EdX 6.00x Notes

Lecture 6:

- Data Types Review:
  - Have seen a sampling of different classes of algorithms
    - Exhaustive enumeration
    - Guess and check
    - Bisection
    - Divide and conquer
  - All have been applied so far to simple data types
    - Numbers
    - Strings
- Compound Data Types:
  - Tuples
  - Lists
  - Dictionaries
- Tuples:
  - Ordered sequence of elements (similar to strings)
  - Elements can be more than just characters
  - Examples
    - t1 = (1, 'two', 3)
    - t2 = (t1, 'four')
  - Operations allowed on Tuples (with examples):
    - *Concatenation*: print(t1+t2)
    - *Indexing*: print((t1+t2)[3])
    - *Slicing*: print((t1+t2)[2:5])
    - *Singletons*: t3 = ('five',)
  - Can iterate over tuples just as we can iterate over strings.
  - Tuples are immutable, meaning you cannot change a tuple's individual elements
- Lists:
  - Look a lot like tuples
    - Ordered sequence of values, each identified by an index
    - Using brackets rather than parenthesis.
      - Example: [1,2,3] rather than (1,2,3)
    - Singletons are now just [4] instead of (4,)
  - BIG DIFFERENCE
    - Lists are mutable!!
    - While tuple, int, float, str are immutable
    - So lists can be modified after they are created

- Importance of immutable versus mutable
    - Some data objects we want to treat as fixed
        - Can create new versions of them
        - Can bind variable names to them
        - But don't want to change them
        - Generally valuable when these data objects will be referenced frequently but elements don't change
    - Some data objects may want to support modifications to elements, either for efficiency because elements are prone to change
    - Mutable structures are more prone to bugs in use, but provide great flexibility.
        - However the downside to this is an increased likelihood of programming errors.
- Append Method
    - Example: Techs.append('RPI')
    - Append is a method (hence the.) that has a side effect
        - It doesn't create a new list, it mutates the existing one to add a new element to the end
    - Note:
        - Tech.append(Ivys)
            - Has a side effect by mutating a list
            - returns ['MIT', 'Cal Tech', 'RPI' ['Harvard', 'Yale', 'Brown']]
        - Flat = Tech + Ivys
            - Creates a new list through concatenation
            - returns ['MIT', 'Cal Tech', 'RPI', 'Harvard', 'Yale', 'Brown']
- Aliasing
    - When you create two distinct paths to a data object.
    - Convenient but treacherous.
- Note: The range() function does not work with floats.
- Operations on Lists:
    - Iteration
    - Note: When we mutate a list, we change it's length.
- Remove Method
    - Example: Techs.remove('RPI')
    - Mutates existing list to remove an element specified in the method argument.
- Cloning a List:
    - Cloning creates a copy of the list.
    - Useful when you want to iterate over a list and mutate elements in original list
    - Example: List2 = List1[:]
    - Note: List2 = List1 is **not sufficient**! That creates a new list that has a pointer to the old list versus a copy.
- Useful link for list methods
    - http://docs.python.org/2/tutorial/datastructures.html

- Functions as Objects:
    - They have types
    - They can be elements of data structures like lists
    - They can appear in expressions
        - As part of an assignment statement
        - As an argument to a function!!
    - Particularly useful to use functions as arguments when coupled with lists.
        - Aka higher order programming
- Generalizations of higher order functions
    - Python provides a general purpose HOP, map
    - Simple form – a unary function and a collection of suitable arguments
        - map(abs, [1,-2,3,-4])
        - Result: [1,2,3,4]
    - General form – an n-ary function and n collections of arguments
        - L1 = [1,28,36]
        - L2 = [2,57,9]
        - Map(min, L1, L2)
        - Result: [1, 28, 9]
- Dictionaries
    - Dict is a generalization of lists, but now indices don't have to be integers – can be values of a immutable type
    - Refer to indices as **keys**, since arbitrary in form
    - A dict is then a collection of <key, value> pairs
    - Syntax
        - monthNumbers = {'Jan':1, 'Feb':2}
    - Entries in a dict are unordered and can only be accessed by a key, not an index
    - Note: Keys must be immutable, so have to use a tuple and not a list
- Operations on Dictionaries
    - Insertion:
        - Example: monthNumbers['Apr'] = 4
    - Iteration:
        - Example: Use append in a for loop to iterate over a dictionary and populate an empty list.
- keys method
    - .keys() returns all the keys in a dictionary
        - Example: monthNumbers.keys()