Python Object-Oriented Programming (OOP) Concepts

Definition: Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects that contain both data (attributes) and methods (functions) to operate on that data.

Advantages of OOP:

1. Code reusability through inheritance.
2. Improved code maintainability and organization.
3. Encapsulation provides data security.
4. Polymorphism allows flexibility in code.
5. Abstraction hides implementation details.

Four Pillars of OOP:

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

Class and Object

Definition: A class is a blueprint for creating objects. An object is an instance of a class.

Real World Example: A class is like a blueprint for a house, and each house built from that blueprint is an object.

Code Example:
```
class Car: def __init__(self, brand, color): self.brand = brand
self.color = color my_car = Car("BMW", "Black") print(my_car.brand)
```

Encapsulation

Definition: Encapsulation means restricting direct access to variables and methods. It helps protect the data.

Real World Example: A bank account hides its balance details and allows controlled access through deposit/withdraw methods.

Code Example:
```
class Account: def __init__(self): self.__balance = 0 def deposit(self,
amount): self.__balance += amount def get_balance(self): return
self.__balance acc = Account() acc.deposit(1000) print(acc.get_balance())
```

Inheritance

Definition: Inheritance allows one class to acquire the properties and methods of another class.

Real World Example: A child inherits traits and behaviors from parents.

Code Example:
```
class Animal: def speak(self): print("Animal speaks") class Dog(Animal):
def speak(self): print("Dog barks") d = Dog() d.speak()
```

Polymorphism

Definition: Polymorphism means having many forms. It allows different classes to use the same method name but behave differently.

Real World Example: A 'sound()' method can produce different outputs for different animals.

Code Example:

```
class Bird: def sound(self): print("Chirp") class Dog: def sound(self):
print("Bark") for animal in [Bird(), Dog()]: animal.sound()
```

Abstraction

Definition: Abstraction means hiding the internal implementation and showing only the necessary functionality.

Real World Example: When you drive a car, you use the steering wheel without knowing how the engine works.

Code Example:

```
from abc import ABC, abstractmethod class Shape(ABC): @abstractmethod def
area(self): pass class Circle(Shape): def __init__(self, r): self.r = r
def area(self): return 3.14 * self.r * self.r c = Circle(5)
print(c.area())
```

Practice Questions:

1 1. What is the difference between class and object?
2 2. Explain encapsulation with a real-world example.
3 3. How is inheritance implemented in Python?
4 4. What is the use of polymorphism in OOP?
5 5. What is abstraction and why is it important?
6 6. Write a Python program to demonstrate multiple inheritance.