TOPIC: Collections (Lists, Tuples, set, Dictionaries)

COURSE: Machine learning using Python Programming

PPT SL.NO.: 1 of 31

VERSION: 01

LAST UPDATED ON: 13/10/2020

NIEL

Presentation By: Saurabh

Lists

Content

- Lists
- Indexing of List
- Accessing Values in Lists
- Updating Lists
- Delete List Elements
- Basic List Operations
- Built-in List Functions
- Built-in List Methods
- Using List as Stacks
- Using List as Queue
- Mutable vs Immutable

Lists

- The list is the most versatile data type available in Python, which can be written as a list of comma-separated values (items) between square brackets [ ]. Important thing about a list is that the items in a list need not be of the same data type.

- Creating a list is as simple as putting different comma-separated values between square brackets.

Indexing of List

- For accessing an element of the list, indexing is used.
- **Syntax is:** **Variable name [index]**

  Note: Variable name is name of the list
- Index here, has to be an integer value which can be positive or negative.
- Positive value of index means counting forward from beginning of the list and negative value means counting backward from end of the list.
- **Example:**

  I N D I A

  0 1 2 3 4

  -5 -4 -3 -2 -1

Index value	Element of the list
0, -size	1st
1, -size +1	2nd
. .	. .
size -1, -1	last

Lists

For example −

- L1 = [1, 2, 3, 4] # list of 4 integer elements.
- L2 = [“Delhi”, “Chennai”, “Mumbai”] #list of 3 string elements.
- L3 = [ ] # empty list i.e. list with no element
- L4 = [“abc”, 10, 20] # list with different types of elements
- L5 = [1, 2, [6, 7, 8], 3] # A list containing another list known as nested list

Lists

Example: Program for print values of list before change and after modification.

```
L1 = [1, 2, 3, 4]
print (L1)        # values of list before change
L1 [2] = 5
print (L1)        # modified list
[1, 2, 5, 4]
```

OUTPUT:

[1, 2, 3, 4]
[1, 2, 5, 4]

List

```
my_list= ("hello")
print(type(my_list))        # <class 'str'>

# Creating a list having one element
my_list = ["hello","india"]
print(type(my_list))        # <class 'list'>
```

OUTPUT:

<class 'str'>
<class 'list'>

Accessing Values in Lists

- To access values in lists, use the square brackets for slicing along with the index to obtain value available at that index.

For example −

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

OUTPUT:

```
list1[0]: physics
list2[1:5]: [2, 3, 4, 5]
```

Updating Lists

- You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

For example −

```
list = ['physics', 'chemistry', 1997, 2000]
print("Value available at index 2 : " , list[2])

list[2] = 2001
print("New value available at index 2 : ", list[2])
```

OUTPUT:
Value available at index 2 : 1997
New value available at index 2 : 2001

Delete List Elements

- To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.
- **For example −**

```
list1 = ['physics', 'chemistry', 1997, 2000];
print (list1)
del (list1[2])
print ("After deleting value at index 2 : ")
print (list1)
```

OUTPUT:
['physics', 'chemistry', 1997, 2000]

After deleting value at index 2 :
['physics', 'chemistry', 2000]

Basic List Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1,2,3] : print (x,end = ' ')	1 2 3	Iteration

Built-in List Functions

- **Len(list) –** Gives the total length of the list.
- **Syntax:** **len(list)**

Example:

```
list1 = ['physics', 1997, 2000]
print (len(list1))
list2 = [‘Ravi', 458]
print (len(list2))
```

OUTPUT:
3
2

Built-in List Functions

- **max(list)** – Returns item from the list with maximum value.
- **Syntax:** **max(list)**

Example:

```
list1 = ['xyz', ‘zara', 'abc']
list2 = [456, 700, 200]
print ("Max value element : ", max(list1))
print ("Max value element : ", max(list2))
```

OUTPUT:

```
Max value element :  zara
Max value element :  700
```

Built-in List Functions

- **min(list)** – Returns item from the list with minimum value.
- **Syntax:** **min(list)**

Example:

```
list1 = ['xyz', 'zara', 'abc']
list2 = [456, 700, 200]
print ("Min value element : ", min(list1))
print ("Min value element : ", min(list2))
```

OUTPUT:

```
Max value element :  abc
Max value element :  200
```

Built-in List Functions

- **list(seq)** – Returns a tuple into a list.
- **Syntax:** **list(seq)**

Example:

```
Tuple = (123, 'xyz', 'zara', 'abc')
List = list(Tuple)
print ("List elements : ", List)
```

OUTPUT:

List elements : [123, 'xyz', 'zara', 'abc']

Built-in List Methods

- **Python List append() Method:** Python list method **append()** appends a passed *obj* into the existing list.

- **Syntax:** **list.append(obj)**

Example:

```
List = [123, 'xyz', 'zara', 'abc']
List.append( 2009 )
print ("Updated List : ", List)
```

OUTPUT:
Updated List : [123, 'xyz', 'zara', 'abc', 2009]

Built-in List Methods

- **Python List count() Method:** Python list method **count()** returns count of how many times *obj* occurs in list.
- **Syntax:** **list.count(obj)**

Example:

```
List = [123, 'xyz', 'zara', 'abc', 123, 123]
print ("Count for 123 : ", List.count(123))
print ("Count for zara : ", List.count('zara'))
```

OUTPUT:

```
Count for 123 :  3
Count for zara :  1
```

Built-in List Methods

- **Python List extend() Method:** Python list method **extend()** appends the contents of *seq* to list.
- **Syntax:** **list.extend(seq)**

Example:

```
List1 = [123, 'xyz', 'zara', 'abc', 123]
List2 = [2009, 'manni']
List1.extend(List2)
print ("Extended List : ", List1)
```

OUTPUT

```
Extended List :  [123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']
```

Built-in List Methods

- **Python List index() Method:**

  Python list method **index()** returns the lowest index in list that *obj* appears.

