Advanced Lists

In this series of lectures we will be diving a little deeper into all the methods available in a list object. These aren't officially "advanced" features, just methods that you wouldn't typically encounter without some additional exploring. Its pretty likely that you've already encountered some of these yourself!

Lets begin!

In [1]:
```
l = [1,2,3]
```

append

You will have definitely have use this method by now, which merely appends an element to the end of a list:

In [2]:
```
l.append(4)

l
```

Out[2]:
```
[1, 2, 3, 4]
```

count

We discussed this during the methods lectures, but here it is again. count() takes in an element and returns the number of times it occures in your list:

In [4]:
```
l.count(10)
```

Out[4]:
```
0
```

In [5]:
```
l.count(2)
```

Out[5]:
```
1
```

extend

Many times people find the difference between extend and append to be unclear. So note:

append: Appends object at end

In [6]:
```
x = [1, 2, 3]
x.append([4, 5])
print x
```

```
[1, 2, 3, [4, 5]]
```

extend: extends list by appending elements from the iterable

In [7]:
```
x = [1, 2, 3]
x.extend([4, 5])
print x
```

```
[1, 2, 3, 4, 5]
```

Note how extend append each element in that iterable. That is the key difference.

index

index will return the index of whatever element is placed as an argument. Note: If the the element is not in the list an error is returned.

In [10]:
```
l.index(2)
```

Out[10]:
```
1
```

In [11]:
```
l.index(12)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-3f090052d656> in <module>()
----> 1 l.index(12)

ValueError: 12 is not in list
```

insert

insert takes in two arguments: insert(index,object) This method places the object at the index supplied. For example:

In [12]:
```
l
```

Out[12]:
```
[1, 2, 3, 4]
```

In [13]:
```
# Place a letter at the index 2
l.insert(2,'inserted')
```

In [14]:
```
l
```

Out[14]:
```
[1, 2, 'inserted', 3, 4]
```

pop

You most likely have already seen pop(), which allows us to "pop" off the last element of a list.

In [15]:
```
ele = l.pop()
```

In [16]:
```
l
```

Out[16]:
```
[1, 2, 'inserted', 3]
```

In [17]:
```
ele
```

Out[17]:
```
4
```

remove

The remove() method removes the first occurrence of a value. For example:

In [18]:
```
l
```

Out[18]:
```
[1, 2, 'inserted', 3]
```

In [19]:
```
l.remove('inserted')
```

In [20]:
```
l
```

Out[20]:
```
[1, 2, 3]
```

In [21]:
```
l = [1,2,3,4,3]
```

In [22]:
```
l.remove(3)
```

In [23]:
```
l
```

Out[23]:
```
[1, 2, 4, 3]
```

reverse

As you might have guessed, reverse() reverses a list. Note this occurs in place! Meaning it effects your list permanently.

In [24]:
```
l.reverse()
```

In [25]:
```
l
```

Out[25]:
```
[3, 4, 2, 1]
```

sort

sort will sort your list in place:

In [26]:
```
l
```

Out[26]:
```
[3, 4, 2, 1]
```

In [27]:
```
l.sort()
```

In [28]:
```
l
```

Out[28]:
```
[1, 2, 3, 4]
```

Great! You should now have an understanding of all the methods available for a list in Python!