ACCUKNOX DJANGO TRAINEE ASSIGNMENT

Question 1: By default, are Django signals executed synchronously or asynchronously?

A: By default, Django signals are executed synchronously. This means that when a signal is triggered, the signal handler (receiver) is executed immediately in the same thread as the triggering event.

To prove this, we can use a simple code snippet that demonstrates the synchronous nature of Django signals. We can use Django's built-in post_save signal and simulate a delay in the receiver to show that the execution waits for the signal handler to complete.

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

# Define a simple model
class MyModel(models.Model):
    name = models.CharField(max_length=100)

# Signal receiver that simulates a time-consuming task
@receiver(post_save, sender=MyModel)
def my_signal_receiver(sender, instance, **kwargs):
    print("Signal received! Processing...")
    time.sleep(5)  # Simulate a delay
    print("Signal processing finished.")

# Example usage
if __name__ == "__main__":
    # Create and save an instance of MyModel
    my_instance = MyModel(name="Test Instance")
    my_instance.save()
    print("Model save complete.")
```

When my_instance.save() is called, the post_save signal is triggered. The my_signal_receiver function receives the signal, prints a message, and then simulates a 5-second delay. The message "Model save complete." will not be executed until the signal handler has finished its work, demonstrating that the signal is processed synchronously.

Question 2: Do Django signals run in the same thread as the caller?

A: Yes, Django signals run in the same thread as the caller by default. This means that when a signal is triggered, the signal handler (receiver) is executed in the same thread that triggered the signal. This is why signals are synchronous by default unless explicitly made as asynchronous.

Since Django signals operate in the same thread, any time-consuming tasks within the signal receiver can block the caller thread until the task is completed.

Example:

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

# Define a simple model
class MyModel(models.Model):
    name = models.CharField(max_length=100)

# Signal receiver that prints the current thread
@receiver(post_save, sender=MyModel)
def my_signal_receiver(sender, instance, **kwargs):
    print(f"Signal running in thread: {threading.current_thread().name}")

# Example usage
if __name__ == "__main__":
    print(f"Caller running in thread: {threading.current_thread().name}")
    my_instance = MyModel(name="Test Instance")
    my_instance.save()
```

Both the caller and the signal receiver print the current thread name. Since both the caller and signal are executed in the same thread (MainThread), this proves that Django signals run in the same thread as the caller.

Question 3: By default, do Django signals run in the same database transaction as the caller?

A: Yes, by default, Django signals run in the same database transaction as the caller. This means if an error occurs in the signal handler, it can roll back the transaction initiated by the caller. To demonstrate this, we can use Django's transaction.atomic() to test if the signal affects the database transaction.

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

# Define a simple model
class MyModel(models.Model):
    name = models.CharField(max_length=100)

# Signal receiver that raises an exception
@receiver(post_save, sender=MyModel)
def my_signal_receiver(sender, instance, **kwargs):
    print("Signal received, raising exception...")
    raise Exception("Error in signal!")

# Example usage
try:
    with transaction.atomic():
        my_instance = MyModel(name="Test")
        my_instance.save()  # Signal will trigger
        print("Instance saved.")
except Exception as e:
    print(f"Transaction rolled back due to: {e}")
```

The my_signal_receiver raises an exception when the post_save signal is triggered. Since the signal runs in the same transaction, the exception causes the transaction to roll back. This proves that the signal runs in the same database transaction as the caller by default.