We did not get far enough in lecture 1, but solutions to the next few in-class questions should help on today’s lab.

Often a for loop iterates over a list generated by the built-in range function which has the syntax of: range([start,] end, [, step]), where [ ] are used to denote optional parameters. Some examples:

- range(5) generates the list [0, 1, 2, 3, 4]
- range(2, 7) generates the list [2, 3, 4, 5, 6]
- range(10, 2) generates the empty list []
- range(10, 2, -1) generates the list [10, 9, 8, 7, 6, 5, 4, 3]

A for loop iterates over a list generated by range would look like:
```python
for count in range(6):
    print count, "  " ,
print \"Done\"
```

Since the list generated by the range function needs to be stored in memory, a more efficient xrange function is typically using in for loops to generate each value one at a time for each iteration of the loop. For example:
```python
for count in xrange(6):
    print count, "  " ,
print \"Done\"
```

3. Suppose that you have an \( \theta(n^2) \) algorithm that required 10 seconds to run on a problem size of 1000. How long would you expect the algorithm to run on a problem size of 10,000?

A \( \theta(n^2) \) algorithm implies execution time, \( T(n) = c_1 n^2 + (\text{slower growing terms like } c_2 n + c_3) \), where \( c_1, c_2, c_3 \) are constants determined by the speed of your computer.

For large n’s, \( T(n) \approx c_1 n^2 \). We are given that the problem requires 10 seconds to run on a problem size of 1000, so \( T(1000) = 10 \text{ seconds} \approx c_1 1000^2 \), solving for \( c_1 = 10 \text{ seconds} / 1000^2 = 1 \text{ second} / 10^5 = 10^{-5} \text{ seconds} \).

Thus, \( T(10,000) \approx c_1 10000^2 = 10^{-5} \text{ seconds} \times 10000^2 = 1000 \text{ seconds} \).

The problem size got 10 times bigger -- it went from 1000 to 10,000. However, the execution time got 100 times longer -- it went from 10 seconds to 1000 seconds, since the algorithm was \( \theta(n^2) \).

4. Analyze the below algorithm to determine its theta notation, \( \Theta(\cdot) \).

```python
i = n
while i > 0:
    for j in xrange(n):
        # something of \( O(1) \)
        
    # end for
    i = i / 2
# end while
```

The outer while-loop executes \( \Theta(\log_2 n) \) times. IMPORTANT: Any time \( n \) is repeatedly halved down to 1 or 0, the loop executes \( \Theta(\log_2 n) \) times. Similarly, any time a variable \( i \) starts at 1 and is repeatedly doubled to reach the value \( n \), the loop executes \( \Theta(\log_2 n) \) times.

For each iteration of the outer-loop, the inner-loop executes \( n \) times, so the whole algorithm is \( \Theta(n \log_2 n) \). Pronounced as “theta of \( n \log n \).”