Section 6

Lists

Mohammed Asir Shahid

2021-08-02

Contents

1 The List Data type

A list is a data type that contains multiple ordered items. Lists begin and end with square brackets and are separated by commas. They can be assigned to variables just like any other value.

```
["cat", "bat", "rat", "elephant"]

spam=["cat", "bat", "rat", "elephant"]

print(spam)
```

```
['cat', 'bat', 'rat', 'elephant']
```

We can use indices to access items within the list. These indices also begin and end with square brackets.

```
["cat", "bat", "rat", "elephant"]

spam=["cat", "bat", "rat", "elephant"]

print(spam)

print(spam[0])
print(spam[1])
print(spam[2])
print(spam[3])
```

```
['cat', 'bat', 'rat', 'elephant']
cat
bat
rat
elephant
```

The items inside of a list can be of any data type, including other lists. In those cases, we can use 2 indices in order to find the items in the list inside of the list. We can also use negative values for the index in order to start counting from the end.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(spam)

print(spam[0][0])
print(spam[1][0])
print(spam[1][4])
print(spammed[-1])
print(spammed[-2])
```

```
[['cat', 'bat'], [10, 20, 30, 40, 50]]
cat
10
```

```
50
elephant
rat
```

We also have slices which can give us several values from inside of a list. This works similar to the range function. A slice of 1 3 starts at index 1 and goes up to, but does not include the value at 3.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(spam)

print(spam[0][0])
print(spam[1][0])
print(spam[1][4])
print(spammed[0:2])
```

```
[['cat', 'bat'], [10, 20, 30, 40, 50]]
cat
10
50
['cat', 'bat']
```

We can also change the values of a list.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(spammed)

spammed[0]="Hello"

print(spammed)

spammed[1:3]=["Good", "Bye"]
```

```
print(spammed)
```

```
['cat', 'bat', 'rat', 'elephant']
['Hello', 'bat', 'rat', 'elephant']
['Hello', 'Good', 'Bye', 'elephant']
```

We have some shortcuts when it comes to lists. Leaving out the first index is the same as 0, or the beginning of the list. Leaving out the second index is the same as using the length of the list which will slice to the end of the list.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(spammed[:2])
print(spammed[2:])
```

```
['cat', 'bat']
['rat', 'elephant']
```

Del statements can delete values from the list.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(spammed)
del spammed[2]

print(spammed)
```

```
['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'elephant']
```

We can get the number of items in a list using the len function. We can also do list concatenation similar to string concatenation. We can also do list replication.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(len(spammed))

print(spammed*3)

print(spammed+spam)
```

```
4
['cat', 'bat', 'rat', 'elephant', 'cat', 'bat', 'rat', 'elephant', 'cat', 'bat', 'rat'
['cat', 'bat', 'rat', 'elephant', ['cat', 'bat'], [10, 20, 30, 40, 50]]
```

There is also a list function that converts our values into a list, similar to the int or str functions.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print(list("Hello"))
```

```
['H', 'e', 'l', 'l', 'o']
```

We can use the in or not in operators with lists.

```
spammed=["cat", "bat", "rat", "elephant"]

spam=[["cat", "bat"], [10,20,30,40,50]]

print("cat" in spammed)
print("dog" in spammed)
print("dog" not in spammed)
```

```
True
False
True
```

2 For Loops with Lists, Multiple Assignment, and Augmented Operators

For Loops can be used to execute a block of code a certain number of times.

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

Python interprets the range object similarly to a list of integers. For example, the above code can also be written as follows:

```
for i in [0,1,2,3]:
    print(i)
```

```
0
1
2
3
```

We can also use the list function in order to convert the range object into a list.

```
print(list(range(4)))
```

```
[0, 1, 2, 3]
```

So if you want to make a long list of integers that follows a pattern, using the range function can be better than writing it out.

```
print(list(range(0,100,2)))
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44
```

One way to use range is to use a for loop over the range of the length of some list.

```
supplies=["pens","staplers","flame-throwers","binders"]

for item in range(len(supplies)):
    print("Index {} in supplies is {}".format(item,supplies[item]))
```

```
Index 0 in supplies is pens
Index 1 in supplies is staplers
Index 2 in supplies is flame-throwers
Index 3 in supplies is binders
```

The nice thing about the above for loop is that it'll work regardless of the length of the supplies list.

```
cat=["fat","orange","loud"]

size=cat[0]

color=cat[1]

disposition=cat[2]
```

```
# Instead of 3 lines of code for something like this, we can assign mutiple variables a

size, color, disposition = cat
print(size)
print(color)
print(disposition)

# We could also have mutiple variables on the right side.

size, color, disposition = "skinny","black","quiet"
print(size)
print(color)
print(disposition)
```

```
fat
orange
loud
skinny
black
quiet
```

We can use the multiple assignment for swapping variables.

```
a="AAA"
b="BBB"

a,b=b,a

print(a,b)
```

```
BBB AAA
```

Python also lets us use Augmented Assignment Operators.

```
spam=42
```

```
spam=spam+1
spam+=1
```

Instead of doing “spam=spam+1”, we can do “spam+=1”. This also applies to other operators such as plus, minus, multiplication, division, and modulus.

3 List methods

A method is the same thing as a function, except it is attached to a certain value.

All list values have a method called index that gives you the index of an item inside of a list. You can not call a method by itself, so the index method must be called on a particular list.

