

# **Questions for Django Trainee at Accuknox**

## [Topic: **Django Signals**](https://docs.djangoproject.com/en/3.2/topics/signals/)

**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 that when a signal is sent, the connected signal handlers run immediately in the same thread that triggered the signal.

Lets Understand by following example

**from django.dispatch import Signal, receiver**

**import time**

**my\_signal = Signal()**

**# Signal handler**

**@receiver(my\_signal)**

**def my\_signal\_handler(sender, \*\*kwargs):**

**print("Signal handler started.")**

**time.sleep(5) # Simulate a long-running process**

**print("Signal handler finished.")**

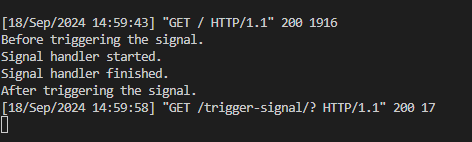
**def trigger\_signal():**

**print("Before signal.")**

**my\_signal.send(sender=None) # Trigger the signal**

**print("After signal.")**

**trigger\_signal()**



**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.

Yes, Django signals run in the same thread as the caller by default. Since signals are executed synchronously, they will be executed in the same thread that triggers the signal.

For Example:

**from django.dispatch import Signal, receiver**

**import threading**

**my\_signal = Signal()**

**# Signal handler**

**@receiver(my\_signal)**

**def my\_signal\_handler(sender, \*\*kwargs):**

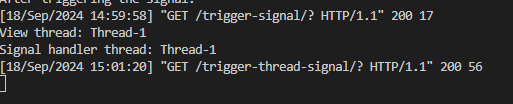
**print(f"Signal handler running in thread: {threading.current\_thread().name}")**

**def trigger\_signal():**

**print(f"Trigger function running in thread: {threading.current\_thread().name}")**

**my\_signal.send(sender=None) # Trigger the signal**

**trigger\_signal()**



**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.

By default, Django signals do not run in the same database transaction as the caller. Django signals are executed immediately after being sent, regardless of whether the transaction that triggered the signal is committed or rolled back.

For Example:

**# models.py**

**from django.db import models**

**class TestModel(models.Model):**

**name = models.CharField(max\_length=100)**

**# signals.py**

**from django.db import transaction**

**from django.dispatch import Signal, receiver**

**from .models import TestModel**

**my\_signal = Signal()**

**# Signal handler that creates a record**

**@receiver(my\_signal)**

**def my\_signal\_handler(sender, \*\*kwargs):**

**TestModel.objects.create(name="Created by signal")**

**print("Signal handler executed and record created.")**

**# views.py**

**from django.db import transaction**

**from .models import TestModel**

**from .signals import my\_signal**

**def trigger\_signal():**

**try:**

**with transaction.atomic(): # Open a transaction block**

**print("Before signal.")**

**TestModel.objects.create(name="Created by function")**

**my\_signal.send(sender=None) # Trigger the signal**

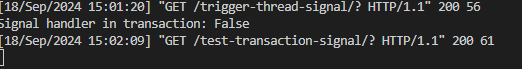
**print("Rolling back transaction.")**

**raise Exception("Rolling back the transaction")**

**except:**

**pass**

**trigger\_signal()**



## Topic: Custom Classes in Python

**Description:** You are tasked with creating a Rectangle class with the following requirements:

1. An instance of the Rectangle class requires length:int and width:int to be initialized.
2. We can iterate over an instance of the Rectangle class
3. When an instance of the Rectangle class is iterated over, we first get its length in the format: **{'length': <VALUE\_OF\_LENGTH>}** followed by the width **{width: <VALUE\_OF\_WIDTH>}**

* Answer: First we will create a rectangle class with parameterized constructor as length and width   
  Then define method \_\_iter\_\_() his makes the class iterable, allowing the user to loop through the instance using a for loop.
* It yields two values a dictionary {'length': self.length} representing the length of the rectangle and a dictionary {'width': self.width} representing the width of the rectangle.
* Now create a object or instance of rectangle class
* Use for loop to iterate over the object of rectangle class this will call \_\_iter\_\_() method that will print the following output  
  {'length': 10} {'width': 5}

**class Rectangle:**

**def \_\_init\_\_(self, length: int, width: int):**

**self.length = length**

**self.width = width**

**def \_\_iter\_\_(self):**

**# Define an iterator to yield length and width in the specified format**

**yield {'length': self.length}**

**yield {'width': self.width}**

**# Example usage:**

**rect = Rectangle(10, 5)**

**# Iterating over the instance**

**for dimension in rect:**

**print(dimension)**

