😊 Advanced List

1. We start off by creating a list.

```
[1]:  list1 = [1,2,3]
```

2. Here we have appended the list with a new number using the append() method.

```
[2]:  list1.append(4)

      list1

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

3. We already have seen the count() method. Here is one more example of it.

```
[3]:  list1.count(10)

[3]:  0

[4]:  list1.count(2)

[4]:  1
```

4. Mostly, the difference between append and extend is unclear to people. Here is an example of it.

append: appends whole object at end:

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

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

extend: extends list by appending elements from the iterable:

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

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

5. The index () method will return the index of whatever element is placed as an argument. Note: If the element is not in the list, an error is raised.

```
[7]: list1.index(2)

[7]: 1

[8]: list1.index(12)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-56b94ada72bf> in <module>()
----> 1 list1.index(12)

ValueError: 12 is not in list
```

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

```
[9]: list1

[9]: [1, 2, 3, 4]

[10]: # Place a letter at the index 2
      list1.insert(2,'inserted')

[11]: list1

[11]: [1, 2, 'inserted', 3, 4]
```

7. You most likely have already seen pop(), which allows us to "pop" off the last element of a list. However, by passing an index position, you can remove and return a specific element.

```
[12]: ele = list1.pop(1)   # pop the second element

[13]: list1

[13]: [1, 'inserted', 3, 4]

[14]: ele

[14]: 2
```

8. The remove () method removes the first occurrence of a value.

```
[15]: list1

[15]: [1, 'inserted', 3, 4]

[16]: list1.remove('inserted')

[17]: list1

[17]: [1, 3, 4]

[18]: list2 = [1,2,3,4,3]

[19]: list2.remove(3)

[20]: list2

[20]: [1, 2, 4, 3]
```

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

```
[21]: list2.reverse()

[22]: list2

[22]: [3, 4, 2, 1]
```

10. The sort method will sort your list in place. The sort() method takes an optional argument for reverse sorting. Note this is different than simply reversing the order of items.

```
[23]: list2

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

[24]: list2.sort()

[25]: list2

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

11. A common programming mistake is to assume you can assign a modified list to a new variable. While this typically works with immutable objects like strings and tuples.

```
[28]: x = 'hello world'
```

```
[29]: y = x.upper()
```

```
[30]: print(y)
```

```
HELLO WORLD
```

This will NOT work the same way with lists:

```
[31]: x = [1,2,3]
```

```
[32]: y = x.append(4)
```

```
[33]: print(y)
```

```
None
```

12. What happened? In this case, since list methods like append() affect the list *in-place*, the operation returns a None value. This is what was passed to **y**. In order to retain **x** you would have to assign a *copy* of **x** to **y**, and then modify **y**:

```
[34]: x = [1,2,3]
      y = x.copy()
      y.append(4)
```

```
[35]: print(x)
```

```
[1, 2, 3]
```

```
[36]: print(y)
```

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