Chapter 9 [OOP]

Introduction to OOP

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to organize code into reusable and modular components. Unlike procedural programming, which relies on functions, OOP structures programs around objects that combine both data and behavior.

Python is an object-oriented language, which means it supports OOP concepts like classes, objects, inheritance, and polymorphism. Understanding these concepts will help you write more efficient, maintainable, and scalable code.

What are Classes and Objects?

- **Classes** are blueprints for creating objects. They define a set of attributes (data) and methods (functions) that the created objects will have. Think of a class as a template or a blueprint.
- **Objects** are instances of classes. When you create an object, you are creating an instance of a class with its own unique set of attributes and methods.

Defining Classes

To define a class in Python, use the `class` keyword followed by the class name (by convention, class names start with an uppercase letter):
This defines a class named `Car`. Currently, it's an empty class with no attributes or methods. Let’s add some attributes and methods to make it more useful.

```
class Car:
    pass  # A simple, empty class
```

Adding Attributes and Methods

Attributes are variables that belong to the class, while methods are functions that belong to the class.

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print(f"{self.year} {self.make} {self.model}")
```

Understanding the Code

- `__init__`: This is a special method called a constructor. It gets called automatically when an object is created from the class. It initializes the object’s attributes.
- `self`: This is a reference to the current instance of the class. It is used to access attributes and methods of the class. It must be the first parameter of any function in the class.

Creating Objects

Once you have defined a class, you can create objects (instances) of that class:

```
my_car = Car("Toyota", "Corolla", 2021)
another_car = Car("Honda", "Civic", 2020)
```

Here, `my_car` and `another_car` are objects of the class `Car`. Each object has its own set of attributes (`make`, `model`, `year`) initialized by the constructor.

Accessing Attributes and Methods

You can access the attributes and methods of an object using the dot notation:

```
print(my_car.make)  # Output: Toyota
my_car.display_info()  # Output: 2021 Toyota Corolla
```

Example: Building a Simple Banking System

Let's create a simple class to represent a bank account, demonstrating how to define attributes and methods.

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance is {self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds!")
        else:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance is {self.balance}")

# Creating an object of the BankAccount class
account = BankAccount("John Doe", 1000)
account.deposit(500)
account.withdraw(200)
```

Coding Questions to Practice OOP Concepts

1. **Create a Class:**
   Define a class named `Book` that has attributes like `title`, `author`, and `pages`. Write a method `display_info()` that prints out the book's details.
2. **Bank Account Simulation:**
   Extend the `BankAccount` class by adding a method `transfer()` that allows transferring money from one account to another.

3. **Basic Inheritance:**
   Create a base class `Animal` with attributes `name` and `species` and a method `make_sound()`. Create two derived classes `Dog` and `Cat` that override the `make_sound()` method with their respective sounds.
4. **Encapsulation:**
   Define a class `Student` that has private attributes for name and grade. Provide public methods to get and set the values of these attributes.
5. **Object Interaction:**
   Create a class `Library` with methods to `add_book()` and `remove_book()` using the `Book` class. Implement a method `list_books()` that prints all the books currently in the library.

Object-Oriented Programming is a powerful way to structure your code, making it more organized and easier to manage. By understanding how to define classes and create objects, you can model real-world entities and their interactions in your programs. Practice using classes and objects by solving the coding questions provided. As you get more comfortable with OOP, you'll find that it opens up new possibilities for writing clean, efficient, and scalable code.