**PYTHON WEEK 3 SUMMARY**

**Day 1 – Data Structures**

**Defining functions -** The keyword def introduces a function definition. It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line and must be indented.

**More on lists –** Here are all of the methods of list objects:

* list.append(x) - Add an item to the end of the list. Equivalent to a[len(a):] = [x].
* list.extend(iterable) - Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.
* list.insert(I, x) - Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).
* list.remove(x) - Remove the first item from the list whose value is equal to x. It raises a ValueError if there is no such item.
* list.pop([i]) - Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position.
* list.clear() - Remove all items from the list. Equivalent to del a[:].
* list.index(x[, start[, end]]) - Return zero-based index in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item. The optional arguments start and end are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the start argument.
* list.count(x) - Return the number of times x appears in the list.
* list.sort(key=None, reverse=False) - Sort the items of the list in place.
* list.reverse() - Reverse the elements of the list in place.
* list.copy() – Return a shallow copy of the list.

**List comprehensions -** List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

**Day 3 – Errors and Exceptions**

**Errors and Exceptions -** Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python.

**Raising an exception –** You can use raise to throw an exception if a condition occurs. The statement can be complemented with a custom exception.

Instead of waiting for a program to crash midway, you can also start by making an assertion in Python. We assert that a certain condition is met. If this condition turns out to be True, then the program can continue. If the condition turns out to be False, you can have the program throw an AssertionError exception.

**Day 4 – Classes**

**Python Classes -** Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.