CSS selectors

CSS selectors allow you to select elements based on their tag name, id, class, attribute, and other characteristics.

- To select all elements with the tag "p" (paragraph), you could use the following selector: p
- To select an element with the ID "main-heading", you could use the following selector: #main-heading
- To select all elements with the class "error", you could use the following selector: .error
- To select all elements with the attribute "disabled", you could use the following selector: [disabled]
- To select all "a" elements that are descendants of a "nav" element, you could use the following selector: nav a

form#login-form input[type="radio"]

CSS [attribute*=value] Selector

The [attribute*=”str”] selector is used to select those elements whose attribute value contains the specified substring str.

CSS [attribute=value] Selector
The [attribute=value] selector in CSS is used to select those elements whose attribute value is equal to “value”.

CSS [attribute$=value] Selector The [attribute$=”value”] selector is used to select those elements whose attribute value ends with a specified value “value”.

CSS [attribute|=value] Selector This is used to select those elements whose attribute value is equal to “value” or whose attribute value started with “value” immediately followed by hyphen (-).

CSS [attribute~=value] Selector The [attribute~=”value”] selector is used to select those elements whose attribute value contains a specified word.

CSS [attribute^=value] Selector The [attribute^=value] selector is used to select those elements whose attribute value begins with a given attribute.

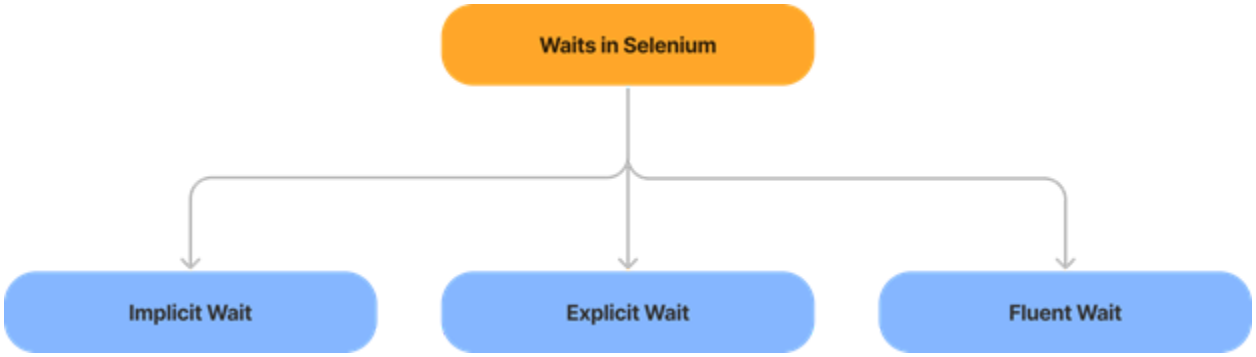
CSS :first-child Selector The :first-child selector is used to select those elements which are the first-child elements.

CSS :last-child Selector The :last-child Selector is used to target the last child element of its parent for styling.

Selenium Waits

Why Do We Need Waits In Selenium?

- Web applications are developed using Ajax and Javascript.
- New JS frameworks are more advanced and use AJax, react, and angular.
- elements which we want to interact with may load at different time intervals.



Implicit Wait

Implicit waits in Selenium are used to provide a general timeout for locating elements on a web page. Unlike explicit waits, which wait for specific conditions to be met, implicit waits set a global timeout for all subsequent find element operations.

Why we use implicit waits:

- **Handling dynamic content:** When dealing with web pages that load content dynamically (e.g., using JavaScript), implicit waits can help avoid `NoSuchElementException` errors by giving the browser additional time to load elements.
- If the element is not located on the web page within that time frame, it will throw an **exception**.
- **Simplifying test code:** By setting a global timeout, you can reduce the need for explicit waits in many cases, making your test code more concise and easier to read.
- WebDriver polls the DOM for a certain duration when trying to find any element.
- **Global timeout:** Sets a general timeout for all subsequent find element operations.
- It tells the web driver to wait for the specified time before moving to the next command.
- Once it is set it is applicable to full automation script.
- Implicit wait is the maximum time between the two commands.
- Different from Thread.sleep - Thread.sleep() - It will sleep time for script.
- Not good way to use it in script as it's sleep without condition.
- Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times.

Syntax for Implicit Wait :

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(20));
```

Pros: Simplifies test code by reducing the need for explicit waits in many cases.
Cons: Can introduce performance overhead if the timeout is set too high. Less granular control over waiting behavior.

However, there are some important considerations:

- **Overuse can lead to performance issues:** If you set an excessively long implicit wait, your tests might take longer to execute, especially for pages that load quickly.
- **Less control:** Compared to explicit waits, implicit waits offer less control over the waiting behavior.

Example :

```
WebDriver driver = new ChromeDriver();
driver.get("https://www.saucedemo.com/");
```

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

//implicit wait
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(20));

String loginPageTitle = driver.getTitle();
Assert.assertEquals(loginPageTitle, "Swag Labs");

driver.findElement(By.id("user-name")).sendKeys("standard_user");
driver.findElement(By.id("password")).sendKeys("secret_sauce");
driver.findElement(By.id("login-button")).click();
```

Explicit Wait

Explicit Wait in Selenium is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing “ElementNotVisibleException” exception

