

Module 5: unittest for Python



Module Objectives

At the end of this module, you will be able to:

- Write Selenium WebDriver tests using unittest class
- Implement a test using the TestCase class
- Discuss assert methods
- Create TestSuite for a group of tests
- Explain generating test reports in HTML



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

Running the test

Topic List

Adding Another Test

Setup() and teardown() Methods at Class Level

Assertions

Test Suite

Generating the HTML Test Report

Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

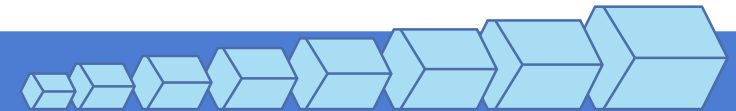
Running the test

Writing Tests for unittest

Why unittest?



- Enables to create pre conditions and post conditions code for tests.
- Performs Validations using assert functions.
- Helps in creating test cases, test suites and test fixtures.



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

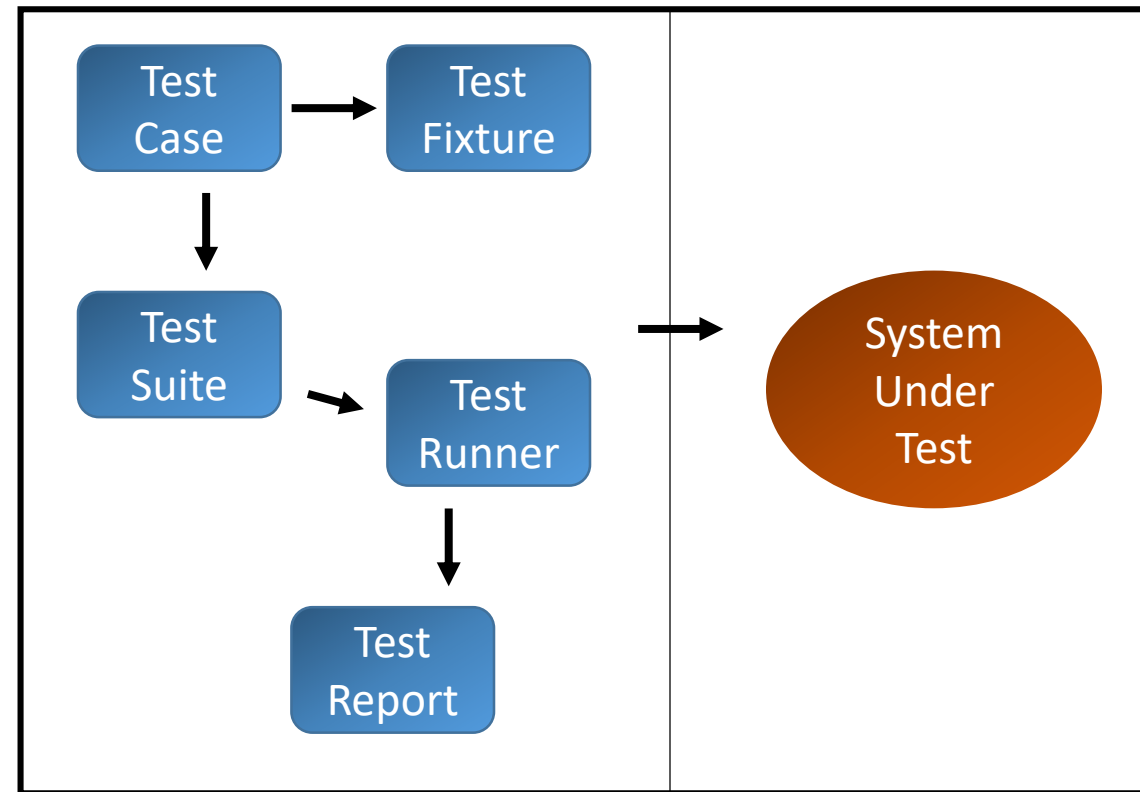
Cleaning Up The Code

Running the test

unittest Library (1)

Unittest Framework

- The unittest library is called as PyUnit (inspired by Junit)



unittest Library (2)

unittest Framework

- Test Fixture:

Represents the memory allocation and clean up actions for one or more tests.

For Example : Starting a server process

- Test Case

- Base class represents the smallest unit of testing
- Validates responses by various assert methods

- Test Suite

- Implemented by TestSuite class
- Allows individual and also the test suites to be aggregated
- Once the suite is executed it allows all test added directly to the suite or as child test suites to be run



unittest Library (3)

unittest Framework

- Test Runner
 - Provides result to end user by managing the execution of tests
 - The runner uses graphical interface or a textual interface, or it returns special value to indicate the result
 - Test runner object provides “run()” method which accepts TestCase or TestSuites objects as parameters
 - TestResult class saves the result object returned by run()
- Test Report
 - Displays the summary of result(pass/fail, expected vs actual and timings)



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

Running the test

TestCase Class (1)

Test Case and Test Fixture

The Test Case and Test Fixture are supported through "TestCase" and "FunctionTestCase" classes

- Instance of a TestClass represents the smallest testable unit
- TestCase class is used when creating new tests
- TestCase instances are used to run a single test method and for each test a new fixture is created
- FunctionTestCase is used when integrating existing code with unittest framework.

