Software Testing

Learning Objectives

The learning objectives are to

- To perform Software Testing with with PyTest and UnitTest
- To learn how to write efficient testing code in Python

Testing : A Simple approach with print statement


```
def add(x,y):
  return x+y
def substract(x, y):
    return x-y
def multiply(x, y):
    return x+y
def divide(x,y):
    return x/y
def max(x,y,z):  m = x
  if(y>m):
    m=y
  if(z>m):
    m=z
  return
```

HW1.py

```
print ("addition", add(4,3))
print ("substraction", substract(4,3))
print ("mulitplication", multiply(4,3))
print ("division", divide(4,3))
print ("maximum", max(4,3,1))
```

Limitations

- Computational logic and test code is intermixed.
- Test code and computational logic are difficult to maintain.
- We need a more cleaner approach to write test code.

Testing : Through Testing Framework


```
import unittest
class TestCalculator(unittest.TestCase):
```

```
def test_add(self):
        '''Test case function for addition'''
        result = add(4, 7)
        expected = 11
        self.assertEqual(result, expected)
```

```
def test_substract(self):
        '''Test case function for subtraction'''
        result = subtract( 7,4)
        expected = 3
        self.assertEqual(result, expected)
```


```
import unittest
class TestCalculator(unittest.TestCase):
```

```
def test_multiply(self):
        '''Test case function for multiplication'''
        result = multiply(4, 7)
        expected = 28
        self.assertEqual(result, expected)
```

```
def test_divide(self):
        '''Test case function for division'''
        result = subtract(10,2)
        expected = 5
        self.assertEqual(result, expected)
```

Focus on Code Coverage

```
def test_max(self):
        '''Test case function for maximum'''
        result = max(10,7,2)
        expected = 10
        self.assertEqual(result, expected)
         result = max(7,10,2)
        expected = 10
        self.assertEqual(result, expected)
         result = max(2,7,10)
         expected = 10
         self.assertEqual(result, expected)
```

OOP in Python: A Simple Example


```
class Student:
def __init__(self):
    self._ca = 0
    self._mse =0
    self._ese =0
    self._name =None
```

Definition of getter and setter methods

```
@property
def ca(self):
    print("getter method called")
    return self._ca

@ca.setter
def ca(self, m):
    print("Setter method called")
    if m < 0 or m > 20 :
    raise ValueError("Marks are not within
    range(0-20)")
    else:
    self._ca = m
```


```
class Student:
def __init__(self):
    self._ca = 0
    self._mse =0
    self._ese =0
    self._name =None
```

Mid-sem
Exam methods

```
@property
def mse(self):
    print("getter method called")
    return self._mse

@mse.setter
def mse(self, m):
    print("Setter method called")
    if m < 0 or m > 20 :
    raise ValueError("Marks are not within
    range(0-20)")
    else:
    self._mse= m
```

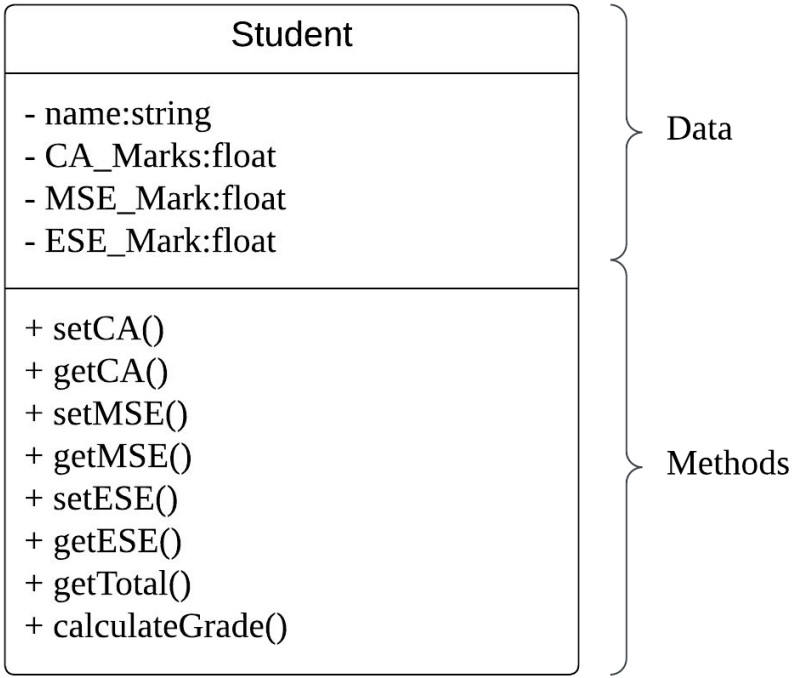
ESE exam and
Total method

```
def total(self, a,b,c):
    return self._ca +
self._mse + self._ese
```

```
@property
def ese(self):
    print("getter method called")
    return self._ese

@ese.setter
def ese(self, m):
    print("Setter method called")
    if m < 0 or m > 60 :
    raise ValueError("Marks are not within
    range(0-60)")
    else:
    self._ese= m
```

