str's format Method
For Introduction to Programming Using Python
By Y. Daniel Liang

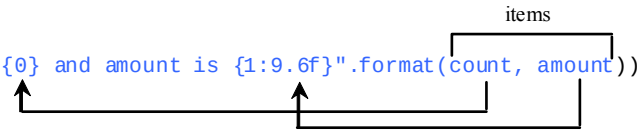
The text introduced the format function to format a number or a string. The str's class has a format method that is very similar to the format function. This supplement introduces the str's format method.

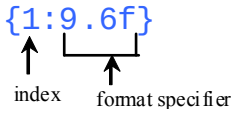
The syntax to invoke this method is

```
s.format(item0, item1, item2, ..., itemk)
```

where s is a string that may consist of substrings and formatting instruction for each item in the form of {index:format-specifier}. index refers to the item and format-specifier specifies how the item is formatted. For example, the following code

```
count = 5
amount = 45.56
print("count is {0} and amount is {1:9.6f}".format(count, amount))
```





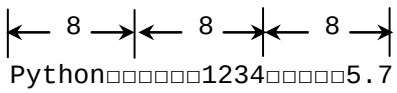
displays

```
count is 5 and amount is 45.560000
```

This example formats two items. The first item is count. No format-specifier is specified for count. So the value of count is simply inserted in the slot indicated by {0}. The second item is amount and its specifier is 9.6f, which specified the width of the item is 9 and precision is 6. f means a fixed point number. You can use d for decimal integer and s for a string. For example, the following code

```
print("{0:8s}{1:8d}{2:8.1f}".format("Python", 1234, 5.68))
```

displays



where the square box (□) denotes a blank space. Note that the last the number in the last item is rounded up to the specified precision.

If an item requires more spaces than the specified width, the width is automatically increased. For example, the following code

```
print("{0:3s}#{1:2d}#{2:3.2f}".format("Python", 111, 924.656))
```

displays

```
Python#111#924.66
```

The specified width for string item Python is 3, which is smaller than the string size 8. The width is automatically increased to 8. The specified width for int item 111 is 2, which is smaller than its actual size 3. The width is automatically increased to 3. The specified width for float item 924.656 is 3, but it needs width 6 to display 924.66. So the width is automatically increased to 6.

By default, the strings are left aligned and the numbers are right aligned. You can put the < or > symbol in the specifier to specify that the item is left or right aligned in the output within the specified field. For example, the following statements

```
print("{0:>8d}#{1:>8s}#{2:>8.1f}".format(1234, "Python", 5.63))
print("{0:<8d}#{1:<8s}#{2:<8.1f}".format(1234, "Python", 5.63))
print("{0:8d}#{1:8s}#{2:8.1f}".format(1234, "Python", 5.63))
```

display

```
□□□□1234#□□Python#□□□□□5.6
1234□□□□#Python□□#5.6□□□□□
□□□□1234#Python□□#□□□□□5.6
```

Note that the specifier types d and s can be omitted. So, the preceding statement can be written as

```
print("{0:>8}#{1:>8}#{2:>8.1f}".format(1234, "Python", 5.63))
print("{0:<8}#{1:<8}#{2:<8.1f}".format(1234, "Python", 5.63))
print("{0:8}#{1:8}#{2:8.1f}".format(1234, "Python", 5.63))
```

Check point

1 What is wrong in the following statements?

```
(a) print("{0} {1}".format(1, 2, 3))
(b) print("{0} {1}".format(1))
```

2 Show the output of the following statements.

```
(a) print("amount is {0:5.4f} {1:10.2}".format(33.32, 33.32))
(b) print("amount is {0} {1:9.2}".format(33.32, 33.32))
(c) print("amount is {0} {1:10.2}".format("New York", 33.32))
```