Computer Science 12th — Chapter 3: Object-Oriented Programming (OOP) Using Python

Complete Short & Long Question Bank • Easy English (Textbook Exercise + Additional Questions) • istudy.com

PART 1: OFFICIAL TEXTBOOK EXERCISE QUESTIONS (SHORT & LONG)

A. TEXTBOOK SHORT QUESTIONS (ALL 10 QUESTIONS)

Q1: What is Object-Oriented Programming (OOP)?
Answer: OOP is a method of programming where software is designed using classes and real-world entities called objects.

Q2: What is the purpose of a class in Python?
Answer: A class acts as a blueprint or template to define properties (attributes) and actions (methods) for creating multiple objects.

Q3: What is the difference between a class and an object?
Answer: A **class** is the blueprint/design, while an **object** is the real instance created from that class containing actual values (e.g., Class = Car, Object = Tesla).

Q4: What is the purpose of __init__()?
Answer: `__init__()` is the constructor method in Python. It automatically runs when a new object is created to initialize its variables.

Q5: What is encapsulation in OOP, and why is it important?
Answer: Encapsulation bundles data (variables) and functions (methods) inside one unit (class) and restricts direct outside access to protect data from unwanted changes.

Q6: How does encapsulation help in keeping data secure and private?
Answer: It hides variables using private naming (like `__balance`) so they can only be viewed or modified through controlled methods like `deposit()` and `withdraw()`.

Q7: What is inheritance in OOP, and how does it help in reusing functionality?
Answer: Inheritance allows a child class to take attributes and methods from an existing parent class, saving time by avoiding rewriting old code.

Q8: How does inheritance enable the extension of functionality in Python?
Answer: A child class keeps all features of its parent class and can also add its own new methods or modify existing ones without altering the parent class.

Q9: What is polymorphism in OOP, and how does it make code more flexible?
Answer: Polymorphism means "many forms". It allows the same method name to perform different actions depending on which object is calling it.

Q10: How does polymorphism help in creating adaptable and reusable code?
Answer: It allows a single function to process different types of objects dynamically at runtime without needing separate code for each class.

B. TEXTBOOK LONG QUESTIONS (ALL 8 QUESTIONS)

Long Q1: Explain the concept of classes and objects in Object-Oriented Programming (OOP).

Classes: A class is a template defining attributes (variables like name, age) and methods (functions like eat(), sleep()).

Objects: An object is a real instance created from that template with actual values (e.g., `student1 = Student("Ali")`). Objects bundle state and behavior together.

Long Q2: What is encapsulation in OOP?

Definition: Encapsulation bundles data variables and functions into a single class unit and hides direct data access.

Access Levels in Python:

- **Public Members:** `self.name` (accessible from anywhere inside or outside the class).
- **Protected Members:** `self._name` (single underscore; convention for class and subclass use).
- **Private Members:** `self.__name` (double underscore; restricted from outside using Name Mangling).

Long Q3: How does inheritance work in OOP?

Mechanism: A child class (subclass) inherits attributes and methods from a parent class (superclass) using syntax `class Child(Parent):`. Child objects can access parent methods directly or override them.

Long Q4: What are the advantages of using inheritance in Python?

- **Code Reusability:** Eliminates duplicate code by sharing parent methods.
- **Extensibility:** Easy to add new features without breaking existing code.
- **Clean Architecture:** Organizes software into clean real-world hierarchies (e.g., Vehicle → Car → ElectricCar).

Long Q5: What is polymorphism in OOP, and how does it enhance flexibility in code?

Concept: Enables uniform treatment of different classes. For example, a base class `Animal` has a method `sound()`; child class Dog returns "Bark" while `Cat` returns "Meow". Calling `animal.sound()` dynamically runs the correct method at runtime.

Long Q6: How can encapsulation help improve data security in OOP?

