



2ND YEAR COMPUTER SCIENCE SOLVED EXERCISE

UNIT #03 — OBJECT ORINETED PROGRAMMING USING PYTHON

Prepared by: Mahmood Khan | MS Cyber Security
[Codymmentor Software House Sahiwal](#) | 03186879182

Multiple Choice Questions

1. The OOP concept that hides data inside a class is:

- (a) Inheritance
- (b) Polymorphism
- (c) Encapsulation
- (d) Abstraction

2. Classes and objects in Object-Oriented Programming are best described as:

- (a) Classes are real-world entities, and objects are blueprints.
- (b) Classes are blueprints, and objects are instances of those blueprints.
- (c) Classes are used for data storage, and objects are used for functionality.
- (d) Objects define functionality, and classes store data.

3. The main idea behind encapsulation in OOP is:

- (a) To hide data inside a class and prevent unauthorized access.
- (b) To create a flexible and adaptable program.
- (c) To reuse code from other classes.
- (d) To allow multiple inheritance in classes.

4. Inheritance in OOP means:

- (a) Reusing and extending functionality from a parent class to a child class.
- (b) Hiding data in a class.
- (c) Creating multiple instances of the same class.
- (d) Managing behavior of classes.

5. Polymorphism in OOP is best defined as:

- (a) Allowing an object to take multiple forms through method overriding.
- (b) Hiding data inside a class for security.
- (c) Using multiple inheritance to extend classes.
- (d) Ensuring that each class has only one method.

6. An example of inheritance in Python is:

- (a) Creating two classes with similar attributes.
- (b) A child class inheriting attributes and methods from a parent class.
- (c) Using private variables within a class.
- (d) Modifying a method inside the same class.

7. The purpose of polymorphism in programming is:

- (a) To create reusable functions with multiple parameter types.
- (b) To allow a function or method to have different implementations.
- (c) To combine different classes into a single class.
- (d) To manage the flow of control in a programme.

8. The keyword used to define a class in Python is:

- (a) def
- (b) class
- (c) function
- (d) object

9. Encapsulation is best described as:

- (a) Combining data and methods into a single unit and controlling access to data.
- (b) Combining multiple classes into one class.
- (c) Inheriting methods from other classes.
- (d) Allowing objects to have multiple forms.

10. Inheritance allows a class to:

- (a) Hide its data only.
- (b) Inherit attributes and methods from another class.
- (c) Define multiple functions only.
- (d) Create objects without a class.

MCQ'S Answer Key

Q. No	1	2	3	4	5	6	7	8	9	10
Answer	C	B	A	A	A	B	B	B	A	B

Short Questions

1. Explain the concept of classes and objects in Object-Oriented Programming.

Answer: A class is a blueprint or template that defines the properties (attributes) and actions (methods) of an entity. An object is an instance of a class that holds actual values for its attributes.

A class is like a blueprint, while an object is the real thing created from that blueprint. Classes and objects allow programmers to model real-world entities in code.

2. What is encapsulation in OOP?

Answer: Encapsulation is a concept in Object Oriented Programming (OOP) where we bundle data (variables) and methods (functions) into a single unit (class) and restrict direct access to the data.

3. How does inheritance work in OOP?

Inheritance allows a class to take properties and methods from another class. The new class, called a subclass, can add or modify the features of the parent class.

Inheritance promotes code reuse and reduces code redundancy. It also makes it easier to extend programs without rewriting existing code.

There are the following types of inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance

4. What are the advantages of using inheritance in Python?

Answer: The advantages of using inheritance in Python are:

- **Code Reuse:** It allows a class to reuse properties and methods of another class.
- **Reduces Code Redundancy:** It avoids writing the same code repeatedly.
- **Easy to Extend:** New features can be added without rewriting existing code.
- **Easy to Maintain:** Changes in the parent class can be reflected in child classes.

5. What is polymorphism in OOP, and how does it enhance flexibility in code?

Answer: Polymorphism means “many forms.” It refers to the ability of a function, method, or operator to perform different actions depending on the object or context.

Polymorphism makes code more flexible and reusable by allowing the same method, function, or operator to behave differently for different objects.

6. How can encapsulation help improve data security in OOP?

Answer: Encapsulation is the process of combining data and methods into a single class and controlling access to that data.

It can improve data security by:

- **Data Hiding:** It prevents direct access to sensitive data.
- **Controlled Access:** Data can be accessed or modified through specific methods.
- **Data Protection:** It helps prevent unauthorized changes to important information.
- **Better Security:** It reduces the risk of accidental or unwanted modification of data.

Example: A bank account class can keep the balance private and allow changes only through methods such as deposit() and withdraw()

7. How does the combination of inheritance, encapsulation, and polymorphism contribute to better software design?

Answer: The combination of inheritance, encapsulation, and polymorphism helps create software that is organized, reusable, flexible, and secure. Inheritance promotes code reuse and reduces code duplication, while encapsulation protects data and controls access to it. Polymorphism allows the same method to behave differently for different objects, making the program more flexible. Together, these concepts make software easier to develop, maintain, modify, and extend.

