Django Trainee at Accuknox

1. **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.

Ans.

Django signals are executed synchronously by default.

That means when a signal is sent, Django executes all connected receivers immediately (blocking) — the caller must wait until all receivers finish before continuing.

Code :

```
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 slow_receiver(sender, instance, **kwargs):
    print("Signal started...")
    time.sleep(5)
    print("Signal finished...")

# Run this test in Django shell:
from django.contrib.auth.models import User
import time

start = time.time()
User.objects.create(username='test_user')
end = time.time()
```

```
print("Total time:", end - start)
```

Explanation:

- **When we create a user the signal post_save runs the slow_receiver function.**
- **We will see the Total time printed is ~5 seconds proving the signal blocked the main thread until it finished.**
- **Hence signals are synchronous by default.**

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.

Ans.

Yes, by default Django signals run in the same thread as the caller.

Code :

```
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 thread_check(sender, instance, **kwargs):
    print("Signal Thread:", threading.current_thread().name)

# Run in Django shell:
```

```
from django.contrib.auth.models import User
import threading

print("Main Thread:", threading.current_thread().name)
User.objects.create(username='thread_test')
```

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.

Ans.

Yes — by default, Django signals execute within the same database transaction as the caller.
That means if the transaction rolls back, the effects inside the signal handler are also rolled back.

Code :

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
from django.db import transaction, models

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.CharField(max_length=100, default='')

@receiver(post_save, sender=User)
def create_profile(sender, instance, **kwargs):
    print("Signal creating Profile...")
```

```
    Profile.objects.create(user=instance, bio="Created in signal")


# Run this in Django shell:
from django.contrib.auth.models import User
from django.db import transaction
from app.models import Profile


try:
    with transaction.atomic():
        user = User.objects.create(username='rollback_test')
        raise Exception("Rollback!")
except:
    pass


print("Profiles count:", Profile.objects.count())
```

Explanation:

- **Even though the signal successfully created a Profile inside create_profile(), the outer transaction.atomic() rolled back everything.**
- **This proves signals run in the same DB transaction by default.**

Topic: Custom Classes in Python

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}

# Example usage:
r = Rectangle(10, 5)
for item in r:
    print(item)
```