### 列表

>>> a = [23, 45, 1, -3434, 43624356, 234]

>>> a.append(45)

>>> a

[23, 45, 1, -3434, 43624356, 234, 45]

首先我们建立了一个列表 a。然后调用列表的方法 a.append(45) 添加元素 45 到列表末尾。你可以看到元素 45 已经添加到列表的末端了。有些时候我们需要将数据插入到列表的任何位置，这时我们可以使用列表的 insert() 方法。

>>> a.insert(0, 1) # 在列表索引 0 位置添加元素 1

>>> a

[1, 23, 45, 1, -3434, 43624356, 234, 45]

>>> a.insert(0, 111) # 在列表索引 0 位置添加元素 111

>>> a

[111, 1, 23, 45, 1, -3434, 43624356, 234, 45]

列表方法 count(s) 会返回列表元素中 s 的数量。我们来检查一下 45 这个元素在列表中出现了多少次。

>>> a.count(45)

2

如果你想要在列表中移除任意指定值，你需要使用 remove() 方法。

>>> a.remove(234)

>>> a

[111, 1, 23, 45, 1, -3434, 43624356, 45]

现在我们反转整个列表。

>>> a.reverse()

>>> a

[45, 43624356, -3434, 1, 45, 23, 1, 111]

怎样将一个列表的所有元素添加到另一个列表的末尾呢，可以使用列表的 extend() 方法。

>>> b = [45, 56, 90]

>>> a.extend(b) # 添加 b 的元素而不是 b 本身

>>> a

[45, 43624356, -3434, 1, 45, 23, 1, 111, 45, 56, 90]

给列表排序，我们使用列表的 sort() 方法，排序的前提是列表的元素是可比较的。

>>> a.sort()

>>> a

[-3434, 1, 1, 23, 45, 45, 45, 56, 90, 111, 43624356]

你也能使用 del 关键字删除指定位置的列表元素。

>>> del a[-1]

>>> a

[-3434, 1, 1, 23, 45, 45, 45, 56, 90, 111]

#### 将列表用作栈和队列

栈是我们通常所说的一种 LIFO （Last In First Out 后进先出）数据结构。它的意思是最后进入的数据第一个出来。一个最简单的例子往一端封闭的管道放入一些弹珠然后取出来，如果你想把弹珠取出来，你必须从你最后放入弹珠的位置挨个拿出来。用代码实现此原理：

>>> a = [1, 2, 3, 4, 5, 6]

>>> a

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

>>> a.pop()

6

>>> a.pop()

5

>>> a.pop()

4

>>> a.pop()

3

>>> a

[1, 2]

>>> a.append(34)

>>> a

[1, 2, 34]

上面的代码中我们使用了一个新方法 pop()。传入一个参数 i 即 pop(i) 会将第 i 个元素弹出。

在我们的日常生活中会经常遇到队列，比如售票窗口、图书馆、超市的结账出口。队列 是一种在末尾追加数据以及在开始弹出数据的数据结构。与栈不同，它是 FIFO （First In First Out 先进先出）的数据结构。

>>> a = [1, 2, 3, 4, 5]

>>> a.append(1)

>>> a

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

>>> a.pop(0)

1

>>> a.pop(0)

2

>>> a

[3, 4, 5, 1]

我们使用 a.pop(0) 弹出列表中第一个元素。

#### 列表推导式

列表推导式为从序列中创建列表提供了一个简单的方法。如果没有列表推导式，一般都是这样创建列表的：通过将一些操作应用于序列的每个成员并通过返回的元素创建列表，或者通过满足特定条件的元素创建子序列。

假设我们创建一个 squares 列表，可以像下面这样创建。

>>> squares = []

>>> for x in range(10):

... squares.append(x\*\*2)

...

>>> squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

注意这个 for 循环中的被创建（或被重写）的名为  x  的变量在循环完毕后依然存在。使用如下方法，我们可以计算 squares 的值而不会产生任何的副作用：。

squares = list(map(lambda x: x\*\*2, range(10)))

等价于下面的列表推导式。

squares = [x\*\*2 for x in range(10)]

上面这个方法更加简明且易读。

列表推导式由包含一个表达式的中括号组成，表达式后面跟随一个  for 子句，之后可以有零或多个  for  或  if  子句。结果是一个列表，由表达式依据其后面的 for  和  if  子句上下文计算而来的结果构成。

例如，如下的列表推导式结合两个列表的元素，如果元素之间不相等的话：

>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

等同于：

>>> combs = []

>>> for x in [1,2,3]:

... for y in [3,1,4]:

... if x != y:

... combs.append((x, y))

...

