Python Programming

Session 5

Seyyed Ali Shohadaalhosseini

TOPICS

Python Fundamentals and Programming

- **List**

Multiple value in one variable

What if we wanted a variable which could have multiple value

We can use:

LIST

Tuple

Dictionary

Set

List

Lists are used to store multiple items in a single variable.

Lists are one of the 4 built-in data types in python used to store collections of data.

Lists are created using square brackets.

Example:

```
fruits = ['banana', 'apple', 'cucumbers']
```

List features

List items are

1. **Ordered**
2. **Changeable**, and
3. **Allow duplicate values**.

- List items are indexed, the first item has index [0], the second item has index [1] etc.

List features - Ordered

When we say that lists are ordered, it means that the items have a define order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow duplicates

Since lists are indexed, lists can have items with the same value.

List Length

To determine how many items a list has, use the *len()* method.

List Items - Data Types

List items can be any data type. And also a list can contain different data types, for example a list with strings, integers and boolean values.

Your Turn ? What is the data type of a list ?

Your Turn ? What is list() Constructor ?

Access List Items

List items are indexed so we can access each items by its index. The index rules is such the rules we saw before.

Change list items

If you want to change the lists items, you can choose an item by its index and then assign a new value to that item.

Even if you wanted to change a range of items for example changing the items from the index 1 to 3, you choose these items by slicing rules we learnt before and then you can assign a value or multiple value in a list to that Items.

```
fruits = ['banana', 'apple', 'cucumbers', 'Orange']
fruits[1:3] = ['new item1', 'new item2']
print((fruits))
```

```
['banana', 'new item1', 'new item2', 'Orange']
```

Add Items to List

1. Append Items
2. Insert Items
3. Extend List

Append Items

To add an item to the end of the list, use the append() method.

Code:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Output: ?

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the *insert()* method.

This method inserts an item at the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
```

Output: ?

Extend List

To append only the elements from another list to the current list, use the *extend()* method.

Notice that elements will add to the end of list. You can use this method to add any iterable object(like: Tuples, sets, dictionaries) too.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Output: ?

List Important Exercise #1

Research about below method of the list:

1. remove()
2. pop()
3. del
4. clear()

More information in you exercise note.

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

For example you can do the following in a shorter form:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]
```

Sort Lists

We can sort list’s elements.

Sort List Alphanumerically

List object have a *sort()* method that will sort the list alphanumerically and ascending(lower to higher, zero to hero, a to z) by default.

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
```

If you want to sort descending(opposite of the ascending) you must initialize reverse to True.

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
```

Copy a List

We can not copy a list only by assigning it to a new list (list2 = list1), Implementing in this way initialize list2 a reference to list1 and that is mean any changes made in list1 will automatically also be made in list2.

To solve this problem we use *copy()* method for that:

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
```

Notice that *copy()* method is one level deep and it does not work good for the list which contains lists(nested list), to do the JOB GREAT we can use *deepcopy()* from copy module.

DeepCopy

Complete full copy from each element of source list to destination list.

```
l1 = [112, 2, 314, 245, 45]
l2 = deepcopy(l1)
li = []
l1 = [222]
```