Tuples
The **`tuple`** Object

- Tuples, like lists, are ordered sequences of items
- Difference – tuples cannot be modified in place
  - Have no **`append`**, **`extend`**, or **`insert`** method
- Items of tuple cannot be directly deleted, sorted, or altered

The `tuple` Object

- All other list functions and methods apply
  - Items can be accessed by indices
  - Tuples can be sliced, concatenated, and repeated
- Tuples written as comma-separated sequences enclosed in parentheses
  - Can also be written without the parentheses.

The `tuple` Object

- Example : Program shows tuples have several of same functions as lists.

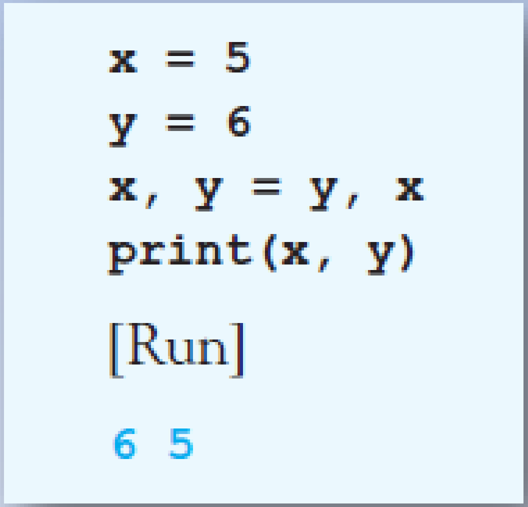
```
t = 5, 7, 6, 2
print(t)
print(len(t), max(t), min(t), sum(t))
print(t[0], t[-1], t[:2])
```

[Run]

```
(5, 7, 6, 2)
4 7 2 20
5 2 (5, 7)
```

The `tuple` Object

- Example: Program swaps values of two variables



```
x = 5
y = 6
x, y = y, x
print(x, y)
```

[Run]

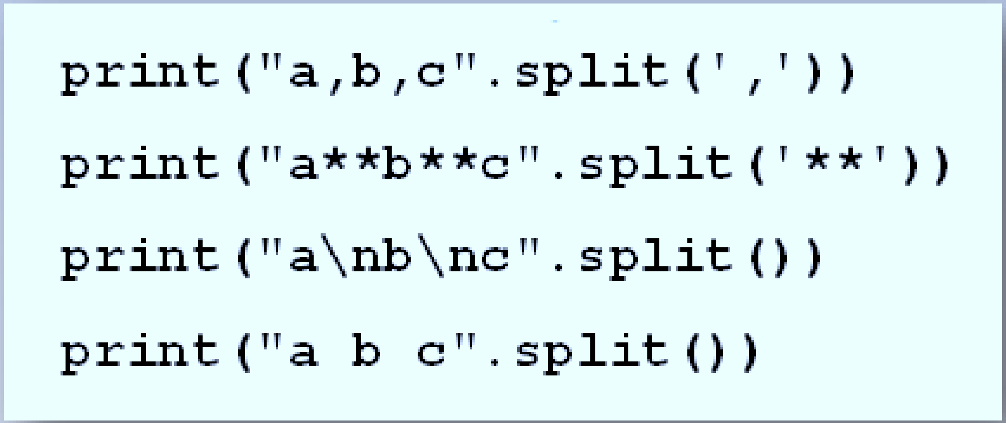
```
6 5
```

More on Lists
The `split` and `join` Methods

- Split method turns single string into list of substrings
- Join method turns a list of strings into a single string.
- Notice that these methods are inverses of each other

The **`split`** and **`join`** Methods

- Example: These statements each display list **['a', 'b', 'c']**.



```
print("a,b,c".split(','))
print("a**b**c".split('**'))
print("a\nb\nc".split())
print("a b c".split())
```

The **split** and **join** Methods

- Example: Program shows how **join** method used to display items from list of strings.

