ProgrammerSought

search

Python+Selenium automated test 4. ActionChains mouse, keyboard events

tags: ActionChains selenium send_keys python

table of Contents

TOP

1. perform # perform all actions in the chain

2. reset_actions #Clear the action stored at the far end

3.click #left mouse click

4.click_and_hold #Click the left mouse button and do not release

5.context_click #

6.double_click #Double left mouse button

7. drag_and_drop(source,target) # Drag and drop to an element and release

8.drag_and_drop_by_offset #Drop to a coordinate and release

9. move_by_offset #Mouse moves to a coordinate

10. move_to_element #Mouse moves to an element

11. move_to_element_with_offset # Move to a position from an element (upper left corner)

12. pause

13. release # Release the left mouse button on an element

Three keyboard operation

1. Introducing the package:

2. key_down #A keyboard key is pressed

3. key_up # release a key

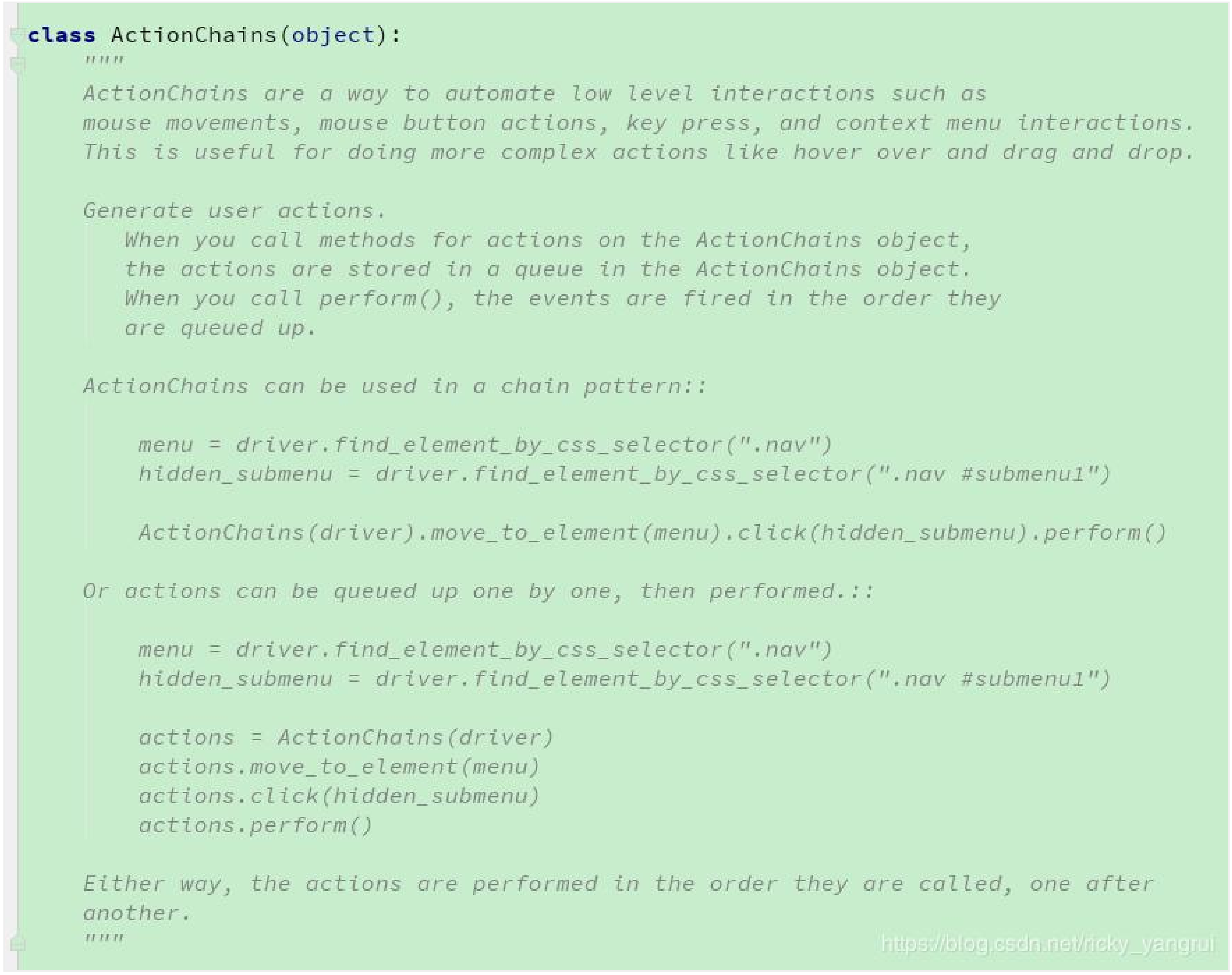
4. send_keys #send some values to the current focus element

5 send_keys_to_element #Send some values to the specified element

6. keys

An ActionChains introduction

Source code:



```
class ActionChains(object):
    """
    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.
    """
```

```
ActionChains is a low-level automated interaction method such as mouse movement, mous

 Generate user actions.
 When calling an action method on an ActionChains object,
 The operation is stored in the queue of the ActionChains object.
 When you call perform(), the events will fire in their order
        Wait in line.

 ActionChains can be used for chaining::

        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 the operation can be queued one by one and then executed. ::

        menu = driver.find_element_by_css_selector(".nav")
```

TOP

```
        hidden_submenu = driver.find_element_by_css_selector(".nav #submenu1")

        actions.move_to_element(menu)
        actions.click(hidden_submenu)
        actions.perform()
```

1. Introduction method

```
from selenium.webdriver.common.action_chains import ActionChains
```

2. Principle of implementation

In fact, the core idea of the implementation of the ActionChains module is that when you call the ActionChains method, it will not be executed immediately, but will store all the operations in a List in order, when you call the perform() method, the queue Time in time will be executed

3. Basic usage

```
Chain writing

ActionChains(driver).click(clk_btn).context_click(right_btn).perform()

 Step by step

 # Action

actions = ActionChains(driver)

 #Load click action

actions.click()

 #Load right click action

actions.context_click()

 # Execute all loaded actions

actions.perform()
```

4. Give a chestnut

```
from selenium import webdriver
 From selenium.webdriver.common.action_chains import ActionChains # Introducing the A

driver = webdriver.Firefox()
driver.get("https://www.baidu.com")

 #Target the element to be right clicked
right_click = driver.find_element_by_id("xxxx")

 # Perform a right mouse button on the positioned element
ActionChains(driver).context_click(right_click).perform()

```

1） ActionChains(driver)

Call the ActionChains() class and pass in the browser driver driver as a parameter.

2） c(right_click)

The context_click() method is used to simulate a mouse right-click operation, and you need to specify the element position when calling.

3） perform()

Executing the behavior stored in all ActionChains can be understood as a commit action to the entire operation.

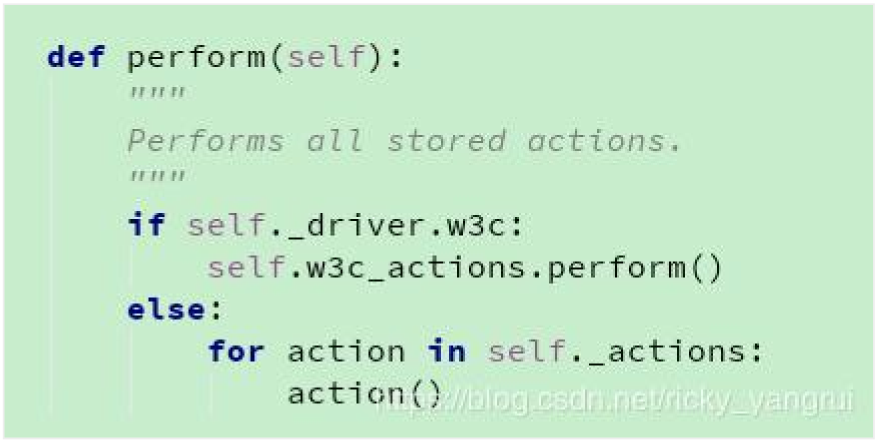
5. API summary

```
Perform(self): ---Execute all actions in the chain
 Reset_actions(self): ---Clear the action stored at the far end
 Click(self, on_element=None): --- Left mouse click
 Click_and_hold(self, on_element=None): -- Left click on the mouse, do not release
 Context_click(self, on_element=None): --- Right mouse click
 Double_click(self, on_element=None): --- Double click with the mouse
 Drag_and_drop(self, source, target): --- Drag and drop to an element and release
 Drag_and_drop_by_offset(self, source, xoffset, yoffset): --- Drag and drop to a cert
 Key_down(self, value, element=None): ---A keyboard key is pressed
 Key_up(self, value, element=None): --- Release a key
 Move_by_offset(self, xoffset, yoffset): ---Mouse moves to a certain coordinate
 Move_to_element(self, to_element): ---Mouse moves to an element
Move_to_element_with_offset(self, to_element, xoffset, yoffset): --- Move to a position
 Release(self, on_element=None): --- Release the mouse on an element
 Send_keys(self, *keys_to_send): ---Send some values to the current focus element
 Send_keys_to_element(self, element, *keys_to_send): ---Send some values to the specifie
```

