Lecture 3
Functions, for loop, and data structures.

1. HOMEWORK REVIEW
   a. **Keep it simple.** Start from the most basic solution. Make it work. Only then refine it, optimize it. Don’t get stuck in trying to make it efficient before you have the less-efficient solution. It’s always easiest to refine a working solution.
   b. **Divide and conquer.** Never tackle the whole problem in one algorithm. Find ways of dividing the task. Solve the simpler problems. Make sure they work. Write functions that your solution can use. Only later, put it all together.
   c. **Run by hand.** Check that the code works for a simpler version of the problem using a table and keeping track of the values and how they are update.
   d. **Debug.** (Funny history to the origin of the term). Find out where your program is at and what it’s doing by putting print statements in certain key locations. Compare with the results above.
   e. **Test functions.** Write functions that test your solutions thoroughly.

2. FUNCTIONS
   a. Why use functions?
      i. It allows to break code into modules.
      ii. It allows to suppress details.
      iii. It allows us to create new primitives that we can reuse.
   b. We need a well-specified contract. What is the input and the output?
   c. Basic form: def, name of function, open close parenthesis, colon, block of code, and return.
   d. You can pass parameters.
   e. Variables are local. Their value is only maintained inside the function. Local environment.

3. IMPORTING LIBRARIES
   a. Python has a set of built-in functions - called the standard library ([http://docs.python.org/library/functions.html](http://docs.python.org/library/functions.html)). You can run things like: print(), raw_input(), abs(), range(), etc.
   b. But, you can also use other peoples’ functions!
      i. import math ([http://docs.python.org/library/math.html](http://docs.python.org/library/math.html))
         1. math.pi, math.sqrt(), ..
      ii. import random ([http://docs.python.org/library/random.html](http://docs.python.org/library/random.html))
         1. random.random(), random.seed()
      iii. import Tkinter ([http://docs.python.org/library/tkinter.html](http://docs.python.org/library/tkinter.html))

4. NESTED ITERATIONS
   a. Quick example in code using while loops and print statements.
   b. Varying where the print statements are located.
   c. Useful for debugging.

5. INFINITE LOOPS.
   a. Write a program that asks for user to answer ‘yes’ or ‘no,’ and doesn’t quit until it gets the answer.
   b. Variation that quits after 3 tries.

6. EXHAUSTIVE ENUMERATION
   a. Trying all reasonable values until you find all solutions.
   b. Write a program that find all divisors of X.

7. ITERATION: FOR LOOPS
   a. If I had a collection of elements, then I can simply walk through each of the items in the collection.
   b. Basic structure: for <variable> in <some_collection>: block of code
   c. Updates happen automatically.
d. range()
e. Re-implement Find all divisors of X.
f. As long as collection is finite, the program will terminate.
g. What if we want to collect things together? A compound of elements?

8. DATA STRUCTURES

9. LISTS
   i. a = [ x1, x2 , , ]
   ii. len(a)
   iii.a.count(value1)
   iv.a.insert(position, value)
   v. a.append(value)
   vi.a.index(value)  [ INDEXING ]
   vii.a.remove(value)
   viii.a.reverse()
   ix.a.sort()  [ SLICING ]
   x. a[0]

b. Lists as STACKS (last in, first out)
i. b = [x1, x2, x3 .. ]
ii. b.append(6)

LIST COMPREHENSION
   i. Demonstration with the “Find all divisors of X” program.
      ii. Then there are shortcuts:
          1. x = 10    divisors = [i for i in range(1,x) if x%i == 0]
      iii.Other interesting shortcuts:
          1. squares = [x**2 for x in range(10)]
          2. [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]

d. FILTER AND MAP
   i. Once you know functions, there are cool things you can do with lists.
      ii. FILTER:
          1. def even(x): return x%2 == 0
          2. filter(even, range(1,10))
      iii.MAP:
          1. def cube(x): return x*x*x
          2. map(cube, range(1,10))

e. NESTED lists
   i. matrix = [[1,2,3],[4,5,6],[7,8,9]]
   ii. matrix = [[i*j for i in range(3)] for j in range(4)]

f. Lists are MUTABLE.
   i. You can delete or modify elements in it.
      1. del a[0]
      2. del a[2:4]
      3. del a[:]

10.TUPLES and Sequences
   a. t = 12345, 54321, 'hello!'  
   b. You can also use parenthesis.
   c. The main difference is that they are INMUTABLE.
   d. (But they can contain mutable objects).
      i. t = [1,2,3], [3,4]
      ii. t[0]
      iii.t[0] = 1
iv. $t[0][2] = 1$
e. Usually they are used in situations when there are heterogenous sequence of elements.
f. **SEQUENCE PACKING**
   i. $t = 1, 2, \text{‘test’}$
   ii. $x, y, z = t$

11. **SETS**
   a. When you have lists, you can do:
      i. $a = [1, 2, 3]$
      ii. $b = [3, 4]$
      iii. $1 \text{ in } a, 3 \text{ in } b$
   b. If we convert them to sets,
      i. $a_1 = \text{set}(a)$
      ii. $b_1 = \text{set}(b)$
      iii. We can do, $a - b, a \mid b, a \& b, a \wedge b$

12. **DICTIONARIES**
   a. Instead of index by a range of numbers, index by key words.
      i. $\text{tel} = \{ \text{‘US’}:1, \text{‘VE’}:58 \}$
      ii. $\text{tel[‘UK’]} = 44$
      iii. $\text{del tel[‘VE’]}$
      iv. $\text{tel.keys()}$
      v. ‘US’ in $\text{tel}$

13. **RECURSION**
   a. Write a simple iterative program that calculates the factorial of $x$.
   b. A program is called recursive when it calls itself.
   c. Write a new version of the same program that is recursive.