General:-
1. How your day goes from starting to end. (in project).

1 Python:

1. Python Intro & Road Map

1. What are the key features of Python, and why is it widely used?
2. How is Python different from other programming languages like Java or C++?
3. Explain the Python interpreter and how Python code is executed.
4. What are Python's strengths in data science, machine learning, and web development?
5. Describe the key steps in the Python learning roadmap for a beginner.

2. Basic Python Syntax & Programming

6. What are the basic syntax rules in Python?
7. Explain how indentation works in Python and why it's important.
8. How are comments written in Python? Why are they useful?
9. What is the difference between `print()` in Python 2.x and Python 3.x?
10. What is the purpose of the `__name__ == "__main__"` block?

3. Python Data Types

11. Explain the difference between mutable and immutable data types in Python.
12. What are lists, tuples, and dictionaries? How are they different?
13. How do sets work in Python, and when would you use them?
14. How do you convert a list to a tuple and vice versa?
15. What are the differences between Python strings and lists?

4. Conditionals and Type Conversion

16. How do `if-else` statements work in Python?
17. What are ternary operators in Python?
18. How do you handle type conversion in Python (implicit and explicit)?
19. What is the difference between `is` and `==` in Python?
20. Explain short-circuit evaluation in conditionals.

5. Functions and Built-in Functions

21. How do you define a function in Python? Provide an example.
22. What are the different ways to pass arguments to a function (positional, keyword, etc.)?
23. Explain how `*args` and `**kwargs` work.
24. What are Python's key built-in functions? Explain `map()`, `filter()`, and `reduce()`.
25. What is a lambda function? Provide a use case.

6. Slicing and Comprehensions

26. What is list slicing in Python? How does it work?
27. How do you reverse a list using slicing?
28. What is list comprehension? Provide an example.
29. How can you use comprehensions with dictionaries?
30. How do you create a generator in Python?

7. Variables & Scope

31. What are the different types of variable scopes in Python (local, global, nonlocal)?
32. What is the `nonlocal` keyword in Python? Provide an example.
33. Explain the use of the `global` keyword in Python.
34. How does Python handle variable shadowing?
35. What are closures, and how do they relate to variable scope?

8. OOPs Concepts in Python

36. What are the core principles of Object-Oriented Programming (OOP)?
37. How do you create a class in Python? Provide an example.
38. What is inheritance in Python? Explain with an example.
39. How does the `super()` function work in Python?
40. What is the difference between class methods and instance methods?

9. Python Modules

41. What are modules in Python, and how do you import them?
42. What is the difference between `import` and `from module import`?
43. How do you create your own module in Python?

44. What is a package in Python, and how does it differ from a module?
45. How do you handle circular imports in Python?

10. Python Garbage Collection

46. How does Python's garbage collection system work?
47. What is reference counting in Python?
48. Explain how cyclic references are handled in Python.
49. How can you manually trigger garbage collection?
50. What are weak references, and how are they used in Python?

11. Exceptions & Custom Exceptions

51. What is the difference between an error and an exception in Python?
52. How do you handle exceptions in Python? Explain the `try-except` block.
53. What are custom exceptions, and how do you create one?
54. What is the `finally` block, and when is it executed?
55. How do you use the `with` statement for resource management?

12. Python-Lambda & Decorators

56. What is a lambda function in Python, and how is it different from a regular function?
57. Provide a use case for lambda functions in Python.
58. What are decorators in Python, and why are they used?
59. How do you create a decorator with arguments?
60. Explain how multiple decorators can be applied to a single function.

13. Property & Context Manager

61. What are properties in Python, and how are they different from regular attributes?
62. How do you implement getter and setter methods using the `property()` function?
63. Explain how context managers work in Python. What is the purpose of the `with` statement?
64. How do you create a custom context manager using the `__enter__` and `__exit__` methods?

14. Parallel Processing & GIL

65. What is the Global Interpreter Lock (GIL) in Python, and how does it affect parallel processing?

66. How can you bypass the GIL for CPU-bound tasks in Python?
67. Explain the difference between multithreading and multiprocessing in Python.
68. What is the `concurrent.futures` module, and how does it simplify parallel processing?

15. Abstract Classes

69. What is an abstract class in Python? How do you define one using the `abc` module?
70. What is the purpose of abstract classes, and how do they enforce design in Python?

2. Django:

---

1. Introduction to Django

1. What is Django, and why is it used for web development?
2. How does Django's "batteries-included" philosophy help in rapid development?
3. Explain the MTV architecture in Django and how it compares to the MVC pattern.

2. Django Basics

4. How do you set up a Django project from scratch?
5. What is the significance of the `manage.py` file in Django?
6. Explain the difference between a project and an app in Django.
7. What is the purpose of `settings.py`, and what are some key configurations?

3. Creating the First Django Application

8. Walk through the steps to create your first Django app.
9. How do you map URLs to views in Django using `urls.py`?
10. How do you start a development server using Django?

4. Views and URL Routing

11. What are Django views, and what is the difference between function-based views (FBVs) and class-based views (CBVs)?

12. How do you use URL routing in Django, and what is the purpose of `path()` and `re_path()`?

5. Rendering Templates and Static HTML

13. What is the role of templates in Django, and how do you render a template in a view?
14. How do you manage static files (CSS, JavaScript) in Django?
15. Explain the use of the `render()` function in Django.

6. Advanced Classes and Concepts

16. What is an **abstract class** in Django models, and when would you use it?
17. Explain the **Meta class** in Django models. What are some common options defined in `Meta`?
18. What are **serializers**, and how are they used in Django Rest Framework (DRF)?

7. Class Method, Property Methods & Django Signals

19. What is a **class method** in Django, and how do you implement it?
20. How do **property methods** work in Django models?
21. What are **Django signals**, and when would you use them? Provide an example.

8. Context and Context Manager

22. What is **context** in Django templates, and how do you pass data to templates?
23. What is a **context manager** in Python, and how can you use it in Django?

9. Model Manager & Nested ORM Queries

24. What is a **Model Manager** in Django, and how do you customize it?
25. Explain how to create and execute **nested ORM queries** in Django.
26. What is **DB-locking**, and how is it managed in Django's ORM?

10. Middleware

27. What is middleware in Django, and how does it interact with requests and responses?
28. How do you create and add custom middleware in a Django project?

11. Working with Django Models

29. How do you define models in Django, and what are fields?
30. How do you query models in Django using the ORM?

31. What are model relationships (OneToOne, ForeignKey, ManyToMany), and how do you implement them?
32. How do you ensure **data integrity** in Django models?

12. Forms and User Input

33. How do you create and handle forms in Django?
34. What is the purpose of `forms.py` in Django, and how do you validate form data?
35. How does Django handle **user input** and protect against common vulnerabilities?

13. User Authentication

36. How does Django's **authentication system** work?
37. How do you create custom user models in Django?
38. What is the purpose of `django.contrib.auth`?

14. Django ORM and Admin Panel

39. What is the Django **ORM**, and how does it abstract database operations?
40. How do you customize the Django **Admin Panel**?
41. How do you implement **aggregate functions** in Django using ORM?
42. What are **annotations** in Django ORM, and how are they useful in queries?

15. Unit Testing and Pytest

43. How do you write **unit tests** for Django models and views?
44. What is **pytest**, and how does it differ from Django's default testing framework?
45. How do you integrate **pytest with Django**?

16. Django Rest Framework (DRF)

46. What is the **Django Rest Framework** (DRF), and why is it used?
47. What is an **APIView** in DRF?
48. What are **viewsets**, and how do they simplify view handling in DRF?
49. How do you create and validate **serializers** in DRF?
50. What is **HyperlinkedModelSerializer** in DRF, and how does it differ from the regular serializer?

17. ASGI Server

51. What is **ASGI**, and how does it differ from WSGI in Django?

52. How do you deploy Django with an **ASGI server** for handling asynchronous requests?

18. Model Relationships & Data Integrity

53. How do you implement **OneToOne**, **ForeignKey**, and **ManyToMany** relationships in Django?
54. How does Django ensure **referential integrity** in model relationships?
55. How do you handle **on_delete** options in Django?

19. Aggregation, Annotation & Nested ORM Queries

56. How do you perform **aggregations** in Django ORM?
57. What is **annotating** in Django ORM, and how do you use it?
58. How do you perform complex queries with **nested ORM queries**?

20. Django Signals and Context Manager

59. How do **Django signals** work, and what is a use case for them?
60. How do you implement a custom **context manager** in Django?

3. DataBase:

1. Basics of DBMS

1. What is a Database Management System (DBMS)?
2. Explain the difference between a database and a DBMS.
3. What are the advantages of using a DBMS?
4. What are the different types of DBMS models?
5. What is the significance of data independence in DBMS?

2. 3 Tier Architecture

6. Explain the 3-tier architecture in database systems.
7. What are the components of a 3-tier architecture?
8. How does the 3-tier architecture improve scalability and maintenance?
9. Can you describe the roles of the presentation, application, and database layers?
10. What are the advantages of using a 3-tier architecture over a 2-tier architecture?

3. Types of Keys

11. What is a primary key, and why is it important?
12. Explain the concept of a foreign key.
13. What is a composite key, and when would you use it?
14. What are surrogate keys, and how do they differ from natural keys?
15. Define candidate keys and their significance in a database.

