JOMHACK
INSPIRE INNOVATION

PYTHON
BOOTCAMP

www.jomhack.com

OBJECT ORIENTED PROGRAMMING

Inheritance:

- Relationship between classes
- Child classes can override or extend parent functionality

Polymorphism:

- Using the same interface for different types
- Same method can behave differently for different objects

OBJECT ORIENTED PROGRAMMING

Inheritance:

- Relationship between classes
- Child classes can override or extend parent functionality

```python
# Inheritance
class Shape:  # Parent class
    def __init__(self, name):
        self.name = name

    def area(self):
        return 0

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

    def area(self):  # Override parent method
        return 3.14 * self.radius * self.radius

class Square(Shape):  # Child inherits from Shape
    def __init__(self, side):
        super().__init__("Square")
        self.side = side

    def area(self):  # Override parent method
        return self.side * self.side

# Both Circle and Square inherit 'name' attribute from Shape
circle = Circle(5)
square = Square(4)

print(circle.name)  # "Circle" (inherited from Shape)
print(square.name)  # "Square" (inherited from Shape)
```

OBJECT ORIENTED PROGRAMMING

Polymorphism:

- Using the same interface for different types
- Same method can behave differently for different objects

```
# Polymorphism
def print_area(shape):  # Takes any Shape
    print(f"{shape.name} area: {shape.area()}")

# Same method call, different behaviors
print_area(circle)  # "Circle area: 78.5"
print_area(square)  # "Square area: 16"

# Or with a list
shapes = [Circle(3), Square(5), Circle(2)]
for shape in shapes:
    print_area(shape)  # Same code, different results
```