

## Chapter 8

# More with Lists

### 8.1 Lists and the random module

There are some nice functions in the `random` module that work on lists.

| Function                  | Description  |
|---------------------------|--|
| <code>choice(L)</code>    | picks a random item from <code>L</code>                          |
| <code>sample(L, n)</code> | picks a group of <code>n</code> random items from <code>L</code> |
| <code>shuffle(L)</code>   | Shuffles the items of <code>L</code>                             |

**Note** The `shuffle` function modifies the original list, so if you don't want your list changed, you'll need to make a copy of it.

**Example 1** We can use `choice` to pick a name from a list of names.

```
from random import choice
names = ['Joe', 'Bob', 'Sue', 'Sally']
current_player = choice(names)
```

**Example 2** The `sample` function is similar to `choice`. Whereas `choice` picks one item from a list, `sample` can be used to pick several.

```
from random import sample
names = ['Joe', 'Bob', 'Sue', 'Sally']
team = sample(names, 2)
```

**Example 3** The `choice` function also works with strings, picking a random character from a string. Here is an example that uses `choice` to fill the screen with a bunch of random characters.

```
from random import choice
s='abcdefghijklmnopqrstuvwxyz1234567890!@#$%^&*() '
for i in range(10000):
    print(choice(s), end='')
```

**Example 4** Here is a nice use of `shuffle` to pick a random ordering of players in a game.

```
from random import shuffle
players = ['Joe', 'Bob', 'Sue', 'Sally']
shuffle(players)
for p in players:
    print(p, 'it is your turn.')
    # code to play the game goes here...
```

**Example 5** Here we use `shuffle` divide a group of people into teams of two. Assume we are given a list called `names`.

```
shuffle(names)
teams = []
for i in range(0, len(names), 2):
    teams.append([names[i], names[i+1]])
```

Each item in `teams` is a list of two names. The way the code works is we shuffle the names so they are in a random order. The first two names in the shuffled list become the first team, the next two names become the second team, etc. Notice that we use the optional third argument to `range` to skip ahead by two through the list of names.

## 8.2 split

The `split` method returns a list of the words of a string. The method assumes that words are separated by whitespace, which can be either spaces, tabs or newline characters. Here is an example:

```
s = 'Hi! This is a test.'
print(s.split())
```

```
['Hi!', 'This', 'is', 'a', 'test.']
```

As we can see, since `split` breaks up the string at spaces, the punctuation will be part of the words. There is a module called `string` that contains, among other things, a string variable called `punctuation` that contains common punctuation. We can remove the punctuation from a string `s` with the following code:

```
from string import punctuation
for c in punctuation:
    s = s.replace(c, '')
```

**Example** Here is a program that counts how many times a certain word occurs in a string.

```
from string import punctuation

s = input('Enter a string: ')
for c in punctuation:
    s = s.replace(c, '')
s = s.lower()
L = s.split()

word = input('Enter a word: ')
print(word, 'appears', L.count(word), 'times.')
```

**Optional argument** The `split` method takes an optional argument that allows it to break the string at places other than spaces. Here is an example:

```
s = '1-800-271-8281'
print(s.split('-'))
```

```
['1', '800', '271', '8281']
```

## 8.3 join

The `join` method is in some sense the opposite of `split`. It is a string method that takes a list of strings and joins them together into a single string. Here are some examples, using the list `L = ['A', 'B', 'C']`

| Operation     | Result    |
|---------------|-----------|
| ' '.join(L)   | A B C     |
| ''.join(L)    | ABC       |
| ', '.join(L)  | A, B, C   |
| '***'.join(L) | A***B***C |

**Example** Write a program that creates an anagram of a given word. An anagram of a word uses the same letters as the word but in a different order. For instance, two anagrams of the word *there* are *three* and *ether*. Don't worry about whether the anagram is a real word or not.

