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.

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

from django.dispatch import receiver

from django.db import models

class MyModel(models.Model):

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

@receiver(post\_save, sender=MyModel)

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

print("Signal received!")

print(f"Processing instance: {instance.name}")

def save\_instance():

instance = MyModel(name="Test Instance")

instance.save() # This will trigger the post\_save signal

if \_\_name\_\_ == "\_\_main\_\_":

save\_instance()

2. Do django signals run in the same thread as the caller?

import threading

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

from django.dispatch import receiver

from django.db import models

class MyModel(models.Model):

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

@receiver(post\_save, sender=MyModel)

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

print(f"Signal received in thread: {threading.current\_thread().name}")

def save\_instance():

instance = MyModel(name="Test Instance")

instance.save() # This will trigger the post\_save signal

print(f"Saved instance in thread: {threading.current\_thread().name}")

if \_\_name\_\_ == "\_\_main\_\_":

save\_instance()

3.By default do django signals run in the same database transaction as the caller?

from django.db import models, transaction

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

from django.dispatch import receiver

from django.db import connection

class MyModel(models.Model):

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

class RelatedModel(models.Model):

related\_name = models.CharField(max\_length=100)

@receiver(post\_save, sender=MyModel)

def create\_related\_model(sender, instance, created, \*\*kwargs):

if created:

RelatedModel.objects.create(related\_name=instance.name)

def save\_instance():

with transaction.atomic():

instance = MyModel(name="Test Instance")

instance.save() # This triggers the post\_save signal

if \_\_name\_\_ == "\_\_main\_\_":

save\_instance()

exists = RelatedModel.objects.filter(related\_name="Test Instance").exists()

print(f"Related model exists: {exists}")

4.

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}