****

**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

PULCHOWK CAMPUS

A REPORT ON

Programming in Python Language

SUBMITTED BY:

SUSHANT THAKUR (081BEL092)

SUBMITTED TO:

PRAKASH CHANDRA PRASAD SIR,

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

**QUESTION 1:**

Define a class Student with attributes name, roll\_number, and marks. Implement a method display\_info() that prints the details of the student. Create an instance of Student and call the display\_info() method to display the student's details..

**CODE 1:**

class Student:

def \_\_init\_\_(self, name, roll\_number, marks):

self.name = name

self.roll\_number = roll\_number

self.marks = marks

def display\_info(self):

print(f"Name: {self.name}")

print(f"Roll Number: {self.roll\_number}")

print(f"Marks: {self.marks}")

s1 = Student("Rahul", 101, 88)

s1.display\_info()

**OUTPUT 1:**



**OUESTION 2:**

**Create a base class Animal with a method speak() that prints "Animal makes a sound". Derive a class Dog from Animal and override the speak() method to print "Dog barks". Instantiate the Dog class and call its speak() method.**

**CODE 2:**

class Animal:

def speak(self):

print("Animal makes a sound")

class Dog(Animal):

def speak(self):

print("Dog barks")

# Example

pet = Dog()

pet.speak()

**OUTPUT 2:**



**QUESTION 3:**

**Define a class BankAccount with private attributes account\_number and balance. Implement methods to deposit and withdraw money, ensuring that the balance cannot go below zero. Provide a method to get the account details. Test the class by performing deposit and withdrawal operations.**

**CODE 3**:

class BankAccount:

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

self.\_\_account\_number = account\_number

self.\_\_balance = balance

def deposit(self, amount):

self.\_\_balance += amount

print(f"Deposited ₹{amount}. New Balance = ₹{self.\_\_balance}")

def withdraw(self, amount):

if amount <= self.\_\_balance:

self.\_\_balance -= amount

print(f"Withdrew ₹{amount}. New Balance = ₹{self.\_\_balance}")

else:

print("Insufficient Balance!")

def get\_details(self):

print(f"Account Number: {self.\_\_account\_number}, Balance: ₹{self.\_\_balance}")

# Example

acc1 = BankAccount(123456789, 5000)

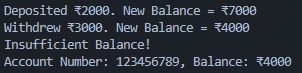
acc1.deposit(2000)

acc1.withdraw(3000)

acc1.withdraw(6000)

acc1.get\_details()

**OUTPUT 3:**

****

**QUESTION 4:**

**Create a base class Shape with a method area(). Derive two classes Rectangle and Circle from Shape. Implement the area() method in both derived classes. Instantiate Rectangle and Circle, and demonstrate polymorphism by calling their area() methods.**

**CODE 4:**

import math

class Shape:

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, length, breadth):

self.length = length

self.breadth = breadth

def area(self):

return self.length \* self.breadth

class Circle(Shape):

def \_\_init\_\_(self, radius):

self.radius = radius

def area(self):

return math.pi \* self.radius \* self.radius

# Example

shapes = [Rectangle(10, 5), Circle(7)]

for s in shapes:

print("Area:", s.area())

**OUTPUT 4:**

****

**QUESTION 5:**

**Define a class Person with attributes name and age. Derive a class Employee from Person with additional attributes employee\_id and salary. Implement a method display\_employee() in Employee that prints all the details. Create an instance of Employee and display the information.**

**CODE 5:**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

class Employee(Person):

def \_\_init\_\_(self, name, age, employee\_id, salary):

super().\_\_init\_\_(name, age)

self.employee\_id = employee\_id

self.salary = salary

def display\_employee(self):

print(f"Name: {self.name}, Age: {self.age}, ID: {self.employee\_id}, Salary: ₹{self.salary}")

# Example

emp1 = Employee("abc", 28, "E101", 50000)

emp1.display\_employee()

**OUTPUT 5:**

****

**QUESTION 6:**

**Define a class Vector with attributes x and y. Overload the + operator to add two Vector objects. Implement the \_\_add\_\_() method and test it by adding two Vector instances.**

**CODE 6:**

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, other):

return Vector(self.x + other.x, self.y + other.y)

def display(self):

print(f"({self.x}, {self.y})")

# Example

v1 = Vector(2, 5)

v2 = Vector(4, 7)

v3 = v1 + v2

v3.display()

**OUTPUT 6:**

****

**QUESTION 7:**

**Create a class Book with attributes title and author. Overload the \_\_str\_\_() method to return a string representation of the Book object in the format "Title by Author". Test this method by printing a Book instance.**

**CODE 7:**

class Book:

def \_\_init\_\_(self, title, author):

self.title = title

self.author = author

def \_\_str\_\_(self):

return f"'{self.title}' by {self.author}"

# Example

b1 = Book("The White Tiger", "Aravind Adiga")

print(b1)

**OUTPUT 7:**

****

**QUESTION 8:**

**Define a class Time with attributes hours, minutes, and seconds. Overload the == operator to compare two Time objects for equality. Implement the \_\_eq\_\_() method and test it by comparing two Time instances.**

**CODE 8:**

class Time:

def \_\_init\_\_(self, hours, minutes, seconds):

self.hours = hours

self.minutes = minutes

self.seconds = seconds

def \_\_eq\_\_(self, other):

return (self.hours == other.hours and

self.minutes == other.minutes and

self.seconds == other.seconds)

# Example

t1 = Time(10, 30, 15)

t2 = Time(10, 30, 15)

t3 = Time(9, 45, 20)

print(t1 == t2) # True

print(t1 == t3) # False

**OUTPUT 8:**

****

**QUESTION 9:**

**Define a class Person with attributes name and age. Define another class**

**Address with attributes street, city, and zipcode. Create a Contact class that**

**contains an instance of Person and Address. Implement methods to display**

**the contact details. Create a Contact object and display its information.**

**CODE 9:**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

class Address:

def \_\_init\_\_(self, street, city, zipcode):

self.street = street

self.city = city

self.zipcode = zipcode

class Contact:

def \_\_init\_\_(self, person, address):

self.person = person

self.address = address

def display\_contact(self):

print(f"Name: {self.person.name}, Age: {self.person.age}")

print(f"Address: {self.address.street}, {self.address.city} - {self.address.zipcode}")

# Example

p1 = Person("Amit", 22)

a1 = Address("MG", "USA", "560001")

c1 = Contact(p1, a1)

c1.display\_contact()

**OUTPUT 9:**

****

==================================================================

**GitHub:** https://github.com/SushanThakur/2nd-sem-assignment/tree/master/lab-6/