Questions for Django Trainee at Accuknox

Topic: **Django Signals**

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.

   Django signals execute **synchronously** by default. This means that once a signal is triggered, the execution flow waits for the signal's associated function to complete before moving forward.

   Code:

```python
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 sample_signal(sender, instance, **kwargs):
    print("Signal execution started...")
    time.sleep(4)  # Delay to observe sync execution order
    print("Signal execution completed.")

# Creating a user
new_user = User.objects.create(username="demo_user", password="test123")
print("User creation function executed.")
```

   O/P:

```
Signal execution started...
(Waits for 4 seconds)
Signal execution completed.
User creation function executed.
```

   Justification:

   - If Django signals were asynchronous, the final print statement would execute immediately while the signal ran separately.
   - Instead, we see that execution is **paused** while the signal function runs, confirming its synchronous nature.

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.

   Yes, Django signals run **within the same thread** as the function that triggers them.

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 check_thread(sender, instance, **kwargs):
    print(f"Signal is running in thread: {threading.current_thread().name}")

# Checking the main thread before calling signal
print(f"Main execution thread: {threading.current_thread().name}")

# Creating a new user to trigger the signal
User.objects.create(username="thread_test_user", password="securepass")
```

O/P:

```
Main execution thread: MainThread
Signal is running in thread: MainThread
```

Justification:

Since both messages confirm the same thread name (MainThread), we can conclude that Django signals do not create separate threads unless explicitly configured to do so.

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.

Yes, Django signals operate **within the same database transaction** as the function that initiates them.

Code:

```
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 transaction_check(sender, instance, **kwargs):
    print(f"Inside signal - Is transaction autocommit enabled?
{transaction.get_autocommit()}")

# Starting a transaction block
try:
    with transaction.atomic():
        user_instance = User.objects.create(username="rollback_user",
password="pass123")
        print(f"Inside transaction block - Is autocommit enabled?
{transaction.get_autocommit()}")
        raise ValueError("Triggering rollback!")  # Rolling back transaction
on purpose
```

MANISHKUMAR

```
except:
    pass

# Checking if the user is present in the database
print(f"User exists in database:
{User.objects.filter(username='rollback_user').exists()}")
```

O/P:

```
Inside transaction block - Is autocommit enabled? False
Inside signal - Is transaction autocommit enabled? False
User exists in database: False
```

Justification:

- The signal is executed **before** the transaction commits.
- Since an exception is raised, the entire transaction (including changes made by the signal) is **rolled back**.
- The database query at the end confirms that no new user was added.

Topic: Custom Classes in Python

```
class Rectangle:
    def __init__(self, length: int, width: int):
        self.length = length
        self.width = width
        self.dimensions = [{"length": self.length}, {"width": self.width}]

    def __iter__(self):
        return iter(self.dimensions)

rect_obj = Rectangle(15, 8)

for value in rect_obj:
    print(value)
```

O/P:

```
{'length': 15}
{'width': 8}
```