

OOPS in Python: Full Session with Code Examples

■ Introduction to OOPS:

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects" that contain both data (attributes) and methods (functions). It helps in organizing complex code by grouping related variables and functions into objects.

■ 4 Pillars of OOPS:

1. **Encapsulation**: Wrapping the data and methods into a single unit (class). 2. **Inheritance**: Deriving a new class from an existing class. 3. **Polymorphism**: Using a single method name but having different implementations. 4. **Abstraction**: Hiding unnecessary details and showing only essential information.

■ Encapsulation Example:

```
class Shape:
    def __init__(self, color):
        self.color = color
    def display_color(self):
        print(f"The color of the shape is {self.color}")
circle = Shape("Red")
square = Shape("Blue")
circle.display_color() # Output: The color of the shape is Red
square.display_color() # Output: The color of the shape is Blue
```

■ Inheritance Example:

```
class Shape:
    def __init__(self, color):
        self.color = color
    def display_color(self):
        print(f"The color of the shape is {self.color}")
class Circle(Shape):
    def __init__(self, color, radius):
        super().__init__(color)
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
circle = Circle("Green", 5)
circle.display_color() # Output: The color of the shape is Green
print(circle.area()) # Output: 78.5
```

■ Polymorphism Example:

```
class Shape:
    def area(self):
        print("Area calculation is not defined for generic shape")
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
circle = Circle(7)
print(circle.area()) # Output: 153.86
```

■ Abstraction Example:

```
from abc import ABC, abstractmethod
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
circle = Circle(4)
print(circle.area()) # Output: 50.24
```

■ Class Method Example:

```
class Shape: shape_count = 0 def __init__(self, name): self.name = name
Shape.shape_count += 1 @classmethod def display_count(cls): print(f"Total
shapes created: {cls.shape_count}") circle = Shape("Circle") square =
Shape("Square") Shape.display_count() # Output: Total shapes created: 2
```

■ Static Method Example:

```
class MathOperations: @staticmethod def add(x, y): return x + y
@staticmethod def multiply(x, y): return x * y
print(MathOperations.add(10, 5)) # Output: 15
print(MathOperations.multiply(4, 3)) # Output: 12
```

■ Combining Both Class and Static Methods:

```
class Account: interest_rate = 0.05 def __init__(self, balance):
self.balance = balance @classmethod def set_interest_rate(cls, rate):
cls.interest_rate = rate @staticmethod def validate_amount(amount): return
amount > 0 Account.set_interest_rate(0.07) print(Account.interest_rate) #
Output: 0.07 print(Account.validate_amount(100)) # Output: True
print(Account.validate_amount(-50)) # Output: False
```

■ Difference Between Class Method and Static Method:

Feature	Class Method	Static Method
Decorator	@classmethod	@staticmethod
First parameter	cls (class reference)	No cls or self parameter
Access	Can access class-level attributes	Cannot access class or instance-level attributes
Usage	Used for class-level operations, alternative constructors	Used for utility or helper functions
Binding	Bound to the class	Independent of the class