Python3基础

1，Python3 基础语法

1-1 编码

默认情况下，Python 3 源码文件以 **UTF-8** 编码，所有字符串都是 unicode 字符串。 当然你也可以为源码文件指定不同的编码：

```
# -*- coding: cp-1252 -*-
```

上述定义允许在源文件中使用 Windows-1252 字符集中的字符编码，对应适合语言为保加利亚语、白罗斯语、马其顿语、俄语、塞尔维亚语。

1-2 标识符

- 第一个字符必须是字母表中字母或下划线 _ 。
- 标识符的其他的部分由字母、数字和下划线组成。
- 标识符对大小写敏感。

在 Python 3 中，可以用中文作为变量名，非 ASCII 标识符也是允许的了。

1-3 python保留字

保留字即关键字，我们不能把它们用作任何标识符名称。Python 的标准库提供了一个 keyword 模块，可以输出当前版本的所有关键字：

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

1-4 注释

Python中单行注释以 **#** 开头，实例如下

```
#!/usr/bin/python3

# 第一个注释
print ("Hello, Python!") # 第二个注释
```

执行以上代码，输出结果为：

```
Hello, Python!
```

多行注释可以用多个 # 号，还有 ''' 和 """：

```
#!/usr/bin/python3

# 第一个注释
# 第二个注释

'''
第三注释
第四注释
'''

"""
第五注释
第六注释
"""
print ("Hello, Python!")
```

执行以上代码，输出结果为：

```
Hello, Python!
```

1-5 行与缩进

python最具特色的就是使用缩进来表示代码块，不需要使用大括号 **{}** 。

缩进的空格数是可变的，但是同一个代码块的语句必须包含相同的缩进空格数。实例如下

```
if True:
    print ("True")
else:
    print ("False")
```

以下代码最后一行语句缩进数的空格数不一致，会导致运行错误：

```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
  print ("False")    # 缩进不一致，会导致运行错误
```

以上程序由于缩进不一致，执行后会出现类似以下错误：

```
 File "test.py", line 6
    print ("False")    # 缩进不一致，会导致运行错误
                                    ^
IndentationError: unindent does not match any outer indentation level
```

1-6 多行语句

Python 通常是一行写完一条语句，但如果语句很长，我们可以使用反斜杠()来实现多行语句，例如：

```
total = item_one + \
        item_two + \
        item_three
```

在 [], {}, 或 () 中的多行语句，不需要使用反斜杠()，例如：

```
total = ['item_one', 'item_two', 'item_three',
        'item_four', 'item_five']
```

1-7 数字(Number)类型

python中数字有四种类型：整数、布尔型、浮点数和复数。

- **int** (整数), 如 1, 只有一种整数类型 int，表示为长整型，没有 python2 中的 Long。
- **bool** (布尔), 如 True。
- **float** (浮点数), 如 1.23、3E-2
- **complex** (复数), 如 1 + 2j、 1.1 + 2.2j

1-8 字符串(String)

- python中单引号和双引号使用完全相同。
- 使用三引号('''或""")可以指定一个多行字符串。
- 转义符 ''
- 反斜杠可以用来转义，使用r可以让反斜杠不发生转义。。 如 r"this is a line with \n" 则\n会显示，并不是换行。
- 按字面意义级联字符串，如"this " "is " "string"会被自动转换为this is string。
- 字符串可以用 + 运算符连接在一起，用 * 运算符重复。
- Python 中的字符串有两种索引方式，从左往右以 0 开始，从右往左以 -1 开始。
- Python中的字符串不能改变。
- Python 没有单独的字符类型，一个字符就是长度为 1 的字符串。
- 字符串的截取的语法格式如下：**变量[头下标:尾下标:步长]**

```
word = '字符串'
sentence = "这是一个句子。"
paragraph = """这是一个段落，
可以由多行组成"""
```

```
#!/usr/bin/python3

str='Runoob'

print(str)                 # 输出字符串
print(str[0:-1])           # 输出第一个到倒数第二个的所有字符
print(str[0])              # 输出字符串第一个字符
print(str[2:5])            # 输出从第三个开始到第五个的字符
print(str[2:])             # 输出从第三个开始后的所有字符
print(str * 2)             # 输出字符串两次
```

```
print(str + '你好')          # 连接字符串

print('------------------------------')

print('hello\nrunoob')      # 使用反斜杠(\)+n转义特殊字符
print(r'hello\nrunoob')     # 在字符串前面添加一个 r，表示原始字符串，不会发生转义
```

这里的 r 指 raw，即 raw string。

输出结果为：

```
Runoob
Runoo
R
noo
noob
RunoobRunoob
Runoob你好
------------------------------
hello
runoob
hello\nrunoob
```

1-9 空行

函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

空行与代码缩进不同，空行并不是Python语法的一部分。书写时不插入空行，Python解释器运行也不会出错。但是空行的作用在于分隔两段不同功能或含义的代码，便于日后代码的维护或重构。

记住：空行也是程序代码的一部分。

1-10 等待用户输入

执行下面的程序在按回车键后就会等待用户输入：

```
#!/usr/bin/python3

input("\n\n按下 enter 键后退出。")
```

以上代码中，"\n\n"在结果输出前会输出两个新的空行。一旦用户按下 enter 键时，程序将退出。

1-11 同一行显示多条语句

Python可以在同一行中使用多条语句，语句之间使用分号(;)分割，以下是一个简单的实例：

```
#!/usr/bin/python3

import sys; x = 'runoob'; sys.stdout.write(x + '\n')
```

使用脚本执行以上代码，输出结果为：

```
runoob
```

使用交互式命令行执行，输出结果为：

```
>>> import sys; x = 'runoob'; sys.stdout.write(x + '\n')
runoob
7
```

此处的 7 表示字符数。

1-12 多个语句构成代码组

缩进相同的一组语句构成一个代码块，我们称之代码组。

像if、while、def和class这样的复合语句，首行以关键字开始，以冒号(:)结束，该行之后的一行或多行代码构成代码组。

我们将首行及后面的代码组称为一个子句(clause)。

如下实例：

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

1-13 Print 输出

print 默认输出是换行的，如果要实现不换行需要在变量末尾加上 **end=""**：

```
#!/usr/bin/python3

x="a"
y="b"
# 换行输出
print( x )
print( y )

print('---------')
# 不换行输出
print( x, end=" " )
print( y, end=" " )
print()
```

以上实例执行结果为：

```
a
b
---------
a b
```

1-14 import 与 from...import

- 在 python 用 **import** 或者 **from...import** 来导入相应的模块。
- 将整个模块(somemodule)导入，格式为： **import somemodule**
- 从某个模块中导入某个函数,格式为： **from somemodule import somefunction**
- 从某个模块中导入多个函数,格式为： **from somemodule import firstfunc, secondfunc, thirdfunc**
- 将某个模块中的全部函数导入，格式为： **from somemodule import ***

```
# 导入 sys 模块
import sys
print('================Python import mode==========================')
print ('命令行参数为:')
for i in sys.argv:
    print (i)
print ('\n python 路径为',sys.path)
```

```
# 导入 sys 模块的 argv,path 成员
from sys import argv,path  #  导入特定的成员

print('================python from import===================================')
print('path:',path) # 因为已经导入path成员，所以此处引用时不需要加sys.path
```

1-15 命令行参数

很多程序可以执行一些操作来查看一些基本信息，Python可以使用-h参数查看各参数帮助信息：

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser (also PYTHONDEBUG=x)
-E     : ignore environment variables (such as PYTHONPATH)
-h     : print this help message and exit

[ etc. ]
```

我们在使用脚本形式执行 Python 时，可以接收命令行输入的参数，具体使用可以参照 Python 3 命令行参数window.location='https://www.runoob.com/python3/dml.html')。

2，Python3 基本数据类型

- Python 中的变量不需要声明。每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。
- 在 Python 中，变量就是变量，它没有类型，我们所说的"类型"是变量所指的内存中对象的类型。
- 等号（=）用来给变量赋值。
- 等号（=）运算符左边是一个变量名,等号（=）运算符右边是存储在变量中的值。例如：

