05-Advanced Lists

July 12, 2018

1 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. It's pretty likely that you've already encountered some of these yourself!

Let's begin!

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

1.1 append

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

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

        list1
```

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

1.2 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 occurs in your list:

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

```
Out[3]: 0
```

```
In [4]: list1.count(2)
```

```
Out[4]: 1
```

1.3 extend

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

append: appends whole object at end:

```
In [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:

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

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

Note how extend() appends each element from the passed-in list. That is the key difference.

1.4 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 raised.

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

Out[7]: 1

In [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
```

1.5 insert

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

```
In [9]: list1

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

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

In [11]: list1

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

1.6 pop

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.

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

In [13]: list1

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

In [14]: ele

Out[14]: 2
```

1.7 remove

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

```
In [15]: list1

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

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

In [17]: list1

Out[17]: [1, 3, 4]

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

In [19]: list2.remove(3)

In [20]: list2

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

1.8 reverse

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

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

In [22]: list2

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

1.9 sort

The sort() method will sort your list in place:

```
In [23]: list2

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

In [24]: list2.sort()

In [25]: list2

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

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

```
In [26]: list2.sort(reverse=True)

In [27]: list2

Out[27]: [4, 3, 2, 1]
```

1.10 Be Careful With Assignment!

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:

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

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

In [30]: print(y)

HELLO WORLD
```

This will NOT work the same way with lists:

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

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

In [33]: print(y)

None
```

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**:

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

In [35]: print(x)

[1, 2, 3]


In [36]: print(y)

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

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