**SELENIUM**

**EMP ID: 43317**

**Prepared By: S.Lochani Vilehya**

**INDEX:**

1. **Selenium WebDriver**
2. **Locators**

* **Id**
* **Name**
* **Link\_text , Partial\_LinkText**
* **Class name**
* **Tag name**

1. **CSS Selectors**

* **Tag Id**
* **Tag Class**
* **Tag attribute**
* **Tag class attribute**

1. **XPATH**

* **Absolute xpath**
* **Relative xpath**

1. **XPATH AXES**
2. **BASIC COMMANDS IN SELENIUM**

* **Application commands**
* **Conditional commands**
* **Browser commands**
* **Navigational commands**
* **Wait commands**

1. **WEB ELEMENTS**

* **Checkbox**
* **Links**
* **Internal**
* **External**
* **Broken ink**
* **Dropdown list**
* **Alerts**
* **Authentication popup**
* **Frames/iframes**
* **Tables**
* **Datepicker**

1. **MOUSE OPERATIONS**

* **Mouse hover**
* **Right click**
* **Double click**
* **Drag and drop**
* **SLIDER**
* **SCROLLING PAGES**

1. **TABS AND WINDOWS**

* **Switching tabs**
* **Switching windows**

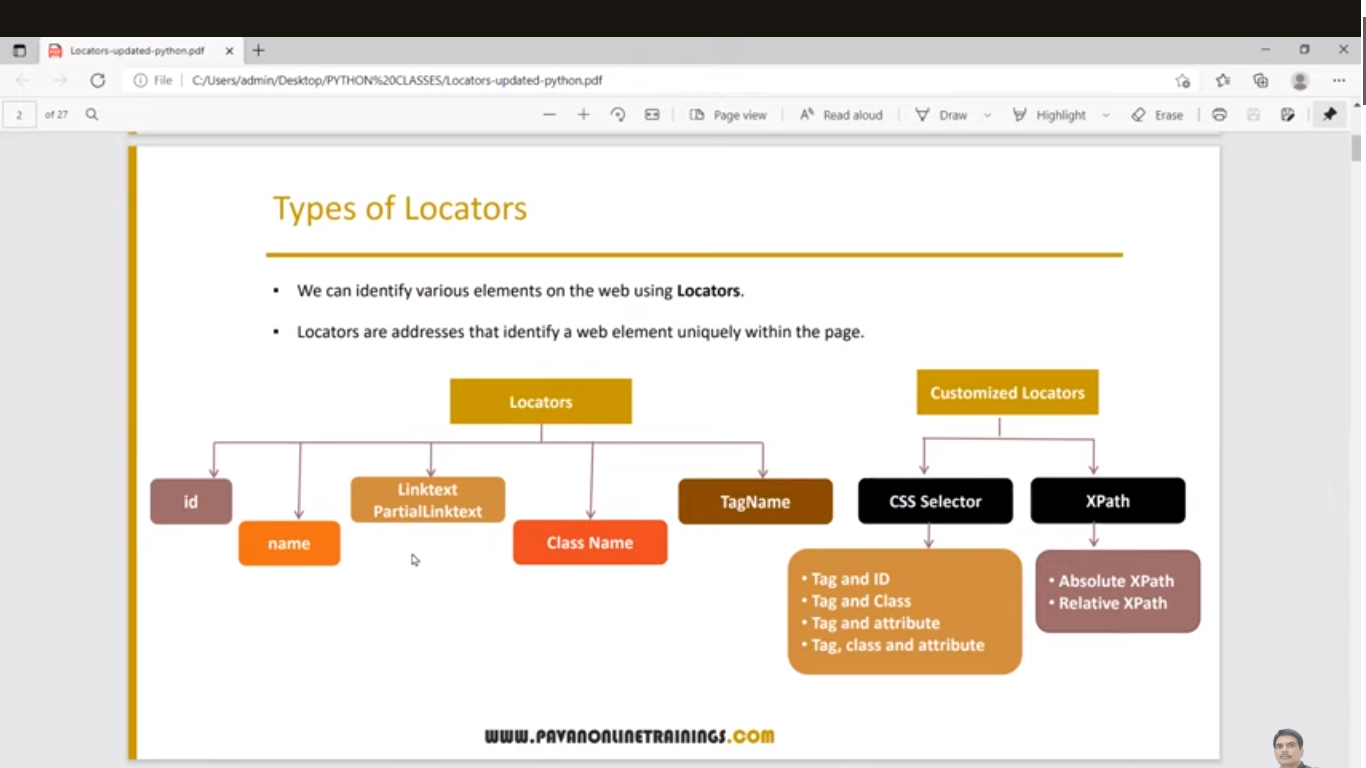
1. **HANDLING COOKIES**
2. **HEADLESS MODE**

