print语句也可以跟上多个字符串，用逗号“,”隔开，就可以连成一串输出

如果一个字符串包含很多需要转义的字符，对每一个字符都进行转义会很麻烦。为了避免这种情况，我们可以在字符串前面加个前缀 r ，表示这是一个 raw 字符串，里面的字符就不需要转义了。例如：

r'\(~\_~)/ \(~\_~)/'

如果要表示多行字符串，可以用'''...'''表示：

'''Line 1

Line 2

Line 3'''

上面这个字符串的表示方法和下面的是完全一样的：

'Line 1\nLine 2\nLine 3'

还可以在多行字符串前面添加 r ，把这个多行字符串也变成一个raw字符串：

r'''Python is created by "Guido".

It is free and easy to learn.

Let's start learn Python in imooc!'''

Python在后来添加了对Unicode的支持，以Unicode表示的字符串用u'...'表示，比如：

print u'中文'

中文

如果中文字符串在Python环境下遇到 UnicodeDecodeError，这是因为.py文件保存的格式有问题。可以在第一行添加注释

# -\*- coding: utf-8 -\*-

目的是告诉Python解释器，用UTF-8编码读取源代码。

list添加新元素

append()：总是把新的元素添加到list的尾部。

insert()：把新的元素插入到指定的位置，第一个参数是索引号，第二个参数是待添加的新元素。

list删除元素

pop()：删掉list最后一个元素，且返回被删除的元素，需要用变量接收。

也可以根据索引号删除指定元素。

tuble：是一种有序的列表，中文翻译为“元组”，与list不同的是用( )代替了[ ],且创建完后不能再更改。

因为()既可以表示tuple，又可以作为括号表示运算时的优先级，结果 (1) 被Python解释器计算出结果 1，导致我们得到的不是tuple，而是整数 1。

正是因为用()定义单元素的tuple有歧义，所以 Python 规定，单元素 tuple 要多加一个逗号“,”，这样就避免了歧义：

>>> t = (1,)

>>> print t

(1,)

**Python之“可变”的tuple**

前面我们看到了tuple一旦创建就不能修改。现在，我们来看一个“可变”的tuple：

>>> t = ('a', 'b', ['A', 'B'])

**注意**到 t 有 3 个元素：**'a'，'b'**和一个list：**['A', 'B']**。list作为一个整体是tuple的第3个元素。list对象可以通过 t[2] 拿到：

>>> L = t[2]

然后，我们把list的两个元素改一改：

>>> L[0] = 'X'

>>> L[1] = 'Y'

再看看tuple的内容：

>>> print t

('a', 'b', ['X', 'Y'])

tuple一开始指向的list并没有改成别的list，所以，tuple所谓的**“不变”**是说，tuple的每个元素，指向永远不变。即**指向'a'，就不能改成指向'b'**，指向一个list，就不能改成指向其他对象，但指向的这个list本身是可变的！

age = 20

if age >= 18:

print 'your age is', age

print 'adult'

print 'END'

**注意:**Python代码的缩进规则。具有相同缩进的代码被视为代码块，上面的3，4行 print 语句就构成一个代码块（但不包括第5行的print）。如果 if 语句判断为 True，就会执行这个代码块。

缩进请严格按照Python的习惯写法：4个空格，不要使用Tab，更不要混合Tab和空格，否则很容易造成因为缩进引起的语法错误。

**注意**: if 语句后接表达式，然后用:表示代码块开始。

如果你在Python交互环境下敲代码，还要特别留意缩进，并且退出缩进需要多敲一行回车：

由于dict也是集合，len() 函数可以计算任意集合的大小：

>>> len(d)

3

**注意:** 一个 key-value 算一个，因此，dict大小为3。

**list 必须使用索引返回对应的元素，而dict使用key：**

>>> print d['Adam']

95

>>> print d['Paul']

Traceback (most recent call last):

File "index.py", line 11, in <module>

print d['Paul']

KeyError: 'Paul'

**注意:** 通过 key 访问 dict 的value，只要 key 存在，dict就返回对应的value。如果key不存在，会直接报错：KeyError。

要避免 KeyError 发生，有两个办法：

**一是先判断一下 key 是否存在，用 in 操作符：**

if 'Paul' in d:

print d['Paul']

如果 'Paul' 不存在，if语句判断为False，自然不会执行 print d['Paul'] ，从而避免了错误。

**二是使用dict本身提供的一个 get 方法，在Key不存在的时候，返回None：**

>>> print d.get('Bart')

59

>>> print d.get('Paul')

None

## Python中dict的特点

**dict的第一个特点是查找速度快，无论dict有10个元素还是10万个元素，查找速度都一样**。而list的查找速度随着元素增加而逐渐下降。

不过dict的查找速度快不是没有代价的，**dict的缺点是占用内存大，还会浪费很多内容**，list正好相反，占用内存小，但是查找速度慢。

由于dict是按 key 查找，所以，在一个dict中，key不能重复。

**dict的第二个特点就是存储的key-value序对是没有顺序的！**这和list不一样：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

当我们试图打印这个dict时：

>>> print d

{'Lisa': 85, 'Adam': 95, 'Bart': 59}

打印的顺序不一定是我们创建时的顺序，而且，不同的机器打印的顺序都可能不同，这说明dict内部是**无序**的，不能用dict存储有序的集合。

**dict的第三个特点是作为 key 的元素必须不可变**，Python的基本类型如字符串、整数、浮点数都是不可变的，都可以作为 key。但是list是可变的，就不能作为 key。

可以试试用list作为key时会报什么样的错误。

不可变这个限制仅作用于key，value是否可变无所谓：

{

'123': [1, 2, 3], # key 是 str，value是list

123: '123', # key 是 int，value 是 str

('a', 'b'): True # key 是 tuple，并且tuple的每个元素都是不可变对象，value是 boolean

}

最常用的key还是字符串，因为用起来最方便。

## Python更新dict

dict是可变的，也就是说，我们可以随时往dict中添加新的 key-value。比如已有dict：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

要把新同学'Paul'的成绩 72 加进去，用赋值语句：

>>> d['Paul'] = 72

d[(1,2,3)] = 'Paul'; #根据key的值不同而不同，不像php一样需要用单双引号包括起来。

## Python之 遍历dict

由于dict也是一个集合，所以，遍历dict和遍历list类似，都可以通过 for 循环实现。

直接使用for循环可以遍历 dict 的 key：

>>> d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

>>> for key in d:

... print key

...

Lisa

Adam

Bart

由于通过 key 可以获取对应的 value，因此，在循环体内，可以获取到value的值。

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

for accelworld in d:

print accelworld,':',d[accelworld]

## Python中什么是set

**dict的作用是建立一组 key 和一组 value 的映射关系，dict的key是不能重复的。**

有的时候，我们只想要 dict 的 key，不关心 key 对应的 value，目的就是保证这个集合的元素不会重复，这时，set就派上用场了。

**set 持有一系列元素，这一点和 list 很像，但是set的元素没有重复，而且是无序的，这点和 dict 的 key很像。**

创建 set 的方式是调用 set() 并传入一个 list，list的元素将作为set的元素：

>>> s = set(['A', 'B', 'C'])

可以查看 set 的内容：

>>> print s

set(['A', 'C', 'B'])

**请注意**，上述打印的形式类似 list， 但它不是 list，仔细看还可以发现，打印的顺序和原始 list 的顺序有可能是不同的，因为set内部存储的元素是**无序**的。

因为set不能包含重复的元素，所以，当我们传入包含重复元素的 list 会怎么样呢？

>>> s = set(['A', 'B', 'C', 'C'])

>>> print s

set(['A', 'C', 'B'])

>>> len(s)

3

结果显示，set会自动去掉重复的元素，原来的list有4个元素，但set只有3个元素。

dict：**dict的作用是建立一组 key 和一组 value 的映射关系，dict的key是不能重复的。**

两者皆不可重复且无序

set：与dict类似，不过没有value，且key不可重复。

list：元素可变

两者皆可重复且有序

tuple：元素不可变，但元素是list则可变

## Python之 访问set

由于**set存储的是无序集合**，所以我们没法通过索引来访问。

访问 set中的某个元素实际上就是判断一个元素是否在set中。

例如，存储了班里同学名字的set：

>>> s = set(['Adam', 'Lisa', 'Bart', 'Paul'])

