7. WebDriver API

Note

This is not an official documentation. Official API documentation is available [here](https://seleniumhq.github.io/selenium/docs/api/py/api.html).

This chapter covers all the interfaces of Selenium WebDriver.

**Recommended Import Style**

The API definitions in this chapter show the absolute location of classes. However, the recommended import style is as given below:

**from** **selenium** **import** webdriver

Then, you can access the classes like this:

webdriver.Firefox

webdriver.FirefoxProfile

webdriver.Chrome

webdriver.ChromeOptions

webdriver.Ie

webdriver.Opera

webdriver.PhantomJS

webdriver.Remote

webdriver.DesiredCapabilities

webdriver.ActionChains

webdriver.TouchActions

webdriver.Proxy

The special keys class (Keys) can be imported like this:

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

The exception classes can be imported like this (Replace the TheNameOfTheExceptionClass with the actual class name given below):

**from** **selenium.common.exceptions** **import** [TheNameOfTheExceptionClass]

**Conventions used in the API**

Some attributes are callable (or methods) and others are non-callable (properties). All the callable attributes are ending with round brackets.

Here is an example for property:

* current\_url

URL of the currently loaded page.

Usage:

driver.current\_url

Here is an example of a method:

* close()

Closes the current window.

Usage:

driver.close()

7.1. Exceptions

Exceptions that may happen in all the webdriver code.

*exception*selenium.common.exceptions.**ElementClickInterceptedException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

The Element Click command could not be completed because the element receiving the events is obscuring the element that was requested clicked.

*exception*selenium.common.exceptions.**ElementNotInteractableException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidElementStateException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidElementStateException)

Thrown when an element is present in the DOM but interactions with that element will hit another element do to paint order

*exception*selenium.common.exceptions.**ElementNotSelectableException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidElementStateException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidElementStateException)

Thrown when trying to select an unselectable element.

For example, selecting a ‘script’ element.

*exception*selenium.common.exceptions.**ElementNotVisibleException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidElementStateException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidElementStateException)

Thrown when an element is present on the DOM, but it is not visible, and so is not able to be interacted with.

Most commonly encountered when trying to click or read text of an element that is hidden from view.

*exception*selenium.common.exceptions.**ErrorInResponseException**(*response*, *msg*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when an error has occurred on the server side.

This may happen when communicating with the firefox extension or the remote driver server.

**\_\_init\_\_**(*response*, *msg*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*exception*selenium.common.exceptions.**ImeActivationFailedException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when activating an IME engine has failed.

*exception*selenium.common.exceptions.**ImeNotAvailableException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when IME support is not available. This exception is thrown for every IME-related method call if IME support is not available on the machine.

*exception*selenium.common.exceptions.**InsecureCertificateException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Navigation caused the user agent to hit a certificate warning, which is usually the result of an expired or invalid TLS certificate.

*exception*selenium.common.exceptions.**InvalidArgumentException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

The arguments passed to a command are either invalid or malformed.

*exception*selenium.common.exceptions.**InvalidCookieDomainException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when attempting to add a cookie under a different domain than the current URL.

*exception*selenium.common.exceptions.**InvalidCoordinatesException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

The coordinates provided to an interactions operation are invalid.

*exception*selenium.common.exceptions.**InvalidElementStateException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a command could not be completed because the element is in an invalid state.

This can be caused by attempting to clear an element that isn’t both editable and resettable.

*exception*selenium.common.exceptions.**InvalidSelectorException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.NoSuchElementException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.NoSuchElementException)

Thrown when the selector which is used to find an element does not return a WebElement. Currently this only happens when the selector is an xpath expression and it is either syntactically invalid (i.e. it is not a xpath expression) or the expression does not select WebElements (e.g. “count(//input)”).

*exception*selenium.common.exceptions.**InvalidSessionIdException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Occurs if the given session id is not in the list of active sessions, meaning the session either does not exist or that it’s not active.

*exception*selenium.common.exceptions.**InvalidSwitchToTargetException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when frame or window target to be switched doesn’t exist.

*exception*selenium.common.exceptions.**JavascriptException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

An error occurred while executing JavaScript supplied by the user.

*exception*selenium.common.exceptions.**MoveTargetOutOfBoundsException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when the target provided to the *ActionsChains* move() method is invalid, i.e. out of document.

*exception*selenium.common.exceptions.**NoAlertPresentException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when switching to no presented alert.

This can be caused by calling an operation on the Alert() class when an alert is not yet on the screen.

*exception*selenium.common.exceptions.**NoSuchAttributeException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when the attribute of element could not be found.

You may want to check if the attribute exists in the particular browser you are testing against. Some browsers may have different property names for the same property. (IE8’s .innerText vs. Firefox .textContent)

*exception*selenium.common.exceptions.**NoSuchCookieException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

No cookie matching the given path name was found amongst the associated cookies of the current browsing context’s active document.

*exception*selenium.common.exceptions.**NoSuchElementException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when element could not be found.

If you encounter this exception, you may want to check the following:

* Check your selector used in your find\_by…
* Element may not yet be on the screen at the time of the find operation, (webpage is still loading) see selenium.webdriver.support.wait.WebDriverWait() for how to write a wait wrapper to wait for an element to appear.

*exception*selenium.common.exceptions.**NoSuchFrameException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidSwitchToTargetException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidSwitchToTargetException)

Thrown when frame target to be switched doesn’t exist.

*exception*selenium.common.exceptions.**NoSuchWindowException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidSwitchToTargetException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidSwitchToTargetException)

Thrown when window target to be switched doesn’t exist.

To find the current set of active window handles, you can get a list of the active window handles in the following way:

print driver.window\_handles

*exception*selenium.common.exceptions.**RemoteDriverServerException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

*exception*selenium.common.exceptions.**ScreenshotException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

A screen capture was made impossible.

*exception*selenium.common.exceptions.**SessionNotCreatedException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

A new session could not be created.

*exception*selenium.common.exceptions.**StaleElementReferenceException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a reference to an element is now “stale”.

Stale means the element no longer appears on the DOM of the page.

Possible causes of StaleElementReferenceException include, but not limited to:

* You are no longer on the same page, or the page may have refreshed since the element was located.
* The element may have been removed and re-added to the screen, since it was located. Such as an element being relocated. This can happen typically with a javascript framework when values are updated and the node is rebuilt.
* Element may have been inside an iframe or another context which was refreshed.

*exception*selenium.common.exceptions.**TimeoutException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a command does not complete in enough time.

*exception*selenium.common.exceptions.**UnableToSetCookieException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a driver fails to set a cookie.

*exception*selenium.common.exceptions.**UnexpectedAlertPresentException**(*msg=None*, *screen=None*, *stacktrace=None*, *alert\_text=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when an unexpected alert is appeared.

Usually raised when when an expected modal is blocking webdriver form executing any more commands.

**\_\_init\_\_**(*msg=None*, *screen=None*, *stacktrace=None*, *alert\_text=None*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*exception*selenium.common.exceptions.**UnexpectedTagNameException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a support class did not get an expected web element.

*exception*selenium.common.exceptions.**UnknownMethodException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](https://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

The requested command matched a known URL but did not match an method for that URL.

*exception*selenium.common.exceptions.**WebDriverException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: **exceptions.Exception**

Base webdriver exception.

**\_\_init\_\_**(*msg=None*, *screen=None*, *stacktrace=None*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

7.2. Action Chains

The ActionChains implementation,

*class*selenium.webdriver.common.action\_chains.**ActionChains**(*driver*)

Bases: **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.

**\_\_init\_\_**(*driver*)

Creates a new ActionChains.

|  |  |
| --- | --- |
| **Args:** | * driver: The WebDriver instance which performs user actions. |

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

Clicks an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to click. If None, clicks on current mouse position. |

**click\_and\_hold**(*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. |

**context\_click**(*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. |

**double\_click**(*on\_element=None*)

Double-clicks an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to double-click. If None, 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.

|  |  |
| --- | --- |
| **Args:** | * source: The element to mouse down. * target: The element to mouse up. |

**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.

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

**key\_down**(*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()

**key\_up**(*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()

**move\_by\_offset**(*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. |

**move\_to\_element**(*to\_element*)

Moving the mouse to the middle of an element.

|  |  |
| --- | --- |
| **Args:** | * to\_element: The WebElement to move to. |

**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.

|  |  |
| --- | --- |
| **Args:** | * to\_element: The WebElement to move to. * xoffset: X offset to move to. * yoffset: Y offset to move to. |

**pause**(*seconds*)

Pause all inputs for the specified duration in seconds

**perform**()

Performs all stored actions.

**release**(*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. |

**reset\_actions**()

Clears actions that are already stored locally and on the remote end

**send\_keys**(*\*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. |

**send\_keys\_to\_element**(*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. |

7.3. Alerts

The Alert implementation.

*class*selenium.webdriver.common.alert.**Alert**(*driver*)

Bases: **object**

Allows to work with alerts.

Use this class to interact with alert prompts. It contains methods for dismissing, accepting, inputting, and getting text from alert prompts.

Accepting / Dismissing alert prompts:

Alert(driver).accept()

Alert(driver).dismiss()

Inputting a value into an alert prompt:

name\_prompt = Alert(driver) name\_prompt.send\_keys(“Willian Shakesphere”) name\_prompt.accept()

Reading a the text of a prompt for verification:

alert\_text = Alert(driver).text self.assertEqual(“Do you wish to quit?”, alert\_text)

**\_\_init\_\_**(*driver*)

Creates a new Alert.

|  |  |
| --- | --- |
| **Args:** | * driver: The WebDriver instance which performs user actions. |

**accept**()

Accepts the alert available.

Usage:: Alert(driver).accept() # Confirm a alert dialog.

**dismiss**()

Dismisses the alert available.

**send\_keys**(*keysToSend*)

Send Keys to the Alert.

|  |  |
| --- | --- |
| **Args:** | * keysToSend: The text to be sent to Alert. |

**text**

Gets the text of the Alert.

7.4. Special Keys

The Keys implementation.

*class*selenium.webdriver.common.keys.**Keys**

Bases: **object**

Set of special keys codes.

**ADD***= u'\ue025'*

**ALT***= u'\ue00a'*

**ARROW\_DOWN***= u'\ue015'*

**ARROW\_LEFT***= u'\ue012'*

**ARROW\_RIGHT***= u'\ue014'*

**ARROW\_UP***= u'\ue013'*

**BACKSPACE***= u'\ue003'*

**BACK\_SPACE***= u'\ue003'*

**CANCEL***= u'\ue001'*

**CLEAR***= u'\ue005'*

**COMMAND***= u'\ue03d'*

**CONTROL***= u'\ue009'*

**DECIMAL***= u'\ue028'*

**DELETE***= u'\ue017'*

**DIVIDE***= u'\ue029'*

**DOWN***= u'\ue015'*

**END***= u'\ue010'*

**ENTER***= u'\ue007'*

**EQUALS***= u'\ue019'*

**ESCAPE***= u'\ue00c'*

**F1***= u'\ue031'*

**F10***= u'\ue03a'*

**F11***= u'\ue03b'*

**F12***= u'\ue03c'*

**F2***= u'\ue032'*

**F3***= u'\ue033'*

**F4***= u'\ue034'*

**F5***= u'\ue035'*

**F6***= u'\ue036'*

**F7***= u'\ue037'*

**F8***= u'\ue038'*

**F9***= u'\ue039'*

**HELP***= u'\ue002'*

**HOME***= u'\ue011'*

**INSERT***= u'\ue016'*

**LEFT***= u'\ue012'*

**LEFT\_ALT***= u'\ue00a'*

**LEFT\_CONTROL***= u'\ue009'*

**LEFT\_SHIFT***= u'\ue008'*

**META***= u'\ue03d'*

**MULTIPLY***= u'\ue024'*

**NULL***= u'\ue000'*

**NUMPAD0***= u'\ue01a'*

**NUMPAD1***= u'\ue01b'*

**NUMPAD2***= u'\ue01c'*

**NUMPAD3***= u'\ue01d'*

**NUMPAD4***= u'\ue01e'*

**NUMPAD5***= u'\ue01f'*

**NUMPAD6***= u'\ue020'*

**NUMPAD7***= u'\ue021'*

**NUMPAD8***= u'\ue022'*

**NUMPAD9***= u'\ue023'*

**PAGE\_DOWN***= u'\ue00f'*

**PAGE\_UP***= u'\ue00e'*

**PAUSE***= u'\ue00b'*

**RETURN***= u'\ue006'*

**RIGHT***= u'\ue014'*

**SEMICOLON***= u'\ue018'*

**SEPARATOR***= u'\ue026'*

**SHIFT***= u'\ue008'*

**SPACE***= u'\ue00d'*

**SUBTRACT***= u'\ue027'*

**TAB***= u'\ue004'*

**UP***= u'\ue013'*

7.5. Locate elements By

These are the attributes which can be used to locate elements. See the [Locating Elements](https://selenium-python.readthedocs.io/locating-elements.html#locating-elements) chapter for example usages.

