Timers

The threading module has a neat class called **Timer** that you can use to represent an action that should take place after a specified amount of time. They actually spin up their own custom thread and are started using the same **start()** method that a regular thread uses. You can also stop a timer using its **cancel** method. It should be noted that you can even cancel the timer before it's even started.

The other day I ran into a use case where I needed to communicate with a subprocess I had started but I needed it to timeout. While there are lots of different approaches to this particular problem, my favorite solution was using the threading module's Timer class.

For this example, we will look at using the **ping** command. In Linux, the ping command will run until you kill it. So the Timer class becomes especially handy in Linux-land. Here's an example:

```
import subprocess

from threading import Timer

kill = lambda process: process.kill()
cmd = ['ping', 'www.google.com']
ping = subprocess.Popen(
    cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

my_timer = Timer(5, kill, [ping])

try:
    my_timer.start()
    stdout, stderr = ping.communicate()
finally:
    my_timer.cancel()

print (str(stdout))
```

Here we just set up a lambda that we can use to kill the process. Then we start our ping job and create a Timer object. You will note that the first argument is

the time in seconds to wait, then the function to call and the argument to pass to the function. In this case, our function is a lambda and we pass it a list of arguments where the list happens to only have one element. If you run this code, it should run for about 5 seconds and then print out the results of the ping.