Mod 02 - Python Lists and Advanced Flow Control

SCRIPTING ESSENTIALS

DR. BURKMAN

Learning Objectives

Understanding Python lists

indexes

slices

length

changing values

concatenating and replicating lists

deleting values

looping through lists

Checking if a value is in (or not in) a list

list methods

index

append

insert

remove

sorting a list

mutability

tuples

converting tuples and lists

list references

copying a list (deepcopy)

Learning Objectives

Functions

Variable Scope

Exception Handling

Lists

Basics:

- A list contains multiple values in an ordered sequence.
- Items are the things inside a list
- Lists are inside square brackets [ ]
- Item are comma delimited

Examples:

- [1, 2, 3]
- [‘cat’, ‘rat’, ‘bat’]
- [1, ‘Bob’, 3.14]

List Indexes

Items in a list are access by integer position, left to right, starting with zero

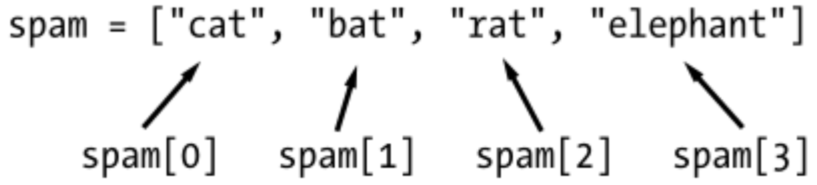
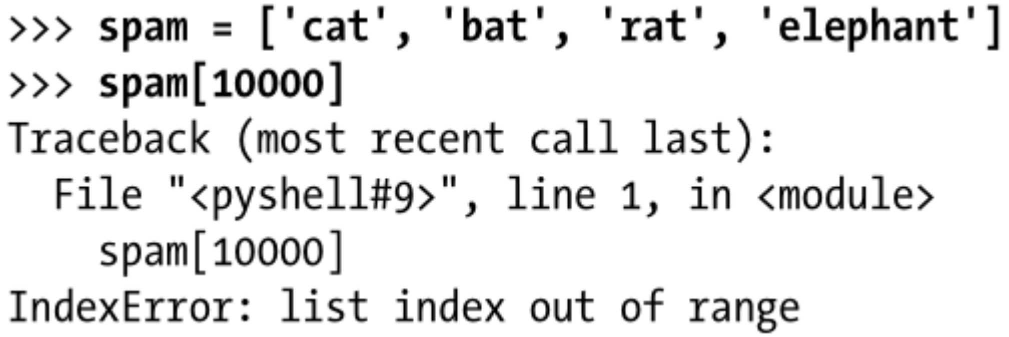


Figure 4-1: A list value stored in the variable spam, showing which value each index refers to

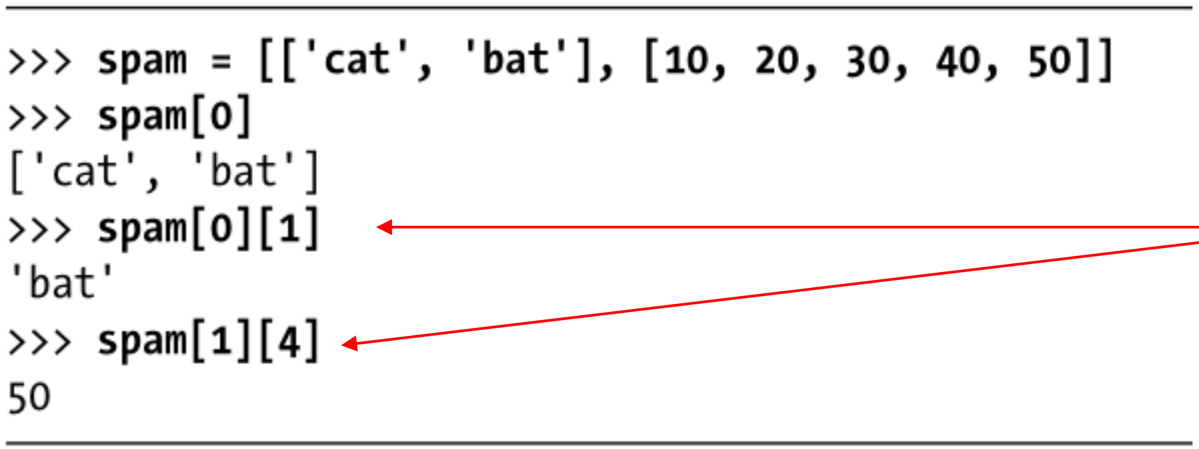
Index out of range error:



```
>>> 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
```

List Indexes

Lists can contain lists:



```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
>>> spam[0][1]
'bat'
>>> spam[1][4]
50
```

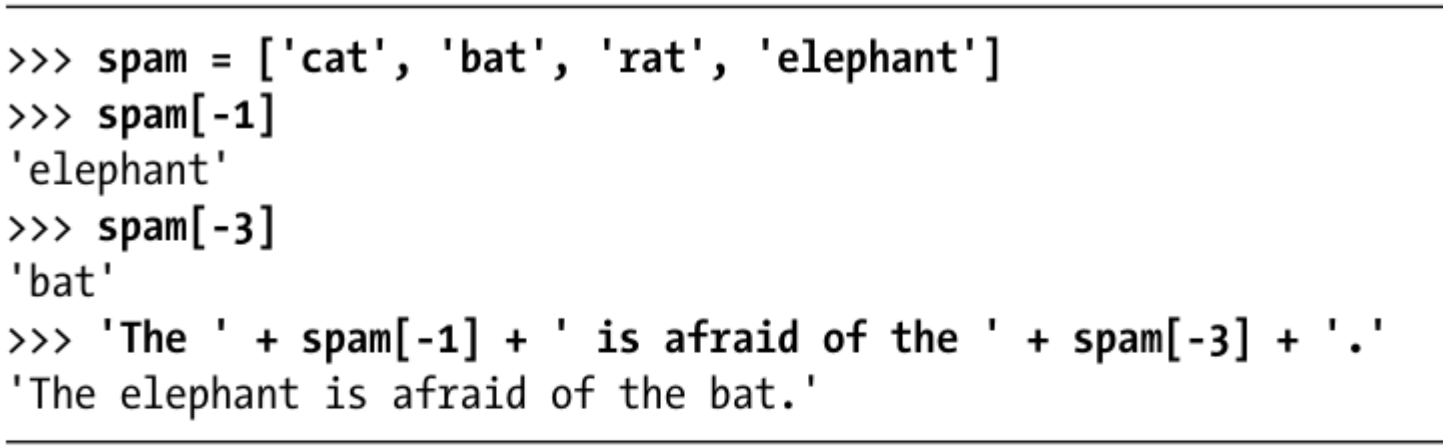
Note how you access items in a list, in a list

```
spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
print (spam[0])
print (spam[0][1])
print (spam[1][4])
```

List Indexes

You can use negative indexes

- They reference right to left



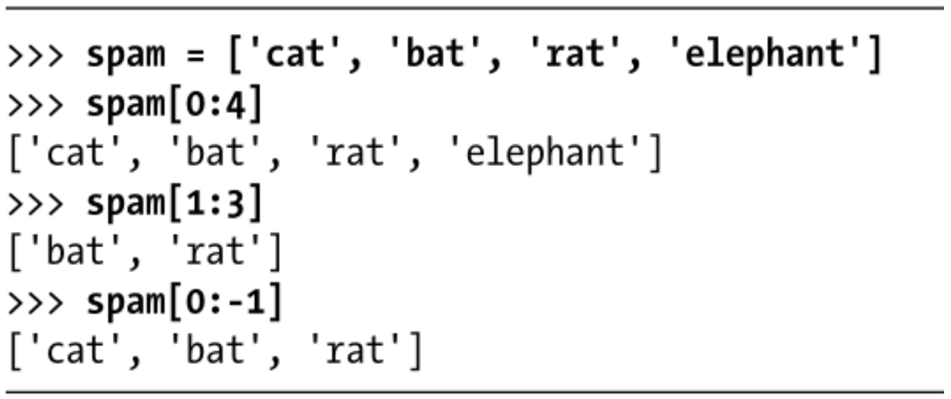
```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
'The elephant is afraid of the bat.'
```

