Introduction to PYTHON

List in Python

By: Atul Kumar Uttam

Assistant Professor

Computer Engineering & Applications Department,

GLA University, Mathura

List

- [ ]
- Mutable
- Can have collection of elements
- Elements can be homogeneous / heterogeneous

User Defined List

```
L=[int(x) for x in input().split()]
1 33  555  56666


print(L)
[1, 33, 555, 56666]
```

User Defined List

```
L = []
for i in range(10):
    a = int(input("enter a number. "))
    L.append(a)
```

Accessing List elements

```
>>>a=[10,20,30,40,45,65,66]

>>>print(a[2])
30

>>>print(a[0:3])
[10, 20, 30]

>>>print(a[2:4])
[30, 40]
```

```
>>>print(a[2:])
[30, 40, 45, 65, 66]

>>>print(a[1:-3])
[20, 30, 40]

>>>print(a[-6:-3])
[20, 30, 40]
```

LIST Updating

Using slicing

```
>>>a = [10, 20, 30, 40, 50, 60]
>>>a[6 : ] = [11, 22, 33]
>>>a
[10, 20, 30, 40, 50, 60, 11, 22, 33]

>>>a[len(a):] = [1000]
```

append(object**)** returns None

list**.append(**x**)**

- Add an item **x** to the end of the list.

```
>>>a=[11,22,33]
>>>a.append(10)
>>>a
[11, 22, 33, 10]
```

clear() returns None

list**.clear()**

- Remove all the items from the list.

```
>>>a=[11,22,33]
>>>a.clear()
>>>a
[]
```

copy() returns a list

list**.copy()**

– Returns a shallow copy of the list.

```
>>>a = [11,22,33]
>>>b = a.copy()
>>>a is b
False
```

count(value**)** returns int

list.count(value**)**

- return number of occurrences of value

```
>>>a= [2, 3, 4, 4, 5, 6, 4]
>>>a.count(4)
3
>>>a.count(11)
0
```

extend(iterable**)** return None

- **list.extend(**iterable**)**
  - extend list by appending elements from the iterable

```
>>>a=[11,22,33]
>>>a.extend([100,200,300])
>>>a
[11, 22, 33, 100, 200, 300]
```

```
>>>a.append([555,444,666])
>>>a
[11, 22, 33, 100, 200, 300, [555, 444, 666]]
```

index(value**, [**start**, [**stop**]])** returns integer

list.**index(**value, [start, [stop]]**)**

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

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

>>>a.index(20, 3)
4
```

insert(index, object**)**

list**.insert(**index**,** object**)**

– insert object before index

```
>>>a = [10, 20, 30, 40, 50]
>>>a.insert(2, 100)
>>>a
[10, 20, 100, 30, 40, 50]
```

pop([index]**)** returns item

list**.pop(**[index]**)**

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

```
>>>a= [11,22,33]
>>>a.pop()
33
>>>a
[11, 22]
```

remove(value**)** returns None

list**.remove(**value**)**

- remove first occurrence of value.
- Raises ValueError if the value is not present.

```
>>>a = [1, 2, 3, 4, 5, 6, 4]
>>>a.remove(4)
[1, 2, 3, 5, 6, 4]
```

reverse() returns None

list.**reverse()** -- reverse *IN PLACE*

```
>>>a = [10, 20, 30]
>>>a.reverse()
>>>a
[30, 20, 10]
```

sort(key= None**, reverse =** False**)**

list**.sort(key=** None**, reverse =** False**)**

- sorts the elements of a given list in a specific order - Ascending or Descending.
- Returns none

```
>>>a = ['e', 'a', 'u', 'o', 'i']
>>>a.sort()
>>>a
['a', 'e', 'i', 'o', 'u']
```

```
>>>a = ['e', 'a', 'u', 'o', 'i']
>>>a.sort(reverse=True)
>>>a
['u', 'o', 'i', 'e', 'a']
```

Use of key argument in sort()

```
>>>def fun(e):
        return e[1]



>>>a=[(1,2),(2,4), (6,3),(11,1),(10,2)]

>>>a.sort(key=fun)

>>>a
[(11, 1), (1, 2), (10, 2), (6, 3), (2, 4)]
```

- Python's in-built function **sorted()** for the same purpose.

>>>sorted(list, key = None**, reverse =** False**)**

Note:

- **sort()** doesn't return any value while, rather, it changes the original list.
- **sorted()** returns an iterable list, does not change original list

del removes the item at a specific index:

```
>>> a = [3, 2, 2, 1]
>>> del a[1]
[3, 2, 1]
```

del vs **pop()** vs **remove()**

- del
  - to remove an element by index,
- pop()
  - to remove it by index if you need the returned value, and
- remove()
  - to delete an element by value.
  - requires searching the list, and raises ValueError if no such value occurs in the list.

Python's Function over List

len(list)

- Gives the total length of the list.

max(list)

- Returns item from the list with max value.

min(list)

- Returns item from the list with min value.

list(seq)

- Converts a tuple into list.

append() vs **extend()**

append(): Appends object at the end.

```
>>>x = [1, 2, 3]
>>>x.append([4, 5])
>>>print (x)
[1, 2, 3, [4, 5]]
```

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

```
>>>x = [1, 2, 3]
>>>x.extend([4, 5])
>>>print (x)
[1, 2, 3, 4, 5]
```

Python List reverse()

- **The reverse() method reverses the elements of a given list.**

>>>list.reverse()

- The reverse() function
  - doesn't take any argument.
  - doesn't return any value.
  - only reverses the elements and **updates the original list**.

>>>os = ['Windows', 'macOS', 'Linux']

>>>os.reverse()

>>>print('Updated List:', os)

Updated List: ['Linux', 'macOS', 'Windows']

reversed()

- If you need to access individual elements of a list in reverse order, it's better to use reversed() method.

>>>os = ['Windows', 'macOS', 'Linux']

>>>list(reversed(os))

[‘Linux’, ‘macOS’ , ‘Windows’]

Python List copy()

- The copy() method **returns a shallow copy** of the list.
- A list can be copied with **=** operator.

>>>old_list = [1, 2, 3]

>>>new_list = old_list

- The problem with copying the list in this way is that if you modify the new_list, the old_list is also modified.

Deep Copy

```
>>>old_list = [1, 2, 3]
>>>new_list = old_list

>>>new_list.append('a')

>>>print('New List:', new_list )
New List: [1, 2, 3, 'a']

>>>print('Old List:', old_list )
Old List: [1, 2, 3, 'a']
```

Shallow Copy of a List Using Slicing

>>>A = ['cat', 0, 6.7]

>>>B = list[:]

>>>B.append('dog')

>>>A

['cat', 0, 6.7]

>>>B

New List: ['cat', 0, 6.7, 'dog']