Lists in Python

Lists in Python

- A list is a standard data type of python that can store a sequence of values the belonging to any type.
- The lists are depicted through brackets.
- List are mutable (i.e., modifiable) i.e., you can change eliminates of a list in place.

022 LISTS IN PYTHON

```
print([])
print([1, 2, 3])
print([1, 2.5, 3.7, 9])
print(['a', 'b', 'c'])
print(['a', 1, 'b', 3.5, 'zero'])
print(['One', 'Two', 'Three'])
```

Creating Lists

- **Creating Empty lists**
- The empty list is [ ].
- You can also create an empty list as:

```
L=list( )
```

Creating Lists

- **Creating List from Existing Sequences**
- You can also use the build-in list type object to create list from sequence as per the Syntax given below:

  L=list(<sequence>)

- where <sequence>can be any kind of sequence object including strings, Tuples, and lists.

023 LIST FROM EXISTING SEQUENCE

```
l=list("hello")
print(l)
t=('v', 'i', 'v', 'a')
print(list(t))
```

Creating Lists

- **Creating list from keyboard input**
- You can use this method of creating list of single characters or single digits via keyboard input.

024 LIST FROM KEYBOARD INPUT

```
l=list(input("Enter List Elements : "))
print(l)
```

Lists Vs. Strings

- List and strings have lots in common yet difference too.
- This section is going to shown Summarize the similarities and difference between list is and strings.

Lists in Python

- **Difference between list and strings**
- The list and strings are different following one another in following ways :
- **Storage :**
  - List are stored in memory exactly like strings,
  - expect that because sum of their objects are larger than others,
  - they store a different at each index instead of single character as in strings

Lists Vs. Strings

- **Mutability**
  - Strings are not mutable, why list are.
  - You can't change individual eliminates of a strings in place,
  - But list is alone you to do so.
  - That is, following statements is fully valid for lists (though not for strings) :

    L[i]=<eliminate>

025 MUTABILITY IN LIST

```
a=['a', 'e', 'i', 'o', 'u']
a[0]='A'
print(a)
b=('hello')
b[0]='H'
print(b)
```

Notice : ***It changes the element in place;('a' changed to 'A'), no new list created – Because list are MUTABLE***

List Operands

- In this section, we shall talk about most common list operations:
  - Traversing a list
  - Joining lists
  - Repeating and Replicating Lists
  - Slicing the lists

List Operands

- **Traversing a list**
- Traversing our list means assessing and processing each elements of it.
- The for loop make it easy to traverse or loop over the items in the list, as per following syntax:

```
for <item> in <List>:
    process each item here
```

026 TRAVERSING A LIST

```
l=['P', 'y', 't', 'h', 'o', 'n']
for a in l:
    print(a)
```

List Operands

- **Joining lists**
- The concatenation operator +,
- when used with two lists, joins to list and returns the concatenated list.
- Consider the example given below :
- The + operator when used with list is required that both the offer and must be of list types.

```
>>> lst1 = [ 1, 4, 9]
>>> lst2 = [6, 12, 20 ]
>>> lst1 + lst2
[1, 4, 9, 6, 12, 20]
```

027 JOINING TWO LISTS

```
l1=[1, 2, 3]

l2=[4, 5, 6]

print(l1+l2)
```

List Operands

- **Repeating and Replicating Lists**
- Like strings, you can use * operator to replicate a list specified numbers of times,
- e.g. considering the same list lst1 = [1,2,3]

```
<<< lst1 * 3
[1, 4, 9, 1, 4, 9, 1, 4, 9]
```

- Like strings, you can only use and integer with a * operator when trying to replicate a list.

028 REPEATING OR REPLICATING LISTS

```
l=[1, 2, 3, 4, 5]
print(l*3)
```

List Operands

- **Slicing the lists**
- List slices, like string slices are the sub part of a list extracted out.
- You can use indexes of list elements to create list slices as per following format :

```
sep=l[start : stop]
```

029 SLICING THE LISTS

```
lst=[2, 4, 6, 8, 10, 12 ,14 ,16, 18, 20]
seq=lst[3:-3]
print(seq)
seq=lst[3:20]
print(seq)
seq=lst[-10:7]
print(seq)
```

List Operands

- Lists also support slice steps too.
- That is, if you want to extract, not consecutive but every other element of the list, there is a way out- the Slice steps.
- The slides steps are used as per following format :

sep = l [start : stop : step]

030 SLICING PART 2

```
l=[2, 4, 6, 8, 10 ,12, 14 ,18 ,20]
print(l[0:10:2])
print(l[2:10:3])
```

List Operands

- **Using slices for list modification**
- You can use slices to overwrite one or more list elements with one or more other elements.

NOTE : Like strings, in list slices, you can give start and stop beyond limits of list and it won't raise index error, tractor it will return the elements felling between specified boundaries.

031 USING SLICES FOR LIST MODIFICATION

```
l=["one", "two", "THREE"]
l[0:2]=[0,1]
print(l)
l[0:2]="a"
print(l)
```

List Manipulation

- You can perform various operations on lists like :

  Appending

  Updating

  Deleting etc.