8. What are the key differences between a class and an object in Object-Oriented Programming?

Answer: A class is a blueprint or template that defines the attributes and methods of an entity, while an object is an instance of a class that contains actual values for those attributes.

A class defines the structure and behavior, whereas an object represents a specific entity created from that class.

For example, **Car** can be a class, while a specific car such as a **red Toyota** can be an object of that class.

Long Questions

1. Explain the concept of classes and objects in Object-Oriented Programming (OOP).

Object-Oriented Programming (OOP) is a method of designing programs using classes and objects. Each object represents a real-world entity with specific properties and behaviors. A class is a blueprint or template that defines the properties (attributes) and actions (methods) of an entity. An object is an instance of a class that holds actual values for its attributes.

For example, a Student class can define attributes such as name and age and a method such as display(). Different student objects can then be created from this class, with each object having its own data, such as Ali or Ahmed.

Classes and objects help programmers model real-world entities in code. They also organize data and related functionality together, making programs more structured, reusable, and easier to manage. A class defines what an object will contain and do, while the object represents a specific instance of that class.

Example:

class Student:

```
def __init__(self, name):  
    self.name = name
```

```
def display(self):  
    print("Student Name:", self.name)
```

```
object1 = Student("Ali")
```

```
object2 = Student("Ahmed")
```

Here, Student is the class, while object1 and object2 are objects of the Student class.

Q.NO 2) What is encapsulation in OOP?

Answer: Encapsulation is a concept in Object Oriented Programming (OOP) where we bundle data (variables) and methods (functions) into a single unit (class) and restrict direct access to the data. Unlike languages like Java or C++, Python does not have strict "private" or "protected" keywords. Instead, it relies on naming conventions to signal how data should be accessed.

Why Encapsulation is important

- Protects data from unwanted access
- Improves security
- Makes code easier to manage
- Allows controlled modification

For example, think of a **bank account** where the balance is hidden. You can only access or modify the balance using methods such as `deposit()` and `withdraw()`. You cannot directly change the balance. This controlled access to data is called **encapsulation**.

In Python, the visibility of class members can be controlled using specific naming prefixes. **Public members** are accessible from anywhere, both inside and outside the class. By default, all members in Python are public. The syntax for a public member is `self.attribute_name`.

Protected members are intended for internal use within the class and its subclasses. They use a **single underscore**, such as `self._attribute_name`. **Private members** are more restricted and cannot be easily accessed from outside the class. Python uses a mechanism called **name mangling** to change the internal name of a private variable and help prevent accidental access. Private members use a **double underscore**, such as `self.__attribute_name`.

Example:

```
class BankAccount:
```

```
    def __init__(self, balance):
```

```
        self.__balance = balance  # Private variable
```

```
    def deposit(self, amount):
```

```
        self.__balance += amount
```

```
    def withdraw(self, amount):
```

```
        if amount <= self.__balance:
```

```
            self.__balance -= amount
```

```
        else:
```

```
            print("Insufficient balance")
```

```
    def get_balance(self):
```

```
        return self.__balance
```

```
# Usage
```

```
account = BankAccount(1000)
```

```
account.deposit(500)
```

```
print(account.get_balance())
```

```
# Output: 1500
```

```
print(account.__balance)
```

```
# Error: Cannot access private variable directly
```

Q.NO 3) How does inheritance work in OOP?

Answer: Inheritance is an OOP concept that allows a class to take properties and methods from another class. The class whose properties and methods are inherited is called the **parent class (base or superclass)**, while the class that inherits them is called the **child class (derived or subclass)**.

A child class can use the existing functionality of its parent class and can also add new features or modify existing methods. This allows programmers to reuse existing code instead of writing it again. A child class is defined by passing the parent class name in parentheses.

For example:

```
class Animal:
```

```
    def eat(self):
```

```
        print("Animal is eating.")
```

```
class Dog(Animal):
```

```
    def bark(self):
```

```
        print("Dog says Woof!")
```

Here, Animal is the parent class and Dog is the child class. The Dog class can use the eat() method inherited from Animal as well as its own bark() method.

A child class can also **override** a method of its parent class by providing its own implementation. Python also supports different forms of inheritance, including **single inheritance, multiple inheritance, and multilevel inheritance**.

Inheritance therefore allows programs to **reuse, extend, and modify existing functionality** efficiently.

Q.NO 4) What are the advantages of using inheritance in Python?

Answer: Inheritance provides several important advantages in Python. It allows a child class to **reuse** the properties and methods of a parent class, which reduces repeated code and promotes code reusability. It also reduces code redundancy because common functionality can be defined once in the parent class and reused by different child classes.

Inheritance improves **modularity** by organizing related functionality into different classes. It also makes programs easier to extend because a child class can add new features or modify existing functionality without rewriting the complete code. A child class can also change the behavior of an inherited method using method overriding.

