SELENIUM

EMP ID: 43317

Prepared By: S.Lochani Vilehya

INDEX:

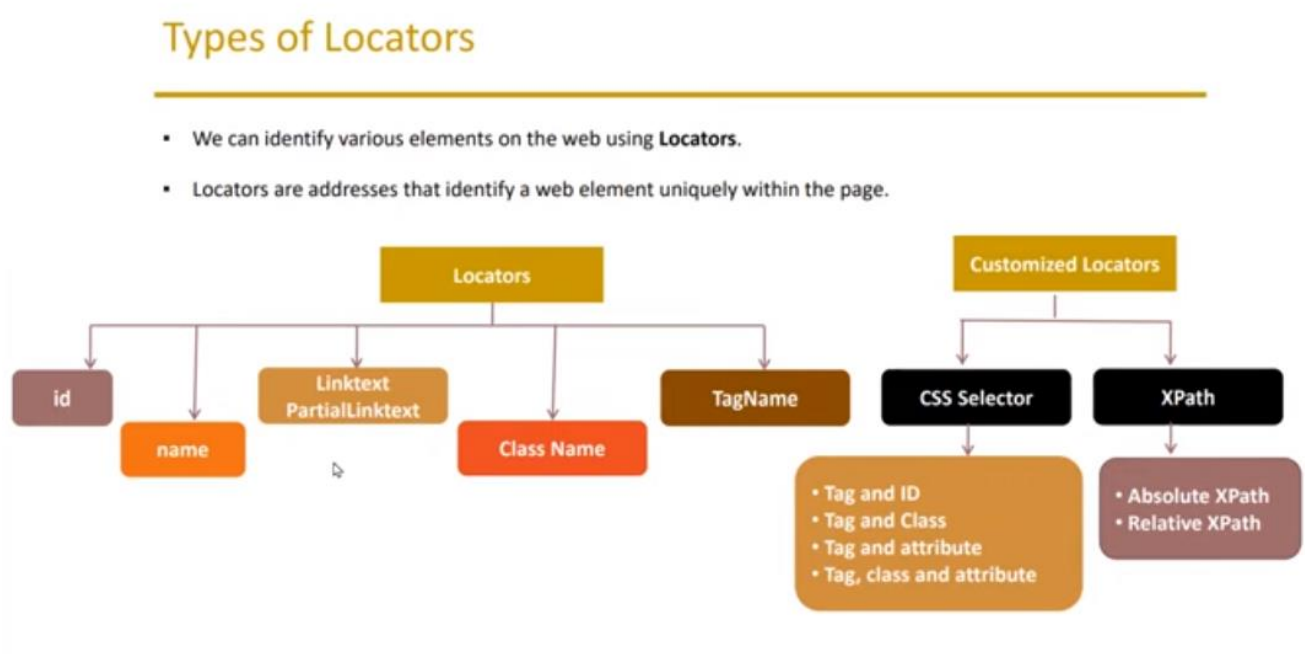
7. **WEB ELEMENTS**
    - **Checkbox**
    - **Links**
        - **Internal**
        - **External**
        - **Broken ink**
    - **Dropdown list**
    - **Alerts**
    - **Authentication popup**
    - **Frames/iframes**
    - **Tables**
    - **Datepicker**
8. **MOUSE OPERATIONS**
    - **Mouse hover**
    - **Right click**
    - **Double click**
    - **Drag and drop**
    - **SLIDER**
    - **SCROLLING PAGES**
9. **TABS AND WINDOWS**
    - **Switching tabs**
    - **Switching windows**
10. **HANDLING COOKIES**
11. **HEADLESS MODE**

Selenium WebDriver

1. WebDriver is one of the components in selenium
2. It is a module

Browser	**Class from WebDriver module**
Firefox browser	Firefox()
Chrome browser	Chrome()
edge	Edge()

3. It is an API(Application Programming Interface)



Verifying the Webpage title:

```
#test case
#----------------
#1)open the web browser
#2)open the url
```

```
#3)enter username and password
#4)click on the login
#5)capture title of the page
#6)verify the title of the page
#7)close browser


from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
sleep(2)
driver.find_element(By.NAME,"username").clear()
driver.find_element(By.NAME,"username").send_keys("Admin")
driver.find_element(By.NAME,"password").send_keys("admin123")

driver.find_element(By.XPATH,"//button[@type='submit']").click()
sleep(2)

act_title = driver.title
exp_title = "OrangeHRM"
def test_title():
    assert act_title == exp_title

driver.close()
```

Link_text and Partial_Link_text:

```
rom time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
```

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

options = Options()
options.add_experimental_option("detach", True)
driver = webdriver.Chrome(options=options)
driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
original_window = driver.current_window_handle
sleep(2)

# Clicking on the link with exact text "OrangeHRM, Inc"
driver.find_element(By.LINK_TEXT, "OrangeHRM, Inc").click()
sleep(2)

# Switch to the new window
for window_handle in driver.window_handles:
    print(window_handle)
    if window_handle != original_window:
        driver.switch_to.window(window_handle)
        break

# Print the title of the newly opened webpage
print("Title of the webpage:", driver.title)
driver.close()
driver.quit()
```

Class_Name and Tag_name:

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("http://www.automationpractice.pl/index.php")
sleep(2)

sliders = driver.find_elements(By.CLASS_NAME,"homeslider-container")
print(f"no.of sliders in webpage : {len(sliders)}")
```

```
# for i in sliders:
#     print(i)

links = driver.find_elements(By.TAG_NAME,"a")
print(f"no.of links in webpage : {len(links)}")
# for i in links:
#     print(i)

driver.close()
```

CSS Selector:

1.	tag id	Tagname#valueofId	Input#email (or) #email
2.	Tag class	Tagname.valueofClass	Input.inputtext (or) .inputtext
3.	Tag attribute	Tagname[attribute=value]	Input[type=text]
4.	Tag class attribute	Tagname.class[attribute=value]	Input.inputtext[type=text]

*Tag is optional while mentioning the value of the locator

1)Using class

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class CSS:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://www.facebook.com/")
        sleep(2)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # tag id
    def tag_id(self):
        self.driver.find_element(By.CSS_SELECTOR, "input#email").clear()
        tab = self.driver.find_element(By.CSS_SELECTOR, "input#email")
        tab.send_keys("lochu5vilehya@gmail.com")
```

```
        return tab.get_attribute('value')

    def tag_class(self):
        clss = self.driver.find_element(By.CSS_SELECTOR, "input.inputtext")
        clss.clear()
        clss.send_keys("lochu5vilehya@gmail.com")
        return clss.get_attribute('value')

    def tag_attribute(self):
        attr = self.driver.find_element(By.CSS_SELECTOR,
"input[type=text]")
        attr.clear()
        attr.send_keys("lochu5vilehya@gmail.com")
        sleep(2)
        return attr.get_attribute('value')

    def tag_classAttribute(self):
        clsattr = self.driver.find_element(By.CSS_SELECTOR,
"input.inputtext[type=text]")
        clsattr.clear()
        clsattr.send_keys("lochu5vilehya@gmail.com")
        sleep(2)
        clspswd = self.driver.find_element(By.CSS_SELECTOR,
"input.inputtext[type=password]")
        clspswd.send_keys("lochu5vilehya")
        sleep(2)
        return
clsattr.get_attribute('value'),clspswd.get_attribute('value')


@pytest.fixture(scope="module")
def css():
    css = CSS()
    css.setup()
    yield css
    css.teardown()