List Manipulation

- **Appending elements to a list**
- You can also add items to an existing sequence.
- The appended ( ) method ads a single item to the end of the list.
- It can be done is per following format:

    I. appended (item)

032 APPENDING ELEMENTS TO A LIST

```
l=[10, 12, 14]

l.append(16)

print(l)
```

List Manipulation

- **Updating elements to a list**
- To update or change in elements of the list in place,
- you just have to assign new value to the element's index in list as per syntax :

L[index] = <new value>

033 UPDATING ELEMENTS IN LIST

```
l1 = [10,12,14,16]
print(l1)
l1[2] = 24
print(l1)
```

List Manipulation

- **Deleting elements from a list**
- You can also remove items from lists.
- The delete statement can be used to remove an individual item,
- or to remove all items identified by a slice.

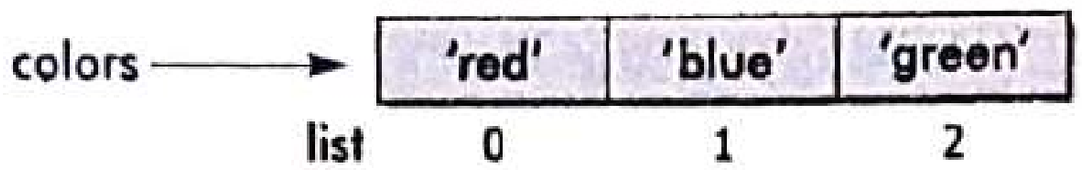
034 DELETING ELEMENTS FROM LIST

```
l1 = [2,4,6,8,10,12,14,16,18,20]
print(l1)
del l1 [4]
print(l1)
l2 = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
print(l2)
del l2[10:15]
print(l2)
```

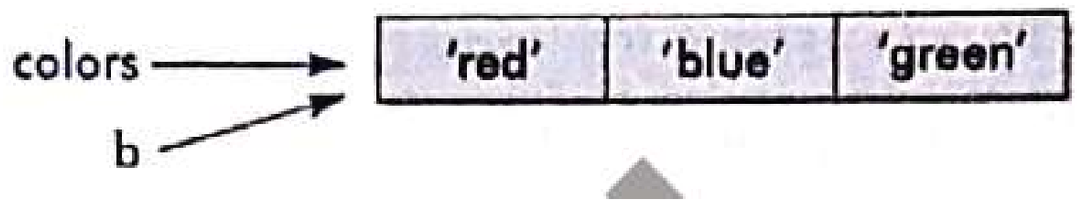
Making True Copy of a List

- Assignment with an assignment operator (=) on lists does not make a copy.
- Instead, assignment makes the two variables point to the one list in memory (called shallow copy).

```
colors = ['red', 'blue', 'green']
```




```
b = colors      ## Does not copy the list
```

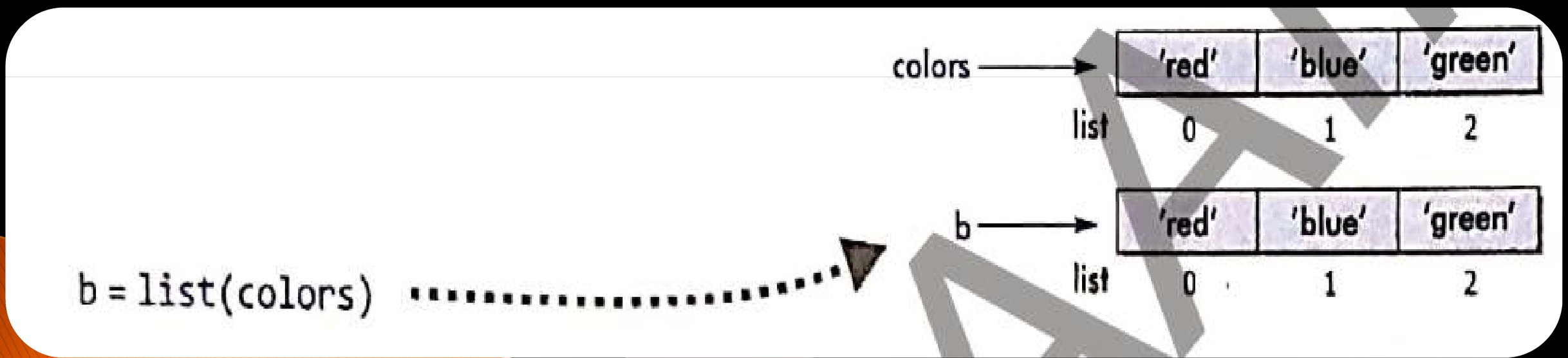


035 MAKING TRUE COPY OF A LIST

```
colors = ['red', 'green', 'blue', 'yellow', 'magenta', 'cyan']
print(colors)
b = colors
print(b)
```

Making True Copy of a List

- To make be true copy of list colors
- i.e. independence list identical to list colors you should create copy of list as follows :
- Now colors and b are separated lists (deep copy).