Another advantage is **easier maintenance**. When common functionality is placed in a parent class, changes can be made in one place instead of modifying the same code in multiple classes. Inheritance also supports the scalability and adaptability of software, making it easier to develop larger and more structured programs.

Therefore, the main advantages of inheritance in Python are code reusability, reduced code redundancy, modularity, easier maintenance, easier extension, scalability, and adaptability.

Q.NO 5) What is polymorphism in OOP, and how does it enhance flexibility in code?

Answer: Polymorphism means “**many forms.**” In OOP, it refers to the ability of the same method, function, or operator to behave differently depending on the object or context in which it is used. This makes code more **flexible and reusable**.

For example, different payment methods such as Credit Card, UPI, NetBanking, and Wallet can use the same method name, such as `processPayment()`, but each payment type can perform a different action. Similarly, a `sound()` method can produce different results for different objects such as Dog and Cat.

Python supports polymorphic behavior through techniques such as **method overriding**. In runtime polymorphism, a child class provides its own version of a method that already

exists in the parent class. The appropriate method is selected at runtime according to the object.

For example:

```
class Animal:
```

```
    def sound(self):
```

```
        return "Some generic sound"
```

```
class Dog(Animal):
```

```
    def sound(self):
```

```
        return "Bark"
```

```
class Cat(Animal):
```

```
    def sound(self):
```

```
        return "Meow"
```

Here, the same **sound()** method produces different results for Dog and Cat objects.

Thus, polymorphism enhances flexibility because the **same interface or method can be used with different objects while allowing each object to perform its appropriate behavior**. This makes programs more reusable, adaptable, and easier to extend.

Q.NO 6) How can encapsulation help improve data security in OOP?

Encapsulation helps improve data security by **bundling data and methods into a single class and restricting direct access to the data**. It prevents unwanted or accidental modification of important information and allows data to be accessed or changed through controlled methods.

In Python, class members can have different levels of access. **Public members** can be accessed from anywhere. **Protected members**, written with a single underscore such as `self._balance`, are intended for internal use. **Private members**, written with a double

underscore such as `self.__balance`, are more restricted and use Python's name-mangling mechanism.

A good example is a **bank account**. The account balance should not be changed directly by anyone. Instead, methods such as `deposit()`, `withdraw()`, and `get_balance()` can be used to control access to the balance.

For example:

```
class BankAccount:
```

```
    def __init__(self, balance):
```

```
        self.__balance = balance
```

```
    def deposit(self, amount):
```

```
        self.__balance += amount
```

```
    def get_balance(self):
```

```
        return self.__balance
```

In this example, the balance is protected from direct access and can be managed through specific methods. Therefore, encapsulation **protects data from unwanted access, improves security, provides controlled modification, and makes code easier to manage**.

Q.NO 7)How does the combination of inheritance, encapsulation, and polymorphism contribute to better software design?

The combination of **inheritance, encapsulation, and polymorphism** helps create software that is structured, reusable, flexible, and easier to maintain. These concepts work together to manage both data and behavior efficiently.

Inheritance promotes code reuse by allowing a child class to inherit properties and methods from a parent class. It reduces code redundancy and makes it easier to extend an existing program without rewriting its code.

Encapsulation protects data by combining data and methods within a class and controlling access to important information. It improves security and prevents unwanted or accidental changes to data.

Polymorphism provides flexibility by allowing the same method, function, or operation to behave differently depending on the object or context. For example, the same sound() method can produce different results for Dog and Cat objects.

When these three concepts are used together, programs become more **modular, reusable, secure, flexible, scalable, and maintainable**. They also make it easier to modify or extend a program as requirements change. Therefore, these OOP concepts provide a strong foundation for designing well-structured and efficient software.

Q.NO 8) What are the key differences between a class and an object in Object-Oriented Programming?

A **class** and an **object** are closely related concepts in OOP, but they have different purposes. A class is a **blueprint or template** that defines the properties (attributes) and actions (methods) of an entity. An object is an **instance of a class** that contains actual values for those attributes.

A class describes what an object will contain and what it can do, while an object represents a specific entity created from that class. For example, Student can be a class that defines the attributes name and age and a method display(). Ali and Ahmed can be two different objects created from the Student class.

Each object created from the same class can have **different data**, even though they share the same structure and methods.

Class

A class is a blueprint or template.

It defines attributes and methods.

It describes the structure and behavior.

One class can be used to create many objects.

Object

An object is an instance of a class.

It contains actual values for attributes.

It represents a specific entity.

Each object has its own data.

For example:

```
class Student:
```

```
    def __init__(self, name):
```

```
        self.name = name
```

```
student1 = Student("Ali")
```

```
student2 = Student("Ahmed")
```

Here, Student is the **class**, while student1 and student2 are **objects**. Thus, a class provides the blueprint, whereas objects are the actual instances created from that blueprint.