Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #28
Lists and it's operations

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M (Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Lists in Python

Introduction

List is a N**on-primitive Linear Data Structure**. It is a **collection of different types of values**. Let us try to understand why list is required through an example.

The requirement is to find the sum of 3 integers. It is very easy to create three variables and store three integers in these and use the addition operator to find the sum of it. Display the result. But if we have to find the sum of 10000 integers, we need to either use 10000 variables or we need to use looping constructs and update the sum every time you take the input from the user. If there is a change in requirement from the client saying sort all these integers first and display the sorted data and then print the sum. But we have not stored any of these data from the user if we are using the looping construct to find the sum of all entered data. How do we get access to all 10000 integers as we need them to sort and print it?

To make it possible, we use lists in python. A single variable of type list can hold all these 10000 integers entered by the user. We can access each value by indexing the elements in the list.

Characteristics of Lists:

- It has **0 or more elements.**
- It allows duplicate elements within it.
- There is no **name for each element** in a list separately.
- Elements are accessed using **indexing operation or by subscripting**.
- List is **indexable** - **Index always starts with 0**. This is known as **zero based indexing.** Negative indices are also supported.
- List is **mutable** – Can grow or shrink as its size is not fixed.
- List is **iterable –** Can get one element at a time.
- **Elements** in the list can be **heterogeneous** in nature.
- It has number of **built-in functions to manipulate itself.**

Syntactically, elements of the list must be inside **square brackets – [** and **]** and **all elements must be separated by a comma** between each of them .

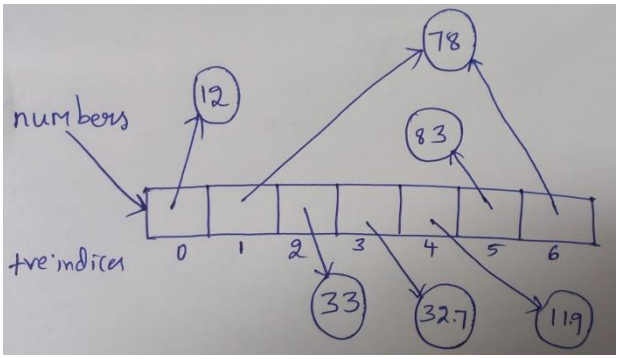
A. Create a list of numbers

```
numbers = [12, 78, 33, 32.7, 11.9, 83, 78]      #Duplicates allowed
                    #List has heterogeneous type of elements in it. As of now int and float
print(type(numbers))    # <class 'list'>
```

Note: Empty list can be created using [] or list()

B. Diagrammatic representation of a list

numbers = [12, 78, 33, 32.7, 11.9, 83, 78]



C. Accessing the elements of the list

The index begins with 0. To access 33, we can use **numbers[2].** If we want to use negative index, -1 is the index for the last element of the list. To access 33, we can say, **numbers[-5]. Max value of index is length of the list – 1. Accessing outside this index results in Index Error. The last element of the list** can be accessed **using -1 as the index or len(numbers) - 1**

```
>>> numbers = [12, 78, 33, 32.7, 11.9, 83, 78]
>>> numbers[2]
33
>>> numbers[-5]
33
>>> numbers[len(numbers)-1]
78
>>> numbers[-1]
78
>>>
```

D. Modifying the elements of the list

Use the **assignment operator to modify the elements of the list.** If we have change to 32.7 to 67, use the index of 32.7 and assign 67 to numbers[index].

```
>>> numbers
[12, 78, 33, 32.7, 11.9, 83, 78]
>>> numbers[3] = 67
>>> numbers
[12, 78, 33, 67, 11.9, 83, 78]
>>> type(numbers[3])
<class 'int'>
>>> numbers[-4] = 67.9
>>> type(numbers[3])
<class 'float'>
>>>
>>>
>>> numbers
[12, 78, 33, 67.9, 11.9, 83, 78]
>>>
```

