Darshan
UNIVERSITY
योग: कर्मसु कौशलम्

Python Programming - 2301CS404

Lab - 13

Name: Jadeja Rudrarajsinh

Enrollment No:23010101411

Roll No:487

Continued..

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

In [2]:

```python
class Rectangle:
    def __init__(self,l,b):
        self.l=l
        self.b=b
    def area(self,r):
        return r.l*r.b
l = int(input("Enter length of rectangle"))
b = int(input("Enter breadth of rectangle"))

rect = Rectangle(l,b)
print("Area =",rect.area(rect))
```

```
Area = 81
```

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 [6]:

```
class Square:
    def __init__(self,l):
        self.l=l
    def area(self):
        area_val = self.l*self.l
        self.output(area_val)
    def output(self,area_val):
        print(f"The area of the square =",area_val)

length = int(input("Enter the length of square : "))
square = Square(length)
square.area()
```

```
The area of the square = 64
```

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 [10]:

```
class Rectangle:
    def __init__(self,l,b):
        self.l=l
        self.b=b
    def area(self):
        area_val=self.l*self.b
        self.output(area_val)
    def output(self,area_val):
        print("Area =",area_val)
    @classmethod
    def compare_sides(cls,length,breadth):
        if length!=breadth:
            return True
        else:
            print("THIS IS SQUARE")
            return False
l = int(input("Enter length of rectangle"))
b = int(input("Enter breadth of rectangle"))
rect = Rectangle(l,b)
if rect.compare_sides(l,b):
    o.area()
```

```
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:
    __side=0
    def get_side(self):
        return self.__side
    def set_side(self,side):
        self.__side=side
s = int(input("Enter side of square"))
square = Square()
square.set_side(s)
print("Side of square = ",square.get_side())
```

```
Side of square =  4
```

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 [14]:

```
class Profit:
    def getProfit(self, profit):
        self.profit = profit

class Loss:
    def getLoss(self, loss):
        self.loss = loss

class BalanceSheet(Profit, Loss):
    def __init__(self, balance):
        self.balance = balance

    def getBalance(self):
        return self.balance

    def printBalance(self):
        print("Current Balance =", self.balance)

balance = int(input("Enter current balance: "))
o = BalanceSheet(balance)

choice = 0
while choice != -1:
    print("\nEnter 1 for profit")
    print("Enter 2 for loss")
    print("Enter 3 for print balance")
    print("Enter -1 to exit")
    choice = int(input("Enter your choice: "))
    match(choice):
        case 1:
            profit = float(input("Enter profit amount: "))
            o.getProfit(profit)
            o.balance += profit
        case 2:
            loss = float(input("Enter loss amount: "))
            o.getLoss(loss)
            o.balance -= loss
        case 3:
            o.printBalance()
        case -1:
            print("Exiting the program. Goodbye!")
        case _:
            print("Invalid choice. Please try again.")
```

```
Enter 1 for profit
Enter 2 for loss
Enter 3 for print balance
Enter -1 to exit
Enter 1 for profit
Enter 2 for loss
Enter 3 for print balance
Enter -1 to exit
Enter 1 for profit
Enter 2 for loss
Enter 3 for print balance
Enter -1 to exit
Exiting the program. Goodbye!
```

15) WAP to demonstrate all types of inheritance.

In [16]:

```
# Single Inheritance
class ParentSingle:
    def func1(self):
        print("This is a parent class for Single Inheritance.")

class ChildSingle(ParentSingle):
    def func2(self):
        print("This is a child class inheriting from ParentSingle.")

# Multiple Inheritance
class ClassA:
    def funcA(self):
        print("This is ClassA.")

class ClassB:
    def funcB(self):
        print("This is ClassB.")

class ClassC(ClassA, ClassB):
    def funcC(self):
        print("This is ClassC inheriting from ClassA and ClassB.")

# Multilevel Inheritance
class GrandParentMulti:
    def funcGP(self):
        print("This is the grandparent class for Multilevel Inheritance.")

class ParentMulti(GrandParentMulti):
    def funcP(self):
        print("This is the parent class inheriting from GrandParentMulti.")

class ChildMulti(ParentMulti):
    def funcC(self):
        print("This is the child class inheriting from ParentMulti.")

# Hierarchical Inheritance
class ParentHierarchical:
    def funcP(self):
        print("This is the parent class for Hierarchical Inheritance.")

class Child1(ParentHierarchical):
    def funcC1(self):
        print("This is Child1 inheriting from ParentHierarchical.")

class Child2(ParentHierarchical):
    def funcC2(self):
        print("This is Child2 inheriting from ParentHierarchical.")

# Hybrid Inheritance
class Base:
    def funcBase(self):
        print("This is the base class.")

class Derived1(Base):
    def funcD1(self):
        print("This is Derived1 inheriting from Base.")

class Derived2(Base):
    def funcD2(self):
        print("This is Derived2 inheriting from Base.")

class Hybrid(Derived1, Derived2):
    def funcHybrid(self):
        print("This is Hybrid inheriting from Derived1 and Derived2.")

# Demonstrate all types of inheritance
print("Single Inheritance:")
single_child = ChildSingle()
single_child.func1()
single_child.func2()

print("\nMultiple Inheritance:")
multi_obj = ClassC()
multi_obj.funcA()
multi_obj.funcB()
multi_obj.funcC()

print("\nMultilevel Inheritance:")
multi_lvl_child = ChildMulti()
multi_lvl_child.funcGP()
multi_lvl_child.funcP()
multi_lvl_child.funcC()

print("\nHierarchical Inheritance:")
hier_child1 = Child1()
hier_child2 = Child2()
hier_child1.funcP()
hier_child1.funcC1()
hier_child2.funcP()
hier_child2.funcC2()

print("\nHybrid Inheritance:")
hybrid_obj = Hybrid()
```

```
hybrid_obj.funcBase()
hybrid_obj.funcD1()
hybrid_obj.funcD2()
hybrid_obj.funcHybrid()
```

```
Single Inheritance:
This is a parent class for Single Inheritance.
This is a child class inheriting from ParentSingle.

Multiple Inheritance:
This is ClassA.
This is ClassB.
This is ClassC inheriting from ClassA and ClassB.

Multilevel Inheritance:
This is the grandparent class for Multilevel Inheritance.
This is the parent class inheriting from GrandParentMulti.
This is the child class inheriting from ParentMulti.

Hierarchical Inheritance:
This is the parent class for Hierarchical Inheritance.
This is Child1 inheriting from ParentHierarchical.
This is the parent class for Hierarchical Inheritance.
This is Child2 inheriting from ParentHierarchical.

Hybrid Inheritance:
This is the base class.
This is Derived1 inheriting from Base.
This is Derived2 inheriting from Base.
This is Hybrid inheriting from Derived1 and Derived2.
```

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 [18]:

```
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_details(self):
        print(f"Name:{self.name}")
        print(f"Age:{self.age}")
        print(f"Salary:{self.salary}")
name=input("Enter name")
age=int(input("Enter age"))
salary=int(input("Enter salary"))
```

```
e = Employee(name,age,salary)
e.display_details()
```

```
Name:Jadeja Rudrarajsinh
Age:20
Salary:9999
```

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]:

```
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 O")
class Triangle(Shape):
    def draw(self):
        print("Drawing a Triangle Δ")
shapes = [Rectangle(),Circle(),Triangle()]
for shape in shapes:
    shape.draw()
```

```
Drawing a Rectangle []
Drawing a Circle O
Drawing a Triangle Δ
```