4. RDBMS

16. What is a Relational Database Management System (RDBMS)?
17. What are the main features that distinguish RDBMS from other database types?
18. Explain the concept of a relation in an RDBMS.
19. What is referential integrity, and how is it enforced in an RDBMS?
20. What are the differences between SQL and NoSQL databases?

5. Normalization

21. What is normalization, and why is it important in database design?
22. Describe the different normal forms (1NF, 2NF, 3NF, BCNF).
23. What are the drawbacks of normalization?
24. When should you consider denormalization in database design?
25. Explain functional dependency and its role in normalization.

6. ACID Properties in DBMS

26. What do ACID properties stand for, and why are they crucial for transactions?
27. Explain the concept of atomicity in ACID properties.
28. How does consistency apply to ACID properties?
29. What is isolation in ACID properties, and why is it important?
30. Describe durability in the context of ACID properties.

7. DDL, DML, DCL Commands

31. What are the differences between DDL, DML, and DCL commands?
32. Give examples of common DDL commands and their purpose.
33. What are DML commands, and how are they used in SQL?
34. Explain the role of DCL commands in database management.
35. What are the effects of using the `DROP` command in SQL?

8. Joins

36. What are joins in SQL, and why are they used?
37. Explain the difference between inner join, outer join, left join, and right join.
38. What is a cross join, and how does it differ from other join types?
39. How do joins affect query performance in databases?
40. Can you describe a use case where a self-join is beneficial?

9. Aggregate Functions

41. What are aggregate functions in SQL, and how are they used?
42. Explain the difference between `COUNT()`, `SUM()`, `AVG()`, `MAX()`, and `MIN()`.
43. How do you use the `GROUP BY` clause with aggregate functions?
44. What is the purpose of the `HAVING` clause in SQL?
45. How do aggregate functions behave with NULL values?

10. Nested Queries and Correlated Queries

46. What is a nested query in SQL, and how does it work?
47. Explain the difference between a subquery and a correlated query.
48. How can you use nested queries to filter results in SQL?
49. Provide an example of a correlated query and explain its purpose.
50. What are the performance implications of using nested queries?

11. Miscellaneous Concepts

51. What is indexing, and how does it improve database performance?
52. Explain the difference between clustered and non-clustered indexes.
53. What is sharding, and how does it contribute to scalability?
54. How do triggers work in a database, and when should they be used?
55. What are views in SQL, and how do they differ from tables?
56. How can you update or delete data in a view?
57. Explain the concept of data warehousing and its relationship with databases.
58. What are stored procedures, and how do they differ from functions?
59. Describe the role of data integrity constraints in a database.
60. How do you perform data migrations in a production environment?
61. What are the differences between OLTP and OLAP systems?

62. How does database replication work, and what are its benefits?
63. What is the purpose of a database schema?
64. Explain the role of metadata in a DBMS.
65. How do you handle database backups and recovery?
66. What are some best practices for database security?
67. Explain the concept of a data model and its types.
68. What is the difference between logical and physical data models?
69. How do you optimize SQL queries for better performance?
70. What is the role of a database administrator (DBA)?

4. AWS:

1. Introduction to AWS

1. What is AWS, and why is it significant in cloud computing?
2. Explain the differences between IaaS, PaaS, and SaaS.
3. What are the key benefits of using AWS?
4. What are Regions and Availability Zones in AWS?

2. Basics of AWS

6. What are the fundamental services provided by AWS?
7. Describe the AWS Management Console and its primary functionalities.
8. What is the AWS CLI, and how does it differ from the Management Console?
9. Explain the concept of AWS Identity and Access Management (IAM).
10. What are IAM roles, and when should you use them?

3. EC2 (Elastic Compute Cloud)

11. What is EC2, and what are its main use cases?
12. Explain the different EC2 instance types and their purposes.
13. How do you launch an EC2 instance using the AWS Management Console?
14. What is an Amazon Machine Image (AMI)?
15. How do security groups work in EC2?

4. S3 (Simple Storage Service)

16. What is Amazon S3, and what are its primary use cases?
17. Explain the concept of S3 buckets and objects.
18. What are S3 storage classes, and how do they differ?
19. How does versioning work in S3?
20. What are the best practices for securing S3 buckets?

5. Lambda Function

21. What is AWS Lambda, and how does it differ from traditional server-based computing?
22. Describe the benefits of using serverless architecture with Lambda.
23. Explain the concept of event-driven architecture in AWS Lambda.
24. How do you manage Lambda function versions and aliases?
25. What are some common use cases for AWS Lambda?

6. RDS (Amazon Relational Database Service)

26. What is Amazon RDS, and what are its key features?
27. Explain the difference between Amazon RDS and traditional database solutions.
28. How do you perform backups and recovery in RDS?
29. What are read replicas in Amazon RDS, and why are they used?
30. Describe the process of scaling an RDS instance.

