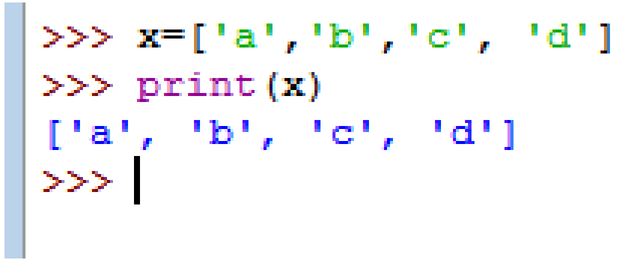
Programming Fundamentals and Python

Data Type – List

Data types – List

Multiple items can be stored on a single variable. For creation of list, we use square brackets [ ], and every item must be in single ‘ ’ or double “ ” quotation mark.
There are <u>four data types in Python</u>, all with different qualities and usage, these are:

i. **List** is a collection which is ordered and changeable. Allows duplicate members.
ii. **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
iii. **Set** is a collection which is unordered, unchangeable(tems are unchangeable, but you can remove and/or add items whenever you like), and unindexed. No duplicate members.
iv. **Dictionary** is a collection which is ordered (As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.) and changeable. No duplicate members.



```
>>> x=['a','b','c', 'd']
>>> print(x)
['a', 'b', 'c', 'd']
>>> 
```

List properties

Python lists have following basic properties:

- <u>Ordered collections of arbitrary objects</u>

  From a functional view, lists are just places to collect other objects. Lists also maintain a **left-to-right positional ordering** among the items contained in them (i.e., they are sequences).
- <u>Accessed by offset</u>

  A component object of list can be accessed by its position.
- <u>Variable-length, heterogeneous, and arbitrarily nestable</u>

  Unlike strings, lists can grow and shrink in place (their lengths can vary), and they can contain any sort of object, not just one-character strings (they're heterogeneous). Because lists can contain other complex objects, they also support arbitrary nesting.

- <u>Of the category "mutable sequence"</u>

  Lists are **mutable** (i.e., can be changed in place) and can respond to all the sequence operations used with strings, such as indexing, slicing, and concatenation. In fact, sequence operations work the same on lists as they do on strings; the only difference is that sequence operations such as concatenation and slicing return new lists instead of new strings when applied to lists. Because lists are mutable, however, they also support other operations that strings don't, such as deletion and index assignment operations, which change the lists in place.

List item and order

List items are ordered, changeable, and allow duplicate values. List items are indexed, the first item has index [0], the second item has index [1] etc.

When we say that lists are **ordered**, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list. The list is **changeable**, meaning that we can change, add, and remove items in a list after it has been created. Since lists are indexed, lists can have items with the same value.

The notable point is that, there are some **list methods** that will change the order, but in general: the order of the items will not change.

Indexing

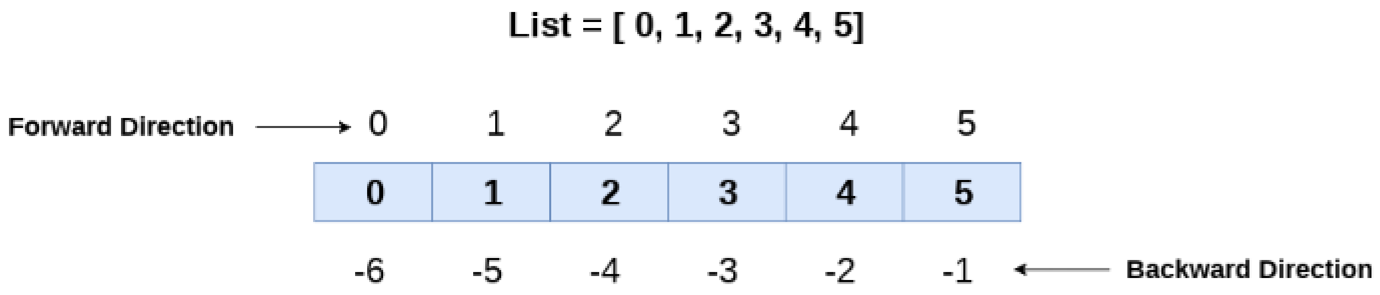
The indexing is processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [ ]. The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