OO Testing (inefficient way)




```
class TestStudent(unittest.TestCase):
def test_ca(self):
    self.awk = Student()
    self.awk.ca = 15
    self.assertEqual(self.awk.ca, 15)
```

```
def test_mse(self):
    self.awk = Student()
    self.awk.mse = 15
    self.assertEqual(self.awk.mse,
    15)

def test_ese(self):
    self.awk = Student()
    self.awk.ese = 55
    self.assertEqual(self.awk.ese,
    55)
```

All test methods have its own data and testing code


```
def test_total(self):
    self.awk = Student()
    self.awk.mse = 10
    self.awk.ese = 40
    self.awk.ca = 10
    sum = self.awk.total(self.awk.ca,
    self.awk.mse,self.awk.ese )
    self.assertEqual(sum, 60)
```

All test methods have its own data and testing code

OO Testing (Efficient way)

All test methods have its own data and testing code

```
def setUp(self):
    self.awk = Student()
    self.awk.mse = 10
    self.awk.ese = 40
    self.awk.ca = 10
```

```
def test_ca(self):
    self.assertEqual(self.awk.ca, 10)

def test_mse(self):
    self.assertEqual(self.awk.mse,
10)

def test_ese(self):
    self.assertEqual(self.awk.ese,
40)

def test_total(self):
    sum = self.awk.total(self.awk.ca,
    self.awk.mse,self.awk.ese )
```

Testing Exception code

```
def test_caValueError(self):
    with self.assertRaises(ValueError):
        self.sanil = Student()
        self.sanil.ca =25
def test_mseValueError(self):
    with self.assertRaises(ValueError):
        self.sanil = Student()
        self.sanil.mse =25
def test_eseValueError(self):
    with self.assertRaises(ValueError):
        self.sanil = Student()
        self.sanil.mse =65
```

Assert Methods

Method	Checks that	New in
`assertEqual(a, b)`	`a == b`	
`assertNotEqual(a, b)`	`a != b`	
`assertTrue(x)`	`bool(x) is True`	
`assertFalse(x)`	`bool(x) is False`	
`assertIs(a, b)`	`a is b`	3.1
`assertIsNot(a, b)`	`a is not b`	3.1
`assertIsNone(x)`	`x is None`	3.1
`assertIsNotNone(x)`	`x is not None`	3.1
`assertIn(a, b)`	`a in b`	3.1
`assertNotIn(a, b)`	`a not in b`	3.1
`assertIsInstance(a, b)`	`isinstance(a, b)`	3.2
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`	3.2

Assert Methods

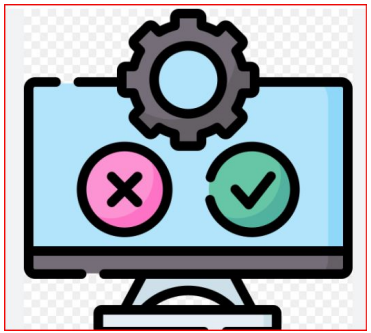
```
import unittest
class TestCalculator(unittest.TestCase):
```

- ***unittest*** has been built into the Python standard library since version 2.1.
- ***unittest*** contains both a testing framework and a test runner. unittest

What is *unittest*

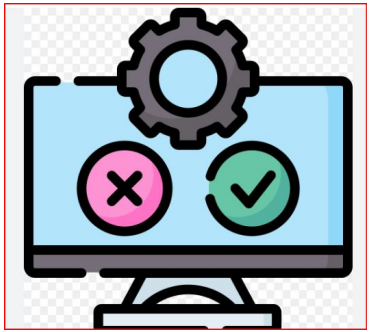
```
import unittest
class TestCalculator(unittest.TestCase):
```

- ***unittest*** has been built into the Python standard library since version 2.1.
- ***unittest*** contains both a testing framework and a test runner. unittest



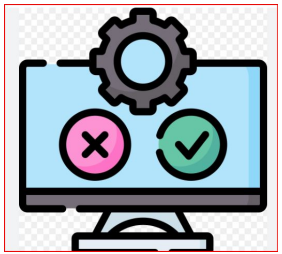
How to write testcases?

