Django Signals
Q1.
Django signals are executed synchronously. This means that when a signal is triggered, the signal handler is executed immediately, within the same thread as the triggering action. The handler must complete before the execution of the triggering function continues.

Here's the example that demonstrates Django signals are executed synchronously by default:

```
class MyModel(models.Model):
          name = models.CharField(max_length=100)

# Signal handler
@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
         print("Signal handler started”)
         time.sleep(5) # Simulate a delay
          print("Signal handler finished")
# Testing the synchronous behavior
if __name__ == “__main__":
         print("Save started")
         my_model = MyModel(name="Test") my_model.save()
print("Save finished”)
```

Q2.
Yes, by default, Django signals run in the same thread as the caller.

```
class MyModel(models.Model):
   name = models.CharField(max_length=100)

# Signal handler
@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
   print(f"Signal handler running in thread: {threading.current_thread().name}")

# Testing if the signal runs in the same thread
if _name_ == "_main_":
   print(f"Main code running in thread: {threading.current_thread().name}")
   my_model = MyModel(name="Test")
   my_model.save()
```

Q3
Django signals do **not** run in the same database transaction as the caller.

```
class MyModel(models.Model):
   name = models.CharField(max_length=100)

# Signal handler for post_save
```

```
@receiver(post_save, sender=MyModel)
def my_signal_handler(sender, instance, **kwargs):
    print("Signal handler triggered")

# Testing if signal runs in the same transaction
if _name_ == "_main_":
    try:
        with transaction.atomic():
            print("Transaction started")
            my_model = MyModel(name="Test")
            my_model.save()  # This triggers post_save signal
            print("Raising exception to rollback")
            raise Exception("Forcing rollback")
    except Exception as e:
        print(f"Exception caught: {e}")

       print("Is object in DB after rollback?:", MyModel.objects.exists())
```

Custom Classes in Pyton

```
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}

rect = Rectangle(10, 5)

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