Exercises: Concurrency and Threads

Part A

The function:

```
import time

def countdown(identity, count):
    for i in range(count):
        for j in range(10000000):
            pass
        print('{} {} {}'.format(identity, i, time.time()))
```

is a computationally intensive way of doing very little, other than waiting a while. We can show this in action with the following script which executes the above function a number of times sequentially:

```
import multiprocessing
import time

import countdown

if __name__ == '__main__':
    startTime = time.time()
    for i in range(8):
        countdown.countdown(i, 5)
    print('Elapse time = {}'.format(time.time() - startTime))
```

Amend the script to replace to sequential loop with the use of 8 threads countdown function – and hence show that threads can make things significantly worse if there are many processors and/or cores)!

Copy and amend the script to use processes instead of threads to execute the countdown function – and hence show that processes are the Pythonic way of accessing real parallelism in the situation where there are many processors and/or cores.

Part B

The "Sleeping Barber" is a classic "toy" problem for investigating issues in operating system resource management.

A barber's shop has one (or more) barber(s) who cut customers' hair. The shop has a "cutting chair" for each barber and a set of seats where customers can wait if there are no barbers immediately available. If there are no customers then barbers will go to sleep in their cutting chair. Customers arrive at random intervals. If there is a barber available (i.e. a barber is asleep in a cutting chair) or there is a waiting seat available, the customer will stay for a hair cut, otherwise they will leave. Barbers always deal with a customer if one is available, even if this means waking up. A barber takes a certain amount of time to cut a customer's hair, but there is randomness to how long an individual one takes. The overall goal is to find out how many customers the shop can deal with in a given period and how many get turned away. Obviously the arrival time, cutting time and number of barbers are the variables that can be altered to investigate the behaviour of the system.

This problem can be modelled using threads, an event loop, or processes (using actors or CSP). Using threads or an event loop some form of shared memory and locking is required. Using processes, there is no shared memory, message passing provides the synchronization without the needs for locks and semaphores.

Overall then there are 8 possible programs to be written:

	Single Barber	**Multiple Barber**
Threads		
Event Loop		
Actor Model		
CSP		

Write some of these programs simulating the “Sleeping Barber Problem” -- and in the process show that threads are a poor tool for handling this, especially when there are multiple barbers.