The By implementation.

*class*selenium.webdriver.common.by.**By**

Bases: **object**

Set of supported locator strategies.

**CLASS\_NAME***= 'class name'*

**CSS\_SELECTOR***= 'css selector'*

**ID***= 'id'*

**LINK\_TEXT***= 'link text'*

**NAME***= 'name'*

**PARTIAL\_LINK\_TEXT***= 'partial link text'*

**TAG\_NAME***= 'tag name'*

**XPATH***= 'xpath'*

7.6. Desired Capabilities

See the [Using Selenium with remote WebDriver](https://selenium-python.readthedocs.io/getting-started.html#selenium-remote-webdriver) section for example usages of desired capabilities.

The Desired Capabilities implementation.

*class*selenium.webdriver.common.desired\_capabilities.**DesiredCapabilities**

Bases: **object**

Set of default supported desired capabilities.

Use this as a starting point for creating a desired capabilities object for requesting remote webdrivers for connecting to selenium server or selenium grid.

Usage Example:

**from** **selenium** **import** webdriver

selenium\_grid\_url = "http://198.0.0.1:4444/wd/hub"

*# Create a desired capabilities object as a starting point.*

capabilities = DesiredCapabilities.FIREFOX.copy()

capabilities['platform'] = "WINDOWS"

capabilities['version'] = "10"

*# Instantiate an instance of Remote WebDriver with the desired capabilities.*

driver = webdriver.Remote(desired\_capabilities=capabilities,

command\_executor=selenium\_grid\_url)

Note: Always use ‘.copy()’ on the DesiredCapabilities object to avoid the side effects of altering the Global class instance.

**ANDROID***= {'browserName': 'android', 'platform': 'ANDROID', 'version': ''}*

**CHROME***= {'browserName': 'chrome', 'platform': 'ANY', 'version': ''}*

**EDGE***= {'browserName': 'MicrosoftEdge', 'platform': 'WINDOWS', 'version': ''}*

**FIREFOX***= {'acceptInsecureCerts': True, 'browserName': 'firefox', 'marionette': True}*

**HTMLUNIT***= {'browserName': 'htmlunit', 'platform': 'ANY', 'version': ''}*

**HTMLUNITWITHJS***= {'browserName': 'htmlunit', 'javascriptEnabled': True, 'platform': 'ANY', 'version': 'firefox'}*

**INTERNETEXPLORER***= {'browserName': 'internet explorer', 'platform': 'WINDOWS', 'version': ''}*

**IPAD***= {'browserName': 'iPad', 'platform': 'MAC', 'version': ''}*

**IPHONE***= {'browserName': 'iPhone', 'platform': 'MAC', 'version': ''}*

**OPERA***= {'browserName': 'opera', 'platform': 'ANY', 'version': ''}*

**PHANTOMJS***= {'browserName': 'phantomjs', 'javascriptEnabled': True, 'platform': 'ANY', 'version': ''}*

**SAFARI***= {'browserName': 'safari', 'platform': 'MAC', 'version': ''}*

**WEBKITGTK***= {'browserName': 'MiniBrowser', 'platform': 'ANY', 'version': ''}*

7.7. Touch Actions

The Touch Actions implementation

*class*selenium.webdriver.common.touch\_actions.**TouchActions**(*driver*)

Bases: **object**

Generate touch actions. Works like ActionChains; actions are stored in the TouchActions object and are fired with perform().

**\_\_init\_\_**(*driver*)

Creates a new TouchActions object.

|  |  |
| --- | --- |
| **Args:** | * driver: The WebDriver instance which performs user actions. It should be with touchscreen enabled. |

**double\_tap**(*on\_element*)

Double taps on a given element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to tap. |

**flick**(*xspeed*, *yspeed*)

Flicks, starting anywhere on the screen.

|  |  |
| --- | --- |
| **Args:** | * xspeed: The X speed in pixels per second. * yspeed: The Y speed in pixels per second. |

**flick\_element**(*on\_element*, *xoffset*, *yoffset*, *speed*)

Flick starting at on\_element, and moving by the xoffset and yoffset with specified speed.

|  |  |
| --- | --- |
| **Args:** | * on\_element: Flick will start at center of element. * xoffset: X offset to flick to. * yoffset: Y offset to flick to. * speed: Pixels per second to flick. |

**long\_press**(*on\_element*)

Long press on an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to long press. |

**move**(*xcoord*, *ycoord*)

Move held tap to specified location.

|  |  |
| --- | --- |
| **Args:** | * xcoord: X Coordinate to move. * ycoord: Y Coordinate to move. |

**perform**()

Performs all stored actions.

**release**(*xcoord*, *ycoord*)

Release previously issued tap ‘and hold’ command at specified location.

|  |  |
| --- | --- |
| **Args:** | * xcoord: X Coordinate to release. * ycoord: Y Coordinate to release. |

**scroll**(*xoffset*, *yoffset*)

Touch and scroll, moving by xoffset and yoffset.

|  |  |
| --- | --- |
| **Args:** | * xoffset: X offset to scroll to. * yoffset: Y offset to scroll to. |

**scroll\_from\_element**(*on\_element*, *xoffset*, *yoffset*)

Touch and scroll starting at on\_element, moving by xoffset and yoffset.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element where scroll starts. * xoffset: X offset to scroll to. * yoffset: Y offset to scroll to. |

**tap**(*on\_element*)

Taps on a given element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to tap. |

**tap\_and\_hold**(*xcoord*, *ycoord*)

Touch down at given coordinates.

|  |  |
| --- | --- |
| **Args:** | * xcoord: X Coordinate to touch down. * ycoord: Y Coordinate to touch down. |

7.8. Proxy

The Proxy implementation.

*class*selenium.webdriver.common.proxy.**Proxy**(*raw=None*)

Bases: **object**

Proxy contains information about proxy type and necessary proxy settings.

**\_\_init\_\_**(*raw=None*)

Creates a new Proxy.

|  |  |
| --- | --- |
| **Args:** | * raw: raw proxy data. If None, default class values are used. |

**add\_to\_capabilities**(*capabilities*)

Adds proxy information as capability in specified capabilities.

|  |  |
| --- | --- |
| **Args:** | * capabilities: The capabilities to which proxy will be added. |

**auto\_detect**

Returns autodetect setting.

**autodetect***= False*

**ftpProxy***= ''*

**ftp\_proxy**

Returns ftp proxy setting.

**httpProxy***= ''*

**http\_proxy**

Returns http proxy setting.

**noProxy***= ''*

**no\_proxy**

Returns noproxy setting.

**proxyAutoconfigUrl***= ''*

**proxyType***= {'ff\_value': 6, 'string': 'UNSPECIFIED'}*

**proxy\_autoconfig\_url**

Returns proxy autoconfig url setting.

**proxy\_type**

Returns proxy type as *ProxyType*.

**socksPassword***= ''*

**socksProxy***= ''*

**socksUsername***= ''*

**socks\_password**

Returns socks proxy password setting.

**socks\_proxy**

Returns socks proxy setting.

**socks\_username**

Returns socks proxy username setting.

**sslProxy***= ''*

**ssl\_proxy**

Returns https proxy setting.

*class*selenium.webdriver.common.proxy.**ProxyType**

Set of possible types of proxy.

Each proxy type has 2 properties:

‘ff\_value’ is value of Firefox profile preference, ‘string’ is id of proxy type.

*classmethod***load**(*value*)

**AUTODETECT***= {'ff\_value': 4, 'string': 'AUTODETECT'}*

**DIRECT***= {'ff\_value': 0, 'string': 'DIRECT'}*

**MANUAL***= {'ff\_value': 1, 'string': 'MANUAL'}*

**PAC***= {'ff\_value': 2, 'string': 'PAC'}*

**RESERVED\_1***= {'ff\_value': 3, 'string': 'RESERVED1'}*

**SYSTEM***= {'ff\_value': 5, 'string': 'SYSTEM'}*

**UNSPECIFIED***= {'ff\_value': 6, 'string': 'UNSPECIFIED'}*

*class*selenium.webdriver.common.proxy.**ProxyTypeFactory**

Factory for proxy types.

*static***make**(*ff\_value*, *string*)

7.9. Utilities

The Utils methods.

selenium.webdriver.common.utils.**find\_connectable\_ip**(*host*, *port=None*)

Resolve a hostname to an IP, preferring IPv4 addresses.

We prefer IPv4 so that we don’t change behavior from previous IPv4-only implementations, and because some drivers (e.g., FirefoxDriver) do not support IPv6 connections.

If the optional port number is provided, only IPs that listen on the given port are considered.

|  |  |
| --- | --- |
| **Args:** | * host - A hostname. * port - Optional port number. |
| **Returns:** | A single IP address, as a string. If any IPv4 address is found, one is returned. Otherwise, if any IPv6 address is found, one is returned. If neither, then None is returned. |

selenium.webdriver.common.utils.**free\_port**()

Determines a free port using sockets.

selenium.webdriver.common.utils.**is\_connectable**(*port*, *host='localhost'*)

Tries to connect to the server at port to see if it is running.

|  |  |
| --- | --- |
| **Args:** | * port - The port to connect. |

selenium.webdriver.common.utils.**is\_url\_connectable**(*port*)

Tries to connect to the HTTP server at /status path and specified port to see if it responds successfully.

|  |  |
| --- | --- |
| **Args:** | * port - The port to connect. |

selenium.webdriver.common.utils.**join\_host\_port**(*host*, *port*)

Joins a hostname and port together.

This is a minimal implementation intended to cope with IPv6 literals. For example, \_join\_host\_port(‘::1’, 80) == ‘[::1]:80’.

|  |  |
| --- | --- |
| **Args:** | * host - A hostname. * port - An integer port. |

selenium.webdriver.common.utils.**keys\_to\_typing**(*value*)

Processes the values that will be typed in the element.

7.10. Service

*class*selenium.webdriver.common.service.**Service**(*executable*, *port=0*, *log\_file=-3*, *env=None*, *start\_error\_message=''*)

Bases: **object**

**\_\_init\_\_**(*executable*, *port=0*, *log\_file=-3*, *env=None*, *start\_error\_message=''*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**assert\_process\_still\_running**()

**command\_line\_args**()

**is\_connectable**()

**send\_remote\_shutdown\_command**()

**start**()

Starts the Service.

|  |  |
| --- | --- |
| **Exceptions:** | * WebDriverException : Raised either when it can’t start the service or when it can’t connect to the service |

**stop**()

Stops the service.

**service\_url**

Gets the url of the Service

7.11. Application Cache

The ApplicationCache implementaion.

*class*selenium.webdriver.common.html5.application\_cache.**ApplicationCache**(*driver*)

Bases: **object**

**\_\_init\_\_**(*driver*)

Creates a new Aplication Cache.

|  |  |
| --- | --- |
| **Args:** | * driver: The WebDriver instance which performs user actions. |

**CHECKING***= 2*

**DOWNLOADING***= 3*

**IDLE***= 1*

**OBSOLETE***= 5*

**UNCACHED***= 0*

**UPDATE\_READY***= 4*

**status**

Returns a current status of application cache.

7.12. Firefox WebDriver

*class*selenium.webdriver.firefox.webdriver.**WebDriver**(*firefox\_profile=None*, *firefox\_binary=None*, *timeout=30*, *capabilities=None*, *proxy=None*, *executable\_path='geckodriver'*, *options=None*, *service\_log\_path='geckodriver.log'*, *firefox\_options=None*, *service\_args=None*, *desired\_capabilities=None*, *log\_path=None*, *keep\_alive=True*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

**\_\_init\_\_**(*firefox\_profile=None*, *firefox\_binary=None*, *timeout=30*, *capabilities=None*, *proxy=None*, *executable\_path='geckodriver'*, *options=None*, *service\_log\_path='geckodriver.log'*, *firefox\_options=None*, *service\_args=None*, *desired\_capabilities=None*, *log\_path=None*, *keep\_alive=True*)

Starts a new local session of Firefox.

Based on the combination and specificity of the various keyword arguments, a capabilities dictionary will be constructed that is passed to the remote end.

The keyword arguments given to this constructor are helpers to more easily allow Firefox WebDriver sessions to be customised with different options. They are mapped on to a capabilities dictionary that is passed on to the remote end.

As some of the options, such as *firefox\_profile* and *options.profile* are mutually exclusive, precedence is given from how specific the setting is. *capabilities* is the least specific keyword argument, followed by *options*, followed by *firefox\_binary* and *firefox\_profile*.

