we'll learn everything about Python lists: creating lists, changing list elements, removing elements, and other list operations with the help of examples

Python lists are one of the most versatile data types that allow us to work with multiple elements at once. For example,

# a list of programming languages

['Python', 'C++', 'JavaScript']

## Create Python Lists

In Python, a list is created by placing elements inside square brackets [], separated by commas.

# list of integers

my\_list = [1, 2, 3]

A list can have any number of items and they may be of different types (integer, float, string, etc.).

# empty list

my\_list = []

# list with mixed data types

my\_list = [1, "Hello", 3.4]

A list can also have another list as an item. This is called a nested list.

# nested list

my\_list = ["mouse", [8, 4, 6], ['a']]

## Access List Elements

There are various ways in which we can access the elements of a list.

### **List Index**

We can use the index operator [] to access an item in a list. In Python, indices start at 0. So, a list having 5 elements will have an index from 0 to 4.

Trying to access indexes other than these will raise an IndexError. The index must be an integer. We can't use float or other types, this will result in TypeError.

Nested lists are accessed using nested indexing.

my\_list = ['p', 'r', 'o', 'b', 'e']

# first itemprint(my\_list[0]) # p

# third itemprint(my\_list[2]) # o

# fifth itemprint(my\_list[4]) # e

# Nested List

n\_list = ["Happy", [2, 0, 1, 5]]

# Nested indexingprint(n\_list[0][1])

print(n\_list[1][3])

# Error! Only integer can be used for indexingprint(my\_list[4.0])

****Output****

p

o

e

a

5

Traceback (most recent call last):

File "<string>", line 21, in <module>

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

### **Negative indexing**

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

# Negative indexing in lists

my\_list = ['p','r','o','b','e']

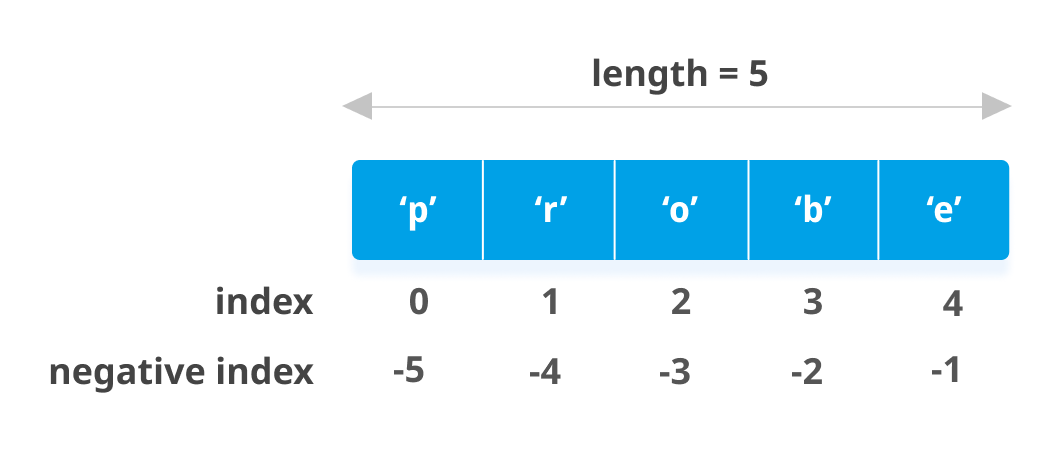
# last itemprint(my\_list[-1])

# fifth last itemprint(my\_list[-5])

****Output****

e

p

List indexing in Python

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | “Chandra” | “Door No:45 Anantapur” | “India” |
| 0 | 1 | 2 | 3 |
| -4 | -3 | -2 | -1 |

0

Total count -1

Total count -1

Mydetails

Details1 = Mydetails[0]

Mydeails[0] = 20

## List Slicing in Python

We can access a range of items in a list by using the slicing operator :.

# List slicing in Python

my\_list = ['p','r','o','g','r','a','m','i','z']

# elements from index 2 to index 4print(my\_list[2:5])

# elements from index 5 to endprint(my\_list[5:])

# elements beginning to endprint(my\_list[:])

****Output****

['o', 'g', 'r']

['a', 'm', 'i', 'z']

['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']

****Note:**** When we slice lists, the start index is inclusive but the end index is exclusive. For example, my\_list[2: 5] returns a list with elements at index 2, 3 and 4, but not 5.

## Add/Change List Elements

Lists are mutable, meaning their elements can be changed unlike [string](https://www.programiz.com/python-programming/string) or [tuple](https://www.programiz.com/python-programming/tuple).

We can use the assignment operator = to change an item or a range of items.

# Correcting mistake values in a list

odd = [2, 4, 6, 8]

# change the 1st item

odd[0] = 1

print(odd)

# change 2nd to 4th items

odd[1:4] = [3, 5, 7]

print(odd)

We can add one item to a list using the append() method or add several items using the extend() method.

# Appending and Extending lists in Python

odd = [1, 3, 5]

odd.append(7)

print(odd)

odd.extend([9, 11, 13])

print(odd)

****Output****

[1, 3, 5, 7]

[1, 3, 5, 7, 9, 11, 13]

We can also use + operator to combine two lists. This is also called concatenation.

The \* operator repeats a list for the given number of times.

# Concatenating and repeating lists

odd = [1, 3, 5]

print(odd + [9, 7, 5])

print(["re"] \* 3)

****Output****

[1, 3, 5, 9, 7, 5]

['re', 're', 're']

Furthermore, we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

# Demonstration of list insert() method

odd = [1, 9]

odd.insert(1,3)

print(odd)

odd[2:2] = [5, 7]

print(odd)

****Output****

[1, 3, 9]

[1, 3, 5, 7, 9]

## Delete List Elements

We can delete one or more items from a list using the [Python del statement](https://www.programiz.com/python-programming/del" \o "Python del). It can even delete the list entirely.

# Deleting list items

my\_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# delete one itemdel my\_list[2]

print(my\_list)

# delete multiple itemsdel my\_list[1:5]

print(my\_list)

# delete the entire listdel my\_list

# Error: List not definedprint(my\_list)

****Output****

['p', 'r', 'b', 'l', 'e', 'm']

['p', 'm']

Traceback (most recent call last):

File "<string>", line 18, in <module>

NameError: name 'my\_list' is not defined

We can use remove() to remove the given item or pop() to remove an item at the given index.

The pop() method removes and returns the last item if the index is not provided. This helps us implement lists as stacks (first in, last out data structure).

And, if we have to empty the whole list, we can use the clear() method.

my\_list = ['p','r','o','b','l','e','m']

my\_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']print(my\_list)

# Output: 'o'print(my\_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']print(my\_list)

# Output: 'm'print(my\_list.pop())

# Output: ['r', 'b', 'l', 'e']print(my\_list)

my\_list.clear()

# Output: []print(my\_list)

****Output****

['r', 'o', 'b', 'l', 'e', 'm']

o

['r', 'b', 'l', 'e', 'm']

m

['r', 'b', 'l', 'e']

[]

Finally, we can also delete items in a list by assigning an empty list to a slice of elements.

>>> my\_list = ['p','r','o','b','l','e','m']>>> my\_list[2:3] = []>>> my\_list

['p', 'r', 'b', 'l', 'e', 'm']>>> my\_list[2:5] = []>>> my\_list

['p', 'r', 'm']