List = [0, 1, 2, 3, 4, 5]

0	**1**	**2**	**3**	**4**	**5**

List[0] = 0	List[0:] = [0,1,2,3,4,5]
List[1] = 1	List[:] = [0,1,2,3,4,5]
List[2] = 2	List[2:4] = [2, 3]
List[3] = 3	List[1:3] = [1, 2]
List[4] = 4	List[:4] = [0, 1, 2, 3]
List[5] = 5	

Python provides the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (rightmost) of the list has the index -1; its adjacent left element is present at the index -2 and so on until the left-most elements are encountered.



Example

‘L’ is the variable which contain a group of items.

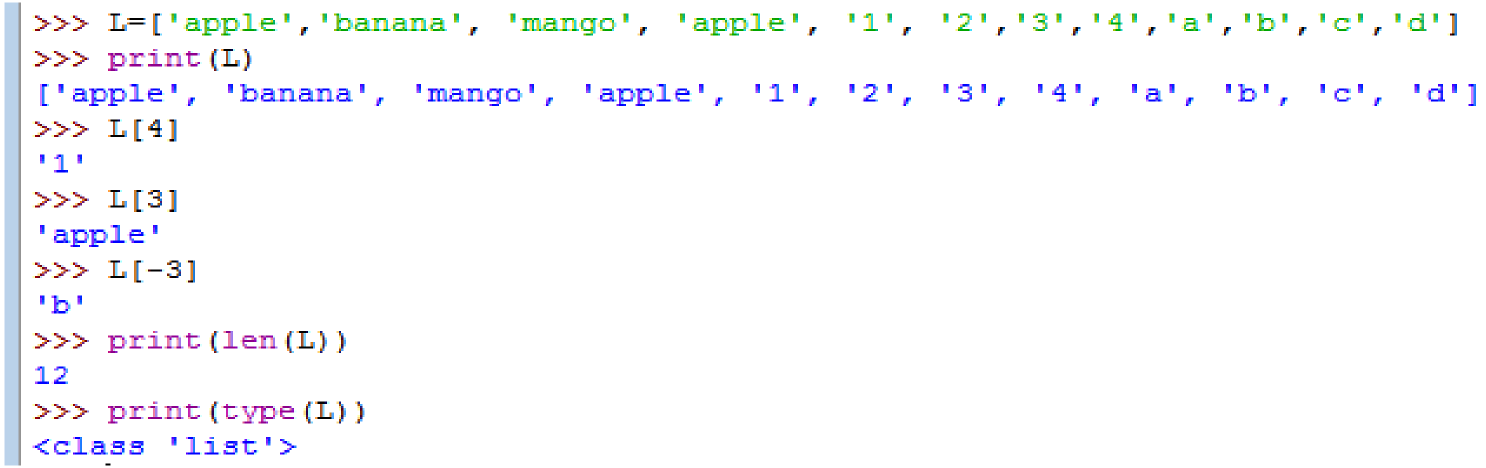
List can contain different type of data, as you can see.

L[2] – is an offset which starts from zero, and designated offset item is called.

Negative offset counts from the right i.e. reverse, but not from zero.

Len( ) – determines the length of list.

From Python's perspective, lists are defined as objects with the data type 'list'



```
>>> L=['apple','banana', 'mango', 'apple', '1', '2','3','4','a','b','c','d']
>>> print(L)
['apple', 'banana', 'mango', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> L[4]
'1'
>>> L[3]
'apple'
>>> L[-3]
'b'
>>> print(len(L))
12
>>> print(type(L))
<class 'list'>
```

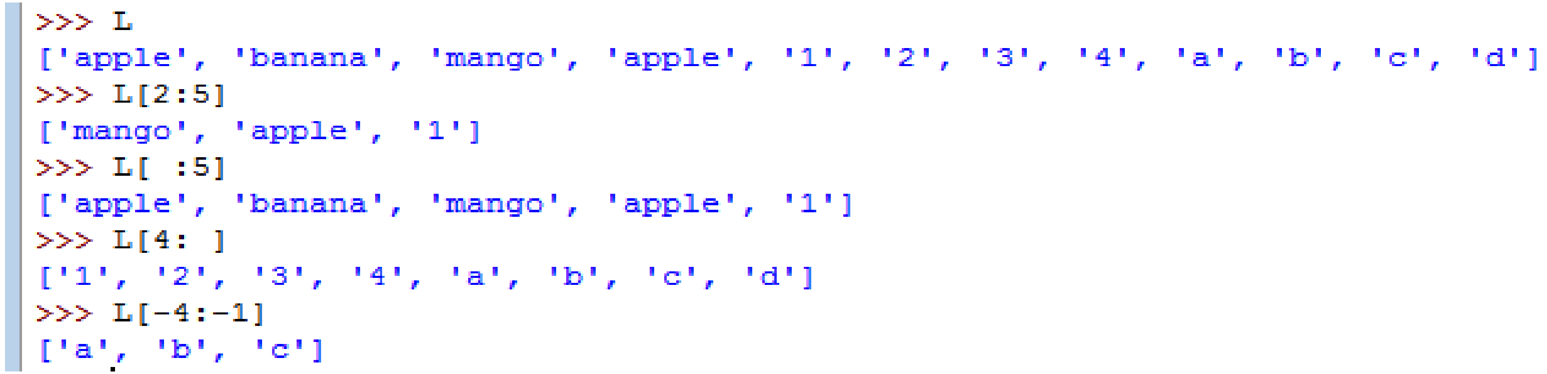
List Operations

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings. **Consider a Lists l1 = [1, 2, 3, 4], and l2 = [5, 6, 7, 8] to perform operation.**

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenatio n	It concatenates the list mentioned on either side of the operator.	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) Output 1 2 3 4

Access Items

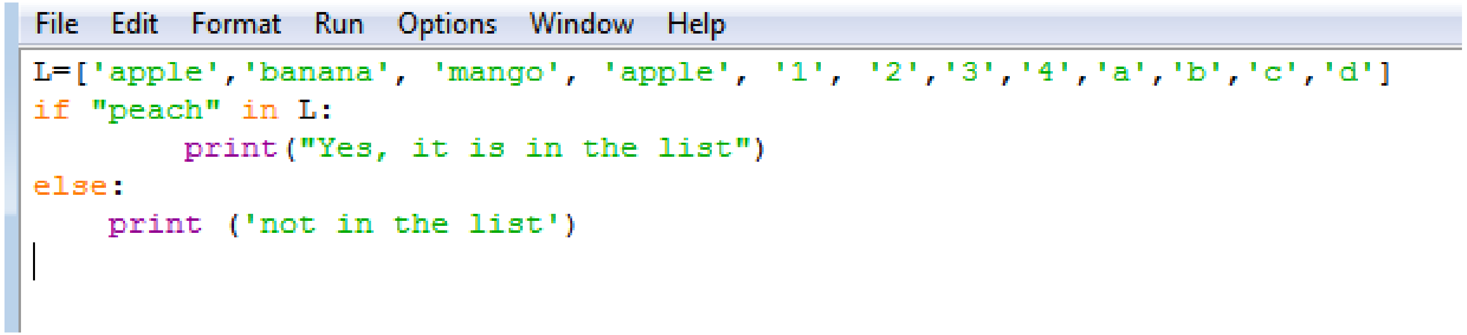
List items are indexed and you can access them by referring to the index number. You can specify a range of indexes by specifying where to start and where to end the range. When specifying a range, the return value will be a new list with the specified items.
The search will start at index 2 (included) and end at index 5 (not included).
By leaving out the start value, the range will start at the first item.
By leaving out the end value, the range will go on to the end of the list.
Specify negative indexes if you want to start the search from the end of the list.