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

List Indexes

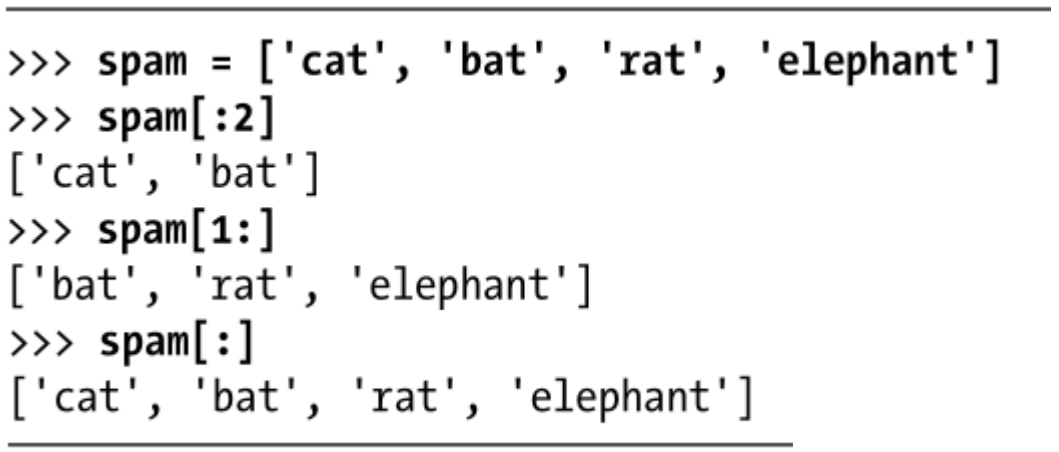
Slices get several values from a list

- List[start:end]
  - The end value is never included



```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

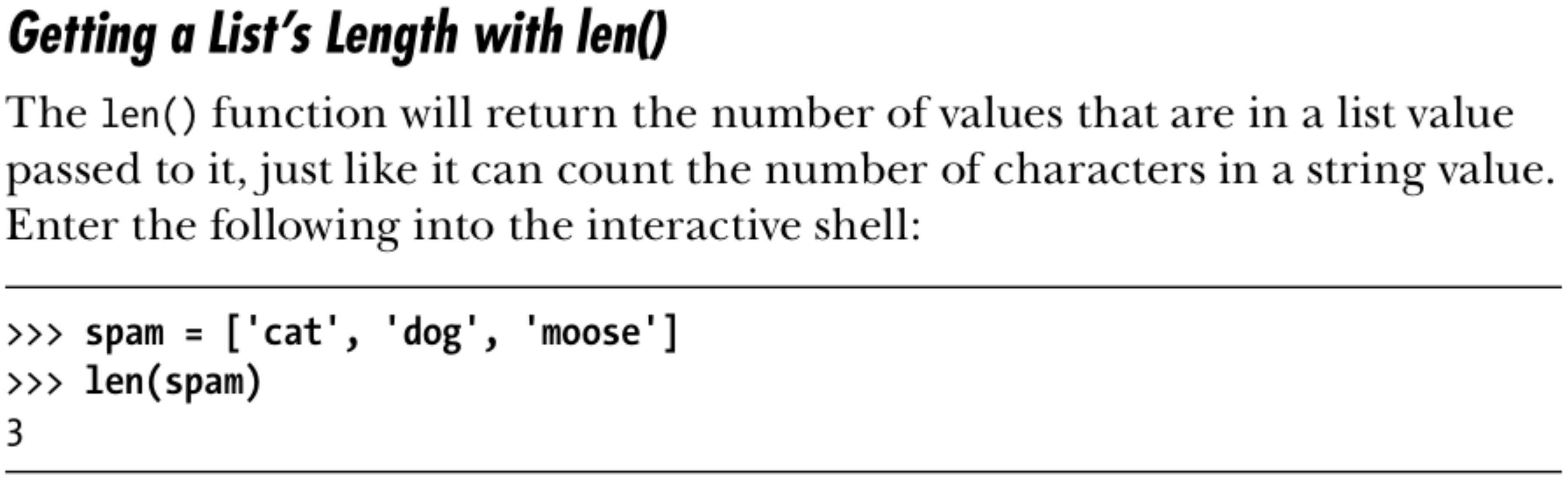
Or you can leave out one end:



```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
['cat', 'bat']
>>> spam[1:]
['bat', 'rat', 'elephant']
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

```
spam = ['cat', 'bat', 'rat', 'elephant']
print (spam[0:4])
print (spam[1:3])
print (spam[0:-1])
print (spam[:2])
print (spam[1:])
print (spam[:])
```

List Length



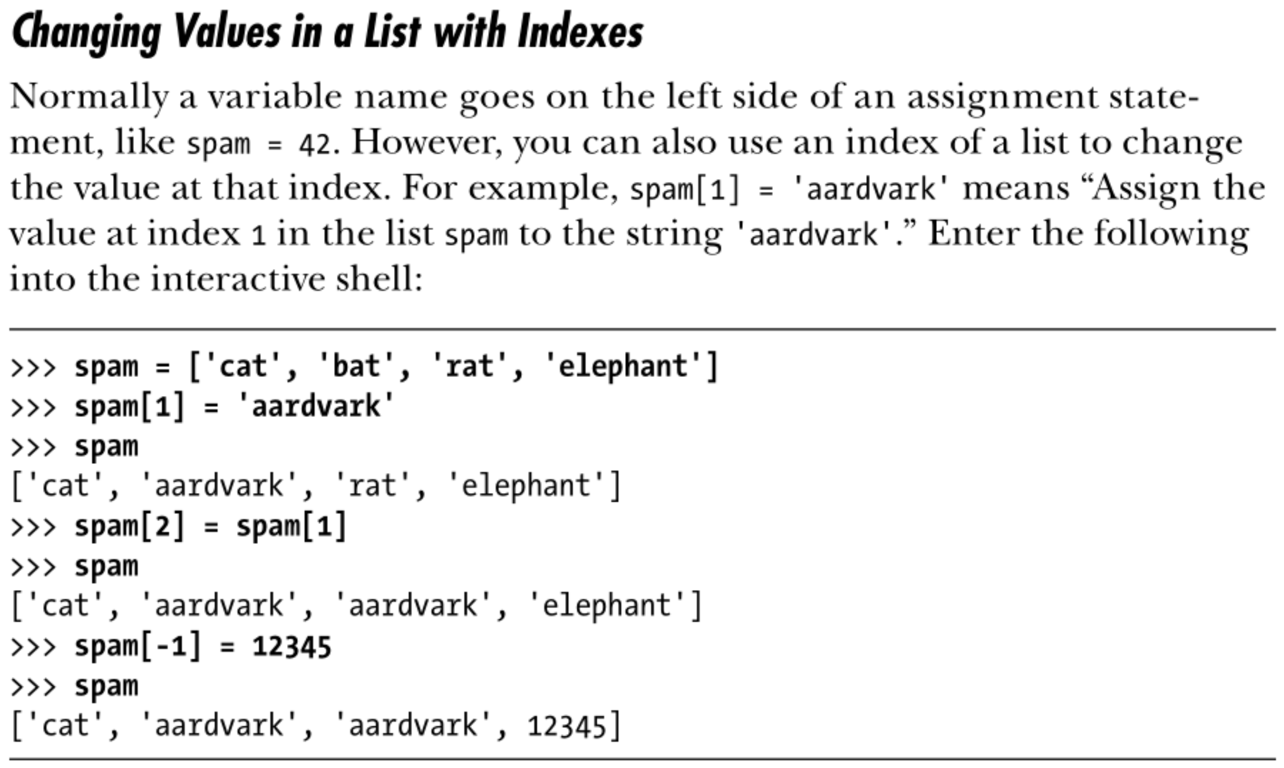
Getting a List's Length with len()

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:

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

spam = ['cat', 'dog', 'moose']
print (len(spam))