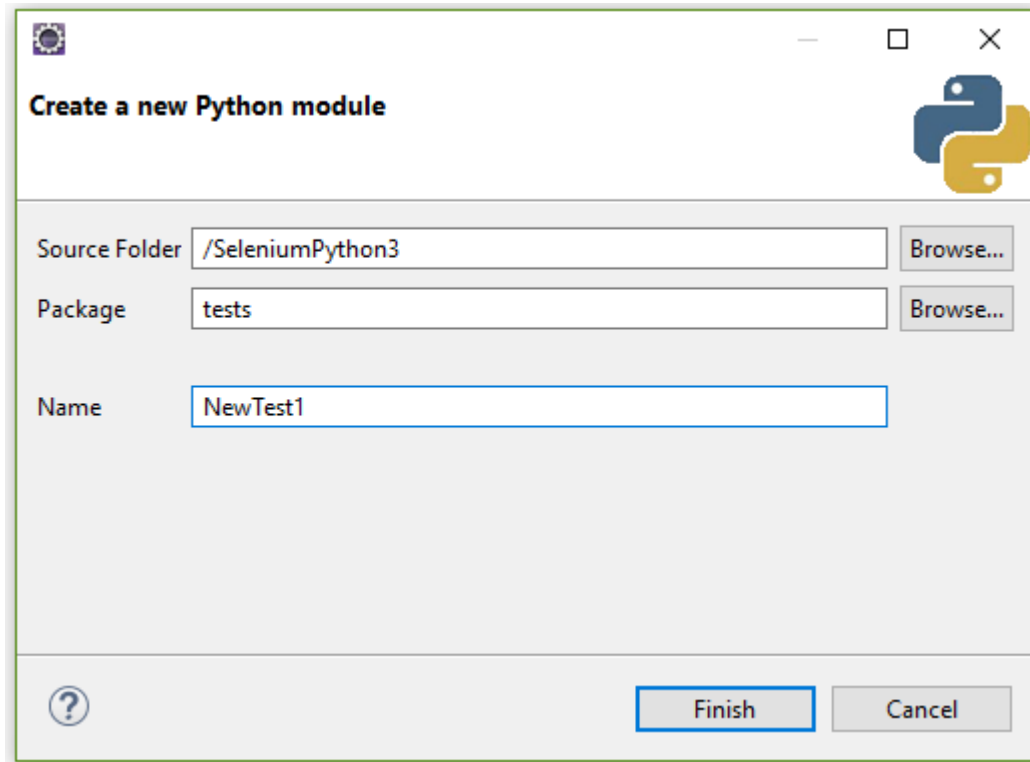
Building Test Fixtures

- Using TestCase class: The overridden setUp() and tearDown() methods are used for initialize and clean
- Using FunctionTestCase class : Existing functions are passed to the constructor

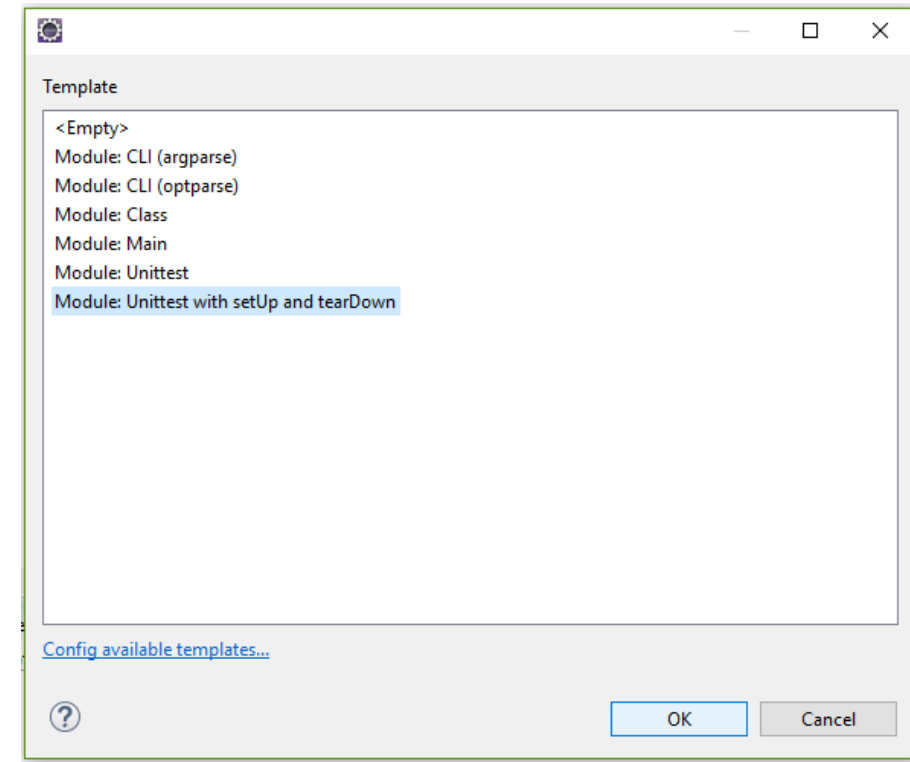


TestCase Class (2)

