

Python

赋值与输出

第一个Python程序

```
a = 6
print a

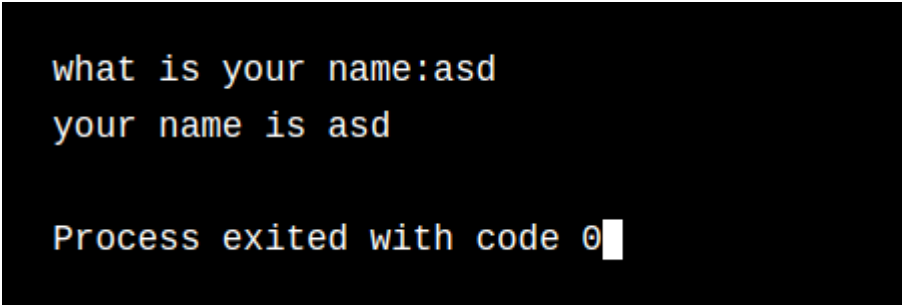
b = 'hello world'
print b

a = b
print a, b
```

数据输入

raw_input是一个等待系统输出的函数

```
name = raw_input('what is your name:')
print 'your name is ' + name
```



```
what is your name:asd
your name is asd
```

```
Process exited with code 0
```

数学运算

```
a = 1
b = 2
print a + b
print b * 3
print b + a * 2
```

```
3
6
4

Process exited with code 0
```

传统除法运算

```
print 3 / 2
print 3.0 / 2.0
print 2. / 2.
```

```
1
1.5
1.0

Process exited with code 0
```

全局精确除法运算

如果我们希望一直提供给我们的都是精确的除法结果而不关心计算的是什么类型的数字我们可以充 `__future__` 包中导入一个叫division的方法

取整除的除法在引入division的方法后并非就完全不存在了我们可以使用//运算符进行实现

```
from __future__ import division

print 3 / 2
print 3.0 / 2.0
print 2. / 2.
print 3 // 2
```

```
1.5
1.5
1.0
1

Process exited with code 0
```

非数学运算

```
a = 'love '
b = 'China'
print a + b
print a * 3 + b
//我们称这种加法运算被重载了
```

```
love China
love love love China

Process exited with code 0
```

字符串长度

使用len函数读取字符串长度

在Python中，存储着整数和字符串的两个不同类型的变量减是没有已经被定义的运算的

我们通过str函数将a_len中存储的数字编程一个字符串

```
a = 'China'
a_len = len(a)
b = 'length of a is'
print b + str(a_len)
```

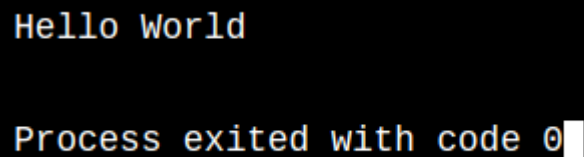
```
length of a is5

Process exited with code 0
```

Hello World

首先声明一个main函数作为这个程序所定义的所有逻辑被一次放置的地方。在用一个标准的方式让程序以刚刚定义的main函数作为程序执行的入口并一次执行main中的语句

```
def main():  
    print 'Hello World'  
  
if __name__ == '__main__':  
    main()
```



```
Hello World  
  
Process exited with code 0
```

Python的函数

组成一个函数中def这个词我们称为定义函数的关键字，这个词后空一个空格后定义的就是这个函数的名称，而再之后的括号中定义的是这个函数接受的函数的参数形式。而真正利用接受到的参数来进行函数功能描述的函数的定义是在def这行之后

我们通过函数的形式进行调用，被传入的数值是被传给参数形式的实际参数，return后是函数被调用后的返回结果

缩进的使用

在Python中缩进是很关键的，相同的缩进确保了逻辑相关语句被组织在一起。比如一个函数的定义部分，就需要有相同的缩进，如果我们错误的使用了缩进，我们就可能在程序运行时被告知存在语法错误

帮助

help返回的是一个函数的形式和它的作用说明

dir返回的不是一个函数定义，而是一个模组中一系列的被定义过的列表如dir(sys)返回的应该是sys下所有的被定义的函数方法的列表

```
import sys  
help(len)  
  
print dir(sys)  
  
help(sys.exit)  
help('china'.split)  
print dir(list)
```

注释

注释内容以#靠头

A+B+C

```
a, b, c = (int(x) for x in raw_input().split(' '))

print a + b + c
```

Python的字符串

Python内建立了一个专门用于处理字符串的“库”（实际是一个名为str的类）

```
a = 3    变量是指向一个存储数值3的盒子
a = '3'  变量是指向一个储存字符串3的盒子
```

当你希望在字符串的内容中包括单引号或者双引号时，你需要使用他们的转义字符分别是在正常的单引号或双引号前加斜线写成\'和\"的形式一个双引号包裹的字符串中可以直接包含单引号而不需要使用单引号的转义字符例：`a = I don't know` 一个单引号包裹的字符串中可以直接包含双引号而不需要使用双引号的转义字符

一个字符串在Python中可以分成多行来写，但是我们需要每行的末尾加一个斜杠

```
a = 'this is a string\
that is so good to \
let you to understand'
```

字符串是不可变的也就是说一个字符串被创建后就不可以被做出任何改变（类似的情况在Java中也存在）。因为字符串是不可变的因此我们在表示一个计算结果时，实际上就是在不断的创建新的字符串。比方说 `a = 'hello' + 'there'` 实际上就是用两个已经被创建的字符串'hello'和'there'在创建一个新的字符串'hellothere'并且让变量a指向装着这个新的字符串的盒子

我们可以通过方括号来访问字符串中的每一个字符(长度为1的在某一个位置的子字符串)，他们开始索引的位置都是0，如果我们访问索引的位置不存在，我们就会出现超出合法范围的错误

```
a = 'hello'
b = a[0] + 'i'
print b
print len(b)
```

```
hi
2

Process exited with code 0
```

大小写

lower()将字符串全部字母转为小写字母

upper()将字符串全部字母转为大写字母

```
a = 'In\na line'
b = r'In\na line'
print a
print b

print a.lower()
print b.upper()
```

```
In
a line
In\na line
in
a line
IN\NA LINE

Process exited with code 0
```

字符串测试

isalpha()检测是否字符串全是有字母组成

isdigit()检测是否字符串全是有数字组成

isspace()检测时候字符串全是由空格组成

检测字符串是否是以Hello开头且World结尾的。这个时候我们会用staetswith()来检测是否是一个子字符串开始的，并用endswith()来检测是否是一个子字符串结束的。

```
s = 'HelloabcdWord'

print s.isalpha()
print s.isdigit()
print s.startswith('Hello')
print s.endswith("World")
```

```
True
False
True
False

Process exited with code 0
```

字符串的索引

一个字符串所在的盒子中有很多的小格子，他们被一次排列并且编号，我们称每隔格子中存的时一个字符串中的字符，而每个格子的编号为位置索引

在Python中字符串的每一个格子中的字符还有一个逆序索引，这个每个字符的逆序索引的值等于它的位置索引的值减去字符串的长度。例：Hello它0的H逆序索引就是0-5=-5

切取字符串

print tower[1:4] 别切取的时第2个到第4个字符 print tower[3:] 打印第四个字符以及之后的字符 print tower[:2] 打印除的是从字符串去掉最后两个字符做成的字符串

```
tower = 'abcdefg'
print tower[1:4]
print tower[3:]
print tower[:2]
```

```
bcd
defg
abcde

Process exited with code 0
```

if语句

写法与c和c++相似

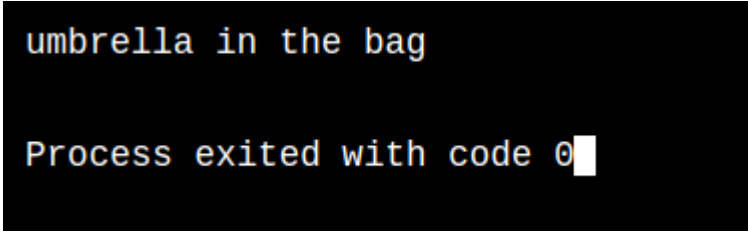
```
if a > 2 :  
    return 1  
elif a == 3 :  
    return 2  
else  
    return 3
```

