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 are executed synchronously by default.

```
import time
from django.core.signals import request_finished
from django.dispatch import receiver

@receiver(request_finished)
def my_callback(sender, **kwargs):
    print("Request finished!")
    time.sleep(5)

request_finished.send(sender=None)
print("After signal")
```

Here, when the request_finished signal is triggered, the my_callback function is executed, simulating a long-running task with a time.sleep(5). The message “After signal” appears only once the receiver has completed, confirming synchronous execution.

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 are executed in the same thread as the caller.

```
import threading
from django.core.signals import request_finished
from django.dispatch import receiver

@receiver(request_finished)
def my_receiver(sender, **kwargs):
    print(f"Receiver thread: {threading.current_thread().name}")

print(f"Main thread: {threading.current_thread().name}")
request_finished.send(sender=None)
```

It will give the output as
Main thread: MainThread
Receiver 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:
Django signals do not run in the same database transaction as the caller by default.

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>}`**

Ans:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def __iter__(self):
        yield {'length': self.length}
        yield {'width': self.width}

a = Rectangle(5,2)
for i in a:
    print(i)
```