Problem
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:
By default, Django signals are executed synchronously. This means that when a signal is sent, the connected signal handlers are executed immediately in the same thread and process as the sender.

Here's a code snippet to demonstrate the synchronous execution of Django signals:

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

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

@receiver(post_save, sender=MyModel)
def my_handler(sender, instance, **kwargs):
    print("Signal handler started")
    # Simulate some time-consuming operation
    time.sleep(5)
    print("Signal handler completed")

# Create an instance of MyModel and save it
print("Creating instance of MyModel...")
instance = MyModel(data="example")
print("Saving instance...")
instance.save()
print("Instance saved")
```

In this code:

1.We define a Django model MyModel with a data field.
2.We define a signal handler function my_handler which is connected to the post_save signal of MyModel. Inside the signal handler, we simulate a time-consuming operation using time.sleep(5).
3.When we create an instance of MyModel and save it, the post_save signal is sent, triggering the execution of the signal handler my_handler.
4.The output of the code will demonstrate that the signal handler is executed synchronously, as the "Signal handler started" message will appear before the "Signal handler completed" message, and there will be a delay of approximately 5 seconds between them.

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, Django signals run in the same thread as the caller by default. This means that when a signal is sent, the connected signal handlers are executed in the same thread and process as the sender.

To demonstrate this, let's use a simple code snippet:

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

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

@receiver(post_save, sender=MyModel)
def my_handler(sender, instance, **kwargs):
    print(f"Signal handler executed in thread: {threading.current_thread().name}")

# Create an instance of MyModel and save it
print(f"Main thread: {threading.current_thread().name}")
print("Creating instance of MyModel...")
instance = MyModel(data="example")
print("Saving instance...")
instance.save()
print("Instance saved")
```

In this code:

1.We define a Django model MyModel with a data field.
2.We define a signal handler function my_handler which is connected to the post_save signal of MyModel.
3.Inside the signal handler, we print the name of the current thread using threading.current_thread().name.
4.In the main section of the code, before creating and saving an instance of MyModel, we also print the name of the current thread.
When you run this code, it will demonstrate that the signal handler executes in the same thread as the main code. This proves that Django signals run in the same thread as the caller by default.

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:

By default, Django signals run in the same database transaction as the caller. This means that when a signal is sent during a database transaction, the connected signal handlers are executed within the same transaction context.

To demonstrate this, let's use a code snippet:

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

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

@receiver(post_save, sender=MyModel)
def my_handler(sender, instance, **kwargs):
    print("Signal handler executed")

# Create an instance of MyModel and save it within a transaction
print("Creating instance of MyModel...")
with transaction.atomic():
    instance = MyModel(data="example")
    print("Saving instance within transaction...")
    instance.save()
    print("Instance saved within transaction")
print("Transaction committed")
```

In this code:

1.We define a Django model MyModel with a data field.
2.We define a signal handler function my_handler which is connected to the post_save signal of MyModel.
3.We create an instance of MyModel and save it within a database transaction using transaction.atomic().
4.Inside the signal handler, we print a message indicating that the signal handler has been executed.
When you run this code, it will demonstrate that the signal handler is executed within the same database transaction as the caller. This proves that Django signals run in the same database transaction context as the caller by default.