查找与替换

find函数会在原字符串中查找参数传入的子字符串，如果没有找到请求的子字符串会返回-1如果找到会返回锁找到的子字符串的第一个字符子在原字符串中的索引位置

replace函数会将原字符串中包含的所有第一个传入参数的子字符串替换为第二个参数的字符串

```
weather = 'rainy day'  
bag = 'nothing in the bag'  
  
if weather.find('rain') != -1 :  
    bag = bag.replace('nothing', 'umbrella')  
  
print bag
```



```
umbrella in the bag
```

```
Process exited with code 0
```

字符串格式化

单引号外的%前定义了输出字符串的格式，其中%s表示字符串变量嵌入占位，%d表示整数变量嵌入占位，%g表示浮点数变量嵌入占位 相应的在%后的括号中，我们按照之前的格式中的占位顺序，一次列出希望将值进行嵌入的变量

```
name = 'Wangmu Niangniang'  
age = 9000  
height = 1.73  
print name + ' is a ' + str(age) + '-year-old woman with height ' + str(height)  
print '%s is a %d-year-old woman with height %g' % (name, age, height)
```



```
Wangmu Niangniang is a 9000-year-old woman with height
1.73
Wangmu Niangniang is a 9000-year-old woman with height
1.73

Process exited with code 0
```

批量修改字符串

将字符串中输入的空格转换成"%20"

```
print raw_input().replace(' ', r'%20');
```

```
we%20are%20happy
```

使用列表

Python中建立了一个称为列表的结构，用于储存一系列数据。字符串是特殊列表（但不是真正的列表不能使用列表的函数方法）我们可以简单的将它看成一系列的长度为1的子字符串组成的特殊列表

使用len函数来给出字符串长度对的原因时因为它给除了字符串这种特殊列表所包含长度为1的子字符串元素的长度

```
list = [100, 23, 45]

print list[0]
print list[1]
print list[2]
print len(list)
```

```
100
23
45
3

Process exited with code 0
```

列表尾部的添加

append用于添加单一元素的函数

extend函数将参数所指向的列表添加到原列表中

```
hello = ['hi', 'hello']
world = ['earth', 'field', 'universe']
hello.append('nihao')
print hello
hello.extend(world)
print hello
```

```
['hi', 'hello', 'nihao']
['hi', 'hello', 'nihao', 'earth', 'field', 'universe']

Process exited with code 0
```

插入数据与元素定位

insert函数将参数添加到指定的列表位置

```
hello = ['hi', 'hello']
hello.insert(0, 'nihao')
print hello
print hello.index('hi')
```

```
['nihao', 'hi', 'hello']
1

Process exited with code 0
```

列表弹出与删除

remove弹出指定字符串

pop删除位置索引处的元素

```
hello = ['nihao', 'hi', 'hello']
hello.remove('nihao')
print hello
hello.pop(0)
print hello
```

```
['hi', 'hello']  
['hello']
```

字符串的切割与列表合成

split函数按参数规则进行切割

join函数按照规则进行合并列表

```
manager = 'tuotatianwang,taibaijinxing,juanliandajiang'  
manager_list = manager.split(',')  
print manager_list  
new_manager = ' '.join(manager_list)  
print new_manager
```

```
['tuotatianwang', 'taibaijinxing', 'juanliandajiang']  
tuotatianwang taibaijinxing juanliandajiang
```