7. EBS (Elastic Block Storage)

31. What is Amazon EBS, and how does it work with EC2?
32. Explain the different EBS volume types and their use cases.
33. How do you create a snapshot of an EBS volume?
34. What are the performance characteristics of EBS?
35. How does EBS encryption work?

8. DynamoDB

36. What is Amazon DynamoDB, and what are its key features?
37. Explain the concept of a NoSQL database and how DynamoDB fits into this model.
38. What are the differences between DynamoDB and traditional relational databases?
39. Describe how partition keys and sort keys work in DynamoDB.

40. How do you handle transactions in DynamoDB?

9. SQS (Simple Queue Service)

41. What is Amazon SQS, and what are its primary use cases?
42. Explain the differences between standard queues and FIFO queues in SQS.
43. How do you manage message visibility in SQS?
44. What are dead-letter queues, and how do they work?
45. Describe the process of integrating SQS with other AWS services.

10. SNS (Amazon Simple Notification Service)

46. What is Amazon SNS, and how does it differ from SQS?
47. Explain the concept of topics and subscriptions in SNS.
48. How do you use SNS for mobile push notifications?
49. Describe the process of sending messages via SNS.
50. What are the use cases for integrating SNS with Lambda functions?

11. General AWS Concepts

51. What is AWS CloudFormation, and how does it facilitate infrastructure as code?
52. Explain the concept of AWS CloudTrail and its importance in monitoring.
53. What is AWS VPC, and what are its key features?
54. How does AWS Route 53 work, and what are its main functionalities?
55. Describe the purpose of AWS Elastic Load Balancing (ELB).

12. Security and Compliance

56. What are the shared responsibility model and its implications in AWS?
57. Explain the importance of AWS Config and its role in compliance.
58. How do you secure AWS resources using IAM policies?
59. What is AWS Shield, and how does it help protect against DDoS attacks?
60. How do you implement encryption for data at rest and in transit in AWS?

13. Monitoring and Management

61. What is Amazon CloudWatch, and what are its main functionalities?
62. How do you set up alarms and notifications using CloudWatch?
63. Explain the purpose of AWS Systems Manager.
64. What is AWS Trusted Advisor, and how can it help optimize AWS resources?

65. Describe the importance of tagging resources in AWS.

14. Cost Management

66. What are some strategies for managing costs in AWS?
67. How can AWS Budgets help in tracking and controlling costs?
68. Explain the concept of Reserved Instances and their pricing model.
69. What is AWS Savings Plans, and how does it differ from Reserved Instances?
70. Describe how to analyze AWS spending using Cost Explorer.

---

5. Docker:

1. Introduction to Docker

1. What is Docker, and how does it differ from traditional virtualization?
2. Explain the concept of containerization and its benefits.
3. What are the key components of Docker architecture?
4. How do Docker containers interact with the host operating system?

2. Installing Docker

5. Describe the steps involved in installing Docker on a Linux system.
6. What are the system requirements for installing Docker?
7. How can you verify that Docker has been installed correctly?

3. Deploying a Spring Boot Application to Docker

8. What are the steps to containerize a Spring Boot application?
9. How do you create a Dockerfile for a Spring Boot application?
10. Explain the process of building and running a Docker container for a Spring Boot application.

4. Docker Concepts

11. What is the difference between a Docker registry and a repository?
12. How do tags work in Docker, and why are they important?

13. What is the difference between Docker images and containers?
14. How are Docker images built and stored?

5. Playing with Docker Images and Containers

15. Explain how to pull an image from a Docker registry.
16. What commands are used to list all Docker images and containers?
17. How can you remove a Docker image and what are the implications?
18. What is the difference between stopping and removing a Docker container?

6. Applications of Docker

19. What are some common use cases for Docker in software development?
20. How does Docker facilitate continuous integration and continuous deployment (CI/CD)?
21. What role does Docker play in microservices architecture?
22. Discuss how Docker can help in environment consistency across development, testing, and production.

7. Deep Dive into Docker Images

23. What is the structure of a Docker image?
24. How do layers in Docker images work, and what are their benefits?
25. Explain how to optimize Docker images for size and performance.
26. What are multi-stage builds in Docker, and how do they work?

8. Deep Dive into Docker Containers

27. What are the lifecycle states of a Docker container?
28. How does Docker manage resource allocation for containers?
29. What are the differences between privileged and non-privileged containers?
30. Explain the purpose and use of Docker volumes.

9. Docker Commands

31. What is the purpose of the `docker stats` command?
32. Describe the information that can be obtained using `docker system df`.
33. How can you check the logs of a running container?