In practice this means that if *firefox\_profile* and *options.profile* are both set, the selected profile instance will always come from the most specific variable. In this case that would be *firefox\_profile*. This will result in *options.profile* to be ignored because it is considered a less specific setting than the top-level *firefox\_profile* keyword argument. Similarily, if you had specified a *capabilities[“moz:firefoxOptions”][“profile”]* Base64 string, this would rank below *options.profile*.

|  |  |
| --- | --- |
| **Parameters:** | * **firefox\_profile** – Instance of FirefoxProfile object or a string. If undefined, a fresh profile will be created in a temporary location on the system. * **firefox\_binary** – Instance of FirefoxBinary or full path to the Firefox binary. If undefined, the system default Firefox installation will be used. * **timeout** – Time to wait for Firefox to launch when using the extension connection. * **capabilities** – Dictionary of desired capabilities. * **proxy** – The proxy settings to us when communicating with Firefox via the extension connection. * **executable\_path** – Full path to override which geckodriver binary to use for Firefox 47.0.1 and greater, which defaults to picking up the binary from the system path. * **options** – Instance of options.Options. * **service\_log\_path** – Where to log information from the driver. * **firefox\_options** – Deprecated argument for options * **service\_args** – List of args to pass to the driver service * **desired\_capabilities** – alias of capabilities. In future versions of this library, this will replace ‘capabilities’. This will make the signature consistent with RemoteWebDriver. * **log\_path** – Deprecated argument for service\_log\_path * **keep\_alive** – Whether to configure remote\_connection.RemoteConnection to use HTTP keep-alive. |

**context**(*\*\*kwds*)

Sets the context that Selenium commands are running in using a *with* statement. The state of the context on the server is saved before entering the block, and restored upon exiting it.

|  |  |
| --- | --- |
| **Parameters:** | **context** – Context, may be one of the class properties *CONTEXT\_CHROME* or *CONTEXT\_CONTENT*. |

Usage example:

**with** selenium.context(selenium.CONTEXT\_CHROME):

*# chrome scope*

... do stuff ...

**install\_addon**(*path*, *temporary=None*)

Installs Firefox addon.

Returns identifier of installed addon. This identifier can later be used to uninstall addon.

|  |  |
| --- | --- |
| **Parameters:** | **path** – Absolute path to the addon that will be installed. |
| **Usage:** | driver.install\_addon(‘/path/to/firebug.xpi’) |

**quit**()

Quits the driver and close every associated window.

**set\_context**(*context*)

**uninstall\_addon**(*identifier*)

Uninstalls Firefox addon using its identifier.

|  |  |
| --- | --- |
| **Usage:** | driver.uninstall\_addon([‘addon@foo.com](mailto:'addon%40foo.com)’) |

**CONTEXT\_CHROME***= 'chrome'*

**CONTEXT\_CONTENT***= 'content'*

**NATIVE\_EVENTS\_ALLOWED***= True*

**firefox\_profile**

7.13. Firefox WebDriver Options

*class*selenium.webdriver.firefox.options.**Log**

Bases: **object**

**\_\_init\_\_**()

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**to\_capabilities**()

*class*selenium.webdriver.firefox.options.**Options**

Bases: **object**

**\_\_init\_\_**()

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**add\_argument**(*argument*)

Add argument to be used for the browser process.

**set\_capability**(*name*, *value*)

Sets a capability.

**set\_headless**(*headless=True*)

Deprecated, options.headless = True

**set\_preference**(*name*, *value*)

Sets a preference.

**to\_capabilities**()

Marshals the Firefox options to a *moz:firefoxOptions* object.

**KEY***= 'moz:firefoxOptions'*

**accept\_insecure\_certs**

**arguments**

Returns a list of browser process arguments.

**binary**

Returns the FirefoxBinary instance

**binary\_location**

Returns the location of the binary.

**capabilities**

**headless**

Returns whether or not the headless argument is set

**preferences**

Returns a dict of preferences.

**profile**

Returns the Firefox profile to use.

**proxy**

returns Proxy if set otherwise None.

7.14. Firefox WebDriver Profile

*exception*selenium.webdriver.firefox.firefox\_profile.**AddonFormatError**

Bases: **exceptions.Exception**

Exception for not well-formed add-on manifest files

*class*selenium.webdriver.firefox.firefox\_profile.**FirefoxProfile**(*profile\_directory=None*)

Bases: **object**

**\_\_init\_\_**(*profile\_directory=None*)

Initialises a new instance of a Firefox Profile

|  |  |
| --- | --- |
| **Args:** | * profile\_directory: Directory of profile that you want to use. If a directory is passed in it will be cloned and the cloned directory will be used by the driver when instantiated. This defaults to None and will create a new directory when object is created. |

**add\_extension**(*extension='webdriver.xpi'*)

**set\_preference**(*key*, *value*)

sets the preference that we want in the profile.

**set\_proxy**(*proxy*)

**update\_preferences**()

**ANONYMOUS\_PROFILE\_NAME***= 'WEBDRIVER\_ANONYMOUS\_PROFILE'*

**DEFAULT\_PREFERENCES***= None*

**accept\_untrusted\_certs**

**assume\_untrusted\_cert\_issuer**

**encoded**

A zipped, base64 encoded string of profile directory for use with remote WebDriver JSON wire protocol

**native\_events\_enabled**

**path**

Gets the profile directory that is currently being used

**port**

Gets the port that WebDriver is working on

7.15. Firefox WebDriver Binary

*class*selenium.webdriver.firefox.firefox\_binary.**FirefoxBinary**(*firefox\_path=None*, *log\_file=None*)

Bases: **object**

**\_\_init\_\_**(*firefox\_path=None*, *log\_file=None*)

Creates a new instance of Firefox binary.

|  |  |
| --- | --- |
| **Args:** | * firefox\_path - Path to the Firefox executable. By default, it will be detected from the standard locations. * log\_file - A file object to redirect the firefox process output to. It can be sys.stdout.   Please note that with parallel run the output won’t be synchronous. By default, it will be redirected to /dev/null. |

**add\_command\_line\_options**(*\*args*)

**kill**()

Kill the browser.

This is useful when the browser is stuck.

**launch\_browser**(*profile*, *timeout=30*)

Launches the browser for the given profile name. It is assumed the profile already exists.

**which**(*fname*)

Returns the fully qualified path by searching Path of the given name

**NO\_FOCUS\_LIBRARY\_NAME***= 'x\_ignore\_nofocus.so'*

7.16. Firefox WebDriver Extension Connection

*exception*selenium.webdriver.firefox.extension\_connection.**ExtensionConnectionError**

Bases: **exceptions.Exception**

An internal error occurred int the extension.

Might be caused by bad input or bugs in webdriver

*class*selenium.webdriver.firefox.extension\_connection.**ExtensionConnection**(*host*, *firefox\_profile*, *firefox\_binary=None*, *timeout=30*)

Bases: [**selenium.webdriver.remote.remote\_connection.RemoteConnection**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.remote_connection.RemoteConnection)

**\_\_init\_\_**(*host*, *firefox\_profile*, *firefox\_binary=None*, *timeout=30*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**connect**()

Connects to the extension and retrieves the session id.

*classmethod***connect\_and\_quit**()

Connects to an running browser and quit immediately.

*classmethod***is\_connectable**()

Trys to connect to the extension but do not retrieve context.

**quit**(*sessionId=None*)

7.17. Chrome WebDriver

*class*selenium.webdriver.chrome.webdriver.**WebDriver**(*executable\_path='chromedriver'*, *port=0*, *options=None*, *service\_args=None*, *desired\_capabilities=None*, *service\_log\_path=None*, *chrome\_options=None*, *keep\_alive=True*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

Controls the ChromeDriver and allows you to drive the browser.

You will need to download the ChromeDriver executable from <http://chromedriver.storage.googleapis.com/index.html>

**\_\_init\_\_**(*executable\_path='chromedriver'*, *port=0*, *options=None*, *service\_args=None*, *desired\_capabilities=None*, *service\_log\_path=None*, *chrome\_options=None*, *keep\_alive=True*)

Creates a new instance of the chrome driver.

Starts the service and then creates new instance of chrome driver.

|  |  |
| --- | --- |
| **Args:** | * executable\_path - path to the executable. If the default is used it assumes the executable is in the $PATH * port - port you would like the service to run, if left as 0, a free port will be found. * options - this takes an instance of ChromeOptions * service\_args - List of args to pass to the driver service * desired\_capabilities - Dictionary object with non-browser specific capabilities only, such as “proxy” or “loggingPref”. * service\_log\_path - Where to log information from the driver. * chrome\_options - Deprecated argument for options * keep\_alive - Whether to configure ChromeRemoteConnection to use HTTP keep-alive. |

**create\_options**()

**execute\_cdp\_cmd**(*cmd*, *cmd\_args*)

Execute Chrome Devtools Protocol command and get returned result

The command and command args should follow chrome devtools protocol domains/commands, refer to link <https://chromedevtools.github.io/devtools-protocol/>

|  |  |
| --- | --- |
| **Args:** | * cmd: A str, command name * cmd\_args: A dict, command args. empty dict {} if there is no command args |
| **Usage:** | driver.execute\_cdp\_cmd(‘Network.getResponseBody’, {‘requestId’: requestId}) |
| **Returns:** | A dict, empty dict {} if there is no result to return. For example to getResponseBody:  {‘base64Encoded’: False, ‘body’: ‘response body string’} |

**get\_network\_conditions**()

Gets Chrome network emulation settings.

|  |  |
| --- | --- |
| **Returns:** | A dict. For example:  {‘latency’: 4, ‘download\_throughput’: 2, ‘upload\_throughput’: 2, ‘offline’: False} |

**launch\_app**(*id*)

Launches Chrome app specified by id.

**quit**()

Closes the browser and shuts down the ChromeDriver executable that is started when starting the ChromeDriver

**set\_network\_conditions**(*\*\*network\_conditions*)

Sets Chrome network emulation settings.

|  |  |
| --- | --- |
| **Args:** | * network\_conditions: A dict with conditions specification. |
| **Usage:** | driver.set\_network\_conditions(  offline=False, latency=5, # additional latency (ms) download\_throughput=500 \* 1024, # maximal throughput upload\_throughput=500 \* 1024) # maximal throughput  Note: ‘throughput’ can be used to set both (for download and upload). |

7.18. Chrome WebDriver Options

*class*selenium.webdriver.chrome.options.**Options**

Bases: **object**

**\_\_init\_\_**()

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**add\_argument**(*argument*)

Adds an argument to the list

|  |  |
| --- | --- |
| **Args:** | * Sets the arguments |

**add\_encoded\_extension**(*extension*)

Adds Base64 encoded string with extension data to a list that will be used to extract it to the ChromeDriver

|  |  |
| --- | --- |
| **Args:** | * extension: Base64 encoded string with extension data |

**add\_experimental\_option**(*name*, *value*)

Adds an experimental option which is passed to chrome.

Args:

name: The experimental option name. value: The option value.

**add\_extension**(*extension*)

Adds the path to the extension to a list that will be used to extract it to the ChromeDriver

|  |  |
| --- | --- |
| **Args:** | * extension: path to the \*.crx file |

**set\_capability**(*name*, *value*)

Sets a capability.

**set\_headless**(*headless=True*)

Deprecated, options.headless = True

**to\_capabilities**()

Creates a capabilities with all the options that have been set and

returns a dictionary with everything

**KEY***= 'goog:chromeOptions'*

**arguments**

Returns a list of arguments needed for the browser

**binary\_location**

Returns the location of the binary otherwise an empty string

**capabilities**

**debugger\_address**

Returns the address of the remote devtools instance

**experimental\_options**

Returns a dictionary of experimental options for chrome.

**extensions**

Returns a list of encoded extensions that will be loaded into chrome

**headless**

Returns whether or not the headless argument is set

7.19. Chrome WebDriver Service

*class*selenium.webdriver.chrome.service.**Service**(*executable\_path*, *port=0*, *service\_args=None*, *log\_path=None*, *env=None*)

Bases: [**selenium.webdriver.common.service.Service**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.common.service.Service)

Object that manages the starting and stopping of the ChromeDriver

**\_\_init\_\_**(*executable\_path*, *port=0*, *service\_args=None*, *log\_path=None*, *env=None*)

Creates a new instance of the Service

|  |  |
| --- | --- |
| **Args:** | * executable\_path : Path to the ChromeDriver * port : Port the service is running on * service\_args : List of args to pass to the chromedriver service * log\_path : Path for the chromedriver service to log to |

**command\_line\_args**()

7.20. Remote WebDriver

The WebDriver implementation.

