List Methods

This notebook looks at list methods. We will first study the concatenation `+` operator and the multiplication `*` operator in the context of lists and then turn our attention to list methods.

There are a number of list methods available for list processing. These methods are accessed in the following way: `nameoflistvariable.methodname().` **We will discuss the following methods in this class:**

`append()` : adds a passed object to the end of an existing list

`extend()` : adds the contents of a sequence to the end of an existing list

`count()` : counts the number of times an element occurs in a list

`index()` : returns the index of the first occurence of the argument

`insert()` : inserts the object at the index provided

`pop()` : removes and returns an item from the list. If no argument is provided, the method returns the last item from the list. If an argument is provided (it should be a valid index), the method returns the element at that index.

`remove()` : removes an item from the list. It accepts the item to be removed as an argument and returns the first matching value. It returns nothing.

`reverse()` : reverses the objects of a list in place. That means, nothing is returned and the list is changed.

`sort()` : sorts objects in a list.

We can concatenate two lists together using the concatenation (`+`) operator. Note that applying the `+` operator to a mix of a list and a single value is not allowed.

In [1]:

```
a = [1, 2, 3]
b = ['4', '5', 6]
c = a + b

print(c)
print(a)
print(b)
```

```
[1, 2, 3, '4', '5', 6]
[1, 2, 3]
['4', '5', 6]
```

We can use the multiplication operator (`*`) to repeat a list multiple times. Note that you cannot multiply a list with another list. The `*` operator is commutative.

In [2]:

```
aList = ['xyz', 'zara', 'abc', 'xyz']
newList = aList*2
print(aList)
print(newList)
```

```
['xyz', 'zara', 'abc', 'xyz']
['xyz', 'zara', 'abc', 'xyz', 'xyz', 'zara', 'abc', 'xyz']
```

The `append()` method adds a passed object to the end of an existing list. This method modifies the original list. This is a void method.

In [3]:

```
aList = [123, 'xyz', 'zara', 'abc']
```

```
aList.append([2009,5678])
print("Updated List : ", aList)
```

```
Updated List :  [123, 'xyz', 'zara', 'abc', [2009, 5678]]
```

In [4]:

```
'''
You can append any element to a list including another list.
'''
aList = [123, 'xyz', 'zara', 'abc']
aList.append( [2009, 'a', 15.9])
print("Updated List : ", aList)
```

```
Updated List :  [123, 'xyz', 'zara', 'abc', [2009, 'a', 15.9]]
```

The `extend()` method adds the contents of a sequence to the end of an existing list. This method modifies the original list. This is a void method.

In [5]:

```
aList = [123, 'xyz', 'zara', 'abc']
aList.extend(['asdf', 5, 8])
print("Updated List : ", aList)
```

```
Updated List :  [123, 'xyz', 'zara', 'abc', 'asdf', 5, 8]
```

The `count()` method counts the number of times an element occurs in a list.

In [6]:

```
aList = [123, 'xyz', 'zara', 'abc', 123]
print("Count for 123 : ", aList.count(123))
print("Count for zara : ", aList.count('zara'))
```

```
Count for 123 :  2
Count for zara :  1
```

The `index()` method returns the index of the first occurence of the argument. The `index()` method takes an optional argument that is the start index at which the search will begin.

In [7]:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz', 567, 123, 'xyz']
print("Index for xyz : ", aList.index( 'xyz'))
print("Index for zara : ", aList.index( 'zara'))
print("Index for xyz starting at index 3: ", aList.index( 'xyz', 3))
print("Index for xyz starting at index -4: ", aList.index( 'xyz', -4))
print("Index for xyz starting at index -3: ", aList.index( 'xyz', -3))
print(aList.index('zzz'))
```

```
Index for xyz :  1
Index for zara :  2
Index for xyz starting at index 3:  4
Index for xyz starting at index -4:  4
Index for xyz starting at index -3:  7
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-33898dba04c8> in <module>
      5 print("Index for xyz starting at index -4: ", aList.index( 'xyz', -4))
      6 print("Index for xyz starting at index -3: ", aList.index( 'xyz', -3))
----> 7 print(aList.index('zzz'))

ValueError: 'zzz' is not in list
```

The `insert()` method inserts the object at the index provided. The first argument is the index at which the second argument is to be inserted. This method modifies the original list.

In [8]:

```
aList = [123, 'xyz', 'zara', 'abc']
a = aList.insert(3,2009)
print("Final List : ", aList)
print(a)
```

```
Final List :  [123, 'xyz', 'zara', 2009, 'abc']
None
```

The `pop()` method removes and returns an item from the list. This method modifies the original list. If no argument is provided, the method returns the last item from the list.

In [9]:

```
aList = [123, 'xyz', 'zara', 'abc']
a = aList.pop()
print(aList)
print(a)
```

```
[123, 'xyz', 'zara']
abc
```

The `pop()` method removes and returns an item from the list. This method modifies the original list. If an argument is provided (it should be a valid index), the method returns the element at that index.

In [10]:

```
aList = [123, 'xyz', 'zara', 'abc']
b = aList.pop(-2)
print(aList)
print(b)
```

```
[123, 'xyz', 'abc']
zara
```

The `remove()` method removes an item from the list. It takes *exactly* one argument. It accepts the item to be removed as an argument and removes the first matching value. It returns nothing. This method modifies the original list. This method returns an error if the element does not exist in the list.

In [11]:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']

# remove 1st occurence of 'xyz'
aList.remove('xyz')
print(aList)
```

```
[123, 'zara', 'abc', 'xyz']
```

In [12]:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
a = aList.remove('xyzz')
print("List : ", aList)
print(a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-aa1b403bc600> in <module>
      1 aList = [123, 'xyz', 'zara', 'abc', 'xyz']
----> 2 a = aList.remove('xyzz')
```

```
print("List : ", aList)
print(a)
```

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

The following provides the differences between `del`, `remove()` and `pop()`

`del` is an operator that removes the item from a list at a specified index. It can remove multiple items as provided by the slicing feature.It returns nothing.

`pop()` removes the item at the index specified by the optional argument. If no argument is provided, it removes the item at the last index. It returns the value that is removed.

`remove()` removes the first matching value. It accepts a value as an argument not an index. It returns nothing.

Note that all three methods modify the original list.

In [13]:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']

# delete whole list
del aList[:]
print(aList)
```

```
[]
```

In [14]:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']

# delete a slice of a list
del aList[2:]
print(aList)
```

```
[123, 'xyz']
```

The `reverse()` method reverses the objects of a list in place. That means, nothing is returned and the list is changed.

In [15]:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
a = aList.reverse()
print("List : ", aList)
print(a)
```

```
List :  ['xyz', 'abc', 'zara', 'xyz', 123]
None
```

The `sort()` method sorts the objects of a list in place. Nothing is returned and the list is changed. Note that all elements of the list have to be of the same type. Having a mix of elements will cause an error.

In [16]:

```
aList = ['xyz', 'zara', 'abc', 'xyz']

a = aList.sort()
print("List : ", aList)
print(a)
```

```
List :  ['abc', 'xyz', 'xyz', 'zara']
None
```

In [17]:

```
'''
```

```
'''
Some things you can do with a list.  You can create an empty list as shown below.  You can then us
e the append()
and insert() methods or the concatenation operator to add elements to the list.
'''
my_lst = []
my_lst = my_lst + [1,2,3]
print(my_lst)
my_lst.append(5)
print(my_lst)
my_lst.extend([8,7,2])
print(my_lst)
```

```
[1, 2, 3]
[1, 2, 3, 5]
[1, 2, 3, 5, 8, 7, 2]
```

In [18]:

```
'''
You can initialize a list to contain all zeros (for example) by using the multiplication operator
as shown below
'''
my_lst = [0]*5
print(my_lst)
```

```
[0, 0, 0, 0, 0]
```