two mouse operations

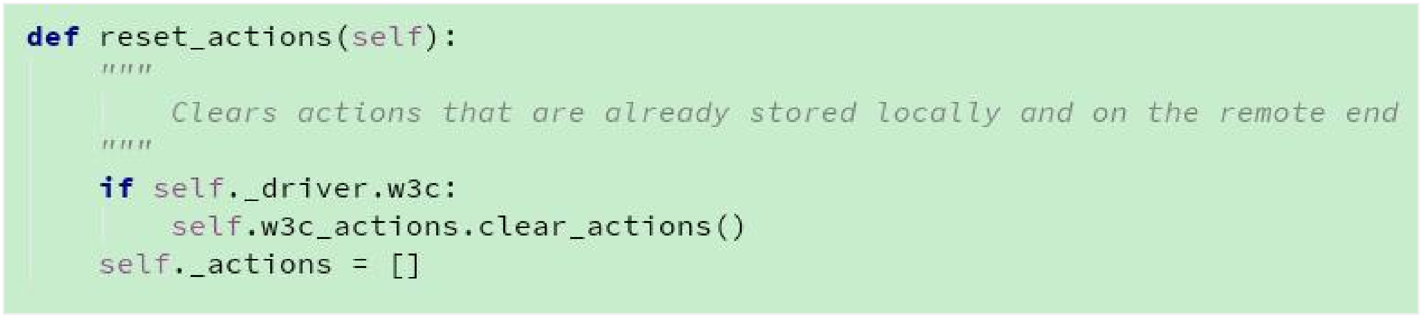
Detailed API introduction

1. perform # perform all actions in the chain



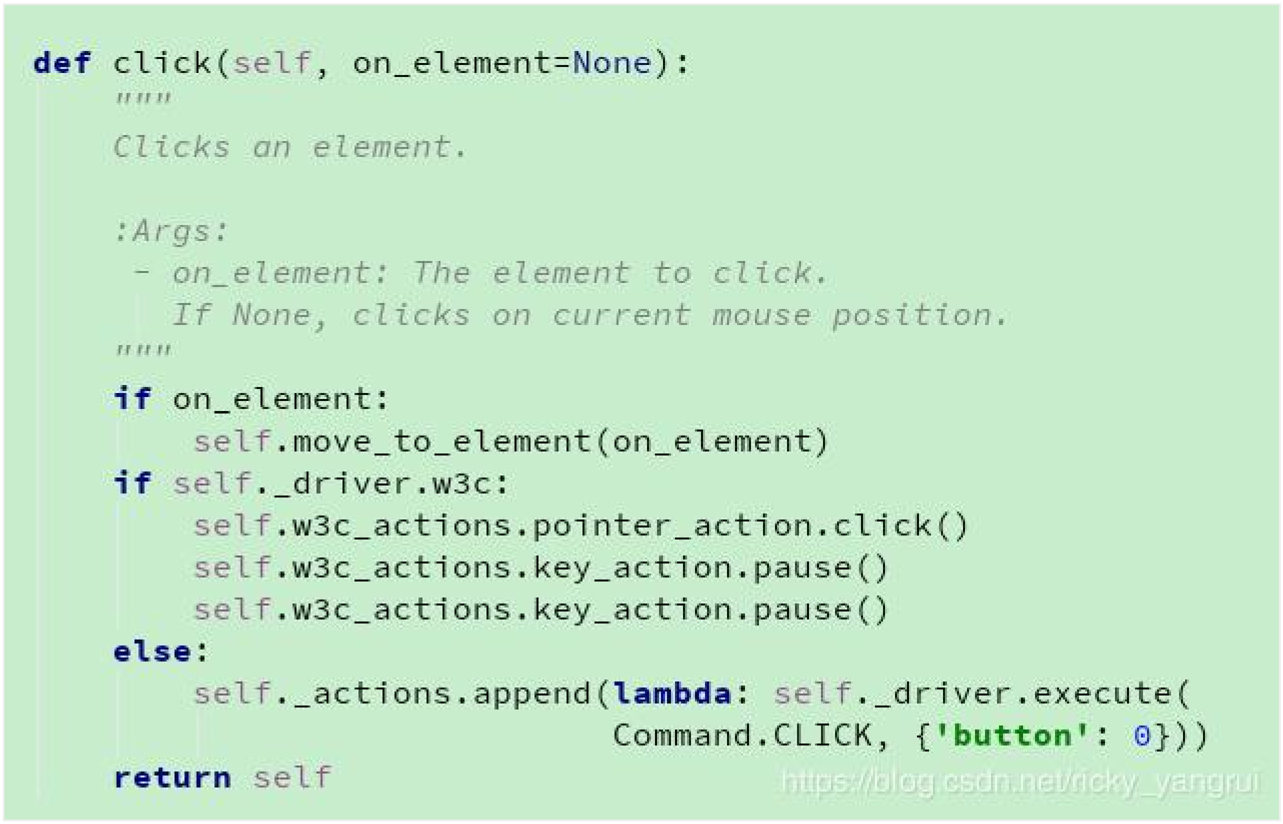
```
def perform(self):
    """
    Performs all stored actions.
    """
    if self._driver.w3c:
        self.w3c_actions.perform()
    else:
        for action in self._actions:
            action()
```

2. reset_actions #Clear the action stored at the far end



```
def reset_actions(self):
    """
        Clears actions that are already stored locally and on the remote end
    """
    if self._driver.w3c:
        self.w3c_actions.clear_actions()
    self._actions = []
```

3.click #left mouse click



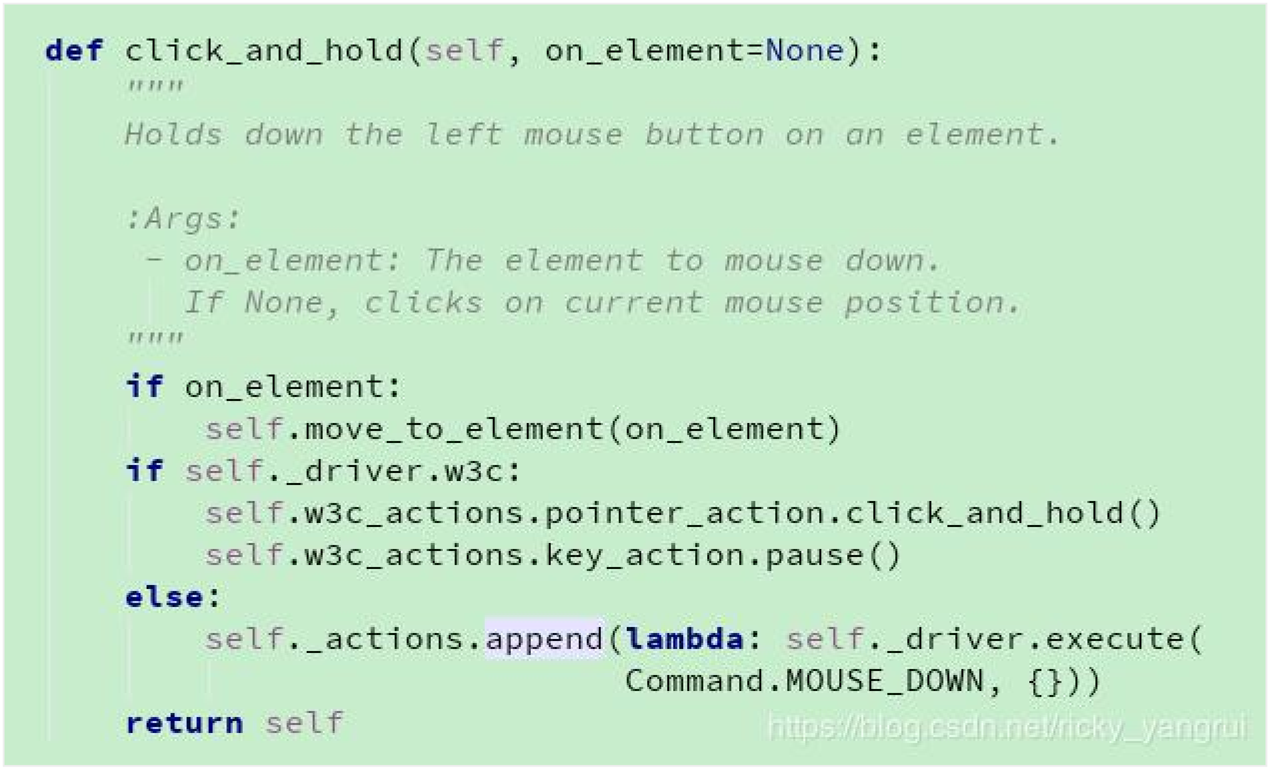
```
def click(self, on_element=None):
    """
    Clicks an element.

    :Args:
     - on_element: The element to click.
       If None, clicks on current mouse position.
    """
    if on_element:
        self.move_to_element(on_element)
    if self._driver.w3c:
        self.w3c_actions.pointer_action.click()
        self.w3c_actions.key_action.pause()
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
                                      Command.CLICK, {'button': 0}))
    return self
```

