Question 1:

A: I'd be glad to provide a comprehensive response to the prompt, combining the best aspects of previous responses and addressing potential shortcomings:

**By default, Django signals are executed synchronously.** This means that the signal handler function is called directly within the same request context where the signal was sent. This behavior is generally efficient for most use cases, as it avoids the overhead of asynchronous processing.

Example:

- 1: my_signal_handler

2: save post

3: my model

Question 2:

A: Yes, Django signals run in the same thread as the caller by default. This is because the signal's `send` method is called synchronously, meaning the signal and the receiver function(s) execute in the same thread as the signal's sender.

Steps:

*We'll send a signal from a view.

*We'll print the thread IDs both from the view (the sender) and from the signal receiver to *verify they are the same.

Ex: If you need asynchronous signal handling (e.g., running receivers in a separate thread), you would have to implement that behavior explicitly, such as by using threading or an asynchronous task queue like Celery.

Conclusion:

The thread ID printed in both the view and the signal receiver is the same, conclusively proving that Django signals run in the same thread as the caller.

Question 3:

A:Yes, by default, Django signals run in the same database transaction as the caller. If a signal is triggered within a transaction block (such as inside a save() method or an atomic block), the signal's receivers are also executed within the same database transaction. This means that if the transaction is rolled back, the changes made by the signal's receivers will also be rolled back

## How to Prove It:

## We will:

1. Create a signal and a receiver that makes a change to the database.
2. Call the signal within a database transaction.
3. Intentionally cause the transaction to roll back (e.g., by raising an exception after the signal is sent).
4. Observe that the changes made by the signal are also rolled back.

## Verification:

After hitting the view, check the database. You should see that no `TestModel` entry exists because the transaction, including the signal, was rolled back.

Conclusion:

This proves that by default, Django signals run within the same database transaction as the caller. If the caller's transaction is rolled back, changes made by the signal's receivers are also rolled back.