Continued..

10) Calculate area of a ractangle using object as an argument to a method.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

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

rect1 = Rectangle(10, 5)
print("Area:", calculate_area(rect1))
```

```
Area: 50
```

11) Calculate the area of a square.

Include a Constructor, a method to calculate area named area() and a method named output() that prints the output and is invoked by area().

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

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

    def output(self, area):
        print(f"Area of Square: {area}")

square1 = Square(4)
square1.area()
```

```
Area of Square: 16
```

12) Calculate the area of a rectangle.

Include a Constructor, a method to calculate area named area() and a method named output() that prints the output and is invoked by area().

Also define a class method that compares the two sides of reactangle. An object is instantiated only if the two sides are different; otherwise a message should be displayed : THIS IS SQUARE.

```python
class Rectangle:
    def __init__(self, length, width):
        if length == width:
            print("THIS IS SQUARE.")
        else:
            self.length = length
            self.width = width

    def area(self):
        return self.output(self.length * self.width)

    def output(self, area):
        print(f"Area of Rectangle: {area}")

    @classmethod
    def compare_sides(cls, length, width):
        if length == width:
            print("THIS IS SQUARE.")
        else:
            return cls(length, width)

rect1 = Rectangle.compare_sides(10, 5)
if rect1:
    rect1.area()

rect2 = Rectangle.compare_sides(6, 6)
```

```
Area of Rectangle: 50
THIS IS SQUARE.
```

13) Define a class Square having a private attribute "side".

Implement get_side and set_side methods to accees the private attribute from outside of the class.

```python
class Square:
    def __init__(self, side):
        self.__side = side
    def get_side(self):
        return self.__side

    def set_side(self, side):
        self.__side = side

square1 = Square(4)
print("Side:", square1.get_side())
square1.set_side(6)
print("Updated Side:", square1.get_side())
```

```
Side: 4
Updated Side: 6
```

14) Create a class Profit that has a method named getProfit that accepts profit from the user.

Create a class Loss that has a method named getLoss that accepts loss from the user.

Create a class BalanceSheet that inherits from both classes Profit and Loss and calculates the balanace. It has two methods getBalance() and printBalance().

```
class Profit:
    def getProfit(self):
        self.profit = float(input("Enter Profit: "))

class Loss:
    def getLoss(self):
        self.loss = float(input("Enter Loss: "))

class BalanceSheet(Profit, Loss):
    def getBalance(self):
        self.balance = self.profit - self.loss

    def printBalance(self):
        print(f"Balance: {self.balance}")

bs = BalanceSheet()
bs.getProfit()
bs.getLoss()
bs.getBalance()
bs.printBalance()
```

```
Enter Profit:  2000
Enter Loss:  1000
```

```
Balance: 1000.0
```

15) WAP to demonstrate all types of inheritance.

```python
# Single Inheritance
class Animal:
    def sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def bark(self):
        print("Dog barks")

dog = Dog()
dog.sound()
dog.bark()

# Multiple Inheritance
class Father:
    def func1(self):
        print("Function from Father")

class Mother:
    def func2(self):
        print("Function from Mother")

class Child(Father, Mother):
    def func3(self):
        print("Function from Child")

child = Child()
child.func1()
child.func2()
child.func3()

# Multilevel Inheritance
class Grandfather:
    def grandparent_func(self):
        print("Function from Grandfather")

class Father(Grandfather):
    def parent_func(self):
        print("Function from Father")

class Son(Father):
    def child_func(self):
        print("Function from Son")

son = Son()
son.grandparent_func()
son.parent_func()
son.child_func()

# Hierarchical Inheritance
class Parent:
    def parent_func(self):
        print("Function from Parent")

class Son(Parent):
    def son_func(self):
        print("Function from Son")

class Daughter(Parent):
    def daughter_func(self):
        print("Function from Daughter")

son = Son()
daughter = Daughter()
son.parent_func()
son.son_func()
daughter.parent_func()
daughter.daughter_func()

# Hybrid Inheritance
class Person:
    def person_func(self):
        print("Function from Person")

class Employee(Person):
    def employee_func(self):
        print("Function from Employee")

class Manager(Person):
    def manager_func(self):
        print("Function from Manager")

class CEO(Employee, Manager):
    def ceo_func(self):
        print("Function from CEO")

ceo = CEO()
ceo.person_func()
ceo.employee_func()
ceo.manager_func()
ceo.ceo_func()
```

```
Animal makes a sound
Dog barks
Function from Father
Function from Mother
Function from Child
Function from Grandfather
Function from Father
Function from Son
Function from Parent
Function from Son
Function from Parent
Function from Daughter
Function from Person
Function from Employee
Function from Manager
Function from CEO
```

16) Create a Person class with a constructor that takes two arguments name and age.

Create a child class Employee that inherits from Person and adds a new attribute salary.

Override the **init** method in Employee to call the parent class's **init** method using the super() and then initialize the salary attribute.

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

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

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}, Salary: 
{self.salary}")

emp = Employee("John", 30, 50000)
emp.display()

Name: John, Age: 30, Salary: 50000
```

17) Create a Shape class with a draw method that is not implemented.

Create three child classes Rectangle, Circle, and Triangle that implement the draw method with their respective drawing behaviors.

Create a list of Shape objects that includes one instance of each child class, and then iterate through the list and call the draw method on each object.

```python
from abc import ABC, abstractmethod

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

class Rectangle(Shape):
    def draw(self):
        print("Drawing a Rectangle")

class Circle(Shape):
    def draw(self):
        print("Drawing a Circle")

class Triangle(Shape):
    def draw(self):
        print("Drawing a Triangle")

shapes = [Rectangle(), Circle(), Triangle()]

for shape in shapes:
    shape.draw()
```

```
Drawing a Rectangle
Drawing a Circle
Drawing a Triangle
```