Testing

→**Unit Testing:** The process of testing whether a particular unit is working properly or not is called Unit Testing.

→**Integration Testing:** The process of testing total application (End to End Testing).

Testing Terminology:

→Test Scenario

→Test Case

→Test Suite

Example: Gmail Application

Testing Login Functionality→Test Scenario

1)Valid Username & Valid Password→Test Case 1

2)Valid Username & Invalid Password→Test Case 2

3)Invalid Username & Valid Password→Test Case 3

4)Invalid Username & Invalid Password→Test Case 4

5)Empty Username & Empty Password→Test Case 5

→Grouping of Test cases is called Test Suite.

How to Perform Unit Testing in Python:

→Module Name: unittest

→Class Name: TestCase

→Instance Methods:

1) setUp()

2) test()

3) teardown()

→Class level Methods:

1) setUpClass(cls)

2) tearDownClass(cls)

Examples:

→**Example 1:**

```
import unittest
Class TestCaseDemo(unittest.TestCase):
    def setUp(self):
        print('Setup method execution')
    def test(self):
        print('test method execution')
    def tearDown(self):
        print('tearDown method execution')
unittest.main()
```

Output:

```
Setup method execution
```

test method execution

tearDown method execution

→**Example 2:**

```
import unittest
Class TestCaseDemo(unittest.TestCase):
    def setUp(self):
        print('Setup method execution')
    def test_method1(self):
        print('test method1 execution')
    def test_method2(self):
        print('test method2 execution')
    def tearDown(self):
        print('tearDown method execution')
unittest.main()
```

Output:

```
Setup method execution
test method1 execution
tearDown method execution
Setup method execution
test method2 execution
```

tearDown method execution

→**Example 3:**

```
import unittest
Class TestCaseDemo(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
       print('Setup class method execution')
    def setUp(self):
        print('Setup method execution')
    def test_method1(self):
        print('test method1 execution')
    def test_method2(self):
        print('test method2 execution')
    def tearDown(self):
        print('tearDown method execution')
    @classmethod
    def tearDownClass(cls):
       print('tearDown class method execution')
unittest.main()
```

Output:

Setup class method execution

Setup method execution

test method1 execution

tearDown method execution

Setup method execution

test method2 execution

tearDown class method execution

→**Selenium:** pip install selenium

→Download web driver from seleniumhq.org

Browser Interaction and Navigation of Web Pages:

→driver=webdriver.Firefox(executable_path= 'path/exe')

1) **driver.get(url)** : to open specified url.

2) **driver.maximize_window()** : to maximize window.

3) **driver.title**

4) **driver.current_url**

5) **driver.refresh() / driver.get(driver.current_url)**

6) **driver.forward()** : goes one step forward in the browser's history.

7) **driver.back()** : goes one step backward in the browser's history.

8) **driver.close()** : to close current window

9) **driver.quit()** : to close associated windows.

How to locate Web elements?

→driver.find_element_by_id('id')

→driver.find_element_by_name('name')

→driver.find_element_by_xpath('xpath')

→driver.find_element_by_css_selector('css')

→driver.find_element_by_link_text('text')

Or

→driver.find_element(By.ID,'id')

→driver.find_element(By.NAME,'name')

→driver.find_element(By.LINK_TEXT,'txt')

→driver.find_element(By.CSS_SELECTOR,'css')

Example:

→**Python Program to test Google Search Functionality by using selenium**

```
from selenium import webdriver
import unittest
import time
class GoogleSearch(unittest.TestCase):
```

```
    def setUp(self):
        global driver
    driver=webdriver.Firefox(executable_path='D:\library\geckodriver.exe')
        driver.get('http://www.google.com')
        driver.maximize_window()
    def test(self):
      driver.find_element_by_name('q').send_keys('Mahesh Babu')
       time.sleep(5)
       driver.find_element_by_name('btnK').click()
        time.sleep(5)
      driver.find_element_by_class_name('LC201b').click()
      time.sleep(10)
    def tearDown(self):
      driver.close()
unittest.main()
```

→ **HMS login & logout using selenium**

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

```
import unittest
import time
class HMSLoginLogout(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        global driver
    driver=webdriver.Firefox(executable_path='D:\library\geckodriver.exe')
        driver.get('http://www.seleniumbymahesh.com/')
        driver.maximize_window()
    def test_login(self):
        driver.find_element(By.LINK_TEXT,'HMS').click()
        time.sleep(5)
    driver.find_element(By.NAME,'username').send_keys('admin')
    driver.find_element(By.NAME,'password').send_keys('admin')
    driver.find_element(By.NAME,'submit').click()
    time.sleep(10)
    def test_logout(self):
        driver.find_element(By.LINK_TEXT,'Logout').click()
```

```
        time.sleep(10)
    @classmethod
    def tearDownClass(cls):
        driver.close()
unittest.main()
```

Limitations of Unit Testing:

→Test results will be displayed only to the console and it is not possible to generate reports.

→Unittest framework always executes test methods in alphabetical order only and it is not possible to customize execute order.

→As the part of batch execution(Test Suite), all test methods from the specified Test Case classes will be executed and it is not possible to specify particular methods.

→In unit testing only limited setup and tearDown methods are available.

1) setUpClass() : Before executing all test methods of a Test Case class.
2) tearDownClass() : After executing all test methods of a Test Case class.
3) setUp() : Before every test method execution
4) tearDown() : After every test method execution

If we want to perform any activity before executing test suite and after executing test suite, unit test framework does not define any methods.

PyTest Framework

Pytest Naming Conventions:

→File name should starts or ends with 'test'.

Ex: test_google_search.py , google_search_test.py

→Class name should starts with 'Test'.

Ex: TestGoogleSearch, TestCaseDemo

→ Test method name should starts with 'test_'.

Ex: test_method()

→To display print statements in console use - s and – v.

Various possible ways to run pytest scripts:

→py.test - v – s : to run all test methods present in all test scripts of current working directory.

→ py.test - v – s test.py :to run all test methods of a particular test script.

→ py.test - v – s test.py test1.py : to run multiple test scripts.

→ py.test - v – s test.py ::test_methodB : to run a particular test method.

→**pip install pytest-ordering :** for customize ordering

→**pip install pytest-html :** to generate test reports.

Django Testing: django.test

Refer documentation