Locate Element

If you want to perform any automated action on a web page, you'll need the locators from it. These are unique identifiers associated with the web elements such as text, buttons, tables, div, etc.

It is not possible to interact with the web page if the test script is not able to find the web elements. Selenium Webdriver provides the following techniques for locating the web elements.

Click here to Go Back to main Selenium Python tutorial.

1. Locate Element by Name
2. Locate Element by ID
3. Locate Element by Link Text
4. Locate Element by Partial Link Text
5. Locate Element by XPath
6. Locate Element by CSS Selector
7. Locate Element by Tagname
8. Locate Element by Classname

1. Locate Elements By Name

It is a standard practice to define unique ids for web elements in an HTML code.

However, there may be cases when these unique identifiers are not present. Instead, the names are there; then we can also use them to select a web element.

Here is the code snippet that demonstrates the use of method. Below code opens Google in the browser and performs a text search.

In [72]:

```
%%writefile selenium_element_by_name.py

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

driver = webdriver.Firefox()
driver.get("http://google.com")
driver.maximize_window()
time.sleep(5)
inputElement = driver.find_element_by_name("q")
inputElement.send_keys("Surendra Panpaliya")
inputElement.submit()
time.sleep(20)

driver.close()
```

```
Writing selenium_element_by_name.py
```

In [ ]:

```

```

```
If the HTML code has more than one web element with “@name” attribute, then this method will select the first web element from the list. If no match occurs, a NoSuchElementException gets raised.
```

2. Locate Elements By ID

We use this method when the Id attribute for the element is available.

It is in fact, the most reliable and the fastest way to locate a particular web element on an HTML page.

An Id will always be unique for any object on a web page.

So, we should prefer using Id attribute for locating the elements over other available options.

Here is the code snippet that demonstrates the use of the method.

Below code opens Google in the browser and performs a text search.

In [83]:

```
%%writefile selenium_google.py

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

driver = webdriver.Chrome("/Users/surendra/chromedriver")
driver.get("http://google.com")
driver.maximize_window()
time.sleep(5)
inputElement = driver.find_element_by_id("gsr")
inputElement.send_keys("GKTCS Innovations")
time.sleep(5)
driver.close()
```

```
Overwriting selenium_google.py
```

In [84]:

```
run selenium_google.py
```

In [87]:

```
%%writefile selenium_element_by_link.py

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

driver = webdriver.Firefox()
driver.get("http://google.com")
driver.maximize_window()
time.sleep(5)
inputElement = driver.find_element_by_name("q")
inputElement.send_keys("GKTCS")
inputElement.submit()
time.sleep(5)
elem = driver.find_element_by_link_text("Contact us")
elem.click()
time.sleep(10)

driver.close()
```

```
Writing selenium_element_by_link.py
```

In [86]:

```
%%writefile selenium_partial_link.py

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

driver = webdriver.Firefox()
driver.get("http://google.com")
driver.maximize_window()
time.sleep(5)
inputElement = driver.find_element_by_name("q")
inputElement.send_keys("GKTCS")
inputElement.submit()
time.sleep(5)
elem = driver.find_element_by_partial_link_text("Contact")
elem.click()
time.sleep(20)

driver.close()
```

```
Writing selenium_partial_link.py
```

In [ ]:

```
1
```

In [27]:

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

driver = webdriver.Firefox()
driver.get("https://www.gktcs.com")
driver.maximize_window()
time.sleep(2)
form_element = driver.find_element_by_xpath("//form[1]")
time.sleep(2)
driver.close()
```

In [ ]:

```
1
```

Drag and Drop ActionChains

Dragging an element, from one position to another position.

Python API for Selenium provides a separate class called ActionChains for this purpose.

We need to create an object of this class and call the various methods that it provides us with, to interact with the elements.

In [88]:

```
%%writefile selenium_Action_Chains.py

import unittest
import time
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionCha


class DragTest(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()

    def test_dragelement(self):
        driver = self.driver
        driver.maximize_window()
        driver.get('http://jqueryui.com/draggable/')
        driver.switch_to.frame(0)
        source1 = driver.find_element_by_id('draggable')
        action = ActionChains(driver)
        #move element by x,y coordinates on the screen
        action.drag_and_drop_by_offset(source1, 100, 100).per
        time.sleep(5)

    def tearDown(self):
        self.driver.close()

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

```
Overwriting selenium_Action_Chains.py
```

In [7]:

```
run selenium_Action_Chains.py
```

```
test_dragelement (__main__.DragTest) ... ok

----------------------------------------------------------
----------------
Ran 1 test in 10.605s

OK
```

```
The ActionChains class provides us with a method, drag_and_drop_by_offset(source,x-offset,y-offset) to move/drag the element to desired x,y co-ordinates on the screen.

In that case, we would use a different method, provided by the same class : drag_and_drop(source,target)

Here’s how we would do it

source1 = driver.find_element_by_id('draggable')
target1 = driver.find_element_by_id('droppable')
actions2 = ActionChains(driver)
actions2.drag_and_drop(source1, target1).perform()
self.assertEqual("Dropped!", target1.text)

We find the source and destination elements (source is the one which is to be dropped in the target element).
```

