0.1 Locators

There are 8 locators strategies included in Selenium:

- Identifier.
- Id.
- Name.
- Link.
- DOM.
- XPath.
- CSS.
- UI-element.

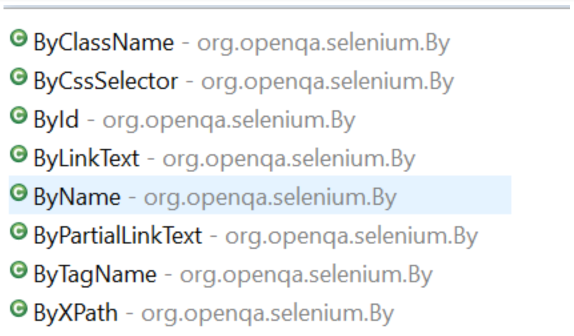
XPath contains the path of the element situated at the web page. Standard syntax for creating XPath is. Hi

```
Xpath=//tagname[@attribute='value']
```

- **// :** Select current node.
- **Tagname:** Tagname of the particular node.
- **@:** Select attribute.
- **Attribute:** Attribute name of the node.
- **Value:** Value of the attribute.

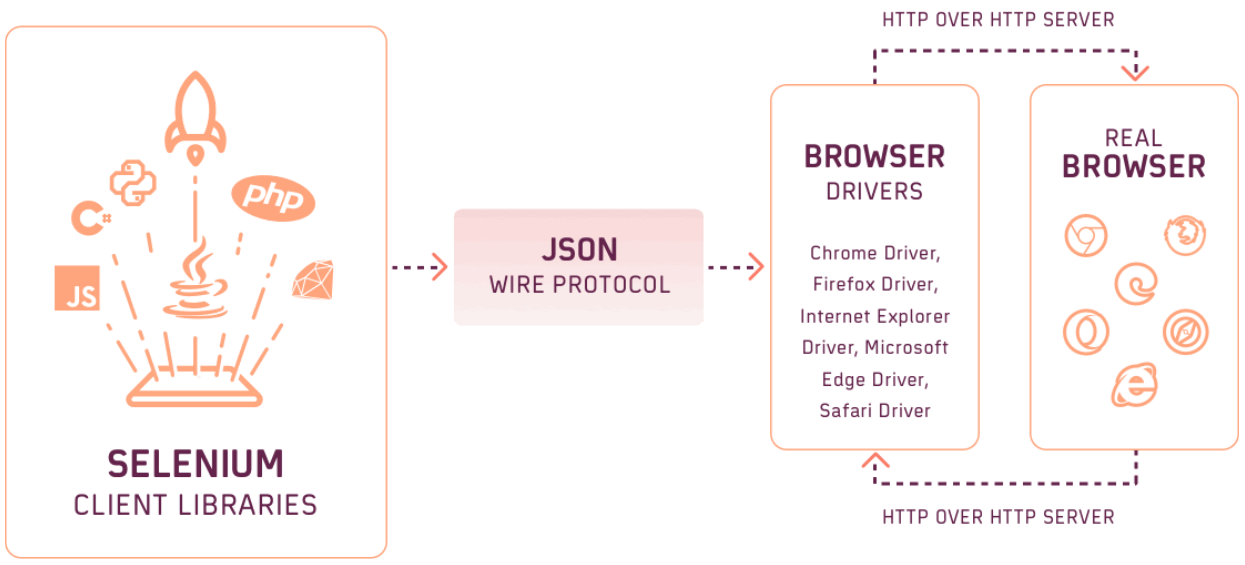
XPath Locators	Find different elements on web page
ID	To find the element by ID of the element
Classname	To find the element by Classname of the element
Name	To find the element by name of the element
Link text	To find the element by text of the link

XPath	XPath required for finding the dynamic element and traverse between various elements of the web page
CSS path	CSS path also locates elements having no name, class or ID.



