# o21lcr7tx

March 12, 2025

Python Programming - 2301CS404

`<center><b><h1> 23010101202 | KHUSHI PATEL | 11-3-2025 </b></center>`

Lab - 13

## 0.1 Continued..

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

[5]:
```
class rectangle:
    def __init__(self,l,b):
        self.l=l;
        self.b=b;

    def area(self):
        return f"Area : {self.l*self.b}"
a=rectangle(5,2)
a.area()
```

[5]:
```
'Area : 10'
```

### 0.1.2 11) Calculate the area of a square.

### 0.1.3 Include a Constructor, a method to calculate area named area() and a method named output() that prints the output and is invoked by area().

[13]:
```
class square:
    def __init__(self,l):
        self.l=l;
    def area(self):
        self.output()
    def output(self):
        print(f'Area:{self.l*self.l}')
s=square(5)
s.area()
```

```
Area:25
```

### 0.1.4 12) Calculate the area of a rectangle.

### 0.1.5 Include a Constructor, a method to calculate area named area() and a method named output() that prints the output and is invoked by area().

### 0.1.6 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.

[36]:
```
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:
            return "THIS IS SQUARE."
        return "THIS IS A RECTANGLE."

rect1 = Rectangle(10, 5)
rect1.area()

rect2 = Rectangle(6, 6)
```

```
Area of Rectangle: 50
THIS IS SQUARE.
```

### 0.1.7 13) Define a class Square having a private attribute "side".

### 0.1.8 Implement get_side and set_side methods to accees the private attribute from outside of the class.

[39]:
```
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(square.get_side())

square.set_side(10)
print(square.get_side())
```

```
5
10
```

### 0.1.9 14) Create a class Profit that has a method named getProfit that accepts profit from the user.

### 0.1.10 Create a class Loss that has a method named getLoss that accepts loss from the user.

### 0.1.11 Create a class BalanceSheet that inherits from both classes Profit and Loss and calculates the balanace. It has two methods getBalance() and printBalance().

[40]:
```
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()
```

```
Enter the profit amount:  10000
Enter the loss amount:  500

Net Balance: 9500.0
```

### 0.1.12 15) WAP to demonstrate all types of inheritance.

[1]:
```python
# Single Inheritance
class Animal:
    def speak(self):
        print("Animal sound")

class Dog(Animal):
    def bark(self):
        print("Woof!")

# Multiple Inheritance
class Swimmer:
    def swim(self):
        print("Swimming")

class Walker:
    def walk(self):
        print("Walking")

class Duck(Swimmer, Walker):
    def quack(self):
        print("Quack!")

# Multilevel Inheritance
class Grandparent:
    def grandparent_method(self):
        print("Grandparent feature")

class Parent(Grandparent):
    def parent_method(self):
        print("Parent feature")

class Child(Parent):
    def child_method(self):
        print("Child feature")

# Hierarchical Inheritance
class Vehicle:
    def start(self):
        print("Vehicle started")

class Car(Vehicle):
    def drive(self):
        print("Driving car")

class Bike(Vehicle):
    def pedal(self):
        print("Pedaling bike")


#Demonstration

print("Single Inheritance:")
my_dog = Dog()
my_dog.speak() # Inherited from Animal
my_dog.bark()  # Specific to Dog

print("\nMultiple Inheritance:")
my_duck = Duck()
my_duck.swim()  # Inherited from Swimmer
my_duck.walk()  # Inherited from Walker
my_duck.quack() # Specific to Duck

print("\nMultilevel Inheritance:")
my_child = Child()
my_child.grandparent_method() # Inherited from Grandparent
my_child.parent_method()     # Inherited from Parent
my_child.child_method()      # Specific to Child

print("\nHierarchical Inheritance:")
my_car = Car()
my_car.start() # Inherited from Vehicle
my_car.drive() # Specific to Car

my_bike = Bike()
my_bike.start() # Inherited from Vehicle
my_bike.pedal() # Specific to Bike
```

```
Single Inheritance:
Animal sound
Woof!

Multiple Inheritance:
Swimming
Walking
Quack!

Multilevel Inheritance:
Grandparent feature
Parent feature
```

```
Child feature

Hierarchical Inheritance:
Vehicle started
Driving car
Vehicle started
Pedaling bike
```

### 0.1.13 16) Create a Person class with a constructor that takes two arguments name and age.

### 0.1.14 Create a child class Employee that inherits from Person and adds a new attribute salary.

### 0.1.15 Override the init method in Employee to call the parent class's init method using the super() and then initialize the salary attribute.

[51]:
```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

    def display(self):
        print(f"Name: {self.name} \nAge: {self.age}\nSalary: {self.salary}")

emp = Employee("Khushi", 30,50000)
emp.display()
```

```
Name: Khushi
Age: 30
Salary: 50000
```

### 0.1.16 17) Create a Shape class with a draw method that is not implemented.

### 0.1.17 Create three child classes Rectangle, Circle, and Triangle that implement the draw method with their respective drawing behaviors.

### 0.1.18 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.

[52]:
```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
```