

| **Course Code:** | **CSE111** |
| --- | --- |
| **Course Title:** | **Programming Language II** |
| **Lab No:** | **07** |
| **Topic:** | **OOP (Inheritance)** |
| **Number of tasks:** | **8** |

**Task 1**

The tea company **Kazi and Kazi (KK)** has decided to produce a new line of flavored teas. Design the **KK\_tea (parent) and KK\_flavoured\_tea (child)** classes so that the following output is produced. The KK\_flavoured\_tea class should inherit KK\_tea. Note that:

* An object of either class represents a **single box of teabags**.
* Each tea bag **weighs 2 grams**.
* The **status** of an object refers to whether it is sold or not

*Hint: you should use class methods/variables*

| **t1 = KK\_tea(250)**  **print("-----------------1-----------------")**  **t1.product\_detail()**  **print("-----------------2-----------------")**  **KK\_tea.total\_sales()**  **print("-----------------3-----------------")**  **t2 = KK\_tea(470, 100)**  **t3 = KK\_tea(360, 75)**  **KK\_tea.update\_sold\_status\_regular(t1, t2, t3)**  **print("-----------------4-----------------")**  **t3.product\_detail()**  **print("-----------------5-----------------")**  **KK\_tea.total\_sales()**  **print("-----------------6-----------------")**  **t4 = KK\_flavoured\_tea("Jasmine", 260, 50)**  **t5 = KK\_flavoured\_tea("Honey Lemon", 270, 45)**  **t6 = KK\_flavoured\_tea("Honey Lemon", 270, 45)**  **print("-----------------7-----------------")**  **t4.product\_detail()**  **print("-----------------8-----------------")**  **t6.product\_detail()**  **print("-----------------9-----------------")**  **KK\_flavoured\_tea.update\_sold\_status\_flavoured(t4, t5, t6)**  **print("-----------------10-----------------")**  **KK\_tea.total\_sales()** | **-----------------1-----------------**  **Name: KK Regular Tea, Weight: 100**  **Tea Bags: 50, Price: 250**  **Status: False**  **-----------------2-----------------**  **Total sales: {'KK Regular Tea': 0}**  **-----------------3-----------------**  **-----------------4-----------------**  **Name: KK Regular Tea, Weight: 150**  **Tea Bags: 75, Price: 360**  **Status: True**  **-----------------5-----------------**  **Total sales: {'KK Regular Tea': 3}**  **-----------------6-----------------**  **-----------------7-----------------**  **Name: KK Jasmine Tea, Weight: 100**  **Tea Bags: 50, Price: 260**  **Status: False**  **-----------------8-----------------**  **Name: KK Honey Lemon Tea, Weight: 90**  **Tea Bags: 45, Price: 270**  **Status: False**  **-----------------9-----------------**  **-----------------10-----------------**  **Total sales: {'KK Regular Tea': 3, 'KK Jasmine Tea': 1, 'KK Honey Lemon Tea': 2}** |
| --- | --- |

**Task 2**

Given the following classes, write the code for the **Cricket\_Tournament** and the **Tennis\_Tournment** class so that the following output is printed.

| class Tournament:      def \_\_init\_\_(self,name='Default'):          self.\_\_name = name      def set\_name(self,name):          self.\_\_name = name      def get\_name(self):          return self.\_\_name  #write your code here  ct1 = Cricket\_Tournament()  print(ct1.detail())  print("-----------------------")  ct2 = Cricket\_Tournament("IPL",10,"t20")  print(ct2.detail())  print("-----------------------")  tt = Tennis\_Tournament("Roland Garros",128)  print(tt.detail()) | *OUTPUT:*  Cricket Tournament Name: Default Number of Teams: 0  Type: No type  -----------------------  Cricket Tournament Name: IPL  Number of Teams: 10  Type: t20  -----------------------  Tennis Tournament Name: Roland Garros  Number of Players: 128 |
| --- | --- |

**Task 3**

Given the following classes, write the code for the **Triangle** and the **Trapezoid** class so that the following output is printed.