chestnut:

```
ActionChains(driver).click()
```

4.click_and_hold #Do not click on the left mouse button

TOP

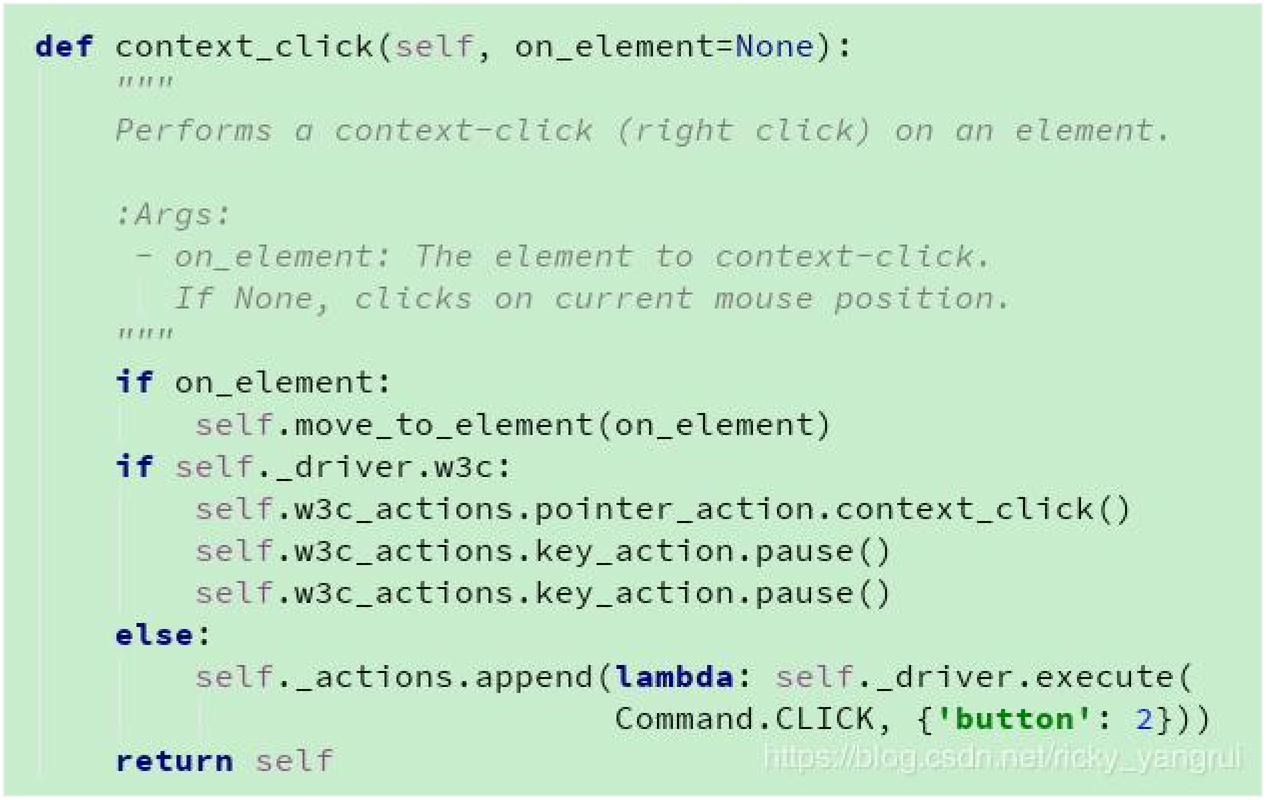


```
def click_and_hold(self, on_element=None):
    """
    Holds down the left mouse button on an element.

    :Args:
     - on_element: The element to mouse down.
       If None, clicks on current mouse position.
    """
    if on_element:
        self.move_to_element(on_element)
    if self._driver.w3c:
        self.w3c_actions.pointer_action.click_and_hold()
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
                                 Command.MOUSE_DOWN, {}))
    return self
```

chestnut:

```
ActionChains(driver).click_and_hold()
```

5.context_click

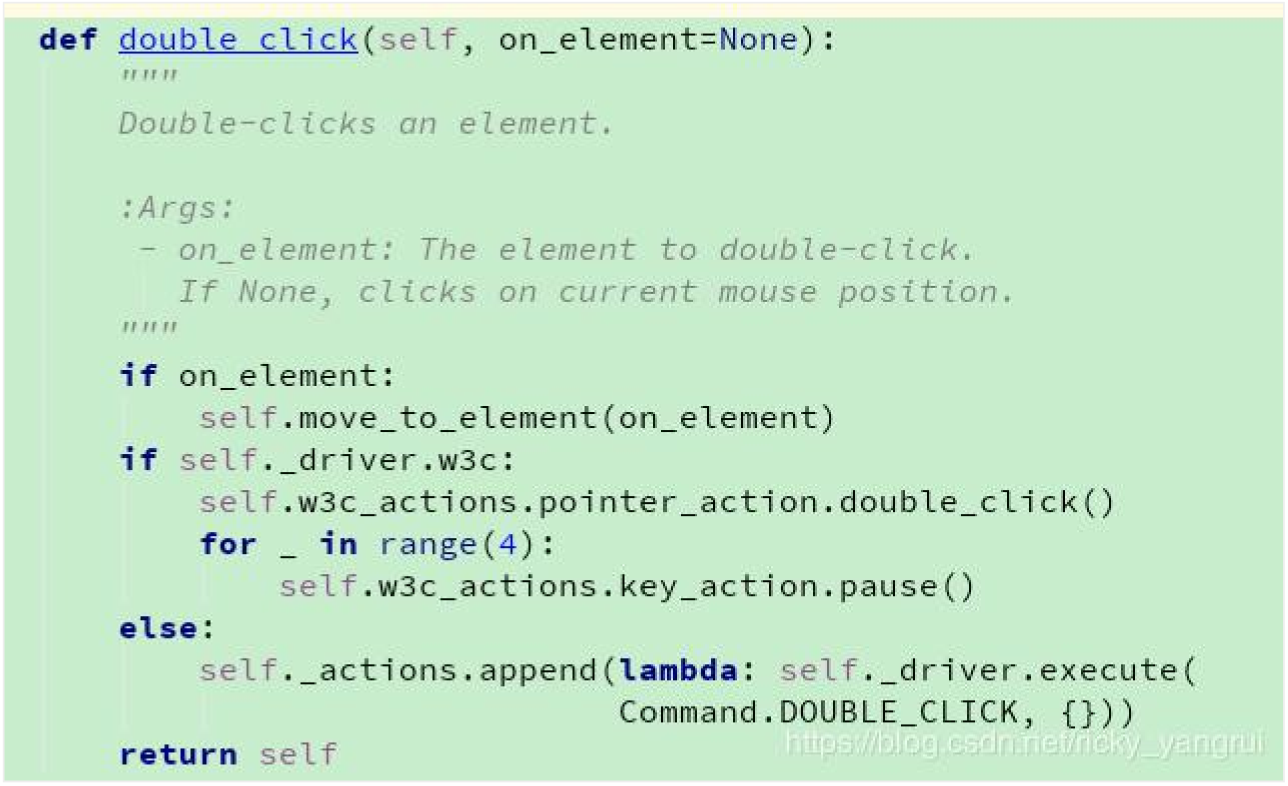


```
def context_click(self, on_element=None):
    """
    Performs a context-click (right click) on an element.

    :Args:
     - on_element: The element to context-click.
       If None, clicks on current mouse position.
    """
    if on_element:
        self.move_to_element(on_element)
    if self._driver.w3c:
        self.w3c_actions.pointer_action.context_click()
        self.w3c_actions.key_action.pause()
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
                                     Command.CLICK, {'button': 2}))
    return self
```

