Python, Selenium

Installing Python and Selenium

Installing Python:

Windows : http://python.org/download/.

Note : IF you are using Linux, MacOS X, Unix operating Systems then python will be installed by default with OS

1.What is PIP installer Tool?

pip is a package management system used to install and manage software packages written in Python pip is a recursive acronym that can stand for either "Pip Installs Packages"

or "Pip Installs Python

2.Where do we get this PIP Tool? And how to configure it in our Local Machines

3.Installing Selenium

Use Below command on PIP to install Selenium Package

pip install selenium

This command will set up the Selenium WebDriver client library on your machine with all modules and classes that we will need to create automated scripts using Python

4.pip install -U selenium

The optional –U flag will upgrade the existing version of the installed package

"{} {}".format("value is", b) -- concatenate string and integer

print("{} {}".format("Value is:", b))

print(type(b)) -- checks the type of the variable

PYTHON DATATYPES

1. Numeric: int (20), float (10.10), complex (100+2j)

2. String : String in a sequence of characters. represented by single or double quotes

3. List: List is a versatile data type exclusive in python. same as array in c/c++. List in python can hold different types of data. formally list is an ordered sequence of some data written using square brackets ([]) and commas (,)

values = [1, 2, "charanjeet", 10.20, 550]  
  
print(values[0]) # 1  
print(values[2]) # charanjeet  
print(values[3]) # 10.20  
  
print(values[-1]) # prints last index value  
  
print(values[1:3]) # prints index from 1 to 2  
  
values.insert(3, "singh") # insert value at index 3  
print(values)  
  
values.append("end") # adds new variable at the end  
print(values)  
  
values[2] = "CHARANJEET" # update the value  
del values[0] # delete the index

4. Tuple: Tuple is another data type which is a sequence of data similar to list but it is **immutable** that means data in a tuple is write protected. Data in a tuple is written using parenthesis and commas.

values = (1, 2, "charanjeet", 10.20, 550)  
  
print(values[1])  
  
values[2] = "CHARANJEET"

OUTPUT:

2

Traceback (most recent call last):

File "C:\Users\CHARANJEET\PycharmProjects\PythonTesting\Demo3.py", line 6, in <module>

values[2] = "CHARANJEET"

TypeError: 'tuple' object does not support item assignment

5. Dictionary : Python dictionary is an unordered sequence of data of key-value pair form. It is similar to the hash table type. Dictionaries are written within curly braces in the form **key:value.** It is very useful to retrieve data in an optimized way among a large amount of data.

dictionary = {'a': 2, 3: 'bcd', 'e':'charanjeet singh'}  
  
print(dictionary['a'])  
print(dictionary[3])  
print(dictionary['e'])

CREATE DICTIONARY AT RUN TIME AND ADD DATA TO IT:

dictionary1 = {}  
  
dictionary1['firstname'] = 'charanjeet'  
dictionary1['lastname'] = 'singh'  
dictionary1['gender'] = 'male'

IF-ELSE:

greeting = 'Good Morning!'  
  
if greeting == 'Good Morning':  
 print('condition matches')  
 print('2nd line')  
else:  
 print('condition donot match')  
  
print('if else condition code is completed')

FOR-LOOP:

A for loop is used for iterating over a sequence

obj = [2, 3, 4, 5, 6]  
  
for i in obj:  
 print(i\*2)  
  
# sum of first five natural numbers  
# range(i,j) -> i to j-1  
add = 0  
for j in range(1,6):  
 add = add + j  
print("{} {}".format('Outcome is: ', add))

for k in range(1, 10, 2):  
 print(k)  
  
for m in range(10):  
 print(m)

WHILE-LOOP:

With while loop we can execute a set of statements as long as condition is true.

it= 4  
  
while it>=1:  
 print(it)  
 it = it-1  
   
print('while loop is completed')

it= 10  
  
while it>=1:  
 if it == 9:  
 it = it - 1  
 continue  
 if it == 2:  
 break  
 print(it)  
 it = it-1

FUNCTIONS:

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return as a result.

