**🟢 Level 1 – Easy (4 DAGs)**

Focus: structure, scheduling, simple PythonOperators, and dependencies.

1. **Print Hello DAG**
   * 3 tasks: start, say\_hello, end
   * Each prints a message.
   * Runs daily.
2. **Data Extraction DAG**
   * One task fetches data from a dummy API (e.g. <https://jsonplaceholder.typicode.com/users>).
   * Another task prints the number of records.
   * Schedule: @hourly.
3. **ETL Skeleton DAG**
   * 3 tasks: extract → transform → load.
   * Each just prints a fake log line for now.
   * Schedule: @daily.
4. **Cron Schedule DAG**
   * Runs every weekday at 9:15 AM.
   * Task: print "Office hours started".
   * Focus on correct schedule\_interval.

**🟡 Level 2 – Intermediate (4 DAGs)**

Focus: XComs, branching, failure handling, retries, and parallel tasks.

1. **Branching DAG**
   * Task A randomly returns “yes” or “no”.
   * Use BranchPythonOperator to decide which downstream task (task\_yes or task\_no) runs.
2. **XCom Communication DAG**
   * extract\_data pushes a dictionary to XCom.
   * transform\_data pulls and modifies it.
   * load\_data prints final output.
   * Mimic a mini ETL.
3. **Parallel Processing DAG**
   * start → runs three parallel tasks (t1, t2, t3) that sleep random durations.
   * All join into one merge\_results task.
   * Purpose: practice parallel dependencies.
4. **Retry & Alert DAG**
   * Task that fails randomly (random.choice([True, False])).
   * Add retry logic (retries=3, retry\_delay=timedelta(minutes=2)).
   * Print success/failure message.

**🔴 Level 3 – Hard (2 DAGs)**

Focus: integration, dynamic task mapping, sensors, or external systems.

1. **Dynamic Task Generation DAG**
   * Create a list of filenames or table names.
   * Dynamically generate one PythonOperator per item using Airflow 2.x task mapping (@task decorator or expand).
   * Each task processes one file.
2. **End-to-End Mini Pipeline**

* Fetch JSON data from an API → store it as file → load into SQLite or Snowflake (mock ok).
* Use XComs or temp file passing.
* Add realistic scheduling (0 6 \* \* \*) and a success message at the end.
* Bonus: include retry and error handling.