**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() |

1. 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()  
  
  
@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**

1. It is defined as XML path
2. It is a syntax or language for finding any element on the web page using XML path expression
3. It is used to find the location of any element on a webpage using HTML DOM structure
4. It is an address of the element
5. 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

1. 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:**

1. And
2. Or
3. Contains()
4. Startswith()
5. 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’)] |

**Tex**t 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"  
  
  
@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.

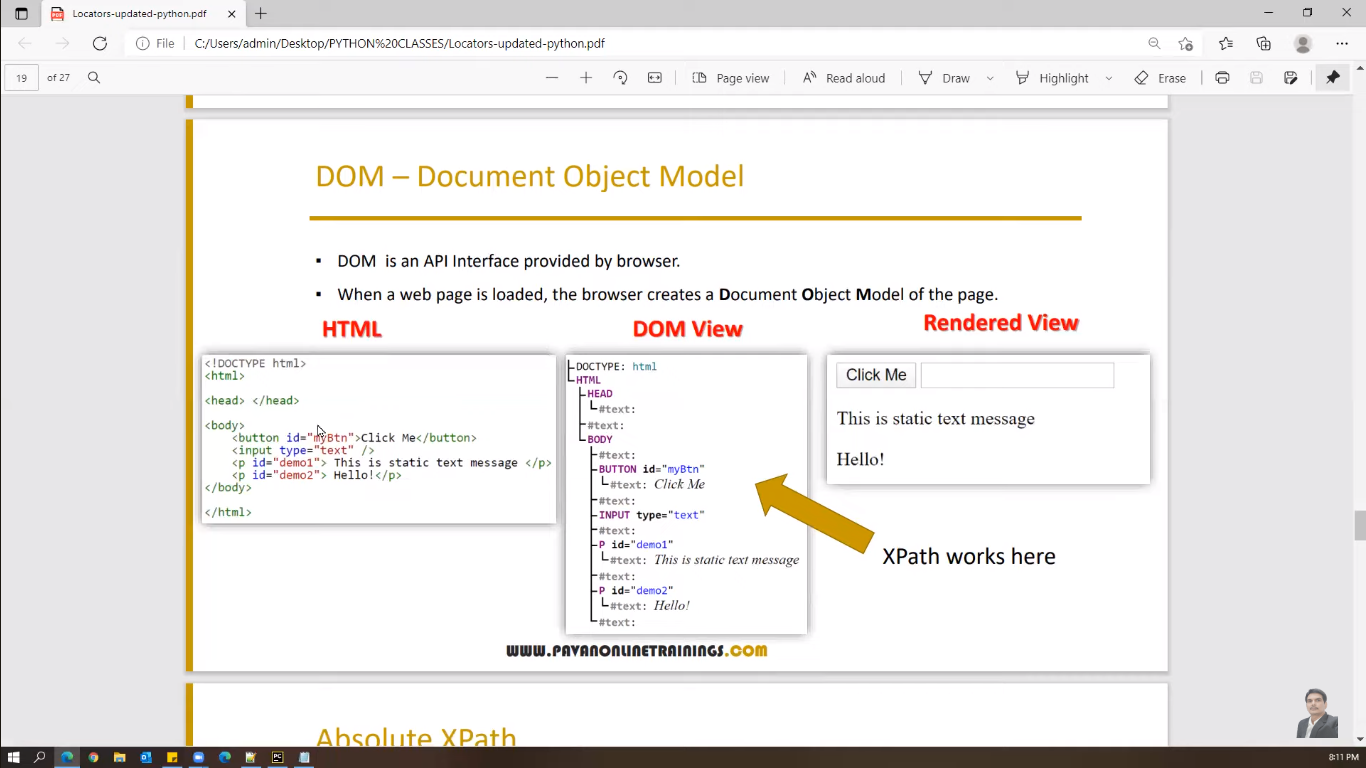
Here are some common XPath axes used in Selenium:

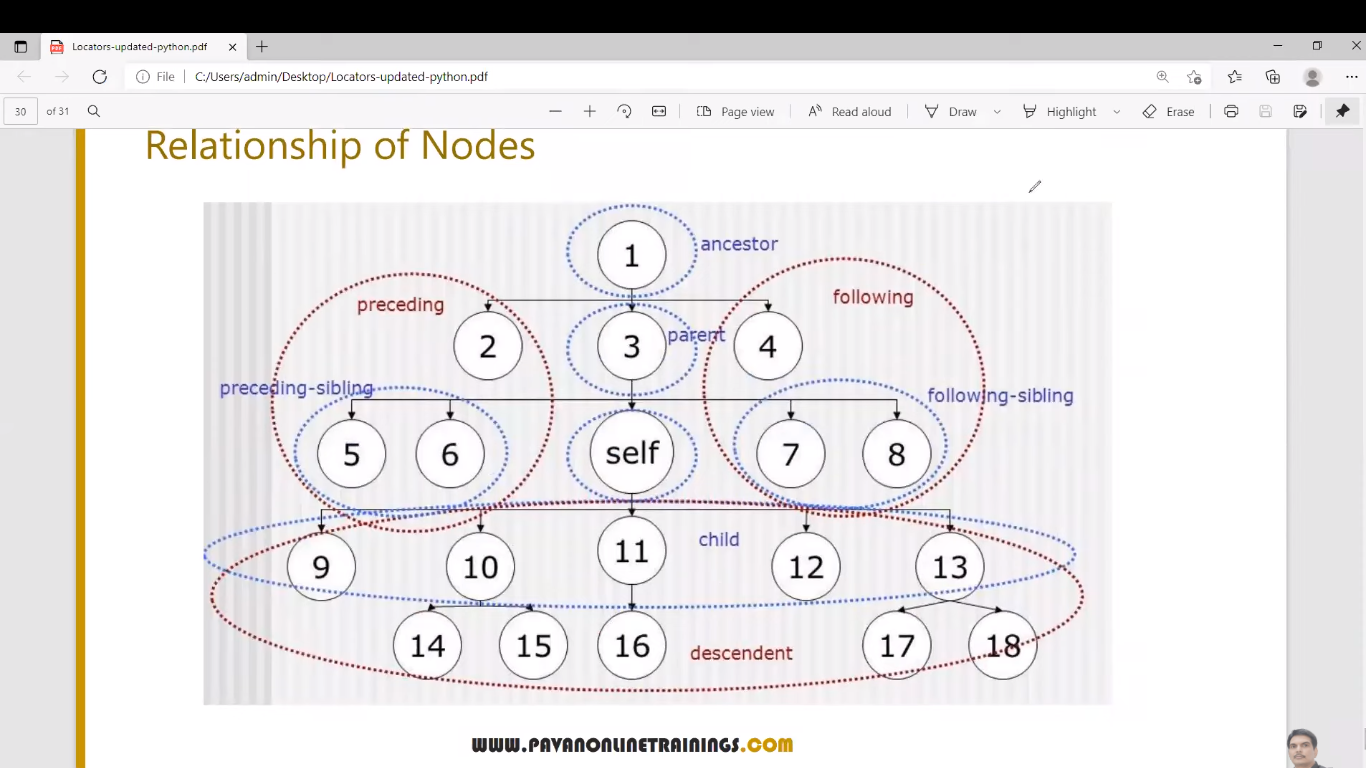
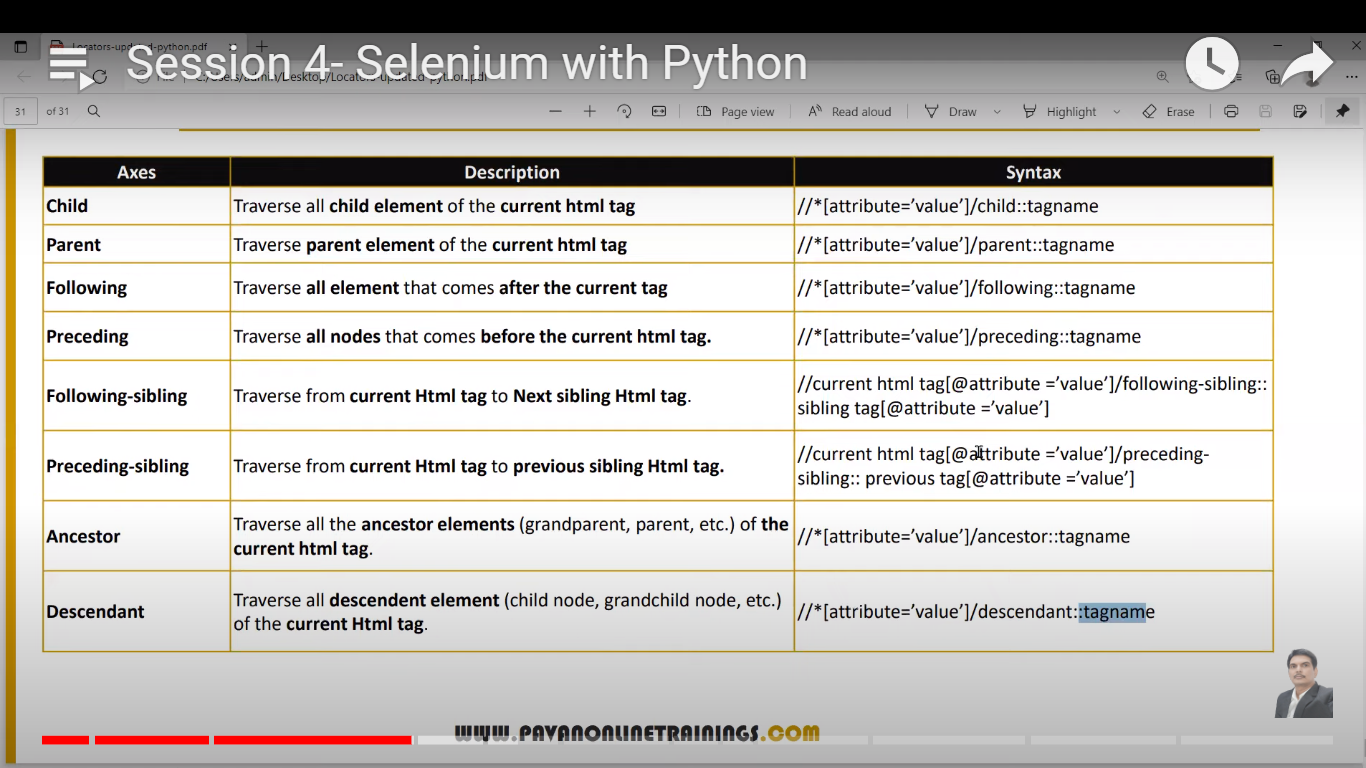
1. **Parent Axis (parent::)**: Selects the parent of the current node. Example:

**//div[@class='parent']/parent::div**

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





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()

1. **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

1. **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"

1. **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()

1. **Authentication popup**

**Link :** [**https://the-internet.herokuapp.com/basic\_auth**](https://the-internet.herokuapp.com/basic_auth)

**Syntax :** [**https://username:password@test.com**](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("welcome")  
 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\_flags")  
 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)  
@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")  
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**

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()