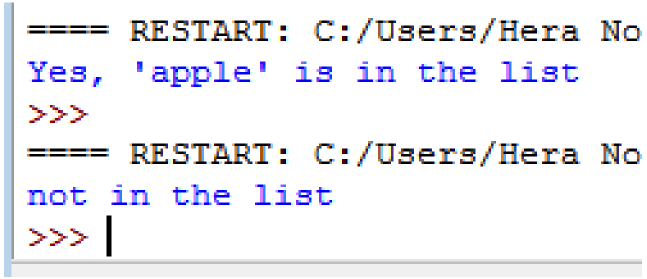
```
>>> L
['apple', 'banana', 'mango', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> L[2:5]
['mango', 'apple', '1']
>>> L[ :5]
['apple', 'banana', 'mango', 'apple', '1']
>>> L[4: ]
['1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> L[-4:-1]
['a', 'b', 'c']
```

Check Item presence in List

To determine if a specified item is present in a list use the ‘in’ keyword.



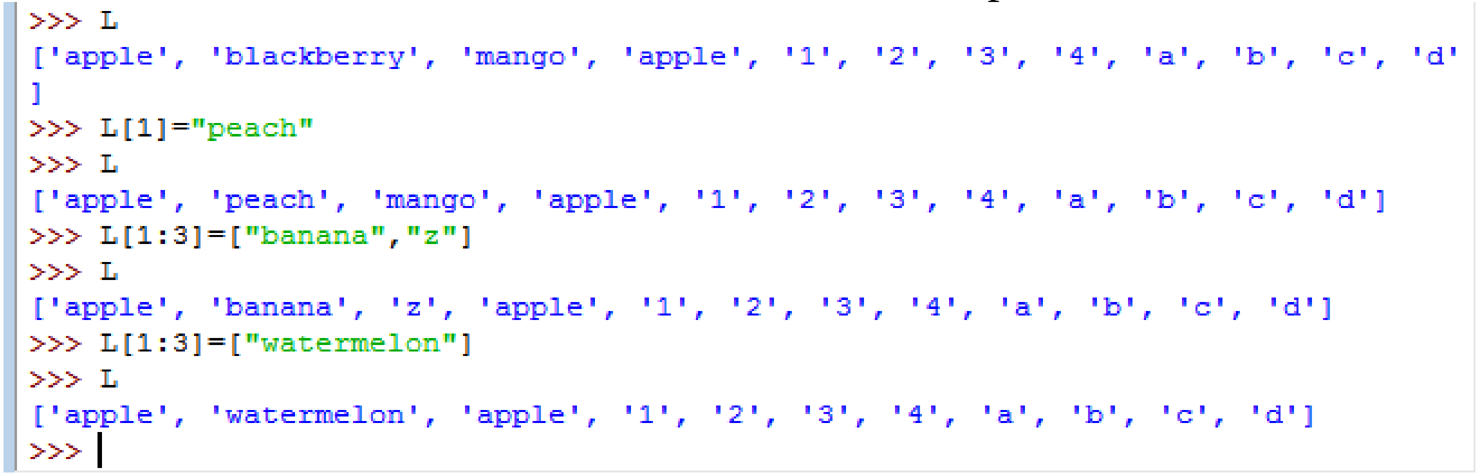
```
L=['apple','banana', 'mango', 'apple', '1', '2','3','4','a','b','c','d']
if "peach" in L:
        print("Yes, it is in the list")
else:
    print ('not in the list')
```



```
==== RESTART: C:/Users/Hera No
Yes, 'apple' is in the list
>>>
==== RESTART: C:/Users/Hera No
not in the list
>>>
```

Change Item value

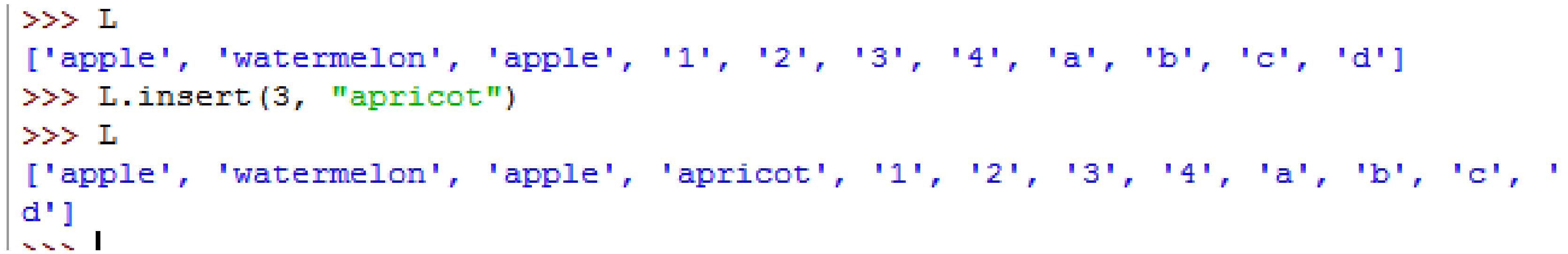
To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values. If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly. The length of the list will change when the number of items inserted does not match the number of items replaced.



```
>>> L
['apple', 'blackberry', 'mango', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd'
]
>>> L[1]="peach"
>>> L
['apple', 'peach', 'mango', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> L[1:3]=["banana","z"]
>>> L
['apple', 'banana', 'z', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> L[1:3]=["watermelon"]
>>> L
['apple', 'watermelon', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> 
```

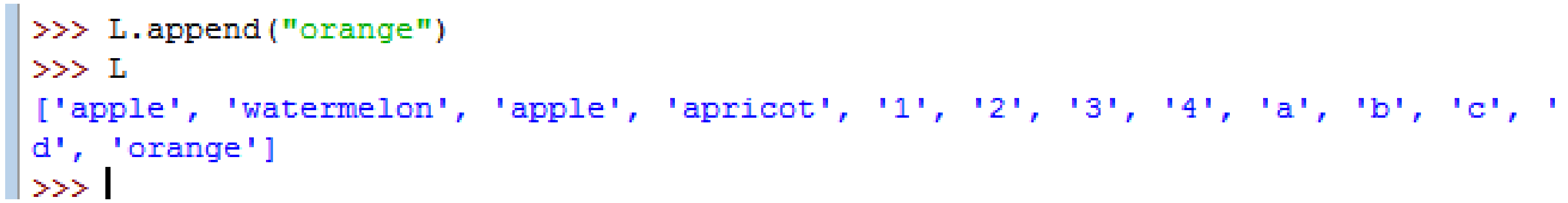
Insert or Add Items

The insert( ) method inserts an item at the specified index.



```
>>> L
['apple', 'watermelon', 'apple', '1', '2', '3', '4', 'a', 'b', 'c', 'd']
>>> L.insert(3, "apricot")
>>> L
['apple', 'watermelon', 'apple', 'apricot', '1', '2', '3', '4', 'a', 'b', 'c', '
d']
>>>
```

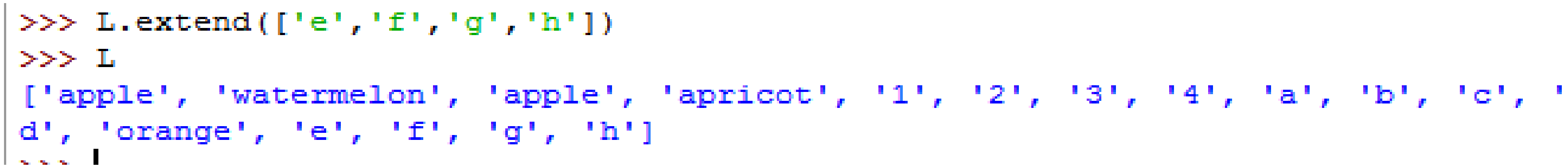
To add an item to the end of the list, use the append( ) method.



```
>>> L.append("orange")
>>> L
['apple', 'watermelon', 'apple', 'apricot', '1', '2', '3', '4', 'a', 'b', 'c', '
d', 'orange']
>>>
```

