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:

Django signals are executed synchronously by default. This signifies that the signal handler will run immediately after the signal is sent, in the same context as the sender.

Here's a code snippet to show synchronous execution:

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import MyModel

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("Signal received !!!")
    time.sleep(5)
    print("Signal Complete !!!")
```

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 are by default run on the same thread as the caller. However, Django signals do not need to run on the same thread as the caller. This indicates that the signal handler method will be run on the same thread as the code that generated the signal by default.

Below is a simple code snippet to demonstrate this:

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from threading import current_thread
from .models import MyModel

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("Signal handler thread:", current_thread().name)

MyModel.objects.create(name="Example")
```

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.

Answers:

Yes, by default, Django signals run in the same database transaction as the caller. This means that if the caller's transaction is rolled back, the signal's changes will also be rolled back.

Below is the code snippet that demonstrates signals run in the same database transaction as the caller:

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

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

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    instance.name = "Modified in Signal"
    instance.save()

def create_model_with_transaction():
    try:
        with transaction.atomic():
            obj = MyModel.objects.create(name="Original")
            print(f"Name after signal: {obj.name}")
            raise Exception("Forcing rollback!")
    except Exception:
        pass

    # Verify if the modification was rolled back
    if MyModel.objects.filter(name="Modified in Signal").exists():
        print("Signal changes persisted!")
    else:
        print("Transaction rolled back, no signal changes.")

create_model_with_transaction()
```