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. This means when the signal is called Django waits for its complete execution before going forward with the next requests. The signal handler runs in the same thread and blocks the further execution of the code.

Code Snippet:

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

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print(f"Signal received for instance: {instance.name}")
    time.sleep(3)  # delay, to check synchronous
    print("Signal completed")

if __name__ == "__main__":
    # Creating a instance of the model
    print("Starting save operation:", datetime.now())
    obj = MyModel(name="New Object")
    obj.save()  #triggering the post_save signal
    print("Save operation completed:", datetime.now())
```

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, by default Django signals run in the same thread as the caller. This means when a signal is triggered, the signal handler runs in the same thread thus blocking the further execution until the execution completes.

Code Snippet:

```
import threading

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

# Signal handler to print the current thread
@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print(f"Signal handler running in thread: {threading.current_thread().name}")
    time.sleep(2)  # some task
    print("Signal handler finished execution")

# Example usage to confirm thread behavior
if __name__ == "__main__":
    print(f"main caller running in thread: {threading.current_thread().name}")

    # Create a new instance of the model
    obj = MyModel(name="Test Object")
    print("Saving model instance...")
    obj.save()  # This will trigger the post_save signal
    print("Save operation completed")
```

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 triggered during a database transaction the signal handler is executed within the same transaction context. If the transaction is rolled back, the changes made by the signal handler are also rolled back.

Code Snippet:

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

# Log model to log actions triggered by the signal
class LogEntry(models.Model):
    message = models.CharField(max_length=255)

# Signal handler that creates a log entry after a MyModel instance is saved
@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("Signal handler triggered, creating log entry.")
    LogEntry.objects.create(message=f"Saved MyModel instance: {instance.name}")

# Test transaction behavior with signal
if __name__ == "__main__":
    # Start a transaction
    try:
        with transaction.atomic():
            print("Starting transaction")

            # Create a new MyModel instance (this triggers the signal)
            obj = MyModel(name="Test Transaction")
            obj.save()  # Triggers the post_save signal and the LogEntry creation

            # After saving, raise an exception to roll back the transaction
            raise Exception("Rolling back the transaction")

    except Exception as e:
        print(f"Transaction rolled back: {e}")

    # Check if the LogEntry created by the signal handler was rolled back
    try:
        log_entry = LogEntry.objects.get(message="Saved MyModel instance: Test Transaction")
```

```
    print(f"LogEntry found: {log_entry.message}")
except ObjectDoesNotExist:
    print("LogEntry was rolled back with the transaction")
```