*class*selenium.webdriver.remote.webdriver.**WebDriver**(*command\_executor='http://127.0.0.1:4444/wd/hub'*, *desired\_capabilities=None*, *browser\_profile=None*, *proxy=None*, *keep\_alive=False*, *file\_detector=None*, *options=None*)

Bases: **object**

Controls a browser by sending commands to a remote server. This server is expected to be running the WebDriver wire protocol as defined at <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol>

|  |  |
| --- | --- |
| **Attributes:** | * session\_id - String ID of the browser session started and controlled by this WebDriver. * capabilities - Dictionaty of effective capabilities of this browser session as returned   by the remote server. See <https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities>   * command\_executor - remote\_connection.RemoteConnection object used to execute commands. * error\_handler - errorhandler.ErrorHandler object used to handle errors. |

**\_\_init\_\_**(*command\_executor='http://127.0.0.1:4444/wd/hub'*, *desired\_capabilities=None*, *browser\_profile=None*, *proxy=None*, *keep\_alive=False*, *file\_detector=None*, *options=None*)

Create a new driver that will issue commands using the wire protocol.

|  |  |
| --- | --- |
| **Args:** | * command\_executor - Either a string representing URL of the remote server or a custom   remote\_connection.RemoteConnection object. Defaults to ‘<http://127.0.0.1:4444/wd/hub>’.   * desired\_capabilities - A dictionary of capabilities to request when   starting the browser session. Required parameter.   * browser\_profile - A selenium.webdriver.firefox.firefox\_profile.FirefoxProfile object.   Only used if Firefox is requested. Optional.   * proxy - A selenium.webdriver.common.proxy.Proxy object. The browser session will   be started with given proxy settings, if possible. Optional.   * keep\_alive - Whether to configure remote\_connection.RemoteConnection to use   HTTP keep-alive. Defaults to False.   * file\_detector - Pass custom file detector object during instantiation. If None,   then default LocalFileDetector() will be used.   * options - instance of a driver options.Options class |

**add\_cookie**(*cookie\_dict*)

Adds a cookie to your current session.

|  |  |
| --- | --- |
| **Args:** | * cookie\_dict: A dictionary object, with required keys - “name” and “value”;   optional keys - “path”, “domain”, “secure”, “expiry” |

Usage:

driver.add\_cookie({‘name’ : ‘foo’, ‘value’ : ‘bar’}) driver.add\_cookie({‘name’ : ‘foo’, ‘value’ : ‘bar’, ‘path’ : ‘/’}) driver.add\_cookie({‘name’ : ‘foo’, ‘value’ : ‘bar’, ‘path’ : ‘/’, ‘secure’:True})

**back**()

Goes one step backward in the browser history.

|  |  |
| --- | --- |
| **Usage:** | driver.back() |

**close**()

Closes the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.close() |

**create\_web\_element**(*element\_id*)

Creates a web element with the specified *element\_id*.

**delete\_all\_cookies**()

Delete all cookies in the scope of the session.

|  |  |
| --- | --- |
| **Usage:** | driver.delete\_all\_cookies() |

**delete\_cookie**(*name*)

Deletes a single cookie with the given name.

|  |  |
| --- | --- |
| **Usage:** | driver.delete\_cookie(‘my\_cookie’) |

**execute**(*driver\_command*, *params=None*)

Sends a command to be executed by a command.CommandExecutor.

|  |  |
| --- | --- |
| **Args:** | * driver\_command: The name of the command to execute as a string. * params: A dictionary of named parameters to send with the command. |
| **Returns:** | The command’s JSON response loaded into a dictionary object. |

**execute\_async\_script**(*script*, *\*args*)

Asynchronously Executes JavaScript in the current window/frame.

|  |  |
| --- | --- |
| **Args:** | * script: The JavaScript to execute. * \*args: Any applicable arguments for your JavaScript. |
| **Usage:** | script = “var callback = arguments[arguments.length - 1]; ” “window.setTimeout(function(){ callback(‘timeout’) }, 3000);” driver.execute\_async\_script(script) |

**execute\_script**(*script*, *\*args*)

Synchronously Executes JavaScript in the current window/frame.

|  |  |
| --- | --- |
| **Args:** | * script: The JavaScript to execute. * \*args: Any applicable arguments for your JavaScript. |
| **Usage:** | driver.execute\_script(‘return document.title;’) |

**file\_detector\_context**(*\*\*kwds*)

Overrides the current file detector (if necessary) in limited context. Ensures the original file detector is set afterwards.

Example:

with webdriver.file\_detector\_context(UselessFileDetector):

someinput.send\_keys(‘/etc/hosts’)

|  |  |
| --- | --- |
| **Args:** | * file\_detector\_class - Class of the desired file detector. If the class is different   from the current file\_detector, then the class is instantiated with args and kwargs and used as a file detector during the duration of the context manager.   * args - Optional arguments that get passed to the file detector class during   instantiation.   * kwargs - Keyword arguments, passed the same way as args. |

**find\_element**(*by='id'*, *value=None*)

Find an element given a By strategy and locator. Prefer the find\_element\_by\_\* methods when possible.

|  |  |
| --- | --- |
| **Usage:** | element = driver.find\_element(By.ID, ‘foo’) |
| **Return type:** | [WebElement](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webelement.WebElement) |

**find\_element\_by\_class\_name**(*name*)

Finds an element by class name.

|  |  |
| --- | --- |
| **Args:** | * name: The class name of the element to find. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_class\_name(‘foo’) |

**find\_element\_by\_css\_selector**(*css\_selector*)

Finds an element by css selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector - CSS selector string, ex: ‘a.nav#home’ |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_css\_selector(‘#foo’) |

**find\_element\_by\_id**(*id\_*)

Finds an element by id.

|  |  |
| --- | --- |
| **Args:** | * id\_ - The id of the element to be found. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_id(‘foo’) |

**find\_element\_by\_link\_text**(*link\_text*)

Finds an element by link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to be found. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_link\_text(‘Sign In’) |

**find\_element\_by\_name**(*name*)

Finds an element by name.

|  |  |
| --- | --- |
| **Args:** | * name: The name of the element to find. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_name(‘foo’) |

**find\_element\_by\_partial\_link\_text**(*link\_text*)

Finds an element by a partial match of its link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to partially match on. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_partial\_link\_text(‘Sign’) |

**find\_element\_by\_tag\_name**(*name*)

Finds an element by tag name.

|  |  |
| --- | --- |
| **Args:** | * name - name of html tag (eg: h1, a, span) |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_tag\_name(‘h1’) |

**find\_element\_by\_xpath**(*xpath*)

Finds an element by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - The xpath locator of the element to find. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = driver.find\_element\_by\_xpath(‘//div/td[1]’) |

**find\_elements**(*by='id'*, *value=None*)

Find elements given a By strategy and locator. Prefer the find\_elements\_by\_\* methods when possible.

|  |  |
| --- | --- |
| **Usage:** | elements = driver.find\_elements(By.CLASS\_NAME, ‘foo’) |
| **Return type:** | list of WebElement |

**find\_elements\_by\_class\_name**(*name*)

Finds elements by class name.

|  |  |
| --- | --- |
| **Args:** | * name: The class name of the elements to find. |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_class\_name(‘foo’) |

**find\_elements\_by\_css\_selector**(*css\_selector*)

Finds elements by css selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector - CSS selector string, ex: ‘a.nav#home’ |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_css\_selector(‘.foo’) |

**find\_elements\_by\_id**(*id\_*)

Finds multiple elements by id.

|  |  |
| --- | --- |
| **Args:** | * id\_ - The id of the elements to be found. |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_id(‘foo’) |

**find\_elements\_by\_link\_text**(*text*)

Finds elements by link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the elements to be found. |
| **Returns:** | * list of webelement - a list with elements if any was found. an empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_link\_text(‘Sign In’) |

**find\_elements\_by\_name**(*name*)

Finds elements by name.

|  |  |
| --- | --- |
| **Args:** | * name: The name of the elements to find. |
| **Returns:** | * list of webelement - a list with elements if any was found. an empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_name(‘foo’) |

**find\_elements\_by\_partial\_link\_text**(*link\_text*)

Finds elements by a partial match of their link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to partial match on. |
| **Returns:** | * list of webelement - a list with elements if any was found. an empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_partial\_link\_text(‘Sign’) |

**find\_elements\_by\_tag\_name**(*name*)

Finds elements by tag name.

|  |  |
| --- | --- |
| **Args:** | * name - name of html tag (eg: h1, a, span) |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_tag\_name(‘h1’) |

**find\_elements\_by\_xpath**(*xpath*)

Finds multiple elements by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - The xpath locator of the elements to be found. |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = driver.find\_elements\_by\_xpath(“//div[contains(@class, ‘foo’)]”) |

**forward**()

Goes one step forward in the browser history.

|  |  |
| --- | --- |
| **Usage:** | driver.forward() |

**fullscreen\_window**()

Invokes the window manager-specific ‘full screen’ operation

**get**(*url*)

Loads a web page in the current browser session.

**get\_cookie**(*name*)

Get a single cookie by name. Returns the cookie if found, None if not.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_cookie(‘my\_cookie’) |

**get\_cookies**()

Returns a set of dictionaries, corresponding to cookies visible in the current session.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_cookies() |

**get\_log**(*log\_type*)

Gets the log for a given log type

|  |  |
| --- | --- |
| **Args:** | * log\_type: type of log that which will be returned |
| **Usage:** | driver.get\_log(‘browser’) driver.get\_log(‘driver’) driver.get\_log(‘client’) driver.get\_log(‘server’) |

**get\_screenshot\_as\_base64**()

Gets the screenshot of the current window as a base64 encoded string

which is useful in embedded images in HTML.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_screenshot\_as\_base64() |

**get\_screenshot\_as\_file**(*filename*)

Saves a screenshot of the current window to a PNG image file. Returns

False if there is any IOError, else returns True. Use full paths in your filename.

|  |  |
| --- | --- |
| **Args:** | * filename: The full path you wish to save your screenshot to. This should end with a *.png* extension. |
| **Usage:** | driver.get\_screenshot\_as\_file(‘/Screenshots/foo.png’) |

**get\_screenshot\_as\_png**()

Gets the screenshot of the current window as a binary data.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_screenshot\_as\_png() |

**get\_window\_position**(*windowHandle='current'*)

Gets the x,y position of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_window\_position() |

**get\_window\_rect**()

Gets the x, y coordinates of the window as well as height and width of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_window\_rect() |

**get\_window\_size**(*windowHandle='current'*)

Gets the width and height of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_window\_size() |

**implicitly\_wait**(*time\_to\_wait*)

Sets a sticky timeout to implicitly wait for an element to be found,

or a command to complete. This method only needs to be called one time per session. To set the timeout for calls to execute\_async\_script, see set\_script\_timeout.

|  |  |
| --- | --- |
| **Args:** | * time\_to\_wait: Amount of time to wait (in seconds) |
| **Usage:** | driver.implicitly\_wait(30) |

**maximize\_window**()

Maximizes the current window that webdriver is using

**minimize\_window**()

Invokes the window manager-specific ‘minimize’ operation

**quit**()

Quits the driver and closes every associated window.

|  |  |
| --- | --- |
| **Usage:** | driver.quit() |

**refresh**()

Refreshes the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.refresh() |

**save\_screenshot**(*filename*)

Saves a screenshot of the current window to a PNG image file. Returns

False if there is any IOError, else returns True. Use full paths in your filename.

|  |  |
| --- | --- |
| **Args:** | * filename: The full path you wish to save your screenshot to. This should end with a *.png* extension. |
| **Usage:** | driver.save\_screenshot(‘/Screenshots/foo.png’) |

**set\_page\_load\_timeout**(*time\_to\_wait*)

Set the amount of time to wait for a page load to complete

before throwing an error.

|  |  |
| --- | --- |
| **Args:** | * time\_to\_wait: The amount of time to wait |
| **Usage:** | driver.set\_page\_load\_timeout(30) |

**set\_script\_timeout**(*time\_to\_wait*)

Set the amount of time that the script should wait during an

execute\_async\_script call before throwing an error.

|  |  |
| --- | --- |
| **Args:** | * time\_to\_wait: The amount of time to wait (in seconds) |
| **Usage:** | driver.set\_script\_timeout(30) |

**set\_window\_position**(*x*, *y*, *windowHandle='current'*)

Sets the x,y position of the current window. (window.moveTo)

|  |  |
| --- | --- |
| **Args:** | * x: the x-coordinate in pixels to set the window position * y: the y-coordinate in pixels to set the window position |
| **Usage:** | driver.set\_window\_position(0,0) |

**set\_window\_rect**(*x=None*, *y=None*, *width=None*, *height=None*)

Sets the x, y coordinates of the window as well as height and width of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.set\_window\_rect(x=10, y=10) driver.set\_window\_rect(width=100, height=200) driver.set\_window\_rect(x=10, y=10, width=100, height=200) |

**set\_window\_size**(*width*, *height*, *windowHandle='current'*)

Sets the width and height of the current window. (window.resizeTo)

|  |  |
| --- | --- |
| **Args:** | * width: the width in pixels to set the window to * height: the height in pixels to set the window to |
| **Usage:** | driver.set\_window\_size(800,600) |

**start\_client**()

Called before starting a new session. This method may be overridden to define custom startup behavior.

**start\_session**(*capabilities*, *browser\_profile=None*)

Creates a new session with the desired capabilities.

|  |  |
| --- | --- |
| **Args:** | * browser\_name - The name of the browser to request. * version - Which browser version to request. * platform - Which platform to request the browser on. * javascript\_enabled - Whether the new session should support JavaScript. * browser\_profile - A selenium.webdriver.firefox.firefox\_profile.FirefoxProfile object. Only used if Firefox is requested. |

**stop\_client**()

Called after executing a quit command. This method may be overridden to define custom shutdown behavior.

**switch\_to\_active\_element**()

Deprecated use driver.switch\_to.active\_element

**switch\_to\_alert**()

Deprecated use driver.switch\_to.alert

**switch\_to\_default\_content**()

Deprecated use driver.switch\_to.default\_content

**switch\_to\_frame**(*frame\_reference*)

Deprecated use driver.switch\_to.frame

**switch\_to\_window**(*window\_name*)

Deprecated use driver.switch\_to.window

**application\_cache**

Returns a ApplicationCache Object to interact with the browser app cache

**current\_url**

Gets the URL of the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.current\_url |

**current\_window\_handle**

Returns the handle of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.current\_window\_handle |

**desired\_capabilities**

returns the drivers current desired capabilities being used

**file\_detector**

**log\_types**

Gets a list of the available log types

|  |  |
| --- | --- |
| **Usage:** | driver.log\_types |

**mobile**

**name**

Returns the name of the underlying browser for this instance.

|  |  |
| --- | --- |
| **Usage:** | name = driver.name |

**orientation**

Gets the current orientation of the device

|  |  |
| --- | --- |
| **Usage:** | orientation = driver.orientation |

**page\_source**

Gets the source of the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.page\_source |

**switch\_to**

|  |  |
| --- | --- |
| **Returns:** | * SwitchTo: an object containing all options to switch focus into |
| **Usage:** | element = driver.switch\_to.active\_element alert = driver.switch\_to.alert driver.switch\_to.default\_content() driver.switch\_to.frame(‘frame\_name’) driver.switch\_to.frame(1) driver.switch\_to.frame(driver.find\_elements\_by\_tag\_name(“iframe”)[0]) driver.switch\_to.parent\_frame() driver.switch\_to.window(‘main’) |

**title**

Returns the title of the current page.

|  |  |
| --- | --- |
| **Usage:** | title = driver.title |

**window\_handles**

Returns the handles of all windows within the current session.

|  |  |
| --- | --- |
| **Usage:** | driver.window\_handles |

7.21. Remote WebDriver WebElement

*class*selenium.webdriver.remote.webelement.**WebElement**(*parent*, *id\_*, *w3c=False*)

Bases: **object**

Represents a DOM element.

Generally, all interesting operations that interact with a document will be performed through this interface.

All method calls will do a freshness check to ensure that the element reference is still valid. This essentially determines whether or not the element is still attached to the DOM. If this test fails, then an StaleElementReferenceException is thrown, and all future calls to this instance will fail.

**\_\_init\_\_**(*parent*, *id\_*, *w3c=False*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**clear**()

Clears the text if it’s a text entry element.

**click**()

Clicks the element.

**find\_element**(*by='id'*, *value=None*)

Find an element given a By strategy and locator. Prefer the find\_element\_by\_\* methods when possible.

|  |  |
| --- | --- |
| **Usage:** | element = element.find\_element(By.ID, ‘foo’) |
| **Return type:** | [WebElement](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webelement.WebElement) |

**find\_element\_by\_class\_name**(*name*)

Finds element within this element’s children by class name.

|  |  |
| --- | --- |
| **Args:** | * name: The class name of the element to find. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_class\_name(‘foo’) |

**find\_element\_by\_css\_selector**(*css\_selector*)

Finds element within this element’s children by CSS selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector - CSS selector string, ex: ‘a.nav#home’ |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_css\_selector(‘#foo’) |

**find\_element\_by\_id**(*id\_*)

Finds element within this element’s children by ID.

|  |  |
| --- | --- |
| **Args:** | * id\_ - ID of child element to locate. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | foo\_element = element.find\_element\_by\_id(‘foo’) |

**find\_element\_by\_link\_text**(*link\_text*)

Finds element within this element’s children by visible link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text - Link text string to search for. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_link\_text(‘Sign In’) |

**find\_element\_by\_name**(*name*)

Finds element within this element’s children by name.

|  |  |
| --- | --- |
| **Args:** | * name - name property of the element to find. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_name(‘foo’) |

**find\_element\_by\_partial\_link\_text**(*link\_text*)

Finds element within this element’s children by partially visible link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to partially match on. |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_partial\_link\_text(‘Sign’) |

**find\_element\_by\_tag\_name**(*name*)

Finds element within this element’s children by tag name.

|  |  |
| --- | --- |
| **Args:** | * name - name of html tag (eg: h1, a, span) |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_tag\_name(‘h1’) |

**find\_element\_by\_xpath**(*xpath*)

Finds element by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - xpath of element to locate. “//input[@class=’myelement’]” |

Note: The base path will be relative to this element’s location.

This will select the first link under this element.

myelement.find\_element\_by\_xpath(".//a")

However, this will select the first link on the page.

myelement.find\_element\_by\_xpath("//a")

|  |  |
| --- | --- |
| **Returns:** | * WebElement - the element if it was found |
| **Raises:** | * NoSuchElementException - if the element wasn’t found |
| **Usage:** | element = element.find\_element\_by\_xpath(‘//div/td[1]’) |

**find\_elements**(*by='id'*, *value=None*)

Find elements given a By strategy and locator. Prefer the find\_elements\_by\_\* methods when possible.

|  |  |
| --- | --- |
| **Usage:** | element = element.find\_elements(By.CLASS\_NAME, ‘foo’) |
| **Return type:** | list of WebElement |

**find\_elements\_by\_class\_name**(*name*)

Finds a list of elements within this element’s children by class name.

|  |  |
| --- | --- |
| **Args:** | * name: The class name of the elements to find. |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_class\_name(‘foo’) |

**find\_elements\_by\_css\_selector**(*css\_selector*)

Finds a list of elements within this element’s children by CSS selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector - CSS selector string, ex: ‘a.nav#home’ |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_css\_selector(‘.foo’) |

**find\_elements\_by\_id**(*id\_*)

Finds a list of elements within this element’s children by ID. Will return a list of webelements if found, or an empty list if not.

|  |  |
| --- | --- |
| **Args:** | * id\_ - Id of child element to find. |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_id(‘foo’) |

**find\_elements\_by\_link\_text**(*link\_text*)

Finds a list of elements within this element’s children by visible link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text - Link text string to search for. |
| **Returns:** | * list of webelement - a list with elements if any was found. an empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_link\_text(‘Sign In’) |

**find\_elements\_by\_name**(*name*)

Finds a list of elements within this element’s children by name.

|  |  |
| --- | --- |
| **Args:** | * name - name property to search for. |
| **Returns:** | * list of webelement - a list with elements if any was found. an empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_name(‘foo’) |

**find\_elements\_by\_partial\_link\_text**(*link\_text*)

Finds a list of elements within this element’s children by link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to partial match on. |
| **Returns:** | * list of webelement - a list with elements if any was found. an empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_partial\_link\_text(‘Sign’) |

**find\_elements\_by\_tag\_name**(*name*)

Finds a list of elements within this element’s children by tag name.

|  |  |
| --- | --- |
| **Args:** | * name - name of html tag (eg: h1, a, span) |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_tag\_name(‘h1’) |

**find\_elements\_by\_xpath**(*xpath*)

Finds elements within the element by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - xpath locator string. |

Note: The base path will be relative to this element’s location.

This will select all links under this element.

myelement.find\_elements\_by\_xpath(".//a")

However, this will select all links in the page itself.

myelement.find\_elements\_by\_xpath("//a")

|  |  |
| --- | --- |
| **Returns:** | * list of WebElement - a list with elements if any was found. An empty list if not |
| **Usage:** | elements = element.find\_elements\_by\_xpath(“//div[contains(@class, ‘foo’)]”) |

**get\_attribute**(*name*)

Gets the given attribute or property of the element.

This method will first try to return the value of a property with the given name. If a property with that name doesn’t exist, it returns the value of the attribute with the same name. If there’s no attribute with that name, None is returned.

Values which are considered truthy, that is equals “true” or “false”, are returned as booleans. All other non-None values are returned as strings. For attributes or properties which do not exist, None is returned.

|  |  |
| --- | --- |
| **Args:** | * name - Name of the attribute/property to retrieve. |

Example:

*# Check if the "active" CSS class is applied to an element.*

is\_active = "active" **in** target\_element.get\_attribute("class")

**get\_property**(*name*)

Gets the given property of the element.

|  |  |
| --- | --- |
| **Args:** | * name - Name of the property to retrieve. |

Example:

text\_length = target\_element.get\_property("text\_length")

**is\_displayed**()

Whether the element is visible to a user.

**is\_enabled**()

Returns whether the element is enabled.

**is\_selected**()

Returns whether the element is selected.

Can be used to check if a checkbox or radio button is selected.

**screenshot**(*filename*)

Saves a screenshot of the current element to a PNG image file. Returns

False if there is any IOError, else returns True. Use full paths in your filename.

|  |  |
| --- | --- |
| **Args:** | * filename: The full path you wish to save your screenshot to. This should end with a *.png* extension. |
| **Usage:** | element.screenshot(‘/Screenshots/foo.png’) |

**send\_keys**(*\*value*)

Simulates typing into the element.

|  |  |
| --- | --- |
| **Args:** | * value - A string for typing, or setting form fields. For setting file inputs, this could be a local file path. |

Use this to send simple key events or to fill out form fields:

form\_textfield = driver.find\_element\_by\_name('username')

form\_textfield.send\_keys("admin")

This can also be used to set file inputs.

file\_input = driver.find\_element\_by\_name('profilePic')

file\_input.send\_keys("path/to/profilepic.gif")

*# Generally it's better to wrap the file path in one of the methods*

*# in os.path to return the actual path to support cross OS testing.*

*# file\_input.send\_keys(os.path.abspath("path/to/profilepic.gif"))*

**submit**()

Submits a form.

**value\_of\_css\_property**(*property\_name*)

The value of a CSS property.

**id**

Internal ID used by selenium.

This is mainly for internal use. Simple use cases such as checking if 2 webelements refer to the same element, can be done using ==:

**if** element1 == element2:

print("These 2 are equal")

**location**

The location of the element in the renderable canvas.

**location\_once\_scrolled\_into\_view**

THIS PROPERTY MAY CHANGE WITHOUT WARNING. Use this to discover where on the screen an element is so that we can click it. This method should cause the element to be scrolled into view.

Returns the top lefthand corner location on the screen, or None if the element is not visible.

**parent**

Internal reference to the WebDriver instance this element was found from.

**rect**

A dictionary with the size and location of the element.

**screenshot\_as\_base64**

Gets the screenshot of the current element as a base64 encoded string.

|  |  |
| --- | --- |
| **Usage:** | img\_b64 = element.screenshot\_as\_base64 |

**screenshot\_as\_png**

Gets the screenshot of the current element as a binary data.

|  |  |
| --- | --- |
| **Usage:** | element\_png = element.screenshot\_as\_png |

**size**

The size of the element.

**tag\_name**

This element’s tagName property.

**text**

The text of the element.

7.22. Remote WebDriver Command

*class*selenium.webdriver.remote.command.**Command**

Bases: **object**

Defines constants for the standard WebDriver commands.

