Python List

A list in Python is used to store the sequence of various types of data. Python lists are mutable type its mean we can modify its element after it created. However, Python consists of six data-types that are capable to store the sequences, but the most common and reliable type is the list.

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

Characteristics of Lists

The lists are ordered.

The element of the list can access by index.

The lists are mutable types.

A list can store the number of various elements.

List indexing and splitting

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.

Consider the following example:

```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
# Slicing the elements
print(list[0:6])
# By default the index value is 0 so its starts from the 0th element and go for index
 -1.
print(list[:])
print(list[2:5])
print(list[1:6:2])
```

Unlike other languages, 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.

```
list = [1,2,3,4,5]
print(list[-1])
print(list[-3:])
print(list[:-1])
print(list[-3:-1])
```

Updating List values

Lists are the most versatile data structures in Python since they are mutable, and their values can be updated by using the slice and assignment operator.

Python also provides append() and insert() methods, which can be used to add values to the list.

Consider the following example to update the values inside the list.

```
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to the second index
list[2] = 10
print(list)
# Adding multiple-element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)
```

Adding elements to the list

Python provides **append()** function which is used to add an element to the list. However, the append() function can only add value to the end of the list.

```
#Declaring the empty list
l =[]
#Number of elements will be entered by the user
n = int(input("Enter the number of elements in the list:"))
# for loop to take the input
for i in range(0,n):
    # The input is taken from the user and added to the list as the item
    l.append(input("Enter the item:"))
print("printing the list items..")
# traversal loop to print the list items
for i in l:
    print(i, end = "  ")
```

Removing elements from the list

```
list = [0,1,2,3,4]
print("printing original list: ");
for i in list:
    print(i,end=" ")
list.remove(2)
print("\nprinting the list after the removal of first element...")
for i in list:
    print(i,end=" ")
```

Example: 1- Write the program to remove the duplicate element of the list.

```
list1 = [1,2,2,3,55,98,65,65,13,29]
# Declare an empty list that will store unique values
list2 = []
for i in list1:
    if i not in list2:
        list2.append(i)
print(list2)
```

Example: 2- Write a program to find the sum of the element in the list.

```
list1 = [3,4,5,9,10,12,24]
sum = 0
for i in list1:
    sum = sum+i
print("The sum is:",sum)
```

Example: 3- Write the program to find the lists consist of at least one common element.

```
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,2,10]
for x in list1:
    for y in list2:
        if x == y:
            print("The common element is:",x)
```

Python List Operations

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings.

Let's see how the list responds to various operators.

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]
Concatenation	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
Length	It is used to get the length of the list	len(l1) = 4

Python List Built-in functions

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

No	Function	Description	Example
1	cmp(list1, list2)	It compares the elements of both the lists.	This method is not used in the Python 3 and the above versions.
2	len(list)	It is used to calculate the length of the list.	L1 = [1,2,3,4,5,6,7,8] print(len(L1)) **ans=8**
3	max(list)	It returns the maximum element of the list.	L1 = [12,34,26,48,72] print(max(L1)) **Ans=72**
4	min(list)	It returns the minimum element of the list.	L1 = [12,34,26,48,72] print(min(L1)) **ans=12**
5	list(seq)	It converts any sequence to the list.	str = "Python" s = list(str) print(type(s)) **ans=<class list>**

Python Tuple

Python Tuple is used to store the sequence of immutable Python objects. The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

Creating a tuple

A tuple can be written as the collection of comma-separated (,) values enclosed with the small () brackets. The parentheses are optional but it is good practice to use. A tuple can be defined as follows.

```
T1 = (101, "Suresh", 22)
T2 = ("Apple", "Banana", "Orange")
T3 = 10,20,30,40,50

print(type(T1))
print(type(T2))
print(type(T3))
```

An empty tuple can be created as follows.

```
T4 = ()
```

Creating a tuple with single element is slightly different. We will need to put comma after the element to declare the tuple.

```
tup1 = ("python developer")
print(type(tup1))

#Creating a tuple with single element
tup2 = ("python developer ",)
print(type(tup2))
```

Output:

```
<class 'str'>
<class 'tuple'>
```

A tuple is indexed in the same way as the lists. The items in the tuple can be accessed by using their specific index value.

Consider the following example of tuple:

Example - 1

```
tuple1 = (10, 20, 30, 40, 50, 60)
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %d"%(count, i))
    count = count+1
```

Example – 2

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %s"%(count, i))
    count = count+1
```

Tuple indexing and slicing

The indexing and slicing in the tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.

The items in the tuple can be accessed by using the index [] operator. Python also allows us to use the colon operator to access multiple items in the tuple.

Consider the following example:

```
tup = (1,2,3,4,5,6,7)
print(tup[0])
print(tup[1])
print(tup[2])
print(tup[1:])
print(tuple[:4])
```

Negative Indexing

The tuple element can also access by using negative indexing. The index of -1 denotes the rightmost element and -2 to the second last item and so on.

```
tuple1 = (1, 2, 3, 4, 5)
print(tuple1[-1])
print(tuple1[-4])
print(tuple1[-3:-1])
```

Deleting Tuple

Unlike lists, the tuple items cannot be deleted by using the **del** keyword as tuples are immutable. To delete an entire tuple, we can use the **del** keyword with the tuple name.

Consider the following example.

```
tuple1 = (1, 2, 3, 4, 5, 6)
print(tuple1)
del tuple1[0]
print(tuple1)
```

Output:

```
(1, 2, 3, 4, 5, 6)
Traceback (most recent call last):
  File "tuple.py", line 4, in <module>
    print(tuple1)
NameError: name 'tuple1' is not defined
```

Python Tuple inbuilt functions

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple
4	min(tuple)	It returns the minimum element of the tuple.
5	tuple(seq)	It converts the specified sequence to the tuple.

List vs. Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the a variable length.	The tuple has the fixed length.
4	The list provides more functionality than a tuple.	The tuple provides less functionality than the list.
5	The list is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items cannot be changed. It can be used as the key inside the dictionary.
6	The lists are less memory efficient than a tuple.	The tuples are more memory efficient because of its immutability.

Python Dictionary:

Python Dictionary is used to store the data in a key-value pair format. The dictionary is the data type in Python, which can simulate the real-life data arrangement where some specific value exists for some particular key. It is the mutable data-structure. The dictionary is defined into element Keys and values.

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.

In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any Python object. In contrast, the keys are the immutable Python object, i.e., Numbers, string, or tuple.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the curly brackets {}, and each key is separated from its value by the colon (:).The syntax to define the dictionary is given below.

Syntax:

Dict = {"Name": "abc", "Age": 22}

In the above dictionary **Dict**, The keys **Name** and **Age** are the string that is an immutable object.

Let's see an example to create a dictionary and print its content.

```
Employee ={"Name": "ABC", "Age":29,"salary":25000,"Company":"TCS"}
print(type(Employee))
print("printing Employee data..... ")
print(Employee)
```

Output

```
<class 'dict'>
Printing Employee data ....
{'Name': 'ABC', 'Age': 29, 'salary': 25000, 'Company': 'TCS'}
```

Python provides the built-in function **dict()** method which is also used to create dictionary. The empty curly braces {} is used to create empty dictionary.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)
```

Adding dictionary values

The dictionary is a mutable data type, and its values can be updated by using the specific keys. The value can be updated along with key **Dict[key] = value**. The update() method is also used to update an existing value.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements to dictionary one at a time
Dict[0] = 'Amar'
Dict[2] = 'Rahul'
Dict[3] = 'Prasad'
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# with a single Key
# The Emp_ages doesn't exist to dictionary
Dict['Emp_ages'] = 20, 22, 25
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[3] = 'Welcome to Python World.'
print("\nUpdated key value: ")
print(Dict)
```

Adding User Input:

```
Employee = {}
print(type(Employee))
print("printing Employee data..... ")
print(Employee)
print("Enter the details of the new employee... ");
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

Deleting elements using del keyword

```
Employee = {"Name": "ABC", "Age": 29, "salary":25000,"Company":"TCS"}
print(type(Employee))
print("printing Employee data..... ")
print(Employee)
print("Deleting some of the employee data")
del Employee["Name"]
del Employee["Company"]
print("printing the modified information ")
print(Employee)
print("Deleting the dictionary: Employee");
del Employee
print("Lets try to print it again ");
print(Employee)
```

Using pop() method

The **pop()** method accepts the key as an argument and remove the associated value. Consider the following example.

```
# Creating a Dictionary
Dict = {1: 'Amar', 2: 'Rahul', 3: 'Ajay'}
# Deleting a key
# using pop() method
pop_ele = Dict.pop(3)
print(Dict)
```

Iterating Dictionary

A dictionary can be iterated using for loop as given below.

Example 1

 # for loop to print all the keys of a dictionary

```
Employee = {"Name": "ABC", "Age": 29, "salary":25000,"Company":"TCS"}
for x in Employee:
    print(x)
```

Example 2

#for loop to print all the values of the dictionary

```
Employee = {"Name": "ABC", "Age": 29, "salary":25000,"Company":"TCS"}
for x in Employee:
    print(Employee[x])
```

Example 3

#for loop to print all the keys and values of the dictionary

```
Employee={"Name":"ABC","Age":29,"Salary":25000,"Company":"TCS"}
for x,y in Employee.items():
    print(x,y)
```