More about Lists & Dictionaries
Multi Dimensional Lists

- All of the lists we have been creating so far have been one dimensional (i.e. linear) in nature.
- High level programming languages also have the ability to construct lists that can store multiple values within the same element.
Multi Dimensional Lists

- A one dimensional list can be thought of as a line
- A two dimensional list can be thought of as a plane
- A three dimensional list can be thought of as a cube
- … etc!
Creating a two dimensional list

You can create a two dimensional list in Python by simply nesting a list inside of another list. Example:

```python
calist = [ ['a', 'b', 'c'], ['d', 'e', 'f'] ]

cprint (calist[0])
cprint (calist[0][0])
cprint (calist[1][1])
```

```
>> [ 'a', 'b', 'c' ]
>> a
>> e
```
Dictionaries

- A Dictionary in Python is a sequence object like a list.
- Unlike a list, a Dictionary doesn’t use integer based index values.
- Instead, Dictionaries use immutable objects (like Strings) to index their content.
- In other languages Dictionaries are often referred to as “Associative Arrays” or “Hashmaps”
Lists vs. Dictionaries

- Sequence Structure
- Mutable (can be changed if you know which index you are modifying)
- Items are stored in a particular order based on index values
- Items can be indexed using an integer
Lists vs. Dictionaries

- Sequence Structure
- Mutable (can be changed if you know which index you are modifying)
- Items are not stored in any particular order
- Items can be indexed using anything that is immutable (String, etc)
Creating a Dictionary

❖ You can create a Dictionary using the curly braces – “{“ and “}”, like this:

```
my_dictionary = { }  
```

❖ This will create an empty Dictionary
Adding items to a Dictionary

❖ In order to add an item to a Dictionary you need to specify a “key” – this is usually in the form of a String

❖ You can then associate some data with that key. For example I could associate the number 3.2 with the key “python” by doing this:

```python
my_dictionary[“python”] = 3.2
```

❖ This will place the number 3.2 into the Dictionary at the position marked by the String “python”
Accessing Dictionary items

- You can access all items in a Dictionary by printing it out, like this:

  ```python
  print ( my_dictionary )
  ```

- However, you often just want to access one item – this works the same as with an array, but you will use a key instead of an integer index:

  ```python
  print ( my_dictionary["python"] )
  ```
Creating a Dictionary with Values

- Dictionaries store key / value pairs. You can initialize a Dictionary with a known set of key / value pairs by using the following syntax:

```python
my_dictionary = { "python":3.2, "java":1.8 }
```

- This will create a Dictionary with the keys “python” and “java”
Invalid Indexes

❖ Note that you cannot access elements in a Dictionary that have not been defined. This would raise an exception if “java” was not a key in the Dictionary:

```python
print ( my_dictionary["java"] )
```

❖ You can use the “in” operator to test to see if a key is in a dictionary like this:

```python
if ( "java" in my_dictionary" ):
```

❖ Note that this will check for the presence of a key in a dictionary, not for the data that the key is storing!
Invalid Indexes

- Note that you cannot access elements in a Dictionary that have not been defined. This would raise an exception if “java” was not a key in the Dictionary:

  ```
  print ( my_dictionary[“java”] )
  ```

- You can use the “in” operator to test to see if a key is in a dictionary like this:

  ```
  if ( “java” in my_dictionary” ):
  ```

- Note that this will check for the presence of a key in a dictionary, not for the data that the key is storing!
Deleting a key in a Dictionary

- You can use the `del` command to delete a key in a Dictionary, like this:

  ```python
del my_dictionary["java"]
```

- Make sure that you know that the key in question has been defined in the Dictionary before you run this command!
Clearing a Dictionary

- You can clear all keys in a Dictionary by doing the following:

```python
my_dictionary.clear()
```
Behind the Scenes: How Dictionaries work

A hash function maps keys to buckets. The diagram shows how keys (John Smith, Lisa Smith, Sandra Dee) are hashed and stored in buckets (00, 01, 02, 03, 13, 14, 15). The buckets store corresponding values (521-8976, 521-1234, 521-9655).
Working with Dictionary Keys and Values
Finding all the keys in a Dictionary

❖ In a list we can easily visit all "slots" in the list by visiting every index value in the list, like this:

```python
mylist = [ 'a', 'b', 'c', 'd' ]
for i in range(0, len(mylist)):
    print (mylist[i])
```

❖ However, we can't do this with a Dictionary since the index values for a dictionary are not necessarily going to be integers

❖ We can use the "keys()" method to ask a Dictionary object to expose all of the keys that are defined within that Dictionary like this:

```python
my_dictionary.keys()
```
Iterating over every item in a Dictionary

- To iterate over every time in a Dictionary you need to be able to visit every key value (as opposed to simply knowing the size of a list and iterating over the integer range of the list)

- You can extract all the keys from a Dictionary by doing the following:

  ```python
  for key in my_dictionary.keys():
  ```

- The target variable “key” will assume each key value in your Dictionary as the loop progresses.

- You can print out all items with their keys by doing the following:

  ```python
  for key in my_dictionary.keys():
      print (key, " == ", my_dictionary[key])
  ```
Iterating over every item in a Dictionary

- There is no guarantee that the keys() method will return the keys of a dictionary in any particular order.
- However, you can ask Python to sort the keys before you iterate over them, like this:

```python
for key in sorted( my_dictionary.keys() ) :
```

- This will sort the keys in ascending order, which then lets you access the elements in the dictionary in ascending order.
Finding all the values in a Dictionary

❖ You can also iterate over just the values in a dictionary (not just the keys) using this syntax:

```python
for v in my_dictionary.values():
    print (v)
```

❖ Note that doing this will only expose the values in a dictionary and not the key – this means that you cannot change the values in the dictionary using this method. This is analogous to iterating over a list like this:

```python
for item in my_list:
    print (item)
```
Iterating over every item in a Dictionary

- You can also iterate over a Dictionary by using the following technique to extract both the key and the value at the same time:

```python
for key, value in my_dictionary.items():
    print (key, value)
```
Programming Challenges
Programming Challenge
Programming Challenge

- Write a program that asks the user to enter in a series of student names and test scores
- Use a Dictionary to store a new record in the dictionary based on the student name (i.e. the name becomes the key) – store the student score at that position.
- When you are finished print out the names and scores of all students
- Extension: print them out in alphabetical order!
Programming Challenge
Programming Challenge

❖ Extension: Allow the user to enter in multiple scores for each student. If the student already has a score simply add it to the total score for the student.

❖ You will also need to keep track of the number of tests somehow
  ❖ You could create a separate dictionary for # of tests
  ❖ You could store a list in a single dictionary that has two fields – one for points and one for number of tests
Programming Challenge

- Count the frequency of all words in the Declaration of Independence