

Django Trainee at Accuknox

Name: Priya Yallappa Shingri

priyashingri107@Gmail.com

Topic: Django 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.

Solution:

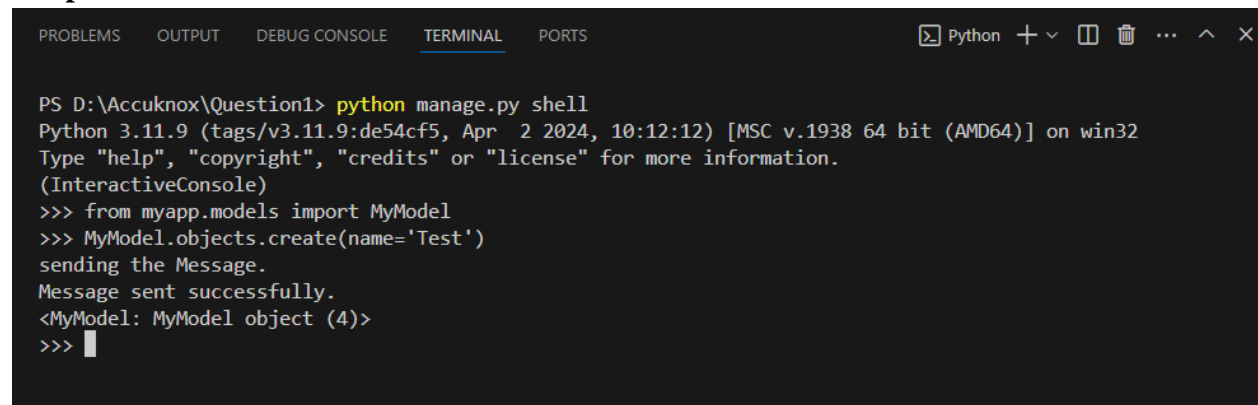
By default, **Django signals are executed synchronously**. This means that when a signal is triggered, Django waits for the signal handler to finish before moving on to the next task.

Example: File name: Question1/myapp/signals.py

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

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("sending the Message.")
    time.sleep(5) # time sleep for 5 sec
    print("Message sent successfully.")
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - [ ] [ ] ... ^ x

PS D:\Accuknox\Question1> python manage.py shell
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from myapp.models import MyModel
>>> MyModel.objects.create(name='Test')
sending the Message.
Message sent successfully.
<MyModel: MyModel object (4)>
>>> █
```

When a MyModel instance is saved, the post_save signal triggers the my_signal_handler function. Here's what happens:

Django and Python solutions

1. **Signal Triggered:** When an instance of MyModel is saved, the message "sending the Message." is printed immediately.
2. **Time Delay:** The `time.sleep(5)` pauses the function for 5 seconds.
3. **Signal Completion:** After the delay, the message "Message sent successfully." is printed.

This shows that the signal runs synchronously because it waits 5 seconds before completing the process.

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.

Solution:

By default, **Django signals are executed synchronously**. This means that when a signal is triggered, Django waits for the signal handler to finish before moving on to the next task.

Example: File name: Question2/myapp/signals.py

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

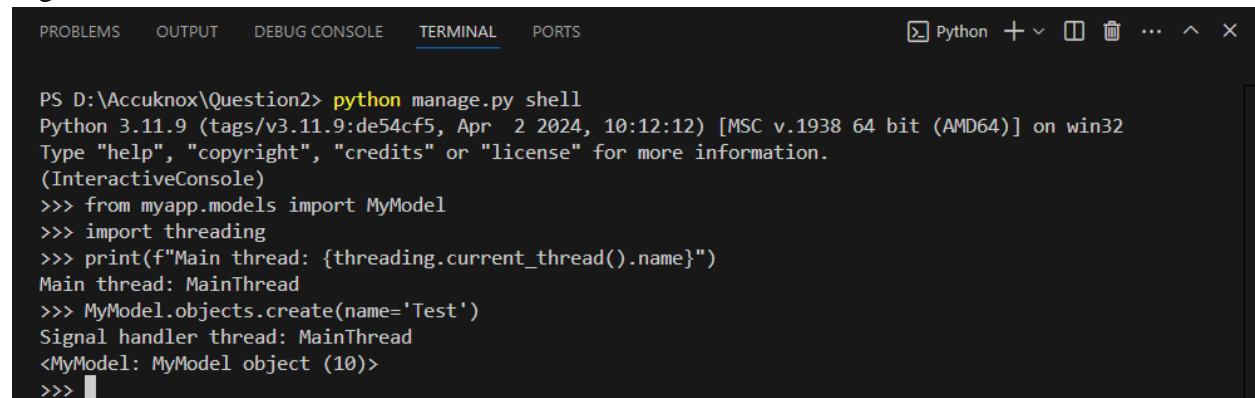
@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print(f"Signal handler thread: {threading.current_thread().name}")
```

Output:

When you create a MyModel instance, the output is,

Main thread: MainThread

Signal handler thread: MainThread



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - [] [X] ... ^ X

PS D:\Accuknox\Question2> python manage.py shell
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from myapp.models import MyModel
>>> import threading
>>> print(f"Main thread: {threading.current_thread().name}")
Main thread: MainThread
>>> MyModel.objects.create(name='Test')
Signal handler thread: MainThread
<MyModel: MyModel object (10)>
>>> 
```

- **Main thread: MainThread:** This shows the thread name of the main process where you executed the command.
- **Signal handler thread: MainThread:** This indicates that the signal handler is running in the same thread as the main process, confirming that Django signals are executed synchronously in the same thread as the caller.

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.

Solution:

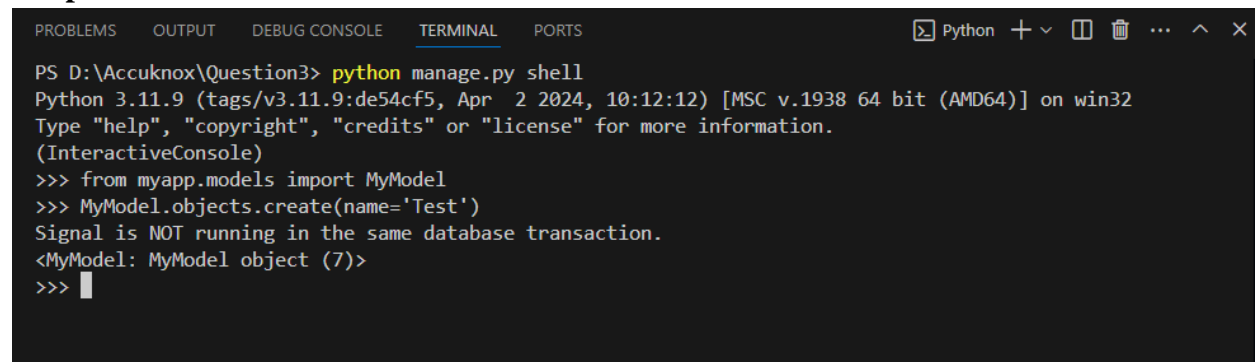
By default, **Django signals connected to the post_save signal do not run in the same database transaction as the caller**. This means that the signal handler operates outside of the transaction block of the model save operation.

Example: File name: Question3/myapp/signals.py

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

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    if transaction.get_connection().in_atomic_block:
        print("Signal is running in the same database transaction.")
    else:
        print("Signal is NOT running in the same database transaction.")
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - [] [X] ... ^ X

PS D:\Accuknox\Question3> python manage.py shell
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from myapp.models import MyModel
>>> MyModel.objects.create(name='Test')
Signal is NOT running in the same database transaction.
<MyModel: MyModel object (7)>
>>> █
```

- `transaction.get_connection().in_atomic_block` checks if the database connection is within an atomic transaction block.
- The signal handler prints a message indicating whether it is running within the same transaction as the model save operation.

Topic: Custom Classes in Python

Description:

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

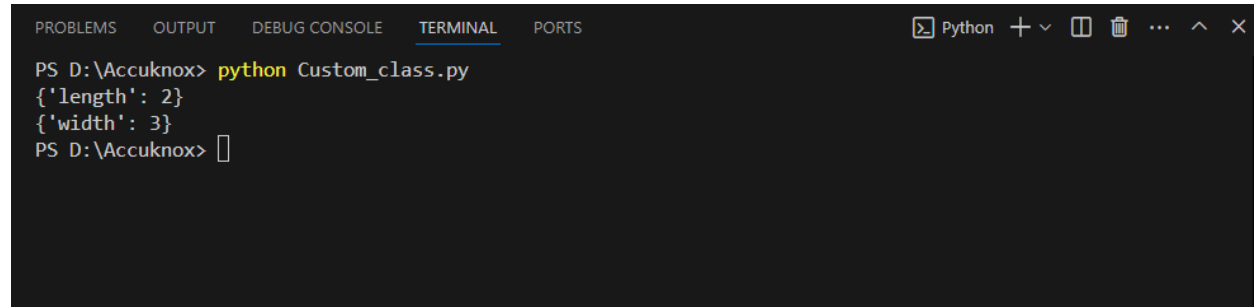
1. An instance of the Rectangle class requires length:int and width:int to be initialized.
2. We can iterate over an instance of the Rectangle class
3. When an instance of the Rectangle class is iterated over, we first get its length in the format: {'length': <VALUE_OF_LENGTH>} followed by the width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

Solution: Filename: Custom_class.py

```
class Rectangle:
    def __init__(self, length: int, width: int):
        #initialisation
        self.length = length
        self.width = width
    def __iter__(self):
        # Define the iteration
        self._iter_data = [
            {'length': self.length},
            {'width': self.width}
        ]
        return iter(self._iter_data)

if __name__ == "__main__":
    #instance of Rectangle
    rect = Rectangle(length=2, width=3)
    # Iterate over Rectangle instance
    for item in rect:
        print(item)
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - [] [X] ... ^ X
PS D:\Accuknox> python Custom_class.py
{'length': 2}
{'width': 3}
PS D:\Accuknox> []
```