LIST

1)What is python list?

Unlike C++ and Java , Python programming language doesn’t have arrays. To hold a sequence of values, it provides a list class.

Properties:

**1)You can also reassign a slice of a list in Python**.

color=[‘caramel’ , ’gold’ , ’silver’ , ’occur’]

1. >>> colors[2:]=['bronze','silver'] // Replaces silver and gold as bronze and silver
2. >>> colors

***[‘caramel’, ‘gold’, ‘bronze’, ‘silver’]***

If we had instead put two values to a single one in the left, see what would’ve happened.

>>> colors=['caramel','gold','silver','occur']

1. >>> colors[2:3]=['bronze','silver']
2. >>> colors // Does not replace but it adds bronze before silver

***[‘caramel’, ‘gold’, ‘bronze’, ‘silver’, ‘occur’]***

colors[2:3] reassigns the element at index 2, which is the third element.

2:2 works too.

1. >>> colors[2:2]=['occur']
2. >>> colors

[‘caramel’, ‘gold’, ‘occur’, ‘bronze’, ‘silver’]

**2. Reassigning a single element**

You can reassign individual elements too.

1. >>> colors=['caramel','gold','silver','occur']
2. >>> colors[3]='bronze'
3. >>> colors

[‘caramel’, ‘gold’, ‘silver’, ‘bronze’]

Now if you want to add another item ‘holographic’ to the list, we cannot do it the conventional way.

1. >>> colors[4]='holographic'
2. >>>colors.append(‘holographic’)
3. >>>colors [‘caramel’, ‘gold’, ‘silver’, ‘bronze’, ‘holographic’]

## **3. How can we Delete a Python List?**

You can delete a Python list, some of its elements, or a single element.

### a. Deleting the entire Python list

Use the del keyword for the same.

1. >>> del colors
2. >>> colors

NameError: name ‘colors’ is not defined

b. Deleting a few elements

Use the slicing [operator in python](https://data-flair.training/blogs/python-operators/) to delete a slice.

1. >>> colors=['caramel','gold','silver','bronze','holographic']
2. >>> del colors[2:4]
3. >>> colors

[‘caramel’, ‘gold’, ‘holographic’]

1. >>> colors[2]

‘holographic’

c. Deleting a single element

To delete a single element from a Python list, use its index.

1. >>> del colors[0]
2. >>> colors

[‘gold’, ‘holographic’]

## **8. Multidimensional Lists in Python**

1)You can also put a list in a list. Let’s look at a multidimensional list. That is nested list

1. >>> grocery\_list=[['caramel','P&B','Jelly'],['onions','potatoes'],['flour','oil']]
2. >>> grocery\_list

[[‘caramel’, ‘P&B’, ‘Jelly’], [‘onions’, ‘potatoes’], [‘flour’, ‘oil’]]

2) you can choose to go deeper.

1. >>> a=[[[1,2],[3,4],5],[6,7]]
2. >>> a

[[[1, 2], [3, 4], 5], [6, 7]]

To access the element 4 here, we type the following code into the shell.

1. >>> a[0][1][1] 4

## **9. Concatenation of Python List**

1) The concatenation operator works for lists as well. It lets us join two lists, with their orders preserved.

1. >>> a,b=[3,1,2],[5,4,6]
2. >>> a+b

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

## **10. Python List Operations**

### a)Multiplication:

This is an arithmetic operation. Multiplying a Python list by an integer makes copies of its items that a number of times while preserving the order.

1. >>> a\*=3
2. >>> a

[3, 1, 2, 3, 1, 2, 3, 1, 2]

However, you can’t multiply it by a float.

1. >>> a\*3.0

TypeError: can’t multiply sequence by non-int of type ‘float’

b)Membership:

You can apply the ‘in’ and ‘not in’ operators on a Python list.

1. >>> 1 in a

True

1. >>> 2 not in a

False

## **12. Python List Comprehension**

You can create a new list just like you would do in mathematics. To do so, type an expression followed by a for statement, all inside square brackets. You may assign it to a variable. Let’s make a list for all even numbers from 1 to 20.

1. >>> even=[2\*i for i in **range**(1,11)]
2. >>> even

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

Optionally, you can add an if-statement to filter out items. If we want to change this list to hold only those items from 1 to 20 that are even and are divisible by 3, we write the following code.

1. >>> even=[2\*i for i in **range**(1,11) if i%3==0]
2. >>> even

[6, 12, 18]



### a. len()

It calculates the length of the list.

>>> len(even)

1. >>> len(even)

>>> len(even)

3

### b. max()

a)It returns the item from the list with the highest value.

>>> max(even)

1. >>> max(even)

>>> max(even)

18

b)If all the items in your list are strings, it will compare.

>>> max(['1','2','3'])

1. >>> max(['1','2','3'])

>>> max(['1','2','3'])

3

c)But it fails when some are numeric, and some are [strings in python](https://data-flair.training/blogs/python-strings/).

>>> max([2,'1','2'])

1. >>> max([2,'1','2'])

>>> max([2,'1','2'])

TypeError: ‘>’ not supported between instances of ‘str’ and ‘int’

### c. min()

a)It returns the item from the Python list with the lowest value.

>>> min(even)

1. >>> min(even)

**>>> min(even)**

**6**

### d. sum()

**a)**It returns the sum of all the elements in the list.

>>> sum(even)

1. >>> sum(even)

>>> sum(even)

36

**b)**However, for this, the Python list must hold all numeric values.

>>> a=['1','2','3']

>>> sum(a)

1. >>> a=['1','2','3']
2. >>> sum(a)

>>> a=['1','2','3']

>>> sum(a)

sum(a)

TypeError: unsupported operand type(s) for +: ‘int’ and ‘str’

**c)**It works on floats.

>>> sum([1.1,2.2,3.3])

1. >>> sum([1.1,2.2,3.3])

>>> sum([1.1,2.2,3.3])

6.6

### E) sorted()

**a)** It returns a sorted version of the list, but does not change the original one.

>>> a=[3,1,2]

>>> sorted(a)

1. >>> a=[3,1,2] 🡪 [1, 2, 3] **// It returns the answer**

2.>>> sorted(a)  **original list not sorted**

3.>>> a.sort()>>> a=[3,1,2]>>> sorted(a)

4.>>> a 🡪 [1, 2, 3] **//The original list itself is sorted**

>>> a

>>> a

**b) *If the Python list members are strings, it sorts them according to their ASCII values.***

>>> sorted(['hello','hell','Hello'])

### >>> b=["Apple","Pineapple","Tomato"]

### >>> b.sort()

### >>> b

### ['Apple', 'Pineapple', 'Tomato']

### F) list()

It converts a different data type into a list.

>>> list("abc")

1. >>> list("abc")

>>> list("abc")

[‘a’, ‘b’, ‘c’]

It can’t convert a single int into a list, though, it only converts iterables.

>>> list(2)

1. >>> list(2)

>>> list(2)

TypeError: ‘int’ object is not iterable

### G] any()

**a)I**t returns True if even one item in the Python list has a True value.

>>> any(['','','1'])

1. >>> my\_list=['','','1']
2. >>> any(my\_list)

>>> any(['','','1'])

True

**b)**It returns False for an empty iterable.

>>> any([])

1. >>> any([])

>>> any([])

False

### H] all()

It returns True if all items in the list have a True value.

>>> all(['','','1'])

1. >>> all(['','','1'])

>>> all(['','','1'])

False

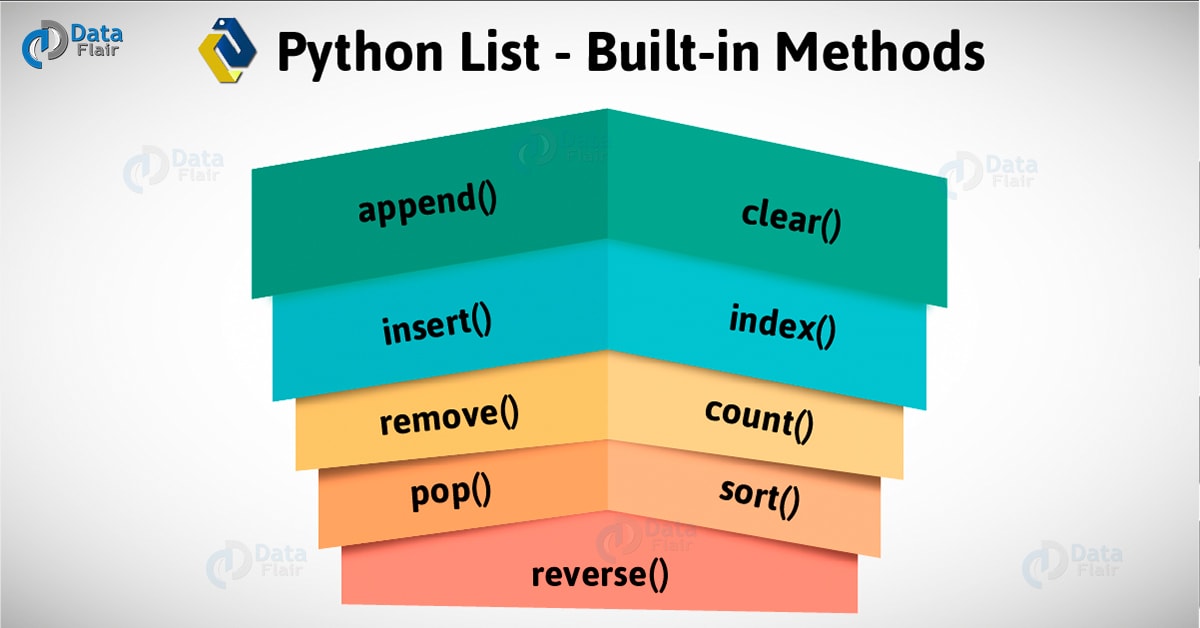
It returns True for an empty iterable.

>>> all([])

1. >>> all([])

>>> all([])

True



#### **append()**

It adds an item to the end of the list.

1. >>> a

[2, 1, 3]

1. >>> a.**append**(4)
2. >>> a

[2, 1, 3, 4]

#### **b)insert()**

It inserts an item at a specified position.

1. >>> a.**insert**(3,5)
2. >>> a

[2, 1, 3, 5, 4]

This inserted the element 5 at index 3.

#### **c)remove()**

It removes the first instance of an item from the Python list.

1. >>> a=[2,1,3,5,2,4]
2. >>> a.**remove**(2)
3. >>> a

[1, 3, 5, 2, 4]

Notice how there were two 2s, but it removed only the first one.

#### **d) pop()**

It removes the element at the specified index, and prints it to the screen.

If no value is specified in the pop() by default the last element of the list will be deleted.

1. >>> a.**pop**(3)

2

2.>>> a

[1, 3, 5, 4]

1. >>>a.pop()
2. >>>a

[1, 3, 5]

#### **e) clear()**

It empties the Python list.

1. >>> a.**clear**()
2. >>> a

[]

It now has a False value.

1.>>> **bool**(a) False

#### **F) index()**

It returns the first matching index of the item specified.

1. >>> a=[1,3,5,3,4]
2. >>> a.**index**(3)

1

#### **g) count()**

It returns the count of the item specified.

1. >>> a.**count**(3)

2

#### **h) sort()**

It sorts the list in an ascending order.

1. >>> a.**sort**()
2. >>> a

[1, 3, 3, 4, 5]

#### **i)reverse()**

It reverses the order of elements in the Python lists.

1. >>> a.**reverse**()
2. >>> a

[5, 4, 3, 3, 1]