- Little intelligent wait, wait for certain conditions.
- They **allow your code to halt program execution, or freeze the thread, until the condition you pass it resolves.**
- The condition is called with a certain frequency until the timeout of the wait is Elapsed.
- This means that for as long as the condition returns a falsy value, it will keep trying and waiting.
- It provides better way to handle the dynamic Ajax elements
- Element not visible exception if element not found.
- Good fit for synchronizing the state between the browser and its DOM, and your.
- Replace Thread.sleep with explicit wait always

The following are the Expected Conditions that can be used in Selenium Explicit Wait

- alertIsPresent()
- elementSelectionStateToBe()

- **elementToBeClickable()**
- elementToBeSelected()
- frameToBeAvailableAndSwitchToIt()
- invisibilityOfTheElementLocated()
- invisibilityOfElementWithText()
- presenceOfAllElementsLocatedBy() :
- presenceOfElementLocated() : waits till element to be present in DOM
- textToBePresentInElement()
- textToBePresentInElementLocated()
- textToBePresentInElementValue()
- titleIs()
- titleContains()
- visibilityOf()
- visibilityOfAllElements()
- visibilityOfAllElementsLocatedBy()
- visibilityOfElementLocated() : wait till element to be visible

Example Code :

```
WebElement productPageText =
driver.findElement(By.xpath("//span[@class='title']"));

WebDriverWait wait1 = new WebDriverWait(driver,
Duration.ofSeconds(10));

wait1.until(ExpectedConditions.visibilityOf(productPageText));
```

Fluent Wait

Fluent Wait instance defines the maximum amount of time to wait for a condition as well as the frequency with which to check the condition

- Exception - NoSuchElementException
- Waiting 30 seconds for an element to be present on the page, checking for its presence once every 5 seconds.

Example Code :

```
//Declare and initialize a fluent wait
FluentWait<WebDriver> wait = new FluentWait<>(driver);

//Specify the timeout of the wait
wait.withTimeout(Duration.ofSeconds(20));

//Specify polling time
wait.pollingEvery(Duration.ofMillis(250));

//Specify what exceptions to ignore
wait.ignoring(NoSuchElementException.class);

//This is how we specify the condition to wait on.
//This is what we will explore more in this chapter
wait.until(ExpectedConditions.visibilityOf(productPageText));
```

Difference between Implicit and Explicit Wait :

Implicit Wait	Explicit Wait
• Implicit Wait time is applied to all the elements in the script	• Explicit Wait time is applied only to those elements which are intended by us
• In Implicit Wait, we need **not** specify “ExpectedConditions” on the element to be located	• In Explicit Wait, we need to specify “ExpectedConditions” on the element to be located

- It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait
- It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)

Interview Questions on Selenium Waits/Synchronization:

What is Synchronization?
o Synchronization refers to handling the timing issues when there are delays in loading elements or pages, ensuring that Selenium waits for elements to be available before performing actions on them.

How to handle Synchronization wait available in WebDriver?
o Using implicit waits, explicit waits, and fluent waits.

Which wait statement will be used to wait until the page gets loaded?
o You can use WebDriverWait with a condition like wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("s omeId")));.

How to handle dynamic objects?
o Use explicit waits to wait for the object to appear, and use flexible locators like XPath or CSS selectors that can handle dynamic attributes.

Difference between thread wait, implicitly wait, explicitly wait?
o **Thread wait (Thread.sleep()):** Pauses the execution for a specified time.
o **Implicit wait:** Sets a default wait time for the WebDriver to wait for elements to appear.
o **Explicit wait:** Waits for a specific condition to occur before proceeding.

Do you have any idea of a fluent wait? If yes, explain.

o Fluent wait is a type of explicit wait with polling intervals. It repeatedly checks for a condition at specified intervals, ignoring exceptions.

If an element fails to load within implicit wait, what exception will you get?
o NoSuchElementException.

Java Interview Coding Questions :

Print the Output of code snippet :

```
/*
	static int a = 20;
	int b= a + 10;
	System.out.println(b); // Compile Time Error

*/
```

Write a java code to reverse a given String in java without using predefined method :

```
String originalString = "hello world";

char[] charArr = originalString.toCharArray();
int start = 0;
int end = charArr.length-1;

while(start < end){
```

```
   char temp  = charArr[start];
   charArr[start] = charArr[end];
   charArr[end] = temp;
   start ++;
   end --;

}

String reversedString = new String(charArr);
System.out.println("Original String : "+originalString);
System.out.println("Reversed String : "+reversedString);
```

Write a code to eliminate the first word from a given String.

```
String str = "Automation Testing Software Engineer";

int automationIndex = str.indexOf("Automation");
//System.out.println(automationIndex);

String extractedString = str.substring(automationIndex +
"Automation".length());

System.out.println(extractedString.trim());
```

Write a code to Remove the duplicate characters from a given String :

```
String str = "characters"; // o/p = chartes
HashSet<Character> set = new HashSet<>();
StringBuilder result = new StringBuilder();

for(int i=0; i<str.length(); i++){
   char ch = str.charAt(i);
   if(set.add(ch)){
       result.append(ch);
   }
}
System.out.println(result);
```