| class Shape:    def \_\_init\_\_(self, name='Default', height=0, base=0):      self.area = 0      self.name = name      self.height = height      self.base = base    def get\_height\_base(self):      return "Height: "+str(self.height)+",Base: "+str(self.base)  #write your code here  tri\_default = triangle()  tri\_default.calcArea()  print(tri\_default.printDetail())  print('--------------------------')  tri = triangle('Triangle', 10, 5)  tri.calcArea()  print(tri.printDetail())  print('---------------------------')  trap = trapezoid('Trapezoid', 10, 6, 4)  trap.calcArea()  print(trap.printDetail()) | *OUTPUT:*  Shape name: Default  Height: 0, Base: 0  Area: 0.0  ---------------------------  Shape name: Triangle  Height: 10, Base: 5  Area: 25.0  ---------------------------  Shape name: Trapezoid  Height: 10, Base: 6, Side\_A: 4  Area: 50.0 |
| --- | --- |

**Task 4**

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 5**

A multinational company has two special types of regular employees. One is Foreign employees and another one is Part time employees. Design the Employee (parent), Foreign\_employee(child) and Parttime\_employee(child) classes so that the following output is produced. The Foreign\_employee and Parttime\_employee classes should inherit the Employee class. Note that:

* Basic salary of a Regular, Foreign employee is 30,000 and for Part-time employees basic is 15,000.
* Regular employees get 10% increment on their salary and Foreign employees get 15% increment on their basic salary.
* Employees from the HR department will collect their work distribution load from the manager, and others will collect their work distribution load from the HR department.

| Driver Code | Output |
| --- | --- |
| print("1------------------------------------------")  emp1=Employee("Nawaz Ali", 102, "Marketing")  print("2------------------------------------------")  emp1.employeeDetails()  print("3------------------------------------------")  emp1.workDistribution("Marketing")  print("4------------------------------------------")  emp1.increment()  emp1.employeeDetails()  print("5------------------------------------------")  f\_emp=Foreign\_employee("Nadvi", 311, "Human Resource")  f\_emp.employeeDetails()  print("6------------------------------------------")  f\_emp.workDistribution("Human Resource")  print("7------------------------------------------")  f\_emp.increment()  f\_emp.employeeDetails()  print("8------------------------------------------")  p1\_emp=Part\_time\_employee("Asif", 210, "Sales")  p2\_emp=Part\_time\_employee("Olive", 223, "Accounts")  print("9------------------------------------------")  p1\_emp.employeeDetails()  print("10------------------------------------------")  p1\_emp.workDistribution("Sales")  print("11------------------------------------------")  p2\_emp.increment()  print("12------------------------------------------")  p2\_emp.employeeDetails() | 1------------------------------------------  2------------------------------------------  Name: Nawaz Ali, Dept Marketing  Employee id: 102, Salary: 30000  3------------------------------------------  Collect work distribution loads from the HR department.  4------------------------------------------  Name: Nawaz Ali, Dept Marketing  Employee id: 102, Salary: 33000.0  5------------------------------------------  Name: Nadvi, Dept Human Resource  Employee id: 311, Salary: 30000  Employee Type: Foreign  6------------------------------------------  Collect work distribution details from the manager.  7------------------------------------------  Name: Nadvi, Dept Human Resource  Employee id: 311, Salary: 34500.0  Employee Type: Foreign  8------------------------------------------  9------------------------------------------  Name: Asif, Dept Sales  Employee id: 210, Salary: 15000  Employee Type: Part Time  10------------------------------------------  Collect work distribution loads from the HR department.  11------------------------------------------  Sadly, there is no increment for the part time employees!!  12------------------------------------------  Name: Olive, Dept Accounts  Employee id: 223, Salary: 15000  Employee Type: Part Time |

**Task 6**

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: πr2

Area of a sphere: 4πr2

Here, 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()}')  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()}') | --------------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)  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 7**

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", 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) | -----------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  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 8**

| **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** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |