List

December 18, 2020

1 List

1.1 Creating a list

```
In [37]: myList = [4, 2, 7, 5, 9]
         print(myList)
```

```
[4, 2, 7, 5, 9]
```

Also note that a list contain any data type, and it need not be homogenous.

```
In [62]: myList = [1, 5.4, 'c', 'pi', True]
         print(myList)
```

```
[1, 5.4, 'c', 'pi', True]
```

Also note that a list can contain another list as a single element.

```
In [61]: myList = [[1, 2, 3], "Hello", "World"]
         print(myList)
```

```
[[1, 2, 3], 'Hello', 'World']
```

1.2 Accessing values in through indices of the list

Note that a list's indices begin from 0.

```
In [66]: myList = [1, 5.4, 'c', 'pi', True]
         print(myList[0])
```

```
1
```

If a positive index is too large, we get an IndexError...

```
In [67]: print(myList[10])
```

```
    ---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call last)

    <ipython-input-67-fa2ec159c82e> in <module>()
----> 1 print(myList[10])


    IndexError: list index out of range
```

1.2.1 Negative indices

Index -1 accesses the last element of the list, index -2 accesses the second last element of the list, and so on...

```
In [64]: myList = [1, 5.4, 'c', 'pi', True]
         print(myList[-1])
```

```
True
```

If a negative index is too small, we get an IndexError...

```
In [68]: print(myList[-10])
```

```
    ---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call last)

    <ipython-input-68-e11d8f6afd4d> in <module>()
----> 1 print(myList[-10])


    IndexError: list index out of range
```

1.3 Operations of lists

1.3.1 Repeating elements

To create a list with repeating elements...

```
In [70]: myList = ["ha"] * 4
         print(myList)
```

```
['ha', 'ha', 'ha', 'ha']
```

1.3.2 Concatenating lists

To concatenate two or more lists...

```
In [71]: list1 = [1, 2, 3]
         list2 = ['a', 'b', 'c']
         list3 = [0.1, 0.2, 0.3]
         myList = list1 + list2 + list3
         print(myList)
```

```
[1, 2, 3, 'a', 'b', 'c', 0.1, 0.2, 0.3]
```

To repeat whole lists' elements in a concatenation...

```
In [74]: list1 = [1, 2, 3]
         list2 = ['a', 'b', 'c']
         list3 = [0.1, 0.2, 0.3]
         myList = list1 + 3 * list2 + 2 * list3
         print(myList)
```

```
[1, 2, 3, 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 0.1, 0.2, 0.3, 0.1, 0.2, 0.3]
```

1.3.3 Slicing lists

This allows you to access a certain range of indices in a list.

Note that when specifying the range, the smaller value should be written before the larger value.

A range is specified for the list as myList[lower : upper], where the lower index is included, and the upper index is excluded.

Positive range

```
In [89]: myList = [1, 2, 3, 'a', 'b', 'c', 0.1, 0.2, 0.3, "pi", "theta", "gamma"]
         portion = myList[3 : 6]
         print(portion)
```

```
['a', 'b', 'c']
```

Negative range Slicing can be done for negative indices also. But note that the smaller i.e. more negative value must come first...

```
In [91]: myList = [1, 2, 3, 'a', 'b', 'c', 0.1, 0.2, 0.3, "pi", "theta", "gamma"]
         portion = myList[-6 : -3]
         print(portion)
```

```
[0.1, 0.2, 0.3]
```

Unspecified range Not specifying the lower index makes the range start from 0.
Not specifying the upper index makes the range end at the last index of the list.