In Python a function is defined using the def keyword

# function declaration  
def greet\_me():  
 print('good morning!')  
  
# function calling  
greet\_me()

Send and call parameter into the function

# function declaration  
def greet\_me(name):  
 print('Good Morning! ' +name)  
  
# function calling  
greet\_me('Charanjeet Singh')

def addinteger(a,b):  
 print(a+b)  
  
addinteger(41,5)

or

return a+b

print(addinteger(41,5))

CLASSES:

Python is an object oriented programming language. Almost everything is python is an object, with its properties and methods. Classes are user defined blueprint for creating objects.

To create a class, use the keyword class:

*#self keyword is mandatory for calling variable names into method  
#instance and class variables have whole different purpose  
#constructor name should be \_\_init\_\_*class Calculator:  
 num = 100  
  
 *# default constructor* def \_\_init\_\_(self, a, b):  
 self.firstNumber = a  
 self.secondNumber = b  
 print('automatically called when object is created')  
  
 def getData(self): *# def keyword inside a class creates method and outside the class creates function* print('now executing as method in a class')  
  
 def addition(self):  
 return self.firstNumber + self.secondNumber + Calculator.num  
  
obj = Calculator(4, 5) *# syntax to create objects in python*obj.getData()  
print(obj.addition())  
  
obj1 = Calculator(5, 10)  
obj1.getData()  
print(obj1.addition())

INHERITANCE:

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from or called base class. Child class is the class that inherits from another class also called derived class.

from OopsDemo import Calculator  
  
class ChildImpl(Calculator):  
 num2 = 200  
  
 def \_\_init\_\_(self):  
 Calculator.\_\_init\_\_(self, 10, 15)  
  
 def getCompleteData(self):  
 return self.num2 + self.num + self.addition()  
  
obj = ChildImpl()  
print(obj.getCompleteData())

STRINGS:

Strings in python are surrounded by either single or double quotes. ‘hello’ is the same as “hello”. You can display a string literal with the print() function.

str1 = 'rahulshettyacademy.com'  
str2 = 'consulting firm'  
str3 = 'consult'  
  
print(str1[1]) #a  
print(str1[0:5]) #rahul  
print(str1 + "-" +str2) #concatenate 2 strings  
  
print(str3 in str2) #check string is present in another string or not  
  
var = str1.split(".") #splits the string  
print(var)  
print(var[0])  
str4= ' great '  
  
print(str4.strip()) #trims the white from string start and end  
print(str4.lstrip()) #trims left space  
print(str4.rstrip()) #trims right space

FILE HANDLING:

The key function for working with files in python is the open() function.

READ A FILE:

file = open('test.txt')  
#read all the contents of the file  
#read n number of character by passing parameter  
#print(file.read(16))  
#read one single line at a time  
#print(file.readline())  
#print(file.readline())  
  
# print line by line using readline method  
#line = file.readline()  
#while line != "":  
 # print(line)  
 # line = file.readline()  
#for line in file:  
 # print(line)  
for line in file.readlines(): # readlines stores all data in the list  
 print(line)  
file.close()

WRITE INTO A FILE:

#read the file and store all the lines in list  
#reverse the list  
#write the reversed list back to the file  
  
with open('test.txt', 'r') as reader:  
 content = reader.readlines() #[alto, baleno, ciaz, dzire, ertiga]  
 reversed(content) #[ertiga, dzire, ciaz, baleno, alto]  
 with open('test.txt', 'w') as writer:  
 for line in reversed(content):  
 writer.write(line)

EXCEPTION HANDLING:

When an error occurs, or exception as we call it, python will normally stop and generate an error message.

These exceptions can be handled using “try” statement.

items = 0  
#2 items will be added to cart  
  
if items != 2:  
 #raise Exception("products cart count not matching")  
 pass  
  
assert(items == 0)  
  
#try, except  
  
try:  
 with open('filelog.txt','r') as reader:  
 reader.read()  
  
except Exception as e:  
 #print('exception occured')  
 print(e)

The ‘finally’ block will be executed regardless if try block raises an exception or not.

