sample_taskflowapi.py

```
'''
Explanation of the TaskFlow API Approach:

Using the @task decorator:
    This eliminates the need for manually defining PythonOperator tasks.
    Each function is decorated with @task, making it an Airflow task.
    The function return values are automatically passed between tasks.

Task Execution Flow:
    start_number() → Returns 10
    add_five(start_value) → 10 + 5 = 15
    multiply_by_two(added_values) → 15 * 2 = 30
    subtract_three(multiplied_value) → 30 - 3 = 27
    square_number(subtracted_value) → 27^2 = 729

How Dependencies Work:
    Instead of manually setting dependencies like task1 >> task2, we simply call the functions.
    Airflow automatically understands the order based on function calls.
    This approach makes the DAG cleaner, more readable, and easier to maintain!
'''

# Import necessary modules from Apache Airflow
from airflow import DAG  # DAG (Directed Acyclic Graph) is used to define a workflow
from airflow.decorators import task  # @task decorator is used to define Python-based tasks
from datetime import datetime  # Used to define the start date of the DAG

# Define the DAG (Directed Acyclic Graph) using a context manager
with DAG(
    dag_id='math_sequence_dag_with_taskflow',  # 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:

    # Task 1: Start with the initial number
    @task
    def start_number():
        """This function initializes the process with a starting number of 10."""
        initial_value = 10
        print(f"Starting number: {initial_value}")
        return initial_value  # Returns the value to be used in the next task

    # Task 2: Add 5 to the number
    @task
    def add_five(number):
        """This function adds 5 to the received number and returns the new value."""
        new_value = number + 5
        print(f"Add 5: {number} + 5 = {new_value}")
```

```
        return new_value  # Returns the updated value

    # Task 3: Multiply by 2
    @task
    def multiply_by_two(number):
        """This function multiplies the received number by 2 and returns the result."""
        new_value = number * 2
        print(f"Multiply by 2: {number} * 2 = {new_value}")
        return new_value  # Returns the updated value

    # Task 4: Subtract 3
    @task
    def subtract_three(number):
        """This function subtracts 3 from the received number and returns the result."""
        new_value = number - 3
        print(f"Subtract 3: {number} - 3 = {new_value}")
        return new_value  # Returns the updated value

    # Task 5: Square the number
    @task
    def square_number(number):
        """This function computes the square of the received number and returns the
result."""
        new_value = number ** 2
        print(f"Square the result: {number}^2 = {new_value}")
        return new_value  # Returns the final value

    # Define the task execution order using function calls
    start_value = start_number()  # Starts with the number 10
    added_values = add_five(start_value)  # Adds 5 to the initial number
    multiplied_value = multiply_by_two(added_values)  # Multiplies the result by 2
    subtracted_value = subtract_three(multiplied_value)  # Subtracts 3 from the result
    square_value = square_number(subtracted_value)  # Squares the final result


```