Inspiring Excellence

Course Code:	**CSE111**
Course Title:	**Programming Language II**
Classwork No:	**11**
Topic:	**OOP (inheritance)**
Number of tasks:	**4**

Task 1

Write the **Mango** and the **Jackfruit** classes so that the following code generates the output below:

```
class Fruit:
    def __init__(self, formalin=False, name=''):
        self.__formalin = formalin
        self.name = name

    def getName(self):
        return self.name

    def hasFormalin(self):
        return self.__formalin

class testFruit:
    def test(self, f):
        print('----Printing Detail----')
        if f.hasFormalin():
            print('Do not eat the',f.getName(),'.')
            print(f)
        else:
            print('Eat the',f.getName(),'.')
            print(f)

m = Mango()
j = Jackfruit()
t1 = testFruit()
t1.test(m)
t1.test(j)
```

OUTPUT:

```
----Printing Detail-----
Do not eat the Mango.
Mangos are bad for you
----Printing Detail-----
Eat the Jackfruit.
Jackfruits are good for you
```

Task 2

You are given the parent class Point:

```
class Point:
  def __init__(self, x=0, y=0):
    self.x = x
    self.y = y
    self.area = 0

  def calculate_area(self):
    return self.area

  def print_details(self):
    print("--------- Printing details ----------")
    print(f'Co-ordinate: ({self.x},{self.y})')
    print(f'Area: {self.area}')
```

Some information about calculating the area of circle and sphere:

Area of a circle: πr^2

Area of a sphere: $4\pi r^2$

Here, the Inheritance tree will be Point=>Circle=>Sphere

Write **Circle** and **Sphere** classes to generate the following output.

Driver Code	Output
print("--------------1---------------") p1 = Point(2,3) print(f'Area of p1: {p1.calculate_area()}') print("--------------2---------------") p1.print_details() print("--------------3---------------") p2 = Point() p2.print_details() print("--------------4---------------") c1 = Circle(4,0,3) print(f'Area of c1: {c1.calculate_area()}')	--------------1--------------- Area of p1: 0 --------------2--------------- --------- Printing details ---------- Co-ordinate: (2,3) Area: 0 --------------3--------------- --------- Printing details ---------- Co-ordinate: (0,0) Area: 0 --------------4--------------- Area of c1: 50.2656 --------------5--------------- --------- Printing details ---------- Co-ordinate: (0,3)

print("----------5----------") c1.print_details() print("----------6----------") c2 = Circle(7) print(f'Area of c2: {c2.calculate_area()}') print("----------7----------") sph1 = Sphere(3,0,2) print(f'Area of sph1: {sph1.calculate_area()}') print("----------8----------") sph1.print_details() print("----------9----------") sph2 = Sphere(6) print(f'Area of sph2: {sph2.calculate_area()}')	Area: 50.2656 Radius: 4 ----------6---------- Area of c2: 153.9384 ----------7---------- Area of sph1: 113.0976 ----------8---------- --------- Printing details ---------- Co-ordinate: (0,2) Area: 113.0976 Radius: 3 ----------9---------- Area of sph2: 452.3904

Task 3

A bank has two types of accounts : **Savings account** and **Fixed-deposit account**. Some features of these accounts are:

- Savings account:
  - An interest rate can be applied
  - You can deposit money anytime you want.
  - Withdrawal can be made unless its crosses the lower limit of the account
- Fixed deposits account:
  - You can not deposit money anytime you want.
  - Withdrawal can be made after the account is matured.

The parent class Account is given below:

```
class Account:
   def __init__(self, account_number, balance):
     self.account_number = account_number
     self.balance = balance
     self.account_type = "General"
     self.maturity = 0
```

```
def print_details(self):
  print("------ Account details ------")
  print(f"Account Type: {self.account_type}, Maturity: {self.maturity} years")
  print(f"Account Number: {self.account_number}, Balance: ${self.balance:.2f}")

def deposit(self, amount):
  self.balance += amount
  print(f"Deposited ${amount:.2f}. New Balance: ${self.balance:.2f}")

def withdraw(self, amount):
  if self.balance >= amount:
      self.balance -= amount
      print(f"Withdrew ${amount:.2f}. New Balance: ${self.balance:.2f}")
  else:
      print("Insufficient funds.")

def year_passed(self, year):
  self.maturity += year
  print(f"Maturity of the account: {self.maturity} years")
```

Write the classes **SavingsAccount** and **FixedDepositAccount** derived from the **Account** class to generate the following output.

Driver Code	Output
print("----------1----------") account = Account("A203", 2000) account.print_details() print("----------2----------") account.deposit(400) account.withdraw(1500) account.year_passed(2) print("----------3----------") account.print_details() print("----------4----------") savings_account = SavingsAccount("Savings","SA123",	----------1---------- ------ Account details ------ Account Type: General, Maturity: 0 years Account Number: A203, Balance: $2000.00 ----------2---------- Deposited $400.00. New Balance: $2400.00 Withdrew $1500.00. New Balance: $900.00 Maturity of the account: 2 years ----------3---------- ------ Account details ------ Account Type: General, Maturity: 2 years Account Number: A203, Balance: $900.00 ----------4---------- ------ Account details ------ Account Type: Savings, Maturity: 0 years

```
1000, 0.05, 500)
savings_account.print_details()
print("----------5----------")
savings_account.deposit(400)
print("----------6----------")
savings_account.withdraw(1000)
print("----------7----------")
savings_account.withdraw(800)
print("----------8----------")
savings_account.apply_interest()
print("----------9----------")
savings_account.print_details()
print("----------10----------")
fixed_account1= FixedDepositAccount("Fixed
Deposit","FDA321", 10000, 5)
fixed_account1.print_details()
print("----------11----------")
fixed_account1.deposit(400)
print("----------12----------")
fixed_account1.year_passed(6)
print("----------13----------")
fixed_account1.withdraw(10000)
print("----------14----------")
fixed_account1.print_details()
print("----------15----------")
fixed_account2 = FixedDepositAccount("Fixed
Deposit","FDA300", 50000, 7)
fixed_account2.print_details()
print("----------16----------")
fixed_account2.withdraw(10000)
```

```
Account Number: SA123, Balance: $1000.00
Interest Rate: 0.05, Minimum Limit: $500
----------5----------
Deposited $400.00. New Balance: $1400.00
----------6----------
Insufficient funds.
----------7----------
Withdrew $800.00. New Balance: $600.00
----------8----------
Interest applied. New Balance: $630.00
----------9----------
------ Account details ------
Account Type: Savings, Maturity: 0 years
Account Number: SA123, Balance: $630.00
Interest Rate: 0.05, Minimum Limit: $500
----------10----------
------ Account details ------
Account Type: Fixed Deposit, Maturity: 0 years
Account Number: FDA321, Balance: $10000.00
----------11----------
You can not deposit in a fixed deposit account.
----------12----------
Maturity of the account: 6 years
----------13----------
Withdrew $10000.00. New Balance: $0.00
----------14----------
------ Account details ------
Account Type: Fixed Deposit, Maturity: 6 years
Account Number: FDA321, Balance: $0.00
----------15----------
------ Account details ------
Account Type: Fixed Deposit, Maturity: 0 years
Account Number: FDA300, Balance: $50000.00
----------16----------
Can not withdraw, Account is not matured
```

Task 4

```
1  class A:
2      temp = 4
3      def __init__(self):
4          self.sum = 0
5          self.y = 0
6          self.y = A.temp - 2
7          self.sum = A.temp + 1
8          A.temp -= 2
9      def methodA(self, m,  n):
10         x = 0
11         self.y = self.y + m + (A.temp)
12         A.temp += 1
13         x = x + 1 + n
14         self.sum = self.sum + x + self.y
15         print(x, self.y, self.sum)
16
17 class B(A):
18     x = 0
19     def __init__(self,b=None):
20         super().__init__()
21         self.sum = 0
22         if b==None:
```

```
23                 self.y = A.temp + 3
24                 self.sum = 3 + A.temp + 2
25                 A.temp -= 2
26             else:
27                 self.sum = b.sum
28                 B.x = b.x
29                 b.methodB(2, 3)
30         def methodB(self, m,  n):
31             y = 0
32             y = y + self.y
33             B.x = self.y + 2 + A.temp
34             self.methodA(B.x, y)
35             self.sum = B.x + y + self.sum
36             print(B.x, y, self.sum)
```

Write the output of the following code:

```
a1 = A()
b1 = B()
b2 = B(b1)
b1.methodA(1, 2)
b2.methodB(3, 2)
```

Output:

x	y	sum