SELENIUM – INVOKE THE BROWSER

from selenium import webdriver  
  
#browser exposes an executable file  
#through selenium test we need to invoke the executable fill which will then invoke the browser  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.maximize\_window()  
driver.get('https://www.google.com/')  
print(driver.title)  
print(driver.current\_url)  
driver.get('https://www.youtube.com')  
print(driver.title)  
print(driver.current\_url)  
driver.back()  
driver.refresh()  
driver.minimize\_window()  
  
driver.close()

WEBDRIVER LOCATIONS

* ID
* NAME
* XPATH
* CSS
* ClassName
* linkText

Customized CSS Syntax:

Tagname[attribute=’value’] – tagname optional

Eg: input[name=’name’]

Customized XPath Syntax:

//tagname[@attribute=’value’] – tagname optional

Eg: //input[@type=‘submit’]

GENERATING CSS FROM ID:

Tagname#ID – tagname optional

GENERATING CSS FROM CLASSNAME:

Tagname.classname – make sure there are no spaces in class names

from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.get('https://rahulshettyacademy.com/angularpractice/')  
#driver.find\_element\_by\_name('name').send\_keys('charanjeet')  
  
driver.find\_element\_by\_css\_selector("input[name='name']").send\_keys('charanjeet')  
#driver.find\_element(By.CSS\_SELECTOR, "input[name='name']").send\_keys('charanjeet')  
driver.find\_element\_by\_id('exampleCheck1').click()  
driver.find\_element\_by\_xpath("//input[@type='submit']").click()  
print(driver.find\_element(By.CLASS\_NAME, 'alert-success').text)  
# driver.find\_element(By.ID, 'exampleCheck1').click()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.get('https://login.salesforce.com/?locale=in')  
driver.find\_element(By.CSS\_SELECTOR, "#username").send\_keys('charanjeet')  
driver.find\_element(By.CSS\_SELECTOR, ".password").send\_keys('password')  
  
driver.find\_element(By.CSS\_SELECTOR, ".password").clear()  
driver.find\_element(By.LINK\_TEXT, "Forgot Your Password?").click()

driver.close()

GENERATING XPATH BASED ON TEXT:

//tagname[text()=’xxx’] - tagname is optional

print(driver.find\_element(By.XPATH, "//label[text()='Username']").text)

CREATING XPATH AND CSS BY TRAVERSING TAGS:

IDENTIFYING STATIC DROPDOWN USING SELECT CLASS OF SELENIUM:

#select class provides the methods to handle the options in dropdown with tagname 'select' only

dropdown = Select(driver.find\_element(By.ID, 'exampleFormControlSelect1'))  
dropdown.select\_by\_visible\_text('Female')  
dropdown.select\_by\_index(0)

ASSERTION:

message = driver.find\_element(By.CLASS\_NAME, 'alert-success').text  
assert 'success' in message

AUTO SUGGESTIVE DROPDOWNS:

import time  
  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
driver.get('https://rahulshettyacademy.com/dropdownsPractise/')  
  
driver.find\_element(By.ID, 'autosuggest').send\_keys('ind')  
time.sleep(2)  
countries = driver.find\_elements(By.CSS\_SELECTOR, "li[class='ui-menu-item'] a")  
print(len(countries))  
  
for country in countries:  
 if country.text == 'India':  
 country.click()  
 break  
print(driver.find\_element(By.ID, 'autosuggest').get\_attribute('value'))  
assert driver.find\_element(By.ID, 'autosuggest').get\_attribute('value') == 'India'

CLICK ON MULTIPLE CHECKBOXES:

import time  
  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
driver.get('https://rahulshettyacademy.com/AutomationPractice/')  
  
checkboxes = driver.find\_elements(By.XPATH, "//input[@type='checkbox']")  
print(len(checkboxes))  
  
for checkbox in checkboxes:  
 #checkbox.click()  
 assert checkbox.is\_selected()

for checkbox in checkboxes:  
 if checkbox.get\_attribute('value') == 'option2':  
 checkbox.click()  
 assert checkbox.is\_selected()

radiobuttons = driver.find\_elements(By.NAME, 'radioButton')  
radiobuttons[2].click()

