Wait Commands

Every automation tool has a Synchronization Problem. To solve this problem, I use Wait Commands from WevDriver.

Synchronization Problem:

Normally when I run my app, I start the execution of automation scripts. All the statements get executed one by one on the app in the browser. My web script execution is usually faster than the web response. If the app performance is poor, my script does not wait. If the script is trying to locate a specific element on a page that has not loaded yet, the script Fails. There is no sync between the app and the script. They work at different speeds.

To solve this Synchronization Problem I can:

1) Include the speed of the app, which depends on multiple factors.
2) Pause my code for some time, while the app loads completely.

I cannot immediately control the speed of the app. It depends on many factors:

1) Many people are accessing the app at the same time, so the performance is slow.
2) Network Speed.

I can pause my code for some time, while the app gets displayed. Then, I can proceed with actions.

Python Selenium WebDriver waits:

1) **Implicit Wait**
2) **Explicit Wait**

In Java, there is also a Fluent Wait, which is another flavor of Explicit Wait. Each Wait statement has its advantages and disadvantages.

If my app is slow and unstable, I insert the Wait Commands. Otherwise, I do not necessarily use them on a daily basis.

I cannot make the app speed up or down. For most apps, at a normal speed the pages and elements are ready and loaded within 1-2 seconds. In those cases, I don’t see a Synchronization Problem.

Whenever I encounter Synchronization Problem points, I add those points to my script to automatically solve it.

95-99% of the time, I do not encounter the Synchronization Problem. Most often I come across it, when searching for something on my app. Data comes from a remote server machine, which can take some time. Then the response arrives at the Presentation Layer and displays on GUI.

There may be a fraction-of-a-second delay. In these scenarios, there's a chance of getting a Synchronization Problem.

1. Go to https://google.com.
2. Identify the search box element.
3. .send_keys("Selenium").
4. .submit() to Enter.

.submit() -> analog of Enter/Return key

After submitting, on the web page, I get multiple links with the "Selenium" keyword. It takes some time to load the results. But I don't need all the links, I only need to search for the link which exactly matches (has link_text) "Selenium". At different times, this link may show either at the beginning, the middle, or the end. The position on the page does not matter. I want to go and find that element. To do that, I write a customized Xpath which exactly points to that element.

The .click() method can throw an exception if the desired link takes some time to load. NoSuchElementException is thrown. Not because the element isn't available, it may be slowly loading on the page. It's just not available at the time the action statement gets executed. In this case, I may also encounter a Synchronization Problem.

A way to solve it is to use the Wait Commands.

Implicit Wait is provided by Selenium WebDriver, because I interact with elements directly while working with the Wait statements. It's not predefined/in-built in Python as in the 'time' module statement time.sleep(5).

As soon as I create my driver instance, I specify **driver.implicitly_wait(10)**.
10 seconds is an optimal max wait time, but I can specify any time.
If a page takes longer than 10 seconds, I have to directly report a performance bug.
Default timeout is 0 seconds.

Set the Implicit Wait at the beginning of the code. Why? Because it applies globally to all the statements in the script. It's one of the greatest advantages of Implicit Wait. When a Synchronization Problem occurs, the single global Implicit Wait takes care of it.

Whenever I create this Implicit Wait, until/unless the 'driver' gets quit, closed or killed by some other process, the Wait always works. It's alive and available. When I run commands driver.quit() or driver.close(), the Implicit Wait process gets automatically killed.

If I place Implicit Wait not at the beginning, but in the middle of the script, it'll apply to all the following lines of code, not the preceding.

Every statement does not wait for the entire max timeout of 10 seconds. As soon as the element is found/available, it immediately moves onto executing the next statement. Even if it's less than 10 seconds. And performance issues, eg, app speed decline, that happen with time.sleep(), are absent.

Implicit Wait

Advantages:

- Single Statement
- Performance is not reduced. If an element is available within the timeout, it proceeds to execute further statements.

Disadvantages:

- If an element is not available within the timeout, there is a chance of getting an exception.

To handle the Implicit Wait **Exception**, I wrap the (find_element) statement in a **try-catch block**. Because I don't want to interrupt the program and exit it unexpectedly.

Implicit Wait is completely based on **time**. The Wait specifies the max timeout. When the time runs out and the element is still not loaded, Implicit Wait throws an Exception, handled in the try-catch block, and proceeds executing further statements.

But in **Explicit Wait** I use a **condition**. I still use time, but specify a condition on top of it. Condition plays a more important role than time in Explicit Wait.
Those conditions are: element_to_be_visible
element_to_be_clickable
element_to_be_located
element_to_be_present
By specifying these conditions, I can wait for the element until it's displayed.
The condition is that I wait for the element.

I use a special WebDriverWair class and create a 'mywait' object:

import WebDriverClass

With Explicit Wait, I do not use the .find_element() method, because the locator is included in the Wait condition:

import expected_conditions as EC

Explicit Wait consists of 2 parts:

1) Declaration
2) Usage/Utilization

Declaration:

```
                         mywait = WebDriverWait(driver,10)
                         ↑                 ↑    ↑
create an object for Wait          driver instance   max time
```

Utilization:

mywait.until(EC.presence_of_element_located((By.XPATH, “...”)))

‘mywait’ is a WebDriverWait object. Through that object I call the method .until() and pass in the condition.

I wait for the condition to turn true. It turns true if the element is present on the web page. If the condition is satisfied, ‘mywait’ statement returns an element. Until the condition is satisfied, the element is not returned.

Store the element in a variable ‘’searchlink and then perform an action:

```
searchlink = mywait.until(EC.presense_of_element_located((By.XPATH, “...”)))
searchlink.click()
```

If the element is not at all available on the web page, there is a bug in my app. The condition (in the Utilization part) does not get satisfied, it never turns true. And the execution cursor stops before the action. When the element is not found within the max time (specified in the Declaration part), it exits the statement without prolonged waiting.

I do not wait for 10 seconds every time. If the element does not load for various reasons, then automatically after 10 seconds, the time expires and execution moves onto the next statement.

If the element is not found, it also throws an **Exception** like Implicit Wait. But there’s a choice of handling that exception automatically in **Explicit Wait**. In Implicit Wait I have to write a try-catch block, no such thing in Explicit Wait. I use a different, more advanced syntax with more params.I have to specify the exceptions explicitly while creating WebDriverWait:

```
mywait = WebDriverWait(driver, 10, ignored_exceptions=
        [NoSuchElementException,
        ElementNotVisibleException,
        ElementNotSelectableException])
```

To avoid exceptions, use the **ignored_exception** parameter. Specify the list of exceptions in brackets [ ]. Can replace all exceptions with one generic ‘Exception’:

```
mywait = WebDriverWait(driver, 10, ignored_exceptions=[Exception])
```

I have to insert Explicit Wait multiple times. Suppose in 10 places there’s a chance of getting a Synchronization Problem. In all 10 places I have to write the conditions, because each element is different. Because Explicit Wait works based on a condition, not time.I go to whichever element takes a long time to load and specify the condition along with the element locator. Although I write the Wait Utilization and use the ‘mywait’ object multiple times, the Declaration needs to be written only once. Explicit Wait is more effective than the rest of Wait statements.

Explicit Wait

Advantages:

- Work more effectively

Disadvantages:

- Multiple places
- More complex lengthy syntax

Right now I wait for the (condition) element presence for a max of 10 seconds. After 10 seconds, it proceeds with an exception and continues with the rest of the statements.

I can pass one more parameter to WebDriverWait *poll_frequency=2* to poll the element every 2 seconds.Polling means going and trying to find that element. *poll_frequency* time should be shorter than the max timeout. Within 10 seconds, it polls the element 5 times total. This makes Explicit Wait faster

mywait = WebDriverWait(driver, 10, poll_frequency=2, ignored _exceptions=[Exception])