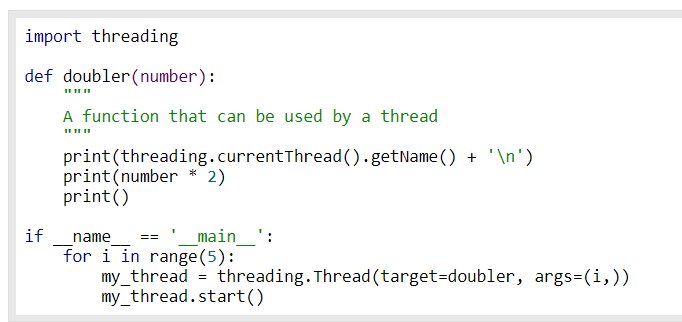
**Тема занятия №23: Многопоточное, асинхронное и мультипроцессорное программирование. GIL**

**1. Threading**

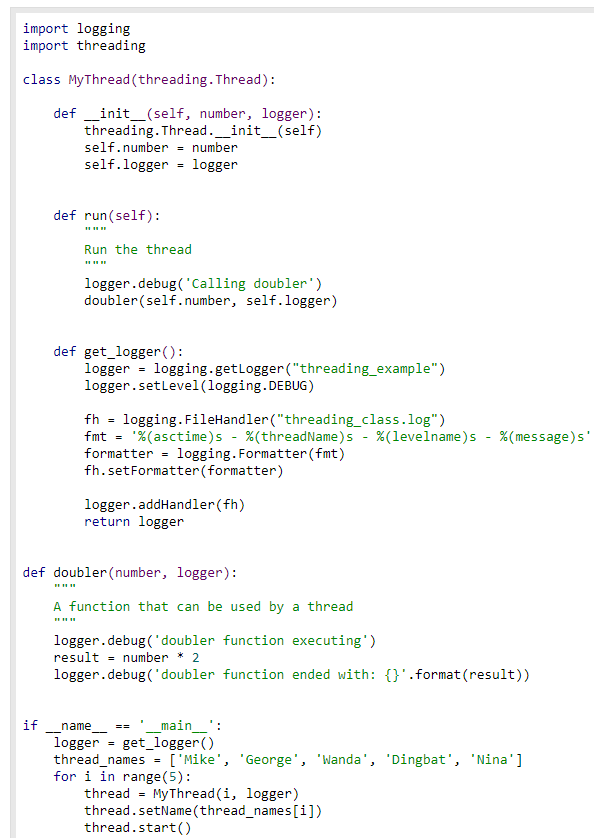
Модуль threading впервые был представлен в Python 1.5.2 как продолжение низкоуровневого модуля потоков. Модуль threading значительно упрощает работу с потоками и позволяет программировать запуск нескольких операций одновременно. Обратите внимание на то, что потоки в Python лучше всего работают с операциями I/O, такими как загрузка ресурсов из интернета или чтение файлов и папок на вашем компьютере.

Если вам нужно сделать что-то, для чего нужен интенсивный CPU, тогда вам, возможно, захочется взглянуть на модуль multiprocessing, вместо threading. Причина заключается в том, что Python содержит Global Interpreter Lock (GIL), который запускает все потоки внутри главного потока. По этой причине, когда вам нужно запустить несколько интенсивных операций с потоками, вы заметите, что все работает достаточно медленно. Так что мы сфокусируемся на том, в чем потоки являются лучшими: операции I/O.

Поток позволяет вам запустить часть длинного кода так, как если бы он был отдельной программой. Это своего рода вызов наследуемого процесса, за исключением того, что вы вызываете функцию или класс, вместо отдельной программы. Я всегда находил конкретные примеры крайне полезными. Давайте взглянем на нечто совершенно простое:



Эта выдача достаточно понятная, так что давайте пойдем дальше. Я хочу разобрать еще один вопрос. Мы поговорим о наследовании класса под названием threading.Thread. Давайте снова рассмотрим предыдущий пример, только вместо вызова потока напрямую, мы создадим свой собственный подкласс. Вот обновленный код:

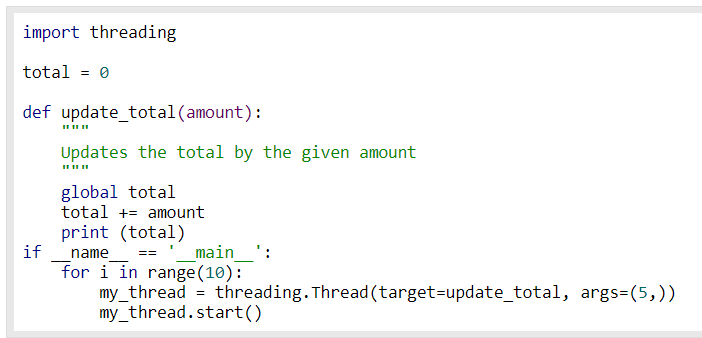


В этом примере мы только что унаследовали класс threading.Thread. Мы передали число, которое хотим удвоить, а также передали объект логгированмя, как делали это ранее. Но на этот раз, мы настроим название потока по-другому, вызвав функцию setName в объекте потока. Нам все еще нужно вызвать старт в каждом потоке, но запомните, что нам не нужно определять это в наследуемом классе. Когда вы вызываете старт, он запускает ваш поток, вызывая метод run. В нашем классе мы вызываем функцию doubler для выполнения наших вычислений. Выдача сильно похожа на ту, что была в примере ранее, за исключением того, что я добавил дополнительную строку в выдаче. Попробуйте сами и посмотрите, что получится.

**Замки и Синхронизация**

Когда у вас в распоряжении более одного потока, тогда вам, возможно, понадобится понять, как избежать конфликтов. Под этим я имею ввиду то, что вы можете использовать случай, где более одного потока нуждаются в доступе к одном и тому же ресурсу в одно и то же время. Если вы не думаете о таких проблемах и соответственном планировании, тогда вы можете столкнуться с самыми худшими проблемами в крайне неудобное время, и, как правило, в момент выпуска кода.

Решение проблемы – это использовать замки. Замок предоставлен модулем Python threading и может держать один поток, или не держать поток вообще. Если поток пытается acquire замок на ресурсе, который уже закрыт, этот поток будет ожидать до тех пор, пока замок не откроется. Давайте посмотрим на практичный пример одного кода, который не имеет никакого замочного функционала, но мы попробуем его добавить:



Здесь мы просто вешаем замок, перед тем как сделать что-либо другое. Далее, мы пытаемся обновить total и finally, мы снимаем замок и выводим нынешний total. Мы можем упростить данную задачу, используя оператор Python под названием with:

