

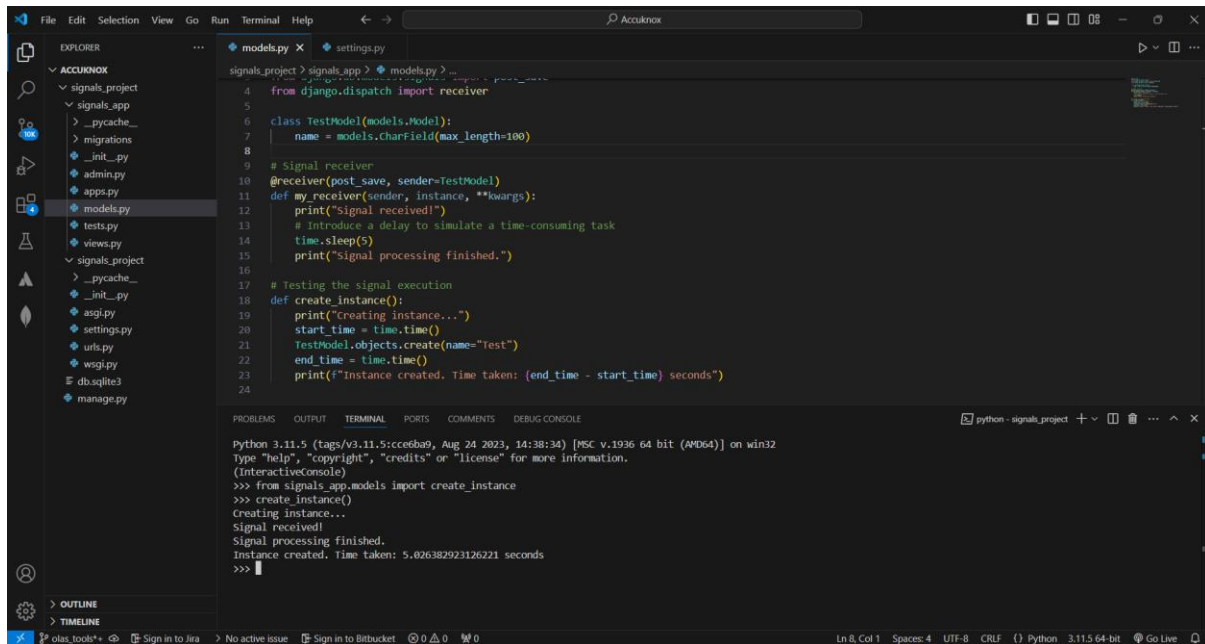
ASSIGNMENT: ACCUKNOX

Topic :- Django Signals

Question 1 :- By Default, Are Django Signals Executed Synchronously or Asynchronously?

Answer with code snippet :-

Django signals are executed **synchronously** by default. This means when a signal is sent, the sender will wait until the signal handlers (receivers) have finished their execution before proceeding. To test this, I wrote a simple code snippet using a model and a signal to see how long it takes to create a model instance when the signal introduces a delay.



```
4 from django.dispatch import receiver
5
6 class TestModel(models.Model):
7     name = models.CharField(max_length=100)
8
9 # Signal receiver
10 @receiver(post_save, sender=TestModel)
11 def my_receiver(sender, instance, **kwargs):
12     print("Signal received!")
13     # Introduce a delay to simulate a time-consuming task
14     time.sleep(5)
15     print("Signal processing finished.")
16
17 # Testing the signal execution
18 def create_instance():
19     print("Creating instance...")
20     start_time = time.time()
21     TestModel.objects.create(name="Test")
22     end_time = time.time()
23     print(f"Instance created. Time taken: {end_time - start_time} seconds")
24
```

```
Python 3.11.5 (tags/v3.11.5:cc00b49, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
(InteractiveConsole)
>>> from signals_app.models import create_instance
>>> create_instance()
Creating instance...
Signal received!
Signal processing finished.
Instance created. Time taken: 5.026382923126221 seconds
>>>
```