```
#!/usr/bin/python3

counter = 100          # 整型变量
miles   = 1000.0       # 浮点型变量
name    = "runoob"     # 字符串

print (counter)
print (miles)
print (name)
```

执行以上程序会输出如下结果：

```
100
1000.0
runoob
```

2-1 多个变量赋值

Python允许你同时为多个变量赋值。例如：

```
a = b = c = 1
```

以上实例，创建一个整型对象，值为 1，从后向前赋值，三个变量被赋予相同的数值。

您也可以为多个对象指定多个变量。例如：

```
a, b, c = 1, 2, "runoob"
```

以上实例，两个整型对象 1 和 2 的分配给变量 a 和 b，字符串对象 "runoob" 分配给变量 c。

2-2 标准数据类型

Python3 中有六个标准的数据类型：

- Number（数字）
- String（字符串）
- List（列表）
- Tuple（元组）
- Set（集合）
- Dictionary（字典）

Python3 的六个标准数据类型中：

- **不可变数据（3 个）：** Number（数字）、String（字符串）、Tuple（元组）；
- **可变数据（3 个）：** List（列表）、Dictionary（字典）、Set（集合）。

2-3 Number（数字）

Python3 支持 **int、float、bool、complex（复数）**。

在Python 3里，只有一种整数类型 int，表示为长整型，没有 python2 中的 Long。

像大多数语言一样，数值类型的赋值和计算都是很直观的。

内置的 type() 函数可以用来查询变量所指的对象类型。

```
>>> a, b, c, d = 20, 5.5, True, 4+3j
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

此外还可以用 isinstance 来判断：

```
>>>a = 111
>>> isinstance(a, int)
True
>>>
```

isinstance 和 type 的区别在于：

- type()不会认为子类是一种父类类型。
- isinstance()会认为子类是一种父类类型。

```
>>> class A:
...     pass
...
>>> class B(A):
...     pass
...
>>> isinstance(A(), A)
True
>>> type(A()) == A
True
>>> isinstance(B(), A)
True
>>> type(B()) == A
False
```

> **注意：**在 Python2 中是没有布尔型的，它用数字 0 表示 False，用 1 表示 True。到 Python3 中，把 True 和 False 定义成关键字了，但它们的值还是 1 和 0，它们可以和数字相加。

当你指定一个值时，Number 对象就会被创建：

```
var1 = 1
var2 = 10
```

您也可以使用del语句删除一些对象引用。

del语句的语法是：

```
del var1[,var2[,var3[....,varN]]]
```

您可以通过使用del语句删除单个或多个对象。例如：

```
del var
del var_a, var_b
```

2-4 数值运算

```
>>>5 + 4  # 加法
9
>>> 4.3 - 2 # 减法
2.3
>>> 3 * 7  # 乘法
21
>>> 2 / 4  # 除法，得到一个浮点数
0.5
>>> 2 // 4 # 除法，得到一个整数
0
>>> 17 % 3 # 取余
2
>>> 2 ** 5 # 乘方
32
```

注意：

- 1、Python可以同时为多个变量赋值，如a, b = 1, 2。
- 2、一个变量可以通过赋值指向不同类型的对象。
- 3、数值的除法包含两个运算符：**/** 返回一个浮点数，**//** 返回一个整数。
- 4、在混合计算时，Python会把整型转换成为浮点数。

2-5 数值类型实例

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3e+18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2E-12	4.53e-7j

2-6 String（字符串）

Python中的字符串用单引号 ' 或双引号 " 括起来，同时使用反斜杠 ** 转义特殊字符。

字符串的截取的语法格式如下：

```
变量[头下标:尾下标]
```

加号 **+** 是字符串的连接符， 星号 ***** 表示复制当前字符串，紧跟的数字为复制的次数。实例如下：

```
#!/usr/bin/python3

str = 'Runoob'

print (str)          # 输出字符串
print (str[0:-1])    # 输出第一个到倒数第二个的所有字符
print (str[0])       # 输出字符串第一个字符
print (str[2:5])     # 输出从第三个开始到第五个的字符
print (str[2:])      # 输出从第三个开始的后的所有字符
print (str * 2)      # 输出字符串两次
print (str + "TEST") # 连接字符串
```