E. Common functions that can be applied are len(), max(), min(), sum(). These functions are self-explanatory

```
>>> numbers = [12, 78, 33, 32.7, 11.9, 83, 78]
>>> len(numbers)
7
>>> max(numbers)
83
>>> min(numbers)
11.9
>>> sum(numbers)
328.6
```

F. Common operators that can be applied are +, *, in, not in, slicing operator(:) within index operator([]) and relational operators

- **+ ->** The **Concatenate operation** is used to merge two lists and creates new list.
- **->** The **Repetition operation** allows multiplying the list n times and create a new list.
- **in and not in ->** Used to **find whether the particular element exists in the list or not**.

```
>>> lst1 = [12,44,55,89,11,24]
>>> lst2 = [34,67,13,67]
>>> lst1 + lst2
[12, 44, 55, 89, 11, 24, 34, 67, 13, 67]
>>> lst2 * 2
[34, 67, 13, 67, 34, 67, 13, 67]
>>> lst1
[12, 44, 55, 89, 11, 24]
>>> lst2
[34, 67, 13, 67]
>>> lst3 = lst2 *3
>>> lst3
[34, 67, 13, 67, 34, 67, 13, 67, 34, 67, 13, 67]
>>>
```

```
>>> lst1
[12, 44, 55, 89, 11, 24]
>>> 44 in lst1
True
>>> '44' in lst1
False
>>> lst2
[34, 67, 13, 67]
>>> 44 not in lst2
True
>>> 67 not in lst2
False
>>>
```

- **Relational operators ->** Compares the corresponding elements until a mismatch or one or both ends.

  In detail: The first two items from two lists are compared, and if they differ, this determines the outcome of the comparison. If they are equal, the next two items from the same two lists are compared, and so on, until either sequence is exhausted.

```
>>> lst1 = [23,44,11,77]
>>> lst2 = [23,44,11,77]
>>> lst1 == lst2
True
>>> lst3 = [23,44,12]
>>> lst1 < lst3
True
>>> lst4 = [23,44,12,54]
>>> lst1 < lst4
True
>>> lst1 <= lst2
True
>>> lst1 > lst3
False
>>> lst1 >= lst3
False
>>>
```

- **Slicing operator [:] ->** Used to **create a new list based on the indices of the existing** list.

A slicing operator selects a range of items in a sequence object (e.g., a string, tuple or list). **Syntax: Lst[Initial : End : IndexJump]**

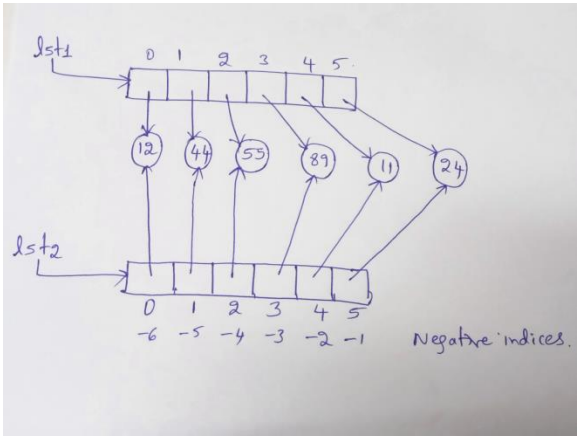
It returns the portion of list from Initial to index end, at a step size IndexJump.

Can use help(':') to know about the operator in detail. Let us see few examples:

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

```
lst2 = lst1[::]   # creates a copy of lst1. Not same as lst2 = lst1
print(id(lst1))
print(id(lst2))     #These two are different values
```

Refer to the below diagram to understand better.



