List Functions

Python List Methods

Python has a set of built-in methods that you can call on list objects.

Method	Description
`append()`	Adds an item to the end of the list
`insert()`	Inserts an item at a given position
`extend()`	Extends the list by appending all the items from the iterable
`remove()`	Removes first instance of the specified item
`pop()`	Removes the item at the given position in the list
`clear()`	Removes all items from the list
`copy()`	Returns a shallow copy of the list
`count()`	Returns the count of specified item in the list
`index()`	Returns the index of first instance of the specified item
`reverse()`	Reverses the items of the list in place
`sort()`	Sorts the items of the list in place

Python List Methods

Usage

The `append()` method adds a single `item` to the end of the list. This method does not return anything; it modifies the list in place.

Syntax

```
list.append(item)
```

Parameter	Condition	Description
`item`	Required	An item you want to append to the list

Python list append() method parameters

Examples

```
# Append 'yellow'
L = ['red', 'green', 'blue']
L.append('yellow')
print(L)
# Prints ['red', 'green', 'blue', 'yellow']
# Append list to a list
L = ['red', 'green', 'blue']
L.append([1,2,3])
print(L)
# Prints ['red', 'green', 'blue', [1, 2, 3]]
```

```
# Append tuple to a list
L = ['red', 'green', 'blue']
L.append((1,2,3))
print(L)
# Prints ['red', 'green', 'blue', (1, 2, 3)]
```

append() vs extend()

append() method treats its argument as a single object.

```
L = ['red', 'green']
L.append('blue')
print(L)
# Prints ['red', 'green', 'blue']
```

Use extend() method, if you want to add every item of an iterable to a list.

```
L = ['red', 'green']
L.extend('blue')
print(L)
# Prints ['red', 'green', 'b', 'l', 'u', 'e']
```

Usage

Use `clear()` method to remove all items from the list. This method does not return anything; it modifies the list in place.

Syntax

```
list.clear()
```

Basic Example

```
L = ['red', 'green', 'blue']
L.clear()
print(L)
# Prints []
```

ⓘ Please note that `clear()` is not same as assigning an empty list `L = []`.

Usage

The `copy()` method returns the Shallow copy of the specified list.

Syntax

```
list.copy()
```

Basic Example

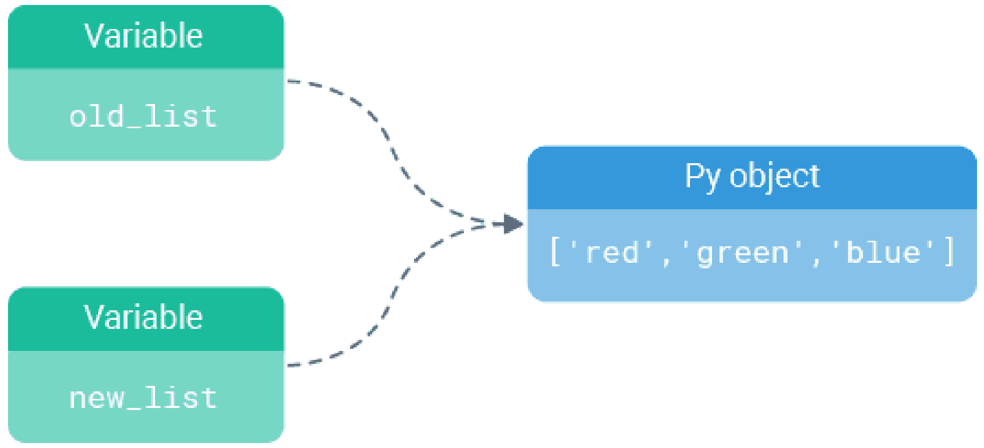
```python
# Create a copy of list 'L'
L = ['red', 'green', 'blue']
X = L.copy()
print(X)
# Prints ['red', 'green', 'blue']
```

copy() vs Assignment statement

Assignment statement does not copy objects. For example,

```
old_List = ['red', 'green', 'blue']
new_List = old_List
new_List[0] = 'xx'
print(old_List)
# Prints ['xx', 'green', 'blue']
print(new_List)
# Prints ['xx', 'green', 'blue']
```

When you execute `new_List = old_List`, you don't actually have two lists. The assignment just makes the two variables point to the one list in memory.



