TOPIC: Django Signals

Answer 1:

In Django, signals are a way for different parts of a Django application to get notified when certain events occur, such as when a model is saved or deleted. By default, Django processes signals synchronously, which means that the code that handles the signal runs immediately after the event that triggers the signal, in the same process and thread.

```
import time
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 my_handler(sender, instance, **kwargs):
    print("Starting handler")
    time.sleep(5)
    print("Handler finished execution")

user = User.objects.create(username="testuser")
print("User save completed.")
```

OUTPUT:

```
Starting handler
Handler finished execution
User save completed.
```

In this example, the User object is saved, and the signal handler starts immediately after. The delay in the signal handler (time.sleep(5)) occurs before the "User save completed" message, proving that signals are executed synchronously.

Answer 2:

Django signals run in the same thread as the caller. This means the signal handler will be executed in the same thread that called the save() or any other signal-triggering function.

```
import threading
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 my_handler(sender, instance, **kwargs):
    print(f"Signal handler running in thread: {threading.current_thread().name}")

print(f"Caller is in thread: {threading.current_thread().name}")
user = User.objects.create(username="testuser")
```

OUTPUT:

```
Caller is in thread: MainThread
Signal handler running in thread: MainThread
```

The output shows that both the caller and the signal handler run in the same thread, which in this case is the MainThread, proving that signals run in the same thread by default.

Answer 3:

Django signals do not automatically run in the same database transaction as the operation that triggers them. For most signals like post_save, they execute after the database transaction completes, not during it.

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

# Define a signal handler
@receiver(post_save, sender=User)
def my_handler(sender, instance, **kwargs):
    print(f"Signal handler executed. In transaction: {transaction.get_autocommit()}")

# Using a transaction block
with transaction.atomic():
    user = User.objects.create(username="testuser")
    print(f"In transaction: {transaction.get_autocommit()}")
```

OUTPUT:

```
In transaction: False
Signal handler executed. In transaction: True
```

The transaction.get_autocommit() function shows whether we are in a transaction. The output proves that the signal handler is executed after the atomic() block is exited, indicating that Django signals are not necessarily part of the same database transaction by default.

TOPIC: Custom Classes in 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
rectangle = Rectangle(10, 5)
for x in rectangle:
    print(x)
```

OUTPUT:

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