Python List

There are many built-in types in Python that allow us to group and store multiple items. Python lists are the most versatile among them.

For example, we can use a Python list to store a playlist of songs so that we can easily add, remove, and update songs as needed.

Create a Python List

We create a list by placing elements inside square brackets `[]`, separated by commas. For example,

```
 # a list of three elements
ages = [19, 26, 29]
print(ages)

# Output: [19, 26, 29]
```

Here, the `ages` list has three items.

List Items of Different Types

We can store data of different data types in a Python list. For example,

```
# a list containing strings and numbers
student = ['Jack', 32, 'Computer Science']
print(student)

# an empty list
empty_list = []
print(empty_list)
```

Using list() to Create Lists

We can use the built-in list() function to convert other iterables (strings, dictionaries, tuples, etc.) to a list.

```
x = "axz"

# convert to list
result = list(x)

print(result)  # ['a', 'x', 'z']
```

List Characteristics

Lists are:

- **Ordered** - They maintain the order of elements.
- **Mutable** - Items can be changed after creation.
- **Allow duplicates** - They can contain duplicate values.

Access List Elements

Each element in a list is associated with a number, known as an **index**.

The index of first item is **0**, the index of second item is **1**, and so on.

Index of List Elements

We use these index numbers to access list items. For example,

```
languages = ['Python', 'Swift', 'C++']

# Access the first element
print(languages[0])   # Python

# Access the third element
print(languages[2])   # C++
```

Access List Elements Using Index

Access List Elements

More on Accessing List Elements

Negative Indexing in Python

Python also supports negative indexing. The index of the last element is **-1**, the second-last element is **-2**, and so on.

Python Negative Indexing

Negative indexing makes it easy to access list items from last.

Let's see an example,

```
languages = ['Python', 'Swift', 'C++']

# Access item at index 0
print(languages[-1])   # C++

# Access item at index 2
print(languages[-3])   # Python
```

Slicing of a List in Python

In Python, it is possible to access a section of items from the list using the slicing operator `:`. For example,

```
my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm']

# items from index 2 to index 4
print(my_list[2:5])

# items from index 5 to end
print(my_list[5:])

# items beginning to end
print(my_list[:])
```

Output

```
['o', 'g', 'r']
['a', 'm']
['p', 'r', 'o', 'g', 'r', 'a', 'm']
```

To learn more about slicing, visit Python program to slice lists.

Note: If the specified index does not exist in a list, Python throws the `IndexError` exception.

Add Elements to a Python List

We use the append() method to add an element to the end of a Python list. For example,

```
fruits = ['apple', 'banana', 'orange']
print('Original List:', fruits)

# using append method
fruits.append('cherry')


print('Updated List:', fruits)
```

Output

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'orange', 'cherry']
```

Add Elements at the Specified Index

The insert() method adds an element at the specified index. For example,

```
fruits = ['apple', 'banana', 'orange']
print("Original List:", fruits)

# insert 'cherry' at index 2
fruits.insert(2, 'cherry')


print("Updated List:", fruits)
```

Output

```
Original List: ['apple', 'banana', 'orange']
Updated List: ['apple', 'banana', 'cherry', 'orange']
```

Add Elements to a List From Other Iterables

We use the extend() method to add elements to a list from other iterables. For example,

```
numbers = [1, 3, 5]
print('Numbers:', numbers)

even_numbers  = [2, 4, 6]

# adding elements of one list to another
numbers.extend(even_numbers)


print('Updated Numbers:', numbers)
```

Output

```
Numbers: [1, 3, 5]
Updated Numbers: [1, 3, 5, 2, 4, 6]
```

Change List Items

We can change the items of a list by assigning new values using the = operator. For example,

```
colors = ['Red', 'Black', 'Green']
print('Original List:', colors)

# changing the third item to 'Blue'
colors[2] = 'Blue'


print('Updated List:', colors)
```

Output

```
Original List: ['Red', 'Black', 'Green']
Updated List: ['Red', 'Black', 'Blue']
```

Here, we have replaced the element at index 2: `'Green'` with `'Blue'`.

Remove an Item From a List

We can remove an item from a list using the remove() method. For example,

```
numbers = [2,4,7,9]

# remove 4 from the list
numbers.remove(4)


print(numbers)

# Output: [2, 7, 9]
```

Remove One or More Elements of a List

The del statement removes one or more items from a list. For example,

```
names = ['John', 'Eva', 'Laura', 'Nick', 'Jack']

# deleting the second item
del names[1]
print(names)

# deleting items from index 1 to index 3
del names[1: 4]
print(names) # Error! List doesn't exist.
```

Output

```
['John', 'Laura', 'Nick', 'Jack']
['John']
```

Note: We can also use the `del` statement to delete the entire list. For example,

```
names = ['John', 'Eva', 'Laura', 'Nick']

# deleting the entire list
del names

print(names)
```

Python List Length

We can use the built-in len() function to find the number of elements in a list. For example,

```
cars = ['BMW', 'Mercedes', 'Tesla']

print('Total Elements: ', len(cars))

# Output: Total Elements:  3
```

Iterating Through a List

We can use a for loop to iterate over the elements of a list. For example,

```
fruits = ['apple', 'banana', 'orange']

# iterate through the list
for fruit in fruits:
    print(fruit)
```

Output

```
apple
banana
orange
```

Python List Methods

Python has many useful list methods that make it really easy to work with lists.

Method	Description
append()	Adds an item to the end of the list
extend()	Adds items of lists and other iterables to the end of the list
insert()	Inserts an item at the specified index
remove()	Removes the specified value from the list
pop()	Returns and removes item present at the given index
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of the specified item in the list
sort()	Sorts the list in ascending/descending order
reverse()	Reverses the item of the list
copy()	Returns the shallow copy of the list

More on Python Lists

List Comprehension in Python

List Comprehension is a concise and elegant way to create a list. For example,

```
# create a list with square values
numbers = [n**2 for n in range(1, 6)]
print(numbers)

# Output: [1, 4, 9, 16, 25]
```

To learn more, visit Python List Comprehension.

Check if an Item Exists in the Python List

We use the `in` keyword to check if an item exists in the list. For example,

```
fruits = ['apple', 'cherry', 'banana']

print('orange' in fruits)    # False
print('cherry' in fruits)    # True
```

Here,

- **orange** is not present in `fruits`, so, `'orange' in fruits` evaluates to `False`.
- **cherry** is present in `fruits`, so, `'cherry' in fruits` evaluates to `True`.

Note: Lists are similar to arrays (or dynamic arrays) in other programming languages. When people refer to arrays in Python, they often mean lists, even though there is a numeric array type in Python.

Also Read

- Python list()

Table of Contents