我们可以用 in 操作符判断：

Bart是该班的同学吗？

>>> 'Bart' in s

True

## Python之 set的特点

**set的内部结构和dict很像，唯一区别是不存储value**，因此，判断一个元素是否在set中速度很快。

**set存储的元素和dict的key类似，必须是不变对象**，因此，任何可变对象是不能放入set中的。

最后，set存储的元素也是没有顺序的。

## Python之 遍历set

注意到set的元素是tuple，因此 for 循环的变量被依次赋值为tuple。

**参考代码:**

s = set([('Adam', 95), ('Lisa', 85), ('Bart', 59)])

for x in s:

print x[0] + ':', x[1]

## Python之 更新set

由于**set存储的是一组不重复的无序元素**，因此，更新set主要做两件事：

**一是把新的元素添加到set中，二是把已有元素从set中删除。**

添加元素时，用set的add()方法：

>>> s = set([1, 2, 3])

>>> s.add(4)

>>> print s

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

如果添加的元素已经存在于set中，add()不会报错，但是不会加进去了：

>>> s = set([1, 2, 3])

>>> s.add(3)

>>> print s

set([1, 2, 3])

删除set中的元素时，用set的remove()方法：

>>> s = set([1, 2, 3, 4])

>>> s.remove(4)

>>> print s

set([1, 2, 3])

如果删除的元素不存在set中，remove()会报错：

>>> s = set([1, 2, 3])

>>> s.remove(4)

Traceback (most recent call last):

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

KeyError: 4

所以用add()可以直接添加，而remove()前需要判断。

s = set(['Adam', 'Lisa', 'Paul'])

L = ['Adam', 'Lisa', 'Bart', 'Paul']

for name in L:

if name in s:

s.remove(name)

else:

s.add(name)

print s

在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。

于是，fact(n)用递归的方式写出来就是：

def fact(n):

if n==1:

return 1

return n \* fact(n - 1)

如果我们计算fact(5)，可以根据函数定义看到计算过程如下：

===> fact(5)

===> 5 \* fact(4)

===> 5 \* (4 \* fact(3))

===> 5 \* (4 \* (3 \* fact(2)))

===> 5 \* (4 \* (3 \* (2 \* fact(1))))

===> 5 \* (4 \* (3 \* (2 \* 1)))

===> 5 \* (4 \* (3 \* 2))

===> 5 \* (4 \* 6)

===> 5 \* 24

===> 120

递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。

例如Python自带的 **int()** 函数，其实就有两个参数，我们既可以传一个参数，又可以传两个参数：

>>> int('123')

123

>>> int('123', 8)

83

int()函数的第二个参数是转换进制，如果不传，默认是十进制 (base=10)，如果传了，就用传入的参数。

可见，**函数的默认参数的作用是简化调用**，你只需要把必须的参数传进去。但是在需要的时候，又可以传入额外的参数来覆盖默认参数值。

由于函数的参数按从左到右的顺序匹配，所以**默认参数只能定义在必需参数的后面**

## Python之定义可变参数

如果想让一个函数能接受任意个参数，我们就可以定义一个可变参数：

def fn(\*args):

print args

可变参数的名字前面有个 **\***号，我们可以传入0个、1个或多个参数给可变参数：

Python解释器会把传入的一组参数组装成一个tuple传递给可变参数，因此，在函数内部，直接把变量 args 看成一个 tuple 就好了

## 对list进行切片

对这种经常取指定索引范围的操作，用循环十分繁琐，因此，Python提供了切片（Slice）操作符，能大大简化这种操作。

对应上面的问题，取前3个元素，用一行代码就可以完成切片：

>>> L[0:3]

['Adam', 'Lisa', 'Bart']

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2，正好是3个元素。

如果第一个索引是0，还可以省略：

>>> L[:3]

['Adam', 'Lisa', 'Bart']

也可以从索引1开始，取出2个元素出来：

>>> L[1:3]

['Lisa', 'Bart']

只用一个**:**，表示从头到尾：

>>> L[:]

['Adam', 'Lisa', 'Bart', 'Paul']

L[:]实际上复制出了一个新list。

切片操作还可以指定第三个参数：

>>> L[::2]

['Adam', 'Bart']

第三个参数表示每N个取一个，上面的 L[::2] 会每两个元素取出两个元素中的第一个。

把list换成tuple，切片操作完全相同，只是切片的结果也变成了tuple。

**map()**是 Python 内置的高阶函数，它接收一个**函数 f** 和一个 **list**，并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回。

由于list包含的元素可以是任何类型，因此，map() 不仅仅可以处理只包含数值的 list，事实上它可以处理包含任意类型的 list，只要传入的函数f可以处理这种数据类型。

以下实例展示了 map() 的使用方法：

>>>def square(x) : # 计算平方数

... return x \*\* 2

...

