Ans1. By default, Django signals are executed synchronously. This means that the signal receiver function is called immediately after the signal is sent, within the same thread of execution.

code: - from django.dispatch import Signal

my\_signal = Signal()

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

print("Signal received!")

my\_signal.connect(my\_receiver)

# Sending the signal

my\_signal.send(sender=None)

# The output will be:

# Signal received!

In this the my\_receiver function is executed immediately after my\_signal.send is called, without any delay or asynchronous processing.

Ans2. By default, Django signals typically run in the same thread as the caller. This means that the signal receiver function is executed within the same thread of execution as the code that sent the signal.

Code: - import threading

def my\_thread\_function():

my\_signal.send(sender=None)

thread = threading.Thread(target=my\_thread\_function)

thread.start()

# The signal will be received in the same thread as the main thread

my\_signal.connect(my\_receiver)

In this, a new thread is created to send the signal. However, when the signal is received, it's still processed in the main thread. This is because Django signals are synchronous by default, and the receiver function is called immediately after the signal is sent.

Ans3. By Default, the Django signals do not run in the same database transaction as the caller. This is to prevent potential deadlocks and ensure that signals are processed independently.

Code: - from django.db import transaction

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

with transaction.atomic():

# Perform database operations within the signal receiver

print("Signal received, performing database operations")

with transaction.atomic():

# Perform database operations in the caller

print("Caller performing database operations")

my\_signal.send(sender=None)

In This, both the caller and the signal receiver are using atomic transactions. However, the transactions are separate, meaning that changes made in one transaction will not be visible to the other until both transactions are committed. If one transaction fails, the other will not be affected.

Ans4: - class Rectangle:

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

self.length = length

self.width = width

def \_\_iter\_\_(self):

yield {'length': self.length}

yield {'width': self.width}