chestnut:

```
ActionChains(driver).context_click()
```

6.double_click #Double left mouse click

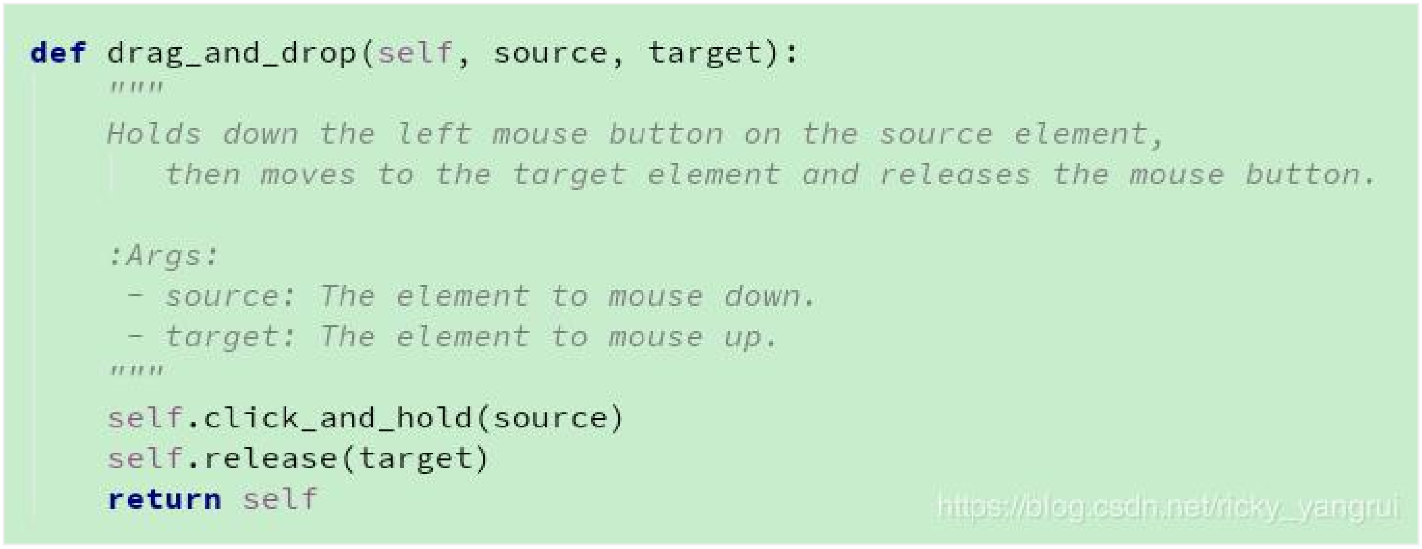


```
def double_click(self, on_element=None):
    """
    Double-clicks an element.

    :Args:
     - on_element: The element to double-click.
       If None, clicks on current mouse position.
    """
    if on_element:
        self.move_to_element(on_element)
    if self._driver.w3c:
        self.w3c_actions.pointer_action.double_click()
        for _ in range(4):
            self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
                                      Command.DOUBLE_CLICK, {}))
    return self
```

```
Chestnuts:
```

```
#Locate the element you want to double-click

double_click = driver.find_element_by_id("xx")

 # Hover the positioned element

ActionChains(driver).double_click(double_click).perform()
```

7. drag_and_drop(source,target) # Drag and drop to an element and release



```
def drag_and_drop(self, source, target):
    """
    Holds down the left mouse button on the source element,
       then moves to the target element and releases the mouse button.

    :Args:
     - source: The element to mouse down.
     - target: The element to mouse up.
    """
    self.click_and_hold(source)
    self.release(target)
    return self
```

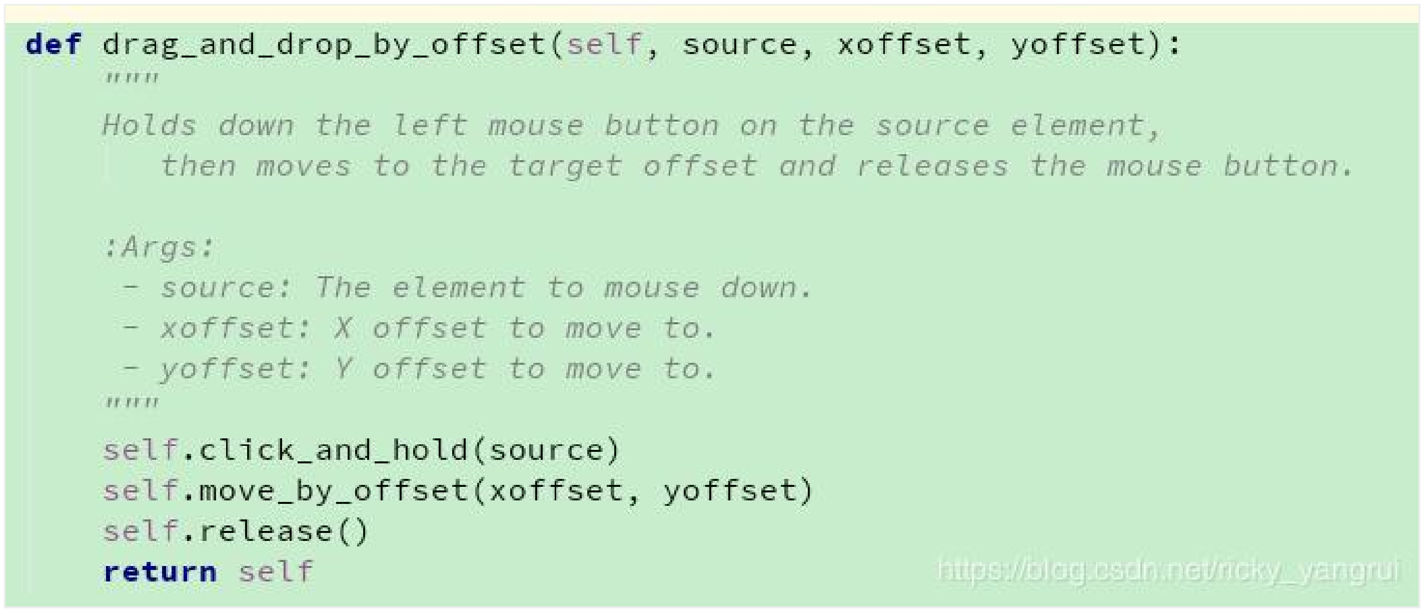
```
Look at the source code explanation, hold down the left mouse button on the source elemen
```

1. Source: The source element of the mouse drag.
2. Target: The target element released by the mouse.

```
Chestnuts:
```

```
# Position to the original position of the element

element = driver.find_element_by_id("xx")

 #Target to the target location where the element is to be moved

target = driver.find_element_by_id("xx")

 #Execution element drag and drop operation

ActionChains(driver).drag_and_drop(element,target).perform()
```

8.drag_and_drop_by_offset #Drop to a coordinate and release

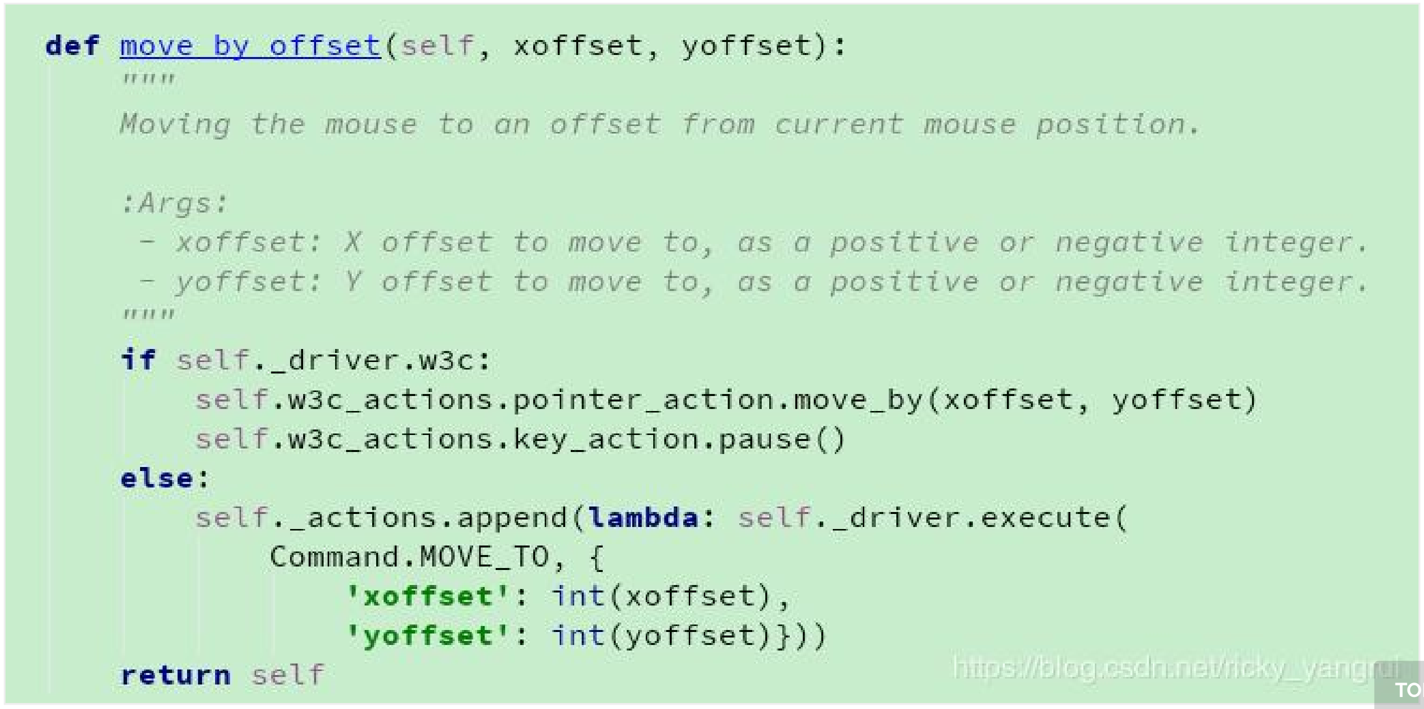


```
def drag_and_drop_by_offset(self, source, xoffset, yoffset):
    """
    Holds down the left mouse button on the source element,
       then moves to the target offset and releases the mouse button.

    :Args:
     - source: The element to mouse down.
     - xoffset: X offset to move to.
     - yoffset: Y offset to move to.
    """
    self.click_and_hold(source)
    self.move_by_offset(xoffset, yoffset)
    self.release()
    return self
```