Explanation :-

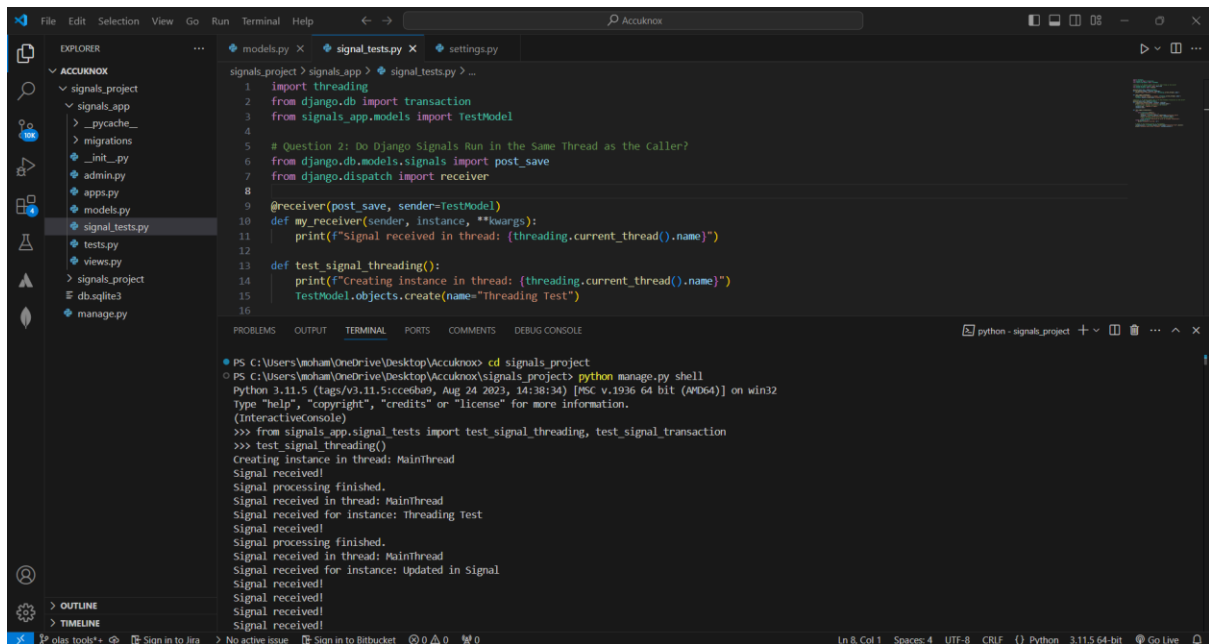
1. I created a model called Test_model and connected a post_save signal to it using the @receiver decorator.
2. The signal receiver (my_receiver) introduces a 5-second delay using time_sleep(5).
3. In the create_instance function, I created a new instance of Test_model and calculated the total time it took.
4. When I ran the function, the output showed that the signal handler's delay was included in the total time, proving that Django signals are executed synchronously.
5. This shows that the signal is processed immediately and synchronously with the model creation.

Question 2 :- Do Django Signals Run in the Same Thread as the Caller ?

Answer with code snippet :-

Yes, Django signals run in the **same thread** as the caller. To test this, I printed the thread name for both the sender (the place where the model instance is created) and the signal receiver. If both show the same thread name, then they run in the same thread.

ASSIGNMENT: ACCUKNOX



```
1 import threading
2 from django.db import transaction
3 from signals_app.models import TestModel
4
5 # Question 2: Do Django Signals Run in the Same Thread as the Caller?
6 from django.db.models.signals import post_save
7 from django.dispatch import receiver
8
9 @receiver(post_save, sender=TestModel)
10 def my_receiver(sender, instance, **kwargs):
11     print(f"Signal received in thread: {threading.current_thread().name}")
12
13 def test_signal_threading():
14     print(f"Creating instance in thread: {threading.current_thread().name}")
15     TestModel.objects.create(name="Threading Test")
16
```

```
PS C:\Users\wcham\OneDrive\Desktop\Accuknox> cd signals_project
PS C:\Users\wcham\OneDrive\Desktop\Accuknox\signals_project> python manage.py shell
Python 3.11.5 (tags/v3.11.5:cc8b4a9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
(InteractiveConsole)
>>> from signals_app.signal_tests import test_signal_threading, test_signal_transaction
>>> test_signal_threading()
Creating instance in thread: MainThread
Signal received!
Signal processing finished.
Signal received in thread: MainThread
Signal received for instance: Threading Test
Signal received!
Signal processing finished.
Signal received in thread: MainThread
Signal received for instance: Updated in Signal
Signal received!
Signal received!
Signal received!
Signal received!
```