#tag_id
@pytest.mark.css_selector
def test_tagId(css):
```

```
    assert css.tag_id() == "lochu5vilehya@gmail.com"


# tag class
@pytest.mark.css_selector
def test_tagclass(css):
    assert css.tag_class() == "lochu5vilehya@gmail.com"

# tag attribute
@pytest.mark.css_selector
def test_attr(css):
    assert css.tag_attribute() == "lochu5vilehya@gmail.com"


# tag class attribute
@pytest.mark.css_selector
def test_clsattr(css):
    val1,val2 = css.tag_classAttribute()
    assert val1 == "lochu5vilehya@gmail.com"
    assert val2 == "lochu5vilehya"
```

2)Without Using Class

```
from selenium.webdriver.common.by import By

import pytest
from selenium import webdriver
from time import sleep
@pytest.fixture(scope="module")
def driver():
    # Create a WebDriver instance (browser) for each test
    driver = webdriver.Chrome()
    yield driver
    # Teardown - close the browser after each test
    driver.quit()
```

```python
@pytest.mark.css
def test_tagId(driver):
    driver.get("https://www.facebook.com/")
    email_input = driver.find_element(By.CSS_SELECTOR, "input#email")
    email_input.clear()
    email_input.send_keys("lochu5vilehya@gmail.com")
    assert email_input.is_displayed()


@pytest.mark.css
def test_tagClass(driver):
    driver.get("https://www.facebook.com/")
    email_input = driver.find_element(By.CSS_SELECTOR, "input.inputtext")
    email_input.clear()
    email_input.send_keys("lochu5vilehya@gmail.com")
    assert email_input.get_attribute('value') == "lochu5vilehya@gmail.com"



# tag attribute
@pytest.mark.css
def test_attr(driver):
    driver.get("https://www.facebook.com/")
    attr = driver.find_element(By.CSS_SELECTOR, "input[type=text]")
    attr.clear()
    attr.send_keys("lochu5vilehya@gmail.com")
    sleep(2)
    assert attr.get_attribute('value') == "lochu5vilehya@gmail.com"


# tag class attribute
@pytest.mark.css
def test_Class_Attr(driver):
    driver.get("https://www.facebook.com/")
    clsAttr = driver.find_element(By.CSS_SELECTOR,
"input.inputtext[type=text]")
    clsAttr.clear()
    clsAttr.send_keys("lochu5vilehya@gmail.com")
    sleep(2)
    clsPswd = driver.find_element(By.CSS_SELECTOR,
"input.inputtext[type=password]")
```

```
    clsPswd.send_keys("lochu5vilehya")
    sleep(2)
    assert clsAttr.get_attribute('value') == "lochu5vilehya@gmail.com"
    assert clsPswd.get_attribute('value') == "lochu5vilehya"
```

XPATH

i. It is defined as XML path
ii. It is a syntax or language for finding any element on the web page using XML path expression
iii. It is used to find the location of any element on a webpage using HTML DOM structure
iv. It is an address of the element
v. It is used to navigate through elements and attributes in DOM

Types of xpath:

1) Absolute/Full xpath
   Eg: /html/body/nav/div/div[2]/ul[3]/li[3]/a
2) Relative/Partial xpath
   Eg: //*[@id="header-navbar"]/ul[3]/li[1]/a

Difference between Absolute and Relative Xpaths

Absolute XPATH	Relative XPATH
path starts from root html node	directly jump to element on DOM
Starts with /	Starts with //
We use only tags and nodes	In this we use attributes

Syntax of writing relative xpath : //tagname[@attribute = 'value']

How to capture xpath automatically:

Right click on element → inspect → highlight html code → right click → copy xpath

Reasons to prefer relative xpath:

1) If developer introduced new element, then absolute xpath is broken
2) If developer changed the location, then absolute xpath will be broken
3) So absolute xpath is unstable

Xpath options:

i. And
ii. Or
iii. Contains()
iv. Startswith()
v. Text()

And eg : **// input[@ name = 'search_query' and @placeholder='Search']**

Or eg: **// input[@ name = 'search_query' or @placeholder='Search']**

If there is a button whose id is ‘start’ but when we click it the id turns into ‘stop’ at this case we can’t use simple relative path as the id is changing on clicking at that time we can use //*[@id=’start’ or @id=’stop’] (or) other options like mentioned below.

options	**Id = start**	**Id = stop**
xpath	//*[@id=’start’]	//*[@id=’stop’]
Contains option	//*[contains(@id,’st’)]	//*[contains(@id,’st’)]
Starts-with option	//*[starts-with(@id,’st’)]	//*[starts-with(@id,’st’)]

Text eg : **//a[text()='Women']**

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class XPATH:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://www.automationpractice.pl/index.php")
        self.driver.maximize_window()
        sleep(2)

    def teardown(self):
```

```
        if self.driver is not None:
            self.driver.quit()

    #absolute path
    def absxpath(self):
        search = self.driver.find_element(By.XPATH,
"/html/body/div/div[1]/header/div[3]/div/div/div[2]/form/input[4]")
        sleep(2)
        search.clear()
        search.send_keys("shirts")
        sleep(2)
        return search.get_attribute('value')

    #relative path
    def relxpath(self):
        #search = self.driver.find_element(By.XPATH,
"//input[@name='search_query']")
        #search = self.driver.find_element(By.XPATH, "// input[ @ name =
'search_query' and @placeholder='Search']")
        #search = self.driver.find_element(By.XPATH, "// input[ @ name =
'search_query' or @id='Search']")
        #search = self.driver.find_element(By.XPATH,
"//input[contains(@id,'search')]")
        search = self.driver.find_element(By.XPATH,"//input[starts-
with(@name,'search')]")
        sleep(2)
        search.clear()
        search.send_keys("shirts")
        sleep(2)
        return search.get_attribute('value')

    #xpath option - text()
    def xpath_text(self):
        link_text = self.driver.find_element(By.XPATH,"//a[text() =
'Women']")
        link_text.click()
        sleep(2)
        self.driver.back()
        if link_text.is_enabled():
            return True
        else:
            return False

@pytest.fixture(scope="module")
def xpath_fix():
    xpath = XPATH()
    xpath.setup()
    yield xpath
    xpath.teardown()


@pytest.mark.xpath
def test_absXpath(xpath_fix):
    assert xpath_fix.absxpath() == "shirts"


@pytest.mark.xpath
def test_relXpath(xpath_fix):
    assert xpath_fix.relxpath() == "shirts"
```

```python
@pytest.mark.xpath
def test_xpathText(xpath_fix):
    assert xpath_fix.xpath_text() == True
```

XPATH AXES:

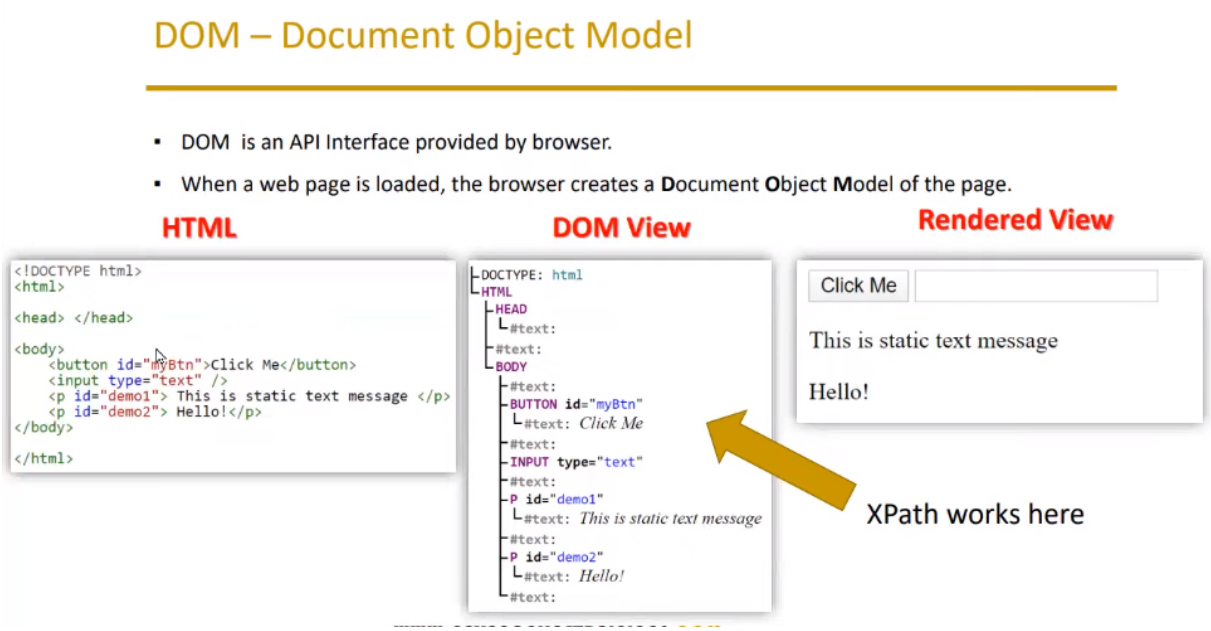
In Selenium, XPath axes allow you to navigate the HTML document's structure to locate elements relative to other elements. You can use XPath axes to traverse the DOM hierarchy and find elements based on their relationships with other elements.

Here are some common XPath axes used in Selenium:

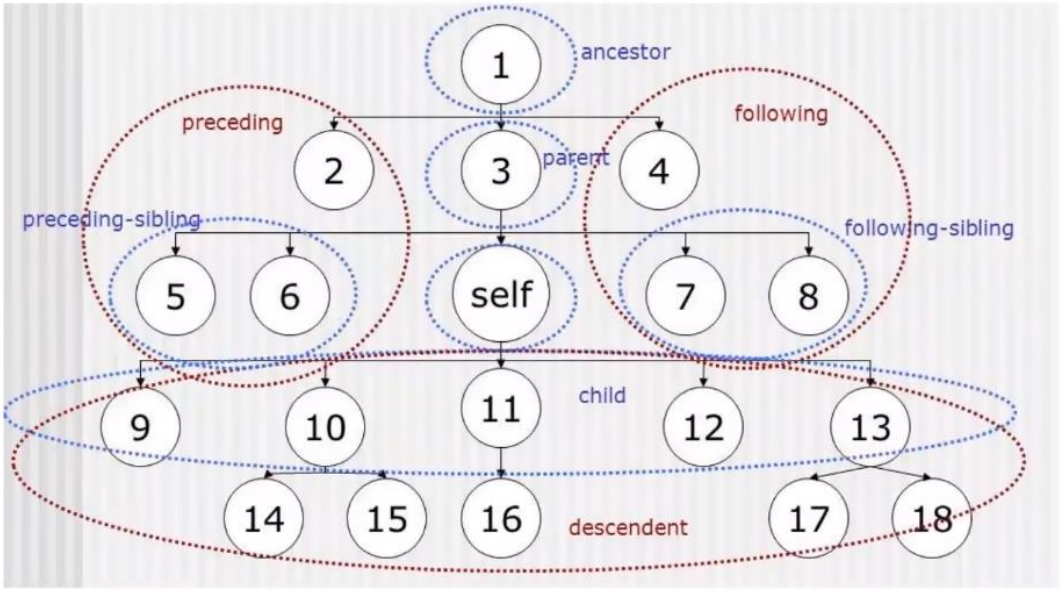
1. **Parent Axis (parent::)**: Selects the parent of the current node. Example: **//div[@class='parent']/parent::div**
2. **Ancestor Axis (ancestor::)**: Selects all ancestors of the current node. Example: **//div[@class='descendant']/ancestor::div**
3. **Child Axis (child::)**: Selects all children of the current node. Example: **//div[@class='parent']/child::p**
4. **Descendant Axis (descendant::)**: Selects all descendants of the current node. Example: **//div[@class='ancestor']/descendant::p**
5. **Following-sibling Axis (following-sibling::)**: Selects all siblings that appear after the current node. Example: **//div[@class='sibling']/following-sibling::div**
6. **Preceding-sibling Axis (preceding-sibling::)**: Selects all siblings that appear before the current node. Example: **//div[@class='sibling']/preceding-sibling::div**
7. **Following Axis (following::)**: Selects all nodes that appear after the current node. Example: **//div[@class='following']/following::p**
8. **Preceding Axis (preceding::)**: Selects all nodes that appear before the current node. Example: **//div[@class='preceding']/preceding::p**

9. **Self Axis(self::):** **self** axis represents the current node itself . Example: **//div[@class='example']/self::div**

These axes can be combined with other XPath expressions to locate specific elements on a web page. By understanding how to use XPath axes effectively, you can write more robust and precise locators in your Selenium tests



Relationship of Nodes



Axes	Description	Syntax
Child	Traverse all **child element** of the **current html tag**	//*[attribute='value']/child::tagname
Parent	Traverse **parent element** of the **current html tag**	//*[attribute='value']/parent::tagname
Following	Traverse **all element** that comes **after the current tag**	//*[attribute='value']/following::tagname
Preceding	Traverse **all nodes** that comes **before the current html tag.**	//*[attribute='value']/preceding::tagname
Following-sibling	Traverse from **current Html tag** to **Next sibling Html tag**.	//current html tag[@attribute ='value']/following-sibling:: sibling tag[@attribute ='value']
Preceding-sibling	Traverse from **current Html tag** to **previous sibling Html tag.**	//current html tag[@attribute ='value']/preceding-sibling:: previous tag[@attribute ='value']
Ancestor	Traverse all the **ancestor elements** (grandparent, parent, etc.) of **the current html tag**.	//*[attribute='value']/ancestor::tagname
Descendant	Traverse all **descendent element** (child node, grandchild node, etc.) of the **current Html tag**.	//*[attribute='value']/descendant::tagname

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class XPATH_AXES:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()

self.driver.get("https://money.rediff.com/gainers/bse/daily/groupa")
        #self.driver.maximize_window()
        sleep(2)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    #absolute path
    def xpath_ax(self):
        #self
        txt = self.driver.find_element(By.XPATH,
"//a[contains(text(),'Indian Overseas')]/self::a").text
        print(txt)

        #parent
        txt1=
self.driver.find_element(By.XPATH,"//a[contains(text(),'Indian
Overseas')]/parent::td").text
        print(txt1)

        #ancestor
        childs = self.driver.find_elements(By.XPATH,
"//a[contains(text(),'Indian Overseas')]/ancestor::tr/child::td")
        for i in childs:
            print(i.text)
        sleep(2)

        #descendant
        des = self.driver.find_elements(By.XPATH,
"//a[contains(text(),'Indian Overseas')]/ancestor::tr/descendant::*")
        print("no.of descendants : ", len(des))
        print("list of descendants")
        for i in des:
            print(i.text)
        sleep(2)

        # following
        follow = self.driver.find_elements(By.XPATH,
"//a[contains(text(),'Indian Overseas')]/ancestor::tr/following::*")
        print("no.of following nodes: ",len(follow))

