Python Lists with Examples – A Comprehensive Tutorial 1

data-flair.training/blogs/python-lists-examples

Contents

1. Objective

In today's tutorial, we will learn about Python lists. We will discuss about how to create, access, slice, and reassign them. Then we will see how to apply functions on them. But before that, you should take a look at our tutorial on Python Variables and Data Types.

Creating lists
Accessing lists
Slicing lists
Reassigning lists(mutable)
Deleting elements
Multidimensional Lists
Python
Lists
Concatenation of Lists
Operations on Lists
Iterating on a list
List Comprehension
Built-in Functions
Built-in Methods

2. An Introduction to Python Lists

Unlike C++ or Java, Python doesn't have arrays. To hold a sequence of values, then, it provides the 'list' class. A list can be seen as a collection of values.

3. Creating lists

To create python lists of items, you need to mention the items, separated by commas, in square brackets. This is the python syntax you need to follow. Then assign it to a variable. Remember once again, you don't need to declare the data type, because Python is dynamically-typed.

```
>>> colors=['red','green','blue']
```

A list may hold different types of values.

```
>>> days=['Monday','Tuesday','Wednesday',4,5,6,7.0]
```

A list may have python lists.

```
>>> languages=[['English'],['Gujarati'],['Hindi'],'Romanian','Spanish']
>>> languages
```

```
[['English'], ['Gujarati'], ['Hindi'], 'Romanian', 'Spanish']
```

```
>>>type(languages[0])
<class'list'>
```

A list may also contain tuples or so.

```
>>> languages=[('English',’Albanian’),'Gujarati','Hindi','Romanian','Spanish']
>>> languages[0]
```

(‘English’, ‘Romanian’)

```
>>>type(languages[0])
<class'tuple'>
>>> languages[0][0]='Albanian'
Traceback(most recent call last):
File "<pyshell#24>", line 1, in <module>
languages[0][0]='Albanian'
```

TypeError: ‘tuple’ object does not support item assignment

4. Accessing Python lists

To access a list as a whole, all you need is its name.

```
>>> days
```

```
[‘Monday’, ‘Tuesday’, ‘Wednesday’, 4, 5, 6, 7.0]
```

Or, you can put it in a print statement.

```
>>> languages=[['English'],['Gujarati'],['Hindi'],'Romanian','Spanish']
>>>print(languages)
```

```
[[‘English’], [‘Gujarati’], [‘Hindi’], ‘Romanian’, ‘Spanish’]
```

To access a single element, use its index in square brackets after the list’s name. Indexing begins at 0.

```
>>> languages[0]
```

```
[‘English’]
```

An index cannot be a float value.

```
>>> languages[1.0]
Traceback(most recent call last):
File "<pyshell#70>", line 1, in <module>
languages[1.0]
```

TypeError: list indices must be integers or slices, not float

5. Slicing lists

When you want only a part of a list, you can use the slicing operator [].

1. >>> indices=['zero','one','two','three','four','five']
2. >>> indices[2:4]

[‘two’, ‘three’]

This returns items from index 2 to index 4-1 (i.e., 3)

>>> indices[:4]

[‘zero’, ‘one’, ‘two’, ‘three’]

This returns items from the beginning of the list to index 3.

>>> indices[4:]

[‘four’, ‘five’]

This returns items from index 4 to the end of the list.

>>> indices[:]

[‘zero’, ‘one’, ‘two’, ‘three’, ‘four’, ‘five’]

This returns the whole list.

> **Negative indices-** The indices we mention can be negative as well. A negative index means traversal from the end of the list.

>>> indices[:-2]

[‘zero’, ‘one’, ‘two’, ‘three’]

This returns items from the list’s beginning to two items from the end.

>>> indices[1:-2]

[‘one’, ‘two’, ‘three’]

This returns items from the item at index 1 to two items from the end.

>>> indices[-2:-1]

[‘four’]

This returns items from two from the end to one from the end.

>>> indices[-1:-2]

[]

This returns an empty list, because the start is ahead of the stop for the traversal.

6. Reassigning lists(mutable)

Python Lists are mutable. This means that you can reassign its items, or you can reassign it as a whole. Let’s take a new list.

>>> colors=['red','green','blue']

a. Reassigning the whole list

You can reassign a list by assigning it like a new list.

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

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

b. Reassigning a few elements

You can also reassign a slice of a list.

1. >>> colors[2:]=['bronze','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.

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

[‘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’]

c. 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.

```
>>> color[4]='holographic'
Traceback(most recent call last):
File "<pyshell#47>", line 1, in <module>
color[4]='holographic'
```

NameError: name ‘color’ is not defined

So, you need to reassign the whole list for the same.

```
>>> colors=['caramel','gold','silver','bronze','holographic']
>>> colors
```

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

7. Deleting elements

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

a. Deleting the entire list

Use the del keyword for the same.

```
>>> del colors
>>> colors
Traceback(most recent call last):
File "<pyshell#51>", line 1, in <module>
colors
```

NameError: name ‘colors’ is not defined

b. Deleting a few elements

Use the slicing operator in python to delete a slice.

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

[‘caramel’, ‘gold’, ‘holographic’]

>>> colors[2]

‘holographic’

Now, ‘holographic’ is at position 2.

c. Deleting a single element

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

```
>>> del colors[0]
>>> colors
```

['gold', 'holographic']

8. Multidimensional Lists

You can also put a list in a list. Let's look at a multidimensional list.

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

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

This is a grocery list with lists in it, where the lists are according to category.

Or, 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.

>>> a[0][1][1]

4

9. Concatenation of Python Lists

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. Operations on Lists in Python

a. Multiplication

This is an arithmetic operation. Multiplying a list by an integer makes copies of its items that 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
2. Traceback(most recent call last):

3. File "<pyshell#89>", line 1, in <module>
4. 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 list.

>>>1 in a

True

>>>2 not in a

False

11. Iterating on a list

A list can be traversed with a for loop in python.

1. >>>for i in [1,2,3]:
2. if i%2==0:
3. print(f"{i} is composite\n")

2 is composite

12. 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]

13. Built-in List Functions

There are some built-in functions in Python that you can use on python lists.

a. len()

It calculates the length of the list.

>>>len(even)

3

b. max()

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

>>>max(even)

18

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

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

‘3’

But it fails when some are numeric, and some are strings in python.

1. >>>max([2,'1','2'])
2. Traceback(most recent call last):
3. File "<pyshell#116>", line 1, in <module>
4. max([2,'1','2'])

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

c. min()

It returns the item from the list with the lowest value.

>>>min(even)

6

d. sum()

It returns the sum of all the elements in the list.

>>>sum(even)

36

However, for this, the list must hold all numeric values.

1. >>> a=['1','2','3']
2. >>>sum(a)
3. Traceback(most recent call last):
4. File "<pyshell#112>", line 1, in <module>
5. sum(a)

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

It works on floats.

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

6.6

e. sorted()

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

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

[1, 2, 3]

>>> a

[3, 1, 2]

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

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

[‘Hello’, ‘hell’, ‘hello’]

Here, since H has an ASCII value of 65, it appears first.

f. list()

It converts a different data type into a list.

>>>list("abc")

[‘a’, ‘b’, ‘c’]

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

1. >>>list(2)
2. Traceback(most recent call last):
3. File "<pyshell#122>", line 1, in <module>
4. list(2)

TypeError: ‘int’ object is not iterable

g. any()

It returns True if even one item in the list has a True value.

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

True

h. all()

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

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

```
False
```

14. Built-in Methods

While a function is what you can apply on a construct and get a result, a method is what you can do to it and change it. To call a method on a construct, you use the dot-operator(.). Python supports some built-in methods to alter a list.

a. append()

It adds an item to the end of the list.

```
>>> a
```

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

```
>>> a.append(4)
>>> a
```

```
[2, 1, 3, 4]
```

b. insert()

It inserts an item at a specified position.

```
>>> a.insert(3,5)
>>> 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 list.

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

```
>>> a.pop(3)

2

>>> a

[1, 3, 5, 4]
```

e. clear()

It empties the list.

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

[]
```

It now has a False value.

```
>>>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.

```
>>> 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 list.

```
1. >>> a.reverse()
```

2. >>> a

[5, 4, 3, 3, 1]

This was all about the Python lists

15. Conclusion

Woah, that was a lot, wasn’t it? Let’s make a quick revision so you don’t forget it. In this lesson on Python Lists, we first looked at how to declare and access a list. That included slicing lists in python. Then we looked at how to delete and reassign elements or an entire list. Next, we learned about multidimensional lists and list comprehension. We saw how to iterate on python lists, concatenate them, and also the operations that you can perform on them. Lastly, we looked at some built-in functions and methods that you can call on lists. Hope you enjoyed, see you again.