HANDLING JAVA/JAVASCRIPT ALERT POPUPS:

from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
driver.get('https://rahulshettyacademy.com/AutomationPractice/')  
validateText = 'CHARANJEET'  
driver.find\_element(By.CSS\_SELECTOR, '#name').send\_keys(validateText)  
driver.find\_element(By.ID, 'alertbtn').click()  
alert = driver.switch\_to.alert  
alertText = alert.text  
assert validateText in alertText  
alert.accept()  
#alert.dismiss()

WAITS:

* Implicit wait: driver.implicitly\_wait(5), waits until 5 seconds if object is not displayed. Also known as global wait. Maximum time it will wait for object is 5 seconds and if the object take 1 sec to show, the code will execute after 1 second.
* Explicit wait: it is used to target specific object or element.

wait = WebDriverWait(driver, 10)  
wait.until(expected\_conditions.presence\_of\_element\_located((By.CLASS\_NAME, "promoCode")))  
  
driver.find\_element(By.CLASS\_NAME, "promoCode").send\_keys("rahulshettyacademy")  
driver.find\_element(By.CLASS\_NAME, "promoBtn").click()  
wait.until(expected\_conditions.presence\_of\_element\_located((By.CSS\_SELECTOR, ".promoInfo")))  
print(driver.find\_element(By.CSS\_SELECTOR, ".promoInfo").text)

* Pause the test for few seconds using “time” class.

import time  
  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
driver.implicitly\_wait(5)  
driver.get('https://rahulshettyacademy.com/seleniumPractise/')  
driver.find\_element(By.CSS\_SELECTOR, "input.search-keyword").send\_keys('ber')  
time.sleep(3)  
count = len(driver.find\_elements(By.XPATH, "//div[@class='products']/div"))  
assert count == 3  
  
buttons = driver.find\_elements(By.XPATH, "//div[@class='product-action']/button")  
for button in buttons:  
 button.click()  
  
driver.find\_element(By.CSS\_SELECTOR, "img[alt='Cart']").click()  
driver.find\_element(By.XPATH, "//button[text()='PROCEED TO CHECKOUT']").click()  
driver.find\_element(By.CLASS\_NAME, "promoCode").send\_keys("rahulshettyacademy")  
driver.find\_element(By.CLASS\_NAME, "promoBtn").click()  
print(driver.find\_element(By.CSS\_SELECTOR, ".promoInfo").text)

FUNCTIONAL EXAMPLE ON GREENKART APPLICATION:

* Validate whether products selected in page1 are showing in page2.
* Verify in price decrease after discount.
* Verify if sum of products in checkout page matches with total amount.

import time  
  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support import expected\_conditions  
from selenium.webdriver.support.wait import WebDriverWait  
  
list1 = []  
list2 = []  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.get('https://rahulshettyacademy.com/seleniumPractise/')  
driver.find\_element(By.CSS\_SELECTOR, "input.search-keyword").send\_keys('ber')  
time.sleep(3)  
count = len(driver.find\_elements(By.XPATH, "//div[@class='products']/div"))  
assert count == 3  
  
buttons = driver.find\_elements(By.XPATH, "//div[@class='product-action']/button")  
for button in buttons:  
 #print(button.find\_element(By.XPATH, "parent::div/parent::div/h4").text)  
 list1.append(button.find\_element(By.XPATH, "parent::div/parent::div/h4").text)  
 button.click()  
  
print(list1)  
  
driver.find\_element(By.CSS\_SELECTOR, "img[alt='Cart']").click()  
driver.find\_element(By.XPATH, "//button[text()='PROCEED TO CHECKOUT']").click()  
wait = WebDriverWait(driver, 10)  
wait.until(expected\_conditions.presence\_of\_element\_located((By.CLASS\_NAME, "promoCode")))  
  
veggies = driver.find\_elements(By.CSS\_SELECTOR, "p.product-name")  
for veg in veggies:  
 list2.append(veg.text)  
print(list2)  
assert list1 == list2  
  
