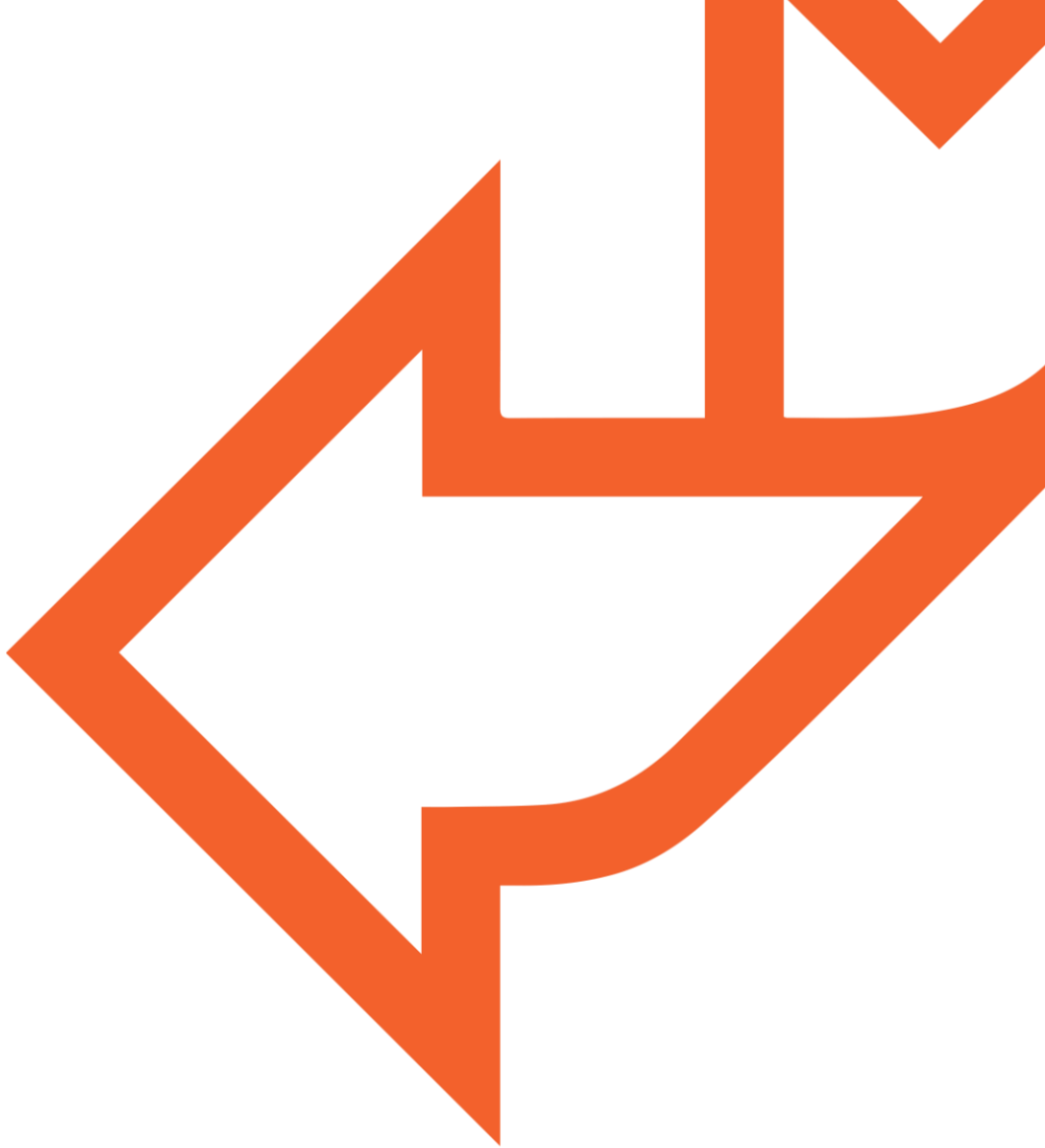
POWERING POTENTIAL

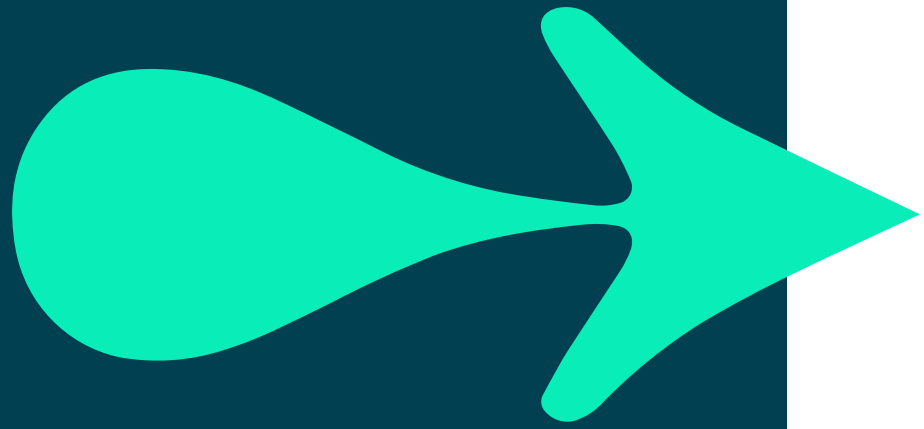


Selenium

Module 3 – Waiting for Elements

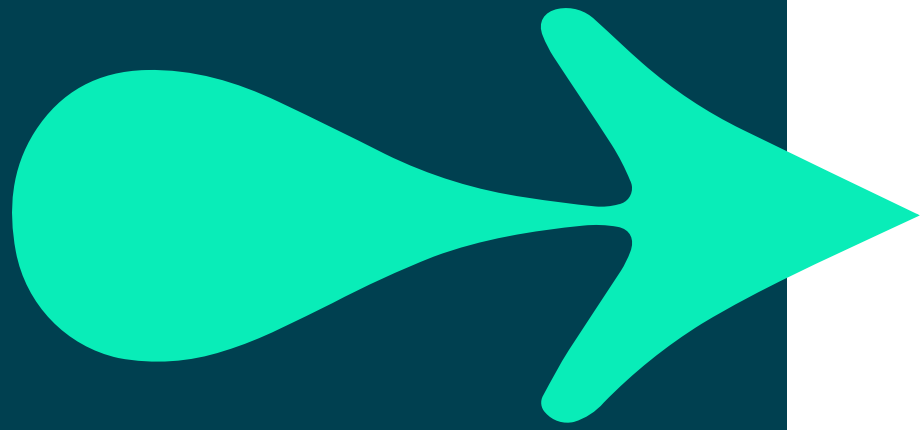
Why wait?

WAITING FOR ELEMENTS



- The WebDriver API is generally blocking in nature, it instructs the browser what to do but doesn't actively track the real-time state of the DOM.
- Web communication is intrinsically asynchronous in nature and would be expensive to constantly track.
- Issues arise from Selenium being a blocking API due to something called race conditions between the browser and the programmatic instructions.
- To combat this, we can use waits within our Selenium code to have automated tasks allow a certain amount of time to elapse before continuing.

TYPES OF WAIT



There are three types of wait:

- Explicit
- Implicit
- Fluent

Waits in Selenium

- If we try to interact with a WebElement in Selenium which isn't on the page, it will return a NoSuchElementException.
- To remedy this, we can tell the Selenium driver to wait until an element appears.
- Having Selenium wait is much more efficient than using something like Thread.sleep(). This is because it will always wait for however many milliseconds you hard-code into it – even if the element you need appears well within time.
- With a wait, Selenium will wait up to 10 seconds, but if the element you need appears before then, it will continue. This saves a huge amount of time in testing, especially if you were originally running multiple tests with Thread.sleep()!

A note on wait times...

- It is important to note that while setting the timings for your waits to the full 10 seconds is more likely to lead to successful tests, you must still bear in mind the short attention-span of your users.

- If a Web site takes a long time to load, users may grow impatient, thus user acceptance testing would fail instead!

- Therefore, while Selenium can add extra benefit by automating whether items will load on a screen in a timely manner, you are still responsible for the timings of your test suites.

Explicit waits

Explicitly waiting for an element

Explicit waits are the most employed type of wait, these are available for imperative languages to halt the program execution or threads until a certain condition has resolved – if the condition returns false, Selenium will wait until some specified duration has elapsed before polling for the element again.

- Useful for synchronising the browser state with Selenium.

- Requires a **WebDriver**, a **Duration** to wait and a function accepting a **WebDriver** instance as input and returning a **WebElement** as output.

```
new WebDriverWait(WebDriver driver, Duration duration)
    .until(new Function<? Super WebDriver, WebElement> isTrue);
```

Explicit wait – Example 1

WebDriver instance

Duration to wait for element

```
WebElement uname = new WebDriverWait(driver, Duration.ofSeconds(5))
    .until(driver -> driver.findElement(By.id("username")));
```

Lambda function which finds the desired element

```
driver -> driver.findElement(By.id("username"))
```

Equivalent

```
new Function<WebDriver, WebElement>() {
    @Override
    public WebElement apply(WebDriver driver) {
      return driver.findElement(By.id("username"));
    }
}
```

Explicit wait – Example 2

```
WebElement uname = new WebDriverWait(driver, Duration.ofSeconds(5))
    .until(ExpectedConditions.visibilityOf(driver.findElement(By.id("username"))));
```

- The **ExpectedConditions** class has helper static methods for common conditions on which to await an element, in this case the code is specifying that we are waiting for the element with an id of *username* to be visible on the page.

ExpectedConditions API

Static method	Return type	Description
visibilityOf(WebElement element)	**WebElement**	Checks that an element present in the pages DOM is visible.
titleIs(String title)	**Boolean**	Checks that the pages title is the given input **String**.
visibilityOfAllElements(List<WebElement> elements)	**List<WebElement>**	Checks that all elements present in the pages DOM are visible.
textToBe(By locator, String value)	**Boolean**	Checks that an element with the given locator has the specified text content given by the string **value**.
elementToBeClickableBy(By locator)	**WebElement**	Checks whether an element is visible and enabled so that it is clickable.

Implicit waits

Implicitly waiting for elements

- Implicit waits are distinctly different from explicit waits, the WebDriver polls the DOM for a certain duration when trying to find any element, this is useful for elements that may not immediately load upon requesting a web page.

- Implicit waiting is disabled by default and should not be mixed with explicit waits, this is because they can cause unpredictable results such as blocking the thread for the maximum amount of time specified even if the element is available or a condition is true.

Configuring implicit waiting

Poll the DOM for 2 seconds when searching for elements

```
Timeouts timeouts = driver.manage().timeouts();
timeouts.implicitlyWait(Duration.ofSeconds(2));
```

- Implicit waits are simple in nature and just require us to access the Timeouts object available on each WebDriver instances Options object.
- The default implicit wait setting is 0 (disabled), and once set is active for the lifetime of the active session.
- Use the Timeouts.implicitlyWait(Duration duration) instance method to activate implicit waiting for elements:

Fluent waits

Waiting fluently for elements

- Fluent waits define a maximum amount of time to wait for some condition to be true alongside a polling frequency with which to check the condition.
- Fluent waits are programmatically built using methods that return a **FluentWait** object.
- Once we have a **FluentWait** object, we then call the **Wait.until(Function isTrue)** instance method and pass it a **ExpectedConditions** object or a function expression.

Fluent Wait – Example 1

1. Create the **FluentWait** object.

```
Wait<WebDriver> fluentWait = new FluentWait<WebDriver>(driver)
        .withTimeout(Duration.ofSeconds(10)) // wait a max of 10 seconds
        .pollingEvery(Duration.ofSeconds(2)) // check every 2 seconds
        .ignoring(NoSuchElementException.class); // ignore missing element when waiting
```

2. Call .until() with the function to find the element as input.

```
WebElement searchBar = fluentWait.until(new Function<WebDriver, WebElement>() {

    @Override
    public WebElement apply(WebDriver driver) {
        return driver.findElement(By.name("q"));
    }

});
```

Fluent Wait – Example 2

1. Pass a lambda expression, an implementation of a functional interface, instead of creating an anonymous class directly:

```
WebElement searchBar = fluentWait.until(driver -> driver.findElement(By.name("q")));
```

2. Using an **ExpectedConditions** object:

```
WebElement searchBar =
  fluentWait.until(ExpectedConditions.elementToBeClickable(By.name("q")));
```

- The **elementToBeClickable** method returns a **ExpectedCondition** object in this case, this inherits from the **Function** interface and takes a **WebDriver** as input in its associated apply method

Exercise 10

See exercise 10 in your Selenium WebDriver exercise book.


Thank you!
Hope you enjoyed this learning journey.