* what are limitations in selenium webdriver ?

Selenium supports testing of only web based applications  
Mobile applications cannot be tested using Selenium  
Captcha and Bar code readers cannot be tested using Selenium  
Reports can only be generated using third party tools like TestNG or Junit.  
As Selenium is a free tool, thus there is no ready vendor support though the user can find numerous helping communities.  
User is expected to possess prior programming language knowledge.

* Install Selenium/configure
* Step 1: Install Java on your computer. Download and install the Java Software Development Kit (JDK). ...
* Step 2: Install Eclipses IDE...
* Step 3 – Download the Selenium Java Client Driver. ...
* Step 4: Create a New project in Eclipse: ...
* Step 5: Time to Code!!
* what are different ways of locating elements in selenium ?

The different types of locator are:

* ID

driver.findElement(By.id("name"));

* Name

driver.findElement(By.className("even"));

 Link Text

* driver.findElement(By.linkText("Software Testing"));
* CSS Selector

driver.findElement(By.cssSelector("primary-button"))

* DOM (Document Object Model)
* getElementById
* getElementsByName
* dom:name
* dom: index
* Xpath
* **Relative Xpath**
* Relative Xpath begins from the current location and is prefixed with a “//”.
* For example: //span[@class=’Email’]
* **Absolute Xpath**
* Absolute Xpath begins with a root path and is prefixed with a “/”.
* For example: /html/body/div/div[@id=’Email’]
* which is fastest way to identify elements in web page?

**by ID is usually the fastest option**, but here is the list of the best and fastest selectors Selenium WebDriver Tests run faster:

ID selectors (By.ID – Matches by @id attribute)

CSS and Name selectors (Matches by CSS selector or @name attribute)

XPath locators (Matches with arbitrary XPath expression)

* what is absolute path and relative path in xpath?
* **Absolute XPath**:
* It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.
* The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.
* **Relative xpath :**
* For Relative Xpath the path starts from the middle of the HTML DOM structure. Its start with the double forward slash (//), which means it can search the element anywhere at the webpage.
* You can starts from the middle of the HTML DOM structure and no need to write long xpath.
* different types of waits or synchronization in selenium webdriver?
* It is a mechanism which involves more than one components to work parallel with Each other.
* Both these components will have their own speed. We should write our scripts in such a way that both the components should move with same and desired speed, so that we will not encounter "Element Not Found" errors which will consume time again in debugging.
* Synchronization can be classified into two categories:
* **1. Unconditional**  
  **2. Conditional Synchronization**
* **Unconditional :**  
  In this we just specify timeout value only. We will make the tool to wait until certain amount of time and then proceed further.
* **Conditional Synchronization:**
* We specify a condition along with timeout value, so that tool waits to check for the condition and then come out if nothing happens.
* implicit wait, explicit wait and fluent wait?
* **1. Implicit Wait.**
* 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.
* **Explicit Wait:**
* We need to define a wait statement for certain condition to be satisfied until the specified timeout period. If the Webdriver finds the element within the timeout period the code will get executed.
* **Fluent Wait:**
* Using FluentWait we can define the maximum amount of time to wait for a condition, as well as the frequency with which to check for the condition.
* Taking screenshot

|  |
| --- |
| import java.io.IOException;    import org.apache.commons.io.FileUtils;    import org.openqa.selenium.OutputType;    import org.openqa.selenium.TakesScreenshot;    import org.openqa.selenium.WebDriver;    import org.openqa.selenium.firefox.FirefoxDriver;    import org.testng.annotations.Test;    public class ScreenshootGoogle {     @Test   public void TestJavaS1()  {  // Open Firefox  WebDriver driver=new FirefoxDriver();    // Maximize the window  driver.manage().window().maximize();    // Pass the url  driver.get("http://www.google.com");    // Take screenshot and store as a file format  File src= ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);  try {   // now copy the  screenshot to desired location using copyFile //method  FileUtils.copyFile(src, new File("C:/selenium/error.png"));  }    catch (IOException e)   {    System.out.println(e.getMessage());     }   }   * how to handle multiple windows in selenium webdriver    Selenium WebDriver assigns an alphanumeric id to each window as soon as the WebDriver object is instantiated. This unique alphanumeric id is called**window handle**. Selenium uses this unique **id** to switch control among several windows. In simple terms, each unique window has a unique ID, so that Selenium can differentiate when it is switching controls from one window to the other. |

* what is desired capabilities in selenium webdriver

The desired capability is a series of key/value pairs that stores the browser properties like browsername, browser version, the path of the browser driver in the system, etc. to determine the behaviour of the browser at run time.