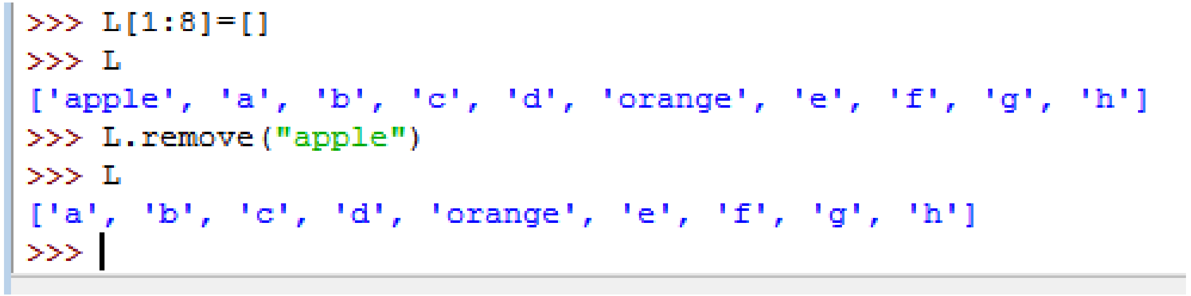
Extend or Delete

To append elements from another list to the current list, use the extend( ) method. This will extend the list at the end.



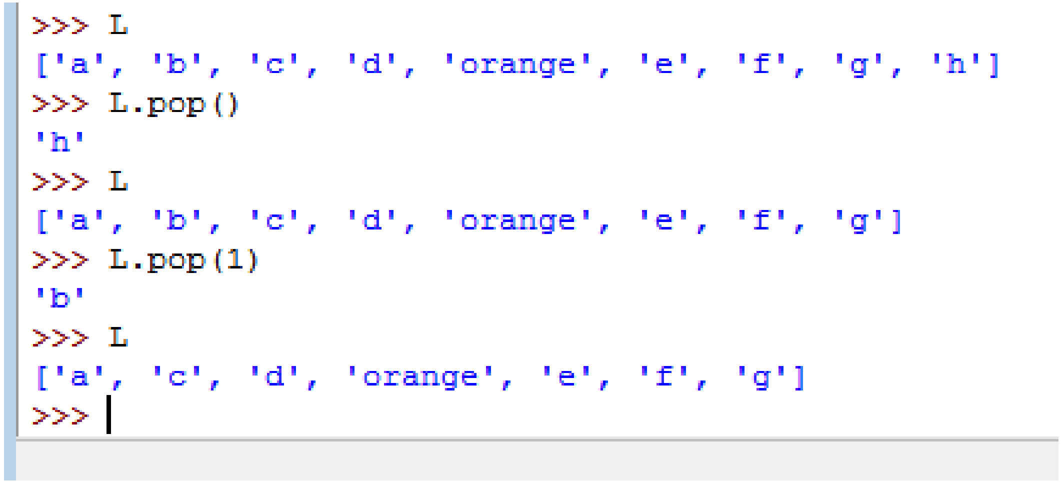
```
>>> L.extend(['e','f','g','h'])
>>> L
['apple', 'watermelon', 'apple', 'apricot', '1', '2', '3', '4', 'a', 'b', 'c', '
d', 'orange', 'e', 'f', 'g', 'h']
>>>
```

To delete the items, we use offset to select them for deletion. Another command is for any specified item is remove( ).



```
>>> L[1:8]=[]
>>> L
['apple', 'a', 'b', 'c', 'd', 'orange', 'e', 'f', 'g', 'h']
>>> L.remove("apple")
>>> L
['a', 'b', 'c', 'd', 'orange', 'e', 'f', 'g', 'h']
>>>
```