036 MAKING A TRUE COPY OF A LIST

```
colors = ['red', 'green', 'blue', 'yellow', 'magenta', 'cyan']
print(colors)
b = list (colors)
print(b)
```

List Functions

- Python offers many build in functions and methods for list manipulation.
- These can be applied to list as per following syntax :

  <List object> . <method name> ( )

List Functions

- Following are List Functions :

1. Index Method
2. Append Method
3. Extend Method
4. Insert Method
5. Pop Method
6. Remove Method
7. Clear Method
8. Count Method
9. Reverse Method
10. Sort Method

List Functions

1. The Index Method

- This function returns the index of first matched item from the list.

List. Index (<item>)

- For example, for a list L1 = [13,18,11,16,18,14],
- However, if the given item is not in the list, Raise expectation value error

```
>>> L1.index (18)
1
```

returns the index of first value 18, even if there is another value 18 at index 4.

037 LIST INDEX METHOD

```
colors = ['red', 'green', 'blue', 'yellow', 'magenta', 'cyan']
print (colors.index('green'))


l1 = 2,4,6,8,10,12,14,16,18,20
print (l1.index(8))
```

List Functions

2. The Append Method

- The append ( ) method adds an item to the end of the list.
- It works as per following it works as per following syntax:

List.append (<item>)

- takes expectedly one eliminate and returns no value

List Functions

- For example, to add a new item "yellow" to a list containing colors, you may write :

```
>>> colours = ['red', 'green', 'blue']
>>> colours.append('yellow')
>>> colours
['red', 'green', 'blue', 'yellow']
```

See the item got added at the end of the list

- The append () does not return the new list, just modifies the original.

038 THE APPEND METHOD

```
colors = ['red', 'green', 'blue', 'yellow', 'magenta', 'cyan']
colors.append("hot pink")
print(colors)
```

List Functions

3. The Extend Method

- The extend () method is also used for adding multiple elements (give one in the form of a list) to a list.
- The extend () function works as per following format:

  List. Extend (<list>)

- takes exactly one element (a list type) and returns no value

List Functions

- That is extend () takes a list as an argument and appends all of the elements of the argument list to the list object one which extend () is applied.
- Consider the following example:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
>>> t2
['d', 'e']
```

Extend the list t1, by adding all elements of t2

See the elements of list t2 are added at the end of list t1

But list t2 remains unchanged.

039 THE EXTEND METHOD

```
colors1 = ["orange","hot pink","magneta","cyan"]
print (colors1)
colors2 = ["red","blue","green","yellow","purple","pink","brown"]
print(colors2)
colors1.extend(colors2)
print(colors1)
```

List Functions

4. The Insert Method

- The insert () function inserts and items at given position.
- It is used as per following syntax:

List. Insert (<pos> , <item>)

- takes to arguments and returns no value.

040 INSERT METHOD

```
vowels = ["a","e","o","u"]
print(vowels)
vowels.insert( __index: 1, __object: "i")
print(vowels)
```

List Functions

5. The Pop Method

- The pop () is used to remove the item from the list.
- It is used as per following syntax:

  list.pop (<index>)

- Takes One Optional argument and returns a value – the item being deleted

041 POP METHOD

```
vowels = ["a","e","p","i","o","u"]
print(vowels)
vowels.pop (2)
print(vowels)
```

List Functions

6. The Remove Method

- The remove ( ) removes the first occurrence of given item from the list.
- It is used as per following format :

  List.remove (<value>)

- Takes one essential argument and does not return anything.
- The remove ( ) well report in error if these is no such item in the list.

042 REMOVE METHOD

```
vowels = ["a","e","i","o","u","i"]
print(vowels)
vowels.remove("i",)
print(vowels)
vowels.remove("i",)
print(vowels)
```

List Functions

7. The Clear Method

- This method remove all the items from the list and the list becomes empty list after this function.
- This function returns nothing.
- It is used as per following format.

List.(clear)

043 THE CLEAR METHD

```
numbers = [1,2,3,4,5,6,7,8,9]
print(numbers)
numbers.clear()
print(numbers)
```

List Functions

8. The Count Method

- This function returns the count of the item that you passed as argument.
- If the given item is not in the list, it returns zero.
- It is used as per following format:

List.count (<item>)

044 THE COUNT METHOD

```
numbers = [2,5,8,9,5,8,5,8,3,]
print(numbers)
print(numbers.count(5))
```

List Functions

9. The Reverse Method

- The reverse () reverses the items of the list.
- This is a done "in place", i.e., it does not create a new list.
- The syntax to use reverse method is:

  List.reverse ( )

- Takes no argument, returns no list ;
- reverses list in place and does not return anything.

045 THE REVERSE METHOD

```
numbers = [1,2,3,4,5,6,7,8,9,10]
print(numbers)
numbers.reverse()
print(numbers)
```

List Functions

10. The Sort Method

- The sort ( ) function sorts the items of the list, by default in increasing order. This is done “in place”,
- i.e., it does not create a new list.
- It is used as per following syntax:

List.sort ( )

046 THE SORT METHOD

```
numbers = [1,3,5,4,6,2,7,9,8,10]
numbers.sort()
print(numbers)
```