1. **getBrowserName()**

public java.lang.String getBrowserName()

1. **setBrowserName()**

public void setBrowserName(java.lang.String browserName)

1. **getVersion()**

public java.lang.String getVersion()

1. **setVersion()**

public void setVersion(java.lang.String version)

1. **getPlatform()**

public Platform getPlatform()

1. **setPlatform()**

public Platform getPlatform()

1. **getCapability Method**

The getCapability method of the Desired Capabilities class can be used to get the capability that is in use currently in the system.

public java.lang.Object getCapability(java.lang.String capabilityName)

1. **setCapabilityMethod**

The setCapability() method of the Desired Capabilities class can be used to set the device name, platform version, platform name, absolute path of the app under test (the .apk file of the app(Android) under test), app Activity (in Android) and appPackage(java).

* difference between assert and verify?
* When an “**assert**” command fails, the test execution will be aborted. So when the Assertion fails, all the test steps after that line of code are skipped. The solution to overcoming this issue is to use a try-catch block. We use the Assertion in the try catch block. Mostly, the assert command is used when the end result of the check value should pass to continue to the next step.
* In simple words, if the assert condition is true then the program control will execute the next test step but if the condition is false, the execution will stop and further test step will not be executed.
* **Verify command in selenium:**
* When a “verify” command fails, the test will continue executing and logging the failure. Mostly, the Verify command is used to check non-critical things. In such cases where we move forward even though the end result of the check value is failed.
* In simple words, there wont be any halt in the test execution even though the verify condition is true or false.
* difference between driver.close and driver.quit methods?
* driver.close – It closes the the browser window on which the focus is set.
* driver.quit – It basically calls driver.dispose method which in turn closes all the browser windows and ends the WebDriver session gracefully.

You should use driver.quit whenever you want to end the program. It will close all opened browser window and terminates the WebDriver session. If you do not use driver.quit at the end of program, WebDriver session will not close properly and files would not be cleared off memory. This may result in memory leak errors.

* writing code with expectedwait conditions

|  |
| --- |
| import static org.junit.Assert.\*; |

|  |  |
| --- | --- |
| 2 | import java.util.concurrent.TimeUnit; |

|  |  |
| --- | --- |
| 3 | import org.junit.After; |

|  |  |
| --- | --- |
| 4 | import org.junit.Before; |

|  |  |
| --- | --- |
| 5 | import org.junit.Test; |

|  |  |
| --- | --- |
| 6 | import org.openqa.selenium.By; |

|  |  |
| --- | --- |
| 7 | import org.openqa.selenium.WebDriver; |

|  |  |
| --- | --- |
| 8 | import org.openqa.selenium.WebElement; |

|  |  |
| --- | --- |
| 9 | import org.openqa.selenium.firefox.FirefoxDriver; |

|  |  |
| --- | --- |
| 10 | import org.openqa.selenium.support.ui.ExpectedConditions; |

|  |  |
| --- | --- |
| 11 | import org.openqa.selenium.support.ui.WebDriverWait; |

|  |  |
| --- | --- |
| 12 |  |

|  |  |
| --- | --- |
| 13 | public class Wait\_Demonstration { |

|  |  |
| --- | --- |
| 14 |  |

|  |  |
| --- | --- |
| 15 | // created reference variable for WebDriver |

|  |  |
| --- | --- |
| 16 | WebDriver drv; |

|  |  |
| --- | --- |
| 17 | @Before |

|  |  |
| --- | --- |
| 18 | public void setup() throws InterruptedException { |

|  |  |
| --- | --- |
| 19 |  |

|  |  |
| --- | --- |
| 20 | // initializing drv variable using FirefoxDriver |

|  |  |
| --- | --- |
| 21 | drv=new FirefoxDriver(); |

|  |  |
| --- | --- |
| 22 | // launching gmail.com on the browser |

|  |  |
| --- | --- |
| 23 | drv.get("https://gmail.com"); |

|  |  |
| --- | --- |
| 24 | // maximized the browser window |

|  |  |
| --- | --- |
| 25 | drv.manage().window().maximize(); |

|  |  |
| --- | --- |
| 26 | drv.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); |

|  |  |
| --- | --- |
| 27 | } |

|  |  |
| --- | --- |
| 28 |  |

|  |  |
| --- | --- |
| 29 | @Test |

|  |  |
| --- | --- |
| 30 | public void test() throws InterruptedException { |

|  |  |
| --- | --- |
| 31 |  |

|  |  |
| --- | --- |
| 32 | // saving the GUI element reference into a "username" variable of WebElement type |

|  |  |
| --- | --- |
| 33 | WebElement username = drv.findElement(By.id("Email")); |

|  |  |
| --- | --- |
| 34 |  |

|  |  |
| --- | --- |
| 35 | // entering username |

|  |  |
| --- | --- |
| 36 | username.sendKeys("shruti.shrivastava.in"); |

|  |  |
| --- | --- |
| 37 |  |

|  |  |
| --- | --- |
| 38 | // entering password |

|  |  |
| --- | --- |
| 39 | drv.findElement(By.id("Passwd")).sendKeys("password"); |

