# **ALL SELENIUM**



# Python Selenium all mouse actions using ActionChains

The Blog To Learn Selenium and Test Automation

# Python Selenium all mouse actions using ActionChains

<u>admin</u> <u>Python Selenium</u> <u>Manuary 10, 2019 | 0 0</u>

# Page Contents



- 1. What is ActionChains?
- 2. All available ActionChains method
  - 2.1. click(on\_element=None)
  - 2.2. click\_and\_hold(on\_element=None)
  - 2.3. context\_click(on\_element=None)
  - 2.4. double\_click(on\_element=None)
  - 2.5. drag\_and\_drop(source, target)
  - 2.6. drag\_and\_drop\_by\_offset(source, xoffset, yoffset)
  - 2.7. key\_down(value, element=None)
  - 2.8. key\_up(value, element=None)
  - 2.9. move\_by\_offset(xoffset, yoffset)
  - 2.10. move\_to\_element(to\_element)
  - 2.11. move\_to\_element\_with\_offset(to\_element, xoffset, yoffset)
  - 2.12. pause(seconds)
  - 2.13. perform()
  - 2.14. release(on\_element=None)
  - 2.15. reset\_actions()
  - 2.16. send\_keys(keys\_to\_send)
  - 2.17. send\_keys\_to\_element(element, keys\_to\_send)
- 3. Common Scenarios
  - 3.1. Open link in new Tab
  - 3.2. Handle Mouse overs
- 4. Related Posts





in





In this article, we are going to see all mouse actions available and supported by Python Selenium. WebDriver as ActionChains object.

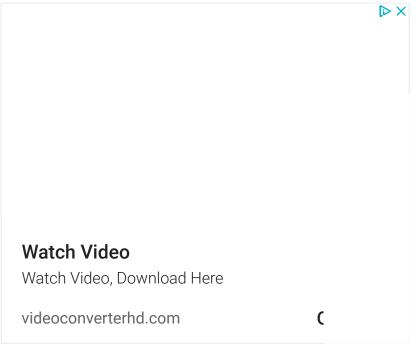
As discussed in our <u>previous article</u>, there might be many scenarios where we need to use mouse actions in automation testing. Some common uses case are given below.

- Mouse over on a menu item to bring hidden sub-menu
- Right click on a link or an element and do some operations
- <u>Drag and drop</u> items
- Double click on an item

# What is ActionChains?

ActionChains are a ways provided by Selenium to automate low level interactions with website such as mouse movements, mouse button actions, key press, and context menu(right click menu) interactions. These special methods are useful for doing more complex actions like mouse over and drag and drop that are not possible by direct webdriver actions.

Using ActionChains object, we call methods that perform specific actions sequentially one by one. Actions called are pushed into a queue. When perform() method is called, the events are fired in the order they are queued up.



The ActionChains implementation is available in below path.



The WebDriver instance should be passed while creating ActionChains object.

ActionChains can be used in a chain pattern, adding all the methods in the same line.

Actions can be added to one by one to queue, then performed.

```
python

1 menu = driver.find_element_by_id("menu")
2 submenu = driver.find_element_by_id("submenu1")
3
4 actions = ActionChains(driver)
5 actions.move_to_element(menu)
6 actions.click(submenu)
7 actions.perform()
```

In both ways, the actions are executed or performed one after another in the order they are added.

# All available ActionChains method

#### click(on\_element=None)

Left clicks on a given element.

#### **Parameters**

*on\_element*: The element to click on. If no element passed, clicks left mouse button on current mouse position.

# click\_and\_hold(on\_element=None)

Click and holds down the left mouse button on an element.

#### **Parameters**

on\_element. The element to mouse down. If no element passed, clicks on current mouse position.











Performs a context-click (right click) on an element.

#### **Parameters**

*on\_element*: The element to context-click or right click on. If no element passed, right clicks on current mouse position.

# double\_click(on\_element=None)

Double-clicks on an element.

#### **Parameters**

*on\_element*: The element to double-click on. If no element passed, double clicks on current mouse position.

# drag\_and\_drop(source, target)

Holds down the left mouse button on the source element, then moves to the target element and releases the mouse button.

