Threads in Python

Python has a number of different concurrency constructs such as threading, queues and multiprocessing. The threading module used to be the primary way of accomplishing concurrency. A few years ago, the multiprocessing module was added to the Python suite of standard libraries. This chapter will be focused on how to use threads and queues.

Using Threads

We will start with a simple example that just demonstrates how threads work. We will sub-class the **Thread** class and make it print out its name to stdout. Let's get coding!

```
import random
import time

from threading import Thread

class MyThread(Thread):
    """
    A threading example
    """

    def __init__(self, name):
        """Initialize the thread"""
        Thread.__init__(self)
        self.name = name

    def run(self):
        """Run the thread"""
        amount = random.randint(1, 3)
        time.sleep(amount)
        msg = "%s is running" % self.name
        print(msg)

def create_threads():
    """
    Create a group of threads
    """

    for i in range(5):
        name = "Thread #%s" % (i+1)
        my_thread = MyThread(name)
        my_thread.start()

if __name__ == "__main__":
    create_threads()
```

In the code above, we import Python's **random** module, the **time** module and we import the **Thread** class from the **threading** module. Next we sub-class Thread and make override its **__init__** method to accept an argument we label "name". To start a thread, you have to call its **start()** method. When you start a thread, it will automatically call the thread's **run** method. We have overridden the thread's run method to make it choose a random amount of time to sleep. The **random.randint** example here will cause Python to randomly choose a number from 1-3. Then we make the thread sleep the number of seconds that we just randomly chose to simulate it actually doing something. Finally we print out the name of the thread to let the user know that the thread has finished.

The create_threads function will create 5 threads, giving each of them a unique name. If you run this code, you should see something like this:

```
Thread #2 is running
Thread #3 is running
Thread #1 is running
Thread #4 is running
Thread #5 is running
```

The order of the output will be different each time. Try running the code a few times to see the order change. Now let's write something a little more practical!