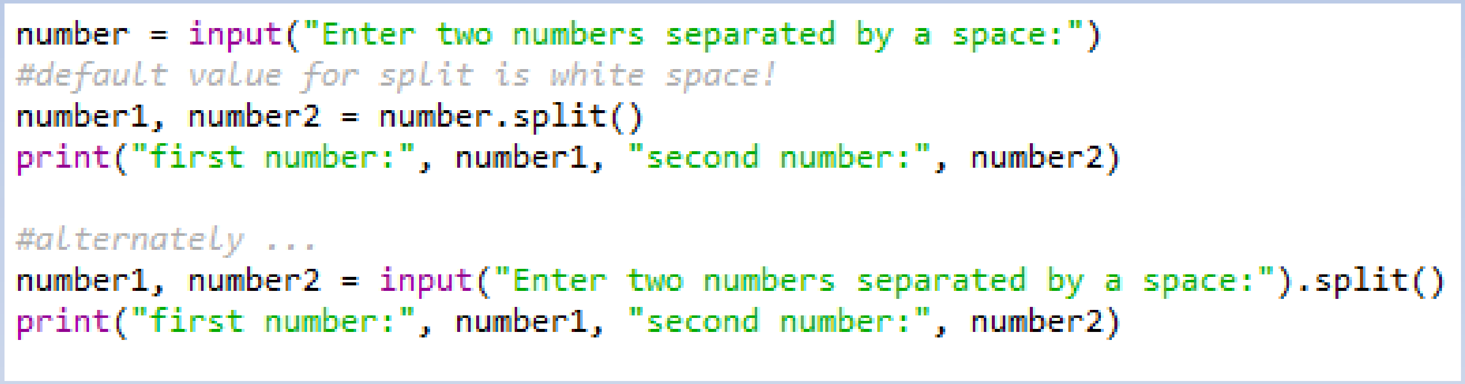
```
line = ["To", "be", "or", "not", "to", "be."]
print(" ".join(line))
krispies = ["Snap", "Crackle", "Pop"]
print(", ".join(krispies))
```

[Run]

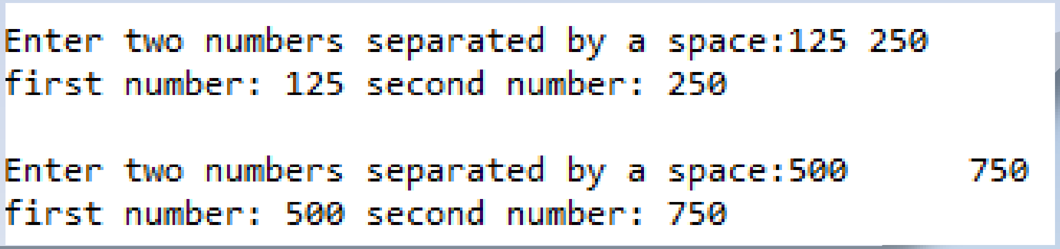
```
To be or not to be.
Snap, Crackle, Pop
```

`tuple` Object for accepting input

- Tuples are convenient when you want to accept multiple values as input all at once (separated by space)



```python
number = input("Enter two numbers separated by a space:")
#default value for split is white space!
number1, number2 = number.split()
print("first number:", number1, "second number:", number2)

#alternately ...
number1, number2 = input("Enter two numbers separated by a space:").split()
print("first number:", number1, "second number:", number2)
```



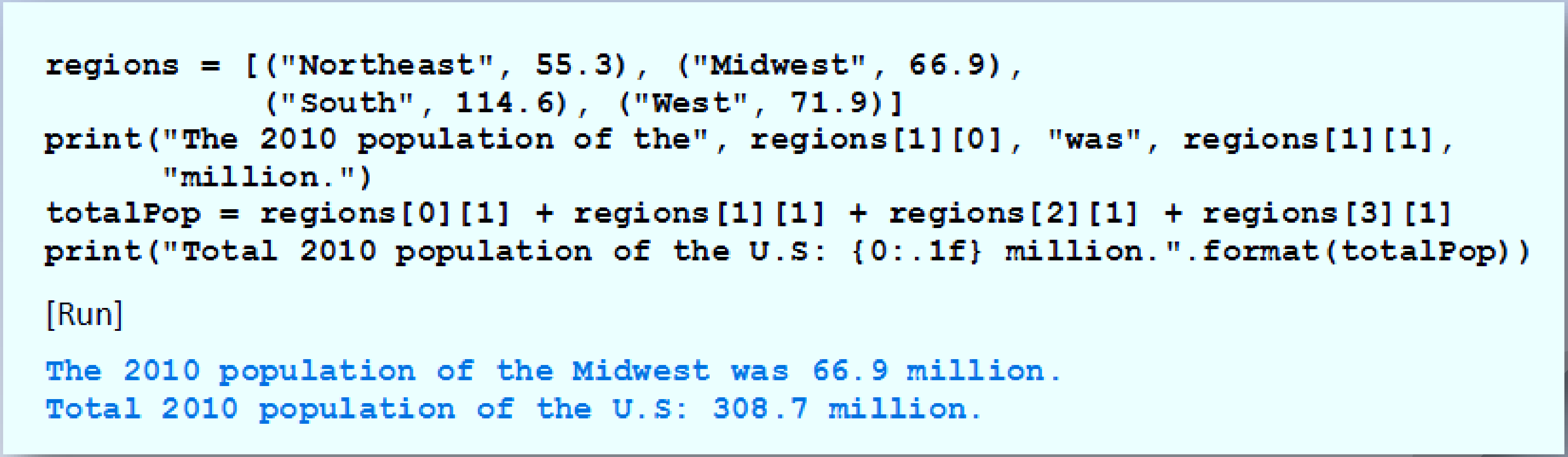
```
Enter two numbers separated by a space:125 250
first number: 125 second number: 250

Enter two numbers separated by a space:500     750
first number: 500 second number: 750
```

