EX NO :5a DATE :	Unit Testing in Python

AIM:

To implement the following functions and Write a testing code to test the accuracy of the written function.

PROCEDURE:

- Create a class or function then write a program.
- Import the unit test in the python package.
- Create a class and write the function for the each program.
- Check the accuracy of function using different test cases.
- Then run the program check whether the test case is passed or not.

PROGRAM:

1.fibonacci series

```
print("Fibonacci series- Kamesh(717822i122)")
def fibonacci_series(n):
    a=0
    b=1
    k=0
    if n==0:
        return [0]
    elif n==1:
        return [0,1]
    res=[0,1]
    while k<n-2:
        c=a+b
        res+=[c]
        a=b
        b=c
        k+=1
    return res
print(fibonacci_series(10))
```

Output:

```
Fibonacci series-Kamesh(717821i122)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

2.factorial

```
print("Factorial- Kamesh(717822i122)")
def fact(n):
    if n<0:
        return "factorial not available for negative numbers"
    if n==0 or n==1:
        return 1
    else:
        return n*fact(n-1)
print(fact(-5))
```

Output:

```
Factorial-Kamesh(717821i122)
factorial not available for negative numbers
```

3.anagram

```
print("Anagram- Kamesh(717822i122)")
def anagram(s1,s2):
    if len(s1)!=len(s2):
        return False
    for i in range(len(s1)):
        if s1[i] in s2:
            s2.replace(s1[i],'')
        else:
            return False
    return True
print(anagram('by','bye'))
```

Output:

```
Anagram-Kamesh(717821i122))
False
```

4.palindrome

```
print("Palindrome- Kamesh(717822i122)")
def is_palindrome(s):
    if s==s[::-1]:
        return True
    return False
print(is_palindrome('dad'))
```

Output:

```
Palindrome-Kamesh(717822i122)
True
```

5.circle

```
print("Class Circle- Kamesh(717822i122)")
import math
class Circle:
    def __init__(self, radius):
        self.radius=radius
    def area(self):
        return math.pi*self.radius*self.radius
    def circumference(self):
        return 2*math.pi*self.radius
a=Circle(3)
print(a.area())
print(a.circumference())
```

Output:

```
Class Circle-Kamesh(717821i122)
28.274333882308138
18.84955592153876
```

6.bank account

```
print("Class Bank account- Kamesh(717822i122) ")
class BankAccount:
    def __init__(self,initial_balance):
        self.initial_balance=initial_balance
    def deposit(self,amount):
        self.initial_balance=self.initial_balance+amount
    def withdraw(self,amount):
        self.initial_balance=self.initial_balance-amount
    def get_balance(self):
        if self.initial_balance<0:
            return f"your balance is too low -> {self.initial_balance}..please make deposit of
certain amount to maintain your minimum balance"
        return f"your balance is {self.initial_balance}"
a=BankAccount(220)
a.deposit(20)
a.withdraw(150)
print(a.get_balance())
```

Output:

```
Class Bank account-Kamesh(717821i122)
your balance is too low -> -10..please make deposit of certain amount to maintain your minimum balance
```

7.student

```
print("Class Student- Kamesh(717822i122)")
class Student:
    def __init__(self, name, age):
        self.name=name
        self.age=age
        self.course=[]
    def set_name(self, name):
        self.name=name
    def set_age(self, age):
        self.age=age
    def enroll_course(self, course):
        self.course+=[course]
    def get_name(self):
        return self.name
    def get_age(self):
        return self.age
    def get_courses(self):
        return self.course
a=Student('Kamesh(717822i122)')
a.enroll_course('python')
a.enroll_course('ds')
a.enroll_course('ml')
```

```
print(a.get_name())
print(a.get_age())
print(a.get_courses())
```

Output:

```
Class Student-Kamesh(717821il22)
Kamesh(717821il22)
19
['python', 'ds', 'ml']
```

8.stack

```
print("Implementation of Stack- Kamesh(717822i122)")
class Stack:
    def __init__(self,size):
        self.stack=[None]*size
        self.size=size
        self.top=-1
    def push(self,val):
        if self.is_full():
            print("stack overflow")
            return
        self.top=self.top+1
        self.stack[self.top]=val
    def pop(self):
        if self.is_empty():
            print("stack underflow")
            return
        self.stack[self.top]=None
        self.top-=1
    def is_full(self):
        return self.top+1==self.size
    def is_empty(self):
        return self.top==-1
    def sizee(self):
        return self.top+1
    def peek(self):
        return self.stack[self.top]
a=Stack(5)
a.push(1)
a.push(2)
a.push(3)
a.push(4)
a.push(5)
a.pop()
print(a.sizee())
print(a.peek())
```

Output:

```
Implementation of Stack-Kamesh(717821il22)
4
4
```

9.Queue

```
print("Implementation of Queue- Kamesh(717822i122)")
class Queue:
    def __init__(self,size):
        self.queue=[None]*size
        self.max=size
        self.front=self.rear=-1
        self.size=0
    def enqueue(self,val):
        if self.is_full():
            print("queue is full")
            return
        if self.rear==-1:
            self.front=self.rear=0
            self.queue[self.rear]=val
        else:
            self.rear+=1
            self.queue[self.rear]=val
        self.size+=1
    def dequeue(self):
        if self.is_empty():
            print("queue is empty")
            return
        if self.front+1==self.max:
            a=self.queue[self.front]
            self.front=self.rear=-1
        else:
            a=self.queue[self.front]
            self.front+=1
        self.size-=1
        return a
    def is_full(self):
        return self.rear+1==self.max
    def is_empty(self):
        return self.front==-1
    def sizee(self):
        return self.size
a=Queue(5)
a.enqueue(1)
a.enqueue(2)
a.enqueue(3)
a.dequeue()
print(a.sizee())
```

Output:

```
Implementation of Queue-Kamesh(717821i122)
2
```

10.Employee

```
print("Class Employee- Kamesh(717822i122)")
class Employee:
    def __init__(self,name,id,salary):
        self.name=name
        self.id=id
        self.salary=salary
    def set_name(self,name):
        self.name=name
    def set_id(self,id):
        self.id=id
    def set_salary(self,salary):
        self.salary=salary
    def get_name(self):
        return self.name
    def get_id(self):
        return self.id
    def get_salary(self):
        return self.salary
    def details(self):
        return f"Employee name:{self.name};Employee id:{self.id};Employee's
Salary:{self.salary}"
a=Employee('XXX',101,10000)
print(a.get_name())
a.set_name('Kamesh(717822i122)')
a.set_id(124)
a.set_salary(100000)
print(a.details()
```

Output:

```
Class Employee-Kamesh(717821i122)
XXX
Employee name:Kamesh(717821i122);Employee id:119;Employee's Salary:100000
```

Unit Testing:

```
import unittest
class testdata(unittest.TestCase):
    def test1(self):
        a=Employee("xxx",24,11000)
        self.assertEqual(a.get_id(),24)
    def test2(self):
        a=Queue(5)
        a.enqueue(1)
        a.enqueue(2)
        self.assertEqual(a.sizee(),2)
    def test3(self):
        a=Stack(5)
        self.assertIsNone(a.peek())
    def test4(self):
        a=Student(Kamesh(717822i122))
```

```
        self.assertIs(a.get_name(),' Kamesh(717822i122)')
    def test5(self):
        a=BankAccount(120)
        a.deposit(20)
        self.assertIsInstance(a.get_balance(),str)
    def test6(self):
        a=Student('Kamesh(717822i122)')
        a.enroll_course('python')
        a.enroll_course('ds')
        self.assertIn('ds',a.get_courses())
    def test7(self):
        a=Circle(3)
        self.assertTrue(a.area())
    def test8(self):
        a=Employee("xxx",19,11000)
        self.assertNotEqual(a.get_id(),190)
    def test9(self):
        a=Queue(5)
        a.enqueue(1)
        a.enqueue(2)
        self.assertEqual(a.sizee(),2)
    def test10(self):
        a=Stack(5)
        a.push(2)
        self.assertIsNotNone(a.peek())
    def test11(self):
        a=Student('Gowthaman',19)
        self.assertIsNot(a.get_name(),' Kamesh(717822i122)')
    def test12(self):
        a=BankAccount(120)
        a.deposit(20)
        self.assertNotIsInstance(a.get_balance(),int)
if __name__=='__main__':
    unittest.main(argv=['first-arg-is-ignored'],exit=False)
```

Output:

```
----------------------------------------------------------------------
Ran 12 tests in 0.009s

FAILED (failures=1)
```

#ASSERTION ERROR

```
def test13(self):
    a=Student("Kamesh(717822i122)")
    a.enroll_course('python')
    a.enroll_course('ds')
    self.assertNotIn('ds',a.get_courses())
def test14(self):
    a=Circle(3)
    self.assertFalse(a.area())
```

OUTPUT:

```
..F.........
======================================================================
FAIL: test11 (__main__.testdata.test11)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/Student.MS-70.000/Documents/dev122.py", line 263, in test11
    self.assertIsNot(a.get_name(),'Kamesh(717821il22)')
AssertionError: unexpectedly identical: 'Kamesh(717821il22)'
```

PREPARATION	**30**	
LAB PERFOMANCE	**30**	
REPORT	**40**	
TOTAL	**100**	
INITIAL OF THE FACULTY		

RESULT:

Thus the implementation of unit test code to check the accuracy of above written functions are executed successfully.