Object Oriented Programming : Core Principles, Advanced

Introduction

- **Programming using OOPs is underpinned by a set of Core Principles and following four a key**:
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

1 - Abstraction

- **Definition:**
  - Hiding implementation details while exposing only essential features.
- **How to Achieve Abstraction in Python?**
  - Use abstract base classes (`abc` module).
  - `abc.abstractmethod(function)` A decorator indicating abstract methods.

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

circle = Circle(5)
print(circle.area())   # Output: 78.5
```

2 - Encapsulation

- **Definition:**
  - Bundling data and methods within a class.
  - Restricts direct access to some attributes for security and integrity.
- **How to Achieve Encapsulation in Python:**
  - Use `_` for protected attributes.

    - Use `__` for private attributes.

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

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
print(account.get_balance())  # Output: 1500
```

3 - Inheritance

- **Definition:**
    - Mechanism to create a new class (child) from an existing class (parent).
    - Child class inherits attributes and methods from the parent class.
- **Benefits:**
    - Code reuse.
    - Establishes relationships between classes.

Example Code:

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

dog = Dog()
dog.speak()  # Output: Dog barks
```

4 - Polymorphism

- **Definition:**
    - Ability to use a single interface for different types of objects.
- **Types of Polymorphism in Python:**

    - Method Overriding (as seen in inheritance).
    - Operator Overloading.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

v1 = Vector(2, 3)
v2 = Vector(4, 5)
result = v1 + v2
print(result.x, result.y)  # Output: 6, 8
```

Abstraction

- **Definition:**
    - Hiding implementation details while exposing only essential features.
- **How to Achieve Abstraction in Python?**
    - Use abstract base classes (`abc` module).