Darshan UNIVERSITY

योग: कर्मसु कौशलम्

Python Programming - 2301CS404

Lab - 13

Charmi Bhalodiya

23010101020

4B 448 8th batch

Continued..

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

In [1]:
```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

    def calculate_area(self, other):
        return other.area()

rect1 = Rectangle(10, 5)
rect2 = Rectangle(8, 6)

print("Area of Rectangle 1:", rect1.area())
print("Area of Rectangle 2 using object as argument:", rect1.calculate_area(rect2))
```

```
Area of Rectangle 1: 50
Area of Rectangle 2 using object as argument: 48
```

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 [2]:
```
class Square:
    def __init__(self, side):
        self.side = side

    def area(self):
        result = self.side ** 2
        self.output(result)

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

side_length = float(input("Enter the side length of the square: "))
square = Square(side_length)
square.area()
```

```
Area of Square: 16.0
```

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 [5]:
```
class Rectangle:
    def __init__(self, length, width):
        if length == width:
            print("THIS IS SQUARE.")
        else:
            self.length = length
            self.width = width

    def area(self):
        result = self.length * self.width
        self.output(result)

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

    @classmethod
    def compare_sides(cls, length, width):
        if length == width:
            print("THIS IS SQUARE.")
            return None
        return cls(length, width)

length = float(input("Enter length of the rectangle: "))
width = float(input("Enter width of the rectangle: "))

rectangle = Rectangle.compare_sides(length, width)

if rectangle:
    rectangle.area()
```

```
Area of Rectangle: 30.0
```

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 [6]:
```
class Square:
    def __init__(self, side):
        self.__side = side

    def get_side(self):
        return self.__side

    def set_side(self, new_side):
        if new_side > 0:
            self.__side = new_side
        else:
            print("Side length must be positive.")

square = Square(5)
print("Current Side:", square.get_side())

square.set_side(7)
print("Updated Side:", square.get_side())

square.set_side(-3)
```

```
Current Side: 5
Updated Side: 7
Side length must be positive.
```

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 [7]:
```
class Profit:
    def getProfit(self):
        self.profit = float(input("Enter the profit amount: "))

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

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

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

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

```
Net Balance: 20900.0
```

15) WAP to demonstrate all types of inheritance.

In [8]:
```
# 1. Single Inheritance
class Parent:
    def show(self):
        print("Single Inheritance: Parent Class")

class Child(Parent):
    def display(self):
        print("Single Inheritance: Child Class")

obj1 = Child()
obj1.show()
obj1.display()

# 2. Multiple Inheritance
class Father:
    def father_method(self):
        print("Multiple Inheritance: Father Class")

class Mother:
    def mother_method(self):
        print("Multiple Inheritance: Mother Class")

class Child(Father, Mother):
    def child_method(self):
        print("Multiple Inheritance: Child Class")

obj2 = Child()
obj2.father_method()
obj2.mother_method()
obj2.child_method()

# 3. Multilevel Inheritance
class Grandparent:
    def grandparent_method(self):
        print("Multilevel Inheritance: Grandparent Class")

class Parent(Grandparent):
    def parent_method(self):
        print("Multilevel Inheritance: Parent Class")

class Child(Parent):
    def child_method(self):
        print("Multilevel Inheritance: Child Class")

obj3 = Child()
obj3.grandparent_method()
obj3.parent_method()
obj3.child_method()

# 4. Hierarchical Inheritance
class Parent:
    def parent_method(self):
        print("Hierarchical Inheritance: Parent Class")

class Child1(Parent):
    def child1_method(self):
        print("Hierarchical Inheritance: Child1 Class")
```

```
class Child2(Parent):
    def child2_method(self):
        print("Hierarchical Inheritance: Child2 Class")

obj4 = Child1()
obj5 = Child2()
obj4.parent_method()
obj4.child1_method()
obj5.parent_method()
obj5.child2_method()

# 5. Hybrid Inheritance
class A:
    def methodA(self):
        print("Hybrid Inheritance: Class A")

class B(A):
    def methodB(self):
        print("Hybrid Inheritance: Class B")

class C(A):
    def methodC(self):
        print("Hybrid Inheritance: Class C")

class D(B, C):
    def methodD(self):
        print("Hybrid Inheritance: Class D")

obj6 = D()
obj6.methodA()
obj6.methodB()
obj6.methodC()
obj6.methodD()
```

```
Single Inheritance: Parent Class
Single Inheritance: Child Class
Multiple Inheritance: Father Class
Multiple Inheritance: Mother Class
Multiple Inheritance: Child Class
Multilevel Inheritance: Grandparent Class
Multilevel Inheritance: Parent Class
Multilevel Inheritance: Child Class
Hierarchical Inheritance: Parent Class
Hierarchical Inheritance: Child1 Class
Hierarchical Inheritance: Parent Class
Hierarchical Inheritance: Child2 Class
Hybrid Inheritance: Class A
Hybrid Inheritance: Class B
Hybrid Inheritance: Class C
Hybrid Inheritance: 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 [9]:
```
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 Doe", 30, 50000)
emp.display()
```

```
Name: John Doe, 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.

In [10]:
```
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 [ ]: