Assignment: Django Trainee at Accuknox

Topic: Django Signals

Question 1: Are Django signals executed synchronously or asynchronously?

By default, Django signals are executed **synchronously**. This means that when a signal is triggered, the execution of the caller waits for the signal's receiver function to complete before continuing.

Code snippet to prove it:

Create a Django signal and log timestamps to verify synchronous execution.

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

# Define a signal receiver
@receiver(post_save, sender=User)
def signal_receiver(sender, instance, **kwargs):
    print("Signal received. Processing...")
    time.sleep(3)  # Simulate a long-running task
    print("Signal processing done.")

# Test the synchronous nature of the signal
if __name__ == "__main__":
    print("Creating user...")
    start_time = time.time()
    user = User.objects.create(username="test_user")
    end_time = time.time()
    print(f"Total execution time: {end_time - start_time:.2f} seconds")
```

Expected Output:

```
Creating user...
Signal received. Processing...
Signal processing done.
Total execution time: ~3 seconds
```

Since the execution waits for the signal receiver to complete, it proves that Django signals are **synchronous by default**.

Question 2: Do Django signals run in the same thread as the caller?

Yes, by default, Django signals run **in the same thread as the caller**.

Code snippet to prove it:

Use Python's `threading` module to check the thread ID.

```
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 signal_receiver(sender, instance, **kwargs):
    print(f"Signal is running in thread: {threading.get_ident()}")

# Test signal threading behavior
if __name__ == "__main__":
    print(f"Main thread ID: {threading.get_ident()}")
    user = User.objects.create(username="test_user")
```

Expected Output:

```
Main thread ID: 140735642768128
Signal is running in thread: 140735642768128
```

Since both the main thread and the signal's execution thread are the same, it confirms that **Django signals execute in the same thread as the caller**.

Question 3: Do Django signals run in the same database transaction as the caller?

Yes, Django signals run **in the same database transaction** by default unless explicitly handled differently.

Code snippet to prove 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

@receiver(post_save, sender=User)
def signal_receiver(sender, instance, **kwargs):
    print(f"Signal received. Transaction ID: {transaction.get_autocommit()}")

# Test database transaction behavior
if __name__ == "__main__":
    with transaction.atomic():  # Start a DB transaction
        user = User.objects.create(username="test_user")
        print(f"Main transaction ID: {transaction.get_autocommit()}")
```

Expected Output:

```
Signal received. Transaction ID: False
Main transaction ID: False
```

Since both the signal and the main transaction have `autocommit=False`, they share the same transaction, proving that **Django signals run in the same transaction as the caller by default**.

Topic: Custom Classes in Python

Rectangle Class Implementation

Here's the implementation of the `Rectangle` class with iteration support.

```
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 usage
rectangle = Rectangle(10, 5)

for attr in rectangle:
    print(attr)
```

Expected Output:

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

This implementation ensures that we can **iterate over a `Rectangle` instance**, returning the length first, followed by the width.