38. Handling or Performing Mouse Operations

- ➔ The `**ActionChains**` class handles **mouse actions** using its methods.
- ➔ Each of the methods is executed by calling `**.perform()**` at the end of the action chain to trigger the actions.

Mouse Actions in Selenium

- ➔ **click()** → Performs a single mouse click at the current mouse position.
  - ◆ **Syntax:** actions.click(WebElement).perform()
- ➔ **click_and_hold()** → Clicks (without releasing) at the current mouse location or on the specified element.
  - ◆ **Syntax:** actions.click_and_hold(WebElement).perform()
- ➔ **move_to_element()** → Moves the mouse pointer to the middle of the specified element.
  - ◆ **Syntax:** actions.move_to_element(WebElement target).perform()
- ➔ **double_click()** → Performs a double-click at the current mouse location or on a specified element.
  - ◆ **Syntax:** actions.double_click(WebElement).perform()
- ➔ **context_click()** → Performs a right-click on the current mouse location or a specified element.
  - ◆ **Syntax:** actions.context_click(WebElement).perform()
- ➔ **drag_and_drop()** → Clicks and holds an element, then moves it to a target location, and releases it.
  - ◆ **Syntax:** actions.drag_and_drop(WebElement source, WebElement target).perform()
- ➔ **drag_and_drop_by_offset()** → Clicks and holds an element, moves it by an offset (x, y) from its original position, and releases it.
  - ◆ **Syntax:** actions.drag_and_drop_by_offset(WebElement source, xOffset, yOffset).perform()

MouseHoverDemo.py

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
options = webdriver.ChromeOptions()
options.add_experimental_option("detach",True)
options.add_argument("--start-argument")
driver = webdriver.Chrome(options=options)
driver.get("https://www.ebay.com/")
driver.implicitly_wait(10)
```

Locate the elements

```
ele = driver.find_element(By.XPATH,'//a[text()="Electronics"]//parent::li')
lnk_app= driver.find_element(By.XPATH,'//a[text()="Apple"]')
```

mouse hover

```
act = ActionChains(driver)
act.move_to_element(ele).move_to_element(lnk_app).click().perform()
driver.quit()
```

DoubleClickDemo.py

```
from selenium import webdriver
from selenium.webdriver import ActionChains
```

```
from selenium.webdriver.common.by import By
options = webdriver.ChromeOptions()
options.add_experimental_option("detach",True)
options.add_argument("--start-maximized")
driver = webdriver.Chrome(options=options)
driver.get("https://testautomationpractice.blogspot.com/")
```

Locate elements

```
box1 = driver.find_element(By.XPATH, "//input[@id='field1']")
box2 = driver.find_element(By.XPATH, "//input[@id='field2']")
copy_button = driver.find_element(By.XPATH, "//button[normalize-space()='Copy Text']")
```

Interact with elements

```
box1.clear()
box1.send_keys("Welcome")
```

Perform double click

```
action = ActionChains(driver)
action.double_click(copy_button).perform()
```

Verify the copied text

```
text = box2.get_attribute("value")
if text == "Welcome":
  print("Double click is successful & Text is copied")
else:
  print("Text is not matched..")
driver.quit()
```

RightClickDemo.py

```
import time
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
opt = webdriver.ChromeOptions()
opt.add_experimental_option("detach",True)
opt.add_argument("--start-maximized")
driver = webdriver.Chrome(options = opt)
driver.implicitly_wait(10)
driver.get("http://swisnl.github.io/jQuery-contextMenu/demo.html")
```

Locate the button

```
button = driver.find_element(By.XPATH, "//span[normalize-space()='right click me']")
```

Perform right-click on the button

```
action = ActionChains(driver)
action.context_click(button).perform()
```

Click on the 'Copy' option in the context menu

```
driver.find_element(By.XPATH, "//span[normalize-space()='Copy']").click()
```

```
Wait for the alert to appear
time.sleep(5)
Switch to the alert box and accept it
alert = driver.switch_to.alert
alert.accept()
driver.quit()
```

DragAndDropDemo.py

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
opt = webdriver.ChromeOptions()
opt.add_experimental_option("detach",True)
opt.add_argument("--start-maximized")
driver = webdriver.Chrome(options = opt)
driver.implicitly_wait(10)
driver.get("https://testautomationpractice.blogspot.com/")
Locate the source and target elements
source_element = driver.find_element(By.XPATH, "//div[@id='draggable']")
target_element = driver.find_element(By.XPATH, "//div[@id='droppable']")
Perform the drag and drop
action = ActionChains(driver)
action.drag_and_drop(source_element, target_element).perform()
Alternatively, you can use click_and_hold and move_to_element
action.click_and_hold(source_element).move_to_element(target_element).release().perform()
driver.quit()
```