How to create a TestCase class in pydev



Step 1: Create pydev module by providing the Name and package



Step 2: A pop appears to select the template for module. Now select unittest / unittest with setUp and tearDown



TestCase Class (3)

The below shown is the template for the module unittest with setUp and tearDown

```
'''
Created on Feb 25, 2018

@author: aswani.kumar.avilala
'''
import unittest

class Test(unittest.TestCase):

    def setUp(self):
        pass

    def tearDown(self):
        pass

    def testName(self):
        pass

if __name__ == "__main__":
    #import sys;sys.argv = ['', 'Test.testName']
    unittest.main()
```

Initializations

Cleaning resources

Design the Test case



TestCase Class (4)

Prepare A Test Case Class To Automate A Scenario

- A test case is created by extending `unittest.TestCase`
- To add a case, provide the corresponding test method (a handler) to the derived class
- Finalizing a test case is done by `assert` or any of its variations for reporting test status
- Most common assert functions are
 - `assertEquals()` - to check for an expected result
 - `assertTrue()` - to verify a condition
 - `assertRaises()` – to verify expected exception is raised

```
import unittest
from selenium import webdriver

class SearchText(unittest.TestCase)
```



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

Running the test

setUp() (1)

Using setUp() Method To Manage Test Pre-Requisites

setUp() method is an entry point for tests

- It is used to run a fixed set of actions before executing tests
- These are pre-requisites which may include the following test setup preparation tasks
 - 1. Create an instance of a browser driver
 - 2. Navigate to a base URL
 - 3. Load tests data for execution
 - 4. Open log files for recording inputs, status, and errors
- This method takes no arguments and returns nothing
- The runner will call setUp() method first before running any of the test handlers

setUp() (2)

Example

```
import unittest

from selenium import webdriver

class SearchText(unittest.TestCase):
    def setUp(self):
        # create a new Firefox session
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.driver.maximize_window()
        # navigate to the application home page
        self.driver.get("http://www.google.com/")
```



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

Running the test

Writing Tests

An Example

- Test methods are implemented in the TestCase class.
- It is important name test methods beginning with the word test.
- The naming convention informs the test runner about the methods which represents test.

```
def test_search_by_text(self):  
    # get the search textbox  
    self.search_field = self.driver.find_element_by_name("q")  
    # enter search keyword and submit  
    self.search_field.send_keys("Selenium WebDriver Interview questions")  
    self.search_field.submit()  
    #get the list of elements which are displayed after the search  
    #currently on result page using find_elements_by_class_name method  
    lists = self.driver.find_elements_by_class_name("r")  
    no=len(lists)  
    self.assertEqual(10, len(lists))
```



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

Running the test

Cleaning Up the Code

Cleanup Strategy to Free Resources Post Test Execution

- `tearDown()` method is called after the test is executed
- It is used to clean up the resources post execution
- Test Runner calls this method when tests are executed

```
def tearDown(self):  
    # close the browser window  
    self.driver.quit()
```



Topic List

Writing Tests Using unittest

unittest library

TestCase Class

setUp()

Writing Tests

Cleaning Up The Code

Running the test

Running the Test (1)

Run the Test from Command Line

1. Running the test needs to add a call to the `main()` method in the test script
2. Pass a verbosity argument to the `main()`
3. The code which facilitates command line execution is

