**SECTION A**

**1. What is Apache Airflow and why is it used?**  
Apache Airflow is an open-source workflow orchestration platform used to **programmatically author, schedule, and monitor data pipelines**.  
It helps automate complex data engineering tasks by managing dependencies, retries, and execution order efficiently.

**2. Define a DAG. What does each part of the acronym stand for?**  
A **DAG (Directed Acyclic Graph)** is a collection of tasks with defined relationships that show how they depend on each other.

* **Directed** → The flow has a specific direction (from one task to another).
* **Acyclic** → No loops or cycles are allowed.
* **Graph** → A set of nodes (tasks) connected by edges (dependencies).

**3. Explain the difference between a DAG and a Task.**

* A **DAG** represents the **overall workflow or pipeline**, defining the structure and dependencies.
* A **Task** is a **single unit of work** inside the DAG (e.g., extract data, run SQL, upload file).  
  In short, a DAG is the *blueprint*, and tasks are the *steps* within it.

**4. Why should workflows be “Directed Acyclic Graphs” in Airflow?**  
Workflows must be DAGs to ensure **deterministic, non-repetitive execution**.  
The “acyclic” nature prevents infinite loops and guarantees that each task runs only once in a clear order, ensuring predictable and traceable pipeline execution.

**SECTION B**

**1. Describe the role of the following Airflow components:**

* **Webserver:**  
  Provides the **Airflow UI** where users can visualize DAGs, monitor task status, trigger runs, and check logs.
* **Scheduler:**  
  Continuously **monitors DAGs and schedules task executions** based on their start\_date and schedule\_interval. It decides what tasks to run and when.
* **Metadata Database:**  
  Stores **all Airflow state and configuration information**, including DAG runs, task statuses, variables, connections, and user settings. It acts as Airflow’s **source of truth**.

**2. What is the purpose of the airflow db init command?**  
The airflow db init command **initializes the metadata database** by creating all necessary tables and default entries required for Airflow to function properly.

**3. What is the significance of start\_date and schedule\_interval in a DAG?**

* **start\_date:** Defines **when the first DAG run is scheduled** to start. Airflow uses it as a reference point for future runs.
* **schedule\_interval:** Specifies **how often the DAG should run** (e.g., daily, hourly, @once, cron expression).  
  Together, they control **the timing and frequency** of pipeline execution.

**4. What does catchup=False do, and when would you use it?**  
catchup=False tells Airflow **not to backfill** (i.e., not to run missed DAG runs between the start\_date and the current date).  
You’d use it when **only the latest run matters**—for example, in streaming or near real-time pipelines where historical runs are unnecessary.

**SECTION C**

**1. What is an Operator? Give two examples.**  
An **Operator** in Airflow defines a **single unit of work (task)** — what action should be performed in a DAG.  
Examples:

* **BashOperator:** Executes a bash command or script.
* **PythonOperator:** Runs a Python function.

(Other examples: EmailOperator, DummyOperator, SQL-related operators, etc.)

**2. How does Airflow handle task failures and retries?**  
When a task fails, Airflow automatically **retries** it based on the retries and retry\_delay parameters defined in the task.  
If all retries fail, the task is marked as **failed**, and downstream tasks depending on it are skipped (unless configured otherwise).

**3. What is XCom and how is it useful?**  
**XCom (Cross-Communication)** is a mechanism that allows **tasks to share small pieces of data** between each other.  
Example: one task pushes a result (xcom\_push), and another retrieves it (xcom\_pull).  
It’s useful for passing intermediate values like file paths, query results, or status flags between tasks.

**4. Explain the difference between BashOperator and PythonOperator.**

* **BashOperator:** Executes shell or bash commands in a system environment.  
  *Example:* running ls or calling a shell script.
* **PythonOperator:** Executes a **Python function** directly within Airflow.  
  *Example:* running data transformations or API calls via Python code.

**SECTION D**

**1. Give one real-world example where Airflow can be used for ETL.**  
Airflow can be used to **automate a daily ETL pipeline** — for example, extracting sales data from an API, transforming it with Python or SQL, and loading it into a data warehouse like Snowflake or BigQuery for analytics.

**2. Why is it recommended to keep DAG scripts lightweight and avoid heavy computations inside them?**  
Because the **DAG file is parsed by the scheduler and webserver repeatedly**, heavy computations can **slow down or crash Airflow’s metadata parsing**.  
Only workflow definitions should be inside the DAG — not data processing logic.

**3. Why should every DAG have a unique dag\_id?**  
Each DAG must have a **unique dag\_id** so Airflow can **track and manage it separately** in the metadata database.  
Duplicate IDs can cause confusion, overwriting, or unexpected behavior in scheduling and logging.

**4. How does Airflow ensure workflows run in the correct order?**  
Airflow enforces task execution order using **explicit dependencies** (set\_upstream, set\_downstream, or >>, << operators).  
The **DAG’s directed edges** define this sequence, ensuring tasks run strictly according to their dependency graph.