Nested Lists

- Beside numbers or strings, items can be lists or tuples.
- Consider a list of tuples named **`L`**
  `L[0]` is the first tuple
  `L[0][0]` is the first item in the first tuple
- And **`L[-1]`** is the last tuple
  `L[-1][-1]` is the last item in the last tuple

Nested Lists

- Example: Program manipulates *regions* contains four tuples, each tuple gives name and 2010 population (in millions) of a region



```
regions = [("Northeast", 55.3), ("Midwest", 66.9),
           ("South", 114.6), ("West", 71.9)]
print("The 2010 population of the", regions[1][0], "was", regions[1][1],
      "million.")
totalPop = regions[0][1] + regions[1][1] + regions[2][1] + regions[3][1]
print("Total 2010 population of the U.S: {0:.1f} million.".format(totalPop))
```

[Run]

```
The 2010 population of the Midwest was 66.9 million.
Total 2010 population of the U.S: 308.7 million.
```

Immutable and Mutable Objects

- An object is an entity
  - Holds data.
  - Has operations and/or methods that can manipulate the data.
- When variable created with assignment statement
  - Value on the right side becomes an object in memory
  - Variable references (points to) object

Immutable and Mutable Objects

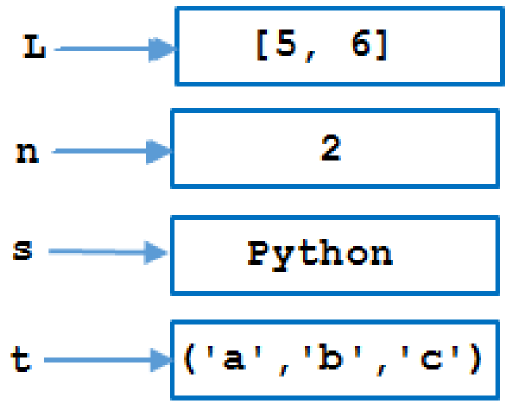
- When list altered
  - Changes made to the object in list's memory location
- Contrast when value of variable is number, string, or tuple … when value changed,
  - Python designates a new memory location to hold the new value
  - And the variable references that new object

Immutable and Mutable Objects

- Another way to say this
  - Lists can be changed in place
  - Numbers, strings, and tuples cannot
- Objects changed in place are *mutable*
- Objects that *cannot* be changed in place are *immutable*