In [41]:

```
%%writefile Selenium_Driver_back_forward.py
import unittest
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions

class toolsqa(unittest.TestCase):

    def setUp(self):
        self.driver=webdriver.Firefox()

    def test_filldetails(self):
        driver=self.driver
        driver.maximize_window()
        driver.get('https://www.toolsqa.com')
        driver.implicitly_wait(10)
        tt1=driver.find_element_by_link_text('ABOUT')
        tt1.click()

    def tearDown(self):
        self.driver.quit()

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

```
Overwriting Selenium_Driver_back_forward.py
```

In [42]:

```
run Selenium_Driver_back_forward.py
```

```
test_filldetails (__main__.toolsqa) ... /Users/suren
dra/anaconda3/lib/python3.6/http/client.py:1210: Res
ourceWarning: unclosed <socket.socket fd=72, family=
AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM,
proto=6, laddr=('127.0.0.1', 49205), raddr=('127.0.0
.1', 49195)>
  for i, one_value in enumerate(values):
ok

----------------------------------------------------
----------------
Ran 1 test in 11.214s

OK
```

In [46]:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as

driver = webdriver.Firefox()
driver.get("https://www.gktcs.com")
try:
    element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "button"))
    )
    driver.back()

finally:
    driver.quit()
```

In [345]:

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as
from selenium.common.exceptions import TimeoutException

browser = webdriver.Firefox()
browser.get("https://www.gktcs.com")
browser.find_element_by_link_text("contac").click()

```

```
try:
    WebDriverWait(browser, 3).until(EC.alert_is_present(),
                                    'Timed out waiting for PA
                                    'confirmation popup to app

    alert = browser.switch_to.alert
    alert.accept()
    print("alert accepted")
except TimeoutException:
    print("no alert")
```

```
---------------------------------------------------------
----------------------
NoSuchElementException                  Traceback
(most recent call last)
<ipython-input-345-2c2bfe1dc427> in <module>
      6 browser = webdriver.Firefox()
      7 browser.get("https://www.gktcs.com")
----> 8 browser.find_element_by_link_text("contac").
click()
      9
     10 try:

~/anaconda3/lib/python3.6/site-packages/selenium/web
driver/remote/webdriver.py in find_element_by_link_t
ext(self, link_text)
    426             element = driver.find_element_by
_link_text('Sign In')
    427         """
--> 428         return self.find_element(by=By.LINK_
TEXT, value=link_text)
    429
    430     def find_elements_by_link_text(self,
text):

~/anaconda3/lib/python3.6/site-packages/selenium/web
driver/remote/webdriver.py in find_element(self, by,
value)
    976         return self.execute(Command.FIND_ELE
MENT, {
    977             'using': by,
--> 978             'value': value})['value']
    979
    980     def find_elements(self, by=By.ID, value=
None):

~/anaconda3/lib/python3.6/site-packages/selenium/web
driver/remote/webdriver.py in execute(self, driver_c
ommand, params)
```

```
    319             response = self.command_executor.exe
cute(driver_command, params)
    320             if response:
--> 321                 self.error_handler.check_respons
e(response)
    322                 response['value'] = self._unwrap
_value(
    323                     response.get('value', None))

~/anaconda3/lib/python3.6/site-packages/selenium/web
driver/remote/errorhandler.py in check_response(self
, response)
    240                     alert_text = value['alert'].
get('text')
    241                 raise exception_class(message,
screen, stacktrace, alert_text)
--> 242             raise exception_class(message,
screen, stacktrace)
    243
    244     def _value_or_default(self, obj, key,
default):


NoSuchElementException: Message: Unable to locate el
ement: contac
```

In [69]:

```
%%writefile Selenium_Login_Form.py

from selenium import webdriver
# For using sleep function because selenium
# works only when the all the elements of the
# page is loaded.

import time
# webdriver path set
browser = webdriver.Chrome()

# To maximize the browser window
browser.maximize_window()

# zomato link set
browser.get("https://www.gktcs.com")

time.sleep(3)
```

```
# Enter your user name and password here.
username = "surendra.panpaliya@gmail.com"
password = "Secret"

# signin element clicked
browser.find_element_by_xpath('//*[@id="basicExampleNav"]/div
time.sleep(2)

# Login clicked
#browser.find_element_by_xpath("//*[@id='id_email']").click()

# username send
a = browser.find_element_by_xpath("//*[@id='id_email']")
a.send_keys(username)

# password send
b = browser.find_element_by_xpath('//*[@id="id_password"]')
b.send_keys(password)

# submit button clicked
browser.find_element_by_xpath('//*[@id="elegantModalForm"]/di

print('Login Successful')
browser.back()
browser.forward()
browser.close()
```

```
Overwriting Selenium_Login_Form.py
```

In [68]:

```
run Selenium_Login_Form.py
```

```
Login Successful
```

In [89]:

```
class ShortInput(Exception):
    '''Short Input Exception for short data'''
    def __init__(self, length, atleast):
        '''Intialisation of Short Input Class'''
        self.length = length
        self.atleast = atleast
        print("You Enter data of %d bytes, Expected atleast %


Ob = ShortInput(4,6)
```

```
You Enter data of 4 bytes, Expected atleast 6 bytes
```

In [90]:

```
while True:
    try:
        data = input("Enter some data: ")
        if len(data) < 6:
            raise ShortInput(len(data), 6)
    except ShortInput as e:
        print("You enter short data %d"%e.length)
    except Exception as e1:
        print("Other Exception",e1)
    else:
        print("Your data is %s"%data)
        break
    finally:
        print("Always Executed")
```