chestnut:

```
# Drag a source element to the x y sitting at the top of the source. There may be a
ActionChains(driver).drag_and_drop_by_offset(source, x, y)
```

9. move_by_offset #Mouse moves to a coordinate

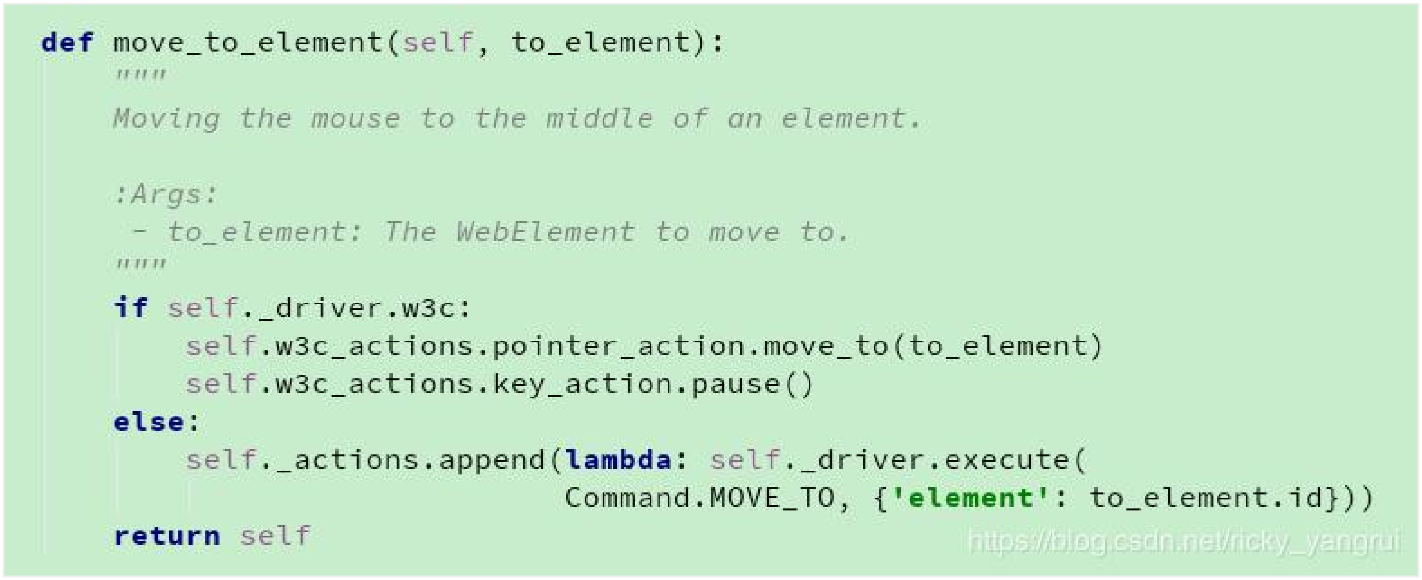


```
def move_by_offset(self, xoffset, yoffset):
    """
    Moving the mouse to an offset from current mouse position.

    :Args:
     - xoffset: X offset to move to, as a positive or negative integer.
     - yoffset: Y offset to move to, as a positive or negative integer.
    """
    if self._driver.w3c:
        self.w3c_actions.pointer_action.move_by(xoffset, yoffset)
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
            Command.MOVE_TO, {
                'xoffset': int(xoffset),
                'yoffset': int(yoffset)}))
    return self
```

chestnut:

```
# This is also a way of dragging, moving the coordinates based on the upper left corr
ActionChains(driver).click_and_hold(dom).move_by_offset(169,188).release().perform()

ActionChains(driver).move_by_offset(a['x'],a['y']).double_click(dis).perform()
```

10. move_to_element #Mouse moves to an element

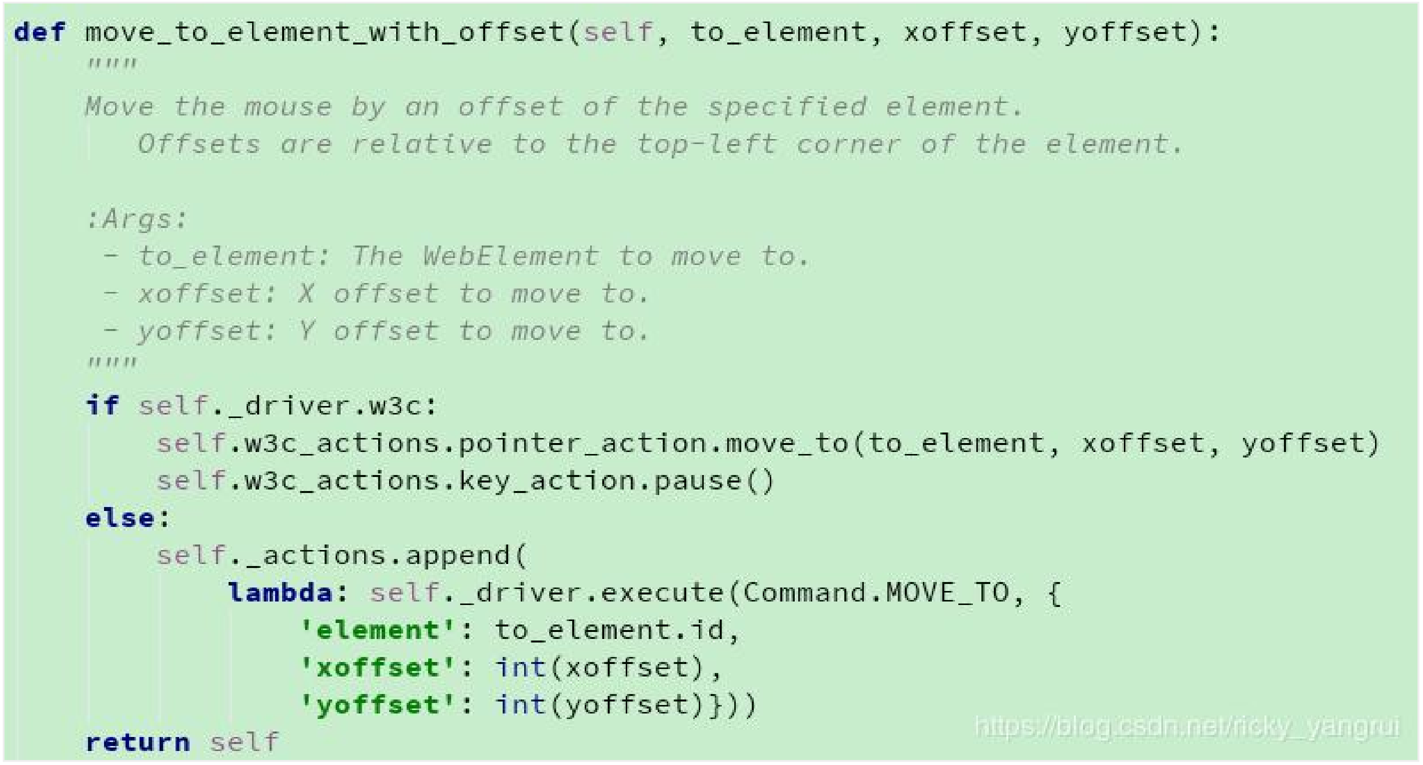


```
def move_to_element(self, to_element):
    """
    Moving the mouse to the middle of an element.

    :Args:
     - to_element: The WebElement to move to.
    """
    if self._driver.w3c:
        self.w3c_actions.pointer_action.move_to(to_element)
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
                             Command.MOVE_TO, {'element': to_element.id}))
    return self
```

chestnut:

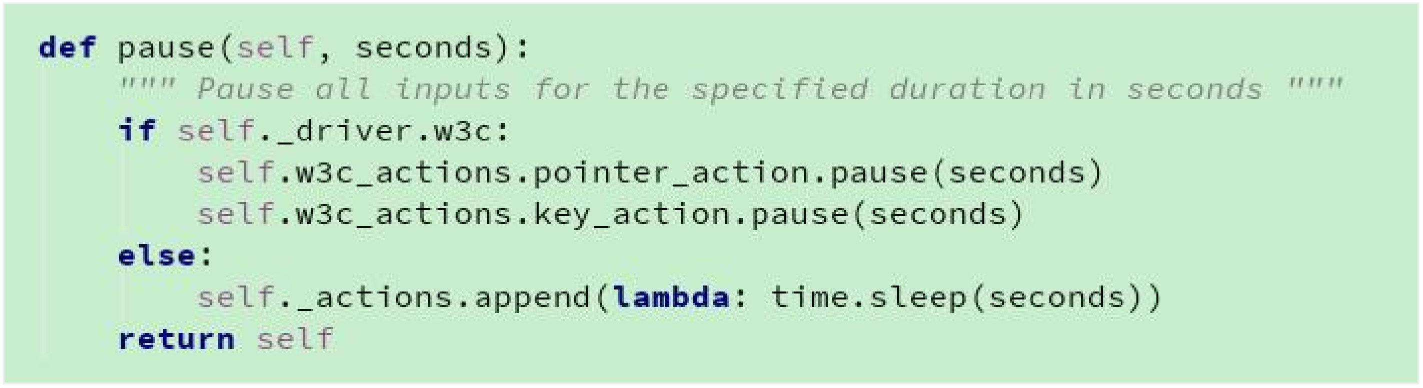
```
ActionChains(driver).move_to_element(e)
```

11. move_to_element_with_offset # Move to a position from an element (upper left corner)



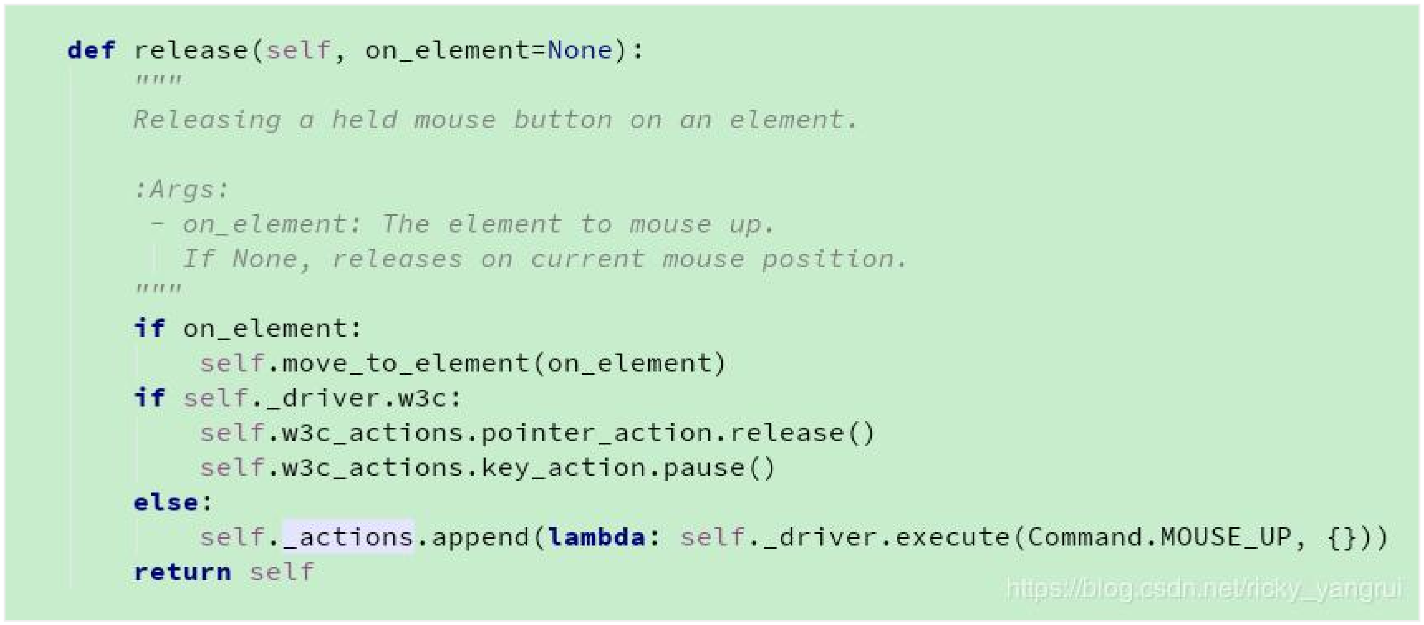
```python
def move_to_element_with_offset(self, to_element, xoffset, yoffset):
    """
    Move the mouse by an offset of the specified element.
       Offsets are relative to the top-left corner of the element.

    :Args:
     - to_element: The WebElement to move to.
     - xoffset: X offset to move to.
     - yoffset: Y offset to move to.
    """
    if self._driver.w3c:
        self.w3c_actions.pointer_action.move_to(to_element, xoffset, yoffset)
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(
            lambda: self._driver.execute(Command.MOVE_TO, {
                'element': to_element.id,
                'xoffset': int(xoffset),
                'yoffset': int(yoffset)}))
    return self
```

12. pause



```python
def pause(self, seconds):
    """ Pause all inputs for the specified duration in seconds """
    if self._driver.w3c:
        self.w3c_actions.pointer_action.pause(seconds)
        self.w3c_actions.key_action.pause(seconds)
    else:
        self._actions.append(lambda: time.sleep(seconds))
    return self
```

13. release # Release the left mouse button on an element



```
def release(self, on_element=None):
    """
    Releasing a held mouse button on an element.

    :Args:
     - on_element: The element to mouse up.
       If None, releases on current mouse position.
    """
    if on_element:
        self.move_to_element(on_element)
    if self._driver.w3c:
        self.w3c_actions.pointer_action.release()
        self.w3c_actions.key_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(Command.MOUSE_UP, {}))
    return self
```