列表求和

for 临时变量 in 数组 临时变量的值和数组索引位置的值相等
sum(数组)对数组各元素进行求和计算

```
gardens = [7204, 3640, 1200, 1240, 71800, 3200, 604]  
total = 0  
for num in gardens :  
    total += num  
print total  
print sum(gardens)
```

```
88888  
88888
```

```
Process exited with code 0
```

range的使用

range函数通过调用它可以快速的生成等差数组列表简单的使用range(10)是以0开始以9结束的列表for num in range(100)num的数组在每一次循环中就会依次为0到99的每一个整数值

range(1, 10)将获得一个从1到9的列表

range(1, 10, 2)将获得一个从1到9等差为2的递增数组

range(10, 1, -2)将获得一个从10到1的等差为-2的递减数组

xrange函数除了循环给出的结果时不去建立完整的列表，从而在性能上优于range函数在使用函数参数上是一致的

while循环

```
first = 0
second = 1
# Please write code here
while first < 100 : //循环条件不超过100
    print first
    first, second = second, first + second
print 'Everything is done'
```

```
0
1
1
2
3
5
8
13
21
34
55
89
Everything is done

Process exited with code 0
```

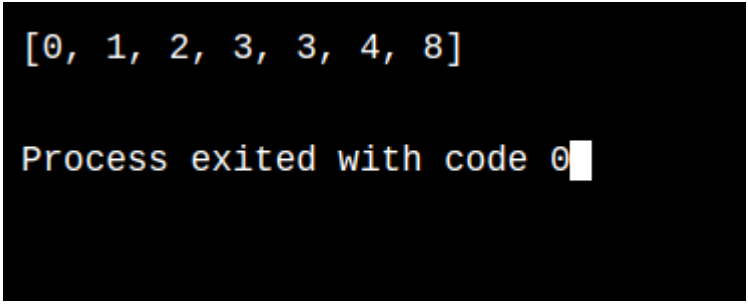
简单斐波那契

```
n = int(raw_input())
first = 0
second = 1
while n != 0 :
    first, second = second, first + second
    n -= 1
print first
```

简单的排序

使用sort函数可以进行从小到大的简单排序

```
numbers = [1, 4, 2, 3, 8, 3, 0]
numbers.sort()
print numbers
```



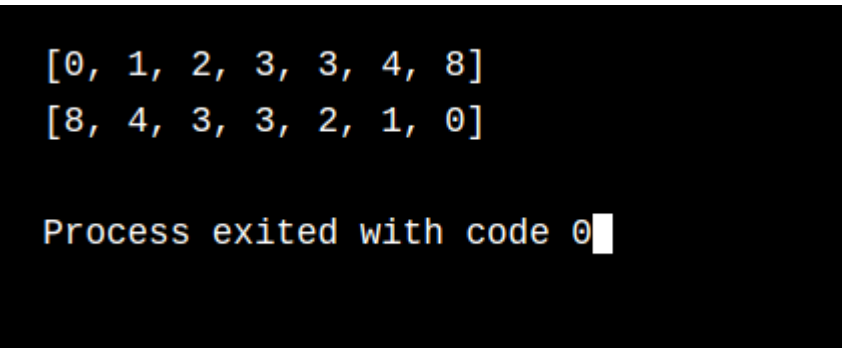
```
[0, 1, 2, 3, 3, 4, 8]
Process exited with code 0
```

基础排序

sort方法并没有返回值在Python中还存在有全局的返回值的排序的方法的

与sort相比sorted函数得到的值时排序后的列表而不是None

```
numbers = [1, 4, 2, 3, 8, 3, 0]
print sorted(numbers)
print sorted(numbers, reverse = True)
```



```
[0, 1, 2, 3, 3, 4, 8]
[8, 4, 3, 3, 2, 1, 0]
Process exited with code 0
```

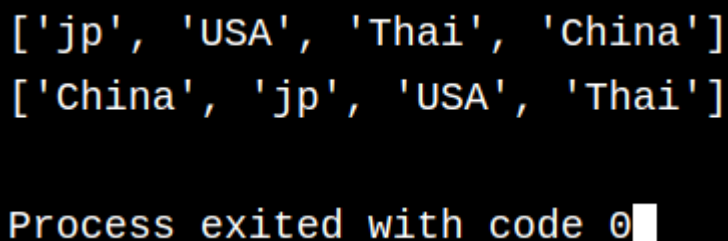
字典序

字典序：它是从两个字符串的开始字符依次比较，当当前位置的字符相同时，对下一位置的字符进行比较，直到出现第一个不相同的字符的时候，取这时字符大的字符串为大的字符串，而这时所在字符小的字符串为小的字符串。