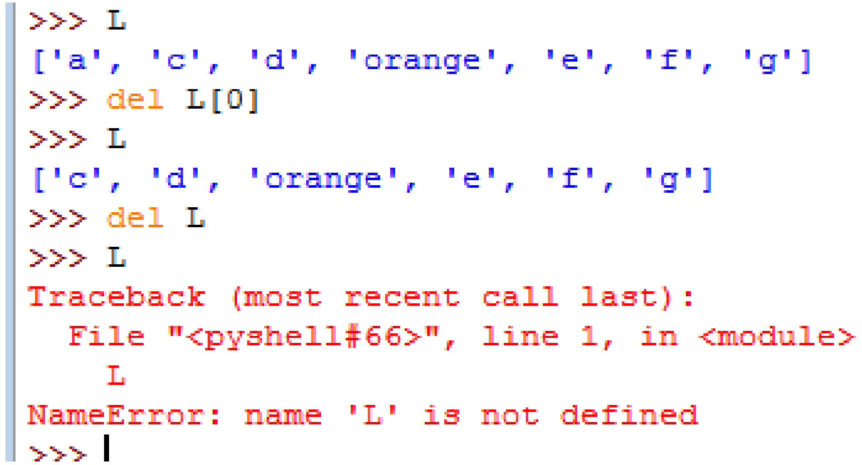
The pop() method removes the specified index. If you do not specify the index, the pop() method removes the last item.



```
>>> L
['a', 'b', 'c', 'd', 'orange', 'e', 'f', 'g', 'h']
>>> L.pop()
'h'
>>> L
['a', 'b', 'c', 'd', 'orange', 'e', 'f', 'g']
>>> L.pop(1)
'b'
>>> L
['a', 'c', 'd', 'orange', 'e', 'f', 'g']
>>> 
```

To delete the items, we use offset to select them for deletion. Another command is for any specified item is remove().

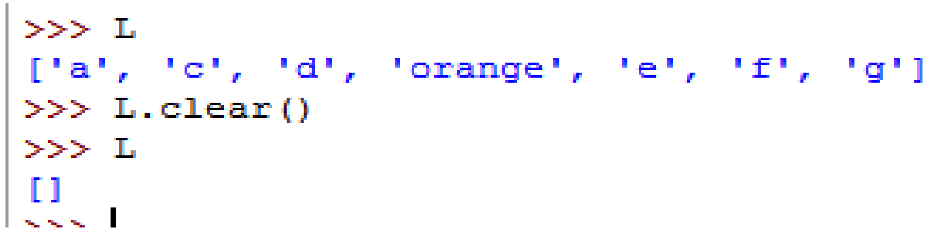
The del keyword also removes the specified index. The del keyword can also delete the list completely.



```
>>> L
['a', 'c', 'd', 'orange', 'e', 'f', 'g']
>>> del L[0]
>>> L
['c', 'd', 'orange', 'e', 'f', 'g']
>>> del L
>>> L
Traceback (most recent call last):
  File "<pyshell#66>", line 1, in <module>
    L
NameError: name 'L' is not defined
>>> 
```

Clear list

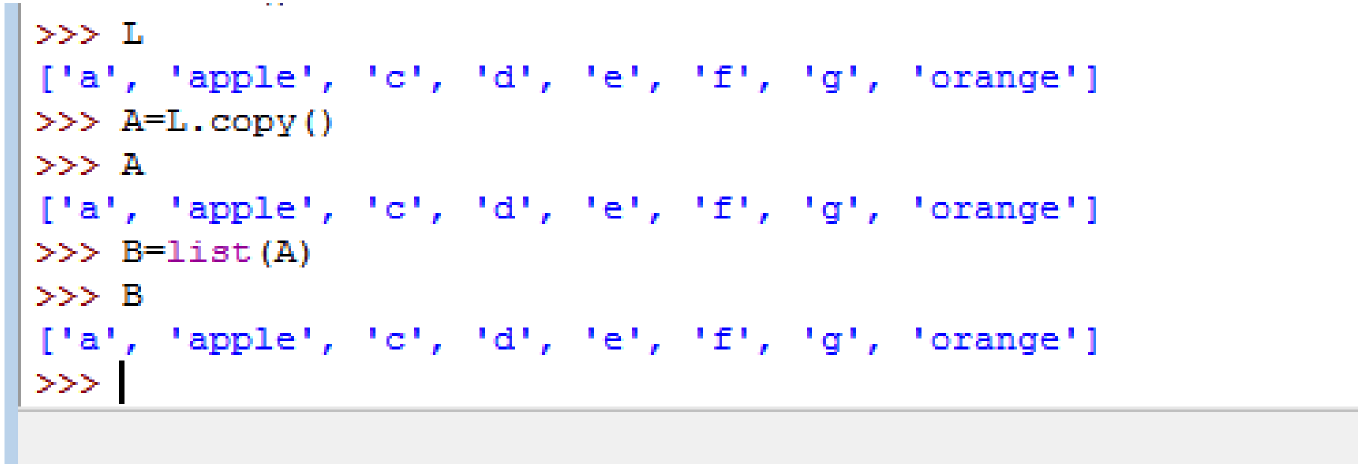
The clear( ) method empties the list. The list still remains, but it has no content.



```
>>> L
['a', 'c', 'd', 'orange', 'e', 'f', 'g']
>>> L.clear()
>>> L
[]
>>>
```

Copy List

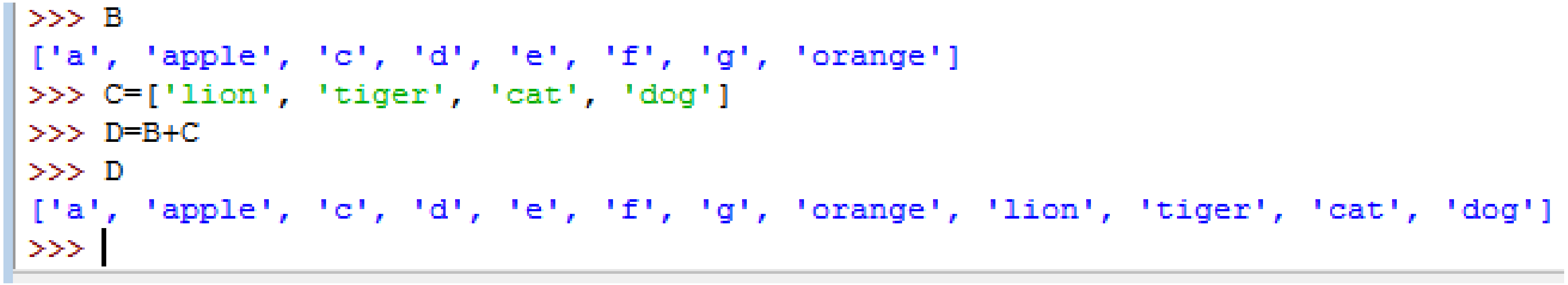
You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2. There are ways to make a copy, one way is to use the built-in List method copy( ). Another way to make a copy is to use the built-in method list( ).



```
>>> L
['a', 'apple', 'c', 'd', 'e', 'f', 'g', 'orange']
>>> A=L.copy()
>>> A
['a', 'apple', 'c', 'd', 'e', 'f', 'g', 'orange']
>>> B=list(A)
>>> B
['a', 'apple', 'c', 'd', 'e', 'f', 'g', 'orange']
>>>
```

Join List

There are several ways to join, or concatenate, two or more lists in Python. One of the easiest ways are by using the + operator. you can use the extend() method, which purpose is to add elements from one list to another list.



```
>>> B
['a', 'apple', 'c', 'd', 'e', 'f', 'g', 'orange']
>>> C=['lion', 'tiger', 'cat', 'dog']
>>> D=B+C
>>> D
['a', 'apple', 'c', 'd', 'e', 'f', 'g', 'orange', 'lion', 'tiger', 'cat', 'dog']
>>> 
```

List functions

Python provides the following built-in functions, which can be used with the lists.

Function	**Description**	**Example**
cmp(list1, list2)	It compares the elements of both the lists.	This method is not used in the Python 3 and the above versions.
len(list)	It is used to calculate the length of the list.	L1 = [1,2,3,4,5,6,7,8] print(len(L1)) 8
max(list)	It returns the maximum element of the list.	L1 = [12,34,26,48,72] print(max(L1)) 72

min(list)	It returns the minimum element of the list.	L1 = [12,34,26,48,72] print(min(L1)) 12
list(seq)	It converts any sequence to the list.	str = "Johnson" s = list(str) print(type(s)) <class list>

List method call - Summary

Some built-in python methods, we have gone through are summarized here.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list

index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list