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 :The django signals are executed synchronously.This means that when a signal is sent, the signal handlers are executed immediately, in the same thread, and before control returns to the caller.

Code Snippet

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

# Receiver function for post_save signal
@receiver(post_save, sender=User)
def user_saved(sender, instance, **kwargs):
    print("Signal received for user:", instance.username)
    print("Starting long task...")
    time.sleep(5)  # Simulate a long task
    print("Task completed!")

# Simulating a user save operation
if __name__ == "__main__":
    user = User(username="testuser")
    user.save()  # This will trigger the post_save signal
    print("User save operation completed")
```

Output:

```
Signal received for user: testuser
Starting long task...
Task completed!
User save operation completed
```

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. When a signal is sent, the receiver is executed synchronously within the same thread that triggered the signal.

Code snippet :

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

# Receiver function for post_save signal
@receiver(post_save, sender=User)
def user_saved(sender, instance, **kwargs):
    print("Signal received for user:", instance.username)
    print("Signal is running in thread:", threading.current_thread().name)

# Simulating a user save operation
if __name__ == "__main__":
    print("Main thread is:", threading.current_thread().name)
    user = User(username="testuser")
    user.save()  # This will trigger the post_save signal
```

Output:

```
Main thread is: MainThread
Signal received for user: testuser
Signal is running in thread: MainThread
```

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 run in the **same database transaction** as the caller, provided the signal is connected to an event that occurs within a database transaction

Code Snippet :

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

# Custom model to track log of operations
class Log(models.Model):
    message = models.CharField(max_length=255)

# Receiver function for post_save signal
@receiver(post_save, sender=User)
```

```
def user_saved(sender, instance, **kwargs):
    print("Signal received for user:", instance.username)
    # Create a log entry when the signal is received
    Log.objects.create(message=f"User {instance.username} saved in signal")

# Simulating a user save operation inside a transaction
if __name__ == "__main__":
    try:
        with transaction.atomic():
            user = User(username="testuser")
            user.save()  # This will trigger the post_save signal
            print("User saved. Raising exception to trigger rollback...")
            raise Exception("Rolling back transaction!")
    except Exception as e:
        print(f"Exception caught: {e}")

    # Check if the log entry was committed or rolled back
    logs = Log.objects.all()
    print("Logs count:", logs.count())  # Expecting 0 if rolled back
```

Output:

```
Signal received for user: testuser
User saved. Raising exception to trigger rollback...
Exception caught: Rolling back transaction!
Logs count: 0
```