Lists Reference Guide

Common List Functions and Methods

Length

The function `len(L)` will return the length of List `L`.

Examples:

```
>>> L = [1, 2, 42, 10]
>>> len(L)
4
>>> len([])
0
```

Slicing

Slicing allows you to "slice" out part of a List, allowing you to focus on only this specific portion that you care about. **Note that constructing a slice of a List does not change the original List!** You can get a slice of some List with the expression:

`myList[`*start* : *stop* : *step*`]`

start represents the first index to be included in the slice.
The default value is 0, which will begin the slice at the beginning of the List.

stop represents the last index in the slice, and is not included in the slice.
The default value is the length of the List, which will end the slice with the final element of the List.

step represents the step size from one element to the next in the slice.
The default value is 1, which will not skip any elements when moving from *start* to *stop*.
This value does not need to be included if the default of 1 is desired.

Examples:

```
>>> letters = ['A', 'B', 'C', 'D', 'E']
>>> letters[2:4]
['C', 'D']
>>> letters[3:]
['D', 'E']
>>> letters[:1]
['A']
>>> letters[1:4:2]
['B', 'D']
>>> letters[::2]
['A', 'C', 'E']
>>> letters
['A', 'B', 'C', 'D', 'E']
```

Looping over a List

There are two ways to loop over the contents of a List.

We can use the indexes of the List:

```
for index in range(len(myList)):
    print(myList[index])
```

Or we can use the values of the elements themselves:

```
for elem in myList:
    print(elem)
```

In both of these cases, every element of the List `myList` will be printed on a separate line.

Experiment with this yourself to master choosing which method is better for a given problem!

Examples:

```
>>> numbers = [32, 4, 7, 0, -42]
>>> for i in range(0, len(numbers)):
...     print(numbers[i], end = " ")
...
32 4 7 0 -42

>>> letters = ['X', 'K', 'C', 'D']
>>> for letter in letters:
...     print(letter)
...
X
K
C
D

>>> matrix = [['a', 'b', 'c'], [1, 2, 3], ['x', 'y', 'z']]
>>> for L in matrix:
...     for index in range(0, len(L)):
...         print(L[index], end = " ")
...     print()
...
a b c
1 2 3
x y z
```

Concatenation

In Python, the + operator is used to concatenate two Lists, **constructing and returning a new List, while leaving the originals the same as before.**

Given two Lists, `L1` and `L2`, the expression `L1 + L2` represents a List with all elements of `L1` followed by all elements of `L2`.

Examples:

```
>>> newList = [2, 4, 6] + [1, 2]
>>> newList
[2, 4, 6, 1, 2]
>>> [1,2] + newList
[1, 2, 2, 4, 6, 1, 2]
```

extend

The method `myList.extend(L)` adds all elements of `L` to the end of `myList`, and returns `None`. This is very similar to `myList + L`, however the original List is modified when using `extend`.

`L1.extend(L2)` is equivalent to `L1 = L1 + L2`.

Examples:

```
>>> myList = [2, 4, 6]
>>> otherList = [1, 2]
>>> newList = myList.extend(otherList)
>>> print(newList)
None
>>> myList
[2, 4, 6, 1, 2]
```

append

The method `myList.append(x)` will add the element `x` to the end of `myList`, and returns `None`.
Examples:

```
>>> L = [17, 10, 4]
>>> newList = L.append(42)
>>> print(newList)
None
>>> L
[17, 10, 4, 42]
>>> L.append([1,2,3])
>>> L
[17, 10, 4, 42, [1, 2, 3]]
```

insert

The method `myList.insert(i,x)` will insert element `x` at index `i` of `myList`, and returns `None`.

Examples:

```
>>> L = ['W', 'Y', 'Z']
>>> newList = L.insert(1, 'X')
>>> print(newList)
None
>>> L
['W', 'X', 'Y', 'Z']
>>> L.insert(0, 'V')
>>> L
['V', 'W', 'X', 'Y', 'Z']
```

pop

The method `myList.pop(i)` will remove the element in `myList` at index `i`.
The the value of the element that was removed is returned by the method.

If no parameter is provided for `i`, the last element of the List will be removed and returned.

Examples:

```
>>> L = [42, 'hello', 'world']
>>> element = L.pop(1)
>>> element
'hello'
>>> L
[42, 'world']
>>> L.pop()
'world'
>>> L
[42]
```

remove

The method `myList.remove(x)` will remove the first occurrence of `x` in `myList`.
The method returns `None`, and will cause an error if no element in `myList` is equal to `x`

Examples:

```
>>> L = [1,2,3,4,5,2,42,0]
>>> newList = L.remove(42)
>>> print(newList)
None
>>> L
[1,2,3,4,5,2,0]
>>> L.remove(2)
>>> L
[1,3,4,5,2,0]
```

index

The method `myList.index(x)` will return the index of the first occurrence of `x` in `myList`.
This method will cause an error If no element in `myList` is equal to `x`.

Examples:

```
>>> L = [34, 8, 'hello', 0, 23, 0, 23, 0, 23]
>>> i = L.index(8)
>>> i
1
>>> L.index('hello')
2
>>> L.index(23)
4
>>> L
[34, 8, 'hello', 0, 23, 0, 23, 0, 23]
```