1. Django signals execute synchronously by default. In Django, signal handler and sender are executed synchronously on the same thread. That is all blocking further execution until all handlers are completed.

Code:

from django.dispatch import Signal, receiver

import time

signal = Signal()

@receiver(signal)

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

    print("Handler started\nWaiting 2 secs")

    time.sleep(2)

    print("Handler finished")

# Sending the signal

print("Sending signal...")

signal.send(sender=None)

print("Signal sent!")

Output:

Sending signal...

Handler started

(after sending the signal 2-second pause due to the time.sleep(2) call)

Handler finished

Signal sent! (will only appear after the handler has completed)

1. Signals sent by Django operate within the thread that sent them. This implies that the code handling the signal executes directly there, within the same thread, whenever a signal is activated. The signal handler will halt all other thread activity until it completes.

Code:

import threading

from django.dispatch import Signal, receiver

import time

signal = Signal()

@receiver(signal)

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

    print("Handler started in thread.")

    time.sleep(3)

    print("Handler finished")

def send\_signal():

    print("Sending signal from thread")

    signal.send(sender=None)

    print("Signal sent!")

# Start a new thread to send the signal

thread = threading.Thread(target=send\_signal)

thread.start()

thread.join()  # Wait for the thread to finish

Output:

Sending signal from thread

Handler started in thread.

Handler finished

Signal sent!

3. Django signals run in the same database transaction as the caller. This means that any changes made to the database within the signal handler will be part of the same transaction and will either be committed or rolled back together with the changes made by the caller. If any error occurs during the execution  the entire transaction will be rolled back.

from django.db import transaction

from django.db import connection

from django.dispatch import Signal, receiver

user\_created = Signal()

@receiver(user\_created)

def log\_user\_creation(sender, \*\*kwargs):

    with connection.cursor() as cursor:

        cursor.execute("INSERT INTO logs (message) VALUES (%s)", ['User created'])

def create\_user(name):

    with transaction.atomic():

        with connection.cursor() as cursor:

            cursor.execute("INSERT INTO users (name) VALUES (%s)", [name])

        user\_created.send(sender=name)

try:

    create\_user("John Doe")

    print("User created successfully.")

except Exception as e:

    print(f"Error occurred: {e}")

Output:

User created successfully.

4.

Code:

class Rectangle:

    def \_\_init\_\_(self, length: int, width: int):

        self.length = length

        self.width = width

    def \_\_iter\_\_(self):

        yield {'length': self.length}

        yield {'width': self.width}

rect = Rectangle(6, 8)

for dim in rect:

    print(dim)

Output:

{'length': 6}

{'width': 8}