```
Enter some data: sure
You Enter data of 4 bytes, Expected atleast 6 bytes
You enter short data 4
Always Executed
Enter some data: surendra
Your data is surendra
Always Executed
```

In [ ]:

```

```

In [94]:

```
import selenium.webdriver
dir(selenium.webdriver)
```

Out[94]:

```
['ActionChains',
 'Android',
 'BlackBerry',
 'Chrome',
 'ChromeOptions',
 'DesiredCapabilities',
 'Edge',
 'Firefox',
 'FirefoxOptions',
 'FirefoxProfile',
 'Ie',
 'IeOptions',
 'Opera',
 'PhantomJS',
 'Proxy',
 'Remote',
 'Safari',
 'TouchActions',
 'WebKitGTK',
 'WebKitGTKOptions',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '__version__',
 'android',
 'blackberry',
 'chrome',
 'common',
 'edge',
 'firefox',
 'ie',
 'opera',
 'phantomjs',
 'remote',
 'safari',
 'support',
 'webkitgtk']
```

In [95]:

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

driver = webdriver.Firefox()
driver.get("http://google.com")
driver.maximize_window()
time.sleep(5)
inputElement = driver.find_element_by_name("q")
inputElement.send_keys("GKTCS")
inputElement.submit()
time.sleep(5)
elem = driver.find_element_by_link_text("Contact us")
elem.click()
time.sleep(10)

driver.close()
```

CheckBox and Radio Buttons

CheckBox and Radio buttons are an example of web elements which can be used to make some kind of selections in a web page.

The crux of handling a check boxes button is a method called is_selected( ), which checks if an option is selected or not.

We can use to verify if a radio button or even a check box is selected or not , and then we can use the result to process further events.

Here we're trying to see if the Wednesday radio button is selected.

If so, then we're going to click on the Monday radio button and then take a screenshot.

Similarly, as with Radio buttons, we can use the is_selected( ) method for check boxes as well.

In [326]:

```
%%writefile selenium_checkbox.py

import unittest
import time
from selenium import webdriver

class MyChkbox(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()

    def test_Chkbox(self):
        driver = self.driver
        driver.maximize_window()
        driver.get('https://openwritings.net/sites/default/fi

        elewed = driver.find_element_by_xpath("//*[@id='daysc
        elemon = driver.find_element_by_xpath("//*[@id='daysc

        if elewed.is_selected():
            print("Wednesday is selected")
            elemon.click()
            self.assertTrue(elemon.is_selected(), "Monday is
            driver.save_screenshot('/Users/surendra/Desktop/c
            time.sleep(5)
        else:
            print("Test failed")

    def tearDown(self):
        self.driver.close()

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

```
Overwriting selenium_checkbox.py
```

In [327]:

```
1 run selenium_checkbox.py
```

```
test_Chkbox (__main__.MyChkbox) ...

Wednesday is selected

ok

----------------------------------------------------------------------
Ran 1 test in 15.120s

OK
```

In [113]:

```
%%writefile selenium_radiobtn.py

import unittest
from selenium import webdriver


class MyRadiobtn(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()

    def test_Radiobtn(self):
        driver = self.driver
        driver.maximize_window()
        driver.get('http://openwritings.net/sites/default/fil

        eleapple= driver.find_element_by_name('apple')
        elebanana= driver.find_element_by_name('orange')

        if eleapple.is_selected():
            eleapple.click()
            elebanana.click()
        driver.save_screenshot('/Users/surendra/Desktop/radio

    def tearDown(self):
        self.driver.close()

if __name__ == '__main__':
    unittest.main(verbosity=2)


```

```
Overwriting selenium_radiobtn.py
```

In [114]:

```
run selenium_radiobtn.py
```

```
test_Radiobtn (__main__.MyRadiobtn) ... ok

----------------------------------------------------------
----------------
Ran 1 test in 7.964s

OK
```

In [122]:

```
%%writefile selenium_radio_chkbox.py

import unittest
import time
from selenium import webdriver

class MyChkbox(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()

    def test_Chkbox(self):
        driver = self.driver
        driver.maximize_window()
        driver.get('https://openwritings.net/sites/default/fi

        elewed = driver.find_element_by_xpath("//*[@id='daysc
        elemon = driver.find_element_by_xpath("//*[@id='daysc

        if elewed.is_selected():
            print("Wednesday is selected")
            elemon.click()
            self.assertTrue(elemon.is_selected(), "Monday is
            driver.save_screenshot('/Users/surendra/Desktop/c
            time.sleep(5)
        else:
            print("Checck box Test failed")

    def test_Radiobtn(self):
        driver = self.driver
        driver.maximize_window()
        driver.get('http://openwritings.net/sites/default/fil

        eleapple= driver.find_element_by_name('apple')
        elebanana= driver.find_element_by_name('orange')

        if eleapple.is_selected():
            print("Radio Button Apple is Selected")
            eleapple.click()
            print("Radio Button Orange is Selected")
            elebanana.click()
            driver.save_screenshot('/Users/surendra/Desktop/r
        else:
            print("Radio Btn Test failed")

    def tearDown(self):
        self.driver.close()
```

```
48
49 if __name__ == '__main__':
50     unittest.main(verbosity=2)
```

```
Overwriting selenium_radio_chkbox.py
```

In [123]:

```
1 run selenium_radio_chkbox.py
```

```
test_Chkbox (__main__.MyChkbox) ...