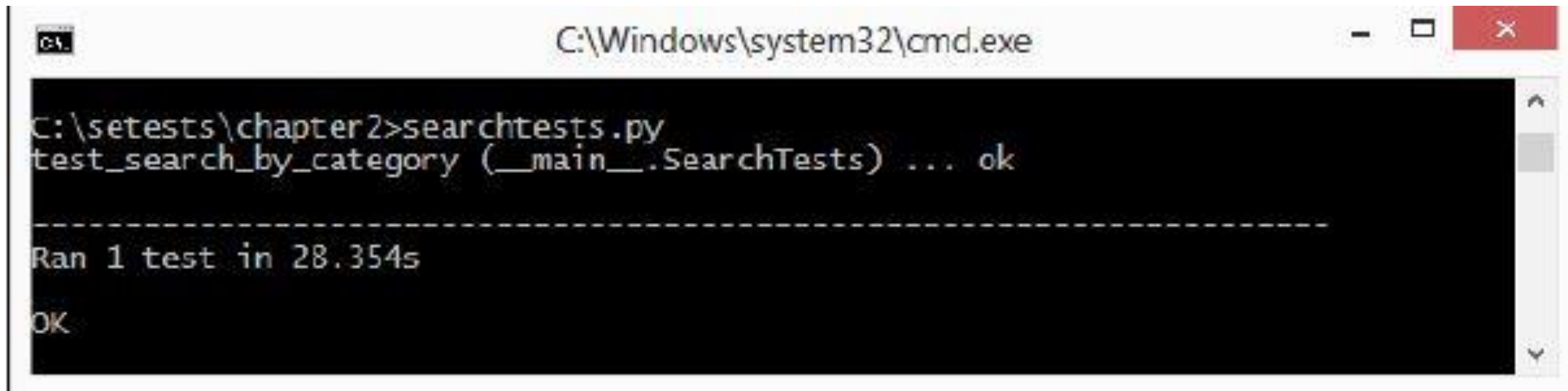
```
if __name__ == '__main__':  
    unittest.main()
```

4. Save the sample test as normal python script by giving the extension “**.py**” (searchTest.py)
5. Execute the file in command line by “ **python searchTest.py** ”



Running the Test (2)

Run the Test from Command Line: Test Case Success



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The command prompt shows the following output:

```
C:\setests\chapter2>searchtests.py
test_search_by_category (__main__.SearchTests) ... ok

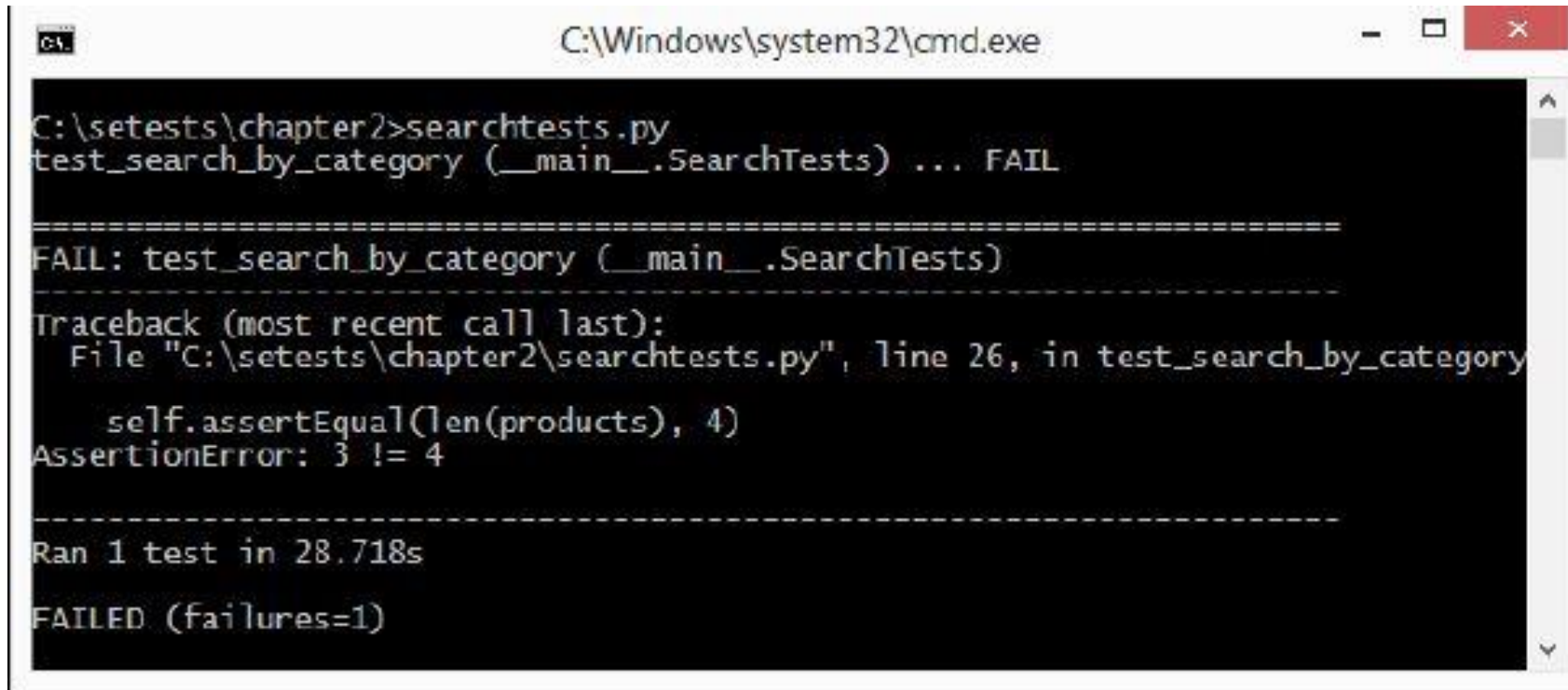
-----
Ran 1 test in 28.354s

OK
```



Running the Test (3)

Run the test from command line- Test Case Failed

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The command prompt shows the execution of a Python script "searchtests.py" in the directory "C:\setests\chapter2". The output indicates a test failure for "test_search_by_category (__main__.SearchTests)". A detailed traceback shows the error occurred in "searchtests.py" at line 26, where an assertion failed: "self.assertEqual(len(products), 4)" because the actual value was 3. The final output shows "Ran 1 test in 28.718s" and "FAILED (failures=1)".

```
C:\Windows\system32\cmd.exe

C:\setests\chapter2>searchtests.py
test_search_by_category (__main__.SearchTests) ... FAIL

=====
FAIL: test_search_by_category (__main__.SearchTests)
=====
Traceback (most recent call last):
  File "C:\setests\chapter2\searchtests.py", line 26, in test_search_by_category
    self.assertEqual(len(products), 4)
AssertionError: 3 != 4

=====
Ran 1 test in 28.718s

FAILED (failures=1)
```



Exercise: 5.1: Test Execution for New Tours demo

Scenario : Create a test class with a new test function to check the login functionality along with setUp () and tearDown() functions in the same test class

To Do

1. Create a setUp() life cycle function
2. Create a test function
3. Create a tearDown() life cycle function
4. Execute the Script as python unittest



Refer Exercise 5.1 in Selenium with Python_wbk.doc for the detailed steps.

Topic List

Adding Another Test

Setup() and teardown() Methods at Class Level

Assertions

Test Suite

Generating the HTML Test Report

Adding Another Test (1)

Adding More Selenium Python Test Case

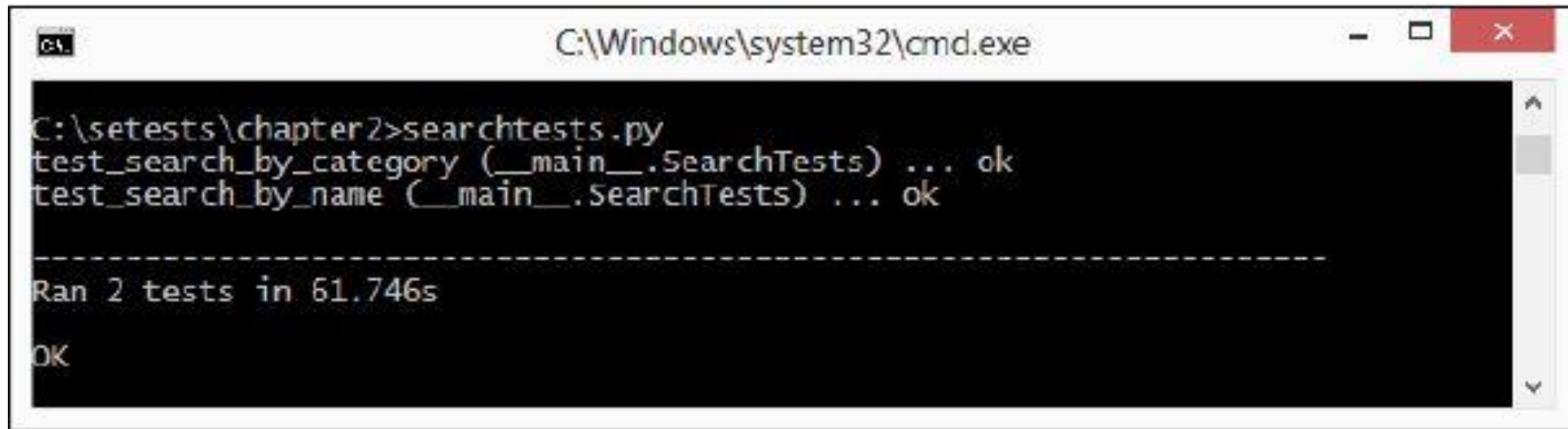
- we can add as many cases as expected in the TestCase class
- helps in creating logical groups of tests that are related to a specific functionality
- Name the new method starting with the word test
- Run the test shows two instances of Firefox opening and closing.

```
def test_search_by_name(self):  
    # get the search textbox  
    self.search_field = self.driver.find_element_by_name("q")  
    # enter search keyword and submit  
    self.search_field.send_keys("Python class")  
    self.search_field.submit()  
    #get the list of elements which are displayed after the search  
    #currently on result page using find_elements_by_class_name method  
    list_new = self.driver.find_elements_by_class_name("r")  
    self.assertEqual(10, len(list_new))
```



Adding Another Test (2)