```
spam=["hello","hi","howdy","heyas"]

print(spam.index("hello"))
print(spam.index("heyas"))
```

```
0
3
```

The method name comes after the value/variable and then a dot.

Each data type has different types of methods.

If the value does not exist inside the list, then an exception is raised. If there are more than one value in the list, then the index for the first one it was seen will be given.

```
spam=["Zophie","Pooka","Fat-tail","Pooka"]

print(spam.index("Pooka"))
```

```
1
```

We can use the append() and insert() list methods in order to add new values to a list.

```
spam=["cat","dog","bat"]

print(spam)

spam.append("moose")

print(spam)

spam.insert(1,"chicken")

print(spam)
```

```
['cat', 'dog', 'bat']
['cat', 'dog', 'bat', 'moose']
['cat', 'chicken', 'dog', 'bat', 'moose']
```

The list is modified in place, meaning we don't need to set spam equal to spam.append()

These methods belong to a single data type. The append and insert methods are list methods and can only be called on list values.

There is also a remove method for lists that can help you remove items.

```
spam=["cat","bat","rat","elephant","bat"]

print(spam)

spam.remove("bat")

print(spam)
```

```
['cat', 'bat', 'rat', 'elephant', 'bat']
['cat', 'rat', 'elephant', 'bat']
```

This differs from the del statement because it allows you to identify the value that you want to remove, not just the index of the value you want to remove.

If a value appears multiple times, the remove method will only remove the first occurrence.

A list with number or string values can be sorted using the sort method. However, this only works when the list contains only string values or only number values. Not when it contains a mix of the two.

```
spam=[2,5,3.14,1,-7]

print(spam)

spam.sort()

print(spam)

spam=["ants","cats","dogs","badgers","elephants"]

print(spam)

spam.sort()

print(spam)
```

```
[2, 5, 3.14, 1, -7]
[-7, 1, 2, 3.14, 5]
['ants', 'cats', 'dogs', 'badgers', 'elephants']
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

The sort method uses ASCII order, not regular alphabetical order. This means that uppercase characters come before lowercase characters, meaning that uppercase Z comes before lowercase a. We can pass an argument into the method in order to change this.

```
spam=["Alice","Bob","ants","badgers","Carol","cats"]

print(spam)

spam.sort()
```

```
print(spam)

spam=["A","a","Z","z"]

print(spam)

spam.sort()

print(spam)

spam.sort(key=str.lower)

print(spam)
```

```
['Alice', 'Bob', 'ants', 'badgers', 'Carol', 'cats']
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
['A', 'a', 'Z', 'z']
['A', 'Z', 'a', 'z']
['A', 'a', 'Z', 'z']
```

4 Similarities Between Lists and strings

Lists are similar due to how strings can be seen as a list of single character strings. Due to this, a lot of what can be done with lists can also be done with strings.

```
print(list("Hello"))

name="Zophie"

print(name[0])

print(name[1:3])

print(name[-2])

print("Zo" in name)
print("xxx" in name)

for letter in name:
    print(letter)
```

```
['H', 'e', 'l', 'l', 'o']
Z
op
i
True
False
Z
o
p
h
i
e
```

One key difference is that lists are mutable while strings are immutable. Lists can have values added, removed, and changed while string values cannot be changed.

```
name="Zophie the cat"
print(name)
print(name[7])
```

```
Zophie the cat
t
```

However, trying to change the value of name[7] above would result in an error.

The correct way to modify a string is using a new string using slices.

```
name="Zophie a cat"
print(name)
```

```
newName="{}the{}".format(name[0:7],name[8:12])
print(newName)
```

```
Zophie a cat
Zophie the cat
```

The reason you can't change the string directly is because there is a large difference between mutable and immutable value in Python.

```
spam=42
cheese=spam


spam=100

print(spam,cheese)
```

```
100 42
```

However, lists do not work this way. When you assign a list to a variable, you are assigning a reference to that variable.

```
spam=[0,1,2,3,4,5]
cheese=spam

cheese[1]="Hello"

print(cheese,spam)
```

```
[0, 'Hello', 2, 3, 4, 5] [0, 'Hello', 2, 3, 4, 5]
```

As we see above, we only modified the cheese variable, however the value in the spam variable also changed. This is because Python uses references here. spam and cheese are the same list due to this distinction.

This does not happen to immutable values since they can't be modified in place, they can only be replaced by new values.

If you don't keep this in mind, it can lead to some issues.

```
def eggs(someParameter):
    someParameter.append("Hello")

spam= [1,2,3]

print(spam)

eggs(spam)

print(spam)
```

```
[1, 2, 3]
[1, 2, 3, 'Hello']
```

Above the changes made inside of the function are still applied outside of the local scope. This is due to the references that Python uses for lists.

Why have this complicated reference system to begin with?

Because lists and other immutable data types can be huge and having to copy the entire huge lists each time we make a function call would be slow and computationally expensive.

However, there is a way to make a true copy of a list. The deepcopy function from the copy module.

```
import copy

spam=["A","B","C","D"]

cheese=copy.deepcopy(spam)

print(spam,cheese)

cheese[1]=42

print(spam,cheese)
```

```
['A', 'B', 'C', 'D'] ['A', 'B', 'C', 'D']
['A', 'B', 'C', 'D'] ['A', 42, 'C', 'D']
```

When making a deepcopy, we can edit the copy of the list without editing the original.

Lists can also spam multiple lines of code, regardless of the indentation.

```
spam=["apples",
      "oranges",
      "bananas",
      "cats"]

print(spam)
```

```
['apples', 'oranges', 'bananas', 'cats']
```

For other non lists, we can use “\” in order to deliminate new lines.