Questions for Django Trainee at Accuknox

Topic: **Django 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**.

Ans : Django signals executed synchronously. when a signal is triggered, the associated signal handlers (functions connected to the signal) are executed in the same thread before the triggering action completes. Django signals operate synchronously because they are designed to allow hooks into various parts of the application lifecycle.

the following code snippet demonstrates that Django signals are executed synchronously.

```python
@receiver(post_save, sender=User)
def signalHandler(sender, instance, **kwargs):
    print("Signal handler started")
    time.sleep(5)  # Simulating a long-running operation
    print("Signal handler finished")

# A function to create a user and observe the signal behavior
def create_user_and_test_signal():
    print("Saving user...")
    user = User.objects.create(username="testuser")
    print("User saved")
```

Output:

Saving user...

Signal handler started

[Pauses for 5 seconds]

Signal handler finished User saved

B) Making Asynchronously

Use threads or asyncio for lightweight asynchronous processing.

```python
@receiver(post_save, sender=User)
def signal_handler(sender, instance, **kwargs):
    def task():
        print("Signal handler started")
        time.sleep(5)  # Simulating a long-running operation
        print("Signal handler finished")

    threading.Thread(target=task).start()
```

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.

Ans: Yes, Django signals run in the same thread as the caller by default. This means that when a signal is triggered, its connected signal handlers execute in the same thread that initiated the action (e.g., saving a model instance). This is part of Django's synchronous behavior

```python
@receiver(post_save, sender=User)
def signal_handler(sender, instance, **kwargs):
    print("Signal handler thread:", threading.current_thread().name)

# Function to create a user and print thread information
def create_user_and_test_thread():
    print("Caller thread:", threading.current_thread().name)
    user = User.objects.create(username="testuser")
```

Output:

Caller thread: MainThread

Signal handler thread: MainThread

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.

Ans: Yes, by default, Django signals run in the same database transaction as the caller. The signal handlers execute within the same transaction context. If the transaction is rolled back, any changes made by the signal handlers are also rolled back.

```python
# Signal handler to insert data into another table
@receiver(post_save, sender=User)
def signal_handler(sender, instance, **kwargs):
    print("Signal handler: Transaction status before insertion:", connection.in_atomic_block)
    # Attempt to insert into the database
    with connection.cursor() as cursor:
        cursor.execute("INSERT INTO example_table (data) VALUES ('Signal Data')")
    print("Signal handler: Data inserted into 'example_table'")

# Function to test signal and transaction behavior
def create_user_and_test_transaction():
    try:
        with transaction.atomic():
            print("Caller: Transaction status before user save:", connection.in_atomic_block)
            user = User.objects.create(username="testuser")
            print("Caller: User saved, raising exception to roll back")
            raise Exception("Force rollback")
    except Exception as e:
        print("Caller: Transaction rolled back")
```

A) Transaction Context:

1)The transaction.atomic() block creates a transaction. Any operation (including the signal handler) executed within this block is part of the same transaction.

2)The connection.in_atomic_block property confirms whether the code is running inside a transaction.

B) Signal Behavior: The post_save signal is triggered after saving the user. The signal handler inserts data into example_table, but since the transaction is rolled back after the exception, the insertion is also rolled back.

C) Rollbacks Affect Signal Handlers:Since the signal handler runs in the same transaction as the caller, any database operations it performs are undone if the transaction is rolled back.

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 Rectangle:
    def __init__(self):
        self.length = int(input("Enter the length: "))
        self.width = int(input("Enter the width: "))

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

rectangle = Rectangle()
print("\nIterating over the Rectangle:")
for dimension in rectangle:
    print(dimension)
```