ASSIGNMENT

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.

By default django signals are executed synchronously, This means that the signal handler is executed immediately in the same thread when the signal is sent, and the response to the request will not complete until the signal processing is done.

Code: let's take a simple example of django signals where we add time.sleep() function.

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
import time

@receiver(post_save, sender=User)
def user_created_signal(sender, instance, created, **kwargs):
        if created:
                print("Signal handler started.")
                time.sleep(5)
                print("Signal handler finished.")
```

Output:

```
Signal handler started.
# Wait for 5 seconds...
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.

Yes, by default, Django signals run in the same thread as the caller. This means that when a signal is triggered, its handlers are executed within the same thread.

Code: Let's create a simple example where we print the thread ID in both the view (which triggers the signal) and the signal handler itself.

```
#signal.py
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
import threading

@receiver(post_save, sender=User)
def user_created_signal(sender, instance, created, **kwargs):
    if created:
        print(f"Signal handler running in thread: {threading.get_ident()}")

#views.py
from django.contrib.auth.models import User
from django.http import HttpResponse
import threading

def create_user_view(request):
    print(f"View running in thread: {threading.get_ident()}")
    User.objects.create(username='mars')
    return HttpResponse("User created!")
```

Output:

```
View running in thread: 3040
Signal handler running in thread: 3040
```

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.

Yes, by default, django signals such as post_save and pre_save are executed in the same database transaction as the caller.

Code:

```
# signals.py
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User

@receiver(post_save, sender=User)
def user_created_signal(sender, instance, created, **kwargs):
    if created:
        print("Signal triggered")
        raise Exception("Error inside signal")

# views.py
from django.contrib.auth.models import User
from django.http import HttpResponse

def create_user_view(request):
    try:
        User.objects.create(username="testuser")
    except Exception as e:
        return HttpResponse(f"User creation failed due to: {str(e)}")

    return HttpResponse("User created successfully")
```

Output:

```
User creation failed due to Error inside 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>}.

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



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

rect = Rectangle(10, 5)

for dimension in rect:
    print(dimension)
```

Output:
```
{'length': 10}
{'width': 5}
```