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
`choice(L)`	picks a random item from `L`
`sample(L,n)`	picks a group of `n` random items from `L`
`shuffle(L)`	Shuffles the items of `L`

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
`[0 for i in range(10)]`	`[0,0,0,0,0,0,0,0,0,0]`
`[i**2 for i in range(1,8)]`	`[1,4,9,16,25,36,49]`
`[i*10 for i in L]`	`[10,140,50,90,120]`
`[c*2 for c in string]`	`['HH','ee','ll','ll','oo']`
`[m[0] for m in M]`	`['o','t','t','f','f','s']`
`[i for i in L if i<10]`	`[1,5,9]`
`[m[0] for m in M if len(m)==3]`	`['o','t','s']`

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 rth row of `L`, use the following:

```
L[r]
```

To get the cth 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]
```