SliderDemo.py

```
from selenium import webdriver
from selenium.webdriver import ActionChains
from selenium.webdriver.common.by import By
opt = webdriver.ChromeOptions()
opt.add_experimental_option("detach",True)
opt.add_argument("--start-maximized")
driver = webdriver.Chrome(options = opt)
driver.implicitly_wait(10)
driver.get("https://testautomationpractice.blogspot.com/")
Locate the min and max slider elements
min_slider = driver.find_element(By.XPATH, "//div[@id='HTML7']//span[1]")
max_slider = driver.find_element(By.XPATH, "//div[@id='HTML7']//span[2]")
Locate slider elements
min_slider = driver.find_element(By.XPATH,"//div[@id='slider-range']/span[1]")
max_slider = driver.find_element(By.XPATH,"//div[@id='slider-range']/span[2]")
```

Before moving

```
print("Before Moving.....")
print(f"Before moving Min slider Location: {min_slider.location['x']}")
print(f"Before moving Max slider Location: {max_slider.location['x']}")
```

Initialize ActionChains

```
action = ActionChains(driver)
```

Drag and Drop by offset

```
actions.drag_and_drop_by_offset(min_slider,20,0).perform()
actions.drag_and_drop_by_offset(max_slider,-33,0).perform()
```

After moving

```
print("After Moving.....")
print(f"After moving Min slider Location: {min_slider.location['x']}")
print(f"After moving Max slider Location: {max_slider.location['x']}")
driver.quit()
```

Capture Element Location and Size Methods

- ➔ **location** ⇒ element.location
  - ◆ Retrieves the x and y coordinates of the element as a dictionary.
- ➔ **location['x']** ⇒ element.location['x']
  - ◆ Retrieves the x-coordinate (horizontal position) of the top-left corner of the web element.
- ➔ **location['y']** ⇒ element.location['y']
  - ◆ Retrieves the y-coordinate (vertical position) of the top-left corner of the web element.
- ➔ **size** ⇒ element.size
  - ◆ Retrieves the width and height of the web element as a dictionary.
- ➔ **size['height']** ⇒ element.size['height']
  - ◆ Retrieves the height of the web element.
- ➔ **size['width']** ⇒ element.size['width']
  - ◆ Retrieves the width of the web element.

SizeAndLocationOfWebElement.py

```
from selenium import webdriver
from selenium.webdriver.common.by import By
opt = webdriver.ChromeOptions()
opt.add_experimental_option("detach",True)
opt.add_argument("--start-maximized")
driver = webdriver.Chrome(options = opt)
driver.implicitly_wait(10)
driver.get("http://www.automationpractice.pl/index.php")
```

Locate the logo element

```
logo = driver.find_element(By.XPATH, "//img[@alt='My Shop']")
```

Get the location (X, Y)

```
location = logo.location
print(f"Location (Point): {location}")
```

```
print(f"X axis: {location['x']}")
print(f"Y axis: {location['y']}")
```

Get the size (width, height)

```
size = logo.size
print(f"Size (Dimension): {size}")
print(f"Height: {size['height']}")
print(f"Width: {size['width']}")
```

Another way to capture height and width of the element

```
height = logo.get_attribute("height")
width = logo.get_attribute("width")
print(f"Height (from attribute): {height}")
print(f"Width (from attribute): {width}")
driver.quit()
```

Note

- ➔ We can handle **send_keys()** and **click actions** using Actions class methods.
- ➔ Passing text to input box
  - ◆ Ex: act.send_keys_to_element(WebElement, "value").perform()
- ➔ Clicking on button/radio button/checkbox/link
  - ◆ Ex: act.click(WebElement).perform()

ClickAndSendKeysFromActionsClassDemo.py

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver import ActionChains
opt = webdriver.ChromeOptions()
opt.add_experimental_option("detach",True)
opt.add_argument("--start-maximized")
driver = webdriver.Chrome(options = opt)
driver.implicitly_wait(10)
driver.get("https://testautomationpractice.blogspot.com/")
```

Actions object

```
act = ActionChains(driver)
```

1. Inputbox ⇒ using send_keys_to_element() method from Actions class

```
name = driver.find_element(By.XPATH, "//input[@id='name']")
act.send_keys_to_element(name,"John Kenedy").perform()  # Correct approach
act.click(name).send_keys("T-shirts").perform() # Correct approach
act.move_to_element(name).click().send_keys("T-Shirts").perform() # Correct approach
```

2. Button ⇒ using click() method from Actions class

```
button = driver.find_element(By.XPATH, "//button[@id='alertBtn']")
act.move_to_element(button).click().perform()  # Correct approach
act.click(button).perform()  # Correct approach
driver.quit()
```