Questions for Django Trainee at Accuknox

Topic: Django Signals

Question 1: By default are jango 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 are executed **synchronously**. When a signal is sent, the receiver function connected to the signal is called immediately and executed within the same thread.

To demonstrate this, we can create a signal and a signal receiver function that adds some delay using time.sleep(). If the execution is synchronous, the delay in the signal handler will block the execution of the main thread.

```
#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 user_saved_handler(sender, instance, **kwargs):
    print("Signal handler starts execution.")
    time.sleep(5)  # Simulate a delay
    print("Signal handler finishes execution.")

def create_user():
    print("Starting user creation.")
    user = User.objects.create(username='testuser')
    print("User created.")
```

```
# Now run the create_user function:

create_user()
```

Output:

```
Starting user creation.

User created.

Signal handler starts execution.

(5 second delay)

Signal handler finishes execution.
```

Question 2: Do jango 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. Django signals do not spawn new threads for signal handling, meaning the signal handler executes in the same thread that sent the signal.

To prove this, we can print the thread ID of both the caller function and the signal handler function using threading.get_ident().

```
#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 user_saved_handler(sender, instance, **kwargs):

    print(f"Signal handler thread ID: {threading.get_ident()}")


def create_user():

    print(f"Caller thread ID: {threading.get_ident()}")

    user = User.objects.create(username='testuser')
```

```
# Now run the create_user function:
create_user()
```

Output:

```
Caller thread ID: 140161149228800
Signal handler thread ID: 140161149228800
```

Question 3: By default do jango 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. If the transaction is rolled back, the changes made by the signal handler (if any) will also be rolled back.

To demonstrate this, we can create a custom model, register a signal, and try saving the model within a database transaction. If we trigger an exception after the save, the transaction will roll back, and the changes made by the signal handler should also be rolled back.

```
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

from myapp.models import Profile


@receiver(post_save, sender=User)

def create_profile(sender, instance, **kwargs):

    print("Signal handler creating Profile.")

    Profile.objects.create(user=instance)


def create_user_with_transaction():
    try:
        with transaction.atomic():
            print("Starting transaction.")
            user = User.objects.create(username='testuser')
            print("User created.")
            raise Exception("Simulating an error!")  # This will roll back the transaction
    except Exception as e:
        print(f"Transaction rolled back due to: {e}")

# Now run the create_user_with_transaction function:
create_user_with_transaction()
```

Output:

```
Starting transaction.

User created.

Signal handler creating Profile.

Transaction rolled back due to: Simulating an error!
```

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>}**

Solution :

```
class Rectangle:
    def __init__(self, length: int, width: int):
        self.length = length
        self.width = width
        self._index = 0  # To keep track of the iteration state
    def __iter__(self):
        # Reset index whenever we start iterating

        self._index = 0

        return self


    def __next__(self):

        if self._index == 0:

            self._index += 1

            return {'length': self.length}

        elif self._index == 1:

            self._index += 1

            return {'width': self.width}

        else:

            # Once both values have been returned, we stop the iteration

            raise StopIteration


# Example usage:
rect = Rectangle(20, 10)
```

```
# Iterating over the Rectangle instance
for value in rect:
    print(value)
```

Output :

```
{'length': 20}
{'width': 10}
```