Step value A step value helps specify the arithmetic sequence of indices to be accessed.
By default, the step value is 1.
A step value is specified by a third argument, separated by a column...

```
In [96]: myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
         portion = myList[0 : 8 : 2]
         print(portion)

[1, 3, 5, 7]
```

```
In [99]: myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
         portion = myList[ : : 3]
         print(portion)

[1, 4, 7, 10]
```

A negative step value can only be applied on an unspecified range i.e. a range including the whole list.
A negative step value makes it so that the order of the range is reversed...

```
In [105]: myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
          print(myList[ : : -1])

[12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[]
```

1.3.4 List comprehension

This technique allows you to easily make a new list that is a function of the old list.

```
In [107]: oldList = [1, 2, 3, 4, 5, 6]
          newList = [i*i for i in oldList]
          print("Old list:")
          print(oldList)
          print("New list:")
          print(newList)

Old list:
[1, 2, 3, 4, 5, 6]
New list:
[1, 4, 9, 16, 25, 36]
```

```
In [109]: oldList = ['alpha', 'beta', 'gamma', 'theta', 'delta']
          newList = [i.upper() for i in oldList]
          print("Old list:")
          print(oldList)
          print("New list:")
          print(newList)
```

```
Old list:
['alpha', 'beta', 'gamma', 'theta', 'delta']
New list:
['ALPHA', 'BETA', 'GAMMA', 'THETA', 'DELTA']
```

1.4 More list modification methods

Append method The append method accepts only one argument.

```
In [57]: someList = []
         print("Before append:")
         print(someList)
         someList.append("Kitty")
         someList.append(1)
         print("After append:")
         print(someList)
```

```
Before append:
[]
After append:
['Kitty', 1]
```

However, that argument can also be a list...

```
In [52]: tmp = ["Alpha", "Beta", "Gamma"]
         someList = [1, 2, 3]
         someList.append(tmp)
         print(someList)
```

```
[1, 2, 3, ['Alpha', 'Beta', 'Gamma']]
```

As you can see, the whole list "tmp" is present as one element in "someList".

Insert method Inserts an element into a list at a specific index...

```
In [53]: someList = [1, 2, 4, 6]
         print("Before insertions:")
         print(someList)
         someList.insert(2, 3)
         someList.insert(4, 5) # You need to consider to previously added element as a part of
         print("After insertions:")
         print(someList)
```

```
Before insertions:
[1, 2, 4, 6]
After insertions:
[1, 2, 3, 4, 5, 6]
```

Pop method Returns and removes the last element of the list...

```
In [54]: someList = [1, 2, 3, 4, 5, 6]
         print("Before pop:")
         print(someList)
         item = someList.pop()
         print("After pop:")
         print(someList)
         print("Item popped:")
         print(item)
```

```
Before pop:
[1, 2, 3, 4, 5, 6]
After pop:
[1, 2, 3, 4, 5]
Item popped:
6
```

Remove method Removes a specific value from the list...

```
In [58]: someList = [1, 2, 3, 'a', 'b', 'c']
         print("Before removing:")
         print(someList)
         someList.remove('a')
         print("After removing:")
         print(someList)
```

```
Before removing:
[1, 2, 3, 'a', 'b', 'c']
After removing:
[1, 2, 3, 'b', 'c']
```

If you enter an element that does not exist, you get a value error...

```
In [59]: someList = [1, 2, 3, 'a', 'b', 'c']
         someList.remove('x')
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-59-0369505f4680> in <module>()
          1 someList = [1, 2, 3, 'a', 'b', 'c']
    ----> 2 someList.remove('x')


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

1.5 Iterating over a list

```
In [11]: from fractions import Fraction
         myList = [1, 4.4, Fraction(2, 3), 5, 4, "pi", "e"]
         for i in myList:
             print(i)
```

```
1
4.4
2/3
5
4
pi
e
```

1.5.1 Iterating with indices

```
In [108]: from fractions import Fraction
          myList = [1, 4.4, Fraction(2, 3), 5, 4, "pi", "e"]
          for index, item in enumerate(myList):
              print(index, item)
```

```
0 1
1 4.4
2 2/3
3 5
4 4
5 pi
6 e
```

1.6 Searching a list

```
In [20]: from fractions import Fraction
         myList = [1, 4.4, Fraction(2, 3), 5, 4, "pi", "e"]
```

```
if "pi" in myList:
    print("Yup")
else:
    print("Nope")
```

```
Yup
```