个性化的排序

```
def china_first(item):
    if item == 'China':
        return 0
    else:
        return len(item)
country = ['jp', 'China', 'USA', 'Thai']

print sorted(country, key = len)
print sorted(country, key = china_first)
//根据定义的china_first函数的返回值作为中间值对原列表输入进行排序
```



```
['jp', 'USA', 'Thai', 'China']
['china', 'jp', 'USA', 'Thai']

Process exited with code 0
```

元组

元组是一个固定大小的一组元素，比如说(x, y)坐标。元组看起来就有点像列表，但是它的大小是固定的，不会改变的。有的时候人们会觉得它有点像Python中的结构体，不仅因为它的大小是固定的，而且它还可以体现出一定的逻辑性。

如果一个函数需要返回多个值，就可以以元组的形式被返回。

在Python中创建一个元组，只要把你期望放在元组的值用逗号分隔并放在圆括号内即可。有的时候我们会遇到的空元组的表示形式就是简单的一对括号。访问元组内某个元组的方式和访问列表的元素是一致的, [], len(), for...in等都可以被同样的用于元组

如果创建一个只有1个元素的元组，则第一个元素的后面需要加一个逗号tuple = ('hi',)这里的逗号是不能省略的。这个设计是为了区分一个正常的括号内放一个元素的这种情况，如果不加这个括号，有的时候Python会直接将它等价成一个没有括号的值。

在接受元组的值的时候我们也可以通过元组型的变量进行接受。接受后我们可以通过其中的单一变量对返回的多个值中的某一个进行访问

元组的使用

```
tuple = (1, 2, 'hi')

print len(tuple)
print tuple[2]
tuple = (1, 2, 'bye')
print tuple
```

```
3
hi
(1, 2, 'bye')

Process exited with code 0
```

用元组做返回值

```
def plus_one(tuple):
    return tuple[0] + 1, tuple[1] + 1, tuple[2] + 1

t = (1, 4, -1)
(x, y, z) = plus_one(t)
print x
print y, z
```

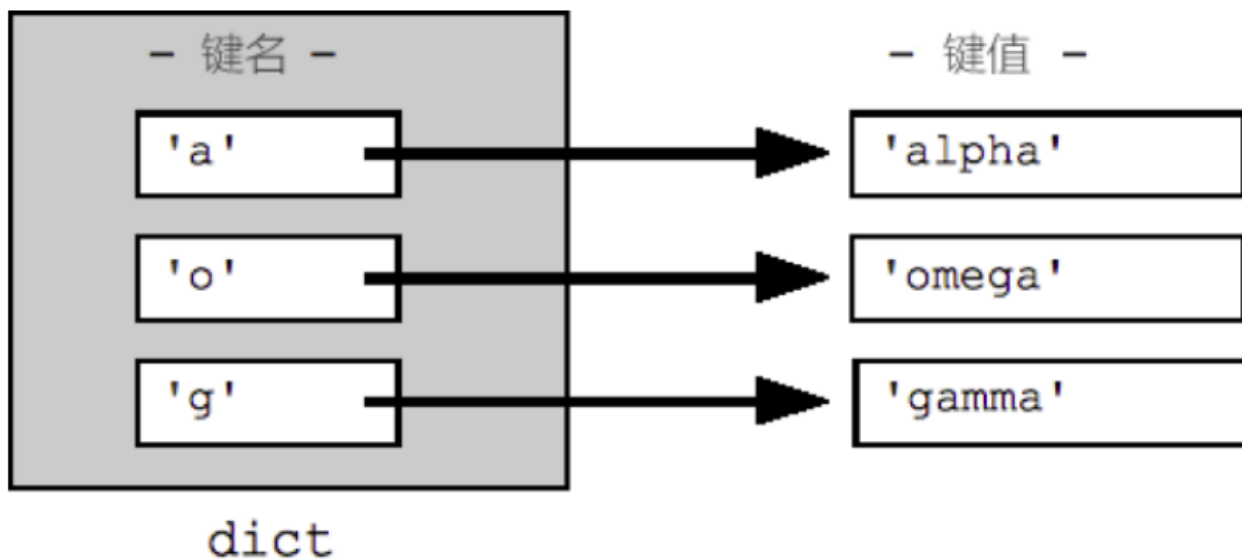
```
2
5 0

Process exited with code 0
```

