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:
By default, Django signals are executed **synchronously**. When a signal is emitted, the handler runs immediately in the same execution flow, meaning it blocks the caller until the handler finishes execution.

Code Snippet:

```
import time

from django.dispatch import receiver, Signal

# Custom signal

my_signal = Signal()

# Signal handler (to prove synchronous execution)

@receiver(my_signal)

def synchronous_handler(sender, **kwargs):

    print("Signal handler started")

    time.sleep(3)  # Simulating a delay

    print("Signal handler finished")

# Trigger the signal

my_signal.send(sender=None)

print("Signal sent")
```

Explanation:

- The `time.sleep(3)` simulates a long-running task.
- If the signal was asynchronous, the `"Signal sent"` message would be printed immediately, and the handler would complete in the background.
- Since the output shows `"Signal handler finished"` before `"Signal sent"`, it proves that the signal is synchronous.

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 run in the **same thread** as the caller by default.

Code Snippet:

```
import threading
from django.dispatch import receiver, Signal

# Custom signal
my_signal = Signal()

# Signal handler to check thread name
@receiver(my_signal)
def thread_check_handler(sender, **kwargs):
    print(f"Signal handler thread: {threading.current_thread().name}")

# Trigger the signal
print(f"Caller thread: {threading.current_thread().name}")
my_signal.send(sender=None)
```

Explanation:

- We print the name of the thread inside the signal handler (`thread_check_handler`) and compare it to the thread name in the caller.
- If the signal runs in a different thread, the thread names would differ.

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.

Answer:
Yes, by default, Django signals run in the **same database transaction** as the caller. If a signal is emitted within a transaction, and that transaction fails, the signal's changes are also rolled back.

Code Snippet:

```
from django.db import transaction
from django.dispatch import receiver, Signal

# Custom signal
my_signal = Signal()

# Signal handler to demonstrate transaction rollback
@receiver(my_signal)
def transaction_handler(sender, **kwargs):
    print("Signal handler inside transaction")

# Function to test transaction behavior
@transaction.atomic
def transactional_operation():
    print("Transaction started")
    my_signal.send(sender=None)  # Trigger signal inside transaction
    raise Exception("Simulated error to rollback transaction")
    print("Transaction finished")

# Trigger the function
try:
    transactional_operation()
except Exception as e:
    print(f"Transaction rolled back due to: {e}")
```

Explanation:

- We use `@transaction.atomic` to ensure the code runs inside a transaction.
- The exception raised after the signal is sent will trigger a rollback.
- If the signal handler was executed outside the transaction, it wouldn't be affected by the