This sounds like something we could use `shuffle` for, but `shuffle` only works with lists. What we need to do is convert our string into a list, use `shuffle` on it, and then convert the list back into a string. To turn a string `s` into a list, we can use `list(s)`. (See Section 10.1.) To turn the list back into a string, we will use `join`.

```
from random import shuffle
word = input('Enter a word: ')

letter_list = list(word)
shuffle(letter_list)
anagram = ''.join(letter_list)

print(anagram)
```

## 8.4 List comprehensions

List comprehensions are a powerful way to create lists. Here is a simple example:

```
L = [i for i in range(5)]
```

This creates the list `[0, 1, 2, 3, 4]`. Notice that the syntax of a list comprehension is somewhat reminiscent of set notation in mathematics. Here are a couple more examples of list comprehensions. For these examples, assume the following:

```
string = 'Hello'
L = [1, 14, 5, 9, 12]
M = ['one', 'two', 'three', 'four', 'five', 'six']
```

| List comprehension                            | Resulting list                              |
|---|---|
| <code>[0 for i in range(10)]</code>           | <code>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]</code> |
| <code>[i**2 for i in range(1, 8)]</code>      | <code>[1, 4, 9, 16, 25, 36, 49]</code>      |
| <code>[i*10 for i in L]</code>                | <code>[10, 140, 50, 90, 120]</code>         |
| <code>[c*2 for c in string]</code>            | <code>['HH', 'ee', 'll', 'll', 'oo']</code> |
| <code>[m[0] for m in M]</code>                | <code>['o', 't', 't', 'f', 'f', 's']</code> |
| <code>[i for i in L if i &lt; 10]</code>      | <code>[1, 5, 9]</code>                      |
| <code>[m[0] for m in M if len(m) == 3]</code> | <code>['o', 't', 's']</code>                |

As we see in the last two examples, we can add an `if` to a list comprehension. Compare the last example with the long way of building the list:

```
L = []
for m in M:
    if len(m) == 3:
        L.append(m)
```

**Multiple fors** You can use more than one `for` in a list comprehension:

```
L = [[i, j] for i in range(2) for j in range(2)]
```

```
[ [0, 0], [0, 1], [1, 0], [1, 1] ]
```

This is the equivalent of the following code:

```
L = []
for i in range(2):
    for j in range(2):
        L.append([i, j])
```

Here is another example:

```
[[i, j] for i in range(4) for j in range(i)]
```

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

## 8.5 Using list comprehensions

To further demonstrate the power of list comprehensions, we will do the first four examples of Section 7.6 in one line apiece using list comprehensions.

**Example 1** Write a program that generates a list *L* of 50 random numbers between 1 and 100.

```
L = [randint(1,100) for i in range(50)]
```

**Example 2** Replace each element in a list *L* with its square.

```
L = [i**2 for i in L]
```

**Example 3** Count how many items in a list *L* are greater than 50.

```
len([i for i in L if i>50])
```

**Example 4** Given a list *L* that contains numbers between 1 and 100, create a new list whose first element is how many ones are in *L*, whose second element is how many twos are in *L*, etc.

```
frequencies = [L.count(i) for i in range(1,101)]
```

**Another example** The `join` method can often be used with list comprehensions to quickly build up a string. Here we create a string that contains a random assortment of 1000 letters.

```
from random import choice
alphabet = 'abcdefghijklmnopqrstuvwxyz'
s = ''.join([choice(alphabet) for i in range(1000)])
```

**One more example** Suppose we have a list whose elements are lists of size 2, like below:

```
L = [[1, 2], [3, 4], [5, 6]]
```

If we want to flip the order of the entries in the lists, we can use the following list comprehension:

```
M = [[y, x] for x, y in L]
```

```
[[2, 1], [4, 3], [6, 5]]
```

**Note** You can certainly get away without using list comprehensions, but once you get the hang of them, you'll find they are both quicker to write and easier to read than the longer ways of creating lists.

## 8.6 Two-dimensional lists

There are a number of common things that can be represented by two-dimensional lists, like a Tic-tac-toe board or the pixels on a computer screen. In Python, one way to create a two-dimensional list is to create a list whose items are themselves lists. Here is an example:

```
L = [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]
```

**Indexing** We use two indices to access individual items. To get the entry in row *r*, column *c*, use the following:

```
L[r][c]
```

**Printing a two-dimensional list** To print a two-dimensional list, you can use nested for loops. The following example prints a 10 × 5 list:

```
for r in range(10):
    for c in range(5):
        print(L[r][c], end=" ")
    print()
```

Another option is to use the `pprint` function of the `pprint` module. This function is used to “pretty-print” its argument. Here is an example to print a list *L*:

```
from pprint import pprint
pprint(L)
```

The `pprint` function can be used to nicely print ordinary lists and other objects in Python.

