OOPS Concepts in Python with Shape Class Example

1. Encapsulation: Wrapping the data (variables) and methods into a single unit (class). It helps in protecting the object's data by making attributes private.

```
class Shape:
    def __init__(self, color):
        self.color = color  # Encapsulating color attribute

    def display_color(self):
        print(f"The color of the shape is {self.color}")

# Creating Objects
circle = Shape("Red")
square = Shape("Blue")

circle.display_color()
square.display_color()
```

2. Inheritance: Creating a new class from an existing class. The child class inherits the properties and methods of the parent class.

```
class Shape:
    def __init__(self, color):
        self.color = color

    def display_color(self):
        print(f"The color of the shape is {self.color}")

# Child class inheriting from Shape
class Circle(Shape):
    def __init__(self, color, radius):
        super().__init__(color)
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

# Creating Objects
circle = Circle("Green", 5)
circle.display_color()
print(f"Circle Area: {circle.area()}")
```

3. Polymorphism: Using the same method name but with different implementations in different classes.

```
class Shape:
    def area(self):
        print("Area calculation is not defined")
```

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

    def area(self):
        return 3.14 * self.radius ** 2

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

# Polymorphism in action
shapes = [Circle(7), Rectangle(4, 6)]
for shape in shapes:
    print(f"Area: {shape.area()}")
```

4. Abstraction: Hiding unnecessary details and showing only the essential information using abstract classes.

```
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

# Using abstract class
circle = Circle(5)
print(f"Circle Area: {circle.area()}")
```