chestnut:

```
ActionChains(driver).release()
```

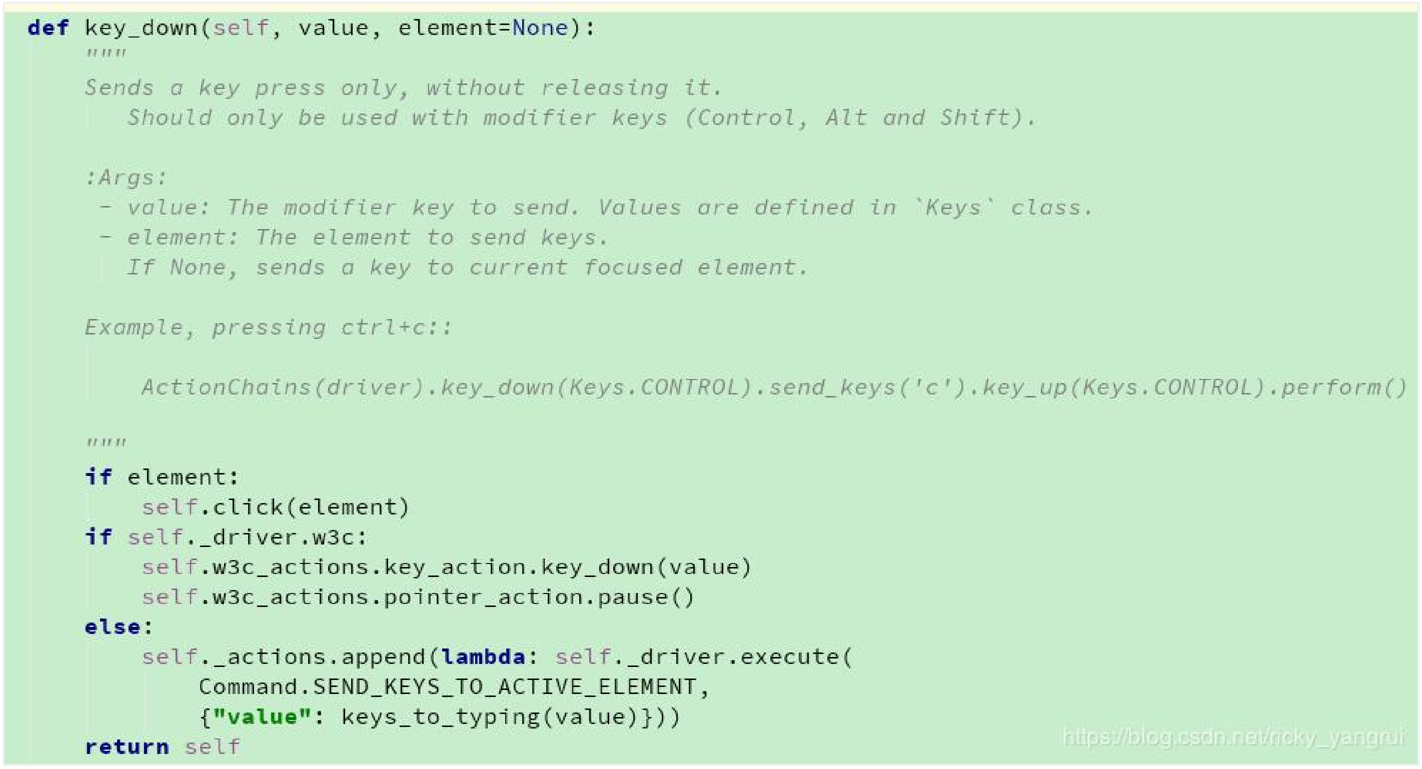
Three keyboard operation

Keys can work with key_down, and key_up to do some keyboard operations.

1. Introducing the package:

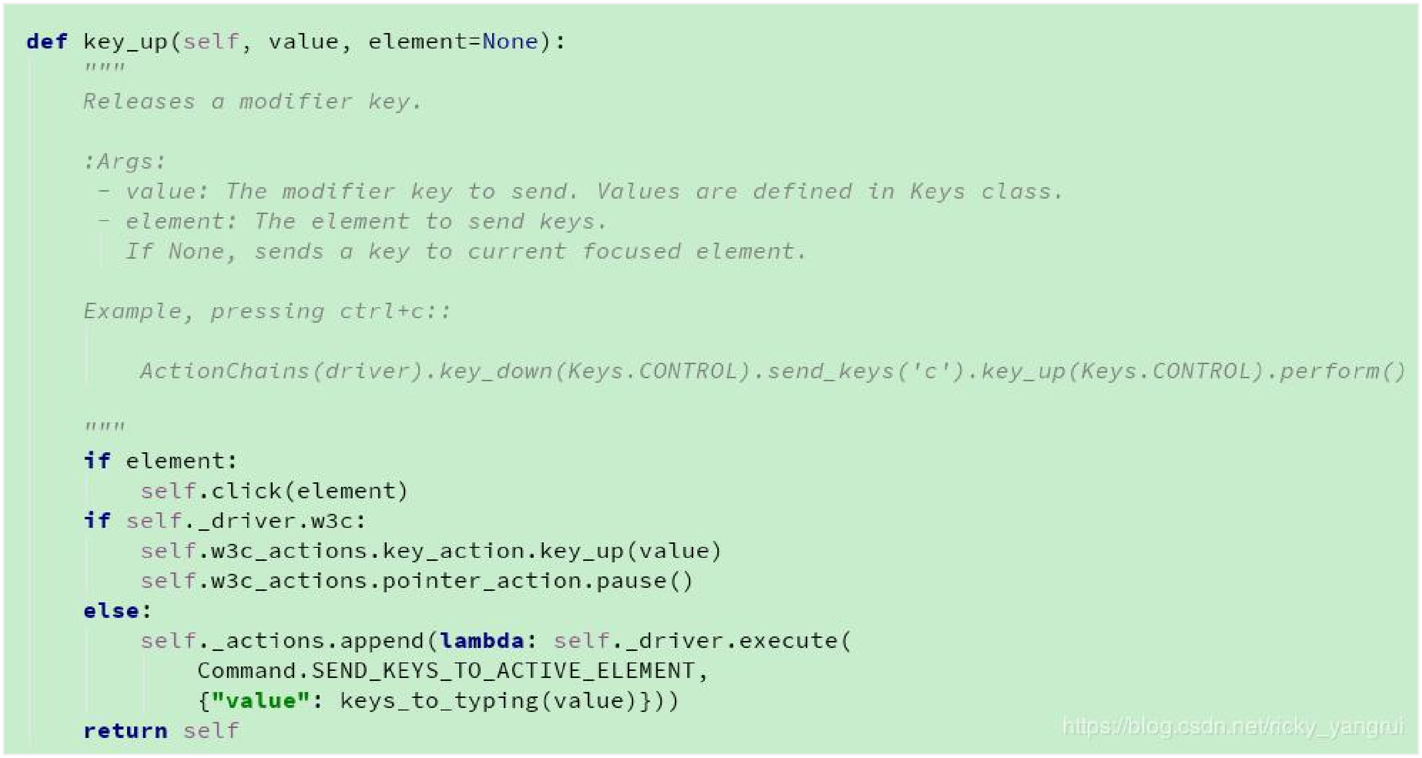
```
from selenium.webdriver.common.keys import Keys
```

2. key_down #A keyboard key is pressed



```python
def key_down(self, value, element=None):
    """
    Sends a key press only, without releasing it.
       Should only be used with modifier keys (Control, Alt and Shift).

    :Args:
     - 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.

    Example, pressing ctrl+c::

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

    """
    if element:
        self.click(element)
    if self._driver.w3c:
        self.w3c_actions.key_action.key_down(value)
        self.w3c_actions.pointer_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
            Command.SEND_KEYS_TO_ACTIVE_ELEMENT,
            {"value": keys_to_typing(value)}))
    return self
```

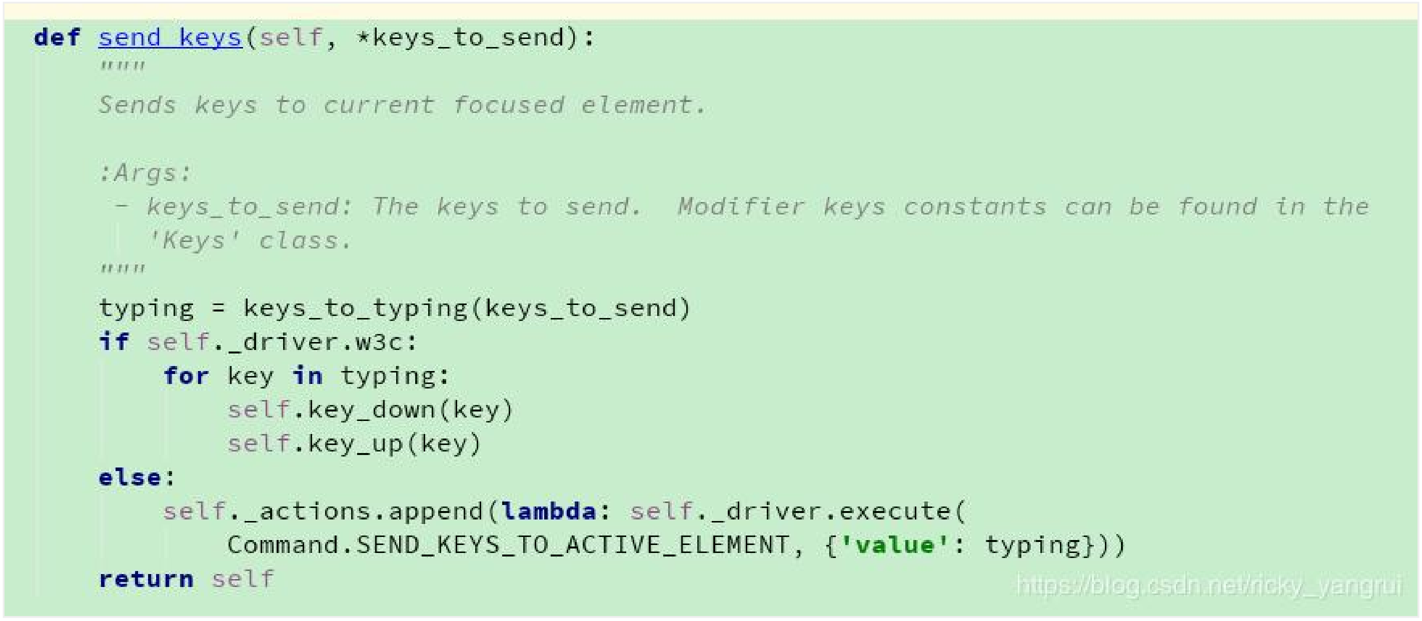
3. key_up # Release a key



```python
def key_up(self, value, element=None):
    """
    Releases a modifier key.

    :Args:
     - 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.

    Example, pressing ctrl+c::

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

    """
    if element:
        self.click(element)
    if self._driver.w3c:
        self.w3c_actions.key_action.key_up(value)
        self.w3c_actions.pointer_action.pause()
    else:
        self._actions.append(lambda: self._driver.execute(
            Command.SEND_KEYS_TO_ACTIVE_ELEMENT,
            {"value": keys_to_typing(value)}))
    return self
```