13. Setting Up Microservices for Creating Container Images

43. What are the best practices for structuring microservices when using Docker?

44. How can you ensure that microservices are independently deployable?
45. Explain the importance of API versioning in microservices.

14. Creating Container Image for Currency Exchange Microservice

46. Describe the steps involved in creating a container image for a currency exchange microservice.
47. What considerations should be taken into account when defining dependencies in the Dockerfile?
48. How can you manage configuration settings for a microservice running in a Docker container?

15. Getting Started with Docker Compose

49. What is Docker Compose, and how does it simplify the management of multi-container applications?
50. Describe the structure of a Docker Compose file.
51. How do services communicate with each other in a Docker Compose setup?

16. Running Eureka Server with Docker Compose

52. What is Eureka, and what role does it play in microservices?
53. How can you configure a Docker Compose file to run a Eureka server?
54. Explain how microservices can register with Eureka for service discovery.

17. Running Microservices with Docker Compose

55. What are the benefits of using Docker Compose for running microservices?
56. How do you define dependencies between services in a Docker Compose file?
57. Describe how to scale services using Docker Compose.

18. Running Cloud API Gateway with Docker Compose

58. What is an API gateway, and why is it important in microservices architecture?
59. How can you set up an API gateway using Docker Compose?
60. Discuss the benefits of using an API gateway for managing microservices.
61.

19. Running RabbitMQ with Docker Compose

64. What is RabbitMQ, and what role does it play in microservices?
65. What are the benefits of using message brokers like RabbitMQ in microservices architecture?

6. Flask API

1. Description of Flask API

1. What is Flask, and what are its main features?
2. Describe the typical architecture of a Flask application.
3. How does Flask handle routing, and what are route decorators?
4. What is the role of request and response objects in Flask?
5. Explain how Flask supports RESTful API development.
6. How do you manage configurations in a Flask application?
7. What is the purpose of Flask extensions, and can you name a few commonly used ones?
8. How does Flask handle errors and exceptions?

2. Checklist for Flask API Development

9. What are the key considerations when setting up a Flask API?
10. How do you ensure that your Flask API is secure?
11. What practices should be followed for input validation in Flask?
12. Explain how to structure a Flask project for scalability.
13. What testing strategies can be employed for a Flask API?
14. How can logging be implemented in a Flask application?
15. Describe how to document a Flask API effectively.

3. Checklist for Serverless Flask Applications

16. What is a serverless architecture, and how does it relate to Flask?
17. Describe the steps to deploy a Flask API on AWS Lambda.
18. What are the advantages of using serverless computing for Flask applications?
19. How does API Gateway integrate with a Flask application in a serverless setup?
20. What challenges might arise when developing a serverless Flask API?

4. Integrating Flask with SQLAlchemy and Alembic

21. What is SQLAlchemy, and how does it interact with Flask?
22. Explain how to set up SQLAlchemy in a Flask application.
23. What is Alembic, and how is it used for database migrations in Flask?

24. Describe the process of creating a new database migration using Alembic.
25. How do you handle database sessions in Flask with SQLAlchemy?
26. What are the benefits of using ORM (Object-Relational Mapping) with Flask?

5. Flask G Object

27. What is the `g` object in Flask, and how is it used?
28. How does the `g` object facilitate storing data during a request?
29. In what scenarios would you use the `g` object over session or request contexts?
30. Explain how to access and manipulate data stored in the `g` object across different routes.

6. Best Practices and Advanced Topics

31. What are the best practices for handling CORS (Cross-Origin Resource Sharing) in a Flask API?
32. How can you implement authentication and authorization in a Flask application?
33. Discuss how to optimize the performance of a Flask API.
34. What role does middleware play in Flask applications?
35. Explain how to use Flask-RESTful to create RESTful APIs easily.

7. FastAPI

1. Description of FastAPI

1. What is FastAPI, and what are its main advantages over other web frameworks?
2. Explain how FastAPI leverages type hints in Python.
3. How does FastAPI achieve high performance in handling requests?
4. Describe the typical architecture of a FastAPI application.
5. What are the key components of a FastAPI application?
6. How does FastAPI support dependency injection?
7. Discuss the automatic generation of API documentation in FastAPI.

2. Asynchronous Programming with FastAPI

8. What is the role of `async` and `await` in FastAPI?
9. Explain how FastAPI manages asynchronous I/O operations.

10. What are the benefits of using asynchronous programming in web applications?
11. How do you define an asynchronous endpoint in FastAPI?
12. Describe the impact of asynchronous programming on performance and scalability.
13. What are common pitfalls when using `async` in FastAPI?

3. WebSocket Support

14. What is a WebSocket, and how does it differ from traditional HTTP?
15. How does FastAPI support WebSocket connections?
16. Explain how to create a simple WebSocket endpoint in FastAPI.
17. What are some use cases for WebSocket in web applications?
18. How can you handle multiple WebSocket connections in FastAPI?

