Question 1: By default, are Django signals executed synchronously or asynchronously? Please support your answer with a code snippet.

By default, **Django signals are executed synchronously**. This means that the signal is sent and handled in the same thread and process as the caller. The process does not continue until all connected signal handlers are finished.

Code Snippet:

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

@receiver(post_save, sender=User)
def user_saved(sender, instance, **kwargs):
    print("Signal started")
    time.sleep(5)  # Simulate a long-running task
    print("Signal finished")

# Creating a user
user = User.objects.create(username='testuser')

# Output in the terminal:
# Signal started
# (Pauses for 5 seconds)
# Signal finished
```

The 5-second pause shows that the signal handler is running synchronously, blocking the main execution.

Question 2: Do Django signals run in the same thread as the caller? Please support your answer with a code snippet.

Yes, **Django signals run in the same thread as the caller by default**. Since signals are executed synchronously, they share the same thread with the caller.

Code Snippet:

```
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 user_saved(sender, instance, **kwargs):
    print(f"Signal thread: {threading.current_thread().name}")

# Creating a user
user = User.objects.create(username='testuser')

# Output in the terminal:
# Signal thread: MainThread
```

The output shows that both the main execution and the signal handler run in the same thread, `MainThread`.

Question 3: Do Django signals run in the same database transaction as the caller? Please support your answer with a code snippet.

Yes, **Django signals (such as `post_save`) run in the same database transaction as the caller**. If the transaction is rolled back, the signal will not persist its changes.

Code Snippet:

```
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 user_saved(sender, instance, **kwargs):
    print(f"Signal fired for: {instance.username}")

try:
    with transaction.atomic():
        user = User.objects.create(username='testuser')
        raise Exception("Rolling back transaction")
except Exception:
    print("Transaction rolled back")

# Output in the terminal:
# (No signal output because the transaction was rolled back)
```

The signal handler doesn't fire because the transaction is rolled back.

Topic: Custom Classes in Python

Description: Create a `Rectangle` class with the following requirements:

1. The class requires `length:int` and `width:int` to be initialized.
2. We can iterate over an instance of the `Rectangle` class.
3. When iterated, it first returns the length in the format `{'length': <VALUE_OF_LENGTH>}`, then the width in the format `{'width': <VALUE_OF_WIDTH>}`.

Code:

```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}

# Creating a rectangle instance
rect = Rectangle(5, 3)

# Iterating over the instance
for dimension in rect:
    print(dimension)

# Output:
# {'length': 5}
# {'width': 3}
```

This class meets all the requirements: it initializes with `length` and `width`, and it can be iterated over to return its dimensions.