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.

Answer 1: By default, Django signals are executed synchronously.
example: Imagine you are building a Django e-commerce application, and you want to send a confirmation email every time an order is placed. You'll use Django signals for this.
In the example below, the signal will trigger after an order is saved to the database. Since Django signals are synchronous by default, the email will only be sent after the signal's receiver finishes its execution. If the email sending process takes time, the order creation process will also be delayed.
Code:
Define Model

```
class Order:
    attributes:
        customer
        total_amount
```

Define Signal Reciever:

```
function send_order_confirmation_email(order):
    print("Starting to send confirmation email for Order", order.id)
    simulate_delay(5)  # Simulates time taken to send an email (5 seconds)
    print("Confirmation email sent for Order", order.id)
```

Connect Signal:

```
on post_save signal for Order:
    call send_order_confirmation_email(order)
```

Create Order:

```
order = create Order(customer="John Doe", total_amount=100)
# This triggers the post_save signal and calls send_order_confirmation_email
```

Explanation:

1. **Order Creation**: When an Order is created using create Order(...), the post_save signal is triggered immediately after saving the order.
2. **Synchronous Execution**: The function send_order_confirmation_email is called right away. It prints a message, simulates a 5-second delay (representing the time taken to send an email), and then prints a confirmation message.
3. **Blocking Behavior**: The entire process of order creation is paused for 5 seconds while the email sending simulation runs. This shows that the signal is executed synchronously, as the code does not continue until the receiver function has finished executing.

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.

Answer 2: Yes, Django signals run in the **same thread** as the caller. As a result, any blocking operations in the signal handler can affect the performance of the code that triggered the signal.

Scenario

Let's consider a simple scenario in an e-commerce application where we need to log an order creation event whenever an order is placed. We'll use a signal to achieve this.

Code Example:

Define Model

```
class Order:
    attributes:
        customer
        total_amount
```

Define Signal Reciever:

```
function log_order_creation(order):
    current_thread = get_current_thread_name()
    print("Order", order.id, "created in thread:", current_thread)
```

Connect Signal:

```
on post_save signal for Order:
    call log_order_creation(order)
```

Create Order:

```
order = create Order(customer="John Doe", total_amount=100)
# This triggers the post_save signal and calls log_order_creation
```

Explanation:

1. **Order Creation**: When you create an Order, the post_save signal is triggered immediately after the order is saved.
2. **Signal Trigger**: The log_order_creation function executes as a result of the signal. It checks the current thread name.
3. **Same Thread Execution**: The thread name printed in log_order_creation confirms that the signal is running in the **same thread** as the order creation. This means that any delays or blocking operations in the signal handler will affect the order creation process.

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.
Answer 3: Yes, by default, Django signals run in the **same database transaction** as the caller.
Code explanation:

```
Define Models
class Order:
    attributes:
        customer
        total_amount
        created_at

Define Signal reciever:
function update_order_status(order):
    if order.total_amount < 0:
        raise Error("Total amount cannot be negative.")
    print("Order status updated for:", order.id)

Connect Signals:
on post_save signal for Order:
    call update_order_status(order)

Transaction block:
try:
    start transaction
    order = create Order(customer="John Doe", total_amount=-50)
    commit transaction  # This will trigger the post_save signal
except Error as e:
    print("Error:", e)  # If there's an error, rollback the transaction
    rollback transaction
```

Explanation:

1. **Order Creation**: When creating an Order, if the total_amount is negative, it raises an error.
2. **Signal Trigger**: The post_save signal is connected to the update_order_status function, which executes when the Order is saved.

3. **Transaction Handling**: If an error occurs (e.g., negative total amount), the transaction is rolled back, and any operations in the signal are also reverted.

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, length: int, width: int):
        self.length = length
        self.width = width

    def __iter__(self):
        # This method allows the instance to be iterable
        yield {'length': self.length}
        yield {'width': self.width}
```

Explanation:

1. **Initialization**:
    - The __init__ method initializes the Rectangle object with the provided length and width.
2. **Iteration**:
    - The __iter__ method is implemented to make the instance of the Rectangle class iterable.