Wednesday is selected

ok
test_Radiobtn (__main__.MyChkbox) ...

Radio Button Apple is Selected
Radio Button Orange is Selected

ok

----------------------------------------------------------------------
Ran 2 tests in 22.397s

OK
```

In [328]:

```
from selenium import webdriver
import time

with webdriver.Firefox() as driver:
    driver.get("https://www.gktcs.com/surendra")
    # Get current URL You can read the current URL from the b
    print(driver.current_url)

    # Back Pressing the browser’s back button

    driver.back()
    print("Pressing the browser’s back button\n")

    time.sleep(5)

    # Forward Pressing the browser’s forward button

    driver.forward()

    time.sleep(5)

    print("Pressing the browser’s forward button\n")

    # Refresh the current page

    driver.refresh()

    time.sleep(5)

    print("Refresh the current page\n")

    # You can read the current page title from the browser

    print(driver.title)
```

```
https://www.gktcs.com/surendra/
(https://www.gktcs.com/surendra/)
Pressing the browser’s back button

Pressing the browser’s forward button

Refresh the current page

Gktcs LMS
```

Windows and tabs

Get window handle

WebDriver does not make the distinction between windows and tabs.

If your site opens a new tab or window, Selenium will let you work with it using a window handle.

Each window has a unique identifier which remains persistent in a single session.

You can get the window handle of the current window by using:

In [126]:

```
1 print(driver.current_window_handle)
```

```
4294967297
```

Switching Windows or tabs

Clicking a link which opens in a new window will focus the new window or tab on screen, but WebDriver will not know which window the Operating System considers active.

To work with the new window you will need to switch to it.

If you have only two tabs or windows open, and you know which window you start with, by the process of elimination you can loop over both windows or tabs that WebDriver can see, and switch to the one which is not the original.

```
WebDriver supports moving between named windows using the

## “switch_to_window” method


driver.switch_to_window("windowName")


All calls to driver will now be interpreted as being
directed to the particular window.

But how do you know the window’s name?

Take a look at the javascript or link that opened it:

<a href="https://www.gktcs.com" target="windowName">
Click here to open a website in new window</a>

### Iterate over every open window


for handle in driver.window_handles:
    driver.switch_to_window(handle)
```

In [329]:

```
%%writefile selenium_window_switch.py

from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as

# Start the driver
with webdriver.Firefox() as driver:
    # Open URL
    driver.get("https://selenium-python.readthedocs.io/")

    # Setup wait for later
    wait = WebDriverWait(driver, 10)

    # Store the ID of the original window
    original_window = driver.current_window_handle

    # Check we don't have other windows open already
    assert len(driver.window_handles) == 1

    # Execute the script to open in a new window

```

```
    driver.execute_script("window.open('https://www.gktcs.com
    # Wait for the new window
    try:
        wait.until(EC.number_of_windows_to_be(2))
    except TimeoutException as e:
        print("Number of Windows doesn't match",e)
    else:
        print("Succefully Open New Window or tab")

    # Loop through until we find a new window handle
    for window_handle in driver.window_handles:
        if window_handle != original_window:
            driver.switch_to.window(window_handle)
            break

    # Wait for the new tab to finish loading content
    try:
        wait.until(EC.title_contains("GKTCS INNOVATIONS"))
    except TimeoutException as e:
        print("No Title Found",e)
    else:
        print("Thanks you for visiting our web site")
```

```
Writing selenium_window_switch.py
```

In [330]:

```
run selenium_window_switch.py
```

```
Succefully Open New Window or tab
Thanks you for visiting our web site
```

Switching between frames

You can also swing from frame to frame (or into iframes):

driver.switch_to.frame("frameName")

It's possible to access subframes by separating the path with a dot, and you can specify the frame by its index too. That is:

driver.switch_to.frame("frameName.0.child")

would go to the frame named "child" of the first subframe of the frame called "frameName".

All frames are evaluated as if from *top*.

Once we are done with working on frames, we will have to come back to the parent frame which can be done using:

driver.switch_to.default_content()

In [337]:

```
%%writefile frames.html

<!DOCTYPE html>
<html>
<head>
<title>Switching Between IFrames Demo</title>
</head>
<body>
<h1>Welcome Viewers</h1>
<iframe name="frame1" id="FR1" src="http://www.seleniumhq.org
width="400"> </iframe>
<iframe name="frame2" id="FR2" height="500" width="400"
src="http://www.seleniumhq.org"> </iframe>
</body>
</html>
```

```
Overwriting frames.html
```

```

```

In [181]:

```
from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.common.by import By

import time

with webdriver.Firefox() as driver:
    driver.maximize_window()

    location = "file:///Users/surendra/frames.html"

    driver.get(location)

    ########Section-1

    # get the list of iframes present on the web page using t

    seq = driver.find_elements_by_tag_name('iframe')

    print("No of frames present in the web page are: ", len(s

    #switching between the iframes based on index

    for index in range(len(seq)):
        driver.switch_to.default_content()

        iframe = driver.find_elements_by_tag_name('iframe')[i
        driver.switch_to.frame(iframe)

        driver.implicitly_wait(30)

        #highlight the contents of the selected iframe

        driver.find_element_by_tag_name('a').send_keys(Keys.C

        time.sleep(2)

