selenium.webdriver.common.action_chains

selenium.dev/selenium/docs/api/py/webdriver/selenium.webdriver.common.action_chains.html

ActionChains(driver[, duration])	ActionChains are a way to automate low level interactions such as mouse movements, mouse button actions, key press, and context menu interactions.

class `selenium.webdriver.common.action_chains.` `ActionChains` **(*driver*, *duration=250*)[source]**

ActionChains are a way to automate low level interactions such as mouse movements, mouse button actions, key press, and context menu interactions. This is useful for doing more complex actions like hover over and drag and drop.

Generate user actions.

When you call methods for actions on the ActionChains object, the actions are stored in a queue in the ActionChains object. When you call perform(), the events are fired in the order they are queued up.

ActionChains can be used in a chain pattern:

```
menu = driver.find_element(By.CSS_SELECTOR, ".nav")
hidden_submenu = driver.find_element(By.CSS_SELECTOR, ".nav #submenu1")

ActionChains(driver).move_to_element(menu).click(hidden_submenu).perform()
```

Or actions can be queued up one by one, then performed.:

```
menu = driver.find_element(By.CSS_SELECTOR, ".nav")
hidden_submenu = driver.find_element(By.CSS_SELECTOR, ".nav #submenu1")

actions = ActionChains(driver)
actions.move_to_element(menu)
actions.click(hidden_submenu)
actions.perform()
```

Either way, the actions are performed in the order they are called, one after another.

Creates a new ActionChains.

Args:

- driver: The WebDriver instance which performs user actions.
- duration: override the default 250 msecs of DEFAULT_MOVE_DURATION in PointerInput

`click` **(*on_element=None*)[source]**

Clicks an element.

on_element: The element to click. If None, clicks on current mouse position.

Args:

`click_and_hold` ***(on_element=None)*** **[source]**

Holds down the left mouse button on an element.

on_element: The element to mouse down. If None, clicks on current mouse position.

Args:

`context_click` ***(on_element=None)*** **[source]**

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

on_element: The element to context-click. If None, clicks on current mouse position.

Args:

`double_click` ***(on_element=None)*** **[source]**

Double-clicks an element.

on_element: The element to double-click. If None, clicks on current mouse position.

Args:

`drag_and_drop` ***(source, target)*** **[source]**

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

- source: The element to mouse down.
- target: The element to mouse up.

Args:

`drag_and_drop_by_offset` ***(source, xoffset, yoffset)*** **[source]**

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

- source: The element to mouse down.
- xoffset: X offset to move to.
- yoffset: Y offset to move to.

Args:

`key_down` ***(value, element=None)*** **[source]**

Sends a key press only, without releasing it.
Should only be used with modifier keys (Control, Alt and Shift).

- value: The modifier key to send. Values are defined in *Keys* class.
- element: The element to send keys. If None, sends a key to current focused element.

Args:

Example, pressing ctrl+c:

```
ActionChains(driver).key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).per
```

`key_up` ***(value, element=None)*** **[source]**

Releases a modifier key.

- value: The modifier key to send. Values are defined in Keys class.
- element: The element to send keys. If None, sends a key to current focused element.

Args:

Example, pressing ctrl+c:

```
ActionChains(driver).key_down(Keys.CONTROL).send_keys('c').key_up(Keys.CONTROL).per
```

`move_by_offset` ***(xoffset, yoffset)*** **[source]**

Moving the mouse to an offset from current mouse position.

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

Args:

`move_to_element` ***(to_element)*** **[source]**

Moving the mouse to the middle of an element.

to_element: The WebElement to move to.

Args:

`move_to_element_with_offset` ***(to_element, xoffset, yoffset)*** **[source]**

Move the mouse by an offset of the specified element.

Offsets are relative to the top-left corner of the element.

- to_element: The WebElement to move to.
- xoffset: X offset to move to.
- yoffset: Y offset to move to.

Args:

`pause` ***(seconds)*** **[source]**

Pause all inputs for the specified duration in seconds

`perform`()[source]
Performs all stored actions.

`release`(*on_element=None*)[source]
Releasing a held mouse button on an element.

Args:
on_element: The element to mouse up. If None, releases on current mouse position.

`reset_actions`()[source]
Clears actions that are already stored locally and on the remote end

`scroll`(*x: int*, *y: int*, *delta_x: int*, *delta_y: int*, *duration: int = 0*, *origin: str = 'viewport'*)[source]
Sends wheel scroll information to the browser to be processed.

Args:
- x: starting X coordinate
- y: starting Y coordinate
- delta_x: the distance the mouse will scroll on the x axis
- delta_y: the distance the mouse will scroll on the y axis

`scroll_by_amount`(*delta_x: int*, *delta_y: int*)[source]
Scrolls by provided amounts with the origin in the top left corner of the viewport.

Args:
- delta_x: Distance along X axis to scroll using the wheel. A negative value scrolls left.
- delta_y: Distance along Y axis to scroll using the wheel. A negative value scrolls up.

`scroll_from_origin`(*scroll_origin: selenium.webdriver.common.actions.wheel_input.ScrollOrigin*, *delta_x: int*, *delta_y: int*)[source]
Scrolls by provided amount based on a provided origin. The scroll origin is either the center of an element or the upper left of the viewport plus any offsets. If the origin is an element, and the element is not in the viewport, the bottom of the element will first be scrolled to the bottom of the viewport.

- origin: Where scroll originates (viewport or element center) plus provided offsets.
- delta_x: Distance along X axis to scroll using the wheel. A negative value scrolls left.
- delta_y: Distance along Y axis to scroll using the wheel. A negative value scrolls up.

Raises: If the origin with offset is outside the viewport. - MoveTargetOutOfBoundsException - If the origin with offset is outside the viewport.

Args:

`scroll_to_element` ***(element: selenium.webdriver.remote.webelement.WebElement)*** **[source]**

If the element is outside the viewport, scrolls the bottom of the element to the bottom of the viewport.

Args: element: Which element to scroll into the viewport.

`send_keys` ***(*keys_to_send)*** **[source]**

Sends keys to current focused element.

Args: keys_to_send: The keys to send. Modifier keys constants can be found in the 'Keys' class.

`send_keys_to_element` ***(element, *keys_to_send)*** **[source]**

Sends keys to an element.

Args:

- element: The element to send keys.
- keys_to_send: The keys to send. Modifier keys constants can be found in the 'Keys' class.