While these constants have no meaning in and of themselves, they are used to marshal commands through a service that implements WebDriver’s remote wire protocol:

<https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol>

**ACCEPT\_ALERT***= 'acceptAlert'*

**ADD\_COOKIE***= 'addCookie'*

**CLEAR\_APP\_CACHE***= 'clearAppCache'*

**CLEAR\_ELEMENT***= 'clearElement'*

**CLEAR\_LOCAL\_STORAGE***= 'clearLocalStorage'*

**CLEAR\_SESSION\_STORAGE***= 'clearSessionStorage'*

**CLICK***= 'mouseClick'*

**CLICK\_ELEMENT***= 'clickElement'*

**CLOSE***= 'close'*

**CONTEXT\_HANDLES***= 'getContextHandles'*

**CURRENT\_CONTEXT\_HANDLE***= 'getCurrentContextHandle'*

**DELETE\_ALL\_COOKIES***= 'deleteAllCookies'*

**DELETE\_COOKIE***= 'deleteCookie'*

**DELETE\_SESSION***= 'deleteSession'*

**DISMISS\_ALERT***= 'dismissAlert'*

**DOUBLE\_CLICK***= 'mouseDoubleClick'*

**DOUBLE\_TAP***= 'touchDoubleTap'*

**ELEMENT\_SCREENSHOT***= 'elementScreenshot'*

**EXECUTE\_ASYNC\_SCRIPT***= 'executeAsyncScript'*

**EXECUTE\_SCRIPT***= 'executeScript'*

**EXECUTE\_SQL***= 'executeSql'*

**FIND\_CHILD\_ELEMENT***= 'findChildElement'*

**FIND\_CHILD\_ELEMENTS***= 'findChildElements'*

**FIND\_ELEMENT***= 'findElement'*

**FIND\_ELEMENTS***= 'findElements'*

**FLICK***= 'touchFlick'*

**FULLSCREEN\_WINDOW***= 'fullscreenWindow'*

**GET***= 'get'*

**GET\_ACTIVE\_ELEMENT***= 'getActiveElement'*

**GET\_ALERT\_TEXT***= 'getAlertText'*

**GET\_ALL\_COOKIES***= 'getCookies'*

**GET\_ALL\_SESSIONS***= 'getAllSessions'*

**GET\_APP\_CACHE***= 'getAppCache'*

**GET\_APP\_CACHE\_STATUS***= 'getAppCacheStatus'*

**GET\_AVAILABLE\_LOG\_TYPES***= 'getAvailableLogTypes'*

**GET\_COOKIE***= 'getCookie'*

**GET\_CURRENT\_URL***= 'getCurrentUrl'*

**GET\_CURRENT\_WINDOW\_HANDLE***= 'getCurrentWindowHandle'*

**GET\_ELEMENT\_ATTRIBUTE***= 'getElementAttribute'*

**GET\_ELEMENT\_LOCATION***= 'getElementLocation'*

**GET\_ELEMENT\_LOCATION\_ONCE\_SCROLLED\_INTO\_VIEW***= 'getElementLocationOnceScrolledIntoView'*

**GET\_ELEMENT\_PROPERTY***= 'getElementProperty'*

**GET\_ELEMENT\_RECT***= 'getElementRect'*

**GET\_ELEMENT\_SIZE***= 'getElementSize'*

**GET\_ELEMENT\_TAG\_NAME***= 'getElementTagName'*

**GET\_ELEMENT\_TEXT***= 'getElementText'*

**GET\_ELEMENT\_VALUE***= 'getElementValue'*

**GET\_ELEMENT\_VALUE\_OF\_CSS\_PROPERTY***= 'getElementValueOfCssProperty'*

**GET\_LOCAL\_STORAGE\_ITEM***= 'getLocalStorageItem'*

**GET\_LOCAL\_STORAGE\_KEYS***= 'getLocalStorageKeys'*

**GET\_LOCAL\_STORAGE\_SIZE***= 'getLocalStorageSize'*

**GET\_LOCATION***= 'getLocation'*

**GET\_LOG***= 'getLog'*

**GET\_NETWORK\_CONNECTION***= 'getNetworkConnection'*

**GET\_PAGE\_SOURCE***= 'getPageSource'*

**GET\_SCREEN\_ORIENTATION***= 'getScreenOrientation'*

**GET\_SESSION\_STORAGE\_ITEM***= 'getSessionStorageItem'*

**GET\_SESSION\_STORAGE\_KEYS***= 'getSessionStorageKeys'*

**GET\_SESSION\_STORAGE\_SIZE***= 'getSessionStorageSize'*

**GET\_TITLE***= 'getTitle'*

**GET\_WINDOW\_HANDLES***= 'getWindowHandles'*

**GET\_WINDOW\_POSITION***= 'getWindowPosition'*

**GET\_WINDOW\_RECT***= 'getWindowRect'*

**GET\_WINDOW\_SIZE***= 'getWindowSize'*

**GO\_BACK***= 'goBack'*

**GO\_FORWARD***= 'goForward'*

**IMPLICIT\_WAIT***= 'implicitlyWait'*

**IS\_ELEMENT\_DISPLAYED***= 'isElementDisplayed'*

**IS\_ELEMENT\_ENABLED***= 'isElementEnabled'*

**IS\_ELEMENT\_SELECTED***= 'isElementSelected'*

**LONG\_PRESS***= 'touchLongPress'*

**MAXIMIZE\_WINDOW***= 'windowMaximize'*

**MINIMIZE\_WINDOW***= 'minimizeWindow'*

**MOUSE\_DOWN***= 'mouseButtonDown'*

**MOUSE\_UP***= 'mouseButtonUp'*

**MOVE\_TO***= 'mouseMoveTo'*

**NEW\_SESSION***= 'newSession'*

**QUIT***= 'quit'*

**REFRESH***= 'refresh'*

**REMOVE\_LOCAL\_STORAGE\_ITEM***= 'removeLocalStorageItem'*

**REMOVE\_SESSION\_STORAGE\_ITEM***= 'removeSessionStorageItem'*

**SCREENSHOT***= 'screenshot'*

**SEND\_KEYS\_TO\_ACTIVE\_ELEMENT***= 'sendKeysToActiveElement'*

**SEND\_KEYS\_TO\_ELEMENT***= 'sendKeysToElement'*

**SET\_ALERT\_CREDENTIALS***= 'setAlertCredentials'*

**SET\_ALERT\_VALUE***= 'setAlertValue'*

**SET\_ELEMENT\_SELECTED***= 'setElementSelected'*

**SET\_LOCAL\_STORAGE\_ITEM***= 'setLocalStorageItem'*

**SET\_LOCATION***= 'setLocation'*

**SET\_NETWORK\_CONNECTION***= 'setNetworkConnection'*

**SET\_SCREEN\_ORIENTATION***= 'setScreenOrientation'*

**SET\_SCRIPT\_TIMEOUT***= 'setScriptTimeout'*

**SET\_SESSION\_STORAGE\_ITEM***= 'setSessionStorageItem'*

**SET\_TIMEOUTS***= 'setTimeouts'*

**SET\_WINDOW\_POSITION***= 'setWindowPosition'*

**SET\_WINDOW\_RECT***= 'setWindowRect'*

**SET\_WINDOW\_SIZE***= 'setWindowSize'*

**SINGLE\_TAP***= 'touchSingleTap'*

**STATUS***= 'status'*

**SUBMIT\_ELEMENT***= 'submitElement'*

**SWITCH\_TO\_CONTEXT***= 'switchToContext'*

**SWITCH\_TO\_FRAME***= 'switchToFrame'*

**SWITCH\_TO\_PARENT\_FRAME***= 'switchToParentFrame'*

**SWITCH\_TO\_WINDOW***= 'switchToWindow'*

**TOUCH\_DOWN***= 'touchDown'*

**TOUCH\_MOVE***= 'touchMove'*

**TOUCH\_SCROLL***= 'touchScroll'*

**TOUCH\_UP***= 'touchUp'*

**UPLOAD\_FILE***= 'uploadFile'*

**W3C\_ACCEPT\_ALERT***= 'w3cAcceptAlert'*

**W3C\_ACTIONS***= 'actions'*

**W3C\_CLEAR\_ACTIONS***= 'clearActionState'*

**W3C\_DISMISS\_ALERT***= 'w3cDismissAlert'*

**W3C\_EXECUTE\_SCRIPT***= 'w3cExecuteScript'*

**W3C\_EXECUTE\_SCRIPT\_ASYNC***= 'w3cExecuteScriptAsync'*

**W3C\_GET\_ACTIVE\_ELEMENT***= 'w3cGetActiveElement'*

**W3C\_GET\_ALERT\_TEXT***= 'w3cGetAlertText'*

**W3C\_GET\_CURRENT\_WINDOW\_HANDLE***= 'w3cGetCurrentWindowHandle'*

**W3C\_GET\_WINDOW\_HANDLES***= 'w3cGetWindowHandles'*

**W3C\_GET\_WINDOW\_POSITION***= 'w3cGetWindowPosition'*

**W3C\_GET\_WINDOW\_SIZE***= 'w3cGetWindowSize'*

**W3C\_MAXIMIZE\_WINDOW***= 'w3cMaximizeWindow'*

**W3C\_SET\_ALERT\_VALUE***= 'w3cSetAlertValue'*

**W3C\_SET\_WINDOW\_POSITION***= 'w3cSetWindowPosition'*

**W3C\_SET\_WINDOW\_SIZE***= 'w3cSetWindowSize'*

7.23. Remote WebDriver Error Handler

*class*selenium.webdriver.remote.errorhandler.**ErrorCode**

Bases: **object**

Error codes defined in the WebDriver wire protocol.

**ELEMENT\_CLICK\_INTERCEPTED***= [64, 'element click intercepted']*

**ELEMENT\_IS\_NOT\_SELECTABLE***= [15, 'element not selectable']*

**ELEMENT\_NOT\_INTERACTABLE***= [60, 'element not interactable']*

**ELEMENT\_NOT\_VISIBLE***= [11, 'element not visible']*

**IME\_ENGINE\_ACTIVATION\_FAILED***= [31, 'ime engine activation failed']*

**IME\_NOT\_AVAILABLE***= [30, 'ime not available']*

**INSECURE\_CERTIFICATE***= ['insecure certificate']*

**INVALID\_ARGUMENT***= [61, 'invalid argument']*

**INVALID\_COOKIE\_DOMAIN***= [24, 'invalid cookie domain']*

**INVALID\_COORDINATES***= ['invalid coordinates']*

**INVALID\_ELEMENT\_COORDINATES***= [29, 'invalid element coordinates']*

**INVALID\_ELEMENT\_STATE***= [12, 'invalid element state']*

**INVALID\_SELECTOR***= [32, 'invalid selector']*

**INVALID\_SESSION\_ID***= ['invalid session id']*

**INVALID\_XPATH\_SELECTOR***= [51, 'invalid selector']*

**INVALID\_XPATH\_SELECTOR\_RETURN\_TYPER***= [52, 'invalid selector']*

**JAVASCRIPT\_ERROR***= [17, 'javascript error']*

**METHOD\_NOT\_ALLOWED***= [405, 'unsupported operation']*

**MOVE\_TARGET\_OUT\_OF\_BOUNDS***= [34, 'move target out of bounds']*

**NO\_ALERT\_OPEN***= [27, 'no such alert']*

**NO\_SUCH\_COOKIE***= [62, 'no such cookie']*

**NO\_SUCH\_ELEMENT***= [7, 'no such element']*

**NO\_SUCH\_FRAME***= [8, 'no such frame']*

**NO\_SUCH\_WINDOW***= [23, 'no such window']*

**SCRIPT\_TIMEOUT***= [28, 'script timeout']*

**SESSION\_NOT\_CREATED***= [33, 'session not created']*

**STALE\_ELEMENT\_REFERENCE***= [10, 'stale element reference']*

**SUCCESS***= 0*

**TIMEOUT***= [21, 'timeout']*

**UNABLE\_TO\_CAPTURE\_SCREEN***= [63, 'unable to capture screen']*

**UNABLE\_TO\_SET\_COOKIE***= [25, 'unable to set cookie']*

**UNEXPECTED\_ALERT\_OPEN***= [26, 'unexpected alert open']*

**UNKNOWN\_COMMAND***= [9, 'unknown command']*

**UNKNOWN\_ERROR***= [13, 'unknown error']*

**UNKNOWN\_METHOD***= ['unknown method exception']*

**XPATH\_LOOKUP\_ERROR***= [19, 'invalid selector']*

*class*selenium.webdriver.remote.errorhandler.**ErrorHandler**

Bases: **object**

Handles errors returned by the WebDriver server.

**check\_response**(*response*)

Checks that a JSON response from the WebDriver does not have an error.

|  |  |
| --- | --- |
| **Args:** | * response - The JSON response from the WebDriver server as a dictionary object. |
| **Raises:** | If the response contains an error message. |

7.24. Remote WebDriver Mobile

*class*selenium.webdriver.remote.mobile.**Mobile**(*driver*)

Bases: **object**

*class***ConnectionType**(*mask*)

Bases: **object**

**\_\_init\_\_**(*mask*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**airplane\_mode**

**data**

**wifi**

**\_\_init\_\_**(*driver*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**set\_network\_connection**(*network*)

Set the network connection for the remote device.

Example of setting airplane mode:

driver.mobile.set\_network\_connection(driver.mobile.AIRPLANE\_MODE)

**AIRPLANE\_MODE***= <selenium.webdriver.remote.mobile.ConnectionType object>*

**ALL\_NETWORK***= <selenium.webdriver.remote.mobile.ConnectionType object>*

**DATA\_NETWORK***= <selenium.webdriver.remote.mobile.ConnectionType object>*

**WIFI\_NETWORK***= <selenium.webdriver.remote.mobile.ConnectionType object>*

**context**

returns the current context (Native or WebView).

**contexts**

returns a list of available contexts

**network\_connection**

7.25. Remote WebDriver Remote Connection

*class*selenium.webdriver.remote.remote\_connection.**RemoteConnection**(*remote\_server\_addr*, *keep\_alive=False*, *resolve\_ip=True*)

Bases: **object**

A connection with the Remote WebDriver server.

Communicates with the server using the WebDriver wire protocol: <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol>

**\_\_init\_\_**(*remote\_server\_addr*, *keep\_alive=False*, *resolve\_ip=True*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

**execute**(*command*, *params*)

Send a command to the remote server.