originalAmount = driver.find\_element(By.CSS\_SELECTOR, ".discountAmt").text  
driver.find\_element(By.CLASS\_NAME, "promoCode").send\_keys("rahulshettyacademy")  
driver.find\_element(By.CLASS\_NAME, "promoBtn").click()  
  
wait.until(expected\_conditions.presence\_of\_element\_located((By.CSS\_SELECTOR, ".promoInfo")))  
  
discountAmount = driver.find\_element(By.CSS\_SELECTOR, ".discountAmt").text  
assert float(discountAmount) < int(originalAmount)  
print(driver.find\_element(By.CSS\_SELECTOR, ".promoInfo").text)  
  
amounts = driver.find\_elements(By.XPATH, "//tr/td[5]/p")  
sum = 0  
for amount in amounts:  
 sum = sum + int(amount.text) #48+160+180  
  
print(sum)  
totalAmount = int(driver.find\_element(By.CSS\_SELECTOR, ".totAmt").text)  
assert sum == totalAmount

HANDLE CHILD WINDOW/TAB:

from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.get('https://the-internet.herokuapp.com/windows')  
driver.find\_element(By.LINK\_TEXT, "Click Here").click()  
  
parentwindow = driver.window\_handles[0] #parent window id  
childwindow = driver.window\_handles[1] #child window id  
driver.switch\_to.window(childwindow) #switches to child window  
print(driver.find\_element(By.TAG\_NAME, "h3").text)  
driver.switch\_to.window(parentwindow) #switches to parent window  
print(driver.find\_element(By.TAG\_NAME, "h3").text)

FRAMES AND TECHNIQUES TO HANDLE:

from selenium import webdriver  
from selenium.webdriver.common.by import By  
#iframe, frame, frameset  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.get('https://the-internet.herokuapp.com/iframe')  
  
#frame id or frame name or index value  
driver.switch\_to.frame("mce\_0\_ifr") #switches to frame  
driver.find\_element(By.CSS\_SELECTOR, "body[id='tinymce']").clear()  
driver.find\_element(By.CSS\_SELECTOR, "body[id='tinymce']").send\_keys('my name is anthony gonzalwis')  
  
driver.switch\_to.default\_content() #switches from frame to html  
assert driver.find\_element(By.TAG\_NAME, "h3").text == "An iFrame containing the TinyMCE WYSIWYG Editor"

ACTIONCHAINS:

from selenium import webdriver  
from selenium.webdriver import ActionChains  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
driver.get("https://rahulshettyacademy.com/AutomationPractice/")  
action = ActionChains(driver)  
menu = driver.find\_element(By.ID, "mousehover")  
action.move\_to\_element(menu).perform()  
  
childMenu = driver.find\_element(By.LINK\_TEXT, "Top")  
action.move\_to\_element(childMenu).click().perform()

driver.get("https://chercher.tech/practice/practice-pop-ups-selenium-webdriver")  
action.double\_click(driver.find\_element(By.ID, "double-click")).perform()  
#action.context\_click() - used for right click  
alert = driver.switch\_to.alert  
print(alert.text)  
alert.accept()

IS DISPLAYED AND ASSERT NOT FOR FALSE CONDITIONS:

assert driver.find\_element(By.ID, "displayed-text").is\_displayed()  
driver.find\_element(By.ID, "hide-textbox").click()  
assert not driver.find\_element(By.ID, "displayed-text").is\_displayed()

JAVASCRIPT EXECUTOR:

# JS DOM can access any elements on web page just like how selenium does  
# selenium has a method to execute javascript code in it  
  
from selenium import webdriver  
from selenium.webdriver.common.by import By  
  
driver = webdriver.Chrome(executable\_path='c:\\chromedriver.exe')  
  
driver.get('https://rahulshettyacademy.com/angularpractice/')  
driver.find\_element(By.NAME, "name").send\_keys('charanjeet')  
print(driver.find\_element(By.NAME, "name").text)  
print(driver.find\_element(By.NAME, "name").get\_attribute("value"))  
#execute script will provide control from selenium to java script  
print(driver.execute\_script("return document.getElementsByName('name')[0].value"))  
  
shopButton = driver.find\_element(By.CSS\_SELECTOR, "a[href\*='shop']")  
driver.execute\_script("arguments[0].click();",shopButton)  
  
driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

CHROME OPTIONS:

chrome\_options = webdriver.ChromeOptions()  
chrome\_options.add\_argument("--start-maximized")  
chrome\_options.add\_argument("headless")  
chrome\_options.add\_argument("--ignore-certificate-errors")

E2E2:

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
from selenium.webdriver.support import expected\_conditions  
from selenium.webdriver.support.wait import WebDriverWait  
  
path = Service("c:\\chromedriver.exe")  
driver = webdriver.Chrome(service=path)  
driver.get('https://rahulshettyacademy.com/angularpractice/')  
driver.find\_element(By.CSS\_SELECTOR, "a[href\*='shop']").click()  
  
products = driver.find\_elements(By.XPATH, "//div[@class='card h-100']")  
for product in products:  
 productName = product.find\_element(By.XPATH, "div/h4").text  
 if productName == "Blackberry":  
 #add item into cart  
 product.find\_element(By.XPATH, "div/button").click()  
  
driver.find\_element(By.CSS\_SELECTOR, "a[class\*='btn-primary']").click()  
driver.find\_element(By.CSS\_SELECTOR, "button[class\*='btn-success']").click()  
driver.find\_element(By.ID, "country").send\_keys("ind")  
wait = WebDriverWait(driver, 7)  
wait.until(expected\_conditions.presence\_of\_element\_located((By.LINK\_TEXT, "India")))  
driver.find\_element(By.LINK\_TEXT, "India").click()  
driver.find\_element(By.XPATH, "//div[@class='checkbox checkbox-primary']").click()  
driver.find\_element(By.CSS\_SELECTOR, "[type='submit']").click()  
  
assert "Success! Thank you! " in driver.find\_element(By.CLASS\_NAME, "alert-success").text  
driver.get\_screenshot\_as\_file("screen.png")

SELENIUM PYTHON FRAMEWORK DESIGN PLAN:

1. PYTEST UNIT TESTING FRAMEWORK.
2. UNDERSTAND LOGGING AND HTML REPORTS.
3. IMPLEMENT SELENIUM PYTHON FRAMEWORK FROM SCRATCH (PAGE OBJECT DESIGN PATTERN).
4. DATA DRIVEN FRAMEWORK WITH EXCEL TO SELENIUM PYTHON TESTS
5. GIT VERSION CONTROL

PYTEST

Pip install pytest

# any pytest file should start with “test\_” or end with “\_test”  
# pytest method names should start with test  
# any code should be wrapped in method  
  
def test\_firstProgram():  
 msg = 'hello'  
 assert msg == 'hi', 'test failed coz strings do not match'

RUN PYTEST FROM TERMINAL:

1. py.test –v –s (it will run all test file in the directory)
2. py.test <filename> -v –s (it will run only this particular file)
3. py.test – k CreditCard –v –s (it will run all test cases with CreditCard name in it)
4. –k stands for method names execution, -s stands for logs in output and –v stands for more info metadata
5. –m stands for mark (it runs tests which are marked)

Eg: py.test –m smoke –v –s

@pytest.mark.smoke – must be defined before each test method

1. Pytest.mark.skip – it will skip the particular tests
2. Pytest.mark.xfail – it will skip the reporting of test

FIXTURES IN PYTEST

Fixture are used as setup and tear down methods for test cases.

@pytest.fixture()  
def setup():  
 print("i will be executing first")  
 yield  
 print("i will run at last")  
  
def test\_fixtureDemo(setup):  
 print("i will execute steps in fixturedemo method")

CONFTEST FILE:

Conftest file is used to generalize fixture and make it available to all test cases. Create a “conftest” and it will shared with all the test file present in that package.

Datadriver and parametrization can be done with return statements in tuple format.

When you define fixture scope to class only, it will run once before class is initiated and at the end.