Immutable and Mutable Objects

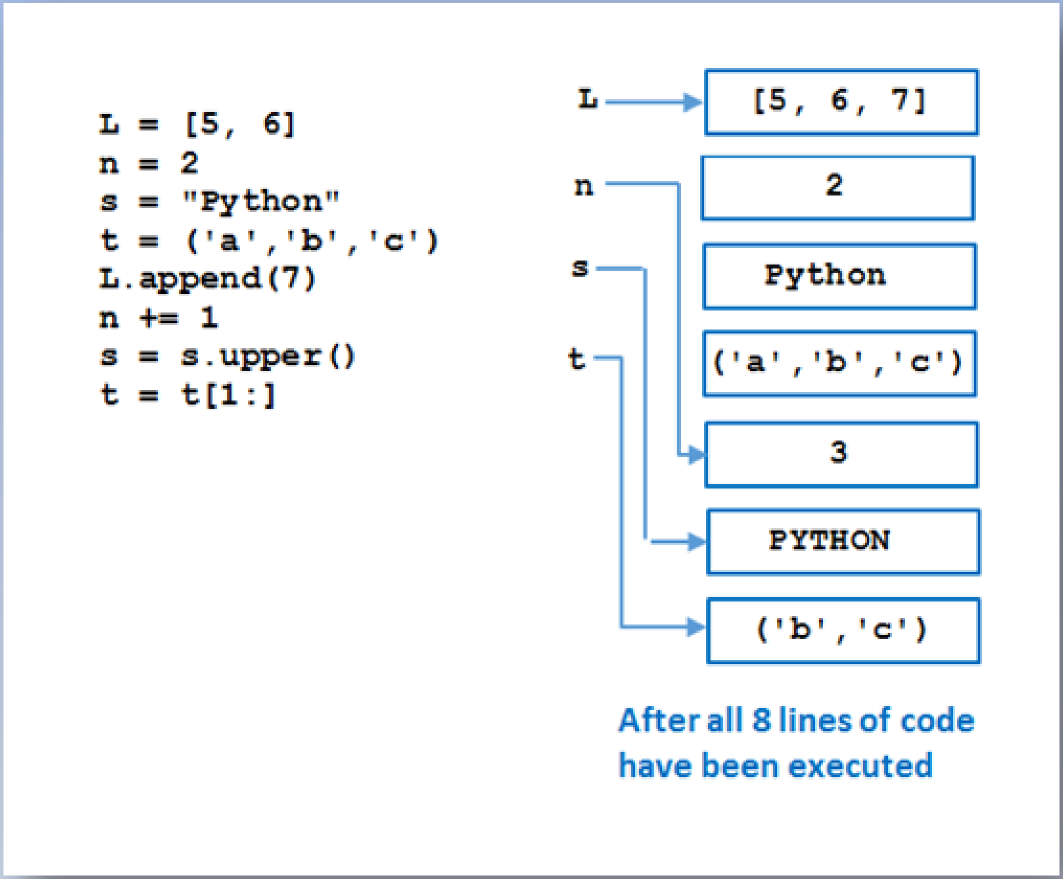
```
L = [5, 6]
n = 2
s = "Python"
t = ('a','b','c')
L.append(7)
n += 1
s = s.upper()
t = t[1:]
```



After first 4 lines of code have been executed

Memory allocation corresponding to a program.

Immutable and Mutable Objects

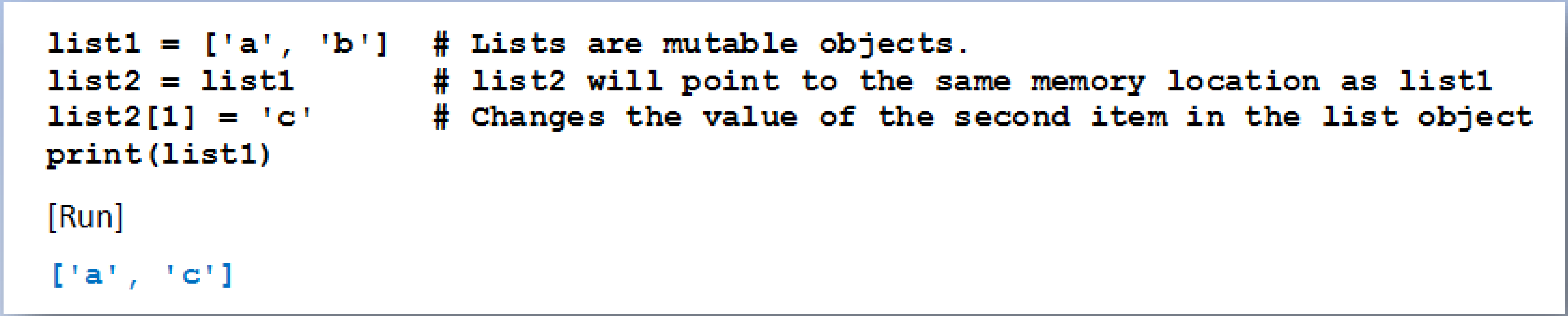


```
L = [5, 6]
n = 2
s = "Python"
t = ('a','b','c')
L.append(7)
n += 1
s = s.upper()
t = t[1:]
```

Memory allocation corresponding to a program.

