**PROGRAM-12 OOP’S**

1. Write a python program to perform polymorphism

class Animal:

def sound(self):

pass

class Cat(Animal):

def sound(self):

print("Meow")

class Dog(Animal):

def sound(self):

print("Woof")

def make\_sound(animal):

animal.sound()

cat = Cat()

dog = Dog()

make\_sound(cat)

make\_sound(dog)

**sample output:**

Meow

Woof

1. Write a python program to perform inheritance?

class Shape:

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

self.color = color

def area(self):

pass

class Circle(Shape):

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

super().\_\_init\_\_(color)

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

class Rectangle(Shape):

def \_\_init\_\_(self, color, width, height):

super().\_\_init\_\_(color)

self.width = width

self.height = height

def area(self):

return self.width \* self.height

circle = Circle("Red", 5)

rectangle = Rectangle("Blue", 4, 6)

print("Circle area:", circle.area())

print("Rectangle area:", rectangle.area())

**sample output:**

Circle area: 78.5

Rectangle area: 24

1. Write a python program to perform attributes and methods?

class MyClass:

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

self.public\_attribute = "I'm a public attribute"

self.\_protected\_attribute = "I'm a protected attribute"

self.\_\_private\_attribute = "I'm a private attribute"

def public\_method(self):

print("I'm a public method")

def \_protected\_method(self):

print("I'm a protected method")

def \_\_private\_method(self):

print("I'm a private method")

obj = MyClass()

print(obj.public\_attribute)

obj.public\_method()

print(obj.\_protected\_attribute)

obj.\_protected\_method()

print(obj.\_MyClass\_\_private\_attribute)

obj.\_MyClass\_\_private\_method()

**sample output:**

I'm a public attribute

I'm a public method

I'm a protected attribute

I'm a protected method

I'm a private attribute

I'm a private method

1. Write a python program to perform encapsulations?

class BankAccount:

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

self.\_account\_number = account\_number

self.\_balance = balance

def get\_account\_number(self):

return self.\_account\_number

def get\_balance(self):

return self.\_balance

def deposit(self, amount):

if amount > 0:

self.\_balance += amount

print("Deposit successful.")

else:

print("Invalid amount for deposit.")

def withdraw(self, amount):

if amount > 0 and amount <= self.\_balance:

self.\_balance -= amount

print("Withdrawal successful.")

else:

print("Insufficient funds or invalid amount for withdrawal.")

account = BankAccount("1234567890", 1000)

print("Account Number:", account.get\_account\_number())

print("Balance:", account.get\_balance())

account.deposit(500)

account.withdraw(200)

account.\_account\_number = "9876543210"

account.\_balance = 5000

print("Account Number:", account.get\_account\_number())

print("Balance:", account.get\_balance())

**sample output:**

Account Number: 1234567890

Balance: 1000

Deposit successful.

Withdrawal successful.

Account Number: 9876543210

Balance: 5000