ByClassName - org.openqa.selenium.By
ByCssSelector - org.openqa.selenium.By
ById - org.openqa.selenium.By
ByLinkText - org.openqa.selenium.By
ByName - org.openqa.selenium.By
ByPartialLinkText - org.openqa.selenium.By
ByTagName - org.openqa.selenium.By
ByXPath - org.openqa.selenium.By

0.2 How selenium works?


HTTP OVER HTTP SERVER
SELENIUM CLIENT LIBRARIES
JSON WIRE PROTOCOL
BROWSER DRIVERS
Chrome Driver, Firefox Driver, Internet Explorer Driver, Microsoft Edge Driver, Safari Driver
REAL BROWSER
HTTP OVER HTTP SERVER

Selenium WebDriver comprises of 4 main components:

1. Selenium Client Libraries
2. JSON Wire Protocol Over HTTP Client
3. Browser Drivers
4. Real Browsers

Selenium is an open-source tool that automates web browsers.

JSON Wire Protocol is responsible for communicating with the browser drivers through their HTTP server. It fetches the information from Selenium Client Libraries and then relays it to the respective Browser Driver.

Each browser has a driver which is responsible for controlling the actions performed within that browser. After JSON Wire Protocol relays information to a Browser Driver, the Browser Driver controls the Browser to execute your Selenium test scripts automatically and sends the response in HTTP protocol through a HTTP server.

- Selenium Script creates an HTTP Request for each selenium command and sends it to the browser driver.
- An HTTP request is then sent to the server using Browser Driver.
- The steps are executed on the HTTP server.
- The execution status is sent to the HTTP server which is then captured by the automation script.

1. Browser property setup

- **Chrome**:

```
System.setProperty("webdriver.chrome.driver",
"/path/to/chromedriver");
```

- **Firefox:**

```
System.setProperty("webdriver.gecko.driver",
"/path/to/geckodriver");
```

- **Edge:**

```
System.setProperty("webdriver.edge.driver",
"P/path/to/MicrosoftWebDriver");
```

2. Browser Initialization

- **Firefox**

```
WebDriver driver = new FirefoxDriver();
```

- **Chrome**
  ```
  WebDriver driver = new ChromeDriver();
  ```
- **Internet Explorer**
  ```
  WebDriver driver = new InternetExplorerDriver();
  ```
- **Safari Driver**
  ```
  WebDriver driver = new SafariDriver();
  ```

3. Desired capabilities

(Doc link)

- **Chrome:**

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("browserName", "chrome");
caps.setCapability("browserVersion", "80.0"");
caps.setCapability("platformName", "win10");WebDriver driver =
new ChromeDriver(caps); // Pass the capabilities as an
argument to the driver object
```

- **Firefox:**

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("browserName", "firefox");
caps.setCapability("browserVersion", "81.0"");
caps.setCapability("platformName", "win10");WebDriver driver =
new FirefoxDriver(caps); // Pass the capabilities as an
argument to the driver object
```

4. Browser options

- **Chrome: (Doc link)**

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setBinary("C:Program Files
(x86)GoogleChromeApplicationchrome.exe"); // if chrome is not
in default location
chromeOptions.addArguments("--headless"); // Passing single
option
chromeOptions.addArguments("--start-maximized",
"--incognito","--disable-notifications" ); // Passing multiple
optionsWebDriver driver = new ChromeDriver(chromeOptions); //
Pass the capabilities as an argument to the driver object
```

- **Firefox: (Doc link)**

```
FirefoxOptions firefoxOptions = new FirefoxOptions();
firefoxOptions.setBinary(new FirefoxBinary(new File("C:Program
FilesMozilla Firefox
irefox.exe")));
firefoxOptions.setHeadless(true);WebDriver driver = new
FirefoxDriver(caps); // Pass the capabilities as an argument
to the driver object
```

Options VS Desired capabilities*:*
There are two ways to specify capabilities.

1. ChromeOptions/FirefoxOptions class — Recommended

2. Or you can specify the capabilities directly as part of the ***DesiredCapabilities*** *— its usage in Java is deprecated*

5. Navigation

- **Navigate to URL — (doc link1 link2)**

```
driver.get("http://google.com")
driver.navigate().to("http://google.com")
```

Myth *— get() method waits till the page is loaded while navigate() does not.*

Referring to the selenium official doc, get() method is a synonym for to() method. Both do the same thing.

Myth *— get() does not store history while navigate() does.*

All the URLs loaded in the browser will be stored in history and the navigate method allows us to access it. Try executing the below code

```
driver.get("http://madhank93.github.io/");
driver.get("https://www.google.com/");
driver.navigate().back();
```

- **Refresh page**

```
driver.navigate().refresh()
```

- **Navigate forwards in the browser history**

```
driver.navigate().forward()
```

- **Navigate backward in the browser history**

```
driver.navigate().back()
```

6. Find element VS Find elements

(doc link) {combine the locators}

- **driver.findElement()**

```
When no match has found(0) throws NoSuchElementException
when 1 match found returns a WebElement instance
when 2+ matches found returns only the first matching web
element
return type is WebElement
```

- **driver.findElements()**

```
when no macth has found (0) returns an empty list
when 1 match found returns a list with one WebElement
when 2+ matches found returns a list with all matching
WebElements
return type is list.
```

7. Locator Strategy

(doc link)

- ***By id***

```
<input id="login" type="text" />


element = driver.findElement(By.id("login"))
```

- ***By Class Name***

```
<input class="Content" type="text" />
```

```
element = driver.findElement(By.className("Content"));
```

- ***By Name***

```
<input name="pswd" type="text" />

element = driver.findElement(By.name("pswd"));
```

- ***By Tag Name***

```
<div id="forgot-password" >…</div>

element = driver.findElement(By.tagName("div"));
```

- ***By Link Text***

```
<a href="#">News</a>

element = driver.findElement(By.linkText("News"));
```

- ***By XPath***

```
<form id="login" action="/action_page.php">
      <input type="text" placeholder="Username"
name="username">
      <input type="text" placeholder="Password" name="psw">
      <button type="submit">Login</button>
</form>

element =
driver.findElement(By.xpath("//input[@placeholder='Username']"
));
```

List of Keywords – and, or, contains(), starts-with(), text(), last()

- ***By CSS Selector***

```
<form id="login" action="submit" method="get">
Username: <input type="text" />
Password: <input type="password" />
</form>
```

```
element =
driver.findElement(By.cssSelector("input.username"));
```

8. Click on an element

- **click()** — method is used to click on an element

```
driver.findElement(By.className("Content")).click();
```

9. Write text inside an element — input and textarea

- **sendKeys()** — method is used to send data

```
driver.findElement(By.className("email")).sendKeys("abc@xyz.co
m");
```

10. Clear text from the text box

- **clear()** — method is used to clear text from the text area

```
driver.findElement(By.xpath("//input[@placeholder='Username']"
)).clear();
```

11. Select a drop-down

(doc link)

```
// single select option
<select id="country">
<option value="US">United States</option>
<option value="CA">Canada</option>
<option value="MX">Mexico</option>
</select>// multiple select option
<select multiple="" id="fruits">
     <option value="banana">Banana</option>
     <option value="apple">Apple</option>
     <option value="orange">Orange</option>
     <option value="grape">Grape</option>
</select>
```

- **selectByVisibleText() / selectByValue() / selectByIndex()**
- **deselectByVisibleText() / deselectByValue() / deselectByIndex()**

