**5.3 Methods for Sequence Types**

**5.3.1 Methods for All Sequence Types**

There are some methods that can be used for all sequence types, such as lists, tuples, and strings. These will be demonstrated here for lists.

The **count** method returns the number of occurrences of an item in a list.

*>>> mylist = list("monty python")*

*>>> print(mylist)*

['m', 'o', 'n', 't', 'y', ' ', 'p', 'y', 't', 'h', 'o', 'n']

*>>> mylist.count('o')*

2

*>>> mylist.count('x')*

0

The **index** method returns the index of the first occurrence of an item within a list, or an error message if it is not found.

*>>> mylist.index('y')*

4

*>>> mylist.index('x')*

ValueError: 'x' is not in list

**5.3.2 Methods for Mutable Sequence Types**

There are methods that can be used for all mutable sequence types. They will be demonstrated here using lists.

The following methods modify the list variable. The value that is returned is not the list, but the default value **None**. These methods will be shown in a series of statements.

The **append** method adds one item to the end of a list.

*>>> numlist = [5, 11, 33, 11, 2, -6]*

*>>> outval = numlist.append(44)*

*>>> print(numlist, outval)*

[5, 11, 33, 11, 2, -6, 44] None

The **insert** method inserts an item into the list at the specified index. The following inserts 14 into *numlist[2]*, and moves the rest of the items.

*>>> numlist.insert(2, 14)*

*>>> numlist*

[5, 11, 14, 33, 11, 2, -6, 44]

The **extend** method is used to concatenate another list to the end of the current list.

*>>> numlist.extend([17, -5])*

*>>> numlist*

*[5, 11, 14, 33, 11, 2, -6, 44, 17, -5]*

The result of this call to **extend** is the same as *numlist += [17, -5]*.

Note that if **append** had been used instead of **extend**, the result would be a nested list ( a list in which the list [17, -5] would be the last element). This is demonstrated with a list of characters.

*>>> charlist = ['x', 'q', 's']*

*>>> charlist.extend(['y', 'z'])*

*>>> charlist*

['x', 'q', 's', 'y', 'z']

*>>> charlist.append(['a', 'b'])*

*>>> charlist*

['x', 'q', 's', 'y', 'z', ['a', 'b']]

The **remove** method can be used to remove a specified item from a list. If the item is not in the list, an error message results.

*>>> numlist*

[5, 11, 14, 33, 11, 2, -6, 44, 17, -5]

*>>> numlist.remove(-6)*

*>>> numlist*

[5, 11, 14, 33, 11, 2, 44, 17, -5]

*>>> numlist.remove(100)*

ValueError: list.remove(x): x not in list

If there are multiple occurrences of the specified item, only the first will be removed from the list.

The **pop** method removes an item from a specified index in a list. Unlike the previous methods, however, it returns a value (not **None**), which is the item that has been removed. The index can be specified using an integer. If no index is given, the default is -1 (which means the last element in the list).

*>>> delval = numlist.pop(3)*

*>>> print(delval, 'is no longer in ', numlist)*

33 is no longer in [5, 11, 14, 11, 2, 44, 17, -5]

*>>> delval = numlist.pop()*

*>>> print(delval, ' is no longer in ', numlist)*

-5 is no longer in [5, 11, 14, 11, 2, 44, 17]

The **reverse** method reverses all elements in a list; modifying the list and returning **None**.

*>>> numlist.reverse()*

*>>> numlist*

[17, 44, 2, 11, 14, 11, 5]

The **clear** method removes everything from a list, so that it becomes an empty list.

*>>> numlist.clear()*

*>>> numlist*

[]

**5.3.3 List Method**

In addition to the methods from the previous section, which can be used on any mutable sequence type, the **list** type includes a method for sorting the list, **sort**. The **sort** method returns **None**, but sorts the list in place.

*>>> numlist = [5, 11, 33, 11, 2, -6]*

*>>> numlist.sort()*

*>>> numlist*

[-6, 2, 5, 11, 11, 33]

By default, it sorts in ***ascending*** order, (lowest to highest). This can be changed to ***descending*** order by setting the reverse flag to **True** in the function call:

*>>> numlist.sort(reverse=True)*

*>>> numlist*

[33, 11, 11, 5, 2, -6]

Note that there is also a built-in function in Python, **sorted**, which will return a new list that has been sorted, but does not sort the argument list in place.

*>>> origlist = list('python')*

*>>> newlist = sorted(origlist)*

*>>> print(origlist, newlist)*

['p', 'y', 't', 'h', 'o', 'n'] ['h', 'n', 'o', 'p', 't', 'y']