Chapter 11: Inheritance & More on Object-Oriented Programming

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit attributes and methods from another class. This enables code reuse and establishes a relationship between classes.

Basic Syntax of Inheritance

```
class Employee:  # Base class
    # Code for Employee class

class Programmer(Employee):  # Derived or child class
    # Code for Programmer class
```

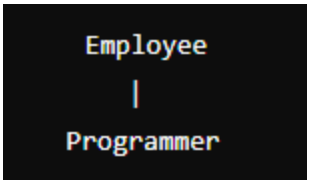
Types of Inheritance

1. Single Inheritance

Single inheritance occurs when a child class inherits only from one parent class.

Example:

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

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

class Programmer(Employee):
    def show_role(self):
        print(f"{self.name} is a Programmer")

shivam = Programmer("Shivam")
shivam.show_name()  # Output: Name: Shivam
shivam.show_role()  # Output: Shivam is a Programmer
```



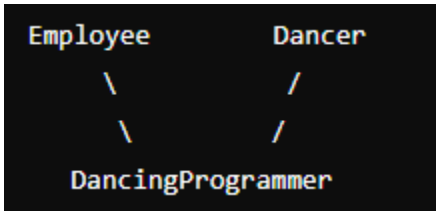
2. Multiple Inheritance

Multiple inheritance occurs when a child class inherits from more than one parent class.

Example:

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

class Dancer:
    def __init__(self, style):
        self.style = style

class DancingProgrammer(Employee, Dancer):
    def __init__(self, name, style):
        Employee.__init__(self, name)
        Dancer.__init__(self, style)

shivam = DancingProgrammer("Shivam", "Hip Hop")
print(f"{shivam.name} dances {shivam.style}")  # Output: Shivam dances Hip Hop
```

Diagram:

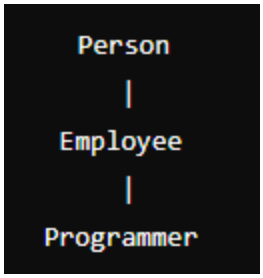


3. Multilevel Inheritance

Multilevel inheritance occurs when a child class becomes a parent for another child class.

Example:

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

class Employee(Person):
    def show_role(self):
        print(f"{self.name} is an Employee")

class Programmer(Employee):
    def show_language(self):
        print(f"{self.name} codes in Python")

shivam = Programmer("Shivam")
shivam.show_role()  # Output: Shivam is an Employee
shivam.show_language()  # Output: Shivam codes in Python
```



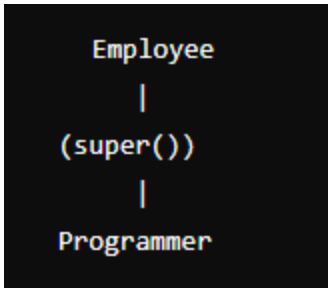
The `super()` Method

The `super()` method is used to call a method from the parent class in 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
```

Diagram:

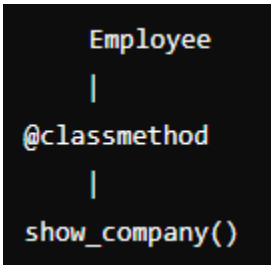


Class Methods

A class method is a method that is bound to the class and not the object of the class. It is created using the `@classmethod` decorator.

Example:

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

    @classmethod
    def show_company(cls):
        print(f"Company: {cls.company}")

Employee.show_company()  # Output: Company: Google
```

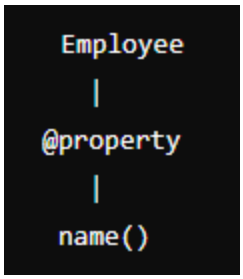


Property Decorators

Property decorators are used to define getter and setter methods in a class.

Example:

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

    @property
    def name(self):
        return self._ename

    @name.setter
    def name(self, value):
        self._ename = value

shivam = Employee("Shivam")
print(shivam.name)  # Output: Shivam
shivam.name = "Shivam Kumar"
print(shivam.name)  # Output: Shivam Kumar
```

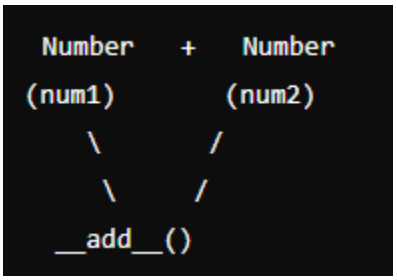


Operator Overloading

Operators in Python can be overloaded using special methods (dunder methods).

Example:

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

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

num1 = Number(5)
num2 = Number(10)
print(num1 + num2)  # Output: 15
```

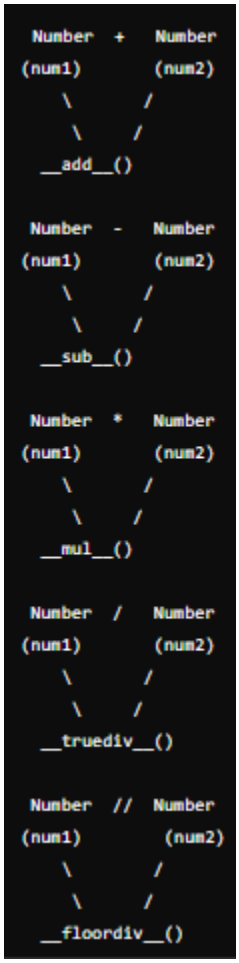


Operator Overloading

Operators in Python can be overloaded using special methods (dunder methods).

Example:

```
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

    def __floordiv__(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
print(num1 // num2)  # Output: 0
```



Example of `__str__()` and `__len__()`

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

    def __str__(self):
        return f"Employee name: {self.name}"

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

shivam = Employee("Shivam")
print(str(shivam))  # Output: Employee name: Shivam
print(len(shivam))  # Output: 6
```