#### **Parameters**

*source*: The element to mouse down or the element to be dragged *target*. The element to mouse up or target element to be dragged into

# **Example**

```
python

from_element = driver.find_element_by_id("source")

to_element = driver.find_element_by_id("target")

ActionChains(driver)

drag_and_drop(from_element, to_element)

perform()
```

# drag\_and\_drop\_by\_offset(source, xoffset, yoffset)

Holds down the left mouse button on the source element, then moves to the target offset and releases the mouse button.

#### **Parameters**

source: The element to mouse down.

xoffset: X offset to move to.

yoffset: Y offset to move to.











Sends a key press only, without releasing it. This is used mainly with modifier keys like Control, Alt and shift.

#### **Parameters**

*value*: The modifier key to send. Keys class defines all values. *element*: The element to send keys. If None, sends a key to current focused element.

Example, below script presses ctrl+c



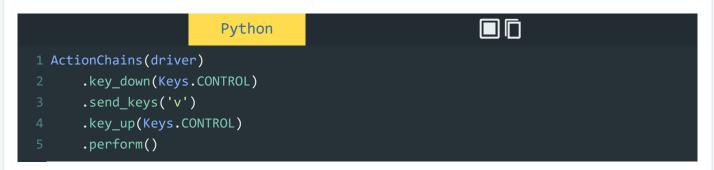
# key\_up(value, element=None)

Releases a modifier key.

#### **Parameters**

*value*: The modifier key to send. Keys class defines all the values. *element*: The element to send keys. If no element passed, sends a key to current focused element.

Example, below script presses ctrl+v



# move\_by\_offset(xoffset, yoffset)

Moving the mouse to an offset from current mouse position.

#### **Parameters**

*xoffset*: X offset to move to, as a positive or negative integer. *yoffset*: Y offset to move to, as a positive or negative integer.











Moving the mouse to the middle of an element. This action helps us to deal with dropdown menu that appears when the <u>user moves the mouse over an element</u> or when the <u>user clicks</u> on an element.

#### **Parameters**

to\_element: The WebElement to move to.

#### **Example**

```
python

1 menu = driver.find_element_by_id("allmenu")

2 ActionChains(driver)

3    .move_to_element(menu)

4    .perform()

5 # wait until sub menu appears on the screen

6 WebDriverWait(self.webdriver, 5)

7    .until(EC.visibility_of_element_located((By.ID, "home")))

8 home_menu = driver.find_element_by_id("home")

9 home_menu.click()
```

# move\_to\_element\_with\_offset(to\_element, xoffset, yoffset)

Move the mouse by an offset of the specified element. Offsets are relative to the top-left corner of the element.

#### **Parameters**

to\_element: The WebElement to move to.xoffset: X offset to move to.yoffset: Y offset to move to.

# pause(seconds)

Pause all actions for the specified duration (in seconds).

# perform()

Performs all stored actions.

# release(on\_element=None)

Releasing a held mouse button on an element.

# **Parameters**

on element. The element to mouse up. If None releases on current mouse position





in





# reset\_actions()

To clear all actions already stored locally and/or on the remote end.

# send\_keys(keys\_to\_send)

Sends keys to current focused element.

#### **Parameters**

keys\_to\_send: The keys to send.

# send\_keys\_to\_element(element, keys\_to\_send)

Sends keys to an element.

#### **Parameters**

*element*: The element to send keys. *keys\_to\_send*: The keys to send.

For both send\_keys and send\_keys\_to\_element method arguments, Modifier keys constants can be found in the 'Keys' class.

# Common Scenarios

# Open link in new Tab

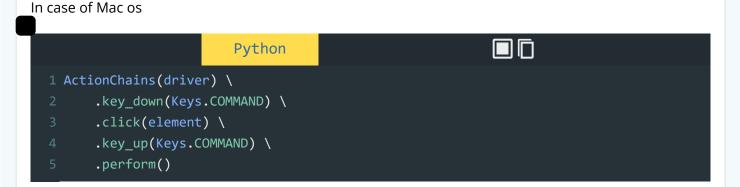
User has to press Control key and click on the link to open it in new browser tab. Same operation can be scripted in Selenium WebDriver as follows











User has to press Shift key and click on the link to open it in new browser window. Same operation can be scripted in Selenium WebDriver as follows

```
Python

ActionChains(driver) \

key_down(Keys.SHIFT) \

click(element) \

key_up(Keys.SHIFT) \

perform()
```

#### Handle Mouse overs

Use pause() method if you like to wait before each operations. For example, if it takes few seconds to display menu items.

Hope this article helps you. Please let us know your thoughts in comments section.





in





