Python Programming - 2301CS404

Lab - 13

Smit Maru - 23010101161 - 260

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

def display_area(rect):
    print(f"Area of the rectangle: {rect.calculate_area()}")

my_rect = Rectangle(10, 5)

display_area(my_rect)
```

```
Area of the rectangle: 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"Area of the square: {area}")

square = Square(6)

square.area()
```

```
Area of the square: 36
```

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 [9]:
```
class Rectangle:
    def __init__(self, length, width):
        if self.is_square(length, width):
            print("THIS IS SQUARE")
            self.length = self.width = None
        else:
            self.length = length
            self.width = width

    def area(self):
        if self.length is not None and self.width is not None:
            area = self.length * self.width
            self.output(area)
            return area

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

    @staticmethod
    def is_square(length, width):
        return length == width


rect1 = Rectangle(5, 10)
if rect1.length is not None:
```

```
    rect1.area()

rect2 = Rectangle(8, 8)
```

```
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 [11]:
```
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 positive.")

sq = Square(5)
print("Side:", sq.get_side())
sq.set_side(10)
print("Updated Side:", sq.get_side())

sq.set_side(-3)
```

```
Side: 5
Updated Side: 10
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 [15]:
```
class Profit:
    def __init__(self):
        self.profit = 0


    def getProfit(self):
```

```
        self.profit = float(input("Enter the profit: "))

class Loss:
    def __init__(self):
        self.loss = 0


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

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: 950000.0
```

15) WAP to demonstrate all types of inheritance.

In [13]:

```
class Fruit:
    def taste(self):
        print("Fruits have different tastes.")

class Mango(Fruit):
    def color(self):
        print("Mango is yellow.")


class Alphonso(Mango):
    def size(self):
        print("Alphonso mango is small.")


class Citrus:
    def vitamin_c(self):
        print("Citrus fruits are rich in Vitamin C.")

class Orange(Fruit, Citrus):
    def peel(self):
```

```
        print("Orange has a thick peel.")

class Apple(Fruit):
    def shape(self):
        print("Apple is round.")


class Banana(Fruit, Citrus):
    def energy(self):
        print("Banana gives instant energy.")


mango = Mango()
mango.taste()
mango.color()

alphonso = Alphonso()
alphonso.taste()
alphonso.color()
alphonso.size()

orange = Orange()
orange.taste()
orange.vitamin_c()
orange.peel()

apple = Apple()
apple.taste()
apple.shape()

banana = Banana()
banana.taste()
banana.vitamin_c()
banana.energy()
```

```
Fruits have different tastes.
Mango is yellow.
Fruits have different tastes.
Mango is yellow.
Alphonso mango is small.
Fruits have different tastes.
Citrus fruits are rich in Vitamin C.
Orange has a thick peel.
Fruits have different tastes.
Apple is round.
Fruits have different tastes.
Citrus fruits are rich in Vitamin C.
Banana gives instant energy.
```

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]:
```
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("namra", 30, 50000)

emp.display()
```

```
Name: namra, 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 [19]:
```
from abc import ABC, abstractmethod

class Shape(ABC):
    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 [ ]: