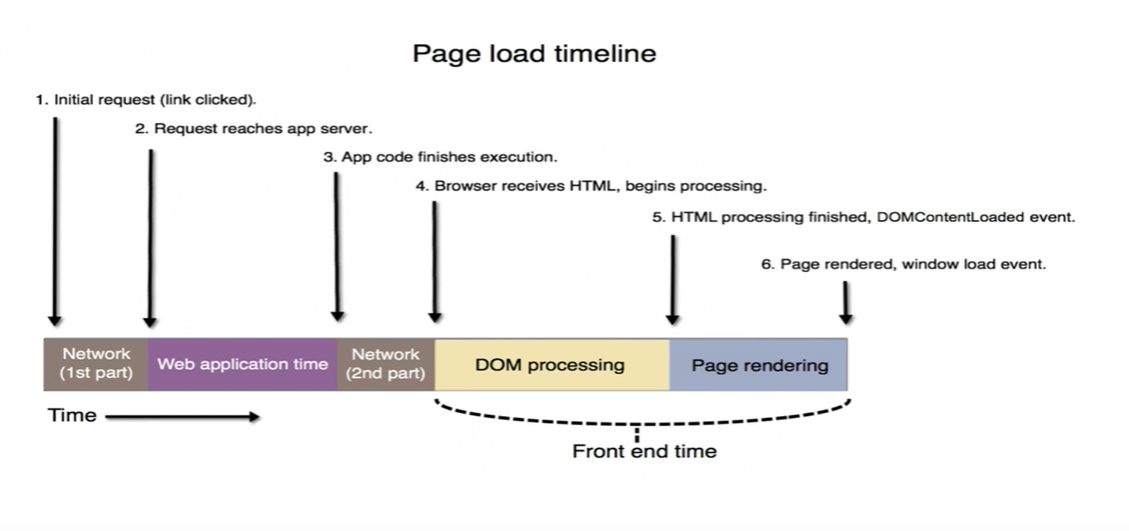
*System.out.printlin(“timeout as page is not loaded after “+60+”seconds”);*

*Asser.assertTrue(true,” timeout as page is not loaded after “+60+”seconds”);*

*}*

1. What is DOM?

Ans- It is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.



When we call a webpage, we get a HTML which is then converted into DOM so that we can call or work with web element.

Now when a Javascript source file is included in a webpage via HTMK <script> tag, there are some script tag which are calling some file. The processing /loading of included file has to be complete first before anything happens on the page. The script is supposed to complete first then any more of such included file is rendered/executed. This we call synchronous loading.

Asynchronous loading-> Sometime we specifically want Javascript source files to load asynchronously. Means while some page is getting rendered, at the same time specified included file in your javascript is also get processed but we will not wait for this script to complete its processing but we will do both the things simultaneously or we may wish that once my page is loaded/rendered then you can run the script later on. This may be handy for files from external sources where we don’t want our including pages to suffer the consequences of: slow connection speed, heavily loaded external servers or waiting for timeouts on external servers that aren’t even there at the moment. Here we want files which are from external sources to get executed inside the JavaScript later on once the page gets rendered then use asynchronously loading.

For these cases, HTML <script> element has ‘async’ and ‘defer’ attributes. Defer requests means loading of specified file occurs after the included page has finished loading. While Async request means that loading of specified file occurs concurrently (or at least independent of) included page.

***Implicit wait: Will wait for the element which are getting loaded asynchronously (i.e. after page has been loaded/rendered)***

***Page Load time: Will wait until complete page gets loaded, not for those elements which are getting loaded asynchronously.***