Topic : Django Signals

Question 1.

Django signals are executed synchronously. This means the signal and its handler are run at the same time, and the code will wait for the signal handler to finish before moving on.

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_saved_handler(sender, instance, kwargs):
print("User saved signal received.")
```

Views.py

```
 def create_user(request):
    user=User.objects.create(username=”testuser”)
    print (“User created.”)
```

Question 2.

Yes, Django signals run in the same thread as the code that triggered them. This means both the original code and the signal handler are executed together in the same thread.

```
import threading


@receiver(post_save, sender=User)
def user_saved_handler(sender, instance, kwargs):
    print(f"Signal running in thread: {threading.current_thread().name}")


def create_user(request):
    print(f"User creation in thread:  {threading.current_thread().name}")
user = User.objects.create(username="testuser")
```

Question 3.

Not always. Some signals, like post_save, run after the database transaction is complete. But you can make them run in the same transaction by using transaction.on_commit()

Example:

```
from django.db import transaction


@receiver(post_save, sender=User)
def user_saved_handler(sender, instance, **kwargs):
    transaction.on_commit(lambda: print("Signal executed after the transaction."))


def create_user(request):
    user = User.objects.create(username="testuser")
```

The signal runs after the database operation is finished.

Topic: Custom Classes in Python

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

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

Example

```
rect = Rectangle(5, 10)
for dimension in rect:
    print(dimension)
```