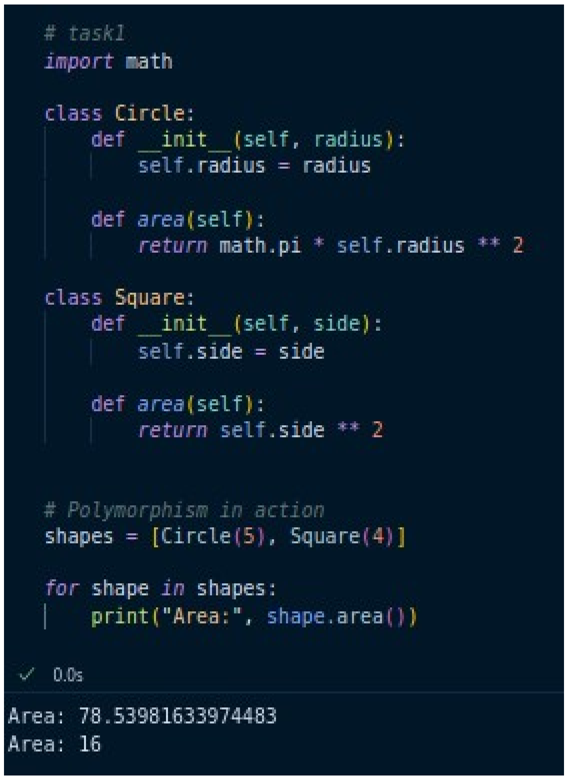
Python OOP Practice Tasks

This document contains 10 tasks designed to practice Polymorphism, Operator Overloading, Magic/Dunder Functions, Dynamic Polymorphism, Abstract Classes, Empty Classes, Data Classes, and Keyword Arguments. The difficulty progresses from upper-basic to lower-advanced.

Polymorphism & Operator Overloading

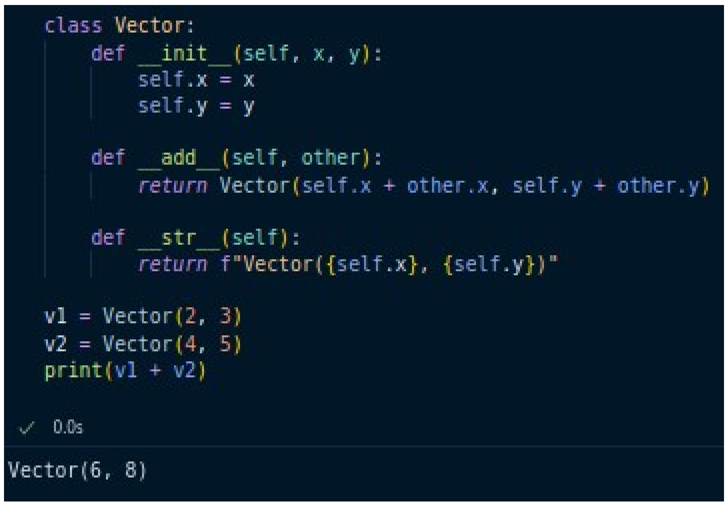
- Task 1: Create two classes Circle and Square with a method area(). Use polymorphism to call the area() method on both objects in a loop.



```python
# task1
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

class Square:
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side ** 2


# Polymorphism in action
shapes = [Circle(5), Square(4)]

for shape in shapes:
    print("Area:", shape.area())
```

```
✓ 0.0s
Area: 78.53981633974483
Area: 16
```

- Task 2: Create a Vector class with two attributes x and y. Implement operator overloading for + (using __add__) so that v1 + v2 adds their coordinates. Test with two Vector objects.



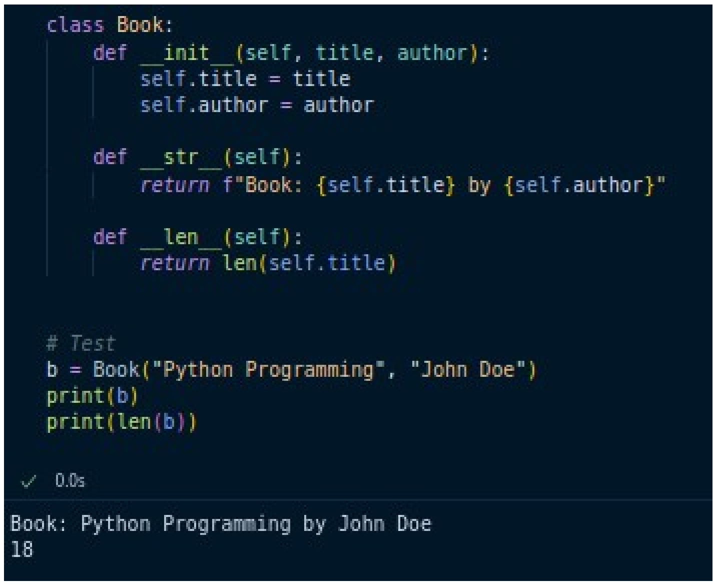
```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"Vector({self.x}, {self.y})"

v1 = Vector(2, 3)
v2 = Vector(4, 5)
print(v1 + v2)
```

✓ 0.0s

```
Vector(6, 8)
```

Magic Functions / Dunder Functions

- Task 3: Create a class Book with attributes title and author. Implement __str__ so that printing the object shows: 'Book: Title by Author'. Implement __len__ to return the length of the title.