什么是字典

在Python中有一个将"名字"和"值"进行配对的结构，叫做字典。就像是一个生活中的字典的每一个词有一个具体的解释一样，Python中的字典也包括一系列的类似配对，我们称之为**键值对**，这里每个配对的"名字"叫做键名，这里的每个"值"我们称为"键值"一个Python中的键值对的形式为冒号分隔键名和键值的形式，如键名:键值

字典在Python被定义为一系列用{}包裹键值对的集合。取键值时，需要用字典对应的变量名加放在方括号里键名的形式，例如对于dict = {'key1':100, 'key2':112, ...}的定义，访问print dict['key1']将会得到一个对应的100的输出，类似的，我们也可以使用dict['key1'] = 999的形式修改dict字典中key1键名对应的键值为999



可以作为键名的常见数据类型包括了字符串，数组和元组，而键值则可以是任何类型的数值。如果使用了错误的键名，将无法取出正确的键值

使用字典

```
bat = {}
bat['b'] = 'baidu'
bat['a'] = 'alibaba'
bat['t'] = 'tencent'
print bat
print bat['a']
bat['a'] = 'amazon'
print bat['a']
print 'b' in bat
print 'x' in bat
```

```
{'a': 'alibaba', 'b': 'baidu', 't': 'tencent'}
alibaba
amazon
True
False

Process exited with code 0
```

查看字典元素


```
bat = {'a': 'alibaba', 'b': 'baidu', 't': 'tencent'}
print bat.keys()
print bat.values()
print bat.items()
```

```
['a', 'b', 't']
['alibaba', 'baidu', 'tencent']
[('a', 'alibaba'), ('b', 'baidu'), ('t', 'tencent')]
```

```
Process exited with code 0
```

for循环打印字典

```
bat = {'a': 'alibaba', 'b': 'baidu', 't': 'tencent'}
for value in bat.values() :
    print value
for key in bat :
    print key
for k, v in bat.items() :
    print k, '>', v
```

```
alibaba
baidu
tencent
a
b
t
a > alibaba
b > baidu
t > tencent
```

```
Process exited with code 0
```

字典数据格式化

```
boss = {}
boss['name'] = 'robin'
boss['age'] = 45
boss['height'] = 1.78
print 'The boss named %(name)s is %(age)d-year-old and %(height)g tall.' %boss
```

```
The boss named robin is 45-year-old and 1.78 ta
ll.
```

```
Process exited with code 0
```

删除表达式

```
num = 6
list = ['a', 'b', 'c', 'd']
dict = {'a': 1, 'b': 2, 'c': 3}
del list[0]
del list[-2 :]
print list
del dict['b']
print dict
del num
print num
```

```
['b']
{'a': 1, 'c': 3}
Traceback (most recent call last):
  File "main.py", line 10, in <module>
    print num
NameError: name 'num' is not defined

Process exited with code 1
```

```
num = 6
list = ['a', 'b', 'c', 'd']
dict = {'a': 1, 'b': 2, 'c': 3}
del list[0]
del list[-2 :]
print list
del dict['b']
print dict
num
print num
```

```
['b']
{'a': 1, 'c': 3}
6

Process exited with code 0
```

文件的使用

在Python有一个open函数，这个函数会返回一个可以用于读出和写入文件的文件操作符。我们可以通过fd = open('filename', 'r')的方式，打开一个文件名叫做filename的文件获取它的文件操作符，并让变量fd指向这个操作符。当我们用完这个文件不再继续使用时，我们可以用fd.close()结束对文件的使用

```
# 按行输出整个文件
f = open('filename', 'rU')
for line in f:    # 访问文件每一行
    print line,   # 打印每一行，加逗号可以不被额外添加换行
```

