Django signals provide a way to allow certain parts of an application to get notified when specific events occur. By default, Django signals are executed synchronously. This means that when a signal is triggered, the signal handler will be executed immediately, blocking the code execution until it completes.

For example, signals can trigger actions like sending emails or updating related data when a user is created or a model is saved.

- Code Snippet of the Django Signals are as Follows;

signal.py

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


@receiver(pre_save, sender=User)
def pre_save_user_handler(sender, instance, **kwargs):
    print(f"Signal handler started for {instance.username}")
    time.sleep(5)  # Simulate delay to show synchronous behavior
    print(f"Signal handler finished for {instance.username}")


'''
Explanation of this code is;

In the signals.py file, the pre_save_user_handler listens for the pre_save signal triggered before a User model is saved.
It introduces a 5-second delay to simulate processing, proving that Django signals are executed synchronously,
as the code waits for the handler to finish before continuing.

'
```

In the Above code;

1. Signal Handler Definition: The pre_save_user_handler function is defined to respond to the pre_save signal for the User model, executing just before a user instance is saved.

2. Simulated Delay: The handler introduces a 5-second delay using time.sleep(5), demonstrating synchronous execution by blocking further code until the delay finishes.

3. Execution Confirmation: Messages are printed before and after the delay to confirm that the signal handler runs synchronously, as the subsequent code only executes after the handler completes.