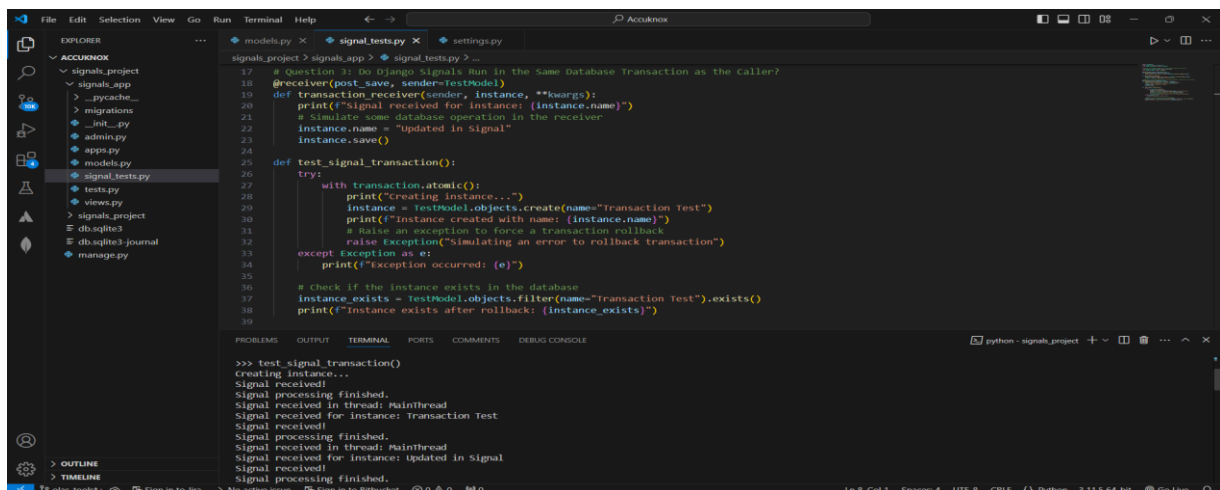
Explanation :-

1. I used the `threading.current_thread().name` method to get the name of the current thread inside both the signal receiver (`my_receiver`) and the function where I create a `TestModel` instance (`test_signal_threading`).
2. When I ran `test_signal_threading()`, both the sender and receiver showed the same thread name.
3. The output confirms that the signal receiver runs in the same thread as the model instance creation.

Question 3 :- By Default, Do Django Signals Run in the Same Database Transaction as the Caller?

Answer with code snippet :-

Yes, Django signals run in the **same database transaction** as the caller by default. I tested this by creating a model instance within a transaction (`transaction.atomic`) and raising an exception to force a rollback. If the signal handler's changes are also rolled back, it indicates that signals run in the same transaction.



```
17 # Question 3: Do Django Signals Run in the Same Database Transaction as the Caller?
18 @receiver(post_save, sender=TestModel)
19 def transaction_receiver(sender, instance, **kwargs):
20     print(f"Signal received for instance: {instance.name}")
21     # Simulate some database operation in the receiver
22     instance.name = "Updated in Signal"
23     instance.save()
24
25 def test_signal_transaction():
26     try:
27         with transaction.atomic():
28             print(f"Creating instance...")
29             instance = TestModel.objects.create(name="Transaction Test")
30             print(f"Instance created with name: {instance.name}")
31             # Raise an exception to force a transaction rollback
32             raise Exception("Simulating an error to rollback transaction")
33         except Exception as e:
34             print(f"Exception occurred: {e}")
35
36     # Check if the instance exists in the database
37     instance_exists = TestModel.objects.filter(name="Transaction Test").exists()
38     print(f"Instance exists after rollback: {instance_exists}")
39
```

```
>>> test_signal_transaction()
Creating instance...
Signal received!
Signal processing finished.
Signal received in thread: MainThread
Signal received for instance: Transaction Test
Signal received!
Signal processing finished.
Signal received in thread: MainThread
Signal received for instance: Updated in Signal
Signal received!
Signal processing finished.
```