每次读取一行的好处在于我们不会受到内存的限制，也就是说，我们可以每次只把文件的一部分放到内存进行处理，而不会需要一次性完整加载大块头文件到内存中。

我们也可以通过调用文件操作符的函数readline(写成fd.readline()),它会一次性加载完整的文件到内存，让文件的每一行作为它这个列表结构的每一个字符串元素。类似的，文件操作符的函数read(写成fd.read())则会一次性将完整文件作为一个字符串读入到内存中，但是着两种方法往往在处理大文件的时候会遇到内存无法完整存下内容的问题。

对于向文件中的写入，我们可以使用write函数(写成fd.write(字符串))将指定的字符串写入到打开的文件中我们也可以通过print >> 文件操作符，字符串的形式达成同样的目的

文件与编码

在Python中有一些已经写好的功能性的“包”称为模组。其中有一个名为codecs的模组，它提供了读取一个非英文的文件所需要的unicode读取支持

使用它时，我们需要通过import将它引入之后在打开文件时，标记明确所需要使用的编码字符集

```
import codecs
fd = codecs.open('foo.txt', 'rU', 'utf-8')
```

当读取完并且成功完成相关处理后，我们需要注意。我们只可以使用fd.write()的形式来进行写出，print并没有完全支持unicode的写出过程。

正则表达式

正则表达式时是一种用于匹配文本形式的强大逻辑表达式，在Python中的re模組提供了正则表达式的支持。正则表达式由一些普通字符和一些元字符组成。普通字符包括大小写的字母和数字，而元字符则具有特殊的含义。

当正则表达式为一个普通字符串的时候，一个正则表达式的匹配行为就是一个普通字符串查找的过程，例:正则表达式"testing"中没有包含任何元字符，它可以匹配"testing"和"testing123"等字符串，但因为大小写敏感，它不会匹配"Testing"。其他一些元字符则不会作为普通字符来处理，他们包括.^\$*+?{}[]|()。

.会匹配除了换行以外的任何字符，\w等价与[a-zA-Z0-9_]会匹配单一字母，数字或下划线字符，而\W则会匹配任何非字母，数字和下划线的单一字符；\b会匹配"单一字母，数字或下划线字符"和"任何非字母，数字和下划线"之间的边界，\s等效于[\n\r\t\f] (包括空行，换行，返回，制表符，表格)，\S则匹配所有非空白字符；\t \n \r依次用于匹配制表符，换行符，返回符；\d等价于[0-9]用于匹配十进制表示的数字。

^作为开始标记，\$作为结束标记，分别用于标记一个字符串的开始和结束位置。\\用于一些字符的转义，比如\\.表示对于一个真实点字符的匹配，\\表示对于一个真实反斜杠字符的匹配等。如果你对不是很确定的字符是否需要转义才能匹配，你大可以加上斜杠，如果对于@你携程\\@时一定没有问题的。

正则表达式查找

```
import re
str = 'A cute word:cat!!'
match = re.search(r'word:\w\w\w', str) //这个正则表达式从字符串str进行匹配查找，正则表达式前的r标记了这个表达式不需要被转义处理，也就是说\n这样的东西在写了r标记的情况下不会被理解成换行
if match:
    print 'found', match.group()
```

```
found word:cat
```

```
Process exited with code 0
```

基础正则使用

```
import re
print re.search(r'..g', 'piiig').group()
print re.search(r'\d\d\d', 'p123g').group()
print re.search(r'\w\w\w', '@abcd!!').group()
```

```
iig  
123  
abc
```

```
Process exited with code 0
```

正则表达式重复

```
import re  
print re.search(r'pi+', 'piiig').group()  
print re.search(r'pi*', 'pg').group()
```

```
piii  
p
```

```
Process exited with code 0
```

```
import re  
print re.search(r'pi+', 'piiig').group()  
print re.search(r'pi*', 'pg').group()  
print re.search(r'pi*', 'piiig').group()  
print re.search(r'pi+', 'pg').group()
```

```
piii
p
piii
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    print re.search(r'pi+', 'pg').group()
AttributeError: 'NoneType' object has no attribute 'group'

Process exited with code 1
```