>>> combs

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

值得注意的是在上面两个方法中的 for  和 if 语句的顺序。

列表推导式也可以嵌套。

>>> a=[1,2,3]

>>> z = [x + 1 for x in [x \*\* 2 for x in a]]

>>> z

[2, 5, 10]

### 元组

元组是由数个逗号分割的值组成。

>>> a = 'Fedora', 'ShiYanLou', 'Kubuntu', 'Pardus'

>>> a

('Fedora', 'ShiYanLou', 'Kubuntu', 'Pardus')

>>> a[1]

'ShiYanLou'

>>> for x in a:

... print(x, end=' ')

...

Fedora ShiYanLou Kubuntu Pardus

你可以对任何一个元组执行拆封操作并赋值给多个变量，就像下面这样：

>>> divmod(15,2)

(7, 1)

>>> x, y = divmod(15,2)

>>> x

7

>>> y

1

元组是不可变类型，这意味着你不能在元组内删除或添加或编辑任何值。如果你尝试这些操作，将会出错：

>>> a = (1, 2, 3, 4)

>>> del a[0]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object doesn't support item deletion

要创建只含有一个元素的元组，在值后面跟一个逗号。

>>> a = (123)

>>> a

123

>>> type(a)

<class 'int'>

>>> a = (123, )

>>> b = 321,

>>> a

(123,)

>>> b

(321,)

>>> type(a)

<class 'tuple'>

>>> type(b)

<class 'tuple'>

通过内建函数 type() 你可以知道任意变量的数据类型。还记得我们使用 len() 函数来查询任意序列类型数据的长度吗？

>>> type(len)

<class 'builtin\_function\_or\_method'>

### 集合

集合是一个无序不重复元素的集。基本功能包括关系测试和消除重复元素。集合对象还支持 union（联合），intersection（交），difference（差）和 symmetric difference（对称差集）等数学运算。

大括号或 set() 函数可以用来创建集合。注意：想要创建空集合，你必须使用 set() 而不是 {}。后者用于创建空字典，我们在下一节中介绍的一种数据结构。

下面是集合的常见操作：

>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

>>> print(basket) # 你可以看到重复的元素被去除

{'orange', 'banana', 'pear', 'apple'}

>>> 'orange' in basket

True

>>> 'crabgrass' in basket

False

>>> # 演示对两个单词中的字母进行集合操作

...

>>> a = set('abracadabra')

>>> b = set('alacazam')

>>> a # a 去重后的字母

{'a', 'r', 'b', 'c', 'd'}

>>> a - b # a 有而 b 没有的字母

{'r', 'd', 'b'}

>>> a | b # 存在于 a 或 b 的字母

{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

>>> a & b # a 和 b 都有的字母

{'a', 'c'}

>>> a ^ b # 存在于 a 或 b 但不同时存在的字母

{'r', 'd', 'b', 'm', 'z', 'l'}

从集合中添加或弹出元素：

>>> a = {'a','e','h','g'}

>>> a.pop() # pop 方法随机删除一个元素并打印

'h'

>>> a.add('c')

>>> a

{'c', 'e', 'g', 'a'}

### 字典

字典是无序的键值对（key:value）集合，同一个字典内的键必须是互不相同的。一对大括号 {} 创建一个空字典。初始化字典时，在大括号内放置一组逗号分隔的键：值对，这也是字典输出的方式。我们使用键来检索存储在字典中的数据。

>>> data = {'kushal':'Fedora', 'kart\_':'Debian', 'Jace':'Mac'}

>>> data

{'kushal': 'Fedora', 'Jace': 'Mac', 'kart\_': 'Debian'}

>>> data['kart\_']

'Debian'

创建新的键值对很简单：

>>> data['parthan'] = 'Ubuntu'

>>> data

{'kushal': 'Fedora', 'Jace': 'Mac', 'kart\_': 'Debian', 'parthan': 'Ubuntu'}

使用 del 关键字删除任意指定的键值对：

>>> del data['kushal']

>>> data

{'Jace': 'Mac', 'kart\_': 'Debian', 'parthan': 'Ubuntu'

使用 in 关键字查询指定的键是否存在于字典中。

>>> 'ShiYanLou' in data

False

必须知道的是，字典中的键必须是不可变类型，比如你不能使用列表作为键。

dict() 可以从包含键值对的元组中创建字典。

>>> dict((('Indian','Delhi'),('Bangladesh','Dhaka')))

{'Indian': 'Delhi', 'Bangladesh': 'Dhaka'}

如果你想要遍历一个字典，使用字典的 items() 方法。

>>> data

{'Kushal': 'Fedora', 'Jace': 'Mac', 'kart\_': 'Debian', 'parthan': 'Ubuntu'}

>>> for x, y in data.items():

... print("{} uses {}".format(x, y))

...

Kushal uses Fedora

Jace uses Mac

kart\_ uses Debian

parthan uses Ubuntu

许多时候我们需要往字典中的元素添加数据，我们首先要判断这个元素是否存在，不存在则创建一个默认值。如果在循环里执行这个操作，每次迭代都需要判断一次，降低程序性能。

我们可以使用 dict.setdefault(key, default) 更有效率的完成这个事情。

>>> data = {}

>>> data.setdefault('names', []).append('Ruby')

>>> data

{'names': ['Ruby']}

>>> data.setdefault('names', []).append('Python')

>>> data

{'names': ['Ruby', 'Python']}

>>> data.setdefault('names', []).append('C')

>>> data

{'names': ['Ruby', 'Python', 'C']}

试图索引一个不存在的键将会抛出一个 keyError 错误。我们可以使用 dict.get(key, default) 来索引键，如果键不存在，那么返回指定的 default 值。

>>> data['foo']

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 'foo'

>>> data.get('foo', 0)

0

如果你想要在遍历列表（或任何序列类型）的同时获得元素索引值，你可以使用 enumerate()。

>>> for i, j in enumerate(['a', 'b', 'c']):

... print(i, j)

...

0 a

1 b

2 c

你也许需要同时遍历两个序列类型，你可以使用 zip() 函数。

>>> a = ['Pradeepto', 'Kushal']

>>> b = ['OpenSUSE', 'Fedora']

>>> for x, y in zip(a, b):

... print("{} uses {}".format(x, y))

...

Pradeepto uses OpenSUSE

Kushal uses Fedora

### 程序示例

本节实验将会通过两个实例程序来熟悉 Python3 的基本数据结构：

1. 判断学生成绩是否达标的程序
2. 计算两个矩阵的 Hadamard 乘积

#### students.py

这是一个判断学生成绩是否达标的程序，要求输入学生数量，以及各个学生物理、数学、历史三科的成绩，如果总成绩小于 120，程序打印 “failed”，否则打印 “passed”。

代码写入 /home/shiyanlou/students.py 文件：

#!/usr/bin/env python3

n = int(input("Enter the number of students: "))

data = {} # 用来存储数据的字典变量

Subjects = ('Physics', 'Maths', 'History') # 所有科目

for i in range(0, n):

name = input('Enter the name of the student {}: '.format(i + 1)) # 获得学生名称

marks = []

for x in Subjects:

marks.append(int(input('Enter marks of {}: '.format(x)))) # 获得每一科的分数

data[name] = marks

for x, y in data.items():

total = sum(y)

print("{}'s total marks {}".format(x, total))

if total < 120:

print(x, "failed :(")

else:

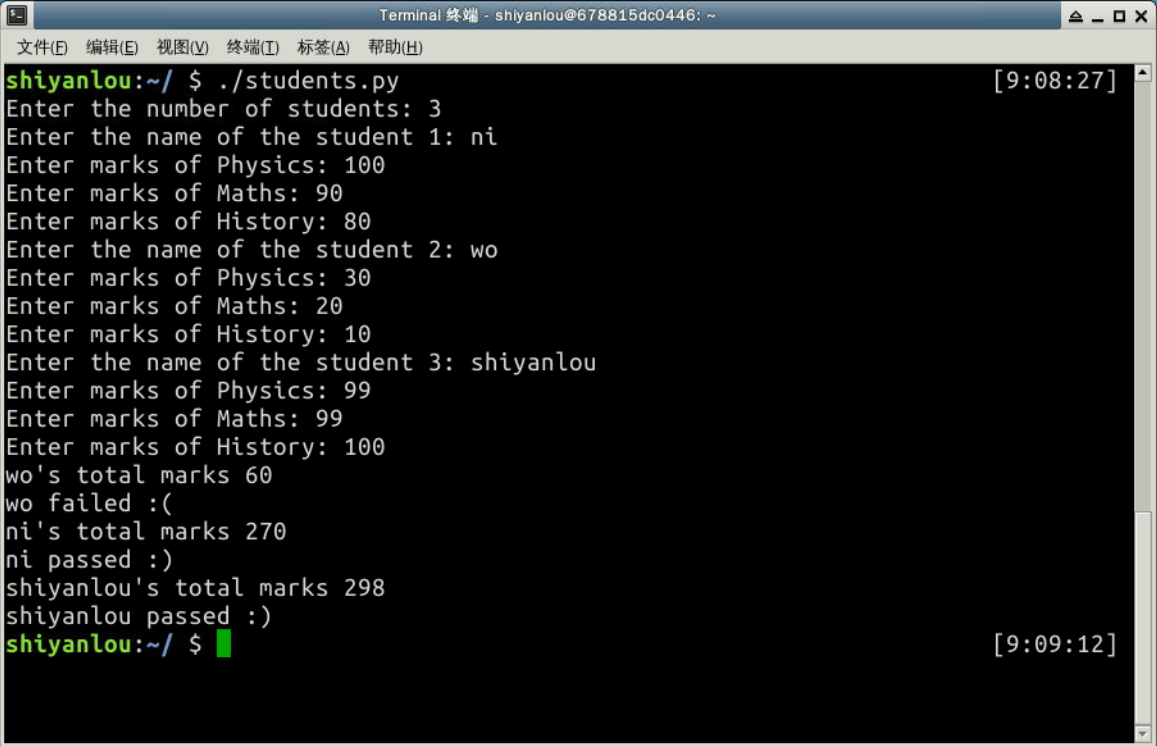
print(x, "passed :)")

name 和 marks 是变量，name 用来存储学生的名称，marks 是个列表，用来存储输入的学生的成绩数据。

data 是个字典，字典的键值对中，键指的是 name 的值，值指的是 marks 的值。因此会使用 data[name] = marks 将学生数据存入到 data 字典。

最后通过 for 循环遍历字典，x 为学生的 name，y 为学生成绩列表 marks，sum() 函数会将传入的列表进行加和。

运行如下：



#### matrixmul.py

这个例子里我们计算两个矩阵的 Hadamard 乘积。要求输入矩阵的行/列数（在这里假设我们使用的是 n × n 的矩阵）。

代码写入 /home/shiyanlou/matrixmul.py 文件：

#!/usr/bin/env python3

n = int(input("Enter the value of n: "))

print("Enter values for the Matrix A")

a = []

for i in range(n):

a.append([int(x) for x in input().split()])

print("Enter values for the Matrix B")

b = []

for i in range(n):

b.append([int(x) for x in input().split()])

c = []

for i in range(n):

c.append([a[i][j] \* b[i][j] for j in range(n)])

print("After matrix multiplication")

print("-" \* 7 \* n)

for x in c:

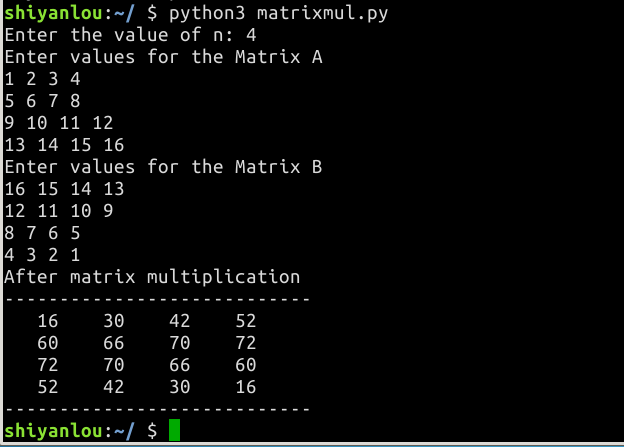
for y in x:

print(str(y).rjust(5), end=' ')

print()

print("-" \* 7 \* n)

运行如下：



这里我们使用了几次列表推导式。[int(x) for x in input().split()] 首先通过 input() 获得用户输入的字符串，再使用 split() 分割字符串得到一系列的数字字符串，然后用 int() 从每个数字字符串创建对应的整数值。我们也使用了 [a[i][j] \* b[i][j] for j in range(n)] 来得到矩阵乘积的每一行数据。

### 字符串

#### 字符串表示

可以通过几种不同的方式表示字符串。如单引号（'...'）或双引号（"..."）。下面的例子能帮助你更好的理解字符串。

>>> s = "I am Chinese"

>>> s

'I am Chinese'

>>> s = 'I am Chinese'

>>> s = "Here is a line \

... split in two lines"

>>> s

'Here is a line split in two lines'

>>> s = "Here is a line \n split in two lines"

>>> s

'Here is a line \n split in two lines'

>>> print(s)

Here is a line

split in two lines

如果你想要分几行输入字符串，并且希望行尾的换行符自动包含到字符串当中，可以使用三对引号："""...""" 或 '''...''' 。

>>> print("""\

... Usage: thingy [OPTIONS]

... -h Display this usage message

... -H hostname Hostname to connect to

... """)

Usage: thingy [OPTIONS]

-h Display this usage message

-H hostname Hostname to connect to

#### 字符串的方法

每一个字符串对象都有几个可用的内建方法，我们已经使用过一些了，比如 s.split()。

>>> s = "shi yan lou"

>>> s.title()

'Shi Yan Lou'

方法 title() 返回字符串的标题版本，即单词首字母大写其余字母小写。

>>> z = s.upper()

>>> z

'SHI YAN LOU'

>>> z.lower()

'shi yan lou'

方法 upper() 返回字符串全部大写的版本，反之 lower() 返回字符串的全部小写版本。

>>> s = "I am A pRoGraMMer"

>> s.swapcase()

'i AM a PrOgRAmmER'

方法 swapcase() 返回字符串大小写交换后的版本 :）

>>> s = "jdwb 2323bjb"

>>> s.isalnum()

False

>>> s = "jdwb2323bjb"

>>> s.isalnum()

True

方法 isalnum() 检查所有字符是否只有字母和数字，上面的代码中第一行的字符串 s 中包含空格字符，所以返回 False。

>>> s = "SankarshanSir"

>>> s.isalpha()

True

>>> s = "Sankarshan Sir"

>>> s.isalpha()

False

方法 isalpha() 检查字符串之中是否只有字母。

>>> s = "1234"

>>> s.isdigit() # 检查字符串是否所有字符为数字

True

>>> s = "ShiYanLou is coming"

>>> s.islower() # 检查字符串是否所有字符为小写

False

>>> s = "Shiyanlou Is Coming"

>>> s.istitle() # To 检查字符串是否为标题样式

True

>>> s = "CHINA"

>>> s.isupper() # 检查字符串是否所有字符为大写

True

我们可以使用 split() 分割任意字符串，split() 允许有一个参数，用来指定字符串以什么字符分隔（默认为 " "），它返回一个包含所有分割后的字符串的列表。

>>> s = "We all love Python"

>>> s.split()

['We', 'all', 'love', 'Python']

>>> x = "shiyanlou:is:waiting"

>>> x.split(':')

['shiyanlou', 'is', 'waiting']

相反的，方法 join() 使用指定字符连接多个字符串，它需要一个包含字符串元素的列表作为输入然后连接列表内的字符串元素。

>>> "-".join("GNU/Linux is great".split())

'GNU/Linux-is-great'

在上面的例子中，我们基于空格 " " 分割字符串 "GNU/Linux is great"，然后用 "-" 连接它们。

字符串剥离

字符串有几个进行剥离操作的方法。最简单的一个是 strip(chars)，用来剥离字符串首尾中指定的字符，它允许有一个字符串参数，这个参数为剥离哪些字符提供依据。不指定参数则默认剥离掉首尾的空格和换行符，代码如下：

>>> s = " a bc\n "

>>> s.strip()

'a bc'

你可以使用 lstrip(chars) 或 rstrip(chars) 只对字符串左或右剥离。

>>> s = "www.foss.in"

>>> s.lstrip("cwsd.") #删除在字符串左边出现的'c','w','s','d','.'字符

'foss.in'

>>> s.rstrip("cnwdi.") #删除在字符串右边出现的'c','n','w','d','i','.'字符

'www.foss'

文本搜索  
字符串有一些方法能够帮助你搜索字符串里的文本或子字符串。下面给出示例：

>>> s = "faulty for a reason"

>>> s.find("for")

7

>>> s.find("fora")

-1

>>> s.startswith("fa") # 检查字符串是否以 fa 开头

True

>>> s.endswith("reason") # 检查字符串是否以 reason 结尾

True

find() 能帮助你找到第一个匹配的子字符串，没有找到则返回

#### 回文检查

回文是一种无论从左还是从右读都一样的字符序列。比如 “madam”。在这个例子中，我们检查用户输入的字符串是否是回文，并输出结果。

代码写入文件 /home/shiyanlou/palindrome.py：

#!/usr/bin/env python3

s = input("Please enter a string: ")

z = s[::-1] #把输入的字符串s 进行倒序处理形成新的字符串z

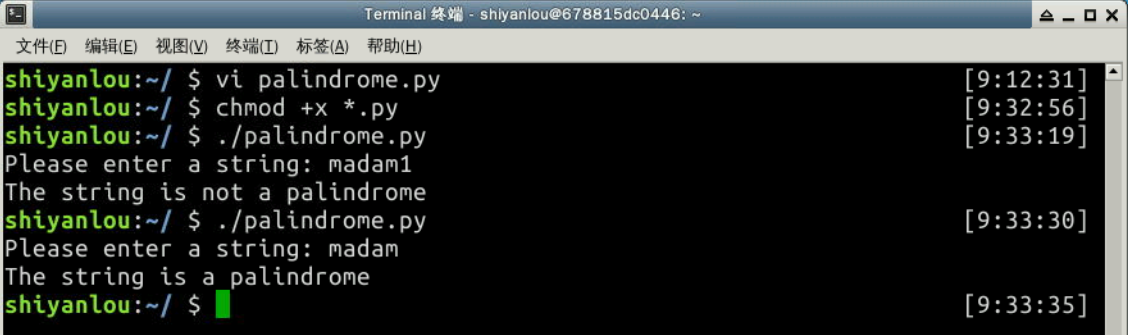
if s == z:

print("The string is a palindrome")

else:

print("The string is not a palindrome")

运行程序：

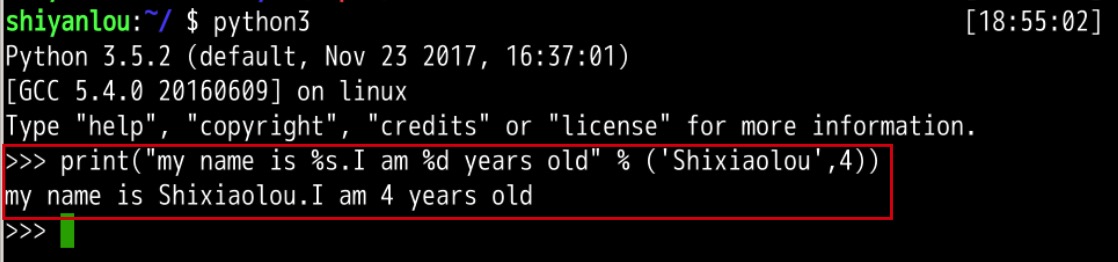


#### 单词计数

在讲解单词计数之前我们先了解一个概念：格式化操作符（%）。

我们先看下面的这个例子：

print("my name is %s.I am %d years old" % ('Shixiaolou',4))



在这个例子中，%s 为第一个格式符，表示一个字符串；%d 为第二个格式符，表示一个整数。格式符为真实值预留位置，并控制显示的格式。常用的有：

* %s 字符串（用 str() 函数进行字符串转换）
* %r 字符串（用 repr() 函数进行字符串转换）
* %d 十进制整数
* %f 浮点数
* %% 字符 %

那么接下来我们对用户输入的一行文本进行单词计数。

代码写入文件 /home/shiyanlou/countwords.py：

#!/usr/bin/env python3

s = input("Enter a line: ")

print("The number of words in the line are %d" % (len(s.split(" "))))

运行程序：