        # following-sibling
        follow_sib =
self.driver.find_elements(By.XPATH,"//a[contains(text(),'Indian
Overseas')]/ancestor::tr/following-sibling::*")
        print("no.of following sibling nodes : ",len(follow_sib))
```

```
        # following
        pre =
self.driver.find_elements(By.XPATH,"//a[contains(text(),'Indian
Overseas')]/ancestor::tr/preceding::*")
        print("no.of preceding nodes: ", len(pre))

        # following-sibling
        pre_sib =
self.driver.find_elements(By.XPATH,"//a[contains(text(),'Indian
Overseas')]/ancestor::tr/preceding-sibling::*")
        print("no.of preceding sibling nodes : ", len(pre_sib))
        return txt



@pytest.fixture(scope="module")
def xpath_axes():
    xpath = XPATH_AXES()
    xpath.setup()
    yield xpath
    xpath.teardown()


@pytest.mark.xpath_axes
def test_absXpath(xpath_axes):
    assert xpath_axes.xpath_ax() == "Indian Overseas"


'''
============================== 1 passed in 46.38s
==============================
PASSED                                  [100%]Indian Overseas
Indian Overseas
Indian Overseas
A
66.76
68.34
+ 2.37
no.of descendants :  7
list of descendants
Indian Overseas
Indian Overseas
A
66.76
68.34
+ 2.37
+ 2.37
no.of following nodes:  2495
no.of following sibling nodes :  288
no.of preceding nodes:  812
no.of preceding sibling nodes :  78
'''
```

BASIC COMMANDS IN SELENIUM

1. application commands
2. Conditional commands
3. Browser commands
4. Navigational commands
5. Wait commands

application commands:

- These are browser specific commands
- **Application commands** :
  - **get()** : opening the application url
  - **title** : to capture the title of the current webpage
  - **current_url** : to capture the current url of the webpage
  - **page_source** : to capture the code of the page

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class APP:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
        sleep(2)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # tag id
    def title_page(self):
```

```
        return self.driver.title

    def url_page(self):
        return self.driver.current_url

    def view_page_src(self):
        return self.driver.page_source

@pytest.fixture(scope="module")
def app():
    app = APP()
    app.setup()
    yield app
    app.teardown()


@pytest.mark.app_cmd
def test_title(app):
   assert app.title_page() == "OrangeHRM"


@pytest.mark.app_cmd
def test_url(app):
    assert app.url_page() == "https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login"

@pytest.mark.app_cmd
def test_url(app):
    print(app.view_page_src())
```

- **conditional commands:**
  - **is_displayed()** : This method checks whether the web element is currently visible or not on the web page. It returns a boolean value - True if the element is visible, and False if it is not.
  - **is_enabled()** : This method checks whether the web element is currently enabled or not. For example, a button may be present on a page but disabled, so is_enabled() would return False in that case. It returns a boolean value - True if the element is enabled, and False if it is disabled.
  - **is_selected()** : This method is generally used with checkboxes, radio buttons, and dropdown options. It checks whether the web element is currently selected or not. For checkboxes and radio buttons, it returns True if the element is checked, and False if it is

unchecked. For dropdown options, it returns True if the option is currently selected, and False if it is not.

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class COND:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()

self.driver.get("https://demo.nopcommerce.com/register?returnUrl=%2F")
        sleep(2)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # tag id
    def displayed(self):
        search = self.driver.find_element(By.ID,"small-searchterms")
        return search.is_displayed()

    def enabled(self):
        en = self.driver.find_element(By.ID, "small-searchterms")
        return en.is_enabled()

    #s_selected() - for radio buttons and check boxes
    def selected(self):
        male = self.driver.find_element(By.ID, "gender-male")
        female = self.driver.find_element(By.ID, "gender-female")
        print("default status of radio button..")
        print("male status : ",male.is_selected())
        print("female status : ",female.is_selected())
        female.click()
        print("after selecting the female radio button..")
        print("male status : ", male.is_selected())
        print("female status : ", female.is_selected())
        return female.is_selected()

@pytest.fixture(scope="module")
def con():
    con = COND()
    con.setup()
    yield con
    con.teardown()


@pytest.mark.cond_cmd
def test_display(con):
   assert con.displayed() == True
```

```
@pytest.mark.cond_cmd
def test_enable(con):
    assert con.enabled() == True

@pytest.mark.cond_cmd
def test_gender(con):
     con.selected()  == True
```

- **Browser commands:**
  - **close** : close single browser window where the driver is focused
  - **quit()** : close multiple browser windows (this will kill the process)

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class Browser:

    def __init__(self):
        self.driver = webdriver.Chrome()

    def open_url(self):
        self.driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
        sleep(3)
        tab = self.driver.find_element(By.PARTIAL_LINK_TEXT, "OrangeHRM")
        tab.click()
        sleep(2)
    def close_tab(self):
        self.driver.close()

    def quit_tab(self):
        self.driver.quit()

@pytest.fixture(scope="module")
def browser():
    browser = Browser()
    yield browser




# @pytest.mark.browser_cmd
# def test_closing(browser):
#     browser.open_url()
#     browser.close_tab()


@pytest.mark.browser_cmd
def test_quitting(browser):
    browser.open_url()
    browser.quit_tab()
```

- **navigational commands:**
  - **back()**
  - **forward()**
  - **refresh()**

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class NAV:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")

self.driver.get("https://demo.nopcommerce.com/register?returnUrl=%2F")
        sleep(2)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # tag id
    def back_page(self):
        self.driver.back()
        sleep(2)
        return self.driver.title

    def forward_page(self):
        self.driver.forward()
        sleep(2)
        return self.driver.title

    def refresh_page(self):
        self.driver.refresh()
@pytest.fixture(scope="module")
def app():
    app = NAV()
    app.setup()
    yield app
    app.teardown()


@pytest.mark.nav_cmd
def test_back(app):
   assert app.back_page() == "OrangeHRM"


@pytest.mark.nav_cmd
def test_forward(app):
    assert app.forward_page() == "nopCommerce demo store. Register"
```

```
@pytest.mark.nav_cmd
def test_refresh(app):
    app.refresh_page()
```

- **find_element() vs find_elements()**
  - **find_element()** : returns single web-element
  - **find_elements()** : returns multiple web-elements
- **text vs get_attribute('value')**
  - **text** : it always returns the inner text of the web-element

    eg: <input id='123' name='xyz'> **Email:** <\input>
  - **get_attribute() :** it returns the value of the attribute of the web-element

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class Attribute:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://admin-
demo.nopcommerce.com/login?returnUrl=%2F")
        sleep(2)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()
    def input_text(self):
        ip = self.driver.find_element(By.ID, "Email")
        ip.clear()
        ip.send_keys("abc@gmail.com")
        sleep(2)
        print(ip.text)
        return ip.get_attribute('value')

    def login_btn(self):
        btn = self.driver.find_element(By.XPATH,
"//button[@type='submit']")
        print("inner text: ",btn.text)
        print("value of attribute: ", btn.get_attribute('value'))
        print("type of attribute: ", btn.get_attribute('type'))
        return btn.text
```

```
@pytest.fixture(scope="module")
def attr():
    attr = Attribute()
    attr.setup()
    yield attr
    attr.teardown()


@pytest.mark.text_cmd
def test_text(attr):
    assert attr.input_text() == "abc@gmail.com"

@pytest.mark.text_cmd
def test_getattr(attr):
    assert attr.login_btn() == "LOG IN"
```