执行以上程序会输出如下结果：

```
Runoob
Runoo
R
noo
noob
RunoobRunoob
RunoobTEST
```

Python 使用反斜杠()转义特殊字符，如果你不想让反斜杠发生转义，可以在字符串前面添加一个 r，表示原始字符串：

```
>>> print('Ru\noob')
Ru
oob
>>> print(r'Ru\noob')
Ru\noob
>>>
```

另外，反斜杠()可以作为续行符，表示下一行是上一行的延续。也可以使用 """...""" 或者 '''...''' 跨越多行。

注意，Python 没有单独的字符类型，一个字符就是长度为1的字符串。

```
>>>word = 'Python'
>>> print(word[0], word[5])
P n
>>> print(word[-1], word[-6])
n P
```

与 C 字符串不同的是，Python 字符串不能被改变。向一个索引位置赋值，比如word[0] = 'm'会导致错误。

注意：

- 1、反斜杠可以用来转义，使用r可以让反斜杠不发生转义。
- 2、字符串可以用+运算符连接在一起，用*运算符重复。
- 3、Python中的字符串有两种索引方式，从左往右以0开始，从右往左以-1开始。
- 4、Python中的字符串不能改变。

2-7 List（列表）

- List（列表） 是 Python 中使用最频繁的数据类型。
- 列表可以完成大多数集合类的数据结构实现。列表中元素的类型可以不相同，它支持数字，字符串甚至可以包含列表（所谓嵌套）。
- 列表是写在方括号 [] 之间、用逗号分隔开的元素列表。
- 和字符串一样，列表同样可以被索引和截取，列表被截取后返回一个包含所需元素的新列表。
- 列表截取的语法格式如下

加号 + 是列表连接运算符，星号 * 是重复操作。如下实例：

```
#!/usr/bin/python3

list = [ 'abcd', 786 , 2.23, 'runoob', 70.2 ]
tinylist = [123, 'runoob']

print (list)            # 输出完整列表
print (list[0])         # 输出列表第一个元素
print (list[1:3])       # 从第二个开始输出到第三个元素
print (list[2:])        # 输出从第三个元素开始的所有元素
print (tinylist * 2)    # 输出两次列表
print (list + tinylist) # 连接列表
```

以上实例输出结果：

```
['abcd', 786, 2.23, 'runoob', 70.2]
abcd
[786, 2.23]
[2.23, 'runoob', 70.2]
[123, 'runoob', 123, 'runoob']
['abcd', 786, 2.23, 'runoob', 70.2, 123, 'runoob']
```

与Python字符串不一样的是，列表中的元素是可以改变的：

```
>>>a = [1, 2, 3, 4, 5, 6]
>>> a[0] = 9
>>> a[2:5] = [13, 14, 15]
>>> a
[9, 2, 13, 14, 15, 6]
>>> a[2:5] = []   # 将对应的元素值设置为 []
>>> a
[9, 2, 6]
```

List 内置了有很多方法，例如 append()、pop() 等等，这在后面会讲到。

注意：

- 1、List写在方括号之间，元素用逗号隔开。
- 2、和字符串一样，list可以被索引和切片。
- 3、List可以使用+操作符进行拼接。
- 4、List中的元素是可以改变的。

Python 列表截取可以接收第三个参数，参数作用是截取的步长，以下实例在索引 1 到索引 4 的位置并设置为步长为 2（间隔一个位置）来截取字符串：

如果第三个参数为负数表示逆向读取，以下实例用于翻转字符串：

```
def reverseWords(input):

    # 通过空格将字符串分隔符，把各个单词分隔为列表
    inputWords = input.split(" ")

    # 翻转字符串
    # 假设列表 list = [1,2,3,4],
    # list[0]=1, list[1]=2 ，而 -1 表示最后一个元素 list[-1]=4 ( 与 list[3]=4 一样)
    # inputWords[-1::-1] 有三个参数
    # 第一个参数 -1 表示最后一个元素
    # 第二个参数为空，表示移动到列表末尾
    # 第三个参数为步长，-1 表示逆向
    inputWords=inputWords[-1::-1]

    # 重新组合字符串
    output = ' '.join(inputWords)

    return output

if __name__ == "__main__":
    input = 'I like runoob'
    rw = reverseWords(input)
    print(rw)
```