Copying Lists

- Consider results of this program



```
list1 = ['a', 'b']  # Lists are mutable objects.
list2 = list1       # list2 will point to the same memory location as list1
list2[1] = 'c'      # Changes the value of the second item in the list object
print(list1)
```

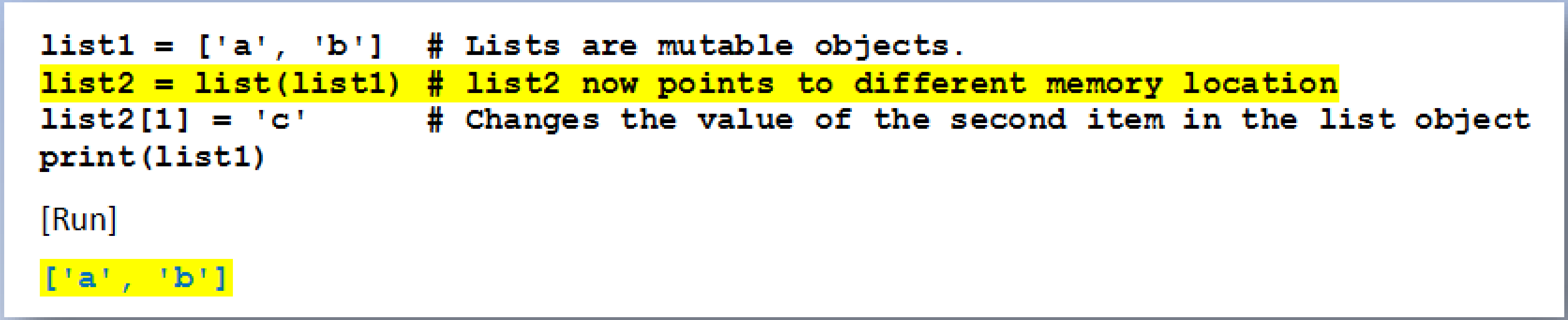
[Run]

```
['a', 'c']
```

- All because *lists are mutable*

Copying Lists

- Now note change in line 2



```
list1 = ['a', 'b']  # Lists are mutable objects.
list2 = list(list1) # list2 now points to different memory location
list2[1] = 'c'      # Changes the value of the second item in the list object
print(list1)
```

[Run]

```
['a', 'b']
```

- Third line of code will not affect memory location pointed to by *list1*

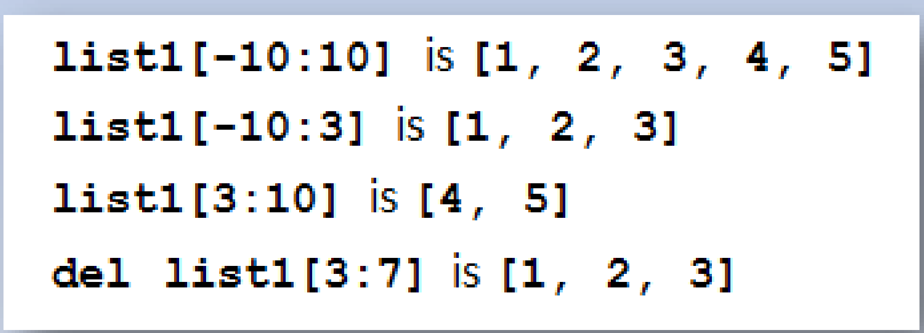
Indexing, Deleting, and Slicing Out of Bounds

- Python does *not* allow out of bounds indexing for individual items in lists and tuples
  - But *does* allow it for slices
- Given `list1 = [1, 2, 3, 4, 5]`
  - Then

```
print(list1[7])
print(list1[-7])
del list1[7]
```

Indexing, Deleting, and Slicing Out of Bounds

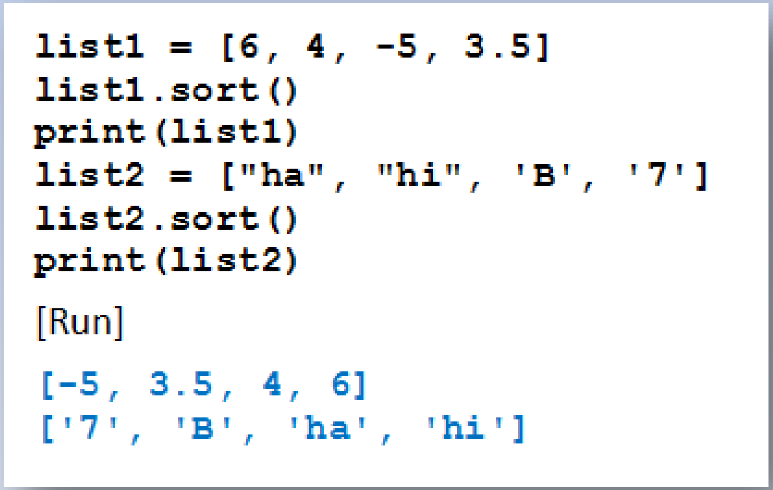
- If left index in slice too far negative
  - Slice will start at the beginning of the list
- If right index is too large,
  - Slice will go to the end of the list.



```
list1[-10:10] is [1, 2, 3, 4, 5]
list1[-10:3] is [1, 2, 3]
list1[3:10] is [4, 5]
del list1[3:7] is [1, 2, 3]
```