- **wait commands:**
  - **sleep : sleep(2)**
    - **advantages**
      1. simple to use
    - **Disadvantages**
      1. performance of the script is very poor
      2. if the element is not available within the time mentioned , still there is a chance of getting exceptions
  - **implicit wait : driver.implicitly(10)**
    - **advantages**
      1. single statement
      2. performance will not be reduced (if element is available within the time it proceeds to execute further statements)
    - **Disadvantages**
      1. if the element is not available within the time mentioned , still there is a chance of getting exceptions

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
```

```
class Attribute:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://www.google.com")
        self.driver.implicitly_wait(4)
    def teardown(self):
        if self.driver is not None:
            self.driver.quit()
    def search_text(self):
        ip = self.driver.find_element(By.NAME, "q")
        ip.send_keys("Selenium")
        ip.submit()

self.driver.find_element(By.XPATH,"(//h3[text()='Selenium'])[1]").click()


@pytest.fixture(scope="module")
def attr():
    attr = Attribute()
    attr.setup()
    yield attr
    attr.teardown()


@pytest.mark.impwait_cmd
def test_text(attr):
     attr.search_text()
```

- **explicit wait : it works based on condition**
  - **advantages**
    1. works more efficiently
  - **Disadvantages**
    1. Multiple places
    2. Feels some difficulties

```
import time
import pytest
from selenium import webdriver
from selenium.common import NoSuchElementException,
ElementNotVisibleException, ElementNotSelectableException
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

class Attribute:
```

```
    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://www.google.com")

self.mywait=WebDriverWait(self.driver,5,poll_frequency=2,ignored_exceptions
=[NoSuchElementException,

ElementNotVisibleException,

ElementNotSelectableException,

Exception]
                                    )

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()
    def search_text(self):
        ip = self.driver.find_element(By.NAME, "q")
        ip.send_keys("Selenium")
        ip.submit()
        search_link =
self.mywait.until(EC.presence_of_element_located((By.XPATH,
"(//h3[text()='Selenium'])[1]")))
        search_link.click()


@pytest.fixture(scope="module")
def attr():
    attr = Attribute()
    attr.setup()
    yield attr
    attr.teardown()


@pytest.mark.expwait_cmd
def test_text(attr):
    attr.search_text()
```

WEB-ELEMENTS

1. CheckBoxes

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By


class CHECK:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://testautomationpractice.blogspot.com/")
        self.driver.implicitly_wait(4)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    #single check box
    def check_box(self):
        ip = self.driver.find_element(By.ID, "sunday")
        ip.click()
        sleep(3)
        return ip.is_selected()

    def clear_check(self):
        ch = self.driver.find_elements(By.XPATH, "//input[@type='checkbox'
and contains(@id,'day')]")
        for i in ch:
            if i.is_selected():
                i.click()
        sleep(3)
        return ch

    #select multiple check boxes at a time
    def multiple_check(self):
        ch = self.clear_check()
        for i in ch:
            i.click()
        sleep(3)
        return len(ch)

    def cond_check(self):
        ch = self.clear_check()
        for i in ch:
            weekname = i.get_attribute('id')
            if weekname == 'monday' or weekname == 'sunday':
                i.click()
        sleep(3)

        #select last 2 checkboxes
        #range(5,7) ----> 6,7
        #totalnumberofelements-2 = starting index
```

```
    def last_two(self):
        ch = self.clear_check()
        for i in range(len(ch)-2,len(ch)):
                ch[i].click()
        sleep(3)

    def first_two(self):
        ch = self.clear_check()
        for i in range(len(ch)):
            if i < 2:
                ch[i].click()
        sleep(3)
@pytest.fixture(scope="module")
def check():
    check = CHECK()
    check.setup()
    yield check
    check.teardown()


@pytest.mark.check_box
def test_checkselect(check):
    assert check.check_box() == True

@pytest.mark.check_box
def test_checkmultiSelect(check):
    assert check.multiple_check() == 7

@pytest.mark.check_box
def test_checkCondSelect(check):
    check.cond_check()

@pytest.mark.check_box
def test_last2select(check):
    check.last_two()

@pytest.mark.check_box
def test_first2select(check):
    check.first_two()
```

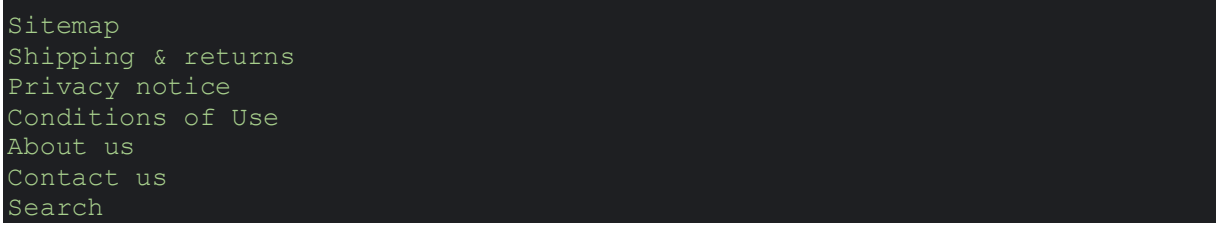
2. Links

External Links: Links that point to pages on a different domain or website. They are often used to provide references, citations, or resources from other sources.

Internal Links: Links that point to other pages within the same website or domain. They help in website navigation and improve user experience by allowing users to explore related content

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException

class LINK:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()

self.driver.get("https://demo.nopcommerce.com/register?returnUrl=%2F")
        self.driver.implicitly_wait(4)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # select multiple check boxes at a time
    def number_of_link(self):
        ip = self.driver.find_elements(By.TAG_NAME, "a")
        for i in ip:
            print(i.text)
        return len(ip)


    #single check box
    def check_link(self):
        try:
            ip = WebDriverWait(self.driver,
10).until(EC.element_to_be_clickable((By.LINK_TEXT, "Books")))
            ip.click()
            return True
        except TimeoutException:
            print("Link not found or clickable within 10 seconds.")
            return False
        except Exception as e:
            print(f"An error occurred: {e}")
            return False
```

```
@pytest.fixture(scope="module")
def link():
    link = LINK()
    link.setup()
    yield link
    link.teardown()

@pytest.mark.check_link
def test_checknooflinks(link):
    assert link.number_of_link() == 60

@pytest.mark.check_link
def test_checklink(link):
    assert link.check_link() == True


/*
PASSED                                  [ 50%]Register
Log in
Wishlist (0)
Shopping cart (0)

Computers



Electronics



Apparel



Digital downloads
Books
Jewelry
Gift Cards
```



```
Sitemap
Shipping & returns
Privacy notice
Conditions of Use
About us
Contact us
Search
```

```
News
Blog
Recently viewed products
Compare products list
New products
My account
Orders
Addresses
Shopping cart
Wishlist
Apply for vendor account
Facebook
Twitter
RSS
YouTube
nopCommerce
PASSED                                          [100%]
Process finished with exit code 0
*/
```

broken links : also known as a dead link, is a hyperlink on a webpage that no longer points to its intended destination. Instead, clicking on a broken link typically results in an error message or a page not found (404 error)

```
from time import sleep
import pytest
import requests
from selenium import webdriver
from selenium.webdriver.common.by import By


class BROKEN_LINK:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://www.deadlinkcity.com/")
        self.driver.implicitly_wait(4)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # select multiple check boxes at a time
    def number_of_link(self):
        res = None
        ip = self.driver.find_elements(By.TAG_NAME, "a")
        count = 0
        for i in ip:
            url = i.get_attribute('href')
            try:
                res = requests.head(url)
            except:
```

