PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

List in python

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

List

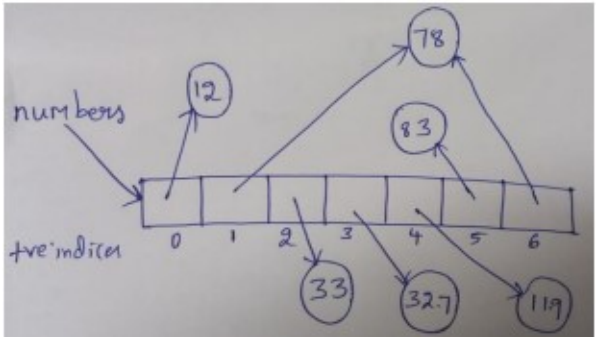
- **A list is a collection** that allows us to put many things / values under a single variable.
- Values in the list are called **elements / items**.
- List is an **ordered sequence of items.**
- List is a **linear data structure** where elements have linear ordering.

List Characteristics

- Elements in the list can be **heterogeneous** (different datatypes) or **homogeneous** (same datatype).

```
heterogeneous = [1,"rahul",3.5,{1,2,3}]
homogeneous = ["tendulkar","bolt"," federer","messi"]
```

- Elements are **accessed using indexing operation** (also called **subscript notation**).




```
>>> numbers = [12, 78, 33, 32.7, 11.9, 83, 78]
>>> print (numbers[1])
    78
```

List index always starts with 0, which is called **zero based indexing.**

List Characteristics

- Lists are mutable, as it can grow and shrink
- List is iterable - is eager and not lazy.
- Assignment of one list to another causes both to refer to the same list.
- List can be sliced. This creates a new (sub)list.

List Characteristics

- Lists are mutable, as list can grow or shrink .
    - We *can* change an element of a list.

Ex:

```
>>> numbers=[55,88,45,12]
>>> numbers[0]=10 # index operation is used.
>>> numbers
    [10, 88, 45, 12]
```

List Characteristics

- List is iterable - is eager and not lazy.

Ex: (i)
```
numbers=[55,88,45,12]
for i in numbers:
        print(i, end =' ')
```

Ex: (ii)
```
number=[10,20,30,40,50]
i=0
while(i<len(number)):
        print(number[i],end=' ')
        i=i+1
```

- List can be nested. We can have list of lists.

Ex: (i)
```
numbers=[55,20,[63,72,33]]
for i in numbers:
        print(i, end =' ')
```

Ex: (ii)
```
number=[55,20,[63,72,33]
i=0
while(i<len(number)):
        print(number[i],end=' ')
        i=i+1
```

PES UNIVERSITY
CELEBRATING 50 YEARS

List Characteristics

- Assignment of one list to another causes both to refer to the same list.

Ex:

```
>>> list1=[12,44,55,89,11,24]
>>> list2=list1
>>> print(id(list1))
    2894590353408
>>> print(id(list2))
    2894590353408
```

Note: In Python, the id() function is a built-in function that returns the unique identifier of an object.

List Characteristics

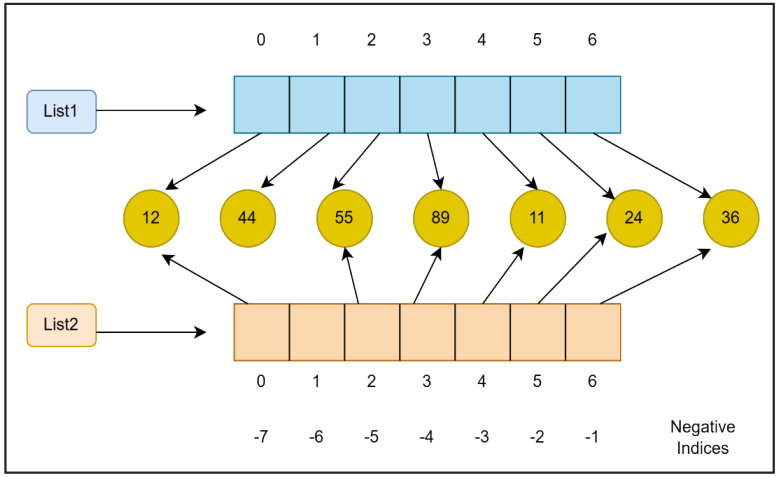
- List can be sliced. This creates a new (sub)list.

