Key Concepts: Abstract Classes and Methods

Definition and Importance

- **Abstract Class**:
  - A class that cannot be instantiated directly.
  - Serves as a blueprint for other classes (derived/child classes).
  - Guarantees that specific methods will be implemented in the child class.
- **Abstract Method**:
  - A method declared in the abstract class but without implementation.
  - Must be overridden in any concrete (non-abstract) child class.
- **Benefits**:
  - Ensures consistency and interoperability across derived classes.
  - Avoids code duplication by providing a unified interface.
  - Hides implementation details while maintaining functionality.

Key Characteristics

1. **Cannot Instantiate Abstract Classes**:
   - Example: You can't create an instance of a `Vehicle` class, but you can create a `Car` or `Boat` derived from `Vehicle`.
2. **Methods in Abstract Classes**:
   - Defined but not implemented.
   - Must be overridden in child classes.
3. **Implementation Possibilities**:
   - Directly implementing in derived classes.
   - Using `super()` for partial implementation.

Implementation in Python

Step 1: Import the ABC Module

- Python does not support abstraction directly, so you use the `ABC` (Abstract Base Class) module.

```
from abc import ABC, abstractmethod
```

Step 2: Define an Abstract Class

- Inherit the `ABC` class.
- Use the `@abstractmethod` decorator to define abstract methods.

```
class Employee(ABC):
    @abstractmethod
    def donate(self):
        pass
```

Step 3: Create a Derived Class

- Inherit from the abstract class.
- Override and implement all abstract methods.

```
class Donation(Employee):
    def donate(self):
        amount = int(input("Enter donation amount: "))
        return amount
```

Step 4: Instantiate and Use

- Create instances of the derived class, not the abstract class.
- Call overridden methods on these instances.

```
# Create instances
john = Donation()
peter = Donation()

# Collect donations
amounts = []
amounts.append(john.donate())
amounts.append(peter.donate())

# Print total donations
print("Total donations:", sum(amounts))
```

Advantages of Abstract Classes

1. **Consistency**:
    - Ensures derived classes follow a uniform structure.
    - Guarantees specific methods exist in all child classes.
2. **Reusability**:
    - Shared behaviors can be implemented in the parent abstract class.
    - Avoids code duplication.
3. **Modularity**:
    - Separates definition from implementation.
    - Makes code easier to maintain and extend.

Code Output Example

- **Input**:

```
Enter donation amount: 50
Enter donation amount: 75
```

- **Output**:

```
Total donations: 125
```

Conclusion

- Abstract classes and methods are essential for organizing code in large projects.
- They enforce a consistent structure across related classes.
- By learning abstract classes, you ensure better reusability, maintainability, and scalability in object-oriented design.