```
                None

            if res.status_code >= 400:
                print(url, "...this is broken link")
                count += 1
            else:
                print(url, "..it is valid link")
        return count




@pytest.fixture(scope="module")
def link():
    link = BROKEN_LINK()
    link.setup()
    yield link
    link.teardown()


@pytest.mark.broken_link
def test_checknoofbrokenlinks(link):
    assert link.number_of_link() == 40



# ============================= test session starts
==============================
# collecting ... collected 1 item
#
# test_brokenLinks.py::test_checknoofbrokenlinks
#
# ============================== 1 passed in 39.76s
==============================
# PASSED
[100%]http://www.deadlinkcity.com/comparison.asp ..it is valid link
# http://www.deadlinkcity.com/errorlist.asp ..it is valid link
# http://www.deadlinkcity.com/error-page.asp?e=400 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=401 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=402 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=403 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=404 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=405 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=406 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=407 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=408 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=409 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=410 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=411 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=412 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=413 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=414 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=415 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=416 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=417 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=420 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=422 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=423 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=424 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=429 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=431 ...this is broken link
```

```
# http://www.deadlinkcity.com/error-page.asp?e=450 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=500 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=501 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=502 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=503 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=504 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=505 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=506 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=507 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=509 ...this is broken link
# http://www.deadlinkcity.com/error-page.asp?e=510 ...this is broken link
# http://www.deadlinkcity.com/page-not-found.asp ...this is broken link
# http://www.domaindoesnot.exist/ ...this is broken link
# http://www.deadlinkcity.com/default.asp?r=1 ..it is valid link
# http://www.deadlinkcity.com/default.asp?r=2 ..it is valid link
# http://www.deadlinkcity.com/default.asp?r=4 ..it is valid link
# http://www.deadlinkcity.com/default.asp?r=5 ..it is valid link
# http://www.deadlinkcity.com/default.asp?r=6 ..it is valid link
# http://www.deadlinkcity.com/default.asp?r=7 ..it is valid link
# http://www.deadlinkcity.com/disallowed/disallowed.html ...this is broken
link
# http://www.deadlinkcity.com/disallowed/nonexistent.html ...this is broken
link
# mailto:info@deadlinkchecker.com?subject=DeadLinkCity.com%20-%20feedback
...this is broken link
#
# Process finished with exit code 0
```

3. Dropdown list

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.select import Select


class DROP_DOWN:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://demo.automationtesting.in/Register.html")
        self.driver.implicitly_wait(10)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    # select multiple check boxes at a time
    def select_country(self):
        ele = self.driver.find_element(By.XPATH,"//select[@id='Skills']")
        country = Select(ele)
        country.select_by_visible_text("Configuration")
        sleep(3)
        country.select_by_index(6)
```

```
        sleep(3)
        country.select_by_value("APIs")
        sleep(3)
        return ele.get_attribute('value')

    def using_optionxpath(self):
        ele = self.driver.find_element(By.XPATH,
"//*[@id='Skills']/option[7]")
        ele.click()
        return ele.get_attribute('value')

    def count_options(self):
        ele = self.driver.find_element(By.XPATH, "//select[@id='Skills']")
        country = Select(ele)
        all_list = country.options
        for i in all_list:
            print(i.text)
        print("total number of options: ", len(all_list))
        return len(all_list)


    #select otion without built in function
    def select_option(self):
        ele = self.driver.find_element(By.XPATH, "//select[@id='Skills']")
        country = Select(ele)
        all_list = country.options
        for i in all_list:
            if i.text == "Email":
                i.click()
        return ele.get_attribute('value')

@pytest.fixture(scope="module")
def drop():
    drop = DROP_DOWN()
    drop.setup()
    yield drop
    drop.teardown()


@pytest.mark.drop_down
def test_checkvalue(drop):
    assert drop.select_country() == "APIs"

@pytest.mark.drop_down
def test_checknoofoptions(drop):
    assert drop.count_options() == 78

@pytest.mark.drop_down
def test_checkselectedoption(drop):
    assert drop.select_option() == "Email"

@pytest.mark.drop_down
def test_xpathcheckoption(drop):
    assert drop.using_optionxpath() == "Art Design"
```

4. Alerts

Myalert = driver.switch_to.alert

Myalert.text

Myalert.accept()

Myalert.dismiss()

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.select import Select


class ALERT:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://the-
internet.herokuapp.com/javascript_alerts")
        self.driver.implicitly_wait(10)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def check_alert_Ok(self):
        self.driver.find_element(By.XPATH,"//button[normalize-
space()='Click for JS Prompt']").click()
        sleep(3)
        alert_window = self.driver.switch_to.alert
        print(alert_window.text)
        sleep(3)
        alert_window.send_keys("lochu")
        #alert_window.accept()
        alert_window.dismiss()
        sleep(3)


@pytest.fixture(scope="module")
def alert():
    alert = ALERT()
    alert.setup()
    yield alert
    alert.teardown()


@pytest.mark.alert
def test_checkalert(alert):
    alert.check_alert_Ok()
```

5. Authentication popup

Link : https://the-internet.herokuapp.com/basic_auth

Syntax : https://username:password@test.com

Example : https://admin:admin@the-internet.herokuapp.com/basic_auth

```
from time import sleep
from selenium import webdriver


driver = webdriver.Chrome()
driver.get("https://admin:admin@the-internet.herokuapp.com/basic_auth")
sleep(3)
driver.quit()
```

6.Frames/Iframes

```
from time import sleep
import pytest
from selenium import webdriver
from selenium.webdriver.common.by import By



class IFRAME:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://demo.automationtesting.in/Frames.html")
        self.driver.implicitly_wait(10)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def check_switchframe(self):
        self.driver.find_element(By.XPATH,"//a[normalize-space()='Iframe
with in an Iframe']").click()
        sleep(3)
        outer =
self.driver.find_element(By.XPATH,"//iframe[@src='MultipleFrames.html']")
        self.driver.switch_to.frame(outer)

        inner = self.driver.find_element(By.XPATH,
"/html/body/section/div/div/iframe")
        self.driver.switch_to.frame(inner)
        sleep(3)
```

```
self.driver.find_element(By.XPATH,"//input[@type='text']").send_keys("welco
me")
        sleep(3)

@pytest.fixture(scope="module")
def frame():
    frame = IFRAME()
    frame.setup()
    yield frame
    frame.teardown()


@pytest.mark.alert
def test_checkalert(frame):
    frame.check_switchframe()
```

7.tables

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import pytest
from time import sleep

class TABLE:

    def __init__(self):
        self.Numcol = None
        self.Numrow = None
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://testautomationpractice.blogspot.com/")
        self.driver.implicitly_wait(10)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def check_table(self):
        row =
self.driver.find_elements(By.XPATH,"//table[@name='BookTable']//tr")
        self.Numrow = len(row)
        sleep(3)
        column =
self.driver.find_elements(By.XPATH,"//table[@name='BookTable']//tr[1]/th")
        self.Numcol = len(column)
        ele = self.driver.find_element(By.XPATH,
"//table[@name='BookTable']/tbody/tr[5]/td[1]").text
        return self.Numrow,self.Numcol,ele
        sleep(3)

    def print_tableElements(self):
        print("printing all rows and columns from table.....")
        for i in range(2,self.Numrow+1):
```

```
            for c in range(1,self.Numcol+1):
                ele = self.driver.find_element(By.XPATH,
"//table[@name='BookTable']/tbody/tr["+str(i)+"]/td["+str(c)+"]").text
                print(ele,end='    ')
            print()

    def print_cond_Author(self):
        print("printing all books of Mukesh from table based on the
condition.....")
        for j in range(2,self.Numrow+1):
            author = self.driver.find_element(By.XPATH,
"//table[@name='BookTable']/tbody/tr["+str(j)+"]/td[2]").text
            if author == "Mukesh":
                book = self.driver.find_element(By.XPATH,
"//table[@name='BookTable']/tbody/tr["+str(j)+"]/td[1]").text
                print(book,"    ",author)
@pytest.fixture(scope="module")
def tb():
    tb = TABLE()
    tb.setup()
    yield tb
    tb.teardown()


@pytest.mark.table
def test_RowCol(tb):
    r,c,e = tb.check_table()
    assert r == 7
    assert c == 4
    assert e == "Master In Selenium"

@pytest.mark.table
def test_print_table(tb):
    tb.print_tableElements()

@pytest.mark.table
def test_print_cond_Author(tb):
    tb.print_cond_Author()
```