        # undo the selection within the iframe

        driver.find_element_by_tag_name('p').click()

        driver.implicitly_wait(30)

    driver.switch_to.default_content()
```

```
No of frames present in the web page are:  2
```

In [335]:

```
%%writefile selenium_switch_frame.py

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.common.by import By

import time

with webdriver.Firefox() as driver:
    driver.maximize_window()

    location = "file:///Users/surendra/frames.html"

    driver.get(location)

    ########Section-1

    # get the list of iframes present on the web page using t

    seq = driver.find_elements_by_tag_name('iframe')

    print("No of frames present in the web page are: ", len(s

    #switching between the iframes based on index

    for index in range(len(seq)):
        driver.switch_to.default_content()

        iframe = driver.find_elements_by_tag_name('iframe')[i
        driver.switch_to.frame(iframe)

        driver.implicitly_wait(30)

        #highlight the contents of the selected iframe

        driver.find_element_by_tag_name('a').send_keys(Keys.C

        time.sleep(2)

        # undo the selection within the iframe

        driver.find_element_by_tag_name('p').click()
```

```
        driver.implicitly_wait(30)

        driver.switch_to.default_content()
        ########Section-2
        #switch to a specific iframe (First frame) using Id a

        iframe = driver.find_element_by_id('FR1')
        print("iframe id is ",iframe)
        driver.switch_to.frame(iframe)

        print("Switch to selected iframe")

        driver.switch_to.default_content()

        iframe = driver.find_element_by_name('frame2')
        driver.switch_to.frame(iframe)
        print("Switch to frame2")
        time.sleep(2)
        driver.find_element_by_tag_name('a').send_keys(Keys.(
```

```
Writing selenium_switch_frame.py
```

In [339]:

```
run selenium_switch_frame.py
```

```
No of frames present in the web page are:  2
iframe id is  <selenium.webdriver.firefox.webelement
.FirefoxWebElement (session="7d3e41f7-b61b-1449-8c42
-b04dc396fc45", element="e69c00a6-f18b-d948-950f-731
9028a681f")>
Switch to selected iframe
Switch to frame2
iframe id is  <selenium.webdriver.firefox.webelement
.FirefoxWebElement (session="7d3e41f7-b61b-1449-8c42
-b04dc396fc45", element="e69c00a6-f18b-d948-950f-731
9028a681f")>
Switch to selected iframe
Switch to frame2
```

In [207]:

```
from selenium import webdriver

with webdriver.Firefox() as driver:
    driver.get("https://www.gktcs.com")
    # Get current URL You can read the current URL from the b
    print(driver.current_url)
```

```
    # You can read the current page title from the browser

    print(driver.title)

    print("\nDriver Desired Capabilities Dictionary\n")

    print(driver.desired_capabilities)

    print("\nBrowser Name: ")
    print(driver.desired_capabilities['browserName'])

    print("\nBrowser Version:")

    print(driver.desired_capabilities['browserVersion'])

    print("\nBrowser Desired Capabilities keys List :\n")

    print(driver.desired_capabilities.keys())

    print("\nBrowser Desired Capabilities Value List :\n")

    print(driver.desired_capabilities.values())
```

```
https://www.gktcs.com/ (https://www.gktcs.com/)
GKTCS INNOVATIONS|Learning Management System

Driver Desired Capabilities Dictionary

{'acceptInsecureCerts': True, 'browserName': 'firefo
x', 'browserVersion': '69.0.1', 'moz:accessibilityCh
ecks': False, 'moz:buildID': '20190917135527', 'moz:
geckodriverVersion': '0.24.0', 'moz:headless': False
, 'moz:processID': 66027, 'moz:profile': '/var/folde
rs/3y/n7r4wvds5l7dcmyjgdmlht200000gn/T/rust_mozprofi
le.XI639CkZ2Oc3', 'moz:shutdownTimeout': 60000, 'moz
:useNonSpecCompliantPointerOrigin': False, 'moz:webd
riverClick': True, 'pageLoadStrategy': 'normal', 'pl
atformName': 'mac', 'platformVersion': '18.7.0', 'ro
tatable': False, 'setWindowRect': True, 'strictFileI
nteractability': False, 'timeouts': {'implicit': 0,
'pageLoad': 300000, 'script': 30000}, 'unhandledProm
ptBehavior': 'dismiss and notify'}

Browser Name:
firefox

Browser Version:
69.0.1
```

```
Browser Desired Capabilities keys List :

dict_keys(['acceptInsecureCerts', 'browserName', 'br
owserVersion', 'moz:accessibilityChecks', 'moz:build
ID', 'moz:geckodriverVersion', 'moz:headless', 'moz:
processID', 'moz:profile', 'moz:shutdownTimeout', 'm
oz:useNonSpecCompliantPointerOrigin', 'moz:webdriver
Click', 'pageLoadStrategy', 'platformName', 'platfor
mVersion', 'rotatable', 'setWindowRect', 'strictFile
Interactability', 'timeouts', 'unhandledPromptBehavi
or'])

Browser Desired Capabilities Value List :

dict_values([True, 'firefox', '69.0.1', False, '2019
0917135527', '0.24.0', False, 66027, '/var/folders/3
y/n7r4wvds5l7dcmyjgdmlht200000gn/T/rust_mozprofile.X
I639CkZ2Oc3', 60000, False, True, 'normal', 'mac', '
18.7.0', False, True, False, {'implicit': 0, 'pageLo
ad': 300000, 'script': 30000}, 'dismiss and notify']
)
```

What is a web table?

A table is a HTML data, that is displayed with table tag.

