Day 9: Abstraction and Encapsulation in Python

Topics Covered:

- *Abstraction*
- *Encapsulation*
- *Benefits of Abstraction and Encapsulation*
- *Python Implementation of Abstraction and Encapsulation*

1. Introduction to Abstraction:

Abstraction is a fundamental concept in object-oriented programming (OOP) that involves hiding the implementation details and exposing only the essential features of an object. It allows the user to interact with an object through a simplified interface without knowing how it works internally.

2. Abstraction in Python:

In Python, abstraction can be achieved using abstract classes and methods. An abstract class is a class that cannot be instantiated directly and serves as a blueprint for subclasses. It contains abstract methods that must be implemented by any subclass.

Example:

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        return "Woof!"

dog = Dog()
print(dog.make_sound())  # Output: Woof!
```

3. Introduction to Encapsulation:

Encapsulation is another key concept in OOP that involves bundling data and methods that operate on the data within a single unit called a class. It restricts direct access to some of an object's components, which can help protect the integrity of the data and prevent unauthorized access.

4. Encapsulation in Python:

In Python, encapsulation is implemented using private and public attributes. By convention, private attributes are prefixed with an underscore `_`, and public attributes can be accessed directly. However, private attributes are intended to be accessed only through getter and setter methods.

Example:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self._age = age  # Private attribute

    def get_age(self):  # Getter method
        return self._age

    def set_age(self, age):  # Setter method
        if age > 0:
            self._age = age
        else:
            print("Age must be positive.")

person = Person("Alice", 25)
print(person.get_age())  # Output: 25
person.set_age(30)
print(person.get_age())  # Output: 30
```

5. Benefits of Abstraction and Encapsulation:

Abstraction helps in reducing complexity by hiding unnecessary details, allowing developers to focus on the important aspects of the application. Encapsulation protects an object's internal state and ensures that the data is accessed only through defined methods, improving security and data integrity.

Summary of Day 9:

Today, we learned about Abstraction and Encapsulation, two core principles of object-oriented programming. We explored how abstraction simplifies complex systems by hiding implementation details, and how encapsulation helps protect data by restricting direct access to an object's components. We also saw how to implement these concepts in Python using abstract classes, getter and setter methods.