4. Implementation of Asynchronous Tasks

19. How can you implement background tasks in FastAPI?
20. What libraries can be used to handle asynchronous tasks with FastAPI?
21. Describe how to manage long-running tasks in a FastAPI application.
22. How can you ensure that asynchronous tasks are not blocking the main event loop?

5. Pydantic

23. What is Pydantic, and how is it used in FastAPI?
24. Explain the role of Pydantic models in data validation and serialization.
25. How do you define a Pydantic model for request and response bodies?
26. What are some advantages of using Pydantic over traditional data validation methods?
27. Describe how to handle nested data structures using Pydantic.

6. Unit Testing

28. Why is unit testing important in web applications?
29. Explain the role of testing in the development lifecycle of a FastAPI application.
30. What are some best practices for writing unit tests for FastAPI endpoints?

7. Pytest

31. What is Pytest, and why is it commonly used for testing in Python?
32. How do you install and set up Pytest for a FastAPI project?
33. Describe how to create a basic test case using Pytest.

34. What are fixtures in Pytest, and how can they be used to simplify tests?
35. How can you parameterize tests in Pytest?

8. FastAPI Testing with Pytest

36. Explain how to test FastAPI endpoints using Pytest.
37. What is the purpose of the `TestClient` in FastAPI?
38. Describe how to simulate requests and responses in your FastAPI tests.
39. How can you assert the correctness of responses in your FastAPI tests?
40. Discuss strategies for testing error handling in FastAPI applications.

Implementation Question:

A. DSA Questions:

1. Given a string, write a function to reverse the characters of each word and then reverse the order of the words in the string. For example, if the input is `'Hello my name is abc'`, the output should be `'cba olleh si ym eman'`. Provide the algorithm, implementation, and discuss the time complexity of your solution.
2. CLOCKWISe rotation of a list.
3. Sorting dictionary based on some key inside it.
4. Generator implementation and use case.

5. How would you find the median of two sorted arrays? Can you explain an efficient algorithm to solve this problem, and discuss its time complexity?

6. **Detect a Cycle in a Linked List**

   You are given a singly linked list. The list may or may not contain a cycle. A cycle occurs when a node in the list points back to a previous node. You need to write a function that detects whether the linked list contains a cycle.

- If a cycle is present, return True.
- If no cycle is found, return False.

Example 1 (No Cycle):

- Linked List: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
- Output: False

Example 2 (With Cycle):

- Linked List: 1 -> 2 -> 3 -> 2 (The second node points back to the first node)
- Output: True

7. **0/1 Knapsack Problem**

   A thief is robbing a store and can carry a maximal weight W in his knapsack. There are N items available, each with a specific weight and value. The objective is to find the maximum value the thief can carry without exceeding the weight limit.

   Input:

- n: Number of items
- weights[]: An array of item weights
- values[]: An array of item values
- W: The maximum weight the knapsack can carry

   Task:

   Write a function that returns the maximum value that can be carried given the weight constraints.

   Example 1:

- n = 3
- weights = [10, 20, 30]
- values = [60, 100, 120]
- W = 50
- **Output:** 220 (The thief can take the items weighing 20 and 30)

   Example 2:

- n = 4
- weights = [1, 2, 4, 5]
- values = [5, 4, 8, 6]
- W = 5
- **Output:** 13 (The thief can take items weighing 1 and 4)

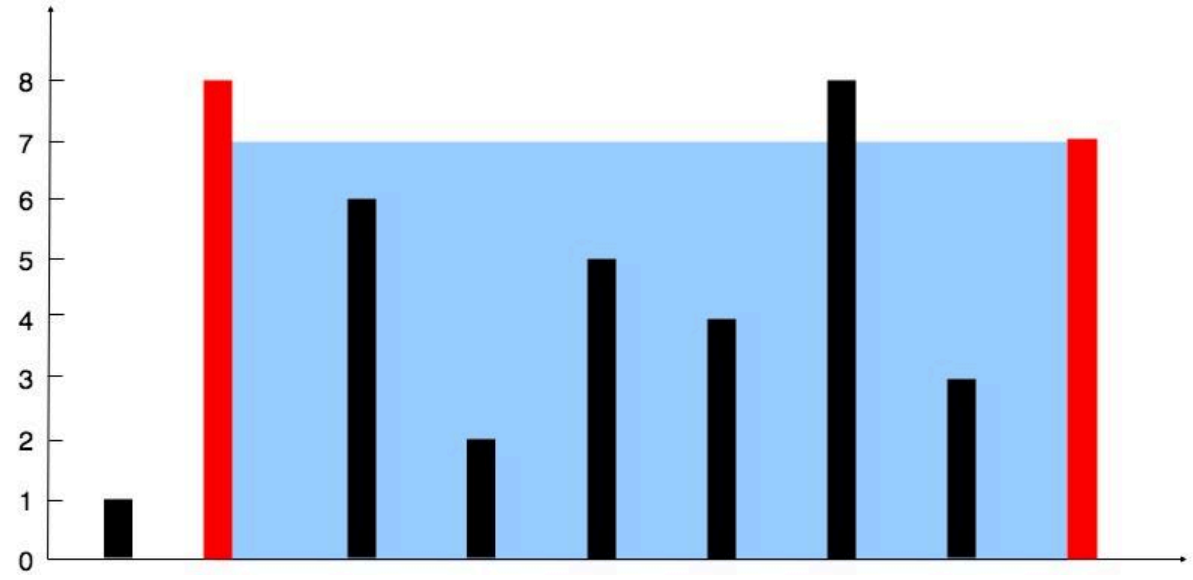
8. You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]

