Questions and Answers for Django Trainee at Accuknox

Topic: **Django 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 run as synchronous code. This means that when the signal is transmitted, it will wait for the execution of all receivers to finish.
If the signals were asynchronous, then the total time taken for execution would be almost close to 5 seconds (more than one signal execution time)

models.py × test_signals.py test_thread.py settings.py

django_signals > models.py > ...

```python
from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver
import time

class MyModel(models.Model):
    name = models.CharField(max_length=100)

@receiver(post_save, sender=MyModel)
def slow_function(sender, instance, created, **kwargs):
    print(f"Signal receiver started for {instance.name}")
    time.sleep(5)  # Simulate a time-consuming operation
    print(f"Signal receiver finished for {instance.name}")
```

File Edit Selection View Go Run Terminal Help new_assignment

EXPLORER

NEW_ASSIGNMENT
- django_signals
  - __pycache__
  - management\commands
    - __pycache__
    - test_signals.py
  - migrations
  - __init__.py
  - admin.py
  - apps.py
  - models.py
  - tests.py
  - views.py
- new_assignment
  - __pycache__
  - __init__.py
  - asgi.py
  - settings.py
  - urls.py
  - wsgi.py
- db.sqlite3
- manage.py

OUTLINE
TIMELINE

models.py test_signals.py settings.py

django_signals > management > commands > test_signals.py > Command > handle

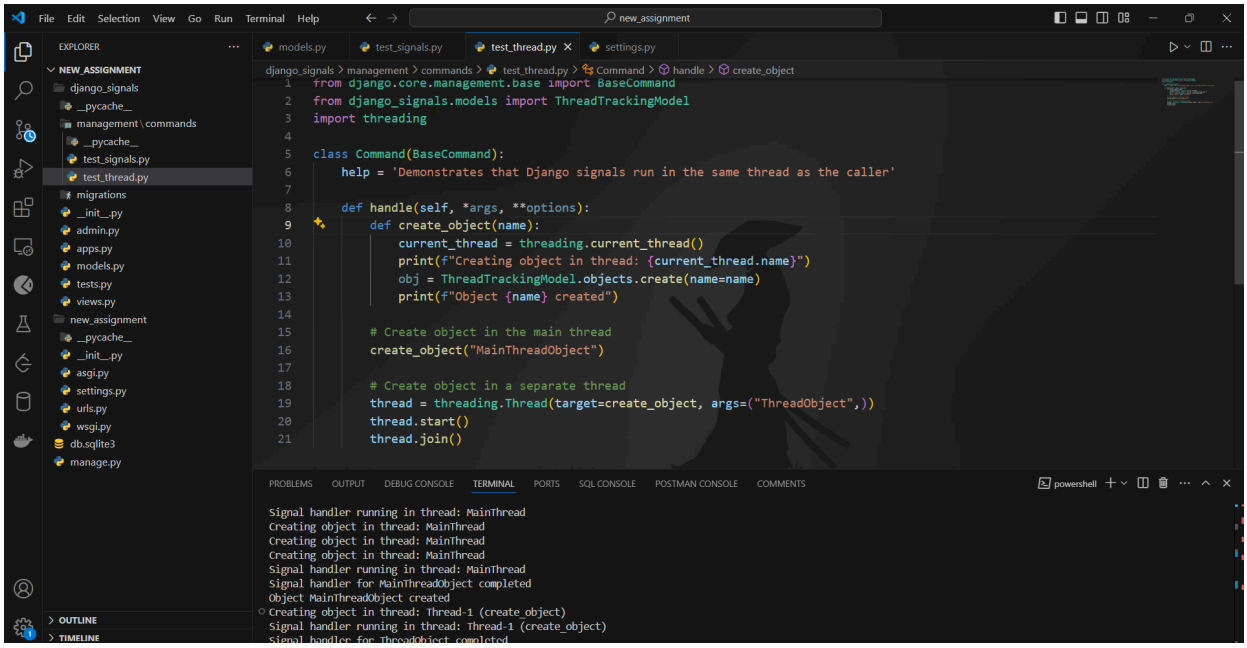
```python
from django.core.management.base import BaseCommand
from django_signals.models import MyModel
import time

class Command(BaseCommand):
    help = 'Demonstration of  synchronous signal execution'

    def handle(self, *args, **options):
        start_time = time.time()

        print("Creating first object")
        obj1 = MyModel.objects.create(name="Object 1")
        print("First object created")

        print("Creating second object")
        obj2 = MyModel.objects.create(name="Object 2")
        print("Second object created")

        end_time = time.time()
        print(f"total execution time: {end_time - start_time} seconds")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE POSTMAN CONSOLE COMMENTS powershell

```
PS C:\Users\ashut\OneDrive\Desktop\ashutosh\internship assignment\new_assignment> python manage.py test_signals
Creating first object
Signal receiver started for Object 1
Signal receiver finished for Object 1
First object created
Creating second object
Signal receiver started for Object 2
Signal receiver finished for Object 2
Second object created
total execution time: 10.025687456130981 seconds
PS C:\Users\ashut\OneDrive\Desktop\ashutosh\internship assignment\new_assignment> s
```

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 by default. This is closely related to their synchronous nature




```python
from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver
import time
import threading

class MyModel(models.Model):
    name = models.CharField(max_length=100)

@receiver(post_save, sender=MyModel)
def slow_function(sender, instance, created, **kwargs):
    print(f"Signal receiver started for {instance.name}")
    time.sleep(5)  # Simulate a time-consuming operation
    print(f"Signal receiver finished for {instance.name}")


class ThreadTrackingModel(models.Model):
    name = models.CharField(max_length=100)

@receiver(post_save, sender=ThreadTrackingModel)
def signal_handler(sender, instance, created, **kwargs):
    current_thread = threading.current_thread()
    print(f"Signal handler running in thread: {current_thread.name}")
    time.sleep(2)  # Simulate some work
    print(f"Signal handler for {instance.name} completed")
```

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, by default, Django signals run in the same database transaction as the caller. This behavior is crucial for maintaining data integrity and consistency.

```python
from django.db import models, transaction
from django.db.models.signals import post_save
from django.core.management.base import BaseCommand
from django.db import IntegrityError
from django_signals.models import SignalBehave, RelatedModel


class Command(BaseCommand):
    help = 'Demonstrates that Django signals run in the same transaction as the caller'

    def handle(self, *args, **options):
        # Scenario 1: Successful transaction
        with transaction.atomic():
            obj = SignalBehave.objects.create(name="Test1")
            print("Main transaction completed")

        # Verify both objects exist
        print(f"SignalBehave count: {SignalBehave.objects.count()}")
        print(f"RelatedModel count: {RelatedModel.objects.count()}")

        # Scenario 2: Failed transaction due to integrity error
        try:
            with transaction.atomic():
                obj = SignalBehave.objects.create(name="Test2")
                print("Object created, now raising IntegrityError")
                raise IntegrityError("Simulated integrity error")
        except IntegrityError:
            print("IntegrityError caught, transaction should be rolled back")


        print(f"SignalBehave count: {SignalBehave.objects.count()}")
        print(f"RelatedModel count: {RelatedModel.objects.count()}")


```

Snipped from >> django_signals/management/commands/3_signal_behave.py

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>}`**

```python
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(5, 3)
for item in rect:
    print(item)
```

Submission Github repo link->> https://github.com/alwaysashutosh/new_assignment