ACCUKNOX
Assignment

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 triggered, all connected receiver functions are executed immediately within the same process. The main thread will wait until each receiver has finished executing before it continues, which can be observed through the following example.
Code Example
Model Definition:We'll create a simple model in models.py to trigger the post_save signal.

```
from django.db import models
class MyModel(models.Model):
    name = models.CharField(max_length=100)
```

Signal Handler:
A signal receiver is defined to log messages before and after a delay to simulate synchronous execution.

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import MyModel
import time
@receiver(post_save, sender=MyModel)
def my_model_saved(sender, instance, **kwargs):
    print("Signal handler started.")
    time.sleep(2)  # Simulates a 2-second delay
    print("Signal handler finished.")
```

Testing the Signal Execution:Create MyModel and observe the execution order and timing.

```
import os
```

```
import django
import time
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'django_signals_example.settings')
django.setup()
from myapp.models import MyModel
print("Creating MyModel instance...")
start_time = time.time()
my_model_instance = MyModel.objects.create(name='Test Model')
end_time = time.time()
print(f"Instance created in {end_time - start_time:.2f} seconds.")
```

Expected Output: When this code is executed, you should see the following output:

```
Creating MyModel instance...
Signal handler started.
(2-second delay)
Signal handler finished.
Instance created in 2.00 seconds.
```

This output shows that the total time taken to save the MyModel instance includes the delay from the signal handler, proving that Django signals are synchronous by default and that the main thread waits for the signal handler to complete.

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. This means that when a signal is emitted, the receiver functions execute in the same thread context as the code that triggered the signal. This behavior can be demonstrated with a simple example that shows the thread information during the execution of the signal handler.

Model Definition:

```
from django.db import models
```

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

Signal Handler:

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

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

Testing the Signal Execution:

```
import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'django_signals_example.settings')
django.setup()

from myapp.models import MyModel

print(f"Creating MyModel instance in thread:
{threading.current_thread().name}")
my_model_instance = MyModel.objects.create(name='Test Model')
```

Expected Output:

Creating MyModel instance in thread: MainThread
Signal handler executed in thread: MainThread
This output confirms that both the code that creates the instance and the signal handler run in the same thread (MainThread).

Conclusion:This demonstrates that Django signals operate within the same thread context as the caller, meaning any signal handlers will run in the same thread that triggered the 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.

Answer: Yes, by default, Django signals run in the same database transaction as the caller. This means that any database changes made within a signal receiver will be included in the same transaction as the code that triggered the signal. If the transaction is rolled back, the changes made by the signal receiver will also be reverted.

Code Example

Model Definition: model that will trigger the post_save signal.

```
from django.db import models
class MyModel(models.Model):
    name = models.CharField(max_length=100)
```

Signal Handler: We'll define a signal receiver that attempts to create another model instance. If an error occurs, it will raise an exception to simulate a rollback.

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

class AnotherModel(models.Model):
    description = models.CharField(max_length=100)

@receiver(post_save, sender=MyModel)
def create_another_model(sender, instance, **kwargs):
    # Intentionally raise an exception to test rollback
    raise ValueError("Triggering rollback!")
    AnotherModel.objects.create(description=f"Created for {instance.name}")
```

Testing the Transaction:We'll create an instance of MyModel within a transaction block to see if the changes are rolled back.

```
import os
import django
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'django_signals_example.settings')
django.setup()
from myapp.models import MyModel
try:
    with transaction.atomic():
        my_model_instance = MyModel.objects.create(name='Test Model')
        print("MyModel instance created.")
```

```
except Exception as e:
    print(f"An error occurred: {e}")
```

Expected Output: When you run this code, you should see output like the following:
An error occurred: Triggering rollback!
Since an exception was raised in the signal handler, the transaction was rolled back. As a result, the MyModel instance was not saved to the database.

Database Check: After running the test, you can check the database to confirm that the MyModel instance does not exist:

```
print(MyModel.objects.all())  # Should return an empty queryset
```
Conclusion:

This demonstrates that Django signals run within the same database transaction as the caller. Any database operations performed by signal handlers are subject to the same transaction rules, including rollbacks.

Topic: Custom Classes in Python

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

An instance of the Rectangle class requires length:int and width:int to be initialized.
We can iterate over an instance of the Rectangle class
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:
```
class Rectangle:
    def __init__(self, length: int, width: int):
        self.length = length
        self.width = width

    def __iter__(self):
        # Yield length and width in specified dictionary format
        yield {'length': self.length}
        yield {'width': self.width}
```

```
# Example usage
rect = Rectangle(10, 5)

# Iterating over the instance
for attribute in rect:
    print(attribute)
```

Example Usage: When iterating over rect, it prints:
{'length': 10}
{'width': 5}