Programming Fundamental

Lab#09

List

Contents

Overview

Python has two other sequence structures:

1. tuples
2. lists

These contain zero or more elements. Unlike strings, the elements can be of different types. In fact, each element can be any Python object. This lets you create structures as deep and complex as you like.

Why does Python contain both lists and tuples? Tuples are immutable; when you assign elements to a tuple, they're baked in the cake and can't be changed. Lists are mutable, meaning you can insert and delete elements

Lists

Lists are good for keeping track of things by their order, especially when the order and contents might change. Unlike strings, lists are mutable. You can change a list in-place, add new elements, and delete or overwrite existing elements. The same value can occur more than once in a list

CREATE

A list is made from zero or more elements, separated by commas, and surrounded by square brackets:

```
>>> empty_list = [ ]
>>> weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
>>> big_birds = ['emu', 'ostrich', 'cassowary']
>>> first_names = ['Graham', 'John', 'Terry', 'Terry', 'Michael']
```

You can also make an empty list with the list() function:

```
>>> another_empty_list = list()
>>> another_empty_list
[]
```

Python's list() function converts other data types to lists. The following example converts a string to a list of one-character strings:

```
>>> list('cat')
['c', 'a', 't']
```

use split() to chop a string into a list by some separator string:

```
>>> birthday = '1/6/1952'
>>> birthday.split('/')
['1', '6', '1952']
```

```
>>> splitme = 'a/b//c/d///e'
>>> splitme.split('/')
['a', 'b', '', 'c', 'd', '', '', 'e']
```

LISTS OF LISTS

Lists can contain elements of different types, including other lists, as illustrated here:

```
>>> small_birds = ['hummingbird', 'finch']
>>> extinct_birds = ['dodo', 'passenger pigeon', 'Norwegian Blue']
>>> carol_birds = [3, 'French hens', 2, 'turtledoves']
>>> all_birds = [small_birds, extinct_birds, 'macaw', carol_birds]
```

So what does `all_birds`, a list of lists, look like?

```
>>> all_birds
[['hummingbird', 'finch'], ['dodo', 'passenger pigeon', 'Norwegian Blue'], 'macaw',
[3, 'French hens', 2, 'turtledoves']]
```

It’s the second item we specified, extinct_birds. If we want the first item of extinct_birds, we can extract it from all_birds by specifying two indexes:

```
>>> all_birds[1][0]
'dodo'
```

Accessing an item

Index

1. As with strings, you can extract a single value from a list by specifying its offset:
2. As with strings, negative indexes count backward from the end:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0]
'Groucho'
>>> marxes[1]
'Chico'
>>> marxes[2]
'Harpo'
```

```
>>> marxes[-1]
'Harpo'
>>> marxes[-2]
'Chico'
>>> marxes[-3]
'Groucho'
>>>
```

Slicing

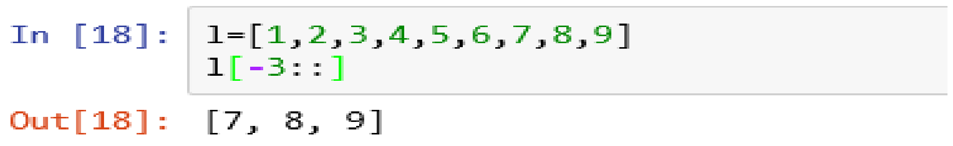
You can extract a subsequence of a list by using a slice:

```
>>> marxes = ['Groucho', 'Chico,' 'Harpo']
>>> marxes[0:2]
['Groucho', 'Chico']
```

A slice of a list is also a list.
As with strings, slices can step by values other than one. The next example starts at the beginning and goes right by 2:

```
>>> marxes[::2]
['Groucho', 'Harpo']
```

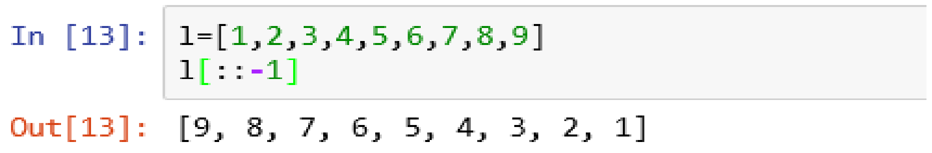
Using Slice operator you can print the last three values



```
In [18]: l=[1,2,3,4,5,6,7,8,9]
         l[-3::]

Out[18]: [7, 8, 9]
```

Using Slice operator you can print list in reverse order



```
In [13]: l=[1,2,3,4,5,6,7,8,9]
         l[::-1]

Out[13]: [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Insert an item

The traditional way of adding items to a list is to **append ()** them one by one to the end. \
Let's suppose in the previous examples, we forgot Zeppo, but that's all right because the list is mutable, so we can add him now:

```
>>> marxes.append('Zeppo')
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo']
```

Update an item

Just as you can get the value of a list item by its offset, you can change it:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[2] = 'Wanda'
>>> marxes
['Groucho', 'Chico', 'Wanda']
```

Again, the list offset needs to be a valid one for this list.
You can't change a character in a string in this way, because strings are immutable. Lists are mutable. You can change how many items a list contains, and the items themselves

Delete an item

When you delete an item by its position in the list, the items that follow it move back to take the deleted item's space, and the list's length decreases by one. If we delete 'Harpo' from the last version of the marxes list, we get this as a result:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Gummo', 'Zeppo']
>>> marxes[2]
'Harpo'
>>> del marxes[2]
>>> marxes
['Groucho', 'Chico', 'Gummo', 'Zeppo']
>>> marxes[2]
'Gummo'
```

Combining two list (extend() & +=)

You can merge one list into another by using extend(). Suppose that a well-meaning person gave us a new list of Marxes called others, and we'd like to merge them into the main marxes list:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes.extend(others)
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

Alternatively, you can use +=:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes += others
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

You can also get an index of the item in list using **index ()**

For example marxes.index('Chico')

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> marxes.index('Chico')
1
```

Basic function to deal with list

1. Check whether a value is in list or not?

The Pythonic way to check for the existence of a value in a list is using in:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> 'Groucho' in marxes
True
>>> 'Bob' in marxes
False
```

2. Count occurrence of a value in a list

To count how many times a particular value occurs in a list, use count():

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.count('Harpo')
1
>>> marxes.count('Bob')
0
```

3. Sort()

```
>>> marxes.sort()
>>> marxes
['Chico', 'Groucho', 'Harpo']
```

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 4.0]
```

The default sort order is ascending, but you can add the argument `reverse=True` to set it to descending:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort(reverse=True)
>>> numbers
[4.0, 3, 2, 1]
```

4. Len()

len() returns the number of items in a list:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> len(marxes)
3
```