Changing List Values



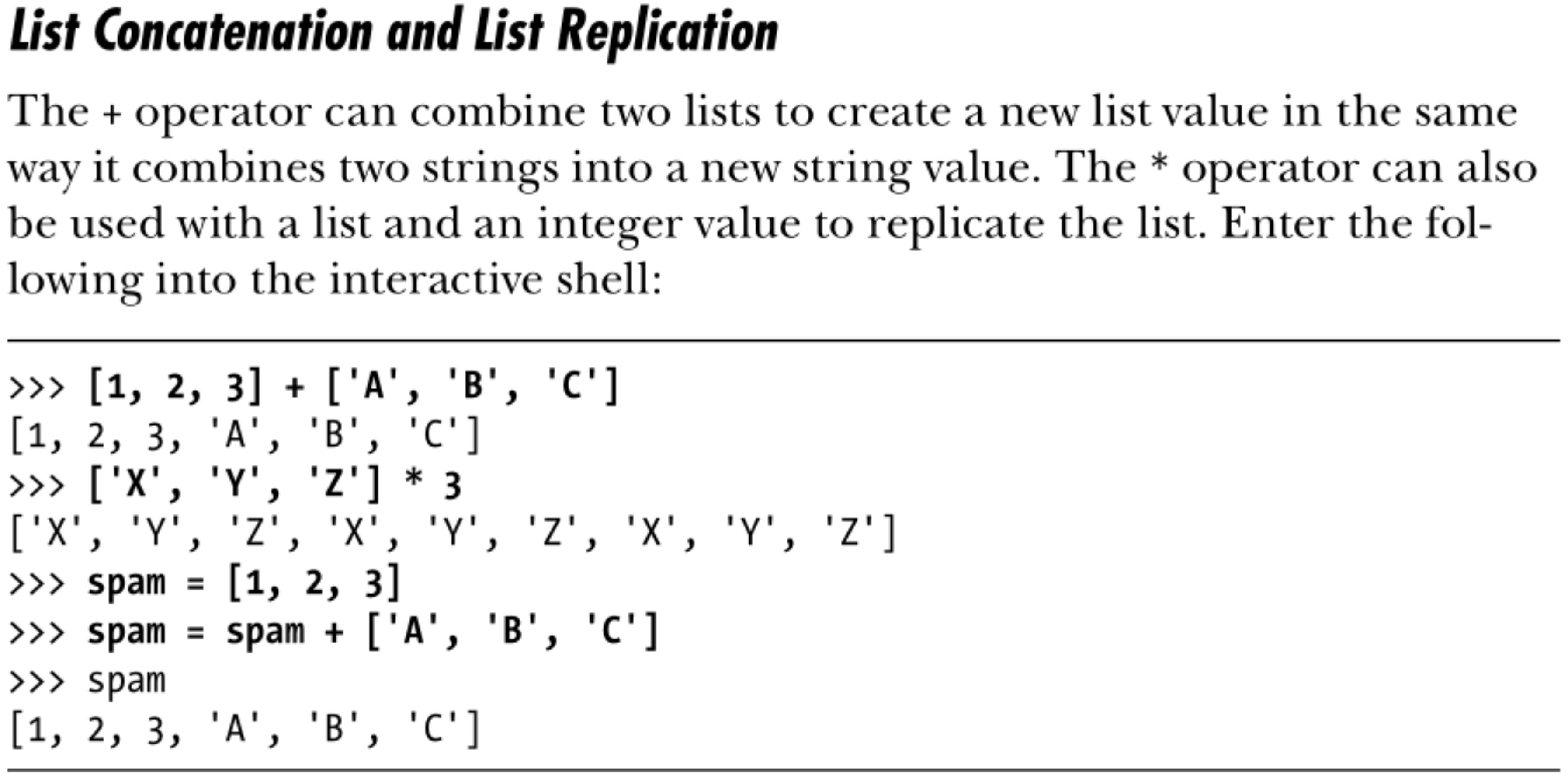
Changing Values in a List with Indexes

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:

```
>>> 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]
```

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[1] = 'aardvark'
print(spam)
spam[2] = spam[1]
print (spam)
spam[-1] = 12345
print (spam)
```

List Concatenation and Replication

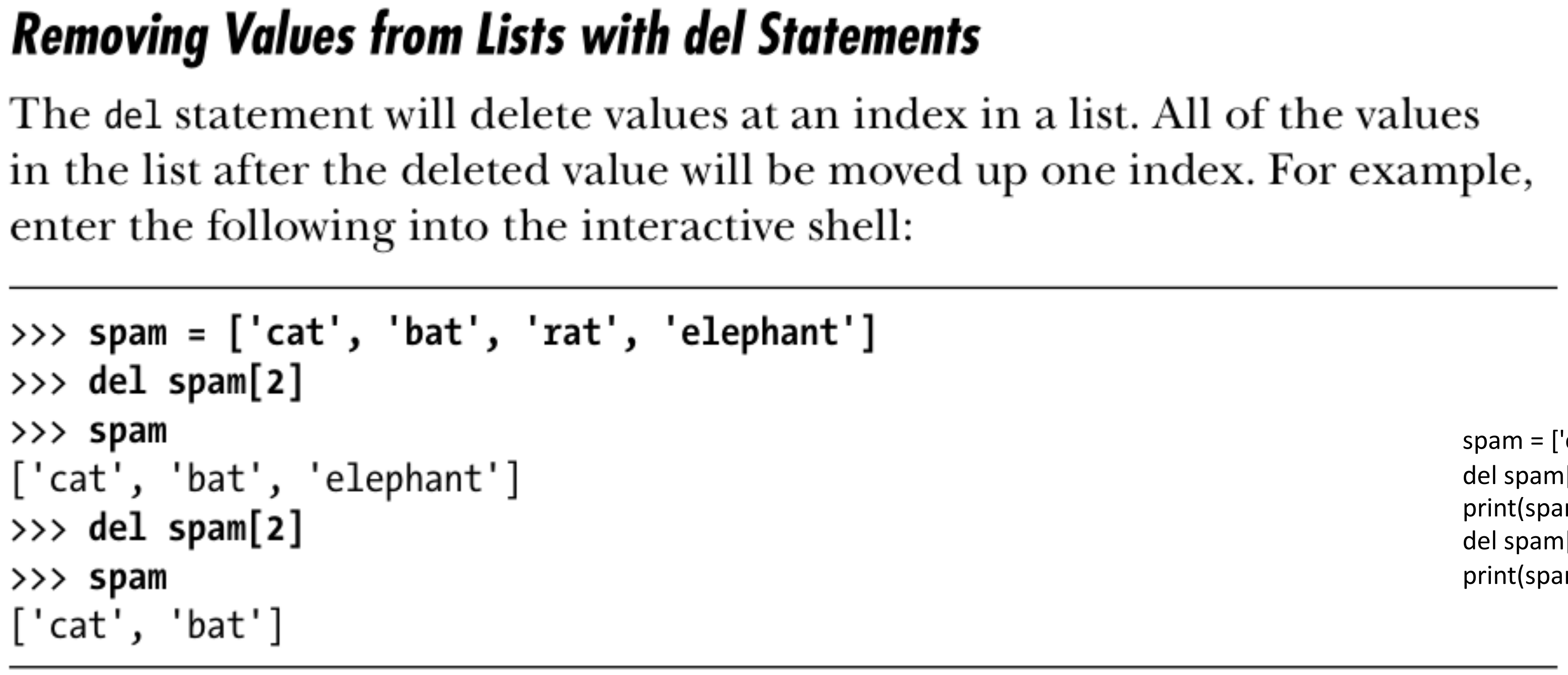


List Concatenation and List Replication

The + operator can combine two lists to create a new list value in the same way it combines two strings into a new string value. The * operator can also be used with a list and an integer value to replicate the list. Enter the following into the interactive shell:

```
>>> [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']
```

Removing Values with del



Removing Values from Lists with del Statements

The `del` statement will delete values at an index in a list. All of the values in the list after the deleted value will be moved up one index. For example, enter the following into the interactive shell:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

```
spam = ['cat', 'bat', 'rat', 'elephant']
del spam[2]
print(spam)
del spam[2]
print(spam)
```

Example Usage (Try this)

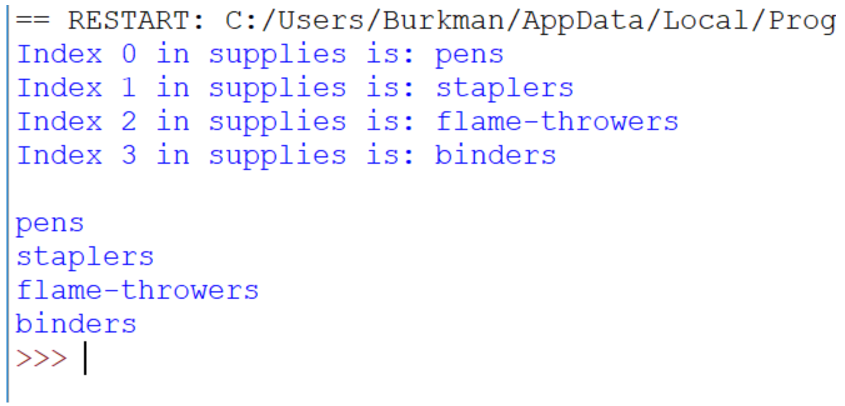
```
cat_names = [] #I am initializing an empty list
while True:
    print('Enter the name of cat ' + str(len(cat_names) +1) +
          ' (Or enter nothing to stop.):')
    name = input()
    if name == '':
        break
    cat_names = cat_names + [name] #list concatenation
print('The cat names are:')
for name in cat_names:
    print('  ' + name)
```

Looping Lists

```
supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
for i in range(len(supplies)):
    print('Index ' + str(i) + ' in supplies is: ' + supplies[i])

print()

for j in supplies:
    print(j)
```

Note: len(supplies) = 4

Instead of using For to iterate through a range, we are iterating through a collection.