So, when you change new_List, old_List is also modified. If you want to change one copy without changing the other, use `copy()` method.

```
old_List = ['red', 'green', 'blue']
new_List = old_List.copy()
new_List[0] = 'xx'
print(old_List)
# Prints ['red', 'green', 'blue']
print(new_List)
# Prints ['xx', 'green', 'blue']
```

Equivalent Method

Assigning a slice of the entire list to a variable is equivalent to `copy()` method.

```
L = ['red', 'green', 'blue']
X = L[:]
print(X)
# Prints ['red', 'green', 'blue']
```

Usage

Use `count()` method to find the number of times the specified `item` appears in the list.

Syntax

```
list.count(item)
```

Parameter	Condition	Description
`item`	Required	Any item (of type string, list, set, etc.) you want to search for.

Python list count() method parameters

Examples

```
# Count number of occurrences of 'red'
L = ['red', 'green', 'blue']
print(L.count('red'))
# Prints 1
```

```
# Count number of occurrences of number '9'
L = [1, 9, 7, 3, 9, 1, 9, 2]
print(L.count(9))
# Prints 3
```

Usage

The `extend()` method extends the list by appending all the items from the `iterable` to the end of the list. This method does not return anything; it modifies the list in place.

Syntax

```
list.extend(iterable)
```

Parameter	Condition	Description
`iterable`	Required	Any iterable (string, list, set, tuple, etc.)

Python list extend() method parameters

Examples

```
# Add multiple items to a list
L = ['red', 'green', 'blue']
L.extend([1,2,3])
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]
```

```
# Add tuple items to a list
L = ['red', 'green', 'blue']
L.extend((1,2,3))
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]
```

```
# Add set items to a list
L = ['red', 'green', 'blue']
L.extend({1,2,3})
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]
```

extend() vs append()

`extend()` method treats its argument as an iterable object.

For example, when you pass a string (iterable object) as an argument, the method adds every character to a list instead of the string.

```
L = ['red', 'green', 'blue']
L.extend('red')
print(L)
# Prints ['red', 'green', 'blue', 'r', 'e', 'd']
```

Use append() method instead:

```
L = ['red', 'green', 'blue']
L.append('red')
print(L)
# Prints ['red', 'green', 'blue', 'red']
```

Equivalent Methods

Specifying a zero-length slice at the end is also equivalent to `extend()` method.

```
L = ['red', 'green', 'blue']
L[len(L):] = [1,2,3]
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]
```

Using concatenation operator `+` or the augmented assignment operator `+=` on a list is equivalent to using `extend()`.

```
# Concatenation operator
L = ['red', 'green', 'blue']
L = L + [1,2,3]
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]

# Augmented assignment operator
L = ['red', 'green', 'blue']
L += [1,2,3]
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]
```

Usage

The `index()` method searches for the first occurrence of the given `item` and returns its index. If specified item is not found, it raises '**ValueError**' exception.

The optional arguments `start` and `end` limit the search to a particular subsequence of the list.

Syntax

```
list.index(item,start,end)
```

Parameter	Condition	Description
`item`	Required	Any item (of type string, list, set, etc.) you want to search for
`start`	Optional	An index specifying where to start the search. Default is 0.
`end`	Optional	An index specifying where to stop the search. Default is the end of the list.

Python list index() method parameters

Basic Example

```
# Find the index of 'green' in a list
L = ['red', 'green', 'blue', 'yellow']
print(L.index('green'))
# Prints 1
```

index() on Duplicate Items

If the list has many instances of the specified `item`, the `index()` method returns the index of first instance only.

```
# Find first occurrence of character 'c'
L = ['a','b','c','d','e','f','a','b','c','d','e','f']
print(L.index('c'))
# Prints 2
```

index() on Item that Doesn't Exist

`index()` method raises a '**ValueError**' if specified `item` is not found in the list.

```
L = ['a','b','c','d','e','f','a','b','c','d','e','f']
print(L.index('x'))
# Triggers ValueError: 'x' is not in list

# also within search bound
L = ['a','b','c','d','e','f','a','b','c','d','e','f']
print(L.index('c',4,7))
# Triggers ValueError: 'c' is not in list
```

Limit index() Search to Subsequence

If you want to search the list from the middle, specify the `start` parameter.

```
# Find 'c' starting a position 5
L = ['a','b','c','d','e','f','a','b','c','d','e','f']
print(L.index('c',5))
# Prints 8
```

The returned index is computed relative to the beginning of the full sequence rather than the `start` argument.