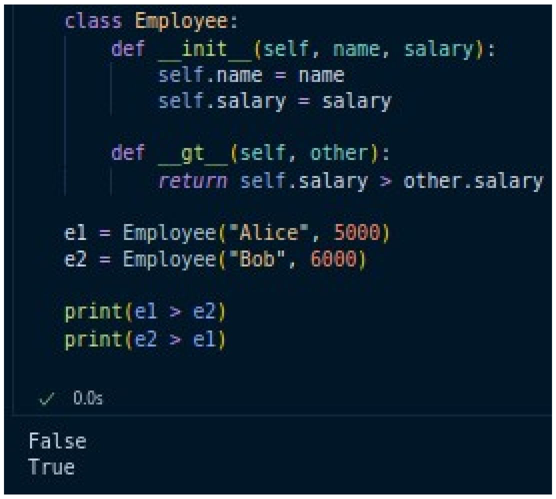


```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"Book: {self.title} by {self.author}"

    def __len__(self):
        return len(self.title)


# Test
b = Book("Python Programming", "John Doe")
print(b)
print(len(b))
```

✓ 0.0s

```
Book: Python Programming by John Doe
18
```

- Task 4: Create a class Employee with attributes name and salary. Overload the > operator (__gt__) to compare two employees by salary. Test by comparing two employees.



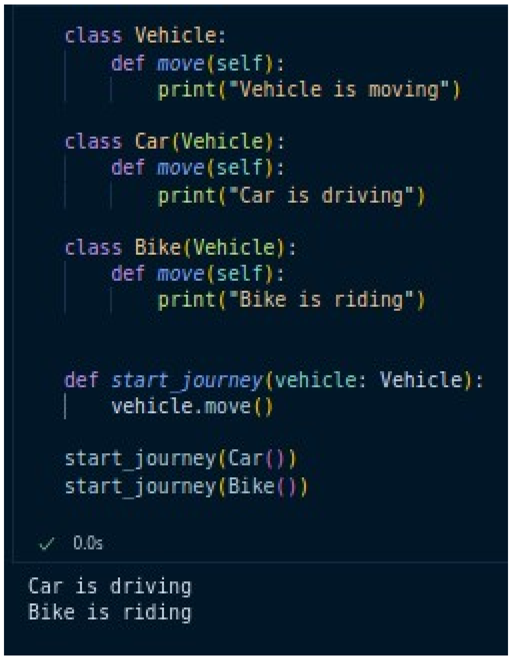
```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def __gt__(self, other):
        return self.salary > other.salary

e1 = Employee("Alice", 5000)
e2 = Employee("Bob", 6000)

print(e1 > e2)
print(e2 > e1)
```

```
✓ 0.0s
```

```
False
True
```

Dynamic Polymorphism (Subclass as Base Class)

- Task 5: Create a base class Vehicle with a method move(). Create subclasses Car and Bike, overriding move(). Write a function start_journey(vehicle: Vehicle) that accepts any vehicle and calls move(). Test with both Car and Bike objects.

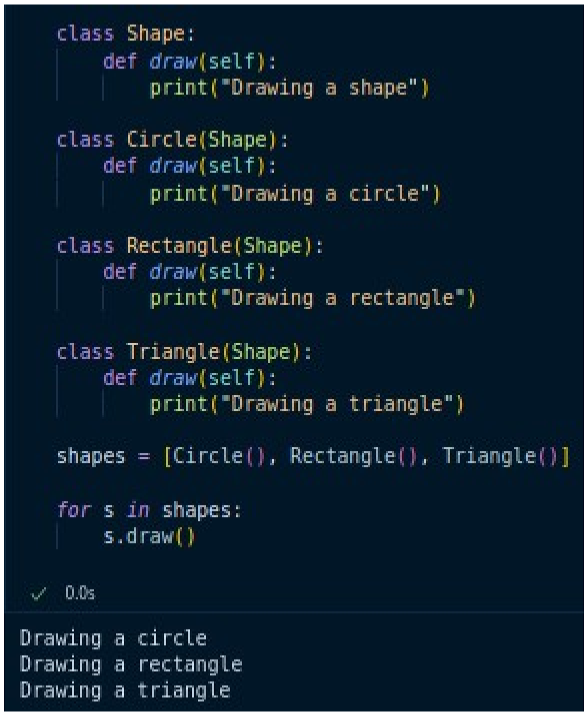


```python
class Vehicle:
    def move(self):
        print("Vehicle is moving")

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

class Bike(Vehicle):
    def move(self):
        print("Bike is riding")


def start_journey(vehicle: Vehicle):
    vehicle.move()

start_journey(Car())
start_journey(Bike())
```

```
✓ 0.0s
```

```
Car is driving
Bike is riding
```

- Task 6: Implement a base class Shape with draw() method. Subclasses: Circle, Rectangle, Triangle. Write a loop that takes a list of mixed shapes and calls draw() on each.



```
class Shape:
    def draw(self):
        print("Drawing a shape")

class Circle(Shape):
    def draw(self):
        print("Drawing a circle")

class Rectangle(Shape):
    def draw(self):
        print("Drawing a rectangle")

class Triangle(Shape):
    def draw(self):
        print("Drawing a triangle")

shapes = [Circle(), Rectangle(), Triangle()]

for s in shapes:
    s.draw()
```

✓ 0.0s

```
Drawing a circle
Drawing a rectangle
Drawing a triangle
```

Abstract Class / Empty Class / Data Class

- Task 7: Create an abstract class Appliance with an abstract method turn_on(). Implement subclasses WashingMachine and Refrigerator. Each should implement turn_on() differently.
- Task 8: Create an empty class Placeholder using pass. Dynamically add attributes name and value after creating the object. Print them.
- Task 9: Create a data class Student with fields name, age, and grade. Create 3 students and store them in a list. Loop through and print their details.

Keyword Arguments

- Task 10: Write a function register_employee(name, age, role, salary) that prints employee details. Call it once using positional arguments and once using keyword arguments in different order.