python-programming-lab-13-2

March 13, 2025

Python Programming - 2301CS404

Gohel Neel

Enrollnment No. : 23010101089

Roll No. 340

Date: 10-03-2025

Lab - 13_2

0.1 Continued..

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

[3]:
```
class Rectangle:
    def __init__(self):
        self.length = int(input("Enter Length "))
        self.width = int(input("Enter Width "))

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

R1 = Rectangle()
area = calculate_area(R1)

print("The area is: ",area)
```

```
Enter Length  10
Enter Width  10
The area is:  100
```

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().

[4]:
```
class Square:
    def __init__(self):
        self.l = int(input("Enter of square: "))

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

    def output(self, area):
        print("area of square is: ",area)

square = Square()
square.area()
```

```
Enter of square:  10
area of square is:  100
```

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.

[17]:
```
class Rectangle:
    def __init__(self):
        self.length = int(input("Enter the length: "))
        self.width = int(input("Enter the width: "))
        if self.is_square():
            print("THIS IS A SQUARE")
        else:
            self.area()

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

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

    def is_square(self):
```

```
        return self.length == self.width

R1 = Rectangle()
```

```
Enter the length:  4
Enter the width:  5

The area of the rectangle is: 20
```

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.

[25]:
```
class Square:
    def __init__(self):
        self._side = int(input("Enter side of square: "))

    def get_side(self):
        return self._side

    def set_side(self):
        new_side = int(input("Enter new side of square: "))
        if new_side > 0:
            self._side = new_side
        else:
            print("Side must be positive.")

square = Square()

print("Side of square:", square.get_side())

square.set_side()

print("Updated side :", square.get_side())
```

```
Enter side of square:  10

Side of square: 10

Enter new side of square:  45

Updated side : 45
```

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().

[26]:
```
class Profit:
    def __init__(self):
        self.profit = 0

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

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

    def getLoss(self):
        self.loss = int(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("The balance is: ",self.balance)

balance_sheet = BalanceSheet()

balance_sheet.getProfit()
balance_sheet.getLoss()

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

```
Enter the profit:  100
Enter the loss:  20
```

```
The balance is:  80
```

0.1.12 15) WAP to demonstrate all types of inheritance.

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

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

# Multiple Inheritance
class Person:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        print(f"Hello, my name is {self.name}")

class Employee:
    def __init__(self, employee_id):
        self.employee_id = employee_id

    def show_id(self):
        print(f"My employee ID is {self.employee_id}")

class Manager(Person, Employee):
    def __init__(self, name, employee_id):
        Person.__init__(self, name)
        Employee.__init__(self, employee_id)

    def work(self):
        print(f"{self.name} is working as a Manager.")

# Multilevel Inheritance
class Vehicle:
    def start_engine(self):
        print("Vehicle engine started")

class Car(Vehicle):
    def drive(self):
        print("Car is driving")

class SportsCar(Car):
    def speed(self):
        print("Sports car is speeding")
```

```python
# Hierarchical Inheritance
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def area(self, radius):
        return 3.14 * radius * radius

class Rectangle(Shape):
    def area(self, length, width):
        return length * width

# Hybrid Inheritance (Combination of Multiple and Multilevel Inheritance)
class School:
    def __init__(self, name):
        self.name = name

    def show_name(self):
        print(f"School Name: {self.name}")

class Teacher(School):
    def __init__(self, name, subject):
        School.__init__(self, name)
        self.subject = subject

    def teach(self):
        print(f"Teaching {self.subject}")

class HeadTeacher(Teacher):
    def __init__(self, name, subject, head_teacher_name):
        Teacher.__init__(self, name, subject)
        self.head_teacher_name = head_teacher_name

    def manage(self):
        print(f"{self.head_teacher_name} manages the teaching process.")


# 1. Single Inheritance
print("Single Inheritance:")
dog = Dog()
dog.speak()
dog.bark()
print()

# 2. Multiple Inheritance
print("Multiple Inheritance:")
```

```
manager = Manager("ABC", 101)
manager.introduce()
manager.show_id()
manager.work()
print()

# 3. Multilevel Inheritance
print("Multilevel Inheritance:")
sports_car = SportsCar()
sports_car.start_engine()
sports_car.drive()
sports_car.speed()
print()

# 4. Hierarchical Inheritance
print("Hierarchical Inheritance:")
circle = Circle()
print(f"Circle Area: {circle.area(5)}")
rectangle = Rectangle()
print(f"Rectangle Area: {rectangle.area(10, 5)}")
print()

# 5. Hybrid Inheritance
print("Hybrid Inheritance:")
head_teacher = HeadTeacher("DEF", "Maths", "ABC")
head_teacher.show_name()
head_teacher.teach()
head_teacher.manage()
```

```
Single Inheritance:
Animal speaks
Dog barks

Multiple Inheritance:
Hello, my name is ABC
My employee ID is 101
ABC is working as a Manager.

Multilevel Inheritance:
Vehicle engine started
Car is driving
Sports car is speeding

Hierarchical Inheritance:
Circle Area: 78.5
Rectangle Area: 50
```

```
Hybrid Inheritance:
School Name: DEF
Teaching Maths
ABC manages the teaching process.
```

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.

[2]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")

class Employee(Person):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary = salary

    def display_employee_info(self):
        self.display_info()
        print(f"Salary: {self.salary}")

emp = Employee("ABC", 30, 50000)
emp.display_employee_info()
```

```
Name: ABC
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.

[3]:
```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
```

[ ]:
```
```