Adding More Selenium Python Test Case- Result



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the execution of a Python script named "searchtests.py" from the directory "C:\setests\chapter2". The script runs two test cases: "test_search_by_category (__main__.SearchTests) ... ok" and "test_search_by_name (__main__.SearchTests) ... ok". A dashed line separates the test results from the summary. The summary indicates that 2 tests were run in 61.746 seconds, resulting in an "OK" status.

```
C:\setests\chapter2>searchtests.py
test_search_by_category (__main__.SearchTests) ... ok
test_search_by_name (__main__.SearchTests) ... ok

-----
Ran 2 tests in 61.746s

OK
```



Topic List

Adding Another Test

Setup() and teardown() Methods at Class Level

Assertions

Test Suite

Generating the HTML Test Report

setUp() and tearDown() Methods at Class Level (1)

Refactoring setUp() And tearDown() Methods For Optimization

- can share a single Firefox instance between the methods instead of creating a new instance every time.
- Achieved by using the setUpClass() and tearDownClass() methods along with the @classmethod decorator.
- This methods allows to initialize values at the class level instead of the method level
- It shares the values between the test methods



setUp() and tearDown() Methods at Class Level (2)

Refactoring setUp() And tearDown() Methods For Optimization

```
class SearchText(unittest.TestCase):
    @classmethod
    def setUpClass(inst):
        # create a new Firefox session

        code for test 1

        code for test 2
    @classmethod
    def tearDownClass(inst):
        # close the browser window
        inst.driver.quit()

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



Topic List

Adding Another Test

Setup() and teardown() Methods at Class Level

Assertions

Test Suite

Generating the HTML Test Report

Assertions (1)

Assertions In Python unittest Framework

Python unittest library implements a list of assert methods

- Used to compare actual values returned by the application with the expected values
- To continue with the execution of the test, assert methods must return true

Three types of assert are available.

1. Checking equivalence
2. Logical comparison
3. Acting in the case of Exceptions

Assertions (2)

List Of Assert Methods In Python unittest

Assert Method	Test Condition	Explanation
<code>assertEqual(a, b [,msg])</code>	<code>a==b</code>	Check whether or not “a” and “b” match with each other. You can also pass a cusotm error message.
<code>assertNotEqual(a,b[,msg])</code>	<code>a!=b</code>	e.g. <code>assertEqual(element.text,"10")</code>
<code>assertTrue(x[,msg])</code>	<code>bool(x) is True</code>	Verify if the given expression evaluates to True or False.
<code>assertFalse(x[,msg])</code>	<code>bool(x) is False</code>	e.g. <code>assertTrue(element.is_displayed())</code>
<code>assertIsNot(a, b[,msg])</code>	<code>a is not b</code>	



Assertions (3)

List Of Assert Methods In Python unittest

Assert Method	Test Condition	Explanation
assertRaises(exc, fun, *args, **kwds)	fun(*args,**kwds)	Check whether the test step raises the specific Exception mentioned. One such example is to use this method to check <NoSuchElementException>.
	raises exc	
assertRaisesRegexp(exc, r, fun, *args, **kwds)	fun(*args,**kwds)raises excand themessagematchesregex r	
assertAlmostEqual(a, b)	round(a-b,7) == 0	It compares the numeric values after rounding them to the number in the second argument.
assertNotAlmostEqual(a,b)	round(a-b,7) != 0	
assertGreater(a, b)	a > b	These methods are similar to the assertEquals() method.
assertGreaterEqual(a,b)	a>=b	
assertLess(a,b)	a<b	
assertLessEqual(a,b)	a<=b	



Assertions (4)

List Of Assert Methods In Python unittest

Assert Method	Test Condition	Explanation
assertRegexpMatches(s, r)	r.search(s)	Verify whether a regexpsearch matches the text.
assertNotRegexpMatches(s, r)	not r.search(s)	
assertMultiLineEqual(a,	strings	This method is an extension to the assertEquals(), designed for multilinestrings.
assertListEqual(a, b)	lists	This method checks whether the lists “a” and “b” match. It helps to work with the drop-down fields.
fail()		This method fails the test unconditionally. It allows the creation of custom conditional blocks.



Topic List

Adding Another Test

Setup() and teardown() Methods at Class Level

Assertions

TestSuite

Generating the HTML Test Report

TestSuite (1)

Create Selenium Python Test Suite Using unittest

- Using the TestSuite feature of unittest, we can collect various tests into logical groups and then into a unified test suite that can be run with a single command
- This is done by using the TestSuite, TestLoader, and TestRunner classes.



TestSuite (2)

Add Multiple Test Methods using Different TestCase classes

```
'''  
Created on Feb 21, 2018  
@author: aswani.kumar.avilala  
'''  
import unittest  
  
class NewTest1(unittest.TestCase):  
  
    def testName1(self):  
        print("in test Name 1")  
        pass  
    def testName2(self):  
        print("in test Name 2")  
  
if __name__ == "__main__":  
    #import sys;sys.argv = ['', 'Test.testName']  
    unittest.main()
```