```
== RESTART: C:/Users/Burkman/AppData/Local/Prog
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

pens
staplers
flame-throwers
binders
>>> |
```

Is X in my list?

In and Not in yield a Boolean evaluation:

```
>>> '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
```

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter a pet name:')
name = input()
if name not in myPets:
    print('I do not have a pet named ' + name)
else:
    print(name + ' is my pet.')
```

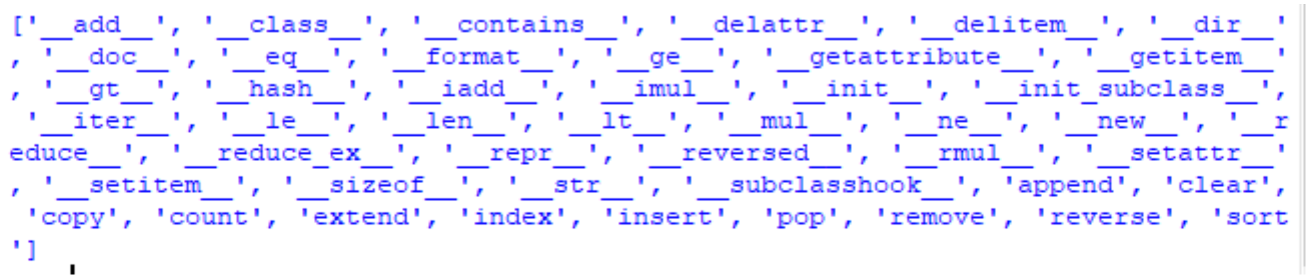
Methods

Same as a function but it is called on a value. Data types have methods.

- Method comes after value, separated by a period.

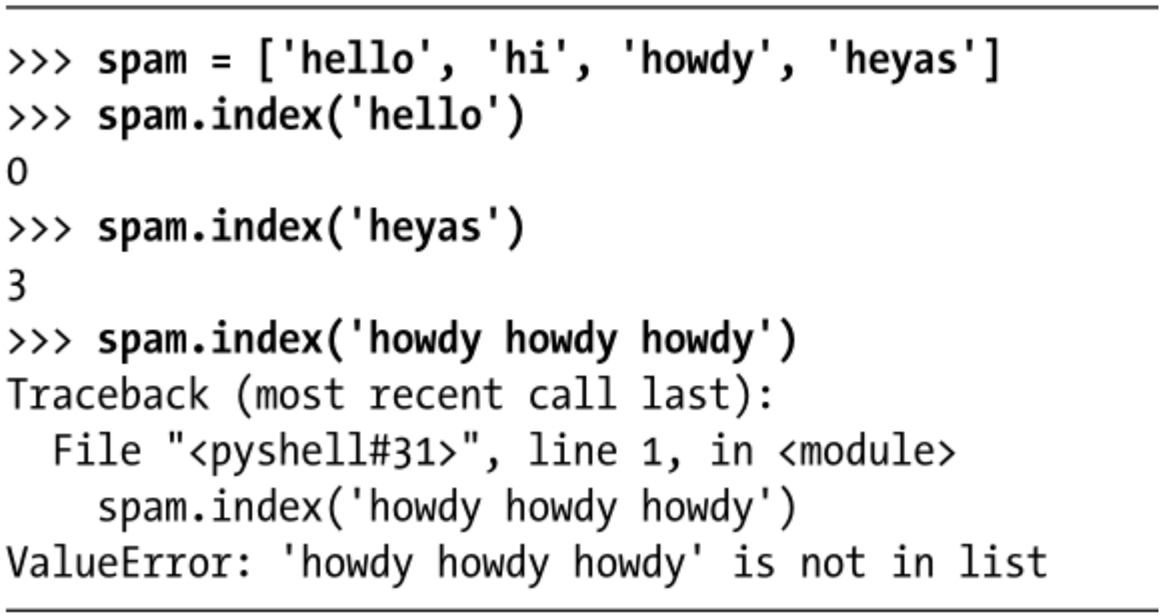
Remember, we can find the methods with dir

- dir(list)



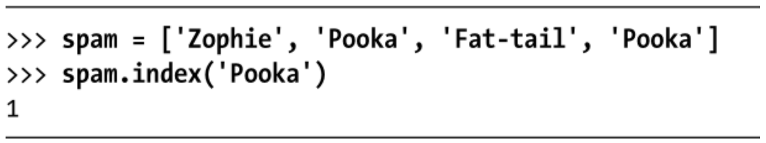
```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__'
, '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__'
, '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
 '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__r
educe__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__'
, '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort
']
```

list.index



```
>>> 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
```

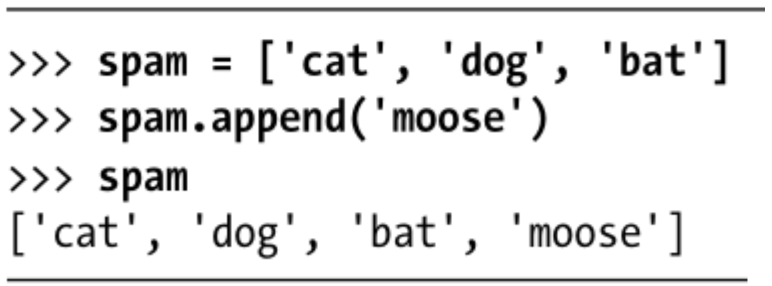
Note: If duplicates are in a list, index only finds the first instance



```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

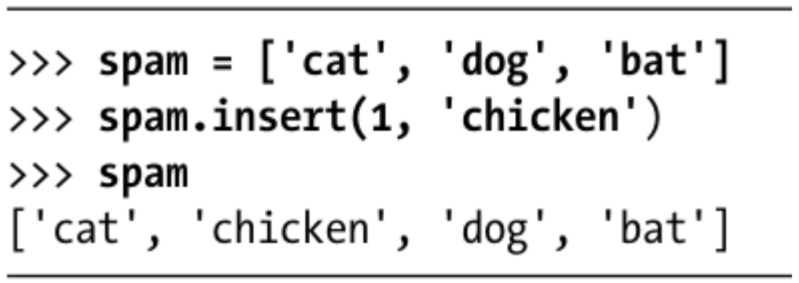
list.append and list.insert

list.append



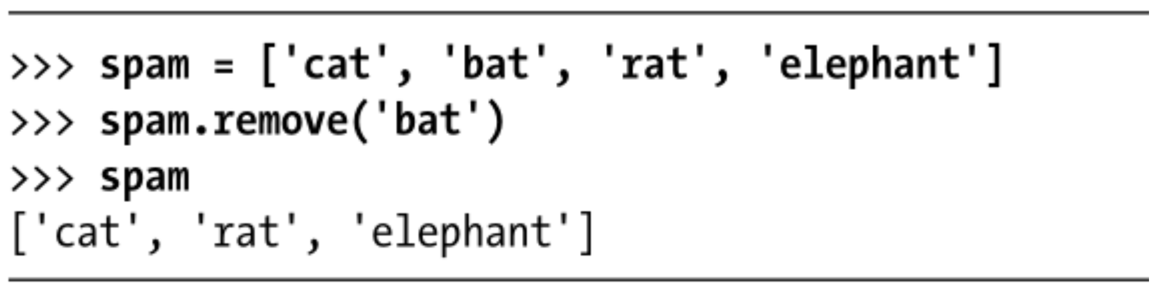
```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