Output: 1

Constraints:

n == height.length

2 <= n <= 105

0 <= height[i] <= 104

9. How would you convert a string of numbers separated by commas into a list of integers? Please provide an example and explain the steps involved.

10. How would you design a database schema for a multi-tenant system? Please include details on how to handle tenant-specific data, shared resources, user authentication, and data isolation between tenants. Explain any strategies you would use for scaling the system and ensuring data security for each tenant.

11. Make a decorator to Capitalise first letter in a sentence

12. How would you write a function that takes a string as input and compresses it by grouping consecutive repeating characters, returning the character followed by its count? For example, given the input "aaaabbbcccc", the function should return "a4b3c4".

13. How would you write a function that flattens a nested list into a single list of elements? For example, given the input [1, [2, 3, [4]], 5, 6], the function should return [1, 2, 3, 4, 5, 6].

B. AWS:

1. EC2 Implementation

1. Write a command to launch an EC2 instance using the AWS CLI.
2. How do you create and configure a security group for an EC2 instance?
3. Explain the steps to connect to an EC2 instance using SSH.
4. Can you explain the steps to deploy a Django application on an EC2 instance? Include details about server setup, configuration, and any required services.
5. What are some strategies to handle incoming traffic to a server or IP address? How would you manage load balancing, scaling, and maintaining server health?
6. What is API rate limiting, and how would you implement it in a web application? Discuss common techniques and tools used to limit API requests.

2. S3 Implementation

4. Write a command to create an S3 bucket and apply versioning.
5. How do you upload a file to an S3 bucket using the AWS CLI?
6. Write a policy to restrict access to an S3 bucket.

3. Lambda Function Implementation

7. Describe how to create a Lambda function using the AWS Management Console.

8. Write a sample Python code for a Lambda function that processes S3 events.
9. How do you set environment variables for a Lambda function?

4. RDS Implementation

10. Write a SQL command to create a new database within an RDS instance.
11. Describe the process to take a manual snapshot of an RDS instance.
12. How would you enable Multi-AZ deployment for an RDS instance?

5. EBS Implementation

13. Write a command to create a snapshot of an EBS volume.
14. Explain how to attach an EBS volume to an EC2 instance.
15. How do you modify the size of an EBS volume after it has been created?

6. DynamoDB Implementation

16. Write a command to create a DynamoDB table with a partition key.
17. How do you insert an item into a DynamoDB table using the AWS SDK?
18. Describe the steps to enable DynamoDB Streams on a table.

7. SQS Implementation

19. Write a command to create an SQS FIFO queue.
20. How do you send a message to an SQS queue using the AWS SDK?
21. Explain how to receive messages from an SQS queue and delete them.

8. SNS Implementation

22. Write a command to create an SNS topic and subscribe an email endpoint.
23. How do you publish a message to an SNS topic using the AWS CLI?
24. Describe how to set up an SNS notification for an S3 event.

9. General Implementation Questions

25. Write a CloudFormation template to create an EC2 instance and an associated security group.
26. Explain the steps to set up a VPC with public and private subnets.
27. How do you implement IAM policies to restrict access to a specific S3 bucket?

10. Monitoring and Management Implementation

28. Write a command to create a CloudWatch alarm based on CPU utilization of an EC2 instance.

29. Describe how to enable AWS CloudTrail and analyze the logs.
30. How would you use AWS Systems Manager to run a command on multiple EC2 instances?

C. DB:

1. Basic Queries

1. Write a SQL query to retrieve all records from a table named `employees`.
2. How would you use the `JOIN` clause to combine data from `orders` and `customers` tables?
3. Write a query to find the total number of employees in each department using the `GROUP BY` clause.
4. Create a query that returns the highest salary from the `employees` table.
5. Write a SQL command to create a table named `products` with appropriate columns.