- Import unittest from the standard library
- Create a class called TestXXX that inherits from the TestCase class
- Define the test methods by adding self as the first argument
- Use the self.assertEqual() method on the TestCase class
- Change the command-line entry point to call unittest.main()



How to execute testcases?

```
unittest.main(argv=[''], verbosity=2, exit=False)
```

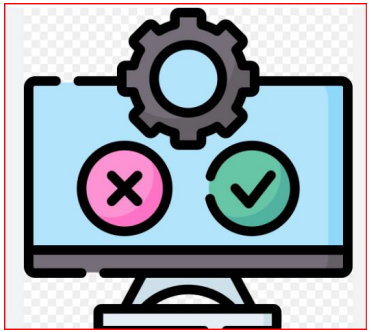


```
if __name__ == '__main__':
    unittest.main()
```

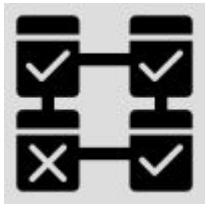
How to structure testcases?

The structure of a test should loosely follow this workflow:

1. Create your inputs
2. Execute the code being tested, capturing the output
3. Compare the output with an expected result



Types of Software Testing



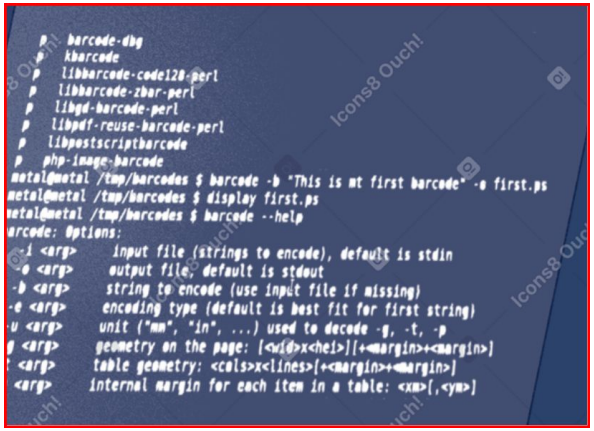
Unit testing tests the working of isolated/independent units which may be a single method or a function.

Integration testing tests the working of independent component (DB, Web Server) in the overall system

User acceptance testing is performed by users to validate the functionality of the software.

Measuring Test Coverage

- Test coverage is a metric in software testing that measures the amount of testing performed by a set of tests.
- It determines whether test cases are covering the application code and how much code is exercised when running those test cases.
- For example, if you have 10,000 lines of code and only 5,000 lines of code are tested, the coverage is 50%



What is Test fixture

- In Python, a test fixture is **a function or method that runs before and after a block of test code executes**.
- Fixtures are used to set up and tear down the test environment, and to provide reusable data to tests.

```python
def setup_module():
    print("Setting up module")

def teardown_module():
    print("Tearing down module")

def test_1():
    print("Running test 1")

def test_2():
    print("Running test 2")
```

Best practices for Software Testing in Python

- Use Descriptive Test Names
- One Assertion per Test
- Test the Edge Cases
- Use Fixtures and Setup Methods
- Use Test Coverage Analysis
- Review and Maintain Test Code

Lab Activity

- Write test case to test the functions sendOTP, validateEmailID and generateOTP functions.

Quiz time

Which of the following is the testing framework to write testcases in Python

A. PyTest
B. Unittest
C. Junit
D. A & B
E. A &B&C

Quiz time

The ***Assert statement***

A. Compares expected and actual result
B. Print error messages
C. Executes test case
D. Is a non-executable statement

Quiz time

A function or method that runs before and after a block of test code executes is called ***test fixture***

A. True
B. False

Quiz time

The user validation is an important statement in

A. System test
B. Unit testing
C. Integration testing
D. User Acceptance testing