Chapter 11: Inheritance & More on Object-Oriented Programming

Question 1:

What is inheritance in Python and how is it useful?

Answer: Inheritance is a way to create a new class (derived class) that inherits attributes and methods from an existing class (base class). It promotes code reuse and establishes a relationship between classes, making the code more modular and easier to maintain.

Example:

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

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

class Programmer(Employee):
    def __init__(self, name, language):
        super().__init__(name)
        self.language = language

    def display_info(self):
        super().display_info()
        print(f"Programming Language: {self.language}")

shivam = Programmer("Shivam", "Python")
shivam.display_info()
```

Output:

```
Employee Name: Shivam
Programming Language: Python
```

Question 2:

What are the types of inheritance in Python? Provide examples.

Answer: There are three main types of inheritance in Python: single inheritance, multiple inheritance, and multilevel inheritance.

1. Single Inheritance:

```python
class Animal:
    def sound(self):
        print("Some sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

dog = Dog()
dog.sound()  # Output: Bark
```

2. Multiple Inheritance:

```python
class Person:
    def info(self):
        print("I am a person")

class Worker:
    def work(self):
        print("I am working")

class WorkingPerson(Person, Worker):
    pass

shivam = WorkingPerson()
shivam.info()  # Output: I am a person
shivam.work()  # Output: I am working
```

3. Multilevel Inheritance:

```python
class Animal:
    def sound(self):
        print("Some sound")

class Mammal(Animal):
    def sound(self):
        print("Mammal sound")

class Dog(Mammal):
    def sound(self):
        print("Bark")

dog = Dog()
dog.sound()  # Output: Bark
```

Question 3:

What is the purpose of the `super()` method in Python inheritance?

Answer: The `super()` method is used to call a method from the parent class in the derived class. It is commonly used to initialize the parent class's attributes within the derived class.

Example:

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

class Programmer(Employee):
    def __init__(self, name, language):
        super().__init__(name)
        self.language = language

shivam = Programmer("Shivam", "Python")
print(f"{shivam.name} codes in {shivam.language}")  # Output: Shivam codes in Python
```

Question 4:

Explain class methods and how they differ from instance methods. Provide an example.

Answer: Class methods are methods that are bound to the class and not the instance of the class. They can access and modify class state that applies across all instances of the class. They are defined using the `@classmethod` decorator.

Example:

```python
class Employee:
    company = "Google"

    @classmethod
    def change_company(cls, new_company):
        cls.company = new_company

shivam = Employee()
print(Employee.company)  # Output: Google
Employee.change_company("YouTube")
print(Employee.company)  # Output: YouTube
```

Difference:

- **Instance Methods:** Operate on an instance of the class. They can access instance attributes and methods.
- **Class Methods:** Operate on the class itself. They can access class attributes and methods.

Question 5:

What is operator overloading in Python? Demonstrate with examples.

Answer: Operator overloading in Python allows us to define custom behavior for operators (like `+`, `-`, `*`, `/`) for user-defined classes. This is done by defining special methods (dunder methods).

Example:

```python
class Number:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return self.value + other.value

    def __sub__(self, other):
        return self.value - other.value

    def __mul__(self, other):
        return self.value * other.value

    def __truediv__(self, other):
        return self.value / other.value

num1 = Number(5)
num2 = Number(10)
print(num1 + num2)  # Output: 15
print(num1 - num2)  # Output: -5
print(num1 * num2)  # Output: 50
print(num1 / num2)  # Output: 0.5
```

Explanation:

- `__add__`: Overloads the `+` operator.
- `__sub__`: Overloads the `-` operator.
- `__mul__`: Overloads the `*` operator.
- `__truediv__`: Overloads the `/` operator.