list.insert



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

list.remove

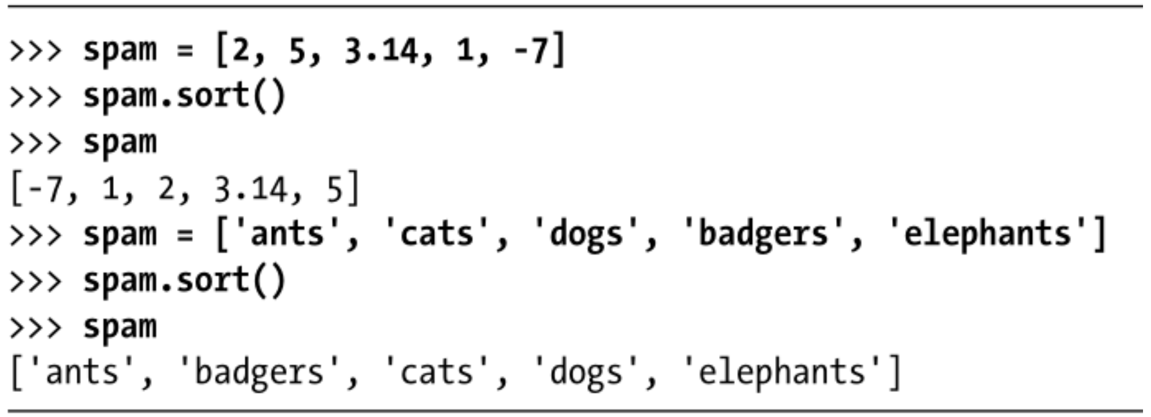


```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

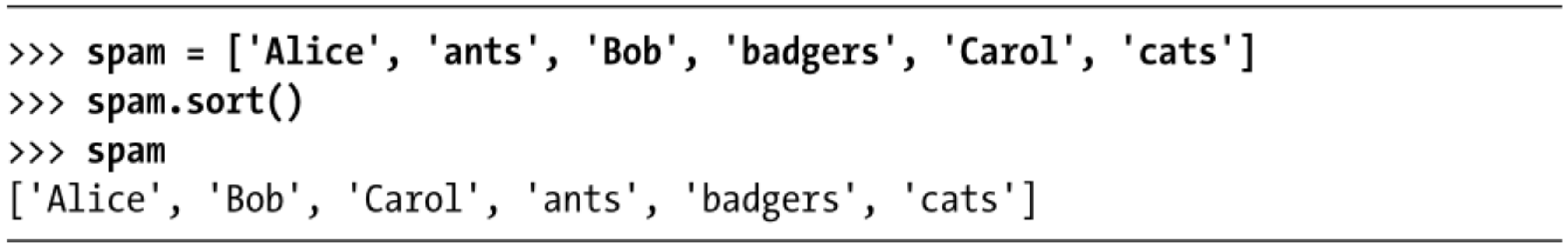
Sorting a list

The sort method can only sort:

- like things (no mixing numbers and strings)
- ASCII style
  - A a B b

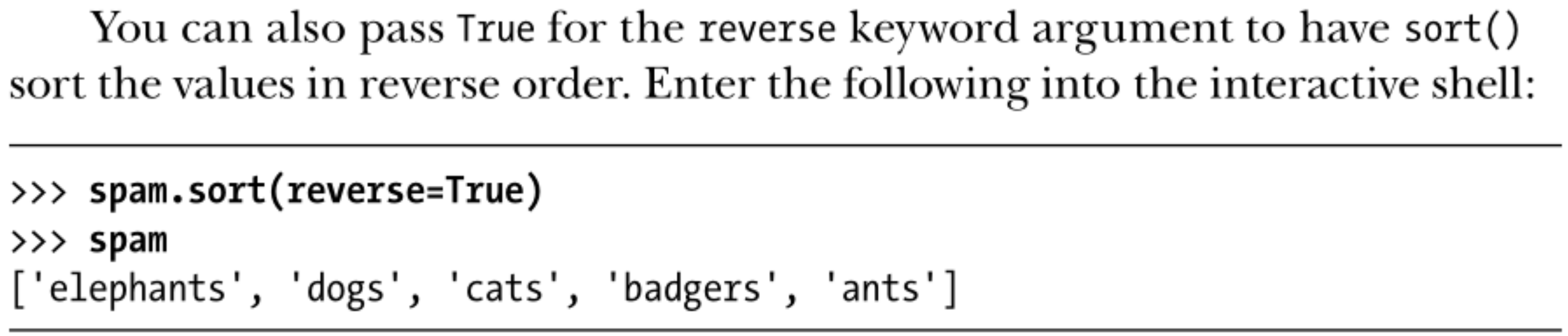


```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```



```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

Sorting a list

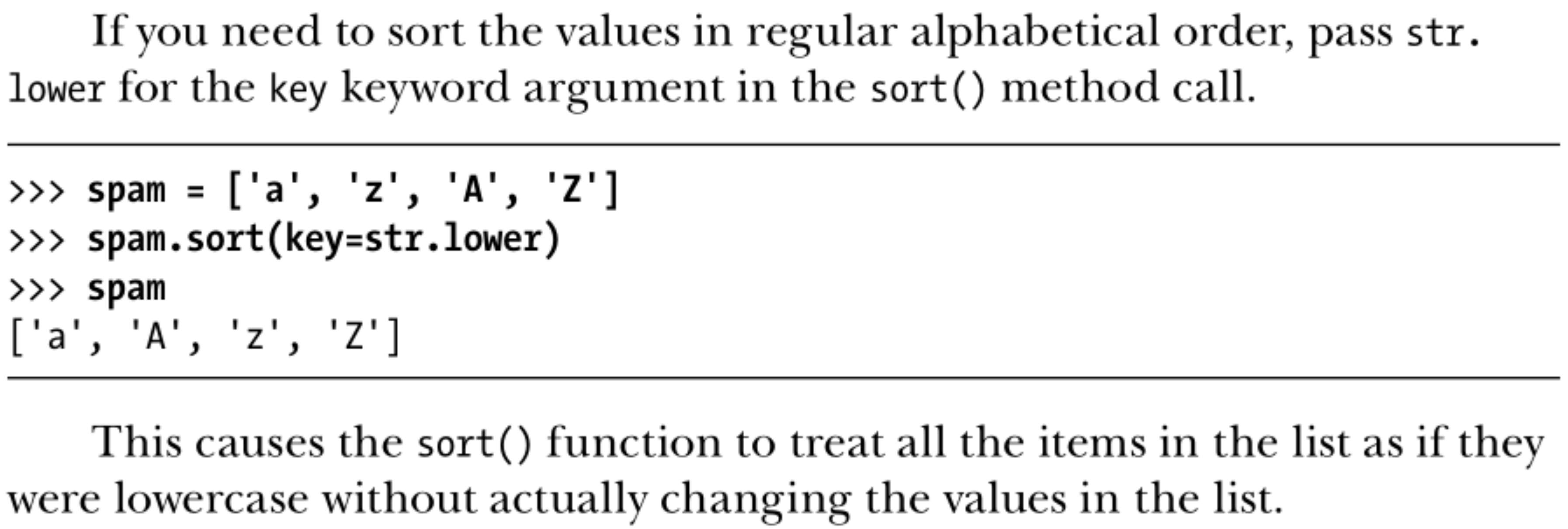


You can also pass True for the reverse keyword argument to have sort() sort the values in reverse order. Enter the following into the interactive shell:

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

Sorting a list

You can also sort using the key keyword argument:



If you need to sort the values in regular alphabetical order, pass `str.lower` for the `key` keyword argument in the `sort()` method call.

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

This causes the `sort()` function to treat all the items in the list as if they were lowercase without actually changing the values in the list.

Try This

```
import random

messages = ['It is certain',
'It is decidedly so',
'Yes definitely',
'Reply hazy try again',
'Ask again later',
'Concentrate and ask again',
'My reply is no',
'Outlook not so good',
'Very doubtful']

print(messages[random.randint(0, len(messages) - 1)])
```

A Slight Change

```
import random

a=0

while a<5:
    messages = ['It is certain',
    'It is decidedly so',
    'Yes definitely',
    'Reply hazy try again',
    'Ask again later',
    'Concentrate and ask again',
    'My reply is no',
    'Outlook not so good',
    'Very doubtful']

    print(messages[random.randint(0, len(messages) - 1)])
    a = a +1
```

Mutability

Mutable

- Can have values added, removed or changed
- Lists are mutable

Immutable

- Cannot be changed
- Strings and tuples are immutable

Tuple

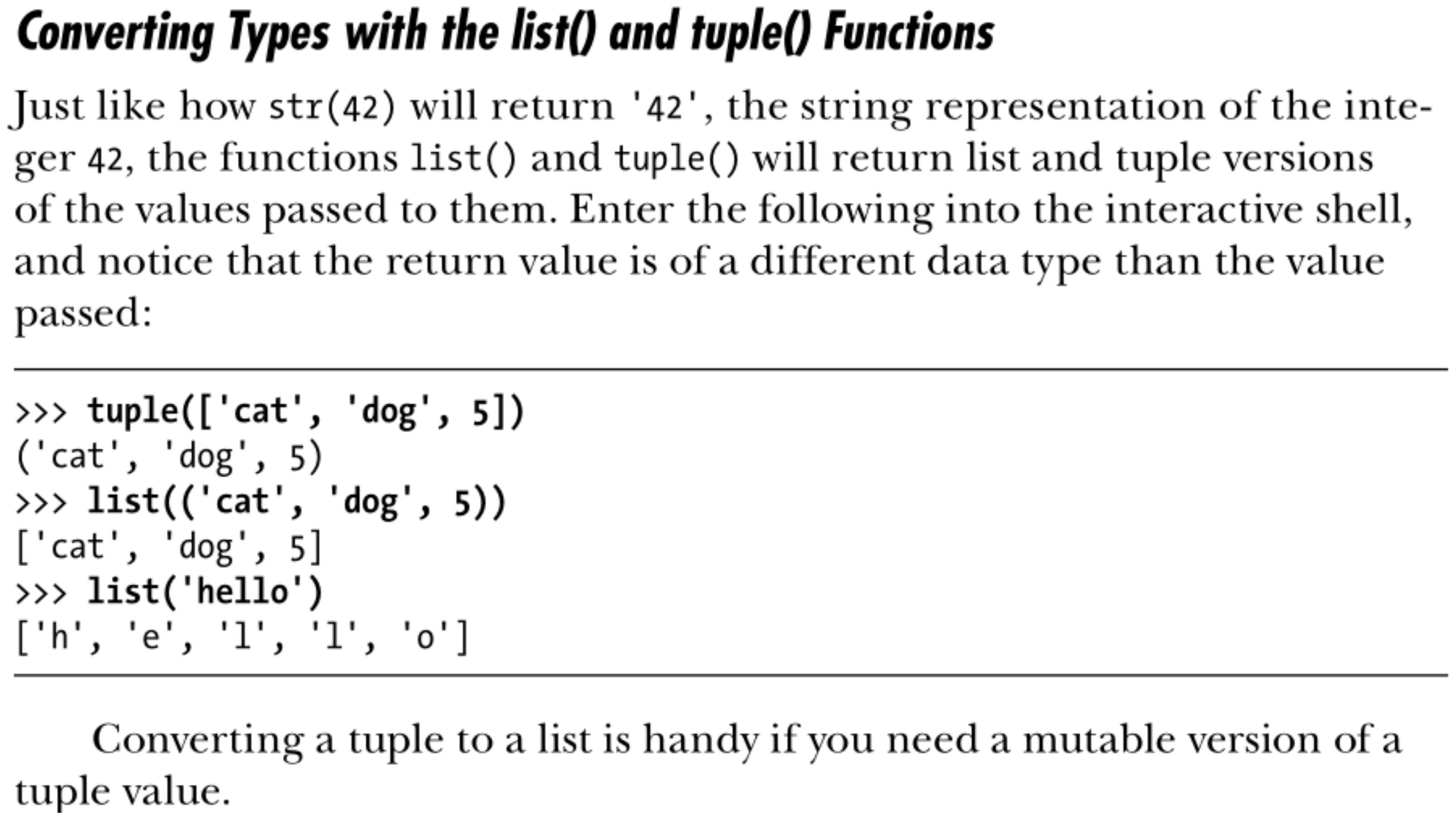
A tuple acts like an immutable list

- Has parentheses instead of square brackets.

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

Converting Types



Converting Types with the list() and tuple() Functions

Just like how `str(42)` will return `'42'`, the string representation of the integer `42`, the functions `list()` and `tuple()` will return list and tuple versions of the values passed to them. Enter the following into the interactive shell, and notice that the return value is of a different data type than the value passed:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

Converting a tuple to a list is handy if you need a mutable version of a tuple value.

Populating a List with a Range

In Python 3 range is an iterator so you have to convert it to a list.

- Note that the upper value isn’t included

```
my_list = list(range(1, 1001))
print(my_list)
```

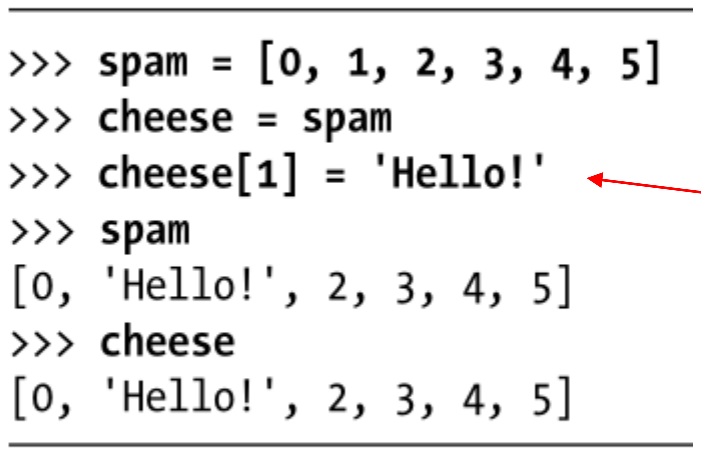
Understanding List References

Variables work like we’d expect. Each variable is its own “container”, even if it gets “filled” by another variable.

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Understanding List References

Lists don’t work this way. When you assign a list to a variable, the variable is only storing a reference (like a pointer) to that list. There are never multiple instances of one particular list.



```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = spam
>>> cheese[1] = 'Hello!'
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```

Here, cheese is just pointing to the original spam list. Changes made to cheese actually go right over to the original spam list.

Copying a list with copy()

```
import copy

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

cheese = copy.copy(spam) #This does make a new list

cheese[1] = 42

print(spam)
print(cheese)
```

Copy.deepcopy

```
#Use deepcopy() if your list contains lists

import copy

spam = [[1,2,3],['a', 'b', 'c']]

cheese = copy.deepcopy(spam) #This does make a new list

cheese[0][1] = 42

print(spam)
print(cheese)
```

Functions

A named chunk of code that may, or may not, accept parameters. The parameter value only exists while the function runs (“name”, in the second example).

```
def hello():
    print('Hello World')


hello()
```

```
def hello(name):
    print('Hello ' + name)


hello('Jim')
hello('Nita')
```

Functions

Return does just that, it returns a value from a function

Here r is assigned a random value, then that value is passed to the getAnswer function. The return value gets assigned to "fortune", which is then printed.

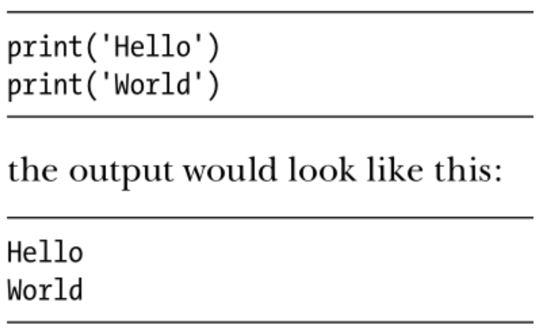
```
import random
def getAnswer(answerNumber):
    if answerNumber == 1:
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

Keyword Arguments in Built-in Functions

Generally the order of function inputs matters

- random.randint(1, 10) take the lower and upper bounds. It is not the same as random.randint(10, 1)

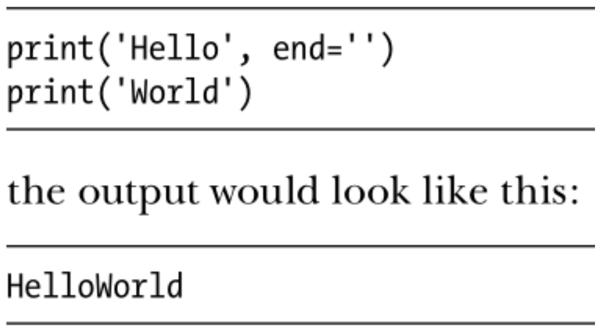
Keyword arguments are used for optional parameters



```
print('Hello')
print('World')
```

the output would look like this:

```
Hello
World
```



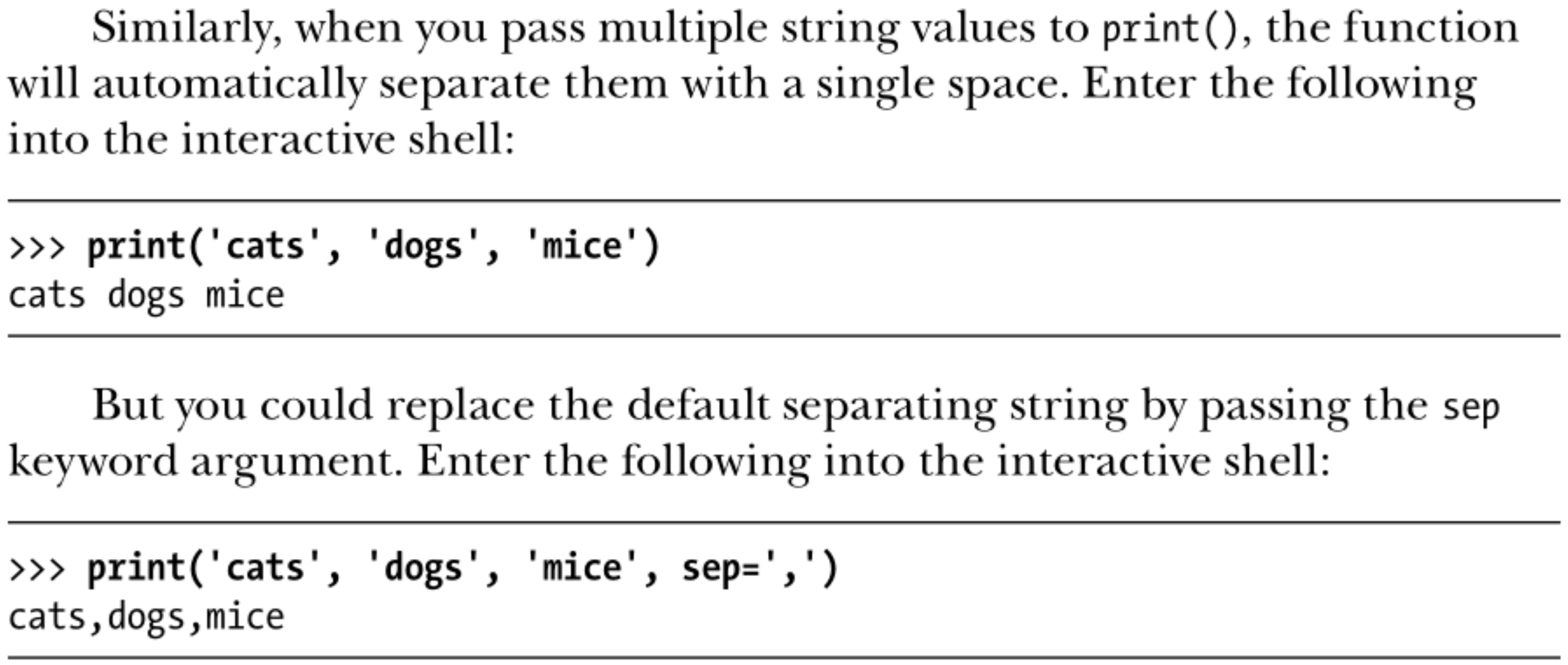
```
print('Hello', end='')
print('World')
```

the output would look like this:

```
HelloWorld
```

Here the “end” parameter changes the newline to the specified delimiter ‘’.

Keyword Arguments in Built-in Functions



Similarly, when you pass multiple string values to `print()`, the function will automatically separate them with a single space. Enter the following into the interactive shell:

```
>>> print('cats', 'dogs', 'mice')
cats dogs mice
```

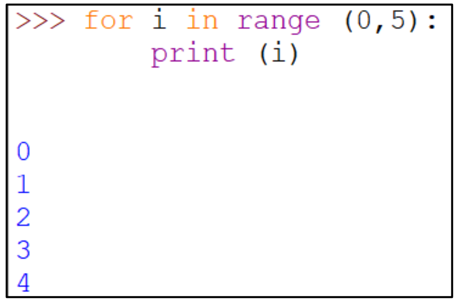
But you could replace the default separating string by passing the `sep` keyword argument. Enter the following into the interactive shell:

```
>>> print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

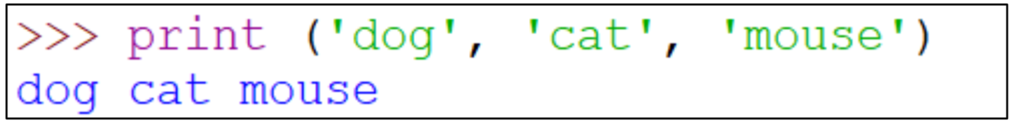
Sep vs End

By default the end parameter in Print is a new line. End changes that, so if you have multiple print statements (or a print statement that runs multiple times in a loop) you can get the output items together with something like end = ' '

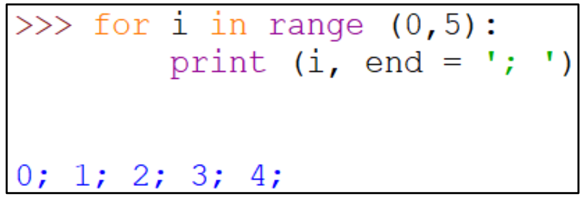
Sep provides the character(s) between items being printed. The default is a space (technically a soft space)



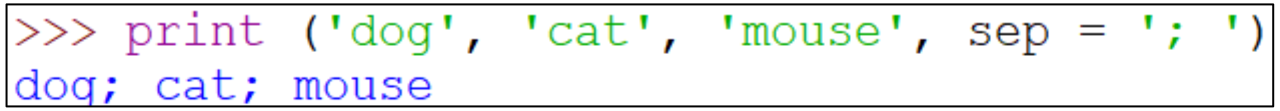
```
>>> for i in range (0,5):
        print (i)


0
1
2
3
4
```



```
>>> print ('dog', 'cat', 'mouse')
dog cat mouse
```



```
>>> for i in range (0,5):
        print (i, end = '; ')


0; 1; 2; 3; 4;
```



```
>>> print ('dog', 'cat', 'mouse', sep = '; ')
dog; cat; mouse
```

Scope

Scope is like a container for all your variables

- Global scope is created when your program begins, and destroyed when the program ends
  - There’s only one global scope
- Local scope is created whenever a function is called, and destroyed when the function ends.
  - There can be many local scopes

Access rights:

- Global cannot access local
- Local cannot access other local
- Local can access Global

Variable names are unique within the scope. Global spam != local spam

- But be sane and use unique variable names everywhere

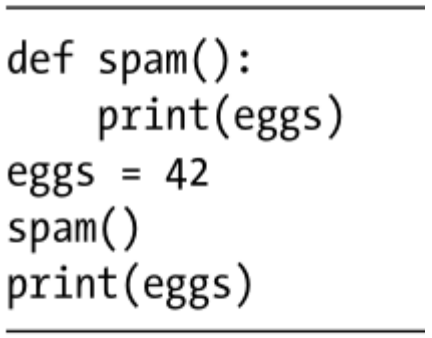
Scope

```
def spam():
    eggs = 31337
spam()
print(eggs)
```

```
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(eggs)
NameError: name 'eggs' is not defined
```

```
def spam():
    eggs = 99
    bacon()
    print(eggs)

def bacon():
    ham = 101
    eggs = 0

spam()
```

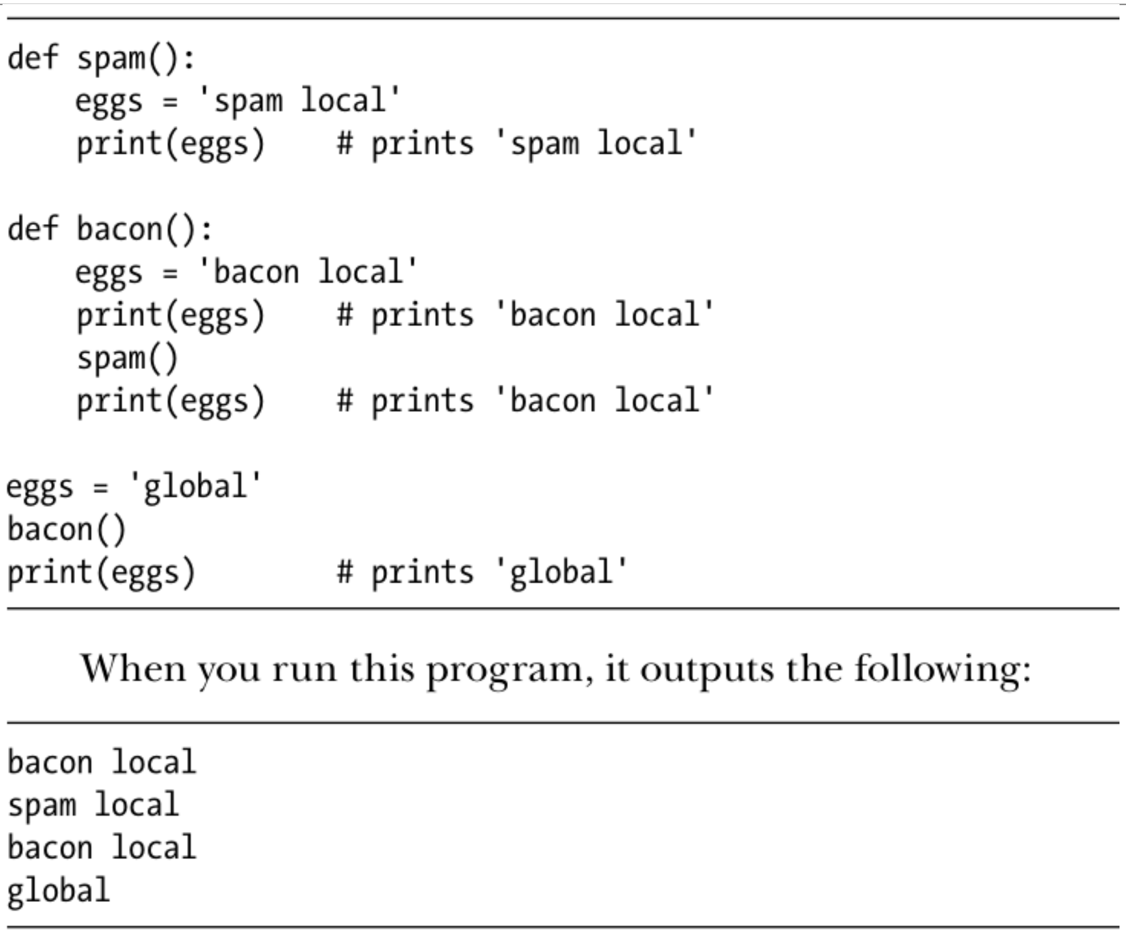
```
99
```



```
def spam():
    print(eggs)
eggs = 42
spam()
print(eggs)
```

```
42
42
```

Scope



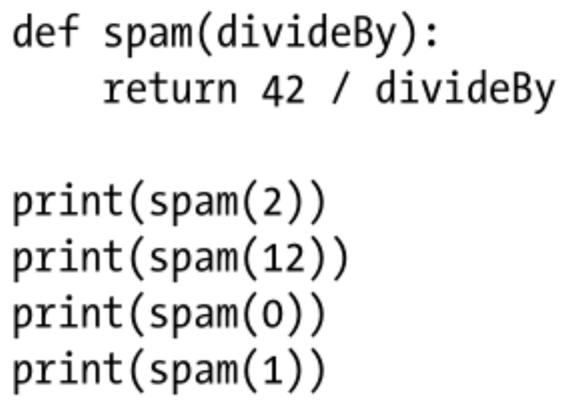
```
def spam():
    eggs = 'spam local'
    print(eggs)    # prints 'spam local'

def bacon():
    eggs = 'bacon local'
    print(eggs)    # prints 'bacon local'
    spam()
    print(eggs)    # prints 'bacon local'

eggs = 'global'
bacon()
print(eggs)        # prints 'global'
```

When you run this program, it outputs the following:

```
bacon local
spam local
bacon local
global
```

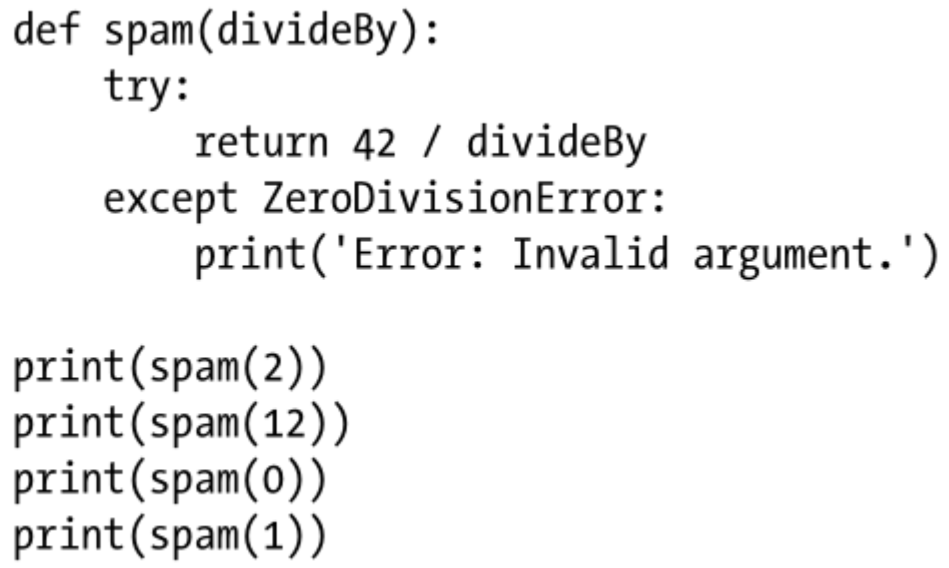
Exception Handling (Try/Except)

Make and run this program:



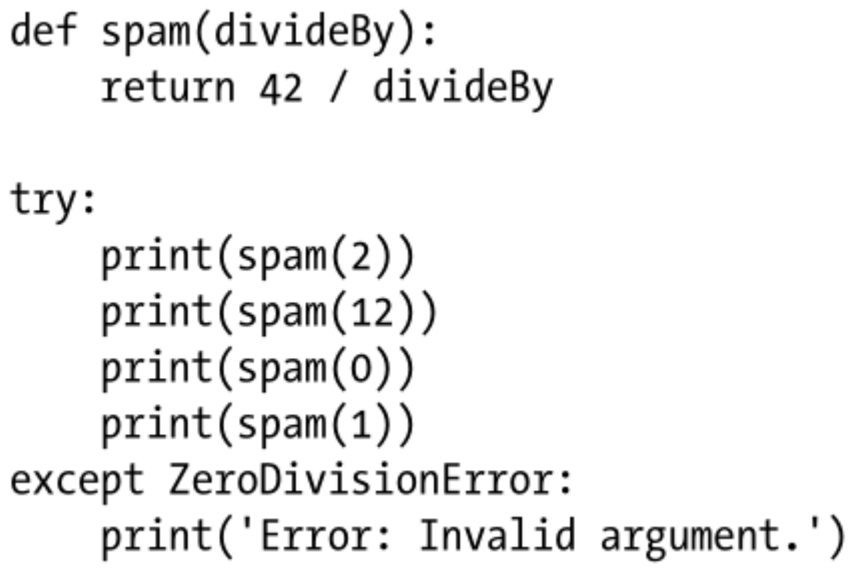
```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Then this program:

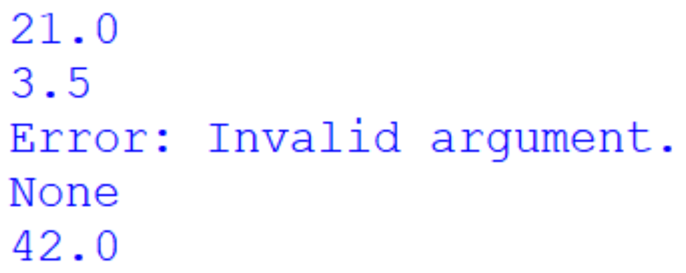


```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Try/Except



```
def spam(divideBy):
    return 42 / divideBy

try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid argument.')
```



```
21.0
3.5
Error: Invalid argument.
None
42.0
```

```
dog = input('Enter a number: ')
try:
   dog_check = int(dog)
except:
   print('That is not an integer')
```

Try/Except

If you don’t specify the kind of exception it will just catch all exceptions.

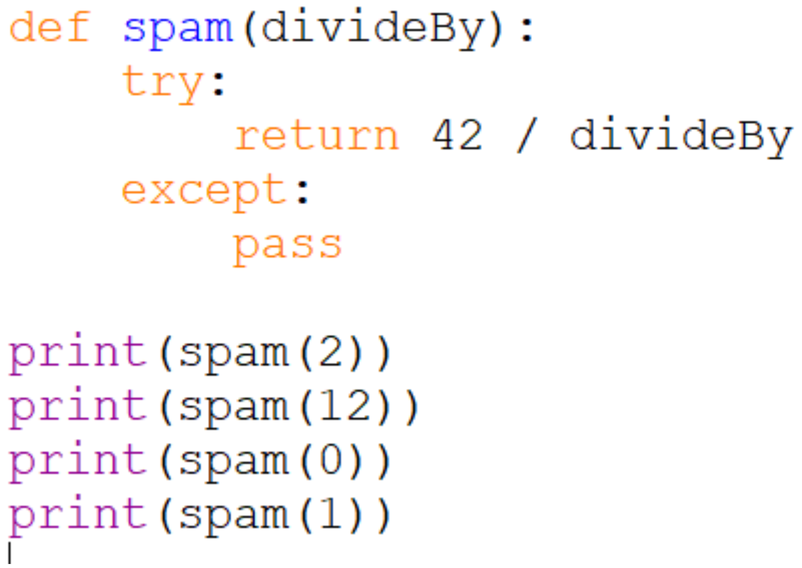
```
try:
    something
except:
    message
```

<- These are equivalent ->

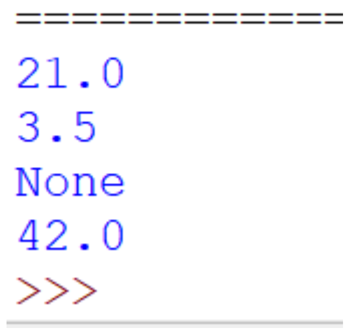
```
try:
    something
except Exception:
    message
```

Try/Except

You have to do something with the exception but you can just say pass



```python
def spam(divideBy):
    try:
        return 42 / divideBy
    except:
        pass

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```



```
=============
21.0
3.5
None
42.0
>>>
```

Error Exceptions

https://docs.python.org/3/library/exceptions.html#bltin-exceptions