输出结果为：

```
runoob like I
```

2-8 Tuple（元组）

元组（tuple）与列表类似，不同之处在于元组的元素不能修改。元组写在小括号 **()** 里，元素之间用逗号隔开。

元组中的元素类型也可以不相同：

```
#!/usr/bin/python3

tuple = ( 'abcd', 786 , 2.23, 'runoob', 70.2  )
tinytuple = (123, 'runoob')

print (tuple)             # 输出完整元组
print (tuple[0])          # 输出元组的第一个元素
print (tuple[1:3])        # 输出从第二个元素开始到第三个元素
print (tuple[2:])         # 输出从第三个元素开始的所有元素
print (tinytuple * 2)     # 输出两次元组
print (tuple + tinytuple) # 连接元组
```

以上实例输出结果：

```
('abcd', 786, 2.23, 'runoob', 70.2)
abcd
(786, 2.23)
(2.23, 'runoob', 70.2)
(123, 'runoob', 123, 'runoob')
('abcd', 786, 2.23, 'runoob', 70.2, 123, 'runoob')
```

元组与字符串类似，可以被索引且下标索引从0开始，-1 为从末尾开始的位置。也可以进行截取（看上面，这里不再赘述）。

其实，可以把字符串看作一种特殊的元组。

```
>>>tup = (1, 2, 3, 4, 5, 6)
>>> print(tup[0])
1
>>> print(tup[1:5])
(2, 3, 4, 5)
>>> tup[0] = 11  # 修改元组元素的操作是非法的
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

虽然tuple的元素不可改变，但它可以包含可变的对象，比如list列表。

构造包含 0 个或 1 个元素的元组比较特殊，所以有一些额外的语法规则：

```
tup1 = ()    # 空元组
tup2 = (20,) # 一个元素，需要在元素后添加逗号
```

string、list 和 tuple 都属于 sequence（序列）。

注意：

- 1、与字符串一样，元组的元素不能修改。
- 2、元组也可以被索引和切片，方法一样。
- 3、注意构造包含 0 或 1 个元素的元组的特殊语法规则。
- 4、元组也可以使用+操作符进行拼接。

2-9 Set（集合）

集合（set）是由一个或数个形态各异的大小整体组成的，构成集合的事物或对象称作元素或是成员。

基本功能是进行成员关系测试和删除重复元素。

可以使用大括号 **{ }** 或者 **set()** 函数创建集合，注意：创建一个空集合必须用 **set()** 而不是 **{ }**，因为 **{ }** 是用来创建一个空字典。

创建格式：

```
parame = {value01,value02,...}
或者
set(value)
```

```
#!/usr/bin/python3
```

```
student = {'Tom', 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}

print(student)   # 输出集合，重复的元素被自动去掉

# 成员测试
if 'Rose' in student :
    print('Rose 在集合中')
else :
    print('Rose 不在集合中')


# set可以进行集合运算
a = set('abracadabra')
b = set('alacazam')

print(a)

print(a - b)     # a 和 b 的差集

print(a | b)     # a 和 b 的并集

print(a & b)     # a 和 b 的交集

print(a ^ b)     # a 和 b 中不同时存在的元素
```

以上实例输出结果：

```
{'Mary', 'Jim', 'Rose', 'Jack', 'Tom'}
Rose 在集合中
{'b', 'a', 'c', 'r', 'd'}
{'b', 'd', 'r'}
{'l', 'r', 'a', 'c', 'z', 'm', 'b', 'd'}
{'a', 'c'}
{'l', 'r', 'z', 'm', 'b', 'd'}
```

2-10 Dictionary（字典）

- 字典（dictionary）是Python中另一个非常有用的内置数据类型。
- 列表是有序的对象集合，字典是无序的对象集合。两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。
- 字典是一种映射类型，字典用 **{ }** 标识，它是一个无序的 **键(key) : 值(value)** 的集合。
- 键(key)必须使用不可变类型。
- 在同一个字典中，键(key)必须是唯一的。

