Week 11, Lecture 2

Today we'll learn about SETS and DICTIONARIES. Both are built-in python objects.

A set can have any objects as members, is mutable, cannot have duplicates, and is unordered. Unordered means we cannot walk through it in any particular order.

File **sets.py** shows you how to (a) construct a set and (b) how to do the usual simple operations on sets (see the last part of the file). Since sets can't have duplicate elements one common trick to remove duplicate elements from a list is to convert it to a set and then reconvert the result to a list. It is good practice for you to write a simple program which allows the user to input n items and then built a set of these items, and then perform a few of the usual set operations on this set.

Dictionaries are objects that you will end up using a lot, as you will see in Examples 4 and 5. Example 5 is also explained in the lecture pdf for today, but perhaps in a slightly different way. You can work with either, but it may be a teeny bit easier for you to look at the explanation in the pdf first if you have trouble understanding the code in Example 5. But at this point you really should have no difficulty with Example 5.

A dictionary is made up of <key:value> pairs. The key is immutable — usually a number or a string (we explain why

in a program file). You can change the value associated with a key. For example if a dictionary is made up for a phone book, you cannot change a person's name (the name is the key) but you can certainly change the person's phone number (the number is the value associated with that name (or key)).

In **Example 1** we show simple examples of dictionaries and how we can add and remove pairs, and also how to look for a particular key in a dictionary (just like looking for a person's phone number if you know the name (or key)). We show how you can walk through a dictionary and print out just the keys, just the values, or both the key and value pairs.

Example 2 is just a simple review of some sequences.

Example 3:

Since dictionaries are unordered, we have no control over the order in which they will be printed. Python does its own optimization to store information in the most efficient way it can, and this order is not available to us and perhaps could change. So we cannot assume anything about the order of dictionary pairs.

However, if we want dictionary pairs to be output in ascending/descending order of either keys or values (we can order only one of the two, we cannot order both, just as we cannot tell if (1,2) is smaller than (2,1) — that pair

cannot be ordered) we have to extract just that part of the pair and make a list out of it. Then we can easily order the list, and finally use that list to walk through and print <key:value> pairs in ascending or descending order. We do this in **Example 3**.

To do the ordering we use the “items()” method of dictionaries which returns to us a “list” of all the pairs in the dictionary. Though you are given the pairs in some order, do not make any assumptions about order. Finally, we put “list” in quotes because it’s only a list in the English sense of the word. If you use the type() function on what is returned by the items() method, you’ll see its a “dictionary object”. So, to get a list on that object, you have to apply the list() function to it. Once that is done you can use list methods such as sort() on it, and you can also give it a key on which to sort. Here the word “key” is used in a more general sense, and you should not link this to a dictionary key. Here, by key we mean a way to tell the sort() method what sort of values to use in arranging things in ascending or descending order. For example, should it order based on dictionary keys or dictionary values? The “key” parameter in sort will give the sort method a helper function which returns a value that the sort() can use. That is all a “key” means to the sort() method. Don’t confuse this use of “key” with dictionary keys. [In a more general sense, both uses of the word “key” refer to a way to reach something …. a value we are looking for.

Example 4: Here we look at how to use a dictionary to

convert numbers (of any length) from one base to another base. You can extend this example to do arbitrary base conversions. We focus on converting from octal numbers (base 8) to binary numbers (base 2).

Example 5: Here we use a poem of Wordsworth (written in 1802, and one ranking lists it as the 5th most liked poem in England) and dictionaries to count the frequencies of words occurring in the poem. It's a simple program. You'll see a small variation of this explained in the lecture pdf. You may want to start with that explanation and code it and then look at this example. Both are simple.

Remember, the best way to understand what objects (e.g., dictionary objects) do is to create a simple problem, such as we have done in Example 5, and use the object. If you do this two or three times you will not ever need to commit anything to memory. You will always remember what you need when the time comes.