```
// import statements for select class
import org.openqa.selenium.support.ui.Select;// Single
selection
Select country = new
Select(driver.findElement(By.id("country")));
country.selectByVisibleText("Canada"); // using
selectByVisibleText() method
country.selectByValue("MX"); //using selectByValue()
method//Selecting Items in a Multiple SELECT elements
Select fruits = new
Select(driver.findElement(By.id("fruits")));
fruits.selectByVisibleText("Banana");
fruits.selectByIndex(1); // using selectByIndex() method
```

12. Get methods in Selenium

- **getTitle()** — used to retrieve the current title of the webpage
- **getCurrentUrl()** — used to retrieve the current URL of the webpage
- **getPageSource()** — used to retrieve the current page source
  of the webpage
- **getText()** — used to retrieve the text of the specified web element
- **getAttribute()** — used to retrieve the value specified in the attribute

13. Handle alerts: (Web-based alert pop-ups)

- **driver.switchTO().alert.getText()** — to retrieve the alert message
- **driver.switchTO().alert.accept()** — to accept the alert box
- **driver.switchTO().alert.dismiss()** — to cancel the alert box
- **driver.switchTO().alert.sendKeys("Text")** — to send data to the alert box

14. Switch frames

- **driver.switchTo.frame(int frameNumber)** — mentioning the frame index number, the Driver will switch to that specific frame
- **driver.switchTo.frame(string frameNameOrID)** — mentioning the frame element or ID, the Driver will switch to that specific frame
- **driver.switchTo.frame(WebElement frameElement)** — mentioning the frame web element, the Driver will switch to that specific frame
- **driver.switchTo().defaultContent()** — Switching back to the main window

15. Handle multiple windows and tabs

- **getWindowHandle()** — used to retrieve the handle of the current page (a unique identifier)
- ***getWindowHandles()*** **—** used to retrieve a set of handles of the all the pages available
- **driver.switchTo().window("windowName/handle")** — switch to a window
- **driver.close()** — closes the current browser window

16. Waits in selenium

There are 3 types of waits in selenium,

- **Implicit Wait —** used to wait for a certain amount of time before throwing an exception

```
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
```

- **Explicit Wait** — used to wait until a certain condition occurs before executing the code.

```
WebDriverWait wait = new WebDriverWait(driver,30);
wait.until(ExpectedConditions.presenceOfElementLocated(By.name
("login")));
```

List of explicit wait:

```
alertIsPresent()
elementSelectionStateToBe()
elementToBeClickable()
elementToBeSelected()
frameToBeAvaliableAndSwitchToIt()
invisibilityOfTheElementLocated()
invisibilityOfElementWithText()
presenceOfAllElementsLocatedBy()
presenceOfElementLocated()
textToBePresentInElement()
textToBePresentInElementLocated()
textToBePresentInElementValue()
titleIs()
titleContains()
```

```
visibilityOf()
visibilityOfAllElements()
visibilityOfAllElementsLocatedBy()
visibilityOfElementLocated()
```

- **Fluent Wait** — defines the maximum amount of time to wait for a certain condition to appear

```
Wait wait = new FluentWait(WebDriver reference)
.withTimeout(Duration.ofSeconds(SECONDS))
.pollingEvery(Duration.ofSeconds(SECONDS))
.ignoring(Exception.class);WebElement foo=wait.until(new
Function<WebDriver, WebElement>() {
public WebElement apply(WebDriver driver) {
return driver.findElement(By.id("foo"));
}
});
```

17. Element validation

- **isEnabled()** — determines if an element is enabled or not, returns a boolean.
- **isSelected()** — determines if an element is selected or not, returns a boolean.
- **isDisplayed()** — determines if an element is displayed or not, returns a boolean.

18. Handling proxy

- **Chrome:**

```
ChromeOptions options = new ChromeOptions();// Create object
Proxy class - Approach 1
Proxy proxy = new Proxy();
proxy.setHttpProxy("username:password.myhttpproxy:3337");//
register the proxy with options class - Approach 1
options.setCapability("proxy", proxy);// Add a
ChromeDriver-specific capability.
ChromeDriver driver = new ChromeDriver(options);
```

- **Firefox:**

```
FirefoxOptions options = new FirefoxOptions();// Create object
Proxy class - Approach 2
Proxy proxy = new Proxy();
proxy.setHttpProxy("myhttpproxy:3337");
proxy.setSocksUsername("username");
proxy.setSocksPassword("password")// register the proxy with
options class - Approach 2
options.setProxy(proxy);// create object to firefx driver
WebDriver driver = new FirefoxDriver(options);
```

19. Window management

- **Get window size:**

```
//Access each dimension individually
int width = driver.manage().window().getSize().getWidth();
int height =
driver.manage().window().getSize().getHeight();//Or store the
dimensions and query them later
Dimension size = driver.manage().window().getSize();
int width1 = size.getWidth();
int height1 = size.getHeight();
```

- **Set window size:**

```
driver.manage().window().setSize(new Dimension(1024, 768));
```

- **Get window position:**

```
// Access each dimension individually
int x = driver.manage().window().getPosition().getX();
int y = driver.manage().window().getPosition().getY();// Or
store the dimensions and query them later
Point position = driver.manage().window().getPosition();
int x1 = position.getX();
int y1 = position.getY();
```

- **Set window position:**

```
// Move the window to the top left of the primary monitor
driver.manage().window().setPosition(new Point(0, 0));
```

- **Maximize window:**

```
driver.manage().window().maximize();
```

- **Fullscreen window:**

```
driver.manage().window().fullscreen();
```

20. Page loading strategy

The `document.readyState` property of a document describes the loading state of the current document. By default, WebDriver will hold off on responding to
a `driver.get()` (or) `driver.navigate().to()` call until the document ready state is `complete`

By default, when Selenium WebDriver loads a page, it follows the normal pageLoadStrategy.

- **normal:**

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setPageLoadStrategy(PageLoadStrategy.NORMAL);
WebDriver driver = new ChromeDriver(chromeOptions);
```

- **eager:** When setting to eager, Selenium WebDriver waits until DOMContentLoaded event fire is returned.

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setPageLoadStrategy(PageLoadStrategy.EAGER);

WebDriver driver = new ChromeDriver(chromeOptions);
```

- **none:** When set to none Selenium WebDriver only waits until the initial page use);

```
   WebDriver driver = new ChromeDriver(chromeOptions);
```

21. Keyboard and Mouse events

Action class is used to handle keyboard and mouse events

keyboard events:(Robot Class)

- keyDown()
- keyUp()

- sendKeys()

Mouse events: (Actions class)

- clickAndHold()
- contextClick() — performs the mouse right-click action
- doubleClick()
- dragAndDrop(source,target)
- dragAndDropBy(source,xOffset,yOffset)
- moveByOffset(xOffset,yOffset)
- moveByElement()
- release()

```
Actions builder = new Actions(driver);
Action actions = builder
 .moveToElement("login-textbox")
 .click()
 .keyDown("login-textbox", Keys.SHIFT)
 .sendKeys("login-textbox", "hello")
 .keyUp("login-textbox", Keys.SHIFT)
 .doubleClick("login-textbox")
 .contextClick()
 .build();

actions.perform() ;
```

Multiple KeyBoard Events:

22. Cookies

- **addCookie(arg)**

```
driver.manage().addCookie(new Cookie("foo", "bar"));
```

- **getCookies()**

```
driver.manage().getCookies(); // to get all cookies
```

- **getCookieNamed()**

```
driver.manage().getCookieNamed("foo");
```

- **deleteCookieNamed()**

```
driver.manage().deleteCookieNamed("foo");
```

- **deleteCookie()**

```
Cookie cookie1 = new Cookie("test2", "cookie2");
driver.manage().addCookie(cookie1);
driver.manage().deleteCookie(cookie1); // deleting cookie
object
```

- **deleteAllCookies()**

```
driver.manage().deleteAllCookies(); // deletes all cookies
```

23. Take screenshot:

(doc link)

- **getScreenshotAs** — used to Capture the screenshot and store it in the specified location. This method throws WebDriverException. copy() method from the File Handler class is used to store the screenshot in a destination folder

