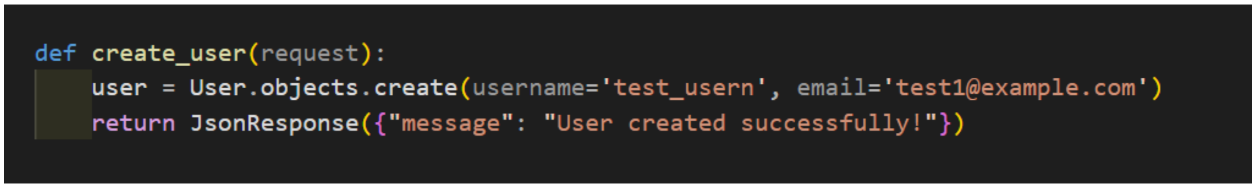
Django Signal(ACCUKNOX)

Django signals provide a way to notify certain parts of an application when actions occur elsewhere in the framework.

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

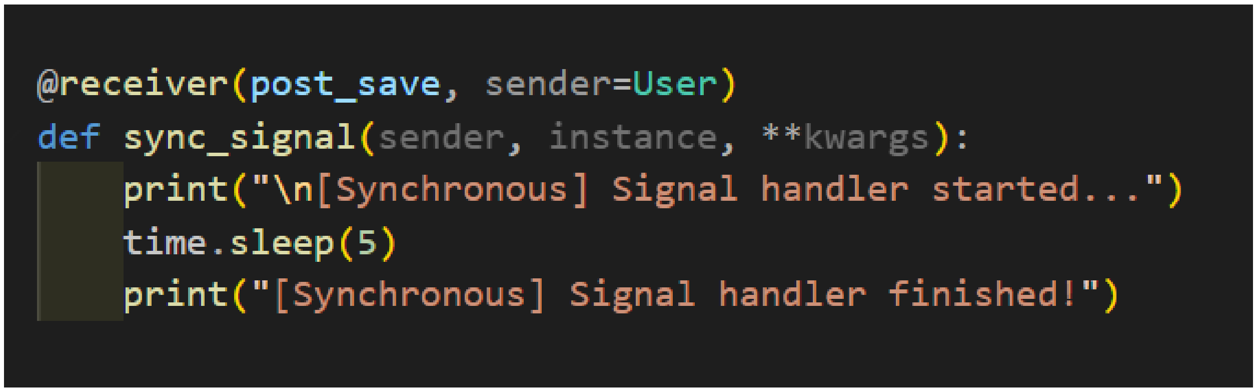
Answer: Django signals are synchronous by default. When a signal is triggered, it executes immediately within the same request-response cycle. If the signal function takes a long time to execute, it can delay the response of the main application.

views.py



```
def create_user(request):
    user = User.objects.create(username='test_usern', email='test1@example.com')
    return JsonResponse({"message": "User created successfully!"})
```

signals.py

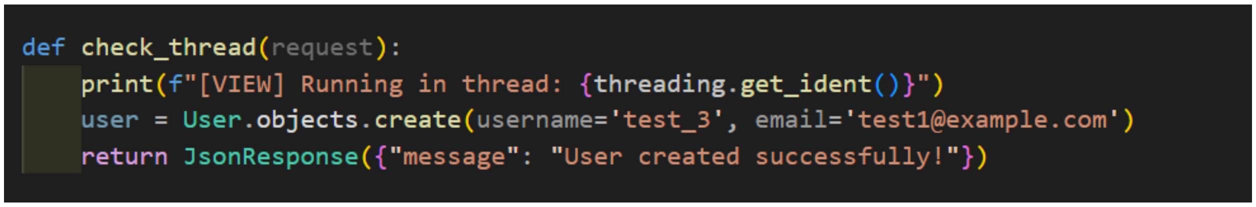


```
@receiver(post_save, sender=User)
def sync_signal(sender, instance, **kwargs):
    print("\n[Synchronous] Signal handler started...")
    time.sleep(5)
    print("[Synchronous] Signal handler finished!")
```

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.

