ICS3U: Adding and Removing List Elements

Adding Elements

While tuples provide a convenient way to group items, we are limited in what we can do with them. Once a tuple is created, it cannot be modified. It is not possible to add or remove elements from a tuple. Lists, on the other hand, *can* be modified. Thus, if we need to make changes to a collection of objects, lists are the way to go.

There are two ways to add elements to a list: by **value**, and by **index**. The simplest way to add an element by value to an existing list is to use the `append` method. This will add the element to the end of the list, so this operation is useful when order is not important.

```
>>> integers = [7, 2, 1]
>>> integers.append(4)
>>> integers
[7, 2, 1, 4]
```

A common mistake is to try to append more than one element in the call to `append`.

```
integers.append(3, 5)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: append() takes exactly one argument (2 given)
```

If a list containing one or more elements is appended to a list, they will not be appended separately. Instead, the appended list or tuple will remain as a sequence.

```
>>> integers.append([3, 5])
>>> integers
[7, 2, 1, 4, [3, 5]]
>>> len(integers)
5
>>> integers[-1]
[3, 5]
```

If order is important, an index can be specified using the `insert` method. This will place the new element in that location in the list, and shift all subsequent elements down one index.

```
>>> integers.insert(0, 99)
>>> integers
[99, 7, 2, 1, 4, [3, 5]]
```

If the index is out of range, Python will simply insert at the end (or beginning) of the list.

```
>>> integers.insert(9999, 42)
>>> integers
[99, 7, 2, 1, 4, [3, 5], 42]
```

Adding all of the elements of one list to another one can be done in a few ways. One is to use the `+` or `+=` operators to concatenate the two lists. This will append one list to the end of the other.

```
>>> more_integers = [10, 20, 30]
>>> integers += more_integers
>>> integers
[99, 7, 2, 1, 4, [3, 5], 42, 10, 20, 30]
```

Another way to achieve the same result is to use the `extend` method.

```
>>> even_more_integers = [2, 3]
>>> integers.extend(even_more_integers)
>>> integers
[99, 7, 2, 1, 4, [3, 5], 42, 10, 20, 30, 2, 3]
```

To insert the elements of one list into another one as separate elements, rather than as a list, concatenation and slicing is probably the best solution.

```
>>> still_more_integers = [25, 16, 7]
>>> integers = integers[:4] + still_more_integers + integers[4:]
>>> integers
[99, 7, 2, 1, 25, 16, 7, 4, [3, 5], 42, 10, 20, 30, 2, 3]
```

Removing Elements

Like adding elements, removing elements can be done either by value or by index. To remove an element by value, Python provides the `remove` method.

```
>>> integers
[99, 7, 2, 1, 25, 16, 7, 4, [3, 5], 42, 10, 20, 30, 2, 3]
>>> integers.remove(10)
>>> integers
[99, 7, 2, 1, 25, 16, 7, 4, [3, 5], 42, 20, 30, 2, 3]
>>> integers.remove([3, 5])
>>> integers
[99, 7, 2, 1, 25, 16, 7, 4, 42, 20, 30, 2, 3]
```

If there are multiple elements with the same value, the *first* element – the one with the lowest index – will be removed.

```
>>> integers
[99, 7, 2, 1, 25, 16, 7, 4, 42, 20, 30, 2, 3]
>>> integers.remove(7)
[99, 2, 1, 25, 16, 7, 4, 42, 20, 30, 2, 3]
```

Attempting to remove an element that does not exist in a list will cause a run-time error.

```
>>> integers.remove(1000)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

To be safe, always check if the element is in the list using `in`, before calling `remove`.

```
val = int(input("Remove what value? "))
if val in integers:
   integers.remove(val)
   print("Value removed from list.")
else:
   print("Value is not in the list.")
```

Removing all instances of a particular value from a list can be done using a loop.

```
val = int(input("Remove what value? "))
count = 0
while val in integers:
   integers.remove(val)
   count += 1
print("Removed", count, "instances of", val)
```

There is no list method to remove by index. However, the `del` function will remove an element from a specific location.

```
>>> integers
[99, 2, 1, 25, 16, 7, 4, 42, 20, 30, 2, 3]
>>> del(integers[3])
>>> integers
[99, 2, 1, 16, 7, 4, 42, 20, 30, 2, 3]
```

Specifying an index that is out of range will cause an error.

```
>>> del(integers[99])
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
IndexError: list assignment index out of range
```

To prevent this, it is a good idea to verify that the index is valid before using `del`.

```
i = int(input("Remove value at which index? "))
if i < len(integers) and i >= -len(integers):
   del(integers[i])
else:
   print("Index is out of range.")
```

Finally, there are a number of different ways to delete all elements from a list. The simplest is to reassign the variable an empty list using either `[]` or `list`.

```
>>> L1 = [1, 2, 3]
>>> L1 = []
>>> L1
[]
>>> L2 = [4, 5, 6]
>>> L2 = list()
>>> L2
[]
```

A third method which is possibly easier to understand semantically is to use `clear`.

```
>>> L3 = [7, 8, 9]
>>> L3.clear()
>>> L3
[]
```