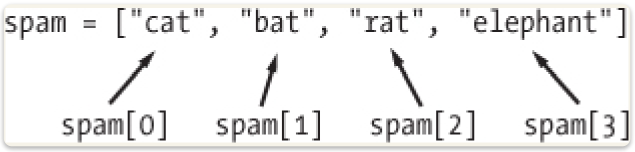
List Data Type

A list is a value that contains multiple values in an ordered sequence. The term list value refers to the list itself (which is a value that can be stored in a variable or passed to a function like any other value), not the values inside the list value. A list value looks like this: `['cat', 'bat', 'rat', 'elephant']`. Just as string values are typed with quote characters to mark where the string begins and ends, a list begins with an opening square bracket and ends with a closing square bracket, `[]`. Values inside the list are also called `items`. Items are separated with comma (that is, they are comma-delimited).

```
lucky_number = [168,888,619]
spam = ['cat', 'bat', 'rat', 'elephant']
```

Getting individual values in a list with Indexes

Say you have the list ['cat', 'bat', 'rat', 'elephant'] stored in a variable named spam. The Python code spam[0] would evaluate to 'cat', and spam[1] would evaluate to 'bat', and so on. The integer inside the square brackets that follows the list is called an index. The first value in the list is at index 0, the second value is at index 1, the third value is at index 2, and so on. Figure 4-1 shows a list value assigned to spam, along with what the index expressions would evaluate to. Note that because the first index is 0, the last index is one less than the size of the list; a list of four items has 3 as its last index



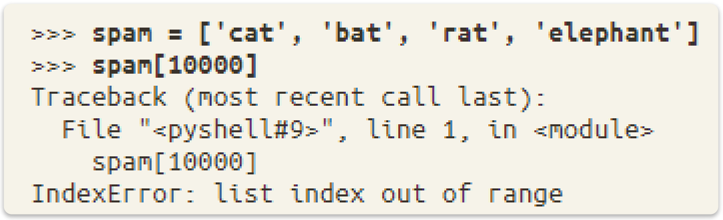
```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[0]
# ->'cat'
```

```
spam[1]
# ->'bat'
spam[2]
# ->'rat'
spam[3]
# ->'elephant'
['cat', 'bat', 'rat', 'elephant'][3]
# ->'elephant'

'Hello, ' + spam[0]
# -> 'Hello, cat'
'The ' + spam[1] + ' ate the ' + spam[0] + '.'
# ->'The bat ate the cat.'
```

Notice that the expression 'Hello, ' + spam[0] evaluates to 'Hello, ' + 'cat' because spam[0] evaluates to the string 'cat'. This expression in turn evaluates to the string value 'Hello, cat' .

Python will give you an IndexError error message if you use an index that exceeds the number of values in your list value.



```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    spam[10000]
IndexError: list index out of range
```

Note

Indexes can be only integer values, not floats. The following example will cause a TypeError error:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    spam[1.0]
TypeError: list indices must be integers or slices, not float
```

Negative indexes

While indexes start at 0 and go up, you can also use negative integers for the index. The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on. Enter the following into the interactive shell:

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[-1]
# -> 'elephant'
spam[-3]
# -> 'bat'
'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
# -> 'The elephant is afraid of the bat.'
```

Getting a list's length with the len() function

The len() function will return the number of values that are in a list value passed to it, just like it can count the number of characters in a string value. Enter the following into the interactive shell:

```python
spam = ['cat', 'dog', 'moose']
len(spam)
# -> 3
```

Changing values in a list with indexes

Normally, a variable name goes on the left side of an assignment statement, like spam = 42. However, you can also use an index of a list to change the value at that index. For example, spam[1] = 'aardvark' means “Assign the value at index 1 in the list spam to the string 'aardvark'.” Enter the following into the interactive shell:

```python
spam = ['cat', 'bat', 'rat', 'elephant']
spam[1] = 'aardvark'
spam
# -> ['cat', 'aardvark', 'rat', 'elephant']
spam[2] = spam[1]
spam
# -> ['cat', 'aardvark', 'aardvark', 'elephant']
spam[-1] = 12345
spam
# -> ['cat', 'aardvark', 'aardvark', 12345]
```

