Day – 11 Evening Assessment

Unittest

1. File : bank

class BankAccount:

def \_\_init\_\_(self, balance = 1000):

self.balance = balance

def deposit(self, amount):

if amount <= 0:

raise ValueError(“Enter a Positive Amount: “)

self.balance += amount

print(“Amount has been deposited”)

def withdraw(self, amount):

if amount > self.balance:

raise ValueError(“Insufficient Funds”)

self.balance -= amount

print(“Amount has been withdrawn”)

def check\_balance(self):

print(self.balance)

File: test\_bank

import unittest

from bank import BankAccount

class testBankAccount(unittest.TestCase):

def setup(self):

self.account = BankAccount(balance=1000)

def test\_deposit(self):

self.account.deposit(300)

self.assertEqual(self.account.check\_balance(),1300)

def test\_withdraw(self):

self.account.withdraw(500)

self.assertEqual(self.account.check\_balance(),800)

def test\_withdraw\_insufficient\_funds(self):

with self.assertRaises(ValueError) as context:

self.account.withdraw(500)

self.asserIn(“Insufficient Funds”, str(context.exception))

def test\_deposit\_negative\_amount(self):

with self.assertRaises(ValueError):

self.account.deposit(-600)

def test\_check\_balance(self):

self.asserEqual(self.account.check\_balance(),1000)

if \_\_name\_\_ == ‘\_\_main\_\_’:

unittest.main()

1. File : bank

class BankAccount:

def \_\_init\_\_(self, balance =0):

self.\_\_balance = balance

def deposit(self, amount):

if amount > 0:

self.\_\_update\_balance(amount)

def check\_balance(self):

print(self.\_\_balance)

def \_\_update\_balance(self, amount):

self.\_\_balance += amount

File: test\_bank

import unittest

from bank import BankAccount

class testBankAccount(unittest.TestCase):

def setup(self):

self.account = BankAccount(balance=100)

def test\_private\_variable\_access(self):

self.assertEqual(self.account.\_\_BankAccount\_\_balance,100)

def test\_private\_method\_call(self):

self.account.\_\_BankAccount\_\_update\_balance(50)

self.assertEqual(self.account.check\_balance(),150)

if \_\_name\_\_ == ‘\_\_main\_\_’:

unittest.main()

1. setUpClass runs before all the test methods.It is used to setup shared across all the tests.

tearDownClass runs after all test methods. It is used to cleanup the shared resources.

import unittest

import tempfile

import os

class TestFileOperations(unitest.TestCase):

@classmethod

def setupClass(cls):

print(“creating temp file….”)

cls.temp\_file = tempfile.NamedTemporaryFile(delete = False, mode = ‘w+’)

cls.temp\_file.write(“Hello, world!\n”)

cls.temp\_file.write(“Second line\n”)

cls.temp\_file.close()

@classmethod

def tearDownClass(cls):

print(“Deleting temp file….”)

os.unlink(cls.temp\_file.name)

def test\_file\_exists(self):

self.assertTrue(os.path.exists(self.temp\_file.name))

def test\_file\_content(self):

with open(self.temp\_file.name, ‘r’) as f:

lines = f.readlines()

self.assertEqual(lines[0].strip(), “Hello, World!”)

def test\_line\_count(self):

with open(self.temp\_file.name, ‘r’) as f:

lines = f.readlines()

self.assertEqual(len(lines), 2)

if \_\_name\_\_ == ‘\_\_main\_\_’:

unittest.main()

1. File: math\_utils

def square\_root(x):

if isinstance(x, str):

raise TypeError(“Input must not be a string”)

if x < 0:

raise ValueError(“Input must be non-negative”)

return x\*\*0.5

File: test\_square

import unittest

from math\_utils import square\_root

class test\_square\_root(unittest.testcase):

def test\_negative\_input(self):

with self.assertRaises(ValueError) as context:

square\_root(-4)

self.assertIn(“non-nagative”, str(context.exception))

def test\_string\_input(self):

with self.assertRaises(TypeError) as context:

square\_root(“hello”)

self.assertIn(“string”, str(context.exception))

def test\_valid\_input(self):

self.assertEqual(square\_root(9), 3.0)

if \_\_name\_\_ == ‘\_\_main\_\_’

unittest.main()

1. File: math\_utils

def factorial(n):

if not isinstance(n, int):

raise TypeError(“Input must be an integer”)

if n < 0:

raise ValueError(“Input must be non-negative”)

if n == 0 or n == 1:

return 1

return n\*factorial(n-1)

File: test\_utils

import unittest

from math\_utils import factorial

class TestFactorial(unittest.TestCase):

def test\_valid\_input(self):

self.assertEqual(factorial(5), 120)

self.assertEqual(factorial(3), 6)

self.assertEqual(factorial(1), 1)

def test\_zero\_inputs(self):

self.assertEqual(factorial(0), 1)

def test\_negative\_input(self):

with self.assertRaises(ValueError):

factorial(-4)

def test\_non\_integer\_input(self):

with self.assertRaises(TypeError):

factorial(3.5)

with self.assertRaises(TypeError):

factorial(“ten”)

if \_\_name\_\_ == ‘\_\_main\_\_’:

unittest.main()

1. import platform

import unittest

import sys

class TestmathFunctions(unittest, TestCase):

@unittest.skip(“test is skipped temporarily for debugging”)

def test\_add(self):

self.assertEqual(1+2,3)

@unittest.skipIf(sys.version\_info < (3,9),”requires python 3.9 or higher”)

def test\_dict\_union\_operator(self):

a = {“x” : 1}

b = {“y” : 2}

self.assertEqual(a|b, {“x”:1,”y”:2})

@unittest.skipUnless(platform.system() == “windows”,”runs on windows”)

def test\_windows\_path(self):

self.assertIn(“c:”,”c:\\programfiles”)

if \_\_name\_\_ == ‘\_\_main\_\_’:

unittest.main()

Pytest

1. @pytest.mark.parametrize  
   def even(num):  
    return num%2 == 0  
     
   def test\_even():  
    even(72)
2. def add(a,b):

return a+b

@pytest.fixture

def sample\_data():

return [(1,2,3),

(0,0,0),

(-1,5,4)]

def test\_addition(sample\_data):

for a,b, expected in sample\_data:

assert add(a,b) == expected

def test\_addition\_not\_none(sample\_data):

for a,b, \_in sample\_data:

assert add(a,b) is not None

1. def double(x):

if not isinatance(x, (int,float)):

raise ValueError(“Input must be a number”)

return x\*2

def test\_double\_with\_string\_input():

with pytest.raises(ValueError, match = “must be a number”):

double(“hello”)

1. def sum(x,y):

return x+y

def test\_sum():

with pytest.mark.skipif(“sys.version\_info < (3,9),”requires python 3.9 or higher”):

sum(5,9)

1. def subtract(a,b):

return a-b

@pytest.mark.xfail

def test\_subtract():

assert subtract(2,-1) == -1

1. def write\_message(file\_path, message):

with open(file\_path, ‘w’) as f:

f.write(message)

def test\_write\_message(temp\_path):

test\_file = tmp\_path / “output.txt”

write\_message(test\_file, “Hello, pytest!”)

with open(test\_file, ‘r’) as f:

content = f.read()

assert content == ‘Hello, pytest!’