Display the sliced list from lst1

```
lst1 = [12,44,55,89,11,24,"sindhu", 23.6, "pai", 12]
print(lst1[0:6])    #   [12, 44, 55, 89, 11, 24]
print(lst1[1:6])    #   [44, 55, 89, 11, 24]
print(lst1[1:6:2])  #   [44, 89, 24]
lst1[1:6:-2]        #   [ ]
lst1[6:1:-1]        #   ['sindhu', 24, 11, 89, 55]
```

Leaving any values like Initial, End, or IndexJump blank will lead to the use of default values i.e. 0 as Initial, length of the list as End, and 1 as IndexJump[When indexJump value is negative, it works little different].

```
>>> lst1 = [23,11,55,89,33,11,90,12,33,55]
>>> lst2 = lst1[::]
>>> lst3 = lst1[::2]
>>> lst4 = lst1[:7:]
>>> lst5 = lst1[2::]
>>> lst2
```

```
[23, 11, 55, 89, 33, 11, 90, 12, 33, 55]
>>> lst3
[23, 55, 33, 90, 33]
>>> lst4
[23, 11, 55, 89, 33, 11, 90]
>>> lst5
[55, 89, 33, 11, 90, 12, 33, 55]
```

A reversed list can be generated by using a negative integer as the IndexJump argument

```
>>>lst1 = [23,11,55,89,33,11,90,12,33,55]
>>> lst6 = lst1[::-1]
>>> lst6
[55, 33, 12, 90, 11, 33, 89, 55, 11, 23]
>>> lst7 = lst1[:1:-2]   #When –ve IndexJump value, Initial is considered as the last index.
>>> lst7
[55, 12, 11, 89]
>>>
```

Specific Functions on Lists: Some of the functions from this list proves that **List is a Mutable type**

Use dir() function to **display the list of functions a type supports**. Can use help() on any of these to know its job and it' usage. Samples are shown below.

```
>>> dir(list)
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt
__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len
__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed_
_', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', '
clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

```
>>> help(append)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'append' is not defined
```

```
>>> help(list.append)
Help on method_descriptor:

append(self, object, /)
    Append object to the end of the list.
```

```
>>> help(list.insert)
Help on method_descriptor:

insert(self, index, object, /)
    Insert object before index.
```

append(): This method inserts an element at the end of the list as one element

```
>>> lst1 = [23,55,11,88]
>>> lst1
[23, 55, 11, 88]
>>> lst1.append(56)
>>> lst1
[23, 55, 11, 88, 56]
>>> lst1.append("pes")
>>> lst1
[23, 55, 11, 88, 56, 'pes']
>>> len(lst1)
6
>>> lst1.append([12,61])
>>> lst1
[23, 55, 11, 88, 56, 'pes', [12, 61]]   #important
>>> len(lst1)
7
```

extend(): Extends list by appending elements from the iterable.

```
>>> lst1 = [23,55,11,88]
>>> lst1
[23, 55, 11, 88]
>>> lst1.extend([44,22])
>>> lst1
[23, 55, 11, 88, 44, 22]
>>> lst1.extend("pes")
>>> lst1
[23, 55, 11, 88, 44, 22, 'p', 'e', 's']
>>> lst1.extend(56)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

insert(): This method inserts an object at a specified position in the sequence.

```
>>> lst1 = [23,55,11,88]
>>> lst1
[23,55,11,88]
>>> lst1.insert(2, "pes")    # position is 2, object is “pes”
>>> lst1
[23, 55, 'pes', 11, 88]
>>> lst1.insert(len(lst1), "sindhu")  # this is similar to append function
>>> lst1
[23, 55, 'pes', 11, 88, 'sindhu']
```

Think about how to insert an element to the beginning of the list?

pop(): Remove and return item at index (default last).Raises IndexError if list is empty or index is out of range.

```
>>> lst1 = [23,55,11,88]
>>> lst1.pop()   #default behavior to remove the last element from the list and return it
88
>>> lst1
[23, 55, 11]
>>> lst1.pop(1)  # removes the element with index 1 and returns it.
55
>>> lst1
[23, 11]
>>> lst2 = []  #empty list created
>>> lst2.pop()   #Error on empty list  and error if there is no element with that index
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop from empty list
>>>
```

remove(): Removes first occurrence of value. Raises ValueError if the value is not present.

```
>>> lst1 = [23, 55, 11, 88, 55]
>>> lst1.remove(55)    # Doesn't return any value and removes the first occurrence only
>>> lst1
[23, 11, 88, 55]
>>> lst1.remove(89)   # Element 89 is not there is lst1.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
>>>
```