```
'''  
Created on Feb 21, 2018  
@author: aswani.kumar.avilala  
'''  
import unittest  
  
class NewTest2(unittest.TestCase):  
  
    def testName3(self):  
        print("in test Name3")  
        pass  
    def testName4(self):  
        print("in test Name4")  
        pass  
  
if __name__ == "__main__":  
    #import sys;sys.argv = ['', 'Test.testName']  
    unittest.main()
```



TestSuite (3)

Create a TestRunner Class to Execute the Tests as a Suite

```
1 '''
2 Created on Feb 21, 2018
3
4 @author: aswani.kumar.avilala
5 '''
6 import unittest
7 from NewTest1 import NewTest1
8 from NewTest2 import NewTest2
9
10 #loader=unittest.TestLoader()
11 suite=unittest.TestSuite()
12 '''
13 To execute the tests as per the order of Test Case methods.
14 Pass the test method name as parameter in the test
15 '''
16 suite.addTest(NewTest1('testName2'))
17 suite.addTest(NewTest2('testName4'))
18 suite.addTest(NewTest1('testName1'))
19 suite.addTest(NewTest2('testName3'))
20 '''
21 #To execute the tests as per the order of Test Case classes
22 suite.addTest(loader.loadTestsFromTestCase(NewTest1))
23 suite.addTest(loader.loadTestsFromTestCase(NewTest2))
24 '''
25 unittest.TextTestRunner().run(suite)
26
```

PyUnit Console [C:\Python36\python.exe]

```
<terminated> TestRunner.py
in test Name 2
..in test Name4
in test Name 1
in test Name3
..
```

Ran 4 tests in 0.008s



Exercise 5.2: Executing Test as Test Suite

Scenario : Execute as a Test Suite for TestCase classes like newtours.demoaut.com and TestMeApp

To Do

1. Create Mercury Tours Test as Test Case
2. Create TestMeApp as Test Case
 - Design test method to sign in mercury tours (<http://newtours.demoaut.com/>)
3. Create TestRunner class which can run all the Test Cases classes using TestSuite
 - Design a test method to sign in TestMeApp (<http://localhost:5432/TestMeApp/login.htm>)
4. Execute the runner



Refer Exercise 5.2 in Selenium with Python_wbk.doc for the detailed steps.

Topic List

Adding Another Test

Setup() and teardown() Methods at Class Level

Assertions

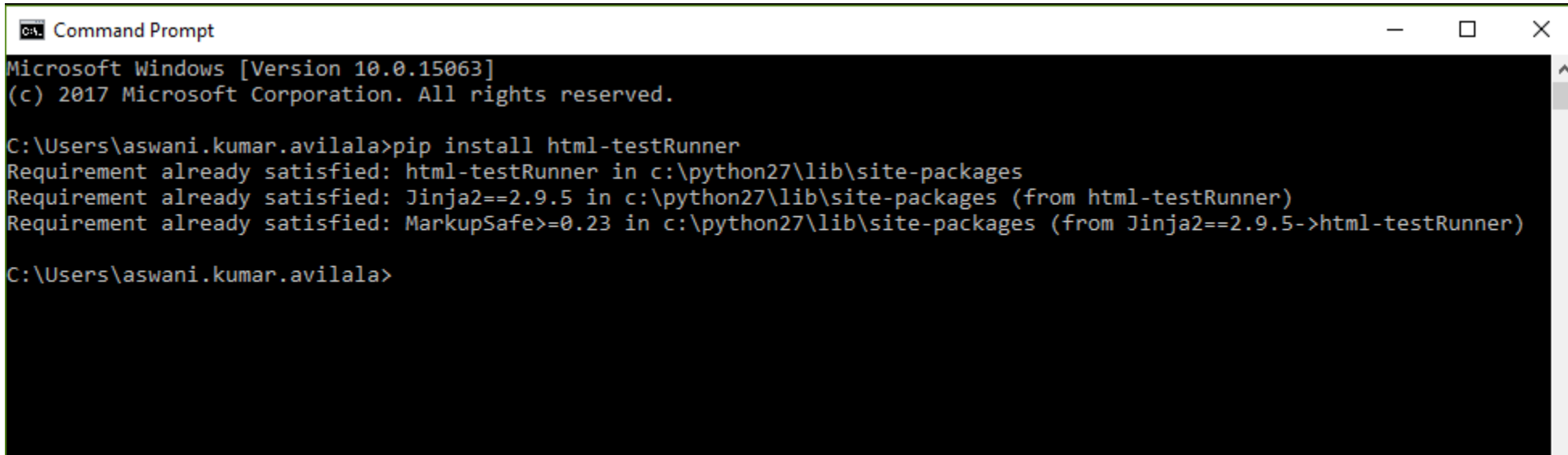
Test Suite

Generating the HTML Test Report

Generate HTML Test Report (1)

How to generate HTML Reports using HTML Test Runner?

- Default Python unittest library generates output on terminal console
- Sending console logs to the stakeholder is not a good idea
- We can use the HTMLTestRunner extension of unittest for generating the HTML reports for unit tests.
- Download the HTMLTestRunner library using pip to use HTMLTestRunner API as shown below

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The window content shows the following text:

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\aswani.kumar.avilala>pip install html-testRunner
Requirement already satisfied: html-testRunner in c:\python27\lib\site-packages
Requirement already satisfied: Jinja2==2.9.5 in c:\python27\lib\site-packages (from html-testRunner)
Requirement already satisfied: MarkupSafe>=0.23 in c:\python27\lib\site-packages (from Jinja2==2.9.5->html-testRunner)

C:\Users\aswani.kumar.avilala>
```