2. Advanced Queries

6. How would you write a nested query to find employees who earn more than the average salary?
7. Create a correlated query to list employees whose salary is higher than the average salary in their department.
8. Write a SQL statement to update the `status` column in the `orders` table based on specific conditions.
9. How would you delete all records from a table while retaining its structure?
10. Write a query to retrieve the top 5 highest-paid employees from the `employees` table.

3. Indexing and Optimization

11. Explain how to create an index on the `last_name` column of the `employees` table.
12. Write a query to analyze the performance of a given SQL query and suggest improvements.
13. How do you drop an index from a table? Provide the SQL command.
14. Create a composite index on the `first_name` and `last_name` columns of the `employees` table.
15. Write a SQL query to analyze the effect of indexing on query performance.

4. Triggers and Views

16. Write a trigger that automatically updates a `last_modified` timestamp whenever a record in the `employees` table is updated.

17. How do you create a view that summarizes total sales per product from the `sales` table?
18. Write a query to update a view and explain how it differs from updating a table directly.
19. How do you drop a view in SQL? Provide the SQL command.
20. Create a view that displays employee names and their respective department names.

5. Normalization and Schema Design

21. Given a denormalized table structure, write a plan for normalizing it to at least 3NF.
22. Create an ER diagram for a simple library management system.
23. Write SQL statements to create the tables and define relationships based on your ER diagram.
24. Explain how to implement a foreign key constraint in SQL.
25. Write a SQL command to perform a bulk insert into the `products` table.

6. Aggregate Functions and Data Manipulation

26. Write a SQL query to calculate the average sales per month from a `sales` table.
27. Use the `HAVING` clause to filter groups based on aggregate functions.
28. Create a query that retrieves sales data for a specific date range using the `BETWEEN` operator.
29. Write a query to count the number of unique products sold.
30. Create a SQL command to perform a complex transaction involving multiple tables.
31. DB design question:

Which db will u prefer?

Provide a raw Database Structure for Dynamic Role and Field Management:

You are tasked with designing a database schema that supports the following requirements:

Role Definition: The system should allow the creation of different roles, such as "Manager," "Dev," and "Team Lead," with each role having its own set of associated fields (e.g., "Name," "Email," "DOB"). The schema should allow for additional roles to be created dynamically by an Admin.

Dynamic Field Types: Each field associated with a role can have different types such as:

Integer

Alphanumeric (Text)

Dropdown (Predefined options)

Radio buttons

These fields will determine how the front-end dynamically renders the form elements for data entry.

Role-Field Relationship: Each role can have multiple fields attached to it, and these fields may vary across roles. For example:

A "Manager" role may have the fields: "Name," "Email."

A "Dev" role may have the fields: "Name," "Email," "DOB."

Multi-Role Capability: A user can have multiple roles. For example:

A "Senior Dev" may hold both the "Dev" role and the "Team Lead" role simultaneously.

The schema should be designed in such a way that users can be assigned multiple roles and their fields are dynamically managed based on the combined set of fields from all roles they hold.

Admin Capabilities: An admin should be able to create new roles dynamically, define which fields are attached to each role, and specify the type of each field.

Your task: Design the database structure that:

Captures the relationship between roles and their fields.

Supports the different field types.

Allows users to have multiple roles, while ensuring that field relationships are respected across all assigned roles.

32. How would you design a database schema for a multi-tenant system? Please include details on how to handle tenant-specific data, shared resources, user authentication, and data isolation between tenants. Explain any strategies you would use for scaling the system and ensuring data security for each tenant.

Important Topics Asked in Interviews:

Django ORM

Transactions in Django implementation

GIT

MVT in DJango

CSS Selectors

Which one is best postgresql, mysql, mongodb?

List to dict

Add two large number out of the range of memory

Generators all use cases

Dependency Injection in FastAPI

Async API

Microservices

database

Celery, Twilio, Send grant

Web Sockets

Implementation of Client Side, Javascript, Server - Side

Stack Uses

SSO Implementation

Docker

Model Designing

Dictionary sort "Hello"

self

JWT Token

Queue Service

prefetch and select related

When to use Sharding

Django ORM

Transactions in Django implementation

GIT

MVT in DJango

CSS Selectors

Which one is best postgresql, mysql, mongodb?

List to dict

Add two large number out of the range of memory

Generators all use cases

Dependency Injection in FastAPI

Async API

Microservices

database

Celery, Twilio, Send grant

Web Sockets

Implementation of Client Side, Javascript, Server - Side

Stack Uses

SSO Implementation

Docker

Model Designing

Dictionary sort "Hello"

self

JWT Token

Queue Service

prefetch and select related

When to use Sharding