>>> map(square, [1,2,3,4,5]) # 计算列表各个元素的平方

[1, 4, 9, 16, 25]

>>> map(lambda x: x \*\* 2, [1, 2, 3, 4, 5]) # 使用 lambda 匿名函数

[1, 4, 9, 16, 25]

# 提供了两个列表，对相同位置的列表数据进行相加

>>> map(lambda x, y: x + y, [1, 3, 5, 7, 9], [2, 4, 6, 8, 10])

[3, 7, 11, 15, 19]

**reduce()**函数也是Python内置的一个高阶函数。reduce()函数接收的参数和 map()类似，**一个函数 f，一个list**，但行为和 map()不同，reduce()传入的函数 f 必须接收两个参数，reduce()对list的每个元素反复调用函数f，并返回最终结果值。

例如，编写一个f函数，接收x和y，返回x和y的和：

def f(x, y):

return x + y

调用 **reduce(f, [1, 3, 5, 7, 9])**时，reduce函数将做如下计算：

先计算头两个元素：f(1, 3)，结果为4；

再把结果和第3个元素计算：f(4, 5)，结果为9；

再把结果和第4个元素计算：f(9, 7)，结果为16；

再把结果和第5个元素计算：f(16, 9)，结果为25；

由于没有更多的元素了，计算结束，返回结果25。

上述计算实际上是对 list 的所有元素求和。虽然Python内置了求和函数sum()，但是，利用reduce()求和也很简单。

**reduce()还可以接收第3个可选参数，作为计算的初始值。**如果把初始值设为100，计算：

reduce(f, [1, 3, 5, 7, 9], 100)

结果将变为125，因为第一轮计算是：

计算初始值和第一个元素：**f(100, 1)**，结果为**101**。

**filter()**函数是 Python 内置的另一个有用的高阶函数，filter()函数接收一个**函数 f**和一个**list**，这个函数 f 的作用是对每个元素进行判断，返回 True或 False，**filter()根据判断结果自动过滤掉不符合条件的元素，返回由符合条件元素组成的新list。**

例如，要从一个list [1, 4, 6, 7, 9, 12, 17]中删除偶数，保留奇数，首先，要编写一个判断奇数的函数：

def is\_odd(x):

return x % 2 == 1

然后，利用filter()过滤掉偶数：

filter(is\_odd, [1, 4, 6, 7, 9, 12, 17])

**结果：**[1, 7, 9, 17]

 s.strip(rm) 删除 s 字符串中开头、结尾处的 rm 序列的字符。

Python的函数不但可以返回int、str、list、dict等数据类型，还可以返回函数！

例如，定义一个函数 f()，我们让它返回一个函数 g，可以这样写：

def f():

print 'call f()...'

# 定义函数g:

def g():

print 'call g()...'

# 返回函数g:

return g

仔细观察上面的函数定义，我们在函数 f 内部又定义了一个函数 g。由于函数 g 也是一个对象，函数名 g 就是指向函数 g 的变量，所以，最外层函数 f 可以返回变量 g，也就是函数 g 本身。

调用函数 f，我们会得到 f 返回的一个函数：

>>> x = f() # 调用f()

call f()...

>>> x # 变量x是f()返回的函数：

<function g at 0x1037bf320>

>>> x() # x指向函数，因此可以调用

call g()... # 调用x()就是执行g()函数定义的代码

python3字符串默认使用Unicode编码，所以支持多语言

**\_\_name\_\_属性**

一个模块被另一个程序第一次引入时，其主程序将运行。如果我们想在模块被引入时，模块中的某一程序块不执行，我们可以用\_\_name\_\_属性来使该程序块仅在该模块自身运行时执行。

#!/usr/bin/python3

# Filename: using\_name.py

if \_\_name\_\_ == '\_\_main\_\_':

print('程序自身在运行')

else:

print('我来自另一模块')

**说明：** 每个模块都有一个\_\_name\_\_属性，当其值是'\_\_main\_\_'时，表明该模块自身在运行，否则是被引入。