Sorting the Items in a List

- Items in a list of items having same data type can be ordered with the **`sort`** method
- Example: Program illustrates how Python orders two simple lists



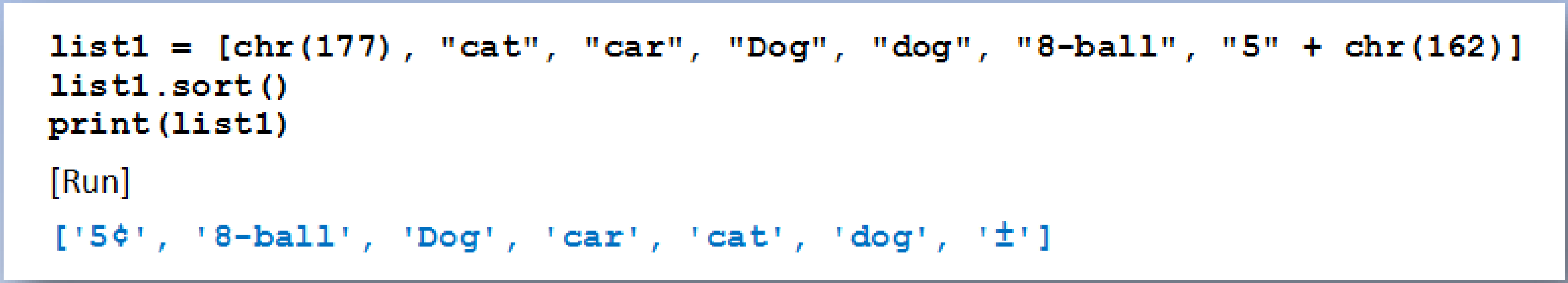
```
list1 = [6, 4, -5, 3.5]
list1.sort()
print(list1)
list2 = ["ha", "hi", 'B', '7']
list2.sort()
print(list2)
```

[Run]

```
[-5, 3.5, 4, 6]
['7', 'B', 'ha', 'hi']
```

Sorting the Items in a List

- Example: Items in a complicated list of strings.

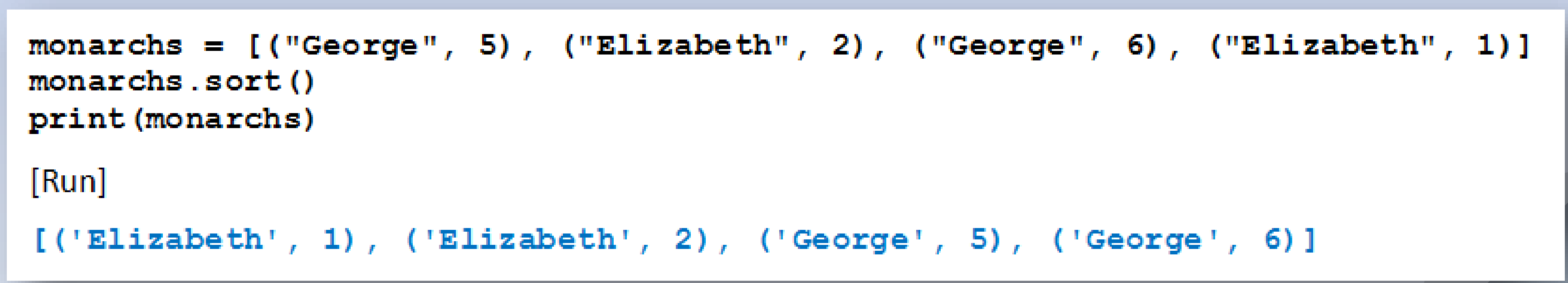


```
list1 = [chr(177), "cat", "car", "Dog", "dog", "8-ball", "5" + chr(162)]
list1.sort()
print(list1)
```

[Run]

```
['5¢', '8-ball', 'Dog', 'car', 'cat', 'dog', '±']
```

- Example: Items in a list of tuples



```
monarchs = [("George", 5), ("Elizabeth", 2), ("George", 6), ("Elizabeth", 1)]
monarchs.sort()
print(monarchs)
```

[Run]

```
[('Elizabeth', 1), ('Elizabeth', 2), ('George', 5), ('George', 6)]
```