```
In the <table> tag

<tr> defined the row data and

<td> defines the column data.

We usually use <table> to display the data in row and
column format.

Let us initially have a look regarding the basic HTML
table. <table>
```

```
<table>
<tbody>
<tr>
<th>Automation Tool</th>
<th>Licensing</th>
<th>Market response</th>
</tr>
<tr>
<td>Selenium</td>
<td>Free</td>
<td>In</td>
</tr>
<tr>
<td>QTP</td>
<td>Paid</td>
<td>Out</td>
</tr>
</tbody>
</table>
```

Automation Tool	Licensing	Market response
Selenium	Free	In
QTP	Paid	Out

This would display the output as shown below

These tables were usually classified as two types :

Static Web tables:

In these tables, the data is static, i.e here rows and columns are fixed

Dynamic Web tables:

In these tables, data is dynamic. i.e here rows and columns are variable.

So to write the XPath consider the above code

Step1 : Select the parent Element

In Web drivers,

X - path locators start with a double forward slash (//) followed by a parent element. Since we are now dealing with tables, the parent will be tables (//tables)

```
## Step2 : Add the child Elements:

Since for every table <tbody> is immediately followed by <table>
And <tbody> is usually considered as the child element of <table>
And all child elements will be placed to the right of the parent element <tbody> and these are separated by one forward flash.
```

```
Add predicates :
From the code shown above, this <tbody> contains two <tr> tags. And these elements were called as the  <tbody> child elements. Moreover, these <tr> elements is called as siblings.

Accessing Nested tables :
A table within a table is called the Nested tables.   In HTML , we can execute the nested tables with the following code
```

In [244]:

```
%%writefile table1.html
<table border : 2px solid black>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Surendra</td>
    <td>Panpaliya</td>
    <td>42</td>
  </tr>
  <tr>
    <td>Narendra</td>
    <td>Panpaliya</td>
    <td>40</td>
  </tr>
<tr>
    <td>Satish</td>
    <td>Panpaliya</td>
    <td>47</td>
  </tr>
 <tr>
    <td>Dev</td>
    <td>Panpaliya</td>
    <td>15</td>
  </tr>
<tr>
    <td>Rutwik</td>
    <td>Panpaliya</td>
    <td>10</td>
  </tr>
</table>
```

```
Overwriting table1.html
```

Firstname	Lastname	Age
Surendra	Panpaliya	42
Narendra	Panpaliya	40
Satish	Panpaliya	47
Dev	Panpaliya	15
Rutwik	Panpaliya	10

Type *Markdown* and LaTeX:

```
1
```

In [270]:

```
%%writefile selenium_table.py

from selenium import webdriver
import time


with webdriver.Firefox() as driver:
    driver.maximize_window()
    location = "file:///Users/surendra/table1.html"
    driver.get(location)

    row_path = '/html/body/table/tbody/tr'
    col_path = '/html/body/table/tbody/tr[1]/th'

    rows = len(driver.find_elements_by_xpath(row_path))

    cols = len(driver.find_elements_by_xpath(col_path))

    print("\n Total Number of Rows in Tables are :",rows)
    print("\n Total Number of Columns in Tables are :",cols)

    print ("\n ----Table Data--- \n")

    print("First Name\tLast Name\tAge\n")

    tdata_path = "/html/body/table/tbody/tr[2]/td[1]"

    for r in range(2,rows+1):
        for c in range(1, cols+1):
            value = driver.find_element_by_xpath("/html/body/
                                        /tr["+str(r)+"]/td["+str(
            print(value, end = '       ')
        print()
    time.sleep(3)


```

```
Overwriting selenium_table.py
```

In [271]:

```
run selenium_table.py
```

```
 Total Number of Rows in Tables are : 6

 Total Number of Columns in Tables are : 3

 ----Table Data---

First Name      Last Name       Age

Surendra       Panpaliya       42
Narendra       Panpaliya       40
Satish       Panpaliya       47
Dev       Panpaliya       15
Rutwik       Panpaliya       10
```

In [294]:

```
%%writefile selenium_upload.py

from selenium import webdriver
import time


with webdriver.Firefox() as driver:
    time.sleep(5)
    driver.get("https://the-internet.herokuapp.com/upload")
    time.sleep(5)
    file_id = driver.find_element_by_id("file-upload")
    # this filename is on your local workstation
    file_id.send_keys("/Users/surendra/surendra_pics1.jpg")
    time.sleep(5)
    file_submit = driver.find_element_by_id("file-submit")
    file_submit.click()
    driver.save_screenshot('/Users/surendra/Desktop/upload1.p
    time.sleep(2)
    print("File Uploaded Successfully\n")
```

```
Overwriting selenium_upload.py
```

In [340]:

```
run selenium_upload.py
```

```
File Uploaded Successfully
```

In [282]:

```
pwd
```

Out[282]:

```
'/Users/surendra'
```

In [291]:

```
%%writefile upload.html

<!DOCTYPE html>
<html>

<body>
<form>
    Pick Any File to upload:
    <input type="file" name="uploadfile" id="uploadfile">
</form>