GENERATING HTML REPORTS FOR PYTEST TESTCASES:

py.test –html = report.html

LOGGING IN PYTHON TESTS:

import logging  
  
def test\_loggingDemo:  
  
 logger = logging.Logger(\_\_name\_\_)  
  
 fileHandler = logging.FileHandler("logfile.log")  
  
 formatter = logging.Formatter("%(asctime)s : %(levelname)s : %(name)s : %(message)s")  
 fileHandler.setFormatter(formatter)  
  
 logger.addHandler(fileHandler) # filehandler object  
  
 logger.setLevel(logging.INFO)  
 logger.debug("a debug stmt is executed")  
 logger.info("info stmt")   
 logger.warning("something is in warning mode")  
 logger.error("an error has occurred")  
 logger.critical("critical issue")

SELENIUM PYTHON FRAMEWORK IMPLEMENTATION FROM SCRATCH:

INTRODUCTION TO GIT:

|  |  |  |
| --- | --- | --- |
| Git task | Notes | Git commands |
| [**Tell Git who you are**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config) | Configure the author name and email address to be used with your commits.  Note that Git [strips some characters](http://stackoverflow.com/questions/26159274/is-it-possible-to-have-a-trailing-period-in-user-name-in-git/26219423#26219423) (for example trailing periods) from user.name. | git config --global user.name "Sam Smith"  git config --global user.email sam@example.com |
| [**Create a new local repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-init) |  | git init |
| [**Check out a repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone) | Create a working copy of a local repository: | git clone /path/to/repository |
| For a remote server, use: | git clone username@host:/path/to/repository |
| [**Add files**](https://www.atlassian.com/git/tutorials/saving-changes#git-add) | Add one or more files to staging (index): | git add <filename>  git add \* |
| [**Commit**](https://www.atlassian.com/git/tutorials/saving-changes#git-commit) | Commit changes to head (but not yet to the remote repository): | git commit -m "Commit message" |
| Commit any files you've added with git add, and also commit any files you've changed since then: | git commit -a |
| [**Push**](https://www.atlassian.com/git/tutorials/syncing#git-push) | Send changes to the master branch of your remote repository: | git push origin master |
| [**Status**](https://www.atlassian.com/git/tutorials/inspecting-a-repository#git-status) | List the files you've changed and those you still need to add or commit: | git status |
| [**Connect to a remote repository**](https://www.atlassian.com/git/tutorials/syncing#git-remote) | If you haven't connected your local repository to a remote server, add the server to be able to push to it: | git remote add origin <server> |
| List all currently configured remote repositories: | git remote -v |
| [**Branches**](https://www.atlassian.com/git/tutorials/using-branches) | Create a new branch and switch to it: | git checkout -b <branchname> |
| Switch from one branch to another: | git checkout <branchname> |
| List all the branches in your repo, and also tell you what branch you're currently in: | git branch |
| Delete the feature branch: | git branch -d <branchname> |
| Push the branch to your remote repository, so others can use it: | git push origin <branchname> |
| Push all branches to your remote repository: | git push --all origin |
| Delete a branch on your remote repository: | git push origin :<branchname> |
| [**Update from the remote repository**](https://www.atlassian.com/git/tutorials/syncing) | Fetch and merge changes on the remote server to your working directory: | git pull |
| To merge a different branch into your active branch: | git merge <branchname> |
| View all the merge conflicts:  View the conflicts against the base file:  Preview changes, before merging: | git diff  git diff --base <filename>  git diff <sourcebranch> <targetbranch> |
| After you have manually resolved any conflicts, you mark the changed file: | git add <filename> |
| **Tags** | You can use tagging to mark a significant changeset, such as a release: | git tag 1.0.0 <commitID> |
| CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using: | git log |
| Push all tags to remote repository: | git push --tags origin |
| [**Undo local changes**](https://www.atlassian.com/git/tutorials/undoing-changes) | If you mess up, you can replace the changes in your working tree with the last content in head:  Changes already added to the index, as well as new files, will be kept. | git checkout -- <filename> |
| Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this: | git fetch origin  git reset --hard origin/master |
| **Search** | Search the working directory for foo(): | git grep "foo()" |