index(): Return first index of value. Raises ValueError if the value is not present.

```
  >>> lst1 = [23, 55, 11, 88, 55]
  >>> lst1.index(55)   # there are two 55 in the list. Index returns the index of first
occurrence of 55
  1
  >>> lst1
  [23, 55, 11, 88, 55]
  >>> lst1.index(555)  # 555 not there in lst1
  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  ValueError: 555 is not in list
  >>>
```

count(): Returns the number of occurrences of value.

```
>>> lst1 = [23, 55, 11, 88, 55, 123, 55, 23, 55, 555, 12]
>>> lst1.count(55)    # 55 is repeated 5 times
4
>>> lst1.count(999)   #999 is not there.
0
>>>
```

sort(): By default, sorts the elements of the list in ascending order.

```
>>> lst1 = [23, 55, 11, 88, 55]
>>> lst1
[23, 55, 11, 88, 55]
>>> lst1.sort()
>>> lst1
[11, 23, 55, 55, 88]
>>> lst2 = ["a","z","pes",56]
>>> lst2
['a', 'z', 'pes', 56]
>>> lst2.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
>>> lst2[:3:].sort()
>>> lst2
['a', 'pes', 'z', 56]
>>>
```

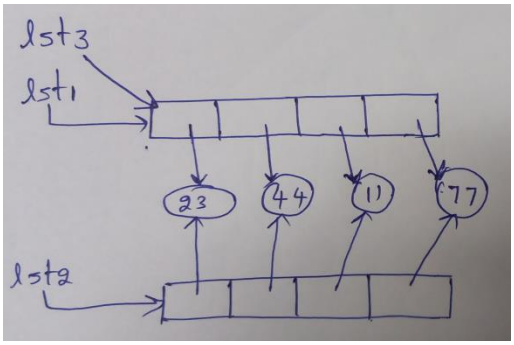
reverse(): This method reverses the order of elements of the list.

```
>>> lst1 = [23, 55, 11, 88, 55]
>>> lst1
[23, 55, 11, 88, 55]
>>>lst1.reverse()
>>> lst1
[55, 88, 11, 55, 23]
```

copy(): Returns the copy of the list.

```
>>> lst1
[23, 44, 11, 77]
>>> lst2 = lst1.copy()
>>> id(lst1)
```

```
2052013597312
>>> id(lst2)
2052013593728
>>> lst3 = lst1
>>> id(lst3)
2052013597312
>>> id(lst1)
2052013597312
>>>
```



clear(): This method removes all items from the list.

```
>>> lst1 = [12,77,14,19]
>>> lst1.clear()
>>> lst1
[]
>>>
```

Understand copy(), assignment operator and clear() function properly through this example.

```
>>> lst1 = [12,77,14,19]
>>> lst2 = lst1
>>> lst3 = lst1.copy()
>>> id(lst1), id(lst2), id(lst3)
(2052013593728, 2052013593728, 2052017030336)
>>> lst2.clear()
>>> lst2
[]
>>> lst1
[]
>>> lst3
[12, 77, 14, 19]
>>>
```

Note: Kindly write the pictorial representation to understand this code completely

There are many functions available for list type as shown in the result of dir() function above. **If you want to find whether some function is supported by list or not, use the in operator to do that.**

Example:

```
>>> 'append' in dir(list)
True
>>> 'apend' in dir(list)
False
>>>
```

TRY IT!

- **If you clear the list, will the list be deleted? If no, how to delete a list variable?**
- **If you want to sort the elements of the list in decreasing order of elements, which two functions you can cascade to get the result? Try solving this using the code.**
- **Is there a way to get the sorted elements of the list without touching the original list? Try sorted() function from built-ins.**
- **Can you pass an integer as an argument to extend() function?**
- **If you pass the set as an argument to append() function, what happens? Try it!**
- **Can you have a list inside list? If yes, how do you access the elements of the inner list?**
- **If the list li = [23,55,11,78,"sindhu", 67.3], then li[4+1] results in what? If error, What kind of error?**