</body>
</html>
```

```
Writing upload.html
```

Pick Any File to upload:
Choose File no file selected

In [293]:

```
%%writefile selenium_upload1.py

from selenium import webdriver
import time


with webdriver.Firefox() as driver:
    time.sleep(5)
    driver.get("file:///Users/surendra/upload.html")
    time.sleep(5)
    file_id = driver.find_element_by_id("file-upload")
    # this filename is on your local workstation
    file_id.send_keys("/Users/surendra/surendra_pics1.jpg")
    time.sleep(5)
    file_submit = driver.find_element_by_id("file-submit")
    file_submit.click()
    driver.save_screenshot('/Users/surendra/Desktop/upload1.p
    time.sleep(2)
    print("File Uploaded Successfully\n")
```

```
Writing selenium_upload1.py
```

In [307]:

```
%%writefile selenium_filedownload.py

from selenium import webdriver
import os
import time
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as
from selenium.webdriver.common.by import By


browser = webdriver.Chrome()
browser.get('https://www.7-zip.org/')
download_button = WebDriverWait(browser, 20).until(EC.element
                                                   ((By.CSS_SELECTOR, '
download_button.click()
time.sleep(10)
```

```
Overwriting selenium_filedownload.py
```

In [341]:

```
run selenium_filedownload.py
```

In [306]:

```
%%writefile selenium_filedownload1.py

from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as
from selenium.webdriver.common.by import By
import os
import time


# To prevent download dialog
profile = webdriver.FirefoxProfile()
profile.set_preference('browser.download.folderList', 2) # cu
profile.set_preference('browser.download.manager.showWhenStar
profile.set_preference('browser.download.dir', '/Users/surend
profile.set_preference("browser.download.manager.alertOnEXEOp
profile.set_preference("browser.download.manager.closeWhenDon
profile.set_preference("browser.download.manager.focusWhenSta

#application/octet-stream,application/vnd.ms-excel
profile.set_preference('browser.helperApps.neverAsk.saveToDis
                       'application/x-msdownload,application/

try:
    browser = webdriver.Firefox(profile)
    browser.get('https://www.7-zip.org/')
    download_button = WebDriverWait(browser, 20).until(EC.ele
                                        ((By.CSS_SELECTOR, 'td.It
    download_button.click()
    print("clicked...")
    time.sleep(10)
    print (os.listdir("/Users/surendra/Desktop/"))
except Exception as ex:
    print(ex)
    browser.close()
```

```
Writing selenium_filedownload1.py
```

In [304]:

```
run selenium_filedownload.py
```

```
clicked...
['upload1.png', '.DS_Store', 'PurchaseOrder', '.loca
lized', 'ImpDocs', 'Surendra_International_Corporate
_Trainer_Consultant.docx', 'Selenium_Python_Dubai',
'radiobtn1.png', 'checkbox2.png', '7z1900.exe', 'Sur
endra_International_Corporate_Trainer_Consultant.pdf
', 'Trainer_Profile', 'Ruby_On_Rails_Badoda', 'Pytho
n_Web_Framework_Pyramid.rtf', 'Surendra_Internationa
l_Corporate_Trainer_Consultant.pages']
```

```
# 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_sub
menu).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.

Mouse Actions in Selenium python

Mouse actions are nothing but the simulating the mouse events like operation performed using Mouse.

In Selenium Python, handling Mouse & keyboard events and mouse events (including actions such as Drag and Drop or clicking multiple elements With Control key) are done using the ActionChains API.

The following are a few methods present in the ActionChains class in python selenium.

Mouse Actions :

```
mouseMove
dragAndDrop
click
doubleClick
mouseUp
mouseDown
```

Keyboard Actions :

```
keyDown
keyUp
send_keys
```

perform() is a function which will make the mouse to perform an operation on the webpage, if you don't use perform() function then ActionChains don't have any effect on the webpage.

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.

We can combine a few mouse and keyboard action methods as per our wish.

mouseMove / hover in python selenium

Sometimes we may need to hover on an element to see it's properties like color, highlight, background, and font; to perform hover, we should use a method called mouseMove() from selenium bindings.

we will not stop with verifying element properties; we also have elements on the HTML page that will be revealed only when you hover on them, for example, sub-menu.

With the actions object, you should first move to the menu element, and then move to the submenu item and click it or perform whatever action you wish.

In below example we are hovering on a sub-menu which has id='sub-menu', sometimes this method is also known as mouseHover()

In [311]:

```
%%writefile selenium_mousehover.py

import unittest
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionCha

class Test(unittest.TestCase):
    def test_move_to_element(self):
        driver = webdriver.Chrome()
        driver.implicitly_wait(30)
        driver.get("https://chercher.tech/practice/practice-p

        # find the element
        menu = driver.find_element_by_id("sub-menu")
        #Create the object for Action Chains
        actions = ActionChains(driver)
        actions.move_to_element(menu)
        # perform the operation on the element
        actions.perform()

if __name__ == "__main__":
    unittest.main(verbosity=2)
```

```
Overwriting selenium_mousehover.py
```

In [342]:

```
1 run selenium_mousehover.py
```

```
test_move_to_element (__main__.Test) ... ok

----------------------------------------------------------------------
Ran 1 test in 9.976s

OK
```

keyUp & keyDown & send_keys in python

keyUp and KeyDown methods are used to press keyboard keys in python selenium with ActionChains API.

