Assignment-7

1. Create a polymorphic function calculate_area() that calculates the area of a Circle, Rectangle, and Triangle based on the shape object passed to it.

```
import math
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * (self.radius ** 2)

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

def calculate_area(shape):
    return shape.area()

circle = Circle(5)
rectangle = Rectangle(4, 6)
triangle = Triangle(4, 3)

print(f"Area of Circle: {calculate_area(circle)}")
print(f"Area of Rectangle: {calculate_area(rectangle)}")
print(f"Area of Triangle: {calculate_area(triangle)}")
```

2. Create a class Animal with an __init__ method that initializes the name of the animal. Create a child class Dog that calls the parent's __init__ method using super() and adds an additional attribute, breed.

```
class Animal:
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def display_info(self):
        print(f"Name: {self.name}, Breed: {self.breed}")
```

```
dog = Dog("Buddy", "Golden Retriever")
dog.display_info()
```

3. Write a class Calculator with a method add() that can handle two or three arguments. Use default arguments to achieve method overloading behavior.

```
class Calculator:
    def add(self, a, b, c=0):
        return a + b + c

calc = Calculator()

result1 = calc.add(5, 10)
print(f"Sum of 5 and 10: {result1}")

result2 = calc.add(5, 10, 15)
print(f"Sum of 5, 10, and 15: {result2}")
```

4. Create two parent classes, Person and Employee. The Person class should have attributes name and age, and the Employee class should have an attribute salary. Create a child class Manager that inherits from both and includes an additional attribute department.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Employee:
    def __init__(self, salary):
        self.salary = salary

class Manager(Person, Employee):
    def __init__(self, name, age, salary, department):
        Person.__init__(self, name, age)
        Employee.__init__(self, salary)
        self.department = department

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}, Salary: {self.sal-
ary}, Department: {self.department}")

manager = Manager("John Doe", 40, 80000, "HR")
manager.display_info()
```

5. Create a parent class BankAccount with a method deposit(). Create two child classes, SavingsAccount and CurrentAccount, that override the deposit() method to include specific rules for each account type.

```
class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance

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

```
class SavingsAccount(BankAccount):
    def deposit(self, amount):
        if amount < 100:
            print("Deposit amount must be at least 100 for Savings Ac-
count.")
        else:
            super().deposit(amount)

class CurrentAccount(BankAccount):
    def deposit(self, amount):
        fee = 5
        amount_after_fee = amount - fee
        if amount_after_fee < 0:
            print("Deposit amount is too small after fee. Please de-
posit more.")
        else:
            self.balance += amount_after_fee
            print(f"Deposited {amount} (after fee of {fee}). New bal-
ance: {self.balance}")

savings_account = SavingsAccount(1000)
savings_account.deposit(50)
savings_account.deposit(200)

current_account = CurrentAccount(500)
current_account.deposit(50)
current_account.deposit(100)
```

6. Create a class Vehicle with attributes name and max_speed. Create a child class Car that adds an attribute num_doors. Write a script to create a Car object and display all its attributes.

```
class Vehicle:
    def __init__(self, name, max_speed):
        self.name = name
        self.max_speed = max_speed

    def show(self):
        print(f"Name: {self.name}, Max Speed: {self.max_speed} km/h")

class Car(Vehicle):
    def __init__(self, name, max_speed, num_doors):
        super().__init__(name, max_speed)
        self.num_doors = num_doors

    def show(self):
        super().show()
        print(f"Number of Doors: {self.num_doors}")

car1 = Car("Toyota", 180, 4)
car2 = Car("Tata", 160, 4)

car1.show()
car2.show()
```

7. Use the abc module to create an abstract class Shape with a method area(). Implement this class in child classes Square and Circle. Write a script to compute and display the areas of a square and a circle using the same method.

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

square = Square(4)
circle = Circle(3)

print(f"Area of the square: {square.area()}")
print(f"Area of the circle: {circle.area()}")
```

8. Write a function describe_pet() that accepts an object and calls the speak() method on it. Create two classes, Parrot and Fish, where only the Parrot class has a speak() method. Pass objects of both classes to describe_pet() and observe the behavior.

```
class Parrot:
    def speak(self):
        print("Parrot says: Squawk!")

class Fish:
    pass

def describe_pet(pet):
    try:
        pet.speak()
    except AttributeError:
        print(f"{pet.__class__.__name__} cannot speak.")

parrot = Parrot()
fish = Fish()

print("Describing a Parrot:")
describe_pet(parrot)

print("\nDescribing a Fish:")
describe_pet(fish)
```

9. Create two classes, EvenNumbers and OddNumbers, with a method generate_numbers(n) that generates the first n even or odd numbers. Write a function that takes an object of either class and prints the numbers.

```
class EvenNumbers:
    def generate_numbers(self, n):
        i = 0
        count = 0
        while count < n:
            if i % 2 == 0:
                print(i, end=" ")
                count += 1
            i += 1
        print()

class OddNumbers:
    def generate_numbers(self, n):
        i = 1
        count = 0
        while count < n:
            if i % 2 != 0:
                print(i, end=" ")
                count += 1
            i += 1
        print()

def print_numbers(number_generator, n):
    number_generator.generate_numbers(n)

even_generator = EvenNumbers()
odd_generator = OddNumbers()

print("First 5 even numbers:")
print_numbers(even_generator, 5)

print("First 5 odd numbers:")
print_numbers(odd_generator, 5)
```

10. Create a class Grandparent with a method say_hello(). Inherit this class into Parent, and then into Child. Demonstrate how the Child class can access methods from Grandparent and Parent.

```
class Grandparent:
    def say_hello(self):
        print("Hello from Grandparent")
class Parent(Grandparent):
    def say_hello_from_parent(self):
        print("Hello from Parent")
class Child(Parent):
    def say_hello_from_child(self):
        print("Hello from child")

ob1 = Child()
ob1.say_hello()
ob1.say_hello_from_parent()
ob1.say_hello_from_child()
```