Question 1: By default, are Django signals executed synchronously or asynchronously?

import time

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

from django.dispatch import receiver

from django.contrib.auth.models import User

@receiver(post\_save, sender=User)

def signal\_handler(sender, instance, \*\*kwargs):

print(f"Signal handler start: {time.time()}")

time.sleep(2) # Simulate a time-consuming task

print(f"Signal handler end: {time.time()}")

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

print(f"Before save: {time.time()}")

user = User(username="test\_user")

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

print(f"After save: {time.time()}")

Question 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.contrib.auth.models import User

@receiver(post\_save, sender=User)

def signal\_handler(sender, instance, \*\*kwargs):

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

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

print(f"Caller thread: {threading.current\_thread().name}")

user = User(username="test\_user")

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

Question 3: By default, do Django signals run in the same database transaction as the caller?

from django.db import transaction

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

from django.dispatch import receiver

from django.contrib.auth.models import User

@receiver(post\_save, sender=User)

def signal\_handler(sender, instance, \*\*kwargs):

print(f"Signal handler, user saved with ID: {instance.id}")

# Simulating a save inside an atomic block

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

with transaction.atomic():

print("Transaction start")

user = User(username="test\_user")

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

print("Transaction end")

Custom Classes in Python: Rectangle Class with Iterable Behavior

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}

rectangle = Rectangle(10, 5)

for dimension in rectangle:

print(dimension)