Given **lst1 = [12,44,55,89,11,24]**

```
>>> lst2 = lst1[::] # creates a copy of lst1. Not same as lst2 = lst1

>>> print(id(lst1))
    2894635511936

>>> print(id(lst2)) #These two are different values
    2894635298304
```



List

Ex:

```
>>> lst = [9, 41, 12, 3, 74, 15]
>>> lst[1:3]
    [41,12]
>>> lst[:4]
    [9, 41, 12, 3]
>>> lst[3:]
    [3, 74, 15]
>>> lst[:]
    [9, 41, 12, 3, 74, 15]
```

```
>>> lst1=[10,20,[30,40,50]]
>>> lst1[2][1]
    40
```

Creation of Lists

- List items are **surrounded by square brackets** and the elements in the list are **separated by commas**.

```
politicians=['modi', 'rahul', 'mamta', 'kejriwal']
```

- **A list element can be any Python object** - even **another list**.

```
politicians=['modi', 'jayalalitha', 'yediyurappa','devegowda',['parikar', 'swaraj', 'jately']]
```

- A list **can be empty**.

```
lst = [ ]
lst = list()
```

Built In Functions

There are a number of functions built into Python that take lists as parameters.

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
```

PES UNIVERSITY
CELEBRATING 50 YEARS

List Operations

- **Concatenation**

We can create a new list by **adding two existing lists** together.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>> list2 =[100,200,300,400,500]
>>> list1 + list2          #concatenates two lists
    [10,20,30,40,50, 100,200,300,400,500]
```

PES UNIVERSITY
CELEBRATING 50 YEARS

List Operations

- **Repetition**

  Allows for the multiplying of the list n times

Ex:

```
>>> list1 = [10,20,30,40,50]
>>> list1 * 2
    [10,20,30,40,50,10,20,30,40,50]
```

PES UNIVERSITY
CELEBRATING 50 YEARS

List Operations

- **Sorting**

  Used to **arrange the elements** of a list.

Ex:

```
>>> list1 = [10, 1, -2, 2, 9]
>>> list1.sort()
>>> list1
    [-2, 1, 2, 9, 10]
```

List Operations

- **append()**

allows to add element at the end of list.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>>list1.append(22)
 >>> list1
      [10,20,30,40,50,22]
```

List Operations

- **insert(pos,val)**

  Allows to add an element at particular position in the list.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>>list1.insert(3,55)
>>> list1
    [10,20,30,55,40,50]
```

List Operations

- **extend()**

adds the specified list elements (or any iterable) to the end of the current list.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>>list1.extend([11,22,33,44,55])
 >>> list1
     [10,20,30,40,50,11,22,33,44,55]
```

- **pop() & remove()**

  allows to remove element from a list by using pop() or remove() functions.

  One uses index value (pop),another uses value(remove) as reference to remove the element.

Ex:

```
>>> list1 = [10,20,30,40,50]

>>>list1.pop(2) # using pop()
>>> list1
[10,20,40,50]

>>> list1.remove(40)  #using remove()
>>> list1
[10,20,30,50]
```

List Operations

- **count(val)**

  returns number of occurrences of value.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>>list1.count(20)
1
```

List Operations

- **index (val)**

return first index of a value. Raises ValueError if the value is not present.

Ex:

```
>>> list1 = [10,20,30,40,50]
>>>list1.index(20)
1
```

PES UNIVERSITY
CELEBRATING 50 YEARS

List Operators

Membership Operator:

in and not in

in returns True if a particular item exists in the list. otherwise False

not in operator returns True if the element is not present, otherwise False

Ex:

```
>>> list1 = [10,2.2,(22,33,43),('python') ] # heterogeneous list.
>>> 'python' in list1
    True
>>> 'ruby' not in list1
    True
```

Comparison

We may at times need to compare data items in the two lists to perform certain operations by using == operator.

Ex:

```
>>> list1 = [10,2.2,(22,33,43]
>>> list2=[2,3,4]
>>> list1==list2
    False
>>>list1!=list2
    True
```

List Traversal

1. For loop
2. While loop

List

1. List using For Loop:

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- Iterating over a sequence is called traversal.
- Loop continues until we reach the last item in the sequence
- The body of for loop is separated from the rest of the code using indentation.

Syntax:

```
for val in sequence:
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

List

Accessing element	output
a=[10,20,30,40,50] for i in a: print(i)	1 2 3 4 5
Accessing index	**output**
a=[10,20,30,40,50] for i in range(0,len(a),1): print(i)	0 1 2 3 4
Accessing element using range:	**output**
a=[10,20,30,40,50] for i in range(0,len(a),1): print(a[i])	10 20 30 40 50

2. List using while loop:

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Syntax:

while (condition):

body of while

List

Example: Sum of elements in list

```
a=[1,2,3,4,5]
i=0
sum=0
while i<len(a):
        sum=sum+a[i]
        i=i+1
print(sum)
```

Output:

15

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD &CDSAML, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Prof. Chitra G M
Prof. Mohan Kumar A V

Ack: Teaching Assistant – Advaith Sanil Kumar