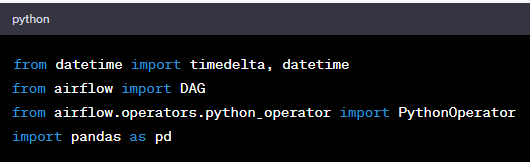
Code Explanation:

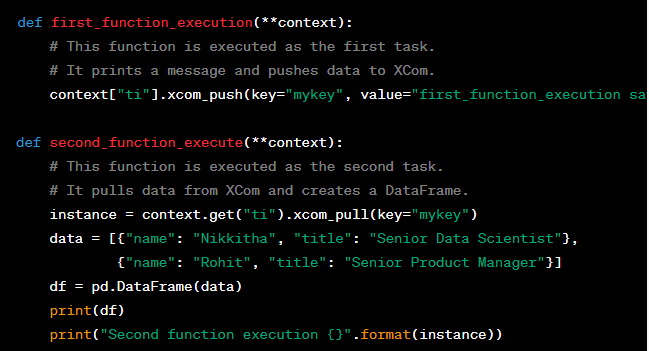
================

Step 1: Importing the necessary libraries:



Here, the necessary libraries are imported: **timedelta** and **datetime** for dealing with time intervals and timestamps, **DAG** for creating the DAG itself, and **PythonOperator** for defining Python tasks within the DAG.

Step 2: Defining the task functions:



These are the two functions that will be executed as tasks within the DAG. The **first\_function\_execution** function pushes data to XCom, and the **second\_function\_execute** function pulls data from XCom and prints a DataFrame along with the XCom data.

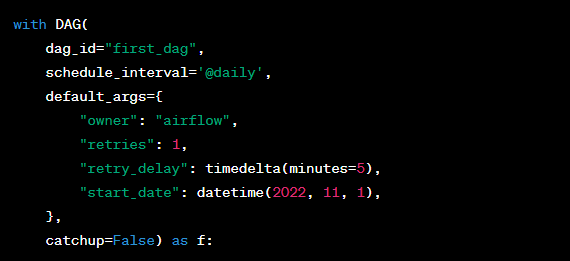
**\*\*context** parameter is a special argument (Apache airflow task function) used in task functions when defining tasks within a DAG. It allows task functions to access and interact with various contextual information and data related to the execution of the task and the DAG run. This context provides a way for tasks to share data, communicate with each other, and access information about the execution environment.

The **context** dictionary contains a variety of information and attributes that can be useful for tasks. Some of the commonly used attributes in the **context** dictionary include:

1. **dag**: A reference to the DAG object that the task belongs to. This allows tasks to access properties and methods of the DAG.
2. **ds**: The execution date of the task in "YYYY-MM-DD" format.
3. **ds\_nodash**: The execution date of the task in "YYYYMMDD" format.
4. **ts**: The execution timestamp of the task in "YYYY-MM-DDTHH:MM:SS" format.
5. **ts\_nodash**: The execution timestamp of the task in "YYYYMMDDTHHMMSS" format.
6. **task\_instance**: An instance of the **TaskInstance** class, providing access to methods and attributes related to the current task instance.
7. **ti**: An alias for **task\_instance**, providing the same access to task instance methods and attributes.
8. **execution\_date**: The execution date and time of the task as a **datetime** object.
9. Custom XComs: Data pushed and pulled using the XCom mechanism, which allows tasks to share data between them.

When you use **provide\_context=True** while defining an operator, it indicates that you want the **context** dictionary to be passed as an argument to the task function. This enables the task function to access all the attributes and information mentioned above.

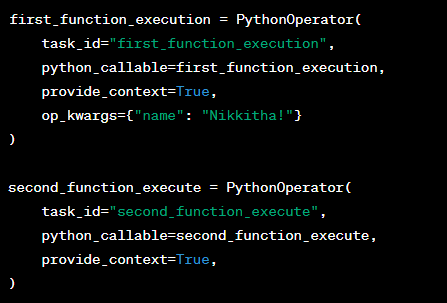
Step 3: Creating a DAG:



Here, a DAG named "first\_dag" is created. It has the following properties:

* **dag\_id**: The unique identifier for the DAG.
* **schedule\_interval**: The frequency at which the DAG should run. In this case, it's set to run daily.
* **default\_args**: Dictionary containing default configuration parameters for the DAG and tasks.
* **start\_date**: The date from which the DAG should start running.
* **catchup**: Whether to backfill or "catch up" on missed runs. In this case, it's set to **False**.

Step 4: Defining Tasks:



Two tasks are defined using **PythonOperator**. These tasks are instances of the functions defined earlier. They have unique **task\_id** identifiers, callable functions, and the **provide\_context** parameter set to **True**, which allows them to receive the context dictionary.

Step 5: Specifying Task Dependencies:



This line defines the dependency between the tasks. It indicates that the **second\_function\_execute** task depends on the successful execution of the **first\_function\_execution** task.