```
TakesScreenshot screenShot =(TakesScreenshot)driver;
FileHandler.copy(screenShot.getScreenshotAs(OutputType.FILE),
new File("path/to/destination/folder/screenshot.png"));
```

24. Execute Javascript:

(doc link)

- **executeAsyncScript()** — executes an asynchronous piece of JavaScript
  - JavascriptExecutor js = (JavascriptExecutor) driver;
  - js.executeScript(Script,Arguments);

```
•   // This  will scroll down the page by  1000 pixel vertical
//executeScript("window.scrollBy(x-pixels,y-pixels)");
    js.executeScript("window.scrollBy(0,1000)");
```

- **executeScript()** — executes JavaScript

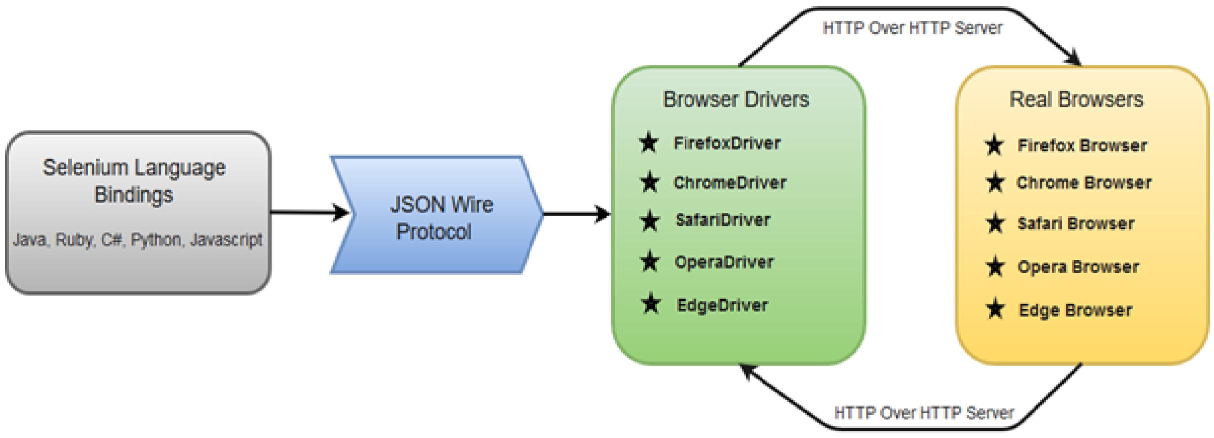
```
if (driver instanceof JavascriptExecutor) {
     ((JavascriptExecutor)driver).executeScript("alert('hello
world');");
```

http://www.uitestpractice.com/

Selenium WebDriver- Architecture

Selenium WebDriver API provides communication facility between languages and browsers.

The following image shows the architectural representation of Selenium WebDriver.



There are four basic components of WebDriver Architecture:

- o Selenium Language Bindings
- o JSON Wire Protocol
- o Browser Drivers
- o Real Browsers

JSON Wire Protocol

JSON (JavaScript Object Notation) is an open standard for exchanging data on web. It supports data structures like object and array. So, it is easy to write and read data from JSON. To learn more about JSON, visit https://www.javatpoint.com/json-tutorial

JSON Wire Protocol provides a transport mechanism to transfer data between a server and a client. JSON Wire Protocol serves as an industry standard for various REST web services. To learn more about Web Services, visit https://www.javatpoint.com/web-services-tutorial

Browser Drivers

Selenium uses drivers, specific to each browser in order to establish a secure connection with the browser without revealing the internal logic of browser's functionality. The browser driver is also specific to the language used for automation such as Java, C#, etc.

When we execute a test script using WebDriver, the following operations are performed internally.

- o HTTP request is generated and sent to the browser driver for each Selenium command.
- o The driver receives the HTTP request through HTTP server.
- o HTTP Server decides all the steps to perform instructions which are executed on browser.
- o Execution status is sent back to HTTP Server which is subsequently sent back to automation script.

https://automationreinvented.blogspot.com/2020/03/how-to-add-repository-in-bitbucket-for.html

Scenario Based Interview Q&A

1. **Handling Dynamic Elements:**

   Question: How would you handle dynamic elements in Selenium WebDriver?

   Explanation: Dynamic elements are those that appear or disappear dynamically on a web page. It's crucial to wait for such elements to be present before interacting with them to avoid NoSuchElementExceptions.

