Lists of Dictionaries

Dictionaries can also be used along with lists to store a great deal of related data about groups of people, places or things. Remember that a list contains a number storage spots for data called elements. Each element is identified with a subscript (index). To this point, each element has contained a single piece of data. However each element in a list can contain a dictionary containing multiple key:value pairs.

Study the example below which loads a 4 element list of student information. Each element is a dictionary with 3 key : value pairs (fields of data).

```
classOfStudents = [ ]        # Creates an empty list called students

for i in range (0, 4): # Loop is required to store information about 4 students. Note that a while loop can also be used
        record ={}      # Creates an empty dictionary called record to store information about a single student. This must be in the loop.

        record ["Name"] = input ("Enter the name")              # Inputs the student name into the dictionary. Note "Name" is the key
        record ["Age"] = input ("Enter the age")                # Inputs the student age into the dictionary. Note "Age" is the key
        record ["Average"] = float (input("Enter average"))     # Inputs the student average into the dictionary. Note "Average" is the key

        classOfStudents .append (record)    #Appends the list with a new element. The element is a dictionary with 3 key:value pairs

Outputs all of the data in element classOfStudents [1]. Note element comes before the key
print (classOfStudents [ 1 ] ["Name"], "is", classOfStudents [ 1 ] ["Age"], "and has an average of", classOfStudents [ 1 ] ["Average"])

for index in classOfStudents:        # Iterates through the list one element at a time

    print (index  ["Name"], "is", index ["Age"], "and has an average of", index ["Average"])
```

Sorting a List of Dictionaries

In order to sort an list of dictionaries, the computer must know which key to base the sort on. This is done with the ***itemgetter ()*** function which must be imported from the *operator* module.

To sort the list that was created in the example above:

1) Add the following as the first line of the program:
   ```
   from operator import itemgetter # Load the operator module which contains the itemgetter function
   ```

2) Add the following line between the loop which is used to input the data and the loop to output the data:
   ```
   classOfStudents.sort (key = itemgetter ("Name"))   # Sorts the list classOfStudents into alphabetical order based on the Name key
   ```

The *sorted ()* Function

Python has a ***sort ()*** function which is used with lists to permanently sort the list - *names.**sort ()*** will sort a list called *names* in alphabetical order). This is useful if you want the data to be permanently sorted.

However, sometimes a programmer wishes to sort the data in the list for some use but keep the initial data in its original non-sorted state. In the unit on lists, the standalone function called ***sorted ()*** was discuss which creates a new sorted list based on the original list without affecting the original list.

In the example above, if you wish to keep the original list as is (ie. not sorted), use the following in step #2 above:

```
sortedClass = sorted (classOfStudents , key= itemgetter ("Name")) # Will create a new list with the names in the list classOfStudents
                                                                  # sorted alphabetically based on the Name key
                                                                  # You will obviously have to adjust your second loop to output
                                                                  # sortedClass instead of classOfStudents
```