**Working with two-dimensional lists** Nested for loops, like the ones used in printing a two-dimensional list, can also be used to process the items in a two-dimensional list. Here is an example that counts how many entries in a 10 × 5 list are even.

```
count = 0
for r in range(10):
    for c in range(5):
        if L[r][c]%2==0:
            count = count + 1
```

This can also be done with a list comprehension:

```
count = sum([1 for r in range(10) for c in range(5) if L[r][c]%2==0])
```

**Creating large two-dimensional lists** To create a larger list, you can use a list comprehension like below:

```
L = [[0]*50 for i in range(100)]
```

This creates a list of zeroes with 100 rows and 50 columns.

**Picking out rows and columns** To get the  $r$ th row of  $L$ , use the following:

```
L[r]
```

To get the  $c$ th column of  $L$ , use a list comprehension:

```
[L[i][c] for i in range(len(L))]
```

**Flattening a list** To flatten a two-dimensional list, that is, return a one-dimensional list of its elements, use the following:

```
[j for M in L for j in M]
```

For instance, suppose we have the following list:

```
L = [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]
```

The flattened list will be:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

**Higher dimensions** Creating and using 3-dimensional and higher lists is similar. Here we create a  $5 \times 5 \times 5$  list:

```
L = [[[0]*5 for i in range(5)] for j in range(5)]
```

It is a list whose items are lists of lists. The first entry in the list is

```
L[0][0][0]
```

---

## 8.7 Exercises

1. Write a program that asks the user to enter some text and then counts how many articles are in the text. Articles are the words 'a', 'an', and 'the'.
2. Write a program that allows the user to enter five numbers (read as strings). Create a string that consists of the user's numbers separated by plus signs. For instance, if the user enters 2, 5, 11, 33, and 55, then the string should be '2+5+11+33+55'.
3. (a) Ask the user to enter a sentence and print out the third word of the sentence.  
(b) Ask the user to enter a sentence and print out every third word of the sentence.
4. (a) Write a program that asks the user to enter a sentence and then randomly rearranges the words of the sentence. Don't worry about getting punctuation or capitalization correct.  
(b) Do the above problem, but now make sure that the sentence starts with a capital, that the original first word is not capitalized if it comes in the middle of the sentence, and that the period is in the right place.
5. Write a simple quote-of-the-day program. The program should contain a list of quotes, and when the user runs the program, a randomly selected quote should be printed.
6. Write a simple lottery drawing program. The lottery drawing should consist of six different numbers between 1 and 48.
7. Write a program that estimates the average number of drawings it takes before the user's numbers are picked in a lottery that consists of correctly picking six different numbers that are between 1 and 10. To do this, run a loop 1000 times that randomly generates a set of user numbers and simulates drawings until the user's numbers are drawn. Find the average number of drawings needed over the 1000 times the loop runs.
8. Write a program that simulates drawing names out of a hat. In this drawing, the number of hat entries each person gets may vary. Allow the user to input a list of names and a list of how many entries each person has in the drawing, and print out who wins the drawing.
9. Write a simple quiz game that has a list of ten questions and a list of answers to those questions. The game should give the player four randomly selected questions to answer. It should ask the questions one-by-one, and tell the player whether they got the question right or wrong. At the end it should print out how many out of four they got right.
10. Write a censoring program. Allow the user to enter some text and your program should print out the text with all the curse words starred out. The number of stars should match the length of the curse word. For the purposes of this program, just use the "curse" words *darn*, *dang*, *freakin*, *heck*, and *shoot*. Sample output is below:

```
Enter some text: Oh shoot, I thought I had the dang problem
figured out. Darn it. Oh well, it was a heck of a freakin try.
```

```
Oh *****, I thought I had the **** problem figured out.
**** it. Oh well, it was a **** of a ***** try.
```



