Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Python Programming - 2301CS404

Lab - 13_2

Jeet Bhalodi (23031701006)
11-03-2025

Continued..

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

In [3]:
```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

rect = Rectangle(10, 5)

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):
```

```
        self.side = side

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

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

square = Square(4)

square.area()
```

```
The area of the square with side 4 is: 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.

In [7]:
```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

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

    def output(self, area):
        print(f"The area of the rectangle with length {self.length} and width {self

rectangle = Rectangle.validate_sides(10, 5)
if rectangle:
    rectangle.area()

square_check = Rectangle.validate_sides(6, 6)
```

```
The area of the rectangle with length 10 and width 5 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 [9]:
```python
class Square:
    def __init__(self, side):
        self.__side = side

    def get_side(self):
        return self.__side

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

square = Square(5)

print(f"Initial side length: {square.get_side()}")
square.set_side(8)
print(f"Updated side length: {square.get_side()}")

square.set_side(-3)
```

```
Initial side length: 5
Updated side length: 8
Side length must be a positive value!
```

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 [11]:
```python
class Profit:
    def __init__(self):
        self.profit = 0

    def getProfit(self):
        self.profit = float(input("Enter the profit amount: "))

class Loss:
    def __init__(self):
```

```
        self.loss = 0

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

class BalanceSheet(Profit, Loss):
    def __init__(self):
        Profit.__init__(self)
        Loss.__init__(self)
        self.balance = 0

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

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

balance_sheet = BalanceSheet()

balance_sheet.getProfit()
balance_sheet.getLoss()

balance_sheet.getBalance()
balance_sheet.printBalance()
```

```
The balance is: 15500.0
```

15) WAP to demonstrate all types of inheritance.

In [27]:

```
class A:
    def displayA(self):
        print("Single Inheritance: Class A")

class B(A):
    def displayB(self):
        print("Derived Class B from A")

class C:
    def displayC(self):
        print("Multiple Inheritance: Class C")

class D(A, C):
    def displayD(self):
        print("Derived Class D from A and C")

class E(A):
    def displayE(self):
        print("Multilevel Inheritance: Class E derived from A")

class F(E):
    def displayF(self):
        print("Class F derived from E (A -> E -> F)")


class G(A):
```

```
    def displayG(self):
        print("Hierarchical Inheritance: Class G derived from A")

class H(A):
    def displayH(self):
        print("Class H derived from A")

class Base:
    def showBase(self):
        print("Hybrid Inheritance: Base Class")

class X(Base):
    def showX(self):
        print("Class X derived from Base")

class Y(Base):
    def showY(self):
        print("Class Y derived from Base")

class Z(X, Y):
    def showZ(self):
        print("Class Z derived from X and Y (Both from Base)")


objB = B()
objB.displayA()
objB.displayB()
print("\n")

objD = D()
objD.displayA()
objD.displayC()
objD.displayD()
print("\n")

objF = F()
objF.displayA()
objF.displayE()
objF.displayF()
print("\n")

objG = G()
objG.displayA()
objG.displayG()

objH = H()
objH.displayA()
objH.displayH()
print("\n")

objZ = Z()
objZ.showBase()
objZ.showX()
objZ.showY()
objZ.showZ()
```

```
Single Inheritance: Class A
Derived Class B from A


Single Inheritance: Class A
Multiple Inheritance: Class C
Derived Class D from A and C


Single Inheritance: Class A
Multilevel Inheritance: Class E derived from A
Class F derived from E (A -> E -> F)


Single Inheritance: Class A
Hierarchical Inheritance: Class G derived from A
Single Inheritance: Class A
Class H derived from A


Hybrid Inheritance: Base Class
Class X derived from Base
Class Y derived from Base
Class Z derived from X and Y (Both from Base)
```

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 [23]:
```python
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

employee = Employee("Alice", 30, 50000)

print(f"Name: {employee.name}")
print(f"Age: {employee.age}")
print(f"Salary: {employee.salary}")
```

```
Name: Alice
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 [13]:
```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
```