8.datepicker

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class Date_picker:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://jqueryui.com/datepicker/")
        self.driver.implicitly_wait(10)
```

```
    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def enter_date(self):
        self.driver.switch_to.frame(0)
        self.driver.find_element(By.XPATH,
"//*[@id='datepicker']").send_keys("10/05/2024")
        sleep(3)

    def select_date(self):
        year = "2020"
        month = "October"
        date = "5"

        res = self.driver.find_element(By.XPATH, "//*[@id='datepicker']")
        res.clear()
        res.click()
        sleep(3)

        while True:
            mon = self.driver.find_element(By.XPATH, "//span[@class='ui-
datepicker-month']").text
            yr = self.driver.find_element(By.XPATH, "//span[@class='ui-
datepicker-year']").text
            if mon == month and yr == year:
                break
            else:
                self.driver.find_element(By.XPATH, "//span[normalize-
space()='Prev']").click()
        sleep(3)
        dates = self.driver.find_elements(By.XPATH, "//div[@id='ui-
datepicker-div']//table/tbody/tr[2]/td/a")
        for ele in dates:
            if ele.text == date:
                ele.click()
                break
        sleep(3)
        return res.get_attribute('value')


@pytest.fixture(scope="module")
def dt():
    dt = Date_picker()
    dt.setup()
    yield dt
    dt.teardown()


@pytest.mark.date
def test_enterDate(dt):
    dt.enter_date()
```

MOUSE OPERATIONS

1. **Mouse hover :** move_to_element(element)
2. **Right click :** context_click(element)
3. **Double click :** double_click(element)
4. **Drag and drop** : drag_drop(source_element,target_element)

MOUSE HOVER

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class MOUSE_OP:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://testsigma.com/automated-api-testing")
        self.driver.implicitly_wait(10)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def mouse_hover(self):
        Products = self.driver.find_element(By.XPATH,"//p[normalize-
space()='Products']")
        Web = self.driver.find_element(By.XPATH,"//a[normalize-
space()='Automated Website Testing']")
        act = ActionChains(self.driver)
        sleep(2)

act.move_to_element(Products).move_to_element(Products).move_to_element(Web
).click().perform()
        sleep(3)




@pytest.fixture(scope="module")
def m():
    m = MOUSE_OP()
    m.setup()
    yield m
    m.teardown()


@pytest.mark.mouse_hover
```

```
def test_enterDate(m):
    m.mouse_hover()
```

MOUSE RIGHT CLICK

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class MOUSE_RIGHT_CLICK:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://swisnl.github.io/jQuery-
contextMenu/demo.html")
        self.driver.implicitly_wait(10)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def right_click(self):
        btn = self.driver.find_element(By.XPATH,"//span[normalize-
space()='right click me']")
        qt = self.driver.find_element(By.XPATH,"//span[normalize-
space()='Quit']")
        act = ActionChains(self.driver)
        sleep(2)
        act.context_click(btn).perform()
        sleep(3)
        qt.click()
        sleep(3)




@pytest.fixture(scope="module")
def m():
    m = MOUSE_RIGHT_CLICK()
    m.setup()
    yield m
    m.teardown()


@pytest.mark.mouse_right
def test_enterDate(m):
    m.right_click()
```

MOUSE DOUBLE CLICK

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class MOUSE_RIGHT_CLICK:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()

self.driver.get("https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5
_ev_ondblclick3")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def double_click(self):
        self.driver.switch_to.frame("iframeResult")
        ip = self.driver.find_element(By.ID,"field1")
        ip.clear()
        ip.send_keys("lochuuu!!")
        btn = self.driver.find_element(By.XPATH,"//button[normalize-
space()='Copy Text']")
        act = ActionChains(self.driver)
        sleep(2)
        act.double_click(btn).perform()
        sleep(3)




@pytest.fixture(scope="module")
def m():
    m = MOUSE_RIGHT_CLICK()
    m.setup()
    yield m
    m.teardown()


@pytest.mark.mouse_double
def test_enterDate(m):
    m.double_click()
```

MOUSE DRAG AND DROP

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class MOUSE_DRAG_DROP:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("http://www.dhtmlgoodies.com/scripts/drag-drop-
custom/demo-drag-drop-3.html#google_vignette")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def double_click(self):
        src = self.driver.find_element(By.ID,"box6")
        target = self.driver.find_element(By.ID,"box106")
        act = ActionChains(self.driver)
        sleep(2)
        act.drag_and_drop(src,target).perform()
        sleep(3)




@pytest.fixture(scope="module")
def m():
    m = MOUSE_DRAG_DROP()
    m.setup()
    yield m
    m.teardown()


@pytest.mark.mouse_drag_drop
def test_enterDate(m):
    m.double_click()
```

SLIDER

Slider : drag_and_drop_by_offeset(element,xoffset,yoffset)

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class SLIDER:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://www.jqueryscript.net/demo/Price-Range-
Slider-jQuery-UI/")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def drag_slider(self):
        min_slider = self.driver.find_element(By.XPATH,"//*[@id='slider-
range']/span[1]")
        max_slider = self.driver.find_element(By.XPATH,"//*[@id='slider-
range']/span[2]")

        print("location of sliders before moving...")
        print(min_slider.location)    #{'x': 59, 'y': 250
        print(max_slider.location)     #{'x': 412, 'y': 250}
        act = ActionChains(self.driver)
        sleep(2)
        act.drag_and_drop_by_offset(min_slider,100,0).perform()
        sleep(3)
        act.drag_and_drop_by_offset(max_slider, -39, 0).perform()

        print("location of sliders before moving...")
        print(min_slider.location)
        print(max_slider.location)




@pytest.fixture(scope="module")
def sl():
    sl = SLIDER()
    sl.setup()
    yield sl
    sl.teardown()


@pytest.mark.slider
def test_enterDate(sl):
    sl.drag_slider(
```