Any path subtitutions required for the URL mapped to the command should be included in the command parameters.

|  |  |
| --- | --- |
| **Args:** | * command - A string specifying the command to execute. * params - A dictionary of named parameters to send with the command as its JSON payload. |

*classmethod***get\_remote\_connection\_headers**(*parsed\_url*, *keep\_alive=False*)

Get headers for remote request.

|  |  |
| --- | --- |
| **Args:** | * parsed\_url - The parsed url * keep\_alive (Boolean) - Is this a keep-alive connection (default: False) |

*classmethod***get\_timeout**()

|  |  |
| --- | --- |
| **Returns:** | Timeout value in seconds for all http requests made to the Remote Connection |

*classmethod***reset\_timeout**()

Reset the http request timeout to socket.\_GLOBAL\_DEFAULT\_TIMEOUT

*classmethod***set\_timeout**(*timeout*)

Override the default timeout

|  |  |
| --- | --- |
| **Args:** | * timeout - timeout value for http requests in seconds |

7.26. Remote WebDriver Utils

selenium.webdriver.remote.utils.**dump\_json**(*json\_struct*)

selenium.webdriver.remote.utils.**format\_json**(*json\_struct*)

selenium.webdriver.remote.utils.**load\_json**(*s*)

selenium.webdriver.remote.utils.**unzip\_to\_temp\_dir**(*zip\_file\_name*)

Unzip zipfile to a temporary directory.

The directory of the unzipped files is returned if success, otherwise None is returned.

7.27. Internet Explorer WebDriver

*class*selenium.webdriver.ie.webdriver.**WebDriver**(*executable\_path='IEDriverServer.exe'*, *capabilities=None*, *port=0*, *timeout=30*, *host=None*, *log\_level=None*, *service\_log\_path=None*, *options=None*, *ie\_options=None*, *desired\_capabilities=None*, *log\_file=None*, *keep\_alive=False*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

Controls the IEServerDriver and allows you to drive Internet Explorer

**\_\_init\_\_**(*executable\_path='IEDriverServer.exe'*, *capabilities=None*, *port=0*, *timeout=30*, *host=None*, *log\_level=None*, *service\_log\_path=None*, *options=None*, *ie\_options=None*, *desired\_capabilities=None*, *log\_file=None*, *keep\_alive=False*)

Creates a new instance of the chrome driver.

Starts the service and then creates new instance of chrome driver.

|  |  |
| --- | --- |
| **Args:** | * executable\_path - path to the executable. If the default is used it assumes the executable is in the $PATH * capabilities: capabilities Dictionary object * port - port you would like the service to run, if left as 0, a free port will be found. * timeout - no longer used, kept for backward compatibility * host - IP address for the service * log\_level - log level you would like the service to run. * service\_log\_path - target of logging of service, may be “stdout”, “stderr” or file path. * options - IE Options instance, providing additional IE options * ie\_options - Deprecated argument for options * desired\_capabilities - alias of capabilities; this will make the signature consistent with RemoteWebDriver. * log\_file - Deprecated argument for service\_log\_path * keep\_alive - Whether to configure RemoteConnection to use HTTP keep-alive. |

**create\_options**()

**quit**()

Quits the driver and closes every associated window.

|  |  |
| --- | --- |
| **Usage:** | driver.quit() |

7.28. Android WebDriver

*class*selenium.webdriver.android.webdriver.**WebDriver**(*host='localhost'*, *port=4444*, *desired\_capabilities={'browserName': 'android'*, *'platform': 'ANDROID'*, *'version': ''}*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

Simple RemoteWebDriver wrapper to start connect to Selendroid’s WebView app

For more info on getting started with Selendroid <http://selendroid.io/mobileWeb.html>

**\_\_init\_\_**(*host='localhost'*, *port=4444*, *desired\_capabilities={'browserName': 'android'*, *'platform': 'ANDROID'*, *'version': ''}*)

Creates a new instance of Selendroid using the WebView app

|  |  |
| --- | --- |
| **Args:** | * host - location of where selendroid is running * port - port that selendroid is running on * desired\_capabilities: Dictionary object with capabilities |

7.29. Opera WebDriver

*class*selenium.webdriver.opera.webdriver.**OperaDriver**(*executable\_path=None*, *port=0*, *options=None*, *service\_args=None*, *desired\_capabilities=None*, *service\_log\_path=None*, *opera\_options=None*, *keep\_alive=True*)

Bases: [**selenium.webdriver.chrome.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.chrome.webdriver.WebDriver)

Controls the new OperaDriver and allows you to drive the Opera browser based on Chromium.

**\_\_init\_\_**(*executable\_path=None*, *port=0*, *options=None*, *service\_args=None*, *desired\_capabilities=None*, *service\_log\_path=None*, *opera\_options=None*, *keep\_alive=True*)

Creates a new instance of the operadriver.

Starts the service and then creates new instance of operadriver.

|  |  |
| --- | --- |
| **Args:** | * executable\_path - path to the executable. If the default is used   it assumes the executable is in the $PATH   * port - port you would like the service to run, if left as 0,   a free port will be found.   * options: this takes an instance of OperaOptions * service\_args - List of args to pass to the driver service * desired\_capabilities: Dictionary object with non-browser specific * service\_log\_path - Where to log information from the driver. * opera\_options - Deprecated argument for options capabilities only, such as “proxy” or “loggingPref”. |

**create\_options**()

*class*selenium.webdriver.opera.webdriver.**WebDriver**(*desired\_capabilities=None*, *executable\_path=None*, *port=0*, *service\_log\_path=None*, *service\_args=None*, *options=None*)

Bases: [**selenium.webdriver.opera.webdriver.OperaDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.opera.webdriver.OperaDriver)

*class***ServiceType**

**CHROMIUM***= 2*

**\_\_init\_\_**(*desired\_capabilities=None*, *executable\_path=None*, *port=0*, *service\_log\_path=None*, *service\_args=None*, *options=None*)

Creates a new instance of the operadriver.

Starts the service and then creates new instance of operadriver.

|  |  |
| --- | --- |
| **Args:** | * executable\_path - path to the executable. If the default is used   it assumes the executable is in the $PATH   * port - port you would like the service to run, if left as 0,   a free port will be found.   * options: this takes an instance of OperaOptions * service\_args - List of args to pass to the driver service * desired\_capabilities: Dictionary object with non-browser specific * service\_log\_path - Where to log information from the driver. * opera\_options - Deprecated argument for options capabilities only, such as “proxy” or “loggingPref”. |

7.30. PhantomJS WebDriver

*class*selenium.webdriver.phantomjs.webdriver.**WebDriver**(*executable\_path='phantomjs'*, *port=0*, *desired\_capabilities={'browserName': 'phantomjs'*, *'javascriptEnabled': True*, *'platform': 'ANY'*, *'version': ''}*, *service\_args=None*, *service\_log\_path=None*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

Wrapper to communicate with PhantomJS through Ghostdriver.

You will need to follow all the directions here: <https://github.com/detro/ghostdriver>

**\_\_init\_\_**(*executable\_path='phantomjs'*, *port=0*, *desired\_capabilities={'browserName': 'phantomjs'*, *'javascriptEnabled': True*, *'platform': 'ANY'*, *'version': ''}*, *service\_args=None*, *service\_log\_path=None*)

Creates a new instance of the PhantomJS / Ghostdriver.

Starts the service and then creates new instance of the driver.

|  |  |
| --- | --- |
| **Args:** | * executable\_path - path to the executable. If the default is used it assumes the executable is in the $PATH * port - port you would like the service to run, if left as 0, a free port will be found. * desired\_capabilities: Dictionary object with non-browser specific capabilities only, such as “proxy” or “loggingPref”. * service\_args : A List of command line arguments to pass to PhantomJS * service\_log\_path: Path for phantomjs service to log to. |

**quit**()

Closes the browser and shuts down the PhantomJS executable that is started when starting the PhantomJS

7.31. PhantomJS WebDriver Service

*class*selenium.webdriver.phantomjs.service.**Service**(*executable\_path*, *port=0*, *service\_args=None*, *log\_path=None*)

Bases: [**selenium.webdriver.common.service.Service**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.common.service.Service)

Object that manages the starting and stopping of PhantomJS / Ghostdriver

**\_\_init\_\_**(*executable\_path*, *port=0*, *service\_args=None*, *log\_path=None*)

Creates a new instance of the Service

|  |  |
| --- | --- |
| **Args:** | * executable\_path : Path to PhantomJS binary * port : Port the service is running on * service\_args : A List of other command line options to pass to PhantomJS * log\_path: Path for PhantomJS service to log to |

**command\_line\_args**()

**send\_remote\_shutdown\_command**()

**service\_url**

Gets the url of the GhostDriver Service

7.32. Safari WebDriver

*class*selenium.webdriver.safari.webdriver.**WebDriver**(*port=0*, *executable\_path='/usr/bin/safaridriver'*, *reuse\_service=False*, *desired\_capabilities={'browserName': 'safari'*, *'platform': 'MAC'*, *'version': ''}*, *quiet=False*, *keep\_alive=True*, *service\_args=None*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

Controls the SafariDriver and allows you to drive the browser.

**\_\_init\_\_**(*port=0*, *executable\_path='/usr/bin/safaridriver'*, *reuse\_service=False*, *desired\_capabilities={'browserName': 'safari'*, *'platform': 'MAC'*, *'version': ''}*, *quiet=False*, *keep\_alive=True*, *service\_args=None*)

Creates a new Safari driver instance and launches or finds a running safaridriver service.

|  |  |
| --- | --- |
| **Args:** | * port - The port on which the safaridriver service should listen for new connections. If zero, a free port will be found. * executable\_path - Path to a custom safaridriver executable to be used. If absent, /usr/bin/safaridriver is used. * reuse\_service - If True, do not spawn a safaridriver instance; instead, connect to an already-running service that was launched externally. * desired\_capabilities: Dictionary object with desired capabilities (Can be used to provide various Safari switches). * quiet - If True, the driver’s stdout and stderr is suppressed. * keep\_alive - Whether to configure SafariRemoteConnection to use   HTTP keep-alive. Defaults to False.   * service\_args : List of args to pass to the safaridriver service |

**debug**()

**get\_permission**(*permission*)

**quit**()

Closes the browser and shuts down the SafariDriver executable that is started when starting the SafariDriver

**set\_permission**(*permission*, *value*)

7.33. Safari WebDriver Service

*class*selenium.webdriver.safari.service.**Service**(*executable\_path*, *port=0*, *quiet=False*, *service\_args=None*)

Bases: [**selenium.webdriver.common.service.Service**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.common.service.Service)

Object that manages the starting and stopping of the SafariDriver

**\_\_init\_\_**(*executable\_path*, *port=0*, *quiet=False*, *service\_args=None*)

Creates a new instance of the Service

|  |  |
| --- | --- |
| **Args:** | * executable\_path : Path to the SafariDriver * port : Port the service is running on * quiet : Suppress driver stdout and stderr * service\_args : List of args to pass to the safaridriver service |

**command\_line\_args**()

**service\_url**

Gets the url of the SafariDriver Service

7.34. Select Support

*class*selenium.webdriver.support.select.**Select**(*webelement*)

Bases: **object**

**\_\_init\_\_**(*webelement*)

Constructor. A check is made that the given element is, indeed, a SELECT tag. If it is not, then an UnexpectedTagNameException is thrown.

|  |  |
| --- | --- |
| **Args:** | * webelement - element SELECT element to wrap |

Example:

from selenium.webdriver.support.ui import Select

Select(driver.find\_element\_by\_tag\_name(“select”)).select\_by\_index(2)

**deselect\_all**()

Clear all selected entries. This is only valid when the SELECT supports multiple selections. throws NotImplementedError If the SELECT does not support multiple selections

**deselect\_by\_index**(*index*)

Deselect the option at the given index. This is done by examing the “index” attribute of an element, and not merely by counting.

|  |  |
| --- | --- |
| **Args:** | * index - The option at this index will be deselected   throws NoSuchElementException If there is no option with specisied index in SELECT |

**deselect\_by\_value**(*value*)

Deselect all options that have a value matching the argument. That is, when given “foo” this would deselect an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * value - The value to match against   throws NoSuchElementException If there is no option with specisied value in SELECT |

**deselect\_by\_visible\_text**(*text*)

Deselect all options that display text matching the argument. That is, when given “Bar” this would deselect an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * text - The visible text to match against |

**select\_by\_index**(*index*)

Select the option at the given index. This is done by examing the “index” attribute of an element, and not merely by counting.

|  |  |
| --- | --- |
| **Args:** | * index - The option at this index will be selected |

throws NoSuchElementException If there is no option with specisied index in SELECT

**select\_by\_value**(*value*)

Select all options that have a value matching the argument. That is, when given “foo” this would select an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * value - The value to match against |

throws NoSuchElementException If there is no option with specisied value in SELECT

**select\_by\_visible\_text**(*text*)

Select all options that display text matching the argument. That is, when given “Bar” this would select an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * text - The visible text to match against   throws NoSuchElementException If there is no option with specisied text in SELECT |

**all\_selected\_options**

Returns a list of all selected options belonging to this select tag

**first\_selected\_option**

The first selected option in this select tag (or the currently selected option in a normal select)

**options**

Returns a list of all options belonging to this select tag

7.35. Wait Support

*class*selenium.webdriver.support.wait.**WebDriverWait**(*driver*, *timeout*, *poll\_frequency=0.5*, *ignored\_exceptions=None*)

Bases: **object**

**\_\_init\_\_**(*driver*, *timeout*, *poll\_frequency=0.5*, *ignored\_exceptions=None*)

Constructor, takes a WebDriver instance and timeout in seconds.

|  |  |
| --- | --- |
| **Args:** | * driver - Instance of WebDriver (Ie, Firefox, Chrome or Remote) * timeout - Number of seconds before timing out * poll\_frequency - sleep interval between calls By default, it is 0.5 second. * ignored\_exceptions - iterable structure of exception classes ignored during calls. By default, it contains NoSuchElementException only. |

Example:

from selenium.webdriver.support.ui import WebDriverWait

element = WebDriverWait(driver, 10).until(lambda x: x.find\_element\_by\_id(“someId”))

is\_disappeared = WebDriverWait(driver, 30, 1, (ElementNotVisibleException)).

until\_not(lambda x: x.find\_element\_by\_id(“someId”).is\_displayed())

**until**(*method*, *message=''*)

Calls the method provided with the driver as an argument until the return value is not False.

