Django Signals:

Question 1:

By default, Django signals are executed synchronously. This means they are executed within the same thread as the signal sender.

Code Example:

Python

```
from django.db.models.signals import post_save
from django.dispatch import receiver

class MyModel(models.Model):
    # ...

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("Signal received")
    # Perform some long-running task here
```

Question 2 :

Django signals typically run in the **same thread** as the caller.

Code Example:

Python

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from threading import current_thread

class MyModel(models.Model):
    # ...

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("Signal thread ID:", current_thread().ident)

# Trigger the signal
my_object = MyModel.objects.create(...)
```

Question 3 :

By default, Django signals run within the same database transactions as the caller.

Code Example:

Python

```
from django.db.models.signals import post_save
from django.dispatch import receiver

class MyModel(models.Model):
    # ...

@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    # Perform database operations here, which will be part of the same transaction
    RelatedModel.objects.create(my_model=instance)
```

Custom Classes in Python:

Python

```
class Rectangle:
    def _init_(self, length, width):
        self.length = length
        self.width = width

    def _iter_(self):
        yield ('length', self.length)
        yield ('width', self.width)

# Example usage:
rect = Rectangle(5, 10)
for item in rect:
    print(item)  # Output: ('length', 5), ('width', 10)
```