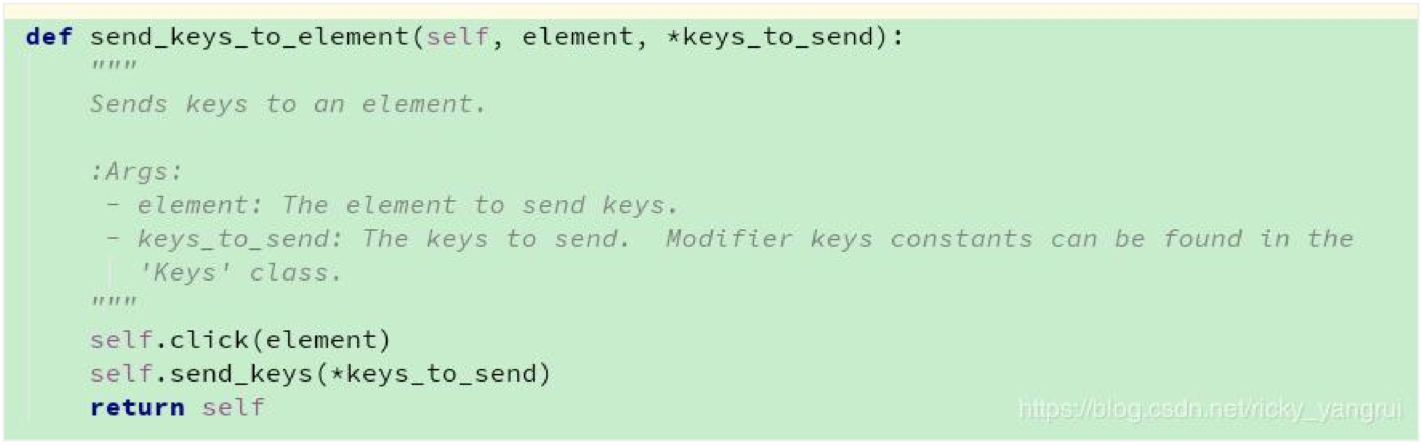
4. send_keys #send some values to the current focus element

TOP



```
def send_keys(self, *keys_to_send):
    """
    Sends keys to current focused element.

    :Args:
     - keys_to_send: The keys to send.  Modifier keys constants can be found in the
       'Keys' class.
    """
    typing = keys_to_typing(keys_to_send)
    if self._driver.w3c:
        for key in typing:
            self.key_down(key)
            self.key_up(key)
    else:
        self._actions.append(lambda: self._driver.execute(
            Command.SEND_KEYS_TO_ACTIVE_ELEMENT, {'value': typing}))
    return self
```

5 send_keys_to_element #Send some values to the specified element



```
def send_keys_to_element(self, element, *keys_to_send):
    """
    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.
    """
    self.click(element)
    self.send_keys(*keys_to_send)
    return self
```

6. keys

chestnut:

```
# key_down Simulate a key on the keyboard key_up Release a button and use it with sen
ActionChains(driver).key_down(Keys.CONTROL,dom).send_keys('a').send_keys('c').key_up(
    .key_down(Keys.CONTROL,dom1).send_keys('v').key_up(Keys.CONTROL).perform()
```

```
#Input box input content
driver.find_element_by_id("kw").send_keys("seleniumm")
```

TOP

```
# Delete one of the multiple inputs m
driver.find_element_by_id("kw").send_keys(Keys.BACK_SPACE)
 #Enter the space bar + "tutorial"
driver.find_element_by_id("kw").send_keys(Keys.SPACE)
 Driver.find_element_by_id("kw").send_keys(u"tutorial")
 #ctrl+a Select all input box contents
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'a')
 #ctrl+x Cut the input box contents
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'x')
 #ctrl+v Paste content into the input box
driver.find_element_by_id("kw").send_keys(Keys.CONTROL,'v')
 #Enter the keyboard instead of clicking
driver.find_element_by_id("su").send_keys(Keys.ENTER)
#  ESC
driver.find_element_by_id("su").send_keys(Keys.ESCAPE)
```

```
Send_keys(Keys.BACK_SPACE) delete key (BackSpace)
 Send_keys(Keys.SPACE) Spacebar (Space)
 Send_keys(Keys.TAB) tab (Tab)
 Send_keys(Keys.ESCAPE) Back button (Esc)
 Send_keys(Keys.ENTER) Enter (Enter)
 Send_keys(Keys.CONTROL,'a') Select all (Ctrl+A)
 Send_keys(Keys.CONTROL,'c') Copy (Ctrl+C)
 Send_keys(Keys.CONTROL,'x') Cut (Ctrl+X)
 Send_keys(Keys.CONTROL,'v') Paste (Ctrl+V)
 Send_keys(Keys.F1) keyboard F1
 Send_keys(Keys.F12) keyboard F12
```

ALL keys：

```
class Keys(object):
    """
    Set of special keys codes.
    """

    NULL = '\ue000'
    CANCEL = '\ue001'  # ^break
    HELP = '\ue002'
    BACKSPACE = '\ue003'
    BACK_SPACE = BACKSPACE
    TAB = '\ue004'
    CLEAR = '\ue005'
    RETURN = '\ue006'
    ENTER = '\ue007'
    SHIFT = '\ue008'
    LEFT_SHIFT = SHIFT
    CONTROL = '\ue009'
    LEFT_CONTROL = CONTROL
    ALT = '\ue00a'
    LEFT_ALT = ALT
    PAUSE = '\ue00b'
    ESCAPE = '\ue00c'
    SPACE = '\ue00d'
    PAGE_UP = '\ue00e'
    PAGE_DOWN = '\ue00f'
    END = '\ue010'
    HOME = '\ue011'
    LEFT = '\ue012'
    ARROW_LEFT = LEFT
    UP = '\ue013'
    ARROW_UP = UP
    RIGHT = '\ue014'
    ARROW_RIGHT = RIGHT
    DOWN = '\ue015'
    ARROW_DOWN = DOWN
    INSERT = '\ue016'
    DELETE = '\ue017'
    SEMICOLON = '\ue018'
    EQUALS = '\ue019'

    NUMPAD0 = '\ue01a'  # number pad keys
    NUMPAD1 = '\ue01b'
    NUMPAD2 = '\ue01c'
    NUMPAD3 = '\ue01d'
    NUMPAD4 = '\ue01e'
    NUMPAD5 = '\ue01f'
    NUMPAD6 = '\ue020'
    NUMPAD7 = '\ue021'
    NUMPAD8 = '\ue022'
    NUMPAD9 = '\ue023'
    MULTIPLY = '\ue024'
    ADD = '\ue025'
    SEPARATOR = '\ue026'
    SUBTRACT = '\ue027'
    DECIMAL = '\ue028'
```

TOP

```
    DIVIDE = '\ue029'

    F1 = '\ue031'  # function  keys
    F2 = '\ue032'
    F3 = '\ue033'
    F4 = '\ue034'
    F5 = '\ue035'
    F6 = '\ue036'
    F7 = '\ue037'
    F8 = '\ue038'
    F9 = '\ue039'
    F10 = '\ue03a'
    F11 = '\ue03b'
    F12 = '\ue03c'

    META = '\ue03d'
    COMMAND = '\ue03d'
```

TOP

Related Posts

- Selenium3 + Python3 Automated Test Series 4 - Mouse Events and Keyboard Events
- Python automated simulation mouse: ActionChains class
- Selenium's fun mouse and keyboard operation (ActionChains)
- Automated learning [4] - ActionChains basic operations

TOP

- Puppeteer automated test mouse keyboard basic operations and precautions
- GUI mouse and keyboard events
- Common mouse and keyboard events
- VTK mouse and keyboard events
- Mouse and keyboard events
- tkinter mouse and keyboard events

Do This Immediately if You Have Diabetes (Watch)

It's Impossible To Last 5 Minutes Playing This Game

TOP