sample_dagfile.py

```
'''
Explanation of Workflow Execution:
    Task 1 (start_task): Starts with the number 10.
    Task 2 (add_five_task): Adds 5, resulting in 15.
    Task 3 (multiply_by_two_task): Multiplies 15 × 2 = 30.
    Task 4 (subtract_three_task): Subtracts 3, resulting in 30 - 3 = 27.
    Task 5 (square_number_task): Squares 27^2 = 729.
This Airflow DAG executes a step-by-step mathematical sequence using PythonOperator tasks, with values passed using XCom for data sharing between tasks.
'''

# Import necessary modules from Apache Airflow
from airflow import DAG  # DAG (Directed Acyclic Graph) is used to define a workflow
from airflow.operators.python import PythonOperator  # PythonOperator is used to define tasks that execute Python functions
from datetime import datetime  # Used to define the start date of the DAG

# Define a function for the first task (starting number)
def start_number(**context):
    # Push the initial number (10) into XCom (Airflow's way to share data between tasks)
    context["ti"].xcom_push(key='current_value', value=10)
    print("Starting number 10")

# Define a function to add 5 to the current number
def add_five(**context):
    # Retrieve the current value from XCom (from start_task)
    current_value = context['ti'].xcom_pull(key='current_value', task_ids='start_task')
    # Perform the addition
    new_value = current_value + 5
    # Store the updated value in XCom for the next task
    context["ti"].xcom_push(key='current_value', value=new_value)
    print(f"Add 5: {current_value} + 5 = {new_value}")

# Define a function to multiply the result by 2
def multiply_by_two(**context):
    # Retrieve the current value from XCom (from add_five_task)
    current_value = context['ti'].xcom_pull(key='current_value', task_ids='add_five_task')
    # Perform the multiplication
    new_value = current_value * 2
    # Store the updated value in XCom for the next task
    context['ti'].xcom_push(key='current_value', value=new_value)
    print(f"Multiply by 2: {current_value} * 2 = {new_value}")

# Define a function to subtract 3 from the result
def subtract_three(**context):
    # Retrieve the current value from XCom (from multiply_by_two_task)
    current_value = context['ti'].xcom_pull(key='current_value', task_ids='multiply_by_two_task')
    # Perform the subtraction
    new_value = current_value - 3
```

```
    # Store the updated value in XCom for the next task
    context['ti'].xcom_push(key='current_value', value=new_value)
    print(f"Subtract 3: {current_value} - 3 = {new_value}")

# Define a function to compute the square of the result
def square_number(**context):
    # Retrieve the current value from XCom (from subtract_three_task)
    current_value = context['ti'].xcom_pull(key='current_value',
task_ids='subtract_three_task')
    # Perform the squaring operation
    new_value = current_value ** 2
    print(f"Square the result: {current_value}^2 = {new_value}")

# Define the DAG (Directed Acyclic Graph)
with DAG(
    dag_id='math_sequence_dag',  # Unique identifier for the DAG
    start_date=datetime(2023, 1, 1),  # The date when the DAG starts running
    schedule_interval='@once',  # Specifies that the DAG runs only once
    catchup=False  # Prevents backfilling (running old, missed executions)
) as dag:

    # Define the first task (starting number)
    start_task = PythonOperator(
        task_id='start_task',  # Unique task identifier
        python_callable=start_number,  # Function to be executed
        provide_context=True  # Allows the task to access context variables like XCom
    )

    # Define the second task (adding 5)
    add_five_task = PythonOperator(
        task_id='add_five_task',
        python_callable=add_five,
        provide_context=True
    )

    # Define the third task (multiplying by 2)
    multiply_by_two_task = PythonOperator(
        task_id='multiply_by_two_task',
        python_callable=multiply_by_two,
        provide_context=True
    )

    # Define the fourth task (subtracting 3)
    subtract_three_task = PythonOperator(
        task_id='subtract_three_task',
        python_callable=subtract_three,
        provide_context=True
    )

    # Define the fifth task (squaring the result)
    square_number_task = PythonOperator(
        task_id='square_number_task',
```

```
        python_callable=square_number,
        provide_context=True
    )

    # Define task dependencies (execution order)
    start_task >> add_five_task >> multiply_by_two_task >> subtract_three_task >>
square_number_task


```