These methods will be useful if you want to press helper keys and normal like CTRL+A, SHIFT+A, CTRL+SHIFT+delete.

Most of all, the keyboard actions are used together.

In below example, set value "GKTCS" in the search field of Google, passing lower case letter in send_keys with the keypress of Shift

In [320]:

```
%%writefile selenium_keysaction.py

import unittest
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionCha
from selenium.webdriver.common import keys

class Test(unittest.TestCase):
    def test_move_to_element(self):
        driver = webdriver.Chrome()
        driver.implicitly_wait(5)
        driver.get("https://www.google.com")
        # perform drag and drop
        search_bar = driver.find_element_by_name("q")
        #Create the object for Action Chains
        actions = ActionChains(driver)
        actions.click(search_bar)
        actions.key_down(keys.Keys.SHIFT)
        actions.send_keys("gktcs")
        actions.key_up(keys.Keys.SHIFT)
        # perform the operation on the element
        actions.perform()

if __name__ == "__main__":
    unittest.main()
```

```
Overwriting selenium_keysaction.py
```

In [343]:

```
run selenium_keysaction.py
```

```
/Users/surendra/anaconda3/lib/python3.6/site-package
s/selenium/webdriver/remote/webdriver.py:270: Resour
ceWarning: unclosed <socket.socket fd=98, family=Add
ressFamily.AF_INET, type=SocketKind.SOCK_STREAM, pro
to=6, laddr=('127.0.0.1', 55075), raddr=('127.0.0.1'
, 55063)>
  for key, val in value.items():
.
----------------------------------------------------
----------------
Ran 1 test in 10.369s

OK
```

```
# 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.
```

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

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

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

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

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

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

Thrown when activating an IME engine has failed.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A new session could not be created.

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

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

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

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

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

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

Page Objects

A page object represents an area in the web application user interface that your test is interacting.

Benefits of using page object pattern:

Creating reusable code that can be shared across multiple test cases Reducing the amount of duplicated code If the user interface changes, the fix needs changes in only one place

Test case

Here is a test case which searches for a word in python.org website and ensure some results are found.

In [ ]:

```
import unittest
from selenium import webdriver
import page

class PythonOrgSearch(unittest.TestCase):
    """A sample test class to show how page object works"""

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.get("http://www.python.org")

    def test_search_in_python_org(self):
        """
        Tests python.org search feature. Searches for the wor
        Note that it does not look for any particular text in
        the results were not empty.
        """

        #Load the main page. In this case the home page of Py
        main_page = page.MainPage(self.driver)
        #Checks if the word "Python" is in title
        assert main_page.is_title_matches(), "python.org titl
        #Sets the text of search textbox to "pycon"
        main_page.search_text_element = "pycon"
        main_page.click_go_button()
        search_results_page = page.SearchResultsPage(self.dri
        #Verifies that the results page is not empty
        assert search_results_page.is_results_found(), "No re

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    unittest.main()
```

Page_object_classes

The page object pattern intends creating an object for each web page. By following this technique a layer of separation between the test code and technical implementation is created.

The page.py will look like this

In [222]:

In [322]:

```
%%writefile Page_object_classes.py

from element import BasePageElement
from locators import MainPageLocators

class SearchTextElement(BasePageElement):
    """This class gets the search text from the specified loc

    #The locator for search box where search string is entere
    locator = 'q'


class BasePage(object):
    """Base class to initialize the base page that will be ca

    def __init__(self, driver):
        self.driver = driver


class MainPage(BasePage):
    """Home page action methods come here. I.e. Python.org"""

    #Declares a variable that will contain the retrieved text
    search_text_element = SearchTextElement()

    def is_title_matches(self):
        """Verifies that the hardcoded text "Python" appears
        return "Python" in self.driver.title

    def click_go_button(self):
        """Triggers the search"""
        element = self.driver.find_element(*MainPageLocators.
        element.click()


class SearchResultsPage(BasePage):
    """Search results page action methods come here"""

    def is_results_found(self):
        # Probably should search for this text in the specifi
        # element, but as for now it works fine
        return "No results found." not in self.driver.page_so


```

```
Writing Page_object_classes.py
```

Page elements

The element.py will look like this

In [ ]:

```
from selenium.webdriver.support.ui import WebDriverWait


class BasePageElement(object):
    """Base page class that is initialized on every page obje

    def __set__(self, obj, value):
        """Sets the text to the value supplied"""
        driver = obj.driver
        WebDriverWait(driver, 100).until(
            lambda driver: driver.find_element_by_name(self.l
        driver.find_element_by_name(self.locator).clear()
        driver.find_element_by_name(self.locator).send_keys(v

    def __get__(self, obj, owner):
        """Gets the text of the specified object"""
        driver = obj.driver
        WebDriverWait(driver, 100).until(
            lambda driver: driver.find_element_by_name(self.l
        element = driver.find_element_by_name(self.locator)
        return element.get_attribute("value")

```

Locators

One of the practices is to separate the locator strings from the place where they are being used.

In this example, locators of the same page belong to same class.

The locators.py will look like this

In [ ]:

```
from selenium.webdriver.common.by import By

class MainPageLocators(object):
    """A class for main page locators. All main page locators
    GO_BUTTON = (By.ID, 'submit')

class SearchResultsPageLocators(object):
    """A class for search results locators. All search result
    pass
```