SCROLLING PAGES

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class SCROLL_PAGE:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()

self.driver.get("https://en.wikipedia.org/wiki/Gallery_of_sovereign_state_f
lags")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()



    def scroll_page_down(self):
        # scroll down page by pixel
        self.driver.execute_script("window.scrollBy(0,3000)", "")
        sleep(3)
        val = self.driver.execute_script("return window.pageYOffset;")
        sleep(3)
        print("number of pixels moved down: ", val)

    def scroll_page_till_requiredElement(self):
        flag = self.driver.find_element(By.XPATH,"//img[@alt='India']")
        self.driver.execute_script("arguments[0].scrollIntoView();",flag)
        sleep(3)
        val = self.driver.execute_script("return window.pageYOffset;")
        sleep(3)
        print("number of pixels moved to required element: ",val)

    def scroll_page_end(self):
        #scroll page till end of the page

self.driver.execute_script("window.scrollBy(0,document.body.scrollHeight)",
"")
        sleep(3)
        val = self.driver.execute_script("return window.pageYOffset;")
        sleep(3)
        print("number of pixels moved to end of page: ",val)

    def scroll_page_top(self):
        #scroll page till end of the page
        self.driver.execute_script("window.scrollBy(0,-
document.body.scrollHeight)","")
        sleep(3)
        val = self.driver.execute_script("return window.pageYOffset;")
        sleep(3)
        print("number of pixels moved to top of the page: ",val)
```

```python
@pytest.fixture(scope="module")
def s():
    s = SCROLL_PAGE()
    s.setup()
    yield s
    s.teardown()


@pytest.mark.scroll_page
def test_scrollPage(s):
    s.scroll_page_down()
    s.scroll_page_till_requiredElement()
    s.scroll_page_end()
    s.scroll_page_top()

# number of pixels moved down:  2125
# number of pixels moved to required element:  4845
# number of pixels moved to end of page:  12821
# number of pixels moved to top of the page:  0
```

KEYBOARD ACTIONS

```
#ctrl+A
#ctrl+C
#tab
#ctrl+V



from selenium import (webdriver)
from selenium.webdriver import ActionChains, Keys
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class KEYBOARD_ACTIONS:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://text-compare.com/")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()



    def check_copy(self):
        # scroll down page by pixel
        ip1 = self.driver.find_element(By.ID,"inputText1")
        ip2 = self.driver.find_element(By.ID, "inputText2")
        sleep(3)
        ip1.send_keys("Lochani")
        act = ActionChains(self.driver)
        sleep(2)
        #ip1 --> ctrl+A select the text

act.key_down(Keys.CONTROL).send_keys("a").key_up(Keys.CONTROL).perform()
        sleep(2)
        #ip1 --> ctrl+C copy text

act.key_down(Keys.CONTROL).send_keys("c").key_up(Keys.CONTROL).perform()
        sleep(2)
        #press tab key to navigate top ip2
        act.send_keys(Keys.TAB).perform()
        sleep(2)
        #ip2 --> ctrl+V paste thetext

act.key_down(Keys.CONTROL).send_keys("v").key_up(Keys.CONTROL).perform()
        sleep(3)



@pytest.fixture(scope="module")
def Key():
```

```
    Key = KEYBOARD_ACTIONS()
    Key.setup()
    yield Key
    Key.teardown()


@pytest.mark.keyboard
def test_scrollPage(Key):
    Key.check_copy()
```

CAPTURE SCREENSHOTS

- save_screenshot(file_location)
- get_screenshot_as_file(file_location)
- get_screenshot_as_png(file_location) #saves image in binary format
- get_screenshot_as_base64(file_location) #saves image in binary format

```
import os
from selenium import (webdriver)
from selenium.webdriver import ActionChains, Keys
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class CAPTURE_SS:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://demo.nopcommerce.com/")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()



    def capture_ss(self):
        self.driver.save_screenshot(os.getcwd()+r"\homepage.png")
        self.driver.get_screenshot_as_file(os.getcwd()+r"\file.png")
        #self.driver.get_screenshot_as_png()
#self.driver.get_screenshot_as_base64() #saves in binary format
        sleep(3)



@pytest.fixture(scope="module")
def ss():
    ss = CAPTURE_SS()
    ss.setup()
    yield ss
    ss.teardown()


@pytest.mark.screenshot
def test_scrollPage(ss):
    ss.capture_ss()
```

TABS AND WINDOWS

Switching tabs

```
from selenium import (webdriver)
from selenium.webdriver import Keys
from selenium.webdriver.common.by import By
import pytest
from time import sleep


class TABS:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://demo.nopcommerce.com/")
        self.driver.implicitly_wait(5)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def new_tab(self):
        #opens new tab but do not switch to that tab
        reglink = Keys.CONTROL + Keys.RETURN
        self.driver.find_element(By.LINK_TEXT,
"Register").send_keys(reglink)
        sleep(3)

    def open_tab(self):
        self.driver.get("https://www.opencart.com/")
        self.driver.switch_to.new_window('tab')
        self.driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
@pytest.fixture(scope="module")
def st():
    st = TABS()
    st.setup()
    yield st
    st.teardown()


@pytest.mark.open_tab
def test_openTabs(st):
    st.new_tab()
    st.open_tab()
```

switching windows

```
from selenium import webdriver
import pytest
from time import sleep


class WINDOWS:
```

```
    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def switch_windows(self):
        self.driver.get("https://www.opencart.com/")
        self.driver.switch_to.new_window('window')
        sleep(2)
        self.driver.get("https://opensource-
demo.orangehrmlive.com/web/index.php/auth/login")
        sleep(2)
@pytest.fixture(scope="module")
def w():
    w = WINDOWS()
    w.setup()
    yield w
    w.teardown()


@pytest.mark.open_window
def test_switchWindows(w):
    w.switch_windows()
```

HANDLING COOKIES

- **get_cookies()**
- **add_cookie()**
- **delete_cookie()**
- **delete_all_cookies()**

```
from selenium import webdriver
import pytest
from time import sleep


class COOKIES:

    def __init__(self):
        self.driver = None

    def setup(self):
        self.driver = webdriver.Chrome()
        self.driver.get("https://demo.nopcommerce.com/")
        self.driver.implicitly_wait(4)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

    def get_cookies(self):
        cookies = self.driver.get_cookies()
        sleep(2)

        for c in cookies:
            print(c)

        #retrieve the name:value of the cookies
        print("names of the cookies...")
        for c in cookies:
            print(c.get('name'),":",c.get('value'))

        return len(cookies)


    def add_cookie(self):
        self.driver.add_cookie({"name":"MyCookie","value":"123456"})
        cookies = self.driver.get_cookies()
        return len(cookies)

    def del_cookie(self):
        self.driver.delete_cookie("MyCookie")
        cookies = self.driver.get_cookies()
        return len(cookies)

    def del_all_cookies(self):
        self.driver.delete_all_cookies()
        cookies = self.driver.get_cookies()
        return len(cookies)
@pytest.fixture(scope="module")
```

```python
def c():
    c = COOKIES()
    c.setup()
    yield c
    c.teardown()


@pytest.mark.cookies
def test_cookiesNum(c):
    print(c.get_cookies())

@pytest.mark.cookies
def test_addCookie(c):
    print(c.add_cookie())

@pytest.mark.cookies
def test_delCookie(c):
    print(c.del_cookie())

@pytest.mark.cookies
def test_delallCookie(c):
    print(c.del_all_cookies())
```

HEADLESS MODE

```python
from selenium import webdriver
from time import sleep
import pytest
from selenium.webdriver.chrome.options import Options


class HEADLESS:

    def __init__(self):
        self.driver = None

    def setup(self):
        ops = Options()
        ops.add_argument("--headless")
        self.driver = webdriver.Chrome(options=ops)
        self.driver.get("https://demo.nopcommerce.com/")
        self.driver.implicitly_wait(4)
        print(self.driver.title)
        print(self.driver.current_url)

    def teardown(self):
        if self.driver is not None:
            self.driver.quit()

@pytest.mark.headless
def test_headlessMode():
    h = HEADLESS()
    h.setup()
    h.teardown()
```