正则里的方括号

```
import re
print re.search(r'[abc]+', 'xxxacbbcbbadddededede').group()
print re.search(r'[a-d]+', 'xxxacbbcbbadddededede').group()
```

```
acbbcbba
acbbcbbaddd
```

```
Process exited with code 0
```

正则提取

```
import re
str = 'purple alice-b@jisuanke.com monkey dishwasher'
match = re.search('([\w.-]+)@([\w.-]+)', str)
if match:
    print match.group()
    print match.group(1)
    print match.group(2)
```

```
alice-b@jisuanke.com
alice-b
jisuanke.com
```

```
Process exited with code 0
```

正则表达式的调试

正则表达式用简单的一些字符的组合包含了太丰富的语义但是它们实在是太过密集了。

你可以设计一系列放在列表里的字符串用于调试，其中一部分是可以产生符合正则表达式的结果的，另一部分是产生不符合正则表达式的结果的，在设计这些字符时，尽可能让它们的特征表现的更为不同一些，便于覆盖到我们可能出现的各种正则表达式没有写对的错误。例如，对于一个存在+的正则表达式，我们可以考虑选用一个符合*但是不符合+的字符串。

然后你可以写一个循环依次验证每个列表内的字符串是否符合指定的某个正则表达式并且和你设定的存在另一个列表内的预期结果进行对比，如果出现了不一致的情况，则你应该考虑看看你的正则表达式是不是还需要修改，如果结果基本一致，那么我们可以考虑进一步修改我们用于调试的字符串或添加新字符串。

查找所有方法

```
import re
str = 'purple alice@jisuanke.com, blah monkey bob@abc.com blah dishwasher'
tuples = re.findall(r'([\w\.-]+)@([\w\.-]+)', str)
print tuples
```

```
[('alice', 'jisuanke.com'), ('bob', 'abc.com')]
```

```
Process exited with code 0
```

选项与贪心匹配

在用于正则表达式的re模组中的函数有一些可选参数，我们可以对search()函数或者findall()函数传入额外的参数来进行使用，如re.search(pat, str, re.IGNORECASE)中的re.IGNORECASE就是使用了re中的一个标记作为额外的参数。

在re模组中，提供了很多不同的可选参数，其中上面提到的re.IGNORECASE表示了让匹配时忽略大小写的区别；而另外一个可选参数DOTALL如果被添加，则会允许正则中去跨行匹配，加了这个参数以后.*这样的匹配方式，将可以跨行进行匹配，而不只是在行内进行。另外，还有一个可选参数是MULTILINE，使用他以后，对于一个多行文本组成的字符串，^和\$只会匹配整个字符串的开始和结束。

除了可选参数之外，我们还需要理解一下正则匹配的“贪心情况”。假设我们有一段文字 `fooand<i>son</i>` 而你希望匹配 `(<.*>)` 提供的所有HTML标签。

事实上结果会出乎你的意料，因为 `.*` 这样的匹配是“贪心”的，它会尽可能去得到较长的匹配结果，因此我们会得到的是一整个 `fooand<i>son</i>` 作为匹配出的结果。如果我们希望获得期望中的结果，我们就需要这个匹配的非贪心的，在正则表达式中，我们对于 `*` 和 `+` 这种默认贪心的匹配可以加上 `?` 使之变为不贪心的。

也就是说，如果我们将 `(<.*>)` 改为 `(<.*?>)`，正则表达还是会先匹配 `` 后匹配 `` 接下来则分别时 `<i>` 和 `</i>`。这样的匹配结果与我们预期完全一致，相应的，对于一些使用了 `+` 的情况，我们可以将 `+` 变为 `+` 的情况来进行非贪心匹配。

由于Python 默认采用 ASCII 编码，若在代码中需要使用 中文，请在第一行声明：

```
# coding=utf-8
```