

Lists, tuples, and sets

Chapter 5

This chapter covers

- Manipulating lists and list indices
- Modifying lists
- Sorting
- Using common list operations
- Handling nested lists and deep copies
- Using tuples
- Creating and using sets

```
1  # This assigns a three-element list to x
2  x = [1, 2, 3]
3
4  # First element is a number, second is a string,
5  # third is another list.
6  x = [2, "two", [1, 2, 3]]
7
8  # get the length/size of list
9  print(len(x))
```

Lists are
like
arrays

- A list in Python is an ordered collection of objects.
- You create a list by enclosing a comma-separated list of elements in square brackets.
- Python lists can contain different types of elements; a list element can be any Python object.
- Probably the most basic built-in list function is the len function, which returns the number of elements in a list

```
11 x = ["first", "second", "third", "fourth"]
12 print(x[0])
13 print(x[2])
14
15 a = x[-1]
16 print(a)
17 print(x[-2])
```

- Python list index start with 0.
- If indices are negative numbers, they indicate positions counting from the end of the list, with -1 being the last position in the list, -2 being the second-to-last position, and so forth.

List indices

```
19 x = ["first", "second", "third", "fourth"]
20 y = x[1:-1]
21 print(y)
22
23 y = x[0:3]
24 print(y)
25
26 y = x[-2:-1]
27 print(y)
28
29 print(x[:3])
30 print(x[2:])
```

Slicing

Creating a subset from the list.



TRY THIS

- Using what you know about the `len()` function and list slices, how would you combine the two to get the second half of a list when you don't know what size it is?
- Experiment in the Python shell to confirm that your solution works.

Modifying lists

- Add
- Append
- Remove
- Extend
- Insert
- Delete
- Reverse

```
36
37 # append
38 x[len(x):] = [5, 6, 7]
39 print(x)
40
41 x[:0] = [-1, 0] # append from front
42 x[1:-1] = []    # remove/clear
43
44 x = [1, 2, 3, 4]
45 y = [5, 6, 7]
46 x.append(y)      # [1, 2, 3, 4, [5, 6, 7]]
47 x.extend(y)      # [1, 2, 3, 4, 5, 6, 7]
48 x.insert(2, "hello")
49
50 del x[1]
```



TRY THIS

- Suppose that you have a list 10 items long.
- How might you move the last three items from the end of the list to the beginning, keeping them in the same order?

Sorting lists

- Lists can be sorted by using the built-in Python sort method.
- To sort a list without changing the original list, you have two options.
 - use the sorted() built-in function,
 - make a copy of the list and sort the copy
- According to the built-in Python rules for comparing complex objects, the sublists are sorted first by ascending first element and then by ascending second element.

```
1  # in-place sorting (change the list)
2  x = [3, 8, 4, 0, 2, 1]
3  x.sort()
4
5  # make copy then sort
6  x = [2, 4, 1, 3]
7  y = x[:]
8  y.sort()
9
10 # strings
11 x = ["Life", "Is", "Enchanting"]
12 x.sort()
13
14 # list of list
15 x = [[3, 5], [2, 9], [2, 3], [4, 1], [3, 2]]
16 x.sort()
```

```
18  # sorted
19  x = (4, 3, 1, 2)
20  y = sorted(x)
21
22  z = sorted(x, reverse=True)
```

The sorted() function

- Python also has the built-in function sorted(), which returns a sorted list from any iterable.
- sorted() uses the same key and reverse parameters as the sort method.



TRY THIS

- Suppose that you have a list in which each element is in turn a list: `[[1, 2, 3], [2, 1, 3], [4, 0, 1]]`.
- If you wanted to sort this list by the second element in each list so that the result would be `[[4, 0, 1], [2, 1, 3], [1, 2, 3]]`, what function would you write to pass as the `key` value to the `sort()` method?

Other common list operations

- List membership with the in operator
- List concatenation with the + operator
- List initialization with the * operator
- List minimum or maximum with min and max
- List search with index
- List matches with count

```
1  # in operator
2  print(3 in [1, 3, 4, 5])
3  print(3 in ["one", "two", "three"])
4  print(3 not in ["one", "two", "three"])
5
6  # + operator
7  print(z = [1, 2, 3] + [4, 5])
8
9  # * operator
10 z = [None] * 4
11 z = [3, 1] * 2
12
13 # min and max
14 print(min([3, 7, 0, -2, 11]))
15 print(max([3, 7, 0, -2, 11]))
16
17 # index
18 x = [1, 3, "five", 7, -2]
19 print(x.index(7))
20
21 # count
22 x = [1, 2, 2, 3, 5, 2, 5]
23 print(x.count(2))
```



TRY THIS

- What would be the result of `len([[1,2]] * 3)`?
- What are two differences between using the `in` operator and a list's `index()` method?
- Which of the following will raise an exception?:

```
min(["a", "b", "c"]);  
max([1, 2, "three"]); [1, 2, 3].count("one")
```



TRY THIS

- If you have a list `x`, write the code to safely remove an item if—and only if - that value is in the list.
- Modify that code to remove the element only if the item occurs in the list more than once.

Nested lists and deep copies

```
1  m = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]
2
3  print(m[0])
4  print(m[0][1])
5  print(m[2])
6  print(m[2][2])
```

- Lists can be nested.
- One application of nesting is to represent two-dimensional matrices.
- The members of these matrices can be referred to by using two-dimensional indices.

Tuples



Tuples are data structures that are very similar to lists, but they can't be modified; they can only be created.



Tuples are so much like lists that you may wonder why Python bothers to include them.



The reason is that tuples have important roles that can't be efficiently filled by lists, such as keys for dictionaries.

Tuple basics

- Creating a tuple is similar to creating a list: assign a sequence of values to a variable.
- A list is a sequence that's enclosed by [and]; a tuple is a sequence that's enclosed by (and)

```
1  x = ('a', 'b', 'c')
2
3  print(x[2])
4  print(x[1:])
5  print(max(x))
6  print(min(x))
7  print(5 in x)
8  print(5 not in x)
9
10 x[2] = 'd'
11
12 print(x + x)
13 print(2 * x)
```

Sets



A set in Python is an unordered collection of objects used when membership and uniqueness in the set are main things you need to know about that object.



Like dictionary keys (discussed in chapter 7), the items in a set must be immutable and hashable.



This means that ints, floats, strings, and tuples can be members of a set, but lists, dictionaries, and sets themselves can't.

Set operations

- Create
- Add
- Remove
- In operator
- Convert list to set
- Logical operations

```
1  x = set([1, 2, 3, 1, 3, 5])
2
3  x.add(6)
4
5  x.remove(5)
6
7  print(1 in x)
8  print(4 in x)
9
10 y = set([1, 7, 8, 9])
11
12 print(x | y)
13 print(x & y)
14 print(x ^ y)
```



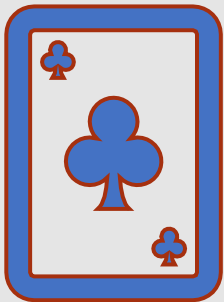
TRY THIS

- If you were to construct a set from the following list, how many elements would the set have?: [1, 2, 5, 1, 0, 2, 3, 1, 1, (1, 2, 3)] ?

Lab

Examining a List

Summary



- Lists and tuples are structures that embody the idea of a sequence of elements, as are strings.
- Lists are like arrays in other languages, but with automatic resizing, slice notation, and many convenience functions.
- Tuples are like lists but can't be modified, so they use less memory and can be dictionary keys (see chapter 7).
- Sets are iterable collections, but they're unordered and can't have duplicate elements.