11. Section 8.3 described how to use the `shuffle` method to create a random anagram of a string. Use the `choice` method to create a random anagram of a string.
12. Write a program that gets a string from the user containing a potential telephone number. The program should print `Valid` if it decides the phone number is a real phone number, and `Invalid` otherwise. A phone number is considered valid as long as it is written in the form *abc-def-hijk* or *1-abc-def-hijk*. The dashes must be included, the phone number should contain only numbers and dashes, and the number of digits in each group must be correct. Test your program with the output shown below.

```
Enter a phone number: 1-301-447-5820
Valid
Enter a phone number: 301-447-5820
Valid
Enter a phone number: 301-4477-5820
Invalid
Enter a phone number: 3X1-447-5820
Invalid
Enter a phone number: 3014475820
Invalid
```

13. Let `L` be a list of strings. Write list comprehensions that create new lists from `L` for each of the following.
  - (a) A list that consists of the strings of `s` with their first characters removed
  - (b) A list of the lengths of the strings of `s`
  - (c) A list that consists of only those strings of `s` that are at least three characters long
14. Use a list comprehension to produce a list that consists of all palindromic numbers between 100 and 1000.
15. Use a list comprehension to create the list below, which consists of ones separated by increasingly many zeroes. The last two ones in the list should be separated by ten zeroes.
 

```
[1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, . . . .]
```
16. Let `L=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]`. Use a list comprehension to produce a list of the gaps between consecutive entries in `L`. Then find the maximum gap size and the percentage of gaps that have size 2.
17. Write a program that finds the average of all of the entries in a  $4 \times 4$  list of integers.
18. Write a program that creates a  $10 \times 10$  list of random integers between 1 and 100. Then do the following:
  - (a) Print the list.
  - (b) Find the largest value in the third row.
  - (c) Find the smallest value in the sixth column.

19. Write a program that creates and prints an  $8 \times 8$  list whose entries alternate between 1 and 2 in a checkerboard pattern, starting with 1 in the upper left corner.
20. Write a program that checks to see if a  $4 \times 4$  list is a magic square. In a magic square, every row, column, and the two diagonals add up to the same value.
21. Write a program that asks the user to enter a length. The program should ask them what unit the length is in and what unit they would like to convert it to. The possible units are inches, yards, miles, millimeters, centimeters, meters, and kilometers. While this can be done with 25 if statements, it is shorter and easier to add on to if you use a two-dimensional list of conversions, so please use lists for this problem.
22. The following is useful as part of a program to play *Battleship*. Suppose you have a  $5 \times 5$  list that consists of zeroes and ones. Ask the user to enter a row and a column. If the entry in the list at that row and column is a one, the program should print `Hit` and otherwise it should print `Miss`.
23. This exercise is useful in creating a *Memory* game. Randomly generate a  $6 \times 6$  list of assorted characters such that there are exactly two of each character. An example is shown below.

```
@ 5 # A A !
5 0 b @ $ z
$ N x ! N z
0 - + # b :
- : + c c x
```

24. The following is useful in implementing computer players in a number of different games. Write a program that creates a  $5 \times 5$  list consisting of zeroes and ones. Your program should then pick a random location in the list that contains a zero and change it to a one. If all the entries are one, the program should say so. [Hint: one way to do this is to create a new list whose items are the coordinates of all the ones in the list and use the `choice` method to randomly select one. Use a two-element list to represent a set of coordinates.]
25. Here is an old puzzle question you can solve with a computer program. There is only one five-digit number  $n$  that is such that every one of the following ten numbers shares exactly one digit in common in the same position as  $n$ . Find  $n$ .

01265, 12171, 23257, 34548, 45970, 56236, 67324, 78084, 89872, 99414

26. We usually refer to the entries of a two-dimensional list by their row and column, like below on the left. Another way is shown below on the right.

```
(0, 0) (0, 1) (0, 2)    0  1  2
(1, 0) (1, 1) (1, 2)    3  4  5
(2, 0) (2, 1) (2, 2)    6  7  8
```

- (a) Write some code that translates from the left representation to the right one. The `//` and `%` operators will be useful. Be sure your code works for arrays of any size.
- (b) Write some code that translates from the right representation to the left one.