Darshan UNIVERSITY
योगः कर्मसु कौशलम्

(https://www.darshan.ac.in/)

Python Programming - 2301CS404

Lab - 13 ¶

Rushi Gambhava

23010101082

Continued..

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

In [6]:
```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def calculate_area(rectangle):
        return rectangle.width * rectangle.height


rect = Rectangle(5, 10)
area = calculate_area(rect)
print(f"The area of the rectangle is: {area}")
```

```
The area of the rectangle is: 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().

In [5]:
```
class Square:
    def __init__(self, side_length):
        self.side_length = side_length

    def area(self):
        area_value = self.side_length ** 2
        self.output(area_value)

    def output(self, area_value):
        print(f"The area of the square with side length {self.side_length} is {area_v

# Example usage
square = Square(5)
square.area()
```

```
The area of the square with side length 5 is 25
```

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 [7]:
```
class Rectangle:
    def __init__(self, length, width):
        if length == width:
            print("THIS IS SQUARE.")
        else:
            self.length = length
            self.width = width

    def area(self):
        if hasattr(self, 'length') and hasattr(self, 'width'):
            area = self.length * self.width
            self.output(area)

    def output(self, calculated_area):
        print(f"The area of the rectangle is: {calculated_area}")

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

# Example usage
rect1 = Rectangle(10, 5)
rect1.area()

rect2 = Rectangle(4, 4)  # This will print "THIS IS SQUARE."
```

```
The area of the rectangle is: 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.

In [12]:
```
class Square:
    def __init__(self, side):
        self.__side = side  # private attribute

    def get_side(self):
        return self.__side

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

# Example usage:
square = Square(5)
print(square.get_side())  # Output: 5
square.set_side(10)
print(square.get_side())  # Output: 10
```

```
5
10
```

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 [13]:
```
class Profit:
    def getProfit(self):
        self.profit = float(input("Enter profit amount: "))
        return self.profit


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


class BalanceSheet(Profit, Loss):
    def getBalance(self):
        self.total_profit = self.getProfit()
        self.total_loss = self.getLoss()
        self.balance = self.total_profit - self.total_loss
        return self.balance

    def printBalance(self):
        print(f"Total Profit: {self.total_profit}")
        print(f"Total Loss: {self.total_loss}")
        print(f"Balance: {self.balance}")


# Example usage
if __name__ == "__main__":
    balance_sheet = BalanceSheet()
    balance_sheet.getBalance()
    balance_sheet.printBalance()
```

```
Enter profit amount: 10000
Enter loss amount: 2000
Total Profit: 10000.0
Total Loss: 2000.0
Balance: 8000.0
```

15) WAP to demonstrate all types of inheritance.

In [14]:

```
# Single Inheritance
class Animal:
    def speak(self):
        return "Animal speaks"

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

# Multiple Inheritance
class Father:
    def skills(self):
        return "Gardening, Painting"

class Mother:
    def skills(self):
        return "Cooking, Dancing"

class Child(Father, Mother):
    def talents(self):
        return "Singing"

# Multilevel Inheritance
class Grandparent:
    def wisdom(self):
        return "Wisdom from Grandparent"

class Parent(Grandparent):
    def experience(self):
        return "Experience from Parent"

class ChildMultilevel(Parent):
    def youthfulness(self):
        return "Youthfulness from Child"

# Hierarchical Inheritance
class Base:
    def base_method(self):
        return "Base method called"

class Derived1(Base):
    def derived1_method(self):
        return "Derived1 method called"

class Derived2(Base):
    def derived2_method(self):
        return "Derived2 method called"

# Hybrid Inheritance
class BaseHybrid:
    def base_hybrid_method(self):
        return "Base method in Hybrid"

class DerivedA(BaseHybrid):
    pass

class DerivedB(BaseHybrid):
    pass

class MoreComplex(DerivedA, DerivedB):
    pass

# Demonstration
# Single Inheritance
dog = Dog()
```

```
print(dog.speak())
print(dog.bark())

# Multiple Inheritance
child = Child()
print(child.skills())  # From Father
print(child.talents())
print(child.skills())  # From Mother

# Multilevel Inheritance
child_ml = ChildMultilevel()
print(child_ml.wisdom())
print(child_ml.experience())
print(child_ml.youthfulness())

# Hierarchical Inheritance
derived1 = Derived1()
derived2 = Derived2()
print(derived1.base_method())
print(derived1.derived1_method())
print(derived2.base_method())
print(derived2.derived2_method())

# Hybrid Inheritance
hybrid_obj = MoreComplex()
print(hybrid_obj.base_hybrid_method())
```

```
Animal speaks
Dog barks
Gardening, Painting
Singing
Gardening, Painting
Wisdom from Grandparent
Experience from Parent
Youthfulness from Child
Base method called
Derived1 method called
Base method called
Derived2 method called
Base method in Hybrid
```

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 [15]:
```
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

# Example of creating an instance of Employee
employee = Employee("Alice", 30, 50000)
print(employee.name)  # Output: Alice
print(employee.age)   # Output: 30
print(employee.salary)  # Output: 50000
```

```
Alice
30
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 [16]:
```
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")

# Creating a list of Shape objects
shapes = [Rectangle(), Circle(), Triangle()]

# Iterating through the list and calling draw method on each object
for shape in shapes:
    shape.draw()
```

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