Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Python Programming - 2301CS404

Lab - 13

223 | Vishal Baraiya | 23010101014

Continued..

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

In [4]:
```
class Rectangle:
    l = 0
    b = 0

    def __init__(self,l,b):
        self.l = l
        self.b = b


def findArea(r : Rectangle):
    return r.l * r.b

r = Rectangle(10,20)
print(f"Area = {findArea(r)}")
```

```
Area = 200
```

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().

In [5]:
```
class square:
    l = 0
```

```
    a = 0

    def __init__(self,l):
        self.l = l

    def area(self):
        self.a = self.l * self.l
        self.output()

    def output(self):
        print(f"Area : {self.a}")

s = square(10)
s.area()
```

```
Area : 100
```

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.

In [14]:
```
class Rectangle:
    length = 0
    breath = 0

    def __init__(self,length,breath):
        self.length = length
        self.breath = breath

    @classmethod
    def compare(cls,l,b):
        if l==b:
            print("THIS IS SQAURE.")
            return None
        else:
            return Rectangle(l,b)

    def area(self):
        self.area = self.length * self.breath
        self.output(self.area)

    def output(self,ans):
        print(f"Area : {ans}")

r1=Rectangle.compare(9,8)
if r1:
    r1.area()
```

```
Area : 72
```

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.

In [15]:
```
class squre:
    _side = 0

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

    def get_side(self):
        return self._side

s = squre()
s.set_side(12)
print(s.get_side())
```

```
12
```

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().

In [4]:
```
class Profit:

    def getProfit(self):
        self.profit = int(input("Enter Your Profit : "))

class Loss:

    def getLoss(self):
        self.loss = int(input("Enter Your Loss : "))

class BalanceSheet(Profit,Loss):

    def getBalance(self):
        self.balance = self.profit - self.loss

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

a = BalanceSheet()

a.getProfit()
a.getLoss()
a.getBalance()
a.printBalance()
```

```
Balance = 500
```

15) WAP to demonstrate all types of inheritance.

In [3]:
```python
# 1. Single Inheritance
class Animal:
    def sound(self):
        print("Animal makes sound")

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

# 2. Multiple Inheritance
class Father:
    def father_name(self):
        print("Father's name is John")

class Mother:
    def mother_name(self):
        print("Mother's name is Mary")

class Child(Father, Mother):
    def child_name(self):
        print("Child's name is Sam")

# 3. Multilevel Inheritance
class Grandparent:
    def grandparent_name(self):
        print("Grandparent's name is George")

class Parent(Grandparent):
    def parent_name(self):
        print("Parent's name is Henry")

class ChildOfParent(Parent):
    def child_name(self):
        print("Child's name is Lisa")

# 4. Hierarchical Inheritance
class Vehicle:
    def type_of_vehicle(self):
        print("This is a vehicle")

class Car(Vehicle):
    def car_type(self):
        print("This is a car")

class Bike(Vehicle):
    def bike_type(self):
        print("This is a bike")

# 5. Hybrid Inheritance (Combination of Multiple and Multilevel)
class A:
    def method_a(self):
        print("Method from class A")

class B(A):
    def method_b(self):
```

```
        print("Method from class B")

class C:
    def method_c(self):
        print("Method from class C")

class D(B, C):
    def method_d(self):
        print("Method from class D")


# Demonstrating Single Inheritance
print("Single Inheritance:")
dog = Dog()
dog.sound()  # Inherited method
dog.bark()   # Class method

# Demonstrating Multiple Inheritance
print("\nMultiple Inheritance:")
child = Child()
child.father_name()  # Method from Father class
child.mother_name()  # Method from Mother class
child.child_name()   # Method from Child class

# Demonstrating Multilevel Inheritance
print("\nMultilevel Inheritance:")
child_of_parent = ChildOfParent()
child_of_parent.grandparent_name()  # Inherited method from Grandparent
child_of_parent.parent_name()       # Inherited method from Parent
child_of_parent.child_name()        # Method from ChildOfParent

# Demonstrating Hierarchical Inheritance
print("\nHierarchical Inheritance:")
car = Car()
car.type_of_vehicle()  # Inherited from Vehicle
car.car_type()         # Method from Car class

bike = Bike()
bike.type_of_vehicle()  # Inherited from Vehicle
bike.bike_type()        # Method from Bike class

# Demonstrating Hybrid Inheritance
print("\nHybrid Inheritance:")
d = D()
d.method_a()  # Inherited from A
d.method_b()  # Inherited from B
d.method_c()  # Inherited from C
d.method_d()  # Method from D class
```

```
Single Inheritance:
Animal makes sound
Dog barks

Multiple Inheritance:
Father's name is John
Mother's name is Mary
Child's name is Sam

Multilevel Inheritance:
Grandparent's name is George
Parent's name is Henry
Child's name is Lisa

Hierarchical Inheritance:
This is a vehicle
This is a car
This is a vehicle
This is a bike

Hybrid Inheritance:
Method from class A
Method from class B
Method from class C
Method from class D
```

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.

In [19]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

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

    def display(self):
        super().display()
        print(f"Salary: {self.salary}")

emp = Employee("Karan", 30, 60000)
emp.display()
```

```
Name: Karan, Age: 30
Salary: 60000
```

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.

In [20]:
```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
```

In [ ]:
```python

```