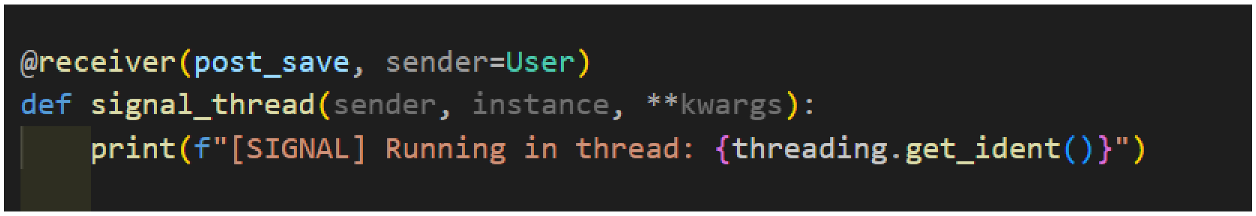
Answer: Yes, Django signals run in the same thread as the function that triggered them. This means that signals do not create separate threads automatically.

views.py



```python
def check_thread(request):
    print(f"[VIEW] Running in thread: {threading.get_ident()}")
    user = User.objects.create(username='test_3', email='test1@example.com')
    return JsonResponse({"message": "User created successfully!"})
```

signals.py

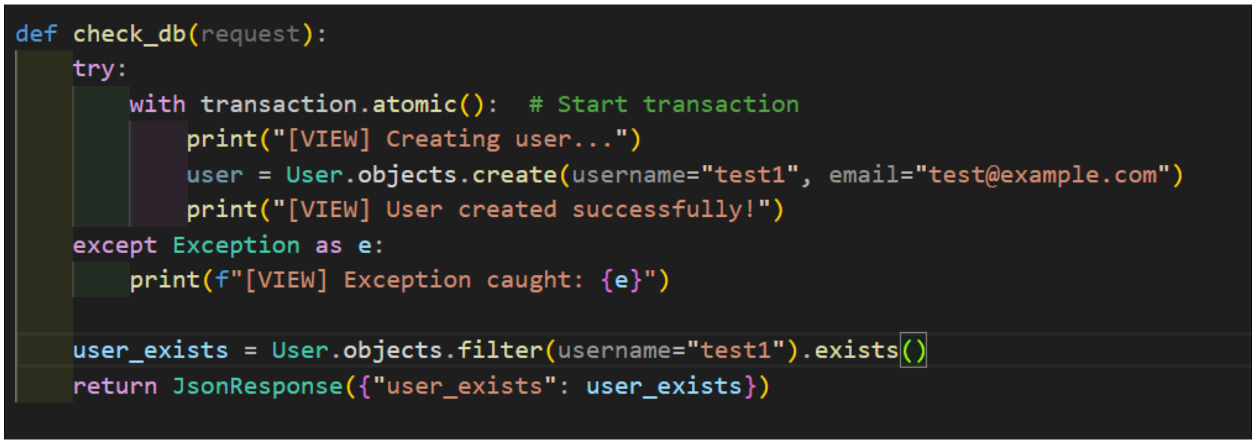


```python
@receiver(post_save, sender=User)
def signal_thread(sender, instance, **kwargs):
    print(f"[SIGNAL] Running in thread: {threading.get_ident()}")
```

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 execute inside the same database transaction as the caller. If an exception is raised in the signal, and it is triggered within a transaction (`transaction.atomic()`), the entire transaction will roll back.

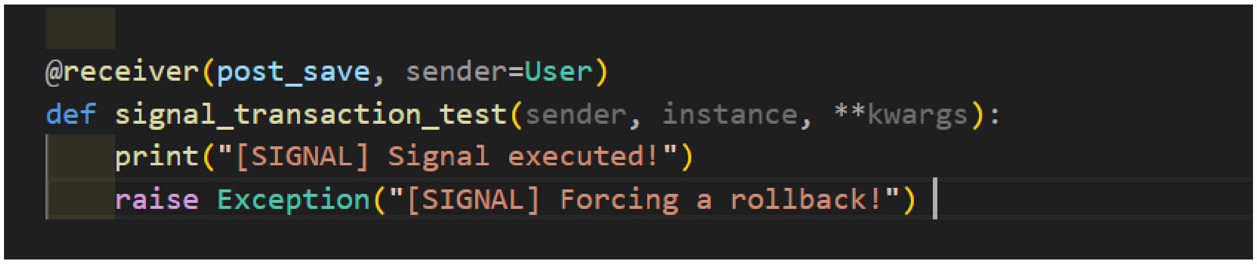
views.py



```python
def check_db(request):
    try:
        with transaction.atomic():  # Start transaction
            print("[VIEW] Creating user...")
            user = User.objects.create(username="test1", email="test@example.com")
            print("[VIEW] User created successfully!")
    except Exception as e:
        print(f"[VIEW] Exception caught: {e}")

    user_exists = User.objects.filter(username="test1").exists()
    return JsonResponse({"user_exists": user_exists})
```

signals.py

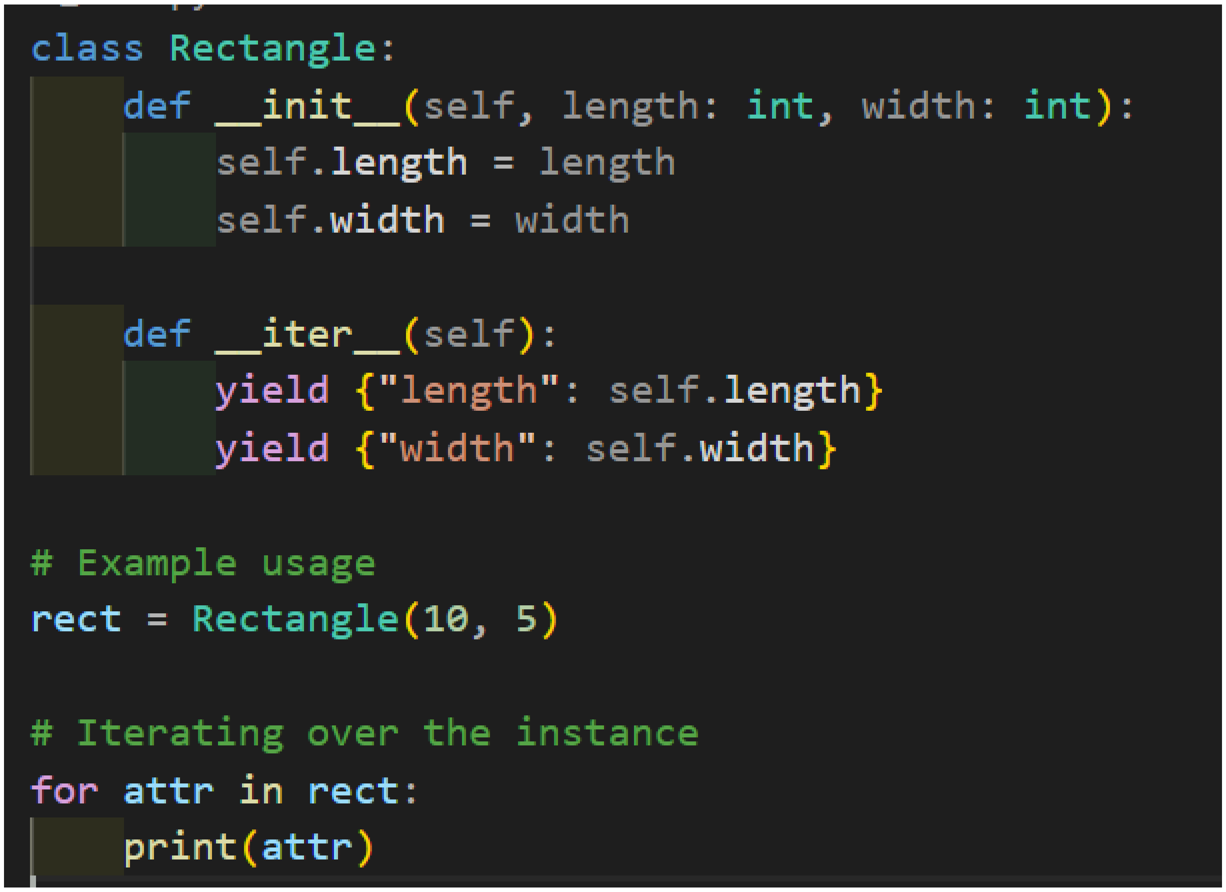


```python
@receiver(post_save, sender=User)
def signal_transaction_test(sender, instance, **kwargs):
    print("[SIGNAL] Signal executed!")
    raise Exception("[SIGNAL] Forcing a rollback!")
```

Question: 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:`
`python_code.py`



```python
class Rectangle:
    def __init__(self, length: int, width: int):
        self.length = length
        self.width = width

    def __iter__(self):
        yield {"length": self.length}
        yield {"width": self.width}

# Example usage
rect = Rectangle(10, 5)

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