**until\_not**(*method*, *message=''*)

Calls the method provided with the driver as an argument until the return value is False.

7.36. Color Support

*class*selenium.webdriver.support.color.**Color**(*red*, *green*, *blue*, *alpha=1*)

Bases: **object**

Color conversion support class

Example:

**from** **selenium.webdriver.support.color** **import** Color

**print**(Color.from\_string('#00ff33').rgba)

**print**(Color.from\_string('rgb(1, 255, 3)').hex)

**print**(Color.from\_string('blue').rgba)

**\_\_init\_\_**(*red*, *green*, *blue*, *alpha=1*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*static***from\_string**(*str\_*)

**hex**

**rgb**

**rgba**

7.37. Event Firing WebDriver Support

*class*selenium.webdriver.support.event\_firing\_webdriver.**EventFiringWebDriver**(*driver*, *event\_listener*)

Bases: **object**

A wrapper around an arbitrary WebDriver instance which supports firing events

**\_\_init\_\_**(*driver*, *event\_listener*)

Creates a new instance of the EventFiringWebDriver

|  |  |
| --- | --- |
| **Args:** | * driver : A WebDriver instance * event\_listener : Instance of a class that subclasses AbstractEventListener and implements it fully or partially |

Example:

**from** **selenium.webdriver** **import** Firefox

**from** **selenium.webdriver.support.events** **import** EventFiringWebDriver, AbstractEventListener

**class** **MyListener**(AbstractEventListener):

**def** before\_navigate\_to(self, url, driver):

**print**("Before navigate to *%s*" % url)

**def** after\_navigate\_to(self, url, driver):

**print**("After navigate to *%s*" % url)

driver = Firefox()

ef\_driver = EventFiringWebDriver(driver, MyListener())

ef\_driver.get("http://www.google.co.in/")

**back**()

**close**()

**execute\_async\_script**(*script*, *\*args*)

**execute\_script**(*script*, *\*args*)

**find\_element**(*by='id'*, *value=None*)

**find\_element\_by\_class\_name**(*name*)

**find\_element\_by\_css\_selector**(*css\_selector*)

**find\_element\_by\_id**(*id\_*)

**find\_element\_by\_link\_text**(*link\_text*)

**find\_element\_by\_name**(*name*)

**find\_element\_by\_partial\_link\_text**(*link\_text*)

**find\_element\_by\_tag\_name**(*name*)

**find\_element\_by\_xpath**(*xpath*)

**find\_elements**(*by='id'*, *value=None*)

**find\_elements\_by\_class\_name**(*name*)

**find\_elements\_by\_css\_selector**(*css\_selector*)

**find\_elements\_by\_id**(*id\_*)

**find\_elements\_by\_link\_text**(*text*)

**find\_elements\_by\_name**(*name*)

**find\_elements\_by\_partial\_link\_text**(*link\_text*)

**find\_elements\_by\_tag\_name**(*name*)

**find\_elements\_by\_xpath**(*xpath*)

**forward**()

**get**(*url*)

**quit**()

**wrapped\_driver**

Returns the WebDriver instance wrapped by this EventsFiringWebDriver

*class*selenium.webdriver.support.event\_firing\_webdriver.**EventFiringWebElement**(*webelement*, *ef\_driver*)

Bases: **object**

” A wrapper around WebElement instance which supports firing events

**\_\_init\_\_**(*webelement*, *ef\_driver*)

Creates a new instance of the EventFiringWebElement

**clear**()

**click**()

**find\_element**(*by='id'*, *value=None*)

**find\_element\_by\_class\_name**(*name*)

**find\_element\_by\_css\_selector**(*css\_selector*)

**find\_element\_by\_id**(*id\_*)

**find\_element\_by\_link\_text**(*link\_text*)

**find\_element\_by\_name**(*name*)

**find\_element\_by\_partial\_link\_text**(*link\_text*)

**find\_element\_by\_tag\_name**(*name*)

**find\_element\_by\_xpath**(*xpath*)

**find\_elements**(*by='id'*, *value=None*)

**find\_elements\_by\_class\_name**(*name*)

**find\_elements\_by\_css\_selector**(*css\_selector*)

**find\_elements\_by\_id**(*id\_*)

**find\_elements\_by\_link\_text**(*link\_text*)

**find\_elements\_by\_name**(*name*)

**find\_elements\_by\_partial\_link\_text**(*link\_text*)

**find\_elements\_by\_tag\_name**(*name*)

**find\_elements\_by\_xpath**(*xpath*)

**send\_keys**(*\*value*)

**wrapped\_element**

Returns the WebElement wrapped by this EventFiringWebElement instance

7.38. Abstract Event Listener Support

*class*selenium.webdriver.support.abstract\_event\_listener.**AbstractEventListener**

Bases: **object**

Event listener must subclass and implement this fully or partially

**after\_change\_value\_of**(*element*, *driver*)

**after\_click**(*element*, *driver*)

**after\_close**(*driver*)

**after\_execute\_script**(*script*, *driver*)

**after\_find**(*by*, *value*, *driver*)

**after\_navigate\_back**(*driver*)

**after\_navigate\_forward**(*driver*)

**after\_navigate\_to**(*url*, *driver*)

**after\_quit**(*driver*)

**before\_change\_value\_of**(*element*, *driver*)

**before\_click**(*element*, *driver*)

**before\_close**(*driver*)

**before\_execute\_script**(*script*, *driver*)

**before\_find**(*by*, *value*, *driver*)

**before\_navigate\_back**(*driver*)

**before\_navigate\_forward**(*driver*)

**before\_navigate\_to**(*url*, *driver*)

**before\_quit**(*driver*)

**on\_exception**(*exception*, *driver*)

7.39. Expected conditions Support

*class*selenium.webdriver.support.expected\_conditions.**alert\_is\_present**

Bases: **object**

Expect an alert to be present.

**\_\_init\_\_**()

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**element\_located\_selection\_state\_to\_be**(*locator*, *is\_selected*)

Bases: **object**

An expectation to locate an element and check if the selection state specified is in that state. locator is a tuple of (by, path) is\_selected is a boolean

**\_\_init\_\_**(*locator*, *is\_selected*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**element\_located\_to\_be\_selected**(*locator*)

Bases: **object**

An expectation for the element to be located is selected. locator is a tuple of (by, path)

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**element\_selection\_state\_to\_be**(*element*, *is\_selected*)

Bases: **object**

An expectation for checking if the given element is selected. element is WebElement object is\_selected is a Boolean.”

**\_\_init\_\_**(*element*, *is\_selected*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**element\_to\_be\_clickable**(*locator*)

Bases: **object**

An Expectation for checking an element is visible and enabled such that you can click it.

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**element\_to\_be\_selected**(*element*)

Bases: **object**

An expectation for checking the selection is selected. element is WebElement object

**\_\_init\_\_**(*element*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**frame\_to\_be\_available\_and\_switch\_to\_it**(*locator*)

Bases: **object**

An expectation for checking whether the given frame is available to switch to. If the frame is available it switches the given driver to the specified frame.

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**invisibility\_of\_element**(*locator*)

Bases: [**selenium.webdriver.support.expected\_conditions.invisibility\_of\_element\_located**](https://selenium-python.readthedocs.io/api.html#selenium.webdriver.support.expected_conditions.invisibility_of_element_located)

An Expectation for checking that an element is either invisible or not present on the DOM.

element is either a locator (text) or an WebElement

*class*selenium.webdriver.support.expected\_conditions.**invisibility\_of\_element\_located**(*locator*)

Bases: **object**

An Expectation for checking that an element is either invisible or not present on the DOM.

locator used to find the element

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**new\_window\_is\_opened**(*current\_handles*)

Bases: **object**

An expectation that a new window will be opened and have the number of windows handles increase

**\_\_init\_\_**(*current\_handles*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**number\_of\_windows\_to\_be**(*num\_windows*)

Bases: **object**

An expectation for the number of windows to be a certain value.

**\_\_init\_\_**(*num\_windows*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**presence\_of\_all\_elements\_located**(*locator*)

Bases: **object**

An expectation for checking that there is at least one element present on a web page. locator is used to find the element returns the list of WebElements once they are located

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**presence\_of\_element\_located**(*locator*)

Bases: **object**

An expectation for checking that an element is present on the DOM of a page. This does not necessarily mean that the element is visible. locator - used to find the element returns the WebElement once it is located

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**staleness\_of**(*element*)

Bases: **object**

Wait until an element is no longer attached to the DOM. element is the element to wait for. returns False if the element is still attached to the DOM, true otherwise.

**\_\_init\_\_**(*element*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**text\_to\_be\_present\_in\_element**(*locator*, *text\_*)

Bases: **object**

An expectation for checking if the given text is present in the specified element. locator, text

**\_\_init\_\_**(*locator*, *text\_*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**text\_to\_be\_present\_in\_element\_value**(*locator*, *text\_*)

Bases: **object**

An expectation for checking if the given text is present in the element’s locator, text

**\_\_init\_\_**(*locator*, *text\_*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**title\_contains**(*title*)

Bases: **object**

An expectation for checking that the title contains a case-sensitive substring. title is the fragment of title expected returns True when the title matches, False otherwise

**\_\_init\_\_**(*title*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**title\_is**(*title*)

Bases: **object**

An expectation for checking the title of a page. title is the expected title, which must be an exact match returns True if the title matches, false otherwise.

**\_\_init\_\_**(*title*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**url\_changes**(*url*)

Bases: **object**

An expectation for checking the current url. url is the expected url, which must not be an exact match returns True if the url is different, false otherwise.

**\_\_init\_\_**(*url*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**url\_contains**(*url*)

Bases: **object**

An expectation for checking that the current url contains a case-sensitive substring. url is the fragment of url expected, returns True when the url matches, False otherwise

**\_\_init\_\_**(*url*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**url\_matches**(*pattern*)

Bases: **object**

An expectation for checking the current url. pattern is the expected pattern, which must be an exact match returns True if the url matches, false otherwise.

**\_\_init\_\_**(*pattern*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**url\_to\_be**(*url*)

Bases: **object**

An expectation for checking the current url. url is the expected url, which must be an exact match returns True if the url matches, false otherwise.

**\_\_init\_\_**(*url*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of**(*element*)

Bases: **object**

An expectation for checking that an element, known to be present on the DOM of a page, is visible. Visibility means that the element is not only displayed but also has a height and width that is greater than 0. element is the WebElement returns the (same) WebElement once it is visible

**\_\_init\_\_**(*element*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of\_all\_elements\_located**(*locator*)

Bases: **object**

An expectation for checking that all elements are present on the DOM of a page and visible. Visibility means that the elements are not only displayed but also has a height and width that is greater than 0. locator - used to find the elements returns the list of WebElements once they are located and visible

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of\_any\_elements\_located**(*locator*)

Bases: **object**

An expectation for checking that there is at least one element visible on a web page. locator is used to find the element returns the list of WebElements once they are located

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of\_element\_located**(*locator*)

Bases: **object**

An expectation for checking that an element is present on the DOM of a page and visible. Visibility means that the element is not only displayed but also has a height and width that is greater than 0. locator - used to find the element returns the WebElement once it is located and visible

**\_\_init\_\_**(*locator*)

x.\_\_init\_\_(…) initializes x; see help(type(x)) for signature