Chapter 7

Lists

Say we need to get thirty test scores from a user and and do something with them, like put them in order. We could create thirty variables, `score1`, `score2`, ..., `score30`, but that would be very tedious. To then put the scores in order would be extremely difficult. The solution is to use lists.

7.1 Basics

Creating lists Here is a simple list:

```
L = [1,2,3]
```

Use square brackets to indicate the start and end of the list, and separate the items by commas.

The empty list The empty list is `[]`. It is the list equivalent of 0 or `''`.

Long lists If you have a long list to enter, you can split it across several lines, like below:

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
        32, 33, 34, 35, 36, 37, 38, 39, 40]
```

Input We can use `eval(input())` to allow the user to enter a list. Here is an example:

```
L = eval(input('Enter a list: '))
print('The first element is ', L[0])
```

```
Enter a list: [5,7,9]
The first element is 5
```

Printing lists You can use the `print` function to print the entire contents of a list.

```
L = [1,2,3]
print(L)
```

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

Data types Lists can contain all kinds of things, even other lists. For example, the following is a valid list:

```
[1, 2.718, 'abc', [5,6,7]]
```

7.2 Similarities to strings

There are a number of things which work the same way for lists as for strings.

- `len` — The number of items in `L` is given by `len(L)`.
- `in` — The `in` operator tells you if a list contains something. Here are some examples:

```
if 2 in L:
    print('Your list contains the number 2.')
if 0 not in L:
    print('Your list has no zeroes.')
```

- Indexing and slicing — These work exactly as with strings. For example, `L[0]` is the first item of the list `L` and `L[:3]` gives the first three items.
- `index` and `count` — These methods work the same as they do for strings.
- `+` and `*` — The `+` operator adds one list to the end of another. The `*` operator repeats a list. Here are some examples:

Expression	Result
`[7,8]+[3,4,5]`	`[7,8,3,4,5]`
`[7,8]*3`	`[7,8,7,8,7,8]`
`[0]*5`	`[0,0,0,0,0]`

  The last example is particularly useful for quickly creating a list of zeroes.

- Looping — The same two types of loops that work for strings also work for lists. Both of the following examples print out the items of a list, one-by-one, on separate lines.

```
for i in range(len(L)):
    print(L[i])
```

```
for item in L:
    print(item)
```

  The left loop is useful for problems where you need to use the loop variable `i` to keep track of where you are in the loop. If that is not needed, then use the right loop, as it is a little simpler.

7.3 Built-in functions

There are several built-in functions that operate on lists. Here are some useful ones:

Function	Description
`len`	returns the number of items in the list
`sum`	returns the sum of the items in the list
`min`	returns the minimum of the items in the list
`max`	returns the maximum of the items in the list

For example, the following computes the average of the values in a list `L`:

```
average = sum(L)/len(L)
```

7.4 List methods

Here are some list methods:

Method	Description
`append(x)`	adds `x` to the end of the list
`sort()`	sorts the list
`count(x)`	returns the number of times `x` occurs in the list
`index(x)`	returns the location of the first occurrence of `x`
`reverse()`	reverses the list
`remove(x)`	removes first occurrence of `x` from the list
`pop(p)`	removes the item at index `p` and returns its value
`insert(p,x)`	inserts `x` at index `p` of the list

Important note There is a big difference between list methods and string methods: String methods do not change the original string, but list methods do change the original list. To sort a list `L`, just use `L.sort()` and not `L=L.sort()`. In fact, the latter will not work at all.

wrong	*right*
`s.replace('X','x')`	`s = s.replace('X','x')`
`L = L.sort()`	`L.sort()`

Other list methods There are a few others list methods. Type `help(list)` in the Python shell to see some documentation for them.

7.5 Miscellaneous

Making copies of lists Making copies of lists is a little tricky due to the way Python handles lists. Say we have a list `L` and we want to make a copy of the list and call it `M`. The expression `M=L` will not work for reasons covered in Section 19.1. For now, do the following in place of `M=L`:

```
M = L[:]
```

Changing lists Changing a specific item in a list is easier than with strings. To change the value in location 2 of `L` to 100, we simply say `L[2]=100`. If we want to insert the value 100 into location 2 without overwriting what is currently there, we can use the `insert` method. To delete an entry from a list, we can use the `del` operator. Some examples are shown below. Assume `L=[6,7,8]` for each operation.

Operation	New `L`	Description
`L[1]=9`	`[6,9,8]`	replace item at index 1 with 9
`L.insert(1,9)`	`[6,9,7,8]`	insert a 9 at index 1 without replacing
`del L[1]`	`[6,8]`	delete second item
`del L[:2]`	`[8]`	delete first two items

7.6 Examples

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

```
from random import randint
L = []
for i in range(50):
    L.append(randint(1,100))
```

We use the `append` method to build up the list one item at a time starting with the empty list, `[]`. An alternative to `append` is to use the following:

```
L = L + [randint(1,100)]
```

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

```
for i in range(len(L)):
    L[i] = L[i]**2
```

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

```
count = 0
for item in L:
    if item>50:
        count=count+1
```

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

The key is the list method `count` that tells how many times a something occurs in a list.

Example 5 Write a program that prints out the two largest and two smallest elements of a list called `scores`.

```
scores.sort()
print('Two smallest: ', scores[0], scores[1])
print('Two largest: ', scores[-1], scores[-2])
```

Once we sort the list, the smallest values are at the beginning and the largest are at the end.

Example 6 Here is a program to play a simple quiz game.

```
num_right = 0

# Question 1
print('What is the capital of France?', end=' ')
guess = input()
if guess.lower()=='paris':
    print('Correct!')
    num_right+=1
else:
    print('Wrong.  The answer is Paris.')
print('You have', num_right, 'out of 1 right')

#Question 2
print('Which state has only one neighbor?', end=' ')
guess = input()
if guess.lower()=='maine':
    print('Correct!')
    num_right+=1
else:
    print('Wrong.  The answer is Maine.')
print('You have', num_right, 'out of 2 right,')
```

The code works, but it is very tedious. If we want to add more questions, we have to copy and paste one of these blocks of code and then change a bunch of things. If we decide to change one of the questions or the order of the questions, then there is a fair amount of rewriting involved. If we decide to change the design of the game, like not telling the user the correct answer, then every single block of code has to be rewritten. Tedious code like this can often be greatly simplified with lists and loops:

```
questions = ['What is the capital of France?',
             'Which state has only one neighbor?']
answers = ['Paris','Maine']

num_right = 0
for i in range(len(questions)):
    guess = input(questions[i])
    if guess.lower()==answers[i].lower():
        print('Correct')
        num_right=num_right+1
    else:
        print('Wrong.  The answer is', answers[i])
    print('You have', num_right, 'out of', i+1, 'right.')
```

If you look carefully at this code, you will see that the code in the loop is the nearly the same as the code of one of the blocks in the previous program, except that in the statements where we print the questions and answers, we use `questions[i]` and `answers[i]` in place of the actual text of the questions themselves.

This illustrates the general technique: If you find yourself repeating the same code over and over, try lists and a for loop. The few parts of your repetitious code that are varying are where the list code will go.

The benefits of this are that to change a question, add a question, or change the order, only the `questions` and `answers` lists need to be changed. Also, if you want to make a change to the program, like not telling the user the correct answer, then all you have to do is modify a single line, instead of twenty copies of that line spread throughout the program.

7.7 Exercises

1. Write a program that asks the user to enter a list of integers. Do the following:
   (a) Print the total number of items in the list.
   (b) Print the last item in the list.
   (c) Print the list in reverse order.
   (d) Print `Yes` if the list contains a 5 and `No` otherwise.
   (e) Print the number of fives in the list.
   (f) Remove the first and last items from the list, sort the remaining items, and print the result.