By making attributes private (`__balance`), external programs cannot directly corrupt data (e.g., setting balance to negative). Data can only be modified through validated methods like `deposit()` and `withdraw()`.

Long Q7: How does the combination of inheritance, encapsulation, and polymorphism contribute to better software design?

Modularity: Encapsulation keeps code clean and secure. **Reusability:** Inheritance saves development time. **Flexibility:** Polymorphism makes large systems easy to maintain, scale, and update.

Long Q8: What are the key differences between a class and an object in OOP?

Feature	Class	Object
Nature	Abstract blueprint / template.	Real physical instance created from class.
Memory	No memory allocated for data when defined.	Allocates physical memory when created.
Syntax	Defined using `class` keyword.	Created using class call `obj = MyClass()`.

PART 2: ADDITIONAL HIGH-YIELD EXAM QUESTIONS (TOPIC-BY-TOPIC)

A. ADDITIONAL SHORT QUESTIONS

Topic 3.1: Python OOP

Q1: What is the purpose of the 'self' keyword in Python?

Answer: `self` represents the current instance/object of the class and allows instance methods to access its own attributes and variables.

Topic 3.2: Encapsulation

Q2: What is Name Mangling in Python?

Answer: Name mangling is Python's internal mechanism that changes the name of a double underscore variable (e.g., `__var` becomes `_ClassName__var`) to prevent accidental outside access.

Topic 3.2: Inheritance Types

Q3: What is Single Inheritance?

Answer: Single inheritance occurs when a child class inherits properties and methods from exactly one parent class.

Topic 3.2: Inheritance Types

Q4: What is Multiple Inheritance?

Answer: Multiple inheritance occurs when a single child class inherits features from more than one parent class (e.g., `class Child(Father, Mother):`).

Topic 3.2: Inheritance Types

Q5: What is Multilevel Inheritance?

Answer: Multilevel inheritance occurs when a class inherits from another derived class, forming a family chain (Grandparent → Parent → Child).

Topic 3.2: Method Overriding

Q6: What is Method Overriding?

Answer: Method overriding occurs when a child class defines a method with the same name as a method in its parent class to provide its own specific behavior.

Topic 3.2: Built-in Functions

Q7: What is the purpose of the super() function?

Answer: `super()` is used inside a child class to call and run methods directly from its parent class.

Topic 3.2: Polymorphism

Q8: Does Python support true compile-time method overloading? Why?

Answer: No, because Python is dynamically typed. Instead, similar behavior is achieved using default arguments or variable-length arguments (`*args`, `**kwargs`).

Topic 3.2: Polymorphism

Q9: What is Runtime Polymorphism in Python?

Answer: It means the method to execute is decided while the program is running based on the object calling it, achieved through method overriding.

Topic 3.1: Modularity

Q10: How does OOP improve code maintenance?

Answer: OOP divides code into self-contained classes. Changes made inside one class do not break other independent parts of the application.

B. ADDITIONAL LONG QUESTIONS

Additional Long Q1: Explain Types of Inheritance in Python (Single, Multiple, Multilevel) with Easy Examples.

- **Single Inheritance:** A child class inherits from one parent class (e.g., `class Dog(Animal):`).
- **Multiple Inheritance:** A child class inherits from two or more parent classes (e.g., `class Smartphone(Camera, Computer):`).
- **Multilevel Inheritance:** A class inherits through a chain of classes (e.g., `class ElectricCar(Car):` where `Car` inherits from `Vehicle`).

Additional Long Q2: Explain Compile-time vs Runtime Polymorphism in Python with Real-World Concepts.

Feature	Compile-time Polymorphism	Runtime Polymorphism
How It Works	Decided before execution (method/operator overloading).	Decided during execution based on calling object.
Python Support	Simulated via default args (`*args`).	Supported natively via Method Overriding.
Real-World Example	A calculator function multiplying 2 or 3 numbers.	Payment system: CreditCard and PayPal sharing `pay()`.