```
WebElement dynamicElement = driver.findElement(By.xpath("xpath_of_dynamic_element"));
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.visibilityOf(dynamicElement));
dynamicElement.click();
```

2. **Handling Pop-up Windows:**

   Question: Describe how you would handle pop-up windows in Selenium.

Explanation: Pop-up windows are secondary browser windows that may appear during a test scenario. We switch to the new window, perform actions, and then switch back to the main window.

```
String mainWindow = driver.getWindowHandle();
Set<String> allWindows = driver.getWindowHandles();
for (String window : allWindows) {
        if (!window.equals(mainWindow)) {
        driver.switchTo().window(window);
        // Perform operations on the pop-up window
        driver.close();
        driver.switchTo().window(mainWindow);
        }
}
```

3. **Handling Dropdowns:**

   Question: How can you handle dropdowns in Selenium WebDriver?

   Explanation: Dropdowns or select elements require special handling. We use the Select class to interact with dropdowns by index, value, or visible text.

```
Select dropdown = new Select(driver.findElement(By.id("dropdown_id")));

dropdown.selectByVisibleText("Option");
```

4. **Handling Frames:**

   Question: Explain how to handle frames in Selenium.

   Explanation: Frames are used to divide a browser window into multiple sections. We need to switch to the frame context to perform operations within it.

```
driver.switchTo().frame("frame_name_or_id");
// Perform operations inside the frame
driver.switchTo().defaultContent(); // Switch back to the main page
```

5. **Performing Mouse Actions:**

   Question: How do you perform mouse actions using Selenium?

   Explanation: Selenium provides the Actions class to simulate mouse actions like hover, click, double click, etc.

```
Actions actions = new Actions(driver);
WebElement element = driver.findElement(By.id("element_id"));
actions.moveToElement(element).perform();
```

6. **Handling File Uploads:**

   Question: How would you handle file uploads in Selenium WebDriver?

   Explanation: File uploads involve interacting with a file input element and providing the file path for uploading.

```
WebElement uploadElement = driver.findElement(By.id("uploadButton"));
uploadElement.sendKeys("/path/to/file.txt");
```

7. **Capturing Screenshots:**

   Question: Explain how to capture screenshots in Selenium.

   Explanation: Selenium can capture screenshots of the browser window, useful for debugging or generating reports.

```
File screenshotFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(screenshotFile, new File("path_to_save_screenshot.png"));
```

8. **Performing Browser Navigation:**

   Question: Describe how to perform browser navigation in Selenium.

   Explanation: Selenium supports browser navigation methods like navigating to a URL, going back, forward, and refreshing the page.

```
driver.navigate().to("https://example.com");
driver.navigate().back();
driver.navigate().forward();
driver.navigate().refresh();
```

9. **Waiting for Element to be Clickable:**

   Question: How can you wait for an element to be clickable in Selenium?

   Explanation: Before interacting with an element, it's essential to ensure it is clickable to avoid ElementClickInterceptedExceptions.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("element_id")));
```

10. **Data-Driven Testing:**

    Question: Explain how to perform data-driven testing using Selenium.

    Explanation: Data-driven testing involves executing the same test scenario with different sets of test data. TestNG's DataProvider annotation helps achieve this.

```
@DataProvider(name = "testdata")
public Object[][] testData() {
        return new Object[][] { { "username1", "password1" }, { "username2", "password2" }};
}

@Test(dataProvider = "testdata")
public void loginTest(String username, String password) {
        driver.findElement(By.id("username")).sendKeys(username);
        driver.findElement(By.id("password")).sendKeys(password);
        driver.findElement(By.id("loginButton")).click();
}
```