ASSIGNMENT: ACCUKNOX

Explanation :-

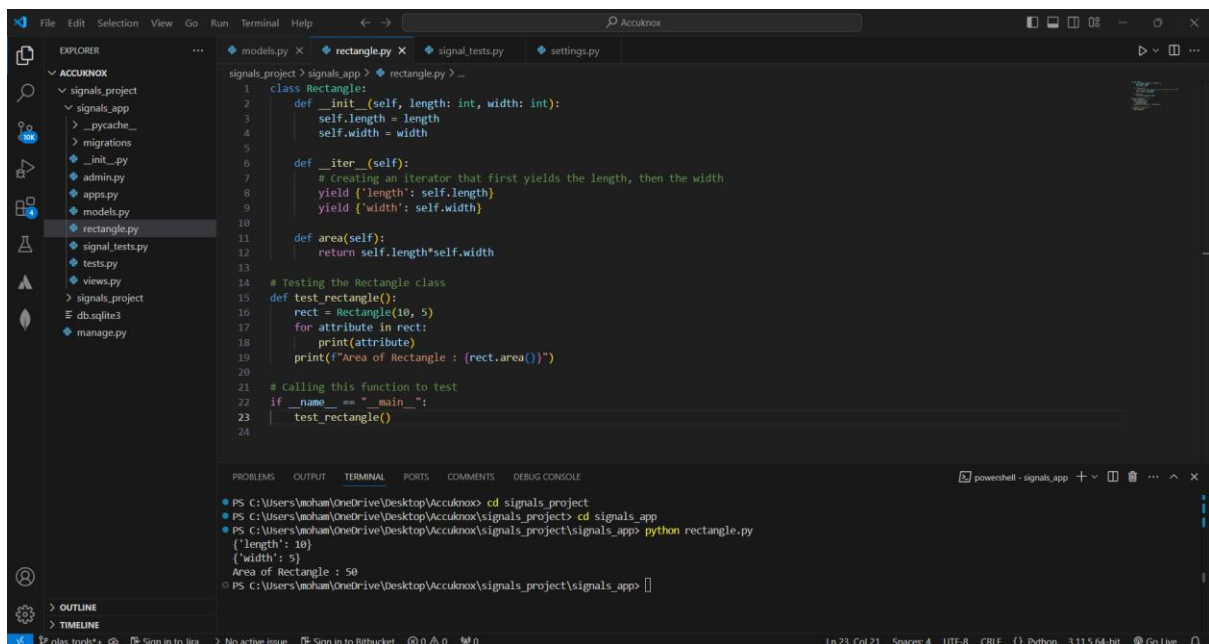
1. I created a post_save signal receiver (transaction_receiver) that updates the model's name when an instance is created.
2. Inside test_signal_transaction, I used transaction.atomic() to start a database transaction.
3. After creating the TestModel instance, I raised an exception to force a rollback.
4. After catching the exception, I checked if the instance still existed in the database.
5. The output showed that the instance was not saved in the database, confirming that both the signal and the model creation were rolled back together.
6. This confirms that the signal was part of the same transaction, as its changes were also rolled back when an exception was raised.

Topic: Custom Classes in Python

Description :- I am tasked with creating a Rectangle class with the few requirements

Answer with code snippet :-

To create a Rectangle class that can be iterated over, I implemented the `__iter__` method in the class. This method makes the class iterable, allowing us to retrieve the length and width in the specified format.



```
1 class Rectangle:
2     def __init__(self, length: int, width: int):
3         self.length = length
4         self.width = width
5
6     def __iter__(self):
7         # creating an iterator that first yields the length, then the width
8         yield ('length': self.length)
9         yield ('width': self.width)
10
11     def area(self):
12         return self.length*self.width
13
14 # Testing the Rectangle class
15 def test_rectangle():
16     rect = Rectangle(10, 5)
17     for attribute in rect:
18         print(attribute)
19     print(f"Area of Rectangle : {rect.area()}")
20
21 # Calling this function to test
22 if __name__ == "__main__":
23     test_rectangle()
24
```

```
PS C:\Users\moham\OneDrive\Desktop\Accuknox> cd signals_project
PS C:\Users\moham\OneDrive\Desktop\Accuknox\signals_project> cd signals_app
PS C:\Users\moham\OneDrive\Desktop\Accuknox\signals_project\signals_app> python rectangle.py
{'length': 10}
{'width': 5}
Area of Rectangle : 50
PS C:\Users\moham\OneDrive\Desktop\Accuknox\signals_project\signals_app>
```

Explanation :-

1. I implemented the `__iter__` method using the yield statement to return the length and width in the required format.
2. To test this, I created a separate test_rectangle function, which I executed to print the attributes of the Rectangle instance.
3. This confirms that the Rectangle class can be iterated over as required.