```
#!/usr/bin/python3

dict = {}
dict['one'] = "1 - 菜鸟教程"
dict[2]     = "2 - 菜鸟工具"

tinydict = {'name': 'runoob','code':1, 'site': 'www.runoob.com'}


print (dict['one'])       # 输出键为 'one' 的值
print (dict[2])           # 输出键为 2 的值
print (tinydict)          # 输出完整的字典
```

```
print (tinydict.keys())   # 输出所有键
print (tinydict.values()) # 输出所有值
```

以上实例输出结果：

```
1 - 菜鸟教程
2 - 菜鸟工具
{'name': 'runoob', 'code': 1, 'site': 'www.runoob.com'}
dict_keys(['name', 'code', 'site'])
dict_values(['runoob', 1, 'www.runoob.com'])
```

构造函数 dict() 可以直接从键值对序列中构建字典如下：

```
>>>dict([('Runoob', 1), ('Google', 2), ('Taobao', 3)])
{'Taobao': 3, 'Runoob': 1, 'Google': 2}

>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}

>>> dict(Runoob=1, Google=2, Taobao=3)
{'Runoob': 1, 'Google': 2, 'Taobao': 3}
```

另外，字典类型也有一些内置的函数，例如clear()、keys()、values()等。

注意：

- 1、字典是一种映射类型，它的元素是键值对。
- 2、字典的关键字必须为不可变类型，且不能重复。
- 3、创建空字典使用 **{ }**。

2-11 Python数据类型转换

有时候，我们需要对数据内置的类型进行转换，数据类型的转换，你只需要将数据类型作为函数名即可。

以下几个内置的函数可以执行数据类型之间的转换。这些函数返回一个新的对象，表示转换的值。

函数	描述
int(x [,base])	将x转换为一个整数
float(x)	将x转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效Python表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。d 必须是一个 (key, value)元组序列。
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

3，Python3 解释器

Linux/Unix的系统上，一般默认的 python 版本为 2.x，我们可以将 python3.x 安装在 **/usr/local/python3** 目录中。

安装完成后，我们可以将路径 **/usr/local/python3/bin** 添加到您的 Linux/Unix 操作系统的环境变量中，这样您就可以通过 shell 终端输入下面的命令来启动 Python3 。

```
$ PATH=$PATH:/usr/local/python3/bin/python3     # 设置环境变量
$ python3 --version
Python 3.4.0
```

在Window系统下你可以通过以下命令来设置Python的环境变量，假设你的Python安装在 C:\Python34 下:

```
set path=%path%;C:\python34
```

3-1 交互式编程

我们可以在命令提示符中输入"Python"命令来启动Python解释器：

```
$ python3
```

执行以上命令后，出现如下窗口信息：

```
$ python3
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

在 python 提示符中输入以下语句，然后按回车键查看运行效果：

```
print ("Hello, Python!");
```

以上命令执行结果如下：

```
Hello, Python!
```

当键入一个多行结构时，续行是必须的。我们可以看下如下 if 语句：

```
>>> flag = True
>>> if flag :
...     print("flag 条件为 True!")
...
flag 条件为 True!
```

3-2 脚本式编程

将如下代码拷贝至 **hello.py**文件中：

```
print ("Hello, Python!");
```

通过以下命令执行该脚本：

```
python3 hello.py
```

输出结果为：

```
Hello, Python!
```

在Linux/Unix系统中，你可以在脚本顶部添加以下命令让Python脚本可以像SHELL脚本一样可直接执行：

```
#! /usr/bin/env python3
```

然后修改脚本权限，使其有执行权限，命令如下：

```
$ chmod +x hello.py
```