You can also specify where to stop the search with `end` parameter.

```
# Find 'c' in between 5 & 10
L = ['a','b','c','d','e','f','a','b','c','d','e','f']
print(L.index('c',5,10))
# Prints 8
```

Usage

Use `insert()` method to insert a single `item` at a specified `index` in a list. Note that other items are shifted to the right.

This method does not return anything; it modifies the list in place.

Syntax

```
list.insert(index,item)
```

Parameter	Condition	Description
`index`	Required	Index of an item before which to insert
`item`	Required	An item you want to insert

Python list insert() method parameters

Examples

```
# Insert 'yellow' at 2nd position
L = ['red', 'green', 'blue']
L.insert(1,'yellow')
print(L)
# Prints ['red', 'yellow', 'green', 'blue']
```

You can also use negative indexing with `insert()` method.

```
# Insert 'yellow' at 2nd position
L = ['red', 'green', 'blue']
L.insert(-2,'yellow')
print(L)
# Prints ['red', 'yellow', 'green', 'blue']
```

Index greater than list length

When you specify an `index` greater than list length, you do not get any exception. Instead, the item is inserted at the end of the list.

```
L = ['red', 'green', 'blue']
L.insert(10,'yellow')
print(L)
# Prints ['red', 'green', 'blue', 'yellow']
```

Usage

The `pop()` method removes a single list item at specified `index` and returns it. If no index is specified, `pop()` method removes and returns the last item in the list.

Syntax

```
list.pop(index)
```

Parameter	Condition	Description
index	Optional	An index of item you want to remove. Default value is -1

Python list pop() method parameters

Return Value

The `pop()` method returns the value of removed item.

Examples

```
# Remove 2nd list item
L = ['red', 'green', 'blue']
L.pop(1)
print(L)
# Prints ['red', 'blue']
```

Usage

Use `remove()` method to remove a single `item` from a list.

The method searches for the first instance of the given item and removes it. If specified item is not found, it raises '**ValueError**' exception.

Syntax

```
list.remove(item)
```

Parameter	Condition	Description
`item`	Required	Any item you want to remove

Python list remove() method parameters

Removing Item that Doesn't Exist

`remove()` method raises an ValueError exception, if specified item doesn't exist in a list.

```
L = ['red', 'green', 'blue']
L.remove('yellow')
# Triggers ValueError: list.remove(x): x not in list
```

Remove Single Item

```
# Remove 'green'
L = ['red', 'green', 'blue']
L.remove('green')
print(L)
# Prints ['red', 'blue']
```

```
# Remove item from the nested list
L = ['red', 'green', [1, 2, 3]]
L.remove([1, 2, 3])
print(L)
# Prints ['red', 'green']
```

The `remove()` method removes item based on specified value and not by index. If you want to delete list items based on the index, use pop() method or del keyword.

Usage

The `reverse()` method reverses the order of list. This method does not return anything; it modifies the list in place.

Syntax

```
list.reverse()
```

Examples

```
L = ['red', 'green', 'blue']
L.reverse()
print(L)
# Prints ['blue', 'green', 'red']
```

```
L = [1, 2, 3, 4, 5]
L.reverse()
print(L)
# Prints [5, 4, 3, 2, 1]
```

Access List Items in Reversed Order

If you don't want to modify the list but access items in reverse order, you can use reversed() built-in function. It returns the reversed iterator object, with which you can loop through the list in reverse order.

```
L = ['red', 'green', 'blue']
for x in reversed(L):
  print(x)
# blue
# green
# red
```

Usage

Use `sort()` method to sort the items of the list.

You can optionally specify parameters for sort customization like sorting order and sorting criteria.

Syntax

```
list.sort(key,reverse)
```

Parameter	Condition	Description
key	Optional	A function to specify the sorting criteria. Default value is None.
reverse	Optional	Settting it to True sorts the list in reverse order. Default value is False.

Python list sort() method parameters

Sort List

`sort()` method sorts the list of strings alphabetically and the list of numbers numerically.

```
# Sort the list of strings
L = ['red', 'green', 'blue', 'orange']
L.sort()
print(L)
# Prints ['blue', 'green', 'orange', 'red']
```

```
# Sort the list of numbers
L = [42, 99, 1, 12]
L.sort()
print(L)
# Prints [1, 12, 42, 99]
```

However, you cannot sort lists that have both numbers and strings in them, since Python doesn't know how to compare these values.

```
L = ['red', 'blue', 1, 12, 'orange',42, 'green', 99]
L.sort()
# Triggers TypeError: '<' not supported between instances of 'int' and 'str'
```