[Topic:](https://docs.djangoproject.com/en/3.2/topics/signals/) **[Django Signals](https://docs.djangoproject.com/en/3.2/topics/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.** Whenever we do write signals and consume signals it is seen that they are going to be executed in the process of request

 In the given code, two signals — pre\_save and post\_save — are connected to the User model:

* pre\_save prints the username and ID before saving the user.
* post\_save checks if the user was newly created:
  + If yes, it simulates sending an email.
  + It even calls instance.save() again, which **triggers the signals once more**, showing the synchronous chain of execution.

When the account is created this wil be output

"POST /admin/auth/user/add/ HTTP/1.1" 200 11958

"GET /admin/jsi18n/ HTTP/1.1" 200 3342

**ghtw None #--> ghtw is the user name**

**send email to ghtw**

**ghtw 4**

**ghtw just saved**

[10/Apr/2025 18:20:54] "POST /admin/auth/user/add/ HTTP/1.1" 302 0

**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, **Django signals run in the same thread as the caller** by default. This means when a signal like post\_save is triggered, it executes in the same thread that performed the model save operation

**Code explanation**

 threading is imported to identify which thread is running the signal.

 Django modules like models, settings, receiver, pre\_save, and post\_save are imported for signal handling.

 User = settings.AUTH\_USER\_MODEL dynamically fetches the current user model used in the project.

 The user\_pre\_save\_receiver function is executed just before a User instance is saved.

 It prints the username of the user being saved.

 It prints the thread name running the pre\_save signal using threading.current\_thread().name.

 The user\_post\_save\_receiver function is executed just after a User instance is saved.

 It prints the username of the user just saved.

 It prints the thread name running the post\_save signal.

 The @receiver(pre\_save, sender=User) and @receiver(post\_save, sender=User) decorators connect the signal functions to the User model.

 Printing the current thread in both signal handlers helps confirm that the signals run in the same thread as the main execution thread.

 This proves that Django signals are synchronous by default.

**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.

By default, **Django signals do not run in the same database transaction** as the caller. This means if a signal (like post\_save) fails or throws an exception, it **won’t rollback** the original save operation

 When a new UserAccount is created, the post\_save signal is triggered.

 Inside the signal, an EmailLog is created and an exception is raised to simulate failure.

 Even though the signal raises an exception, the user is still saved to the database.

 This **proves that the signal does not share the same transaction** as the model creation.

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>}**

 **Class & Object-Oriented Programming**:  
Rectangle is a custom class with \_\_init\_\_() for initialization.

 **Generator Function**:  
The \_\_iter\_\_() method uses yield to return values one at a time.

 **Iteration Protocol**:  
The class behaves like an iterable because it defines \_\_iter\_\_().

 **For Loop Mechanics**:  
The for loop uses iter() and next() internally to retrieve items from the generator.