Generate HTML Test Report (2)

Generate the HTML report using HTMLTestRunner for the Test Suite as shown below

```
'''
Created on Feb 21, 2018

@author: aswani.kumar.avilala
'''

import unittest
from NewTest1 import NewTest1
from NewTest2 import NewTest2
import HtmlTestRunner

#loader=unittest.TestLoader()
suite=unittest.TestSuite()
'''

To execute the tests as per the order of Test Case methods.
Pass the test method name as parameter in the test
'''

suite.addTest(NewTest1('testName2'))
suite.addTest(NewTest2('testName4'))
suite.addTest(NewTest1('testName1'))
suite.addTest(NewTest2('testName3'))
'''

#To execute the tests as per the order of Test Case classes
suite.addTest(loader.loadTestsFromTestCase(NewTest1))
suite.addTest(loader.loadTestsFromTestCase(NewTest2))
'''

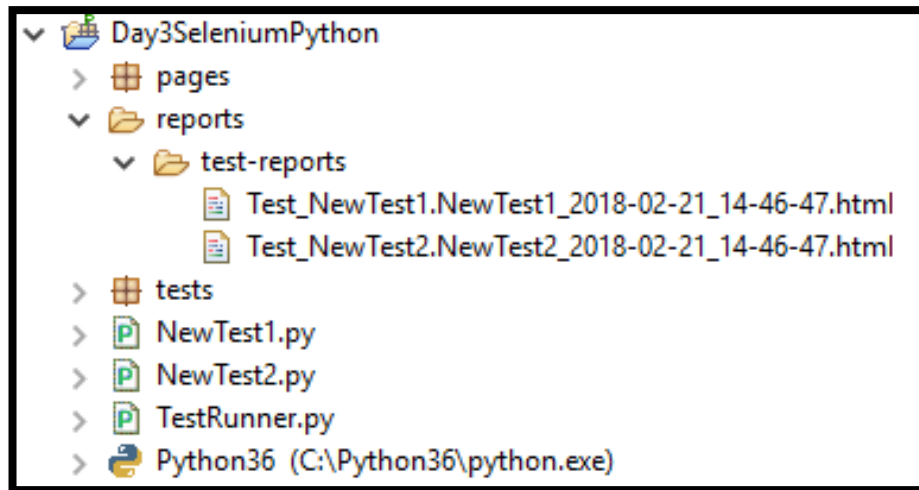
#unittest.TextTestRunner().run(suite)
runner=HtmlTestRunner.HTMLTestRunner(output='test-reports',report_title='Test Report')
runner.run(suite)
```



Generate HTML Test Report (3)

Run the Test Suite to View HTML Test Reports

- View in the reports folder under the current project



Note : After execution, refresh the project to view the reports folder



Generate HTML Test Report (4)

The HTML Test Reports

NewTest1NewTest2TestRunnerTest Report

19/Day3SeleniumPython/reports/test-reports/Test_NewTest1.NewTest1_2018-02-25_23-56-41.html

Test Report

Start Time: 2018-02-25 23:56:41

Duration: 0:00:00

Status: Pass: 2

NewTest1.NewTest1	Status
testName2 (NewTest1.NewTest1)	Pass
testName1 (NewTest1.NewTest1)	Pass

Total Test Runned:Pass: 2

Test Report

19/Day3SeleniumPython/reports/test-reports/Test_NewTest2.NewTest2_2018-02-25_23-56-41.html

Test Report

Start Time: 2018-02-25 23:56:41

Duration: 0:00:00

Status: Pass: 2

NewTest2.NewTest2	Status
testName4 (NewTest2.NewTest2)	Pass
testName3 (NewTest2.NewTest2)	Pass

Total Test Runned:Pass: 2



Module Summary

Now, you should be able to:

Write Selenium WebDriver tests using unittest class

- Implement a test using the TestCase class
- Discuss assert methods
- Create TestSuite for a group of tests
- Explain generating test reports in HTML



Thank You