List Concatenation and List Replication

Lists can be concatenated and replicated just like strings. The + operator combines two lists to create a new list value and the * operator can be used with a list and an integer value to replicate the list. Enter the following into the interactive shell:

```python
[1, 2, 3] + ['A', 'B', 'C']
# -> [1, 2, 3, 'A', 'B', 'C']

['X', 'Y', 'Z'] * 3
# -> ['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
```

```
spam = [1, 2, 3]
spam = spam + ['A', 'B', 'C']
spam
# -> [1, 2, 3, 'A', 'B', 'C']
```

Argmented Assignment Operators

We can also assign new item to a list by using an assigment operator

```
a = [1,2,3]
a += [4]

a
# -> [1,2,3,4]
```

List Method

A method is the same thing as a function, except it is “called on” a value. For example, if a list value were stored in spam, you would call the index() list method (which I’ll explain shortly) on that list like so: spam.index('hello'). The method part comes after the value, separated by a period.

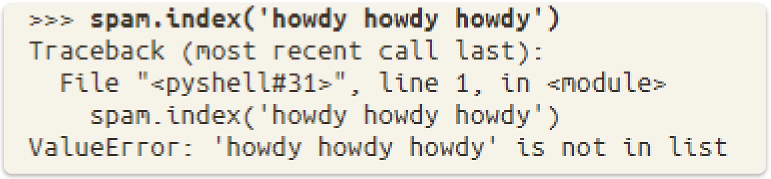
Each data type has its own set of methods. The list data type, for example, has several useful methods for finding, adding, removing, and otherwise manipulating values in a list.

FIND A VALUE IN A LIST WITH INDEX() METHOD

List values have an index() method that can be passed a value, and if that value exists in the list, the index of the value is returned. If the value isn’t in the list, then Python produces a ValueError error. Enter the following into the interactive shell:

```
spam = ['hello', 'hi', 'howdy', 'heyas']
spam.index('hello')
```

```
# -> 0
spam.index('heyas')
# -> 3
```



```
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

When there are duplicates of the value in the list, the index of its first appearance is returned. Enter the following into the interactive shell, and notice that index() returns 1, not 3:

```
spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
spam.index('Pooka')
```

ADDING VALUES TO LISTS WITH THE APPEND() AND INSERT() METHODS

To add new values to a list, use the append() and insert() methods. Enter the following into the interactive shell to call the append() method on a list value stored in the variable spam:

```
spam = ['cat', 'dog', 'bat']
spam.append('moose')
spam
# -> ['cat','dog','bat','moose']
```

The previous append() method call adds the argument to the end of the list. The insert() method can insert a value at any index in the list. The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted. Enter the following into the interactive shell:

```
spam = ['cat', 'dog', 'bat']
spam.insert(1, 'chicken')
spam
# -> ['cat', 'chicken', 'dog', 'bat']
```

For Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). With the for loop we can execute a set of statements, once for each item in a list, string, etc.

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

for x in fruits:
        print(x)

# -> "apple"
# -> "banana"
# -> "cherry"
```

Looping through a string

even strings are iterable objects, they contain a sequence of characters:

```
for x in "banana":
        print(x)

# -> "b"
# -> "a"
# -> "n"
# -> "a"
# -> "n"
# -> "a"
```

the in and not in operators

You can determine whether a value is or isn’t in a list with the in and not in operators. Like other operators, in and not in are used in expressions and connect two values: a value to look for in a list and the list where it may be found. These expressions will evaluate to a Boolean value. Enter the following into the interactive shell:

```
'howdy' in ['hello', 'hi', 'howdy', 'heyas']
# -> True
spam = ['hello', 'hi', 'howdy', 'heyas']
'cat' in spam
# -> False
'howdy' not in spam
# -> False
'cat' not in spam
# -> True
```

the range() function

to loop through a set of code a specified number of times, we can use the `range()` function,
The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
        print(x)

# -> 0
# -> 1
# -> 2
# -> 3
# -> 4
# -> 5
```

the enumerate() function

Instead of using the range(len(someList)) technique with a for loop to obtain the integer index of the items in the list, you can call the enumerate() function instead. On each iteration of the loop, enumerate() will return two values:

```python
supplies = ['pens', 'staplers', 'flamethrowers', 'binders']
for index, item in enumerate(supplies):
        print('Index ' + str(index) + ' in supplies is: ' + item)

# Index 0 in supplies is: pens
# Index 1 in supplies is: staplers
# Index 2 in supplies is: flamethrowers
# Index 3 in supplies is: binders
```