

# **Answers for Django Trainee at Accuknox**

## [Topic: **Django Signals**](https://docs.djangoproject.com/en/3.2/topics/signals/)

**Question 1**: By default are django signals executed synchronously or asynchronously? Please support your answer with a code snippet that conclusively proves your stance. The code does not need to be elegant and production ready, we just need to understand your logic.

Answer :- By default, Django signals are executed synchronously. Which means that when signal is sent, the execution of code is blocked until all connected receivers have finished processing the signal.

Code snippet:

from django.dispatch import Signal

import time

# Define the custom signal

my\_signal = Signal()

# Receiver functions

def receiver1(sender, \*\*kwargs):

print("Receiver 1 started")

time.sleep(2) # I am trying to simulate some work by time.sleep

print("Receiver 1 finished")

def receiver2(sender, \*\*kwargs):

print("Receiver 2 started")

time.sleep(2) # I am trying to simulate some work by time.sleep

print("Receiver 2 finished")

# Connect the receivers to the signal

my\_signal.connect(receiver1)

my\_signal.connect(receiver2)

# Send the signal

print("Send signal...")

my\_signal.send(sender="sender", message="Hello, world!")

print("Signal sent...")

Execution of this code will be completed only after both receivers have finished processing the signal. “The Signal Sent.” message is only printed after both receivers have completed their work. This demonstrates that Django signals are executed synchronously by default.

**Question 2**: Do django signals run in the same thread as the caller? Please support your answer with a code snippet that conclusively proves your stance. The code does not need to be elegant and production ready, we just need to understand your logic.

Answer :-

Yes, Django Signals run in the same thread as the caller. To prove that Django signals run in the same thread as the caller, I will try to capture and print the thread IDs of both the signal sender and the signal handler. If both thread IDs match, then they are running in the same thread.

Code Snippet:-

import threading

import time

from django.db import models

from django.db.models.signals import post\_save

from django.dispatch import receiver

# Define a simple model

class MyModel(models.Model):

name = models.CharField(max\_length=100)

# Signal handler

@receiver(post\_save, sender=MyModel)

def my\_signal\_handler(sender, instance, \*\*kwargs):

print(f"Signal received for: {instance.name}")

print(f"Signal handler thread ID: {threading.get\_ident()}")

time.sleep(2) # Simulate a delay

print("Signal handler finished")

# Create an instance of MyModel

obj = MyModel.objects.create(name="Test")

print("Object created")

print(f"Main thread ID: {threading.get\_ident()}")

**Question 3**: By default do django signals run in the same database transaction as the caller? Please support your answer with a code snippet that conclusively proves your stance. The code does not need to be elegant and production ready, we just need to understand your logic.

Answer :-

Yes, Django signals run in the same database transaction as the caller. If a signal is connected to a model operation like post\_save, pre\_save etc, it is executed within the same database transaction. If the transaction is rolled back, any database changes made by the signal handler will also be rolled back.

Code Snippet:-

from django.db import models

from django.db.models.signals import post\_save

from django.dispatch import receiver

from django.db import transaction

class MyModel(models.Model):

name = models.CharField(max\_length=100)

class Log(models.Model):

message = models.CharField(max\_length=255)

# Signal handler

@receiver(post\_save, sender=MyModel)

def my\_signal\_handler(sender, instance, \*\*kwargs):

print("Signal handler called. Inserting a log entry...")

Log.objects.create(message=f"Log entry for: {instance.name}")

def create\_model\_instance():

try:

with transaction.atomic():

print("Creating model instance...")

# This will trigger the signal

MyModel.objects.create(name="Test Instance")

# Checking log entries created by the signal handler

print(f"Log count inside transaction: {Log.objects.count()}")

# Intentionally raise an error to roll back the transaction

raise Exception("Simulated error to roll back transaction")

except Exception as e:

print(f"Transaction failed: {e}")

# After rollback, check log entries again

print(f"Log count after transaction rollback: {Log.objects.count()}")

## Topic: Custom Classes in Python

**Description:** You are tasked with creating a Rectangle class with the following requirements:

1. An instance of the Rectangle class requires length:int and width:int to be initialized.
2. We can iterate over an instance of the Rectangle class
3. When an instance of the Rectangle class is iterated over, we first get its length in the format: **{'length': <VALUE\_OF\_LENGTH>}** followed by the width **{width: <VALUE\_OF\_WIDTH>}**