Chapter 1

OOPS

1. What is **Object-Oriented Programming (OOP)**
   Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which can contain data and code.

   Think of it as a way to organize your code to mirror the real world. Instead of writing a long list of instructions (procedural programming), you create "blueprints" for things.

   To understand OOP, you must understand the relationship between a **Class** and an **Object**.

   Class: The "blueprint" or "template." It defines what something *is* and what it can *do.*

   Example: A blueprint for a "Car.

   Object: The "instance." It is the actual thing built from the blueprint.

   Example: Your specific red Toyota Corolla with the license plate “XYZ-123”

2. **The Four Pillars of OOP**
   OOP is built on four main principles that help make code reusable, organized, and easy to maintain.

3. **Constructors**
   Used to initialize objects. parameterized/non-parameterized

```
class User:

    def __init__(self, name):

            self.name = name
```

4. **Access Modifiers (Python Style)**

Type	Meaning
Public	name
Protected	_name

Private	__name

```
class A:
  def __init__(self):
     self.name = "public"
     self._age = 20
     self.__salary = 5000
```

5. **Polymorphism** (Flexibility) [Same function behaves differently]
   This allows different classes to be treated as instances of the same general class through the same interface. The word means "many shapes."
   Analogy: If you tell a "Dog," a "Cat," and a "Cow" to "speak," the Dog barks, the Cat meows, and the Cow moos. They all perform the action "Speak," but in their own way.

```
class Dog:
   def speak(self):
      return “Bark"
class Cat:
   def speak(self):
      return "Meow"
```

6. **Abstraction** (Simplicity) [Hiding implementation]

   Abstraction hides complex implementation details and only shows the necessary features of an object.
   Analogy: When you use a remote control, you press "Power." you don’t care about the infrared signals or the circuit board logic behind that button.

```
```

```
from abc import ABC, abstractmethod
class Payment(ABC):
   @abstractmethod
   def pay(self): pass
```

7. **Encapsulation** (Security) [Wrapping data + methods]

This is the practice of bundling data (variables) and methods (functions) into a single unit (a class) and **restricting access** to the inner workings.
Analogy: You can drive a car by using the steering wheel and pedals without needing to understand how the fuel injection system works under the hood.

```

```
class Bank:

  def __init__(self):

    self.__balance = 1000
```

8. **Inheritance**

```
class A:
    pass
class B(A):
    pass
```

9. **Difference between Abstraction and Encapsulation**

10. **Multiple vs single inheritance with ex**

11. Why Use OOP?

As a software engineer, you use OOP because it solves several common problems:

DRY (Don't Repeat Yourself): Through inheritance, you write code once and reuse it.

Modularity: If there’s a bug in the "Parking Lot" payment logic, you know exactly which class to look at without breaking the "Gate" logic.

Scalability: It is much easier to manage 100,000 lines of code when they are organized into distinct objects than when they are in one giant script.
```

Algorithms & Data Structures (DSA)

12. Binary search

13. Linked List and detect cycle and list

14. Kadane`s

```
max_sum = curr = arr[0]

for i in arr[1:]:

  curr = max(i, curr+i)

  max_sum = max(max_sum, curr)
```

15. Find duplicate occurrences

```
from collections import Counter
arr = [1, 2, 2, 3, 1, 4, 2, 1, 3]

# Create a Counter object from the list 'arr'
count_result = Counter(arr)

# Print the result

print(count_result)


Output:

Counter({1: 3, 2: 3, 3: 2, 4: 1})
```

Low-Level Design (LLD) & Problem Solving

1. Parking lot problem
    1. 3 floors, car 100 and bike 60 per hour, ticketing system, employee
    2. LLD, DB queries, GROUP BY, JOINS, APIs

3. total sum(cache)
4. How many cars left

Database

1. Normalization
   Remove duplicate data. Example: Instead of storing customer name multiple times → store customer_id
2. DB - PG [Expected]

Frontend

1. F8850
2. Vue js

Behavioral & Professional Background

3. **Tell me about yourself**

System Architecture & Project Walkthroughs

4. **Past project experience**

5. **Explain recent project architecture**
6. **Explain about your Python projects**

Agile

What is Agile

Agile is an iterative software development methodology where work is delivered in small cycles called sprints.

Agile focuses on iterative delivery, customer feedback, and continuous improvement.

Why used?

Fast delivery

Continuous feedback

Easy to adapt to change

Example

Sprint (2 weeks):

Day 1 → planning

Daily → stand-up

End → demo + retrospective

GIT

1. Git commands

2. How to handle merge conflicts
3. Diff Git fetch and git pull
4. What is git and version control system.
5. Git stash

===== ===== ===== ===== ===== ===== ===== ===== ===== ===== ===== =====

Expected

1. Design
2. ORM [Expected]
3. Redis [Expected]
4. API - Rest API/ GraphQL [Expected]
5. PySpark [Expected]
6. Distributed Systems [Expected]