|  |  |
| --- | --- |
| 40 |  |

|  |  |
| --- | --- |
| 41 | // clicking signin button |

|  |  |
| --- | --- |
| 42 | drv.findElement(By.id("signIn")).click(); |

|  |  |
| --- | --- |
| 43 |  |

|  |  |
| --- | --- |
| 44 | // explicit wait - to wait for the compose button to be click-able |

|  |  |
| --- | --- |
| 45 | WebDriverWait wait = newWebDriverWait(drv,30); |

|  |  |
| --- | --- |
| 46 |  |

|  |  |
| --- | --- |
| 47 | wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[contains(text(),'COMPOSE')]"))); |

|  |  |
| --- | --- |
| 48 | // click on the compose button as soon as the "compose" button is visible |

|  |  |
| --- | --- |
| 49 | drv.findElement(By.xpath("//div[contains(text(),'COMPOSE')]")).click(); |

|  |  |
| --- | --- |
| 50 | } |

|  |  |
| --- | --- |
| 51 |  |

|  |  |
| --- | --- |
| 52 | @After |

|  |  |
| --- | --- |
| 53 | public void teardown() { |

|  |  |
| --- | --- |
| 54 | // closes all the browser windows opened by web driver |

|  |  |
| --- | --- |
| 55 | drv.quit(); |

|  |  |
| --- | --- |
| 56 | } |

|  |  |
| --- | --- |
| 57 | } |

Common exceptions in selenium ?

* **NoSuchElement** : An element could not be located on the page using the given search parameters.
* **NoSuchFrame** : A request to switch to a frame could not be satisfied because the frame could not be found.
* **StaleElementReference** : An element command failed because the referenced element is no longer attached to the DOM.
* **Firefox Not Connected** : Firefox browser upgraded toop new version.
* **ElementIsNotSelectable** : An attempt was made to select an element that cannot be selected.
* **UnknownCommand** : The requested resource could not be found, or a request was received using an HTTP method that is not supported by the mapped resource.
* **ElementNotVisible** : An element command could not be completed because the element is not visible on the page.
* **InvalidElementState** : An element command could not be completed because the element is in an invalid state (e.g. attempting to click a disabled element).
* **UnknownError** : An unknown server-side error occurred while processing the command.
* **JavaScriptError** : An error occurred while executing JavaScript code.
* **XPathLookupError** : An error occurred while searching for an element by XPath.
* **Timeout** : An operation did not complete before its timeout expired.
* **NoSuchWindow** : A request to switch to a different window could not be satisfied because the window could not be found.
* **InvalidCookieDomain** : An illegal attempt was made to set a cookie under a different domain than the current page.
* **UnableToSetCookie** : A request to set a cookie’s value could not be satisfied.
* **UnexpectedAlertOpen** : A modal dialog was open, blocking this operation
* **NoAlertOpenError** : An attempt was made to operate on a modal dialog when one was not open.
* **ScriptTimeout** : A script did not complete before its timeout expired.
* **InvalidElementCoordinates** : The coordinates provided to an interactions operation are invalid.
* **IMENotAvailable** : IME was not available.  
  **IMEEngineActivationFailed** : An IME engine could not be started.
* **InvalidSelector** : Argument was an invalid selector (e.g. XPath/CSS).
* how to handle Ajax calls in selenium?

The **biggest challenge in handling Ajax call is knowing the loading time for the web page.**Since the loading of the web page will last only for a fraction of seconds, it is difficult for the tester to test such application through automation tool. For that, Selenium Webdriver has to use the wait method on this Ajax Call.

So by executing this wait command, selenium will suspend the execution of current test case and wait for the expected or new value. When the new value or field appears, the suspended test cases will get executed by Selenium Webdriver.

Following are the wait methods that Selenium Webdriver can use

1. **Thread.Sleep()**

* Thread.Sleep () is not a wise choice as it suspends the current thread for the specified amount of time.
* In AJAX, you can never be sure about the exact wait time. So, your test will fail if the element won't show up within the wait time. Moreover, it increases the overhead because calling Thread.sleep(t) makes the current thread to be moved from the running queue to the waiting queue.
* After the time 't' reached, the current thread will move from the waiting queue to the ready queue, and then it takes some time to be picked by the CPU and be running.

1. **Implicit Wait()**

* This method tells webdriver to wait if the element is not available immediately, but this wait will be in place for the entire time the browser is open. So any search for the elements on the page could take the time the implicit wait is set for.

1. **Explicit Wait()**

* [Explicit wait](http://www.guru99.com/implicit-explicit-waits-selenium.html) is used to freeze the test execution till the time a particular condition is met or maximum time lapses.

1. **WebdriverWait**

* It can be used for any conditions. This can be achieved with WebDriverWait in combination with ExpectedCondition
* The best way to wait for an element dynamically is checking for the condition every second and continuing to the next command in the script as soon as the condition is met.

But the problem with all these waits is, you have to mention the time out unit. What if the element is still not present within the time? So there is one more wait called Fluent wait.

1. **Fluent Wait**

* This is an implementation of the Wait interface having its timeout and polling interval. Each FluentWait instance determines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.