- **Syntax:** **list.index(obj)**

Example:

```
List = [123, 'xyz', 'zara', 'abc‘, 123, 123]
print ("Index for xyz : ", List.index( 'xyz' ))
print ("Index for zara : ", List.index( 'zara' ))
```

OUTPUT:

```
Index for xyz :  1
Index for zara :  2
```

Built-in List Methods

- **Python List insert() Method:**

  Python list method **insert()** inserts object *obj* into list at offset *index*.

Syntax: **list.insert(index, obj)**

Example:

```
List = [123, 'xyz', 'zara', 'abc']
List.insert( 3, 2009)
print ("Final List : ", List)
```

OUTPUT:
Final List : [123, 'xyz', 'zara', 2009, 'abc']

Built-in List Methods

- **Python List pop() Method :** Python list method **pop()** removes and returns last object or *obj* from the list.
- **Syntax:** **list.pop(obj = list[-1])**

Example:

```
List = [123, 'xyz', 'zara', 'abc']
print (" List   : ", List.pop(3))
print("List =", List)
print (" List  : ", List.pop(2))
print("List =", List)
```

OUTPUT:

```
List   :  abc
List = [123, 'xyz', 'zara']
List  :  zara
List = [123, 'xyz']
```

Built-in List Methods

- **Python List remove() Method**: Python list method **remove()** searches for the given element in the list and removes the first matching element.
- **Syntax:** **list.remove(obj)**

Example:

```
list = [123, 'xyz', 'zara', 'abc', 'xyz']
list.remove('xyz')
print ("List : ", list)
list.remove('abc')
print ("List : ", list)
```

OUTPUT:

```
List :  [123, 'zara', 'abc', 'xyz']
List :  [123, 'zara', 'xyz']
```

Built-in List Methods

- **Python List.reverse() Method**: Python list method **reverse()**, Revserve the objects in the list.
- **Syntax:** **list.reverse(obj)**

Example:

```
list = [123, 'zara', 'abc', 'xyz']
list.reverse()
print ("List : ", list)
```

OUTPUT:
List : ['xyz', 'abc', 'zara', 123]

Built-in List Methods

- **Python list.clear() Method**: Python list method **list.clear()**, Clear the objects in the list.
- **Syntax:** **list.clear(obj)**

Example:

```
list = [123, 'zara', 'abc', 'xyz']
print ("List before clearing : ", list)
list.clear()
print ("List after clearing : ", list)
```

OUTPUT:
List before clearing : [123, 'zara', 'abc', 'xyz']
List after clearing : []

Built-in List Methods

- **Python list.copy() Method**: Python list method **list.copy()**, Copy the objects of the list.
- **Syntax:** **list.copy(obj)**

Example:

```
list1 = [123, 'zara', 'abc', 'xyz']
print (“Object of list1 : ", list1)
list2= []
print("before copying list2:", list2)
list2=list1.copy()
print ("List1 copy into list2: ", list2)
```

OUTPUT:

Object of list1 : [123, 'zara', 'abc', 'xyz']

before copying list2: [ ]

List1 copy into list2: [123, 'zara', 'abc', 'xyz']

Using List as Stacks

- List can be use as a stack (Last In- First Out). Stack is a data structure where the last element added is the first retrieved (or popup).
- To add an item to the top of the stack, use **append()**.
- To rerieve an item from the top of the stack, use **pop()** without an explicit index.

Using List as Stacks

Eaxmple:

```
Stack = [10, 20, 30, 40, 50]
Stack.append(60)
print("Stack after appending:", Stack)
Stack.pop()
print("Stack after poping:", Stack)
```

OUTPUT:

```
Stack after appending: [10, 20, 30, 40, 50, 60]

Stack after poping: [10, 20, 30, 40, 50]
```

Using List as Queues

- List can be use as a Queue (First In- First Out) data structure. But lists are not efficient for this purpose. While appends and pops from the end of list are fast, doing inserts or pops from the beginning of a list is slow since all of the other elements have to be shifted by one.
- To implement a queue, Python provides a module called collections in which a method called **deque** is designed to have fast appends and pops from both ends.

Using List as Queue

Example:

```
from collections import deque
queue = deque(["apple","orange","pear"])
queue.append("cerry")
queue.append("grapes")
queue.popleft()
queue.popleft()
print(queue)
```

OUTPUT:

```
(['pear', 'cerry', 'grapes'])
```

Mutable vs Immutable

- Every variable in python holds an instance of an object. There are two types of objects in python i.e. Mutable and Immutable objects. Whenever an object is instantiated, it is assigned a unique object id. The type of the object is defined at the runtime and it can't be changed afterwards. However, it's state can be changed if it is a mutable object.
- To summaries the difference, mutable objects can change their state or contents and immutable objects can't change their state or content.
- Immutable Objects **:** These are of in-built types like int, float, bool, string, tuple. In simple words, an immutable object can't be changed after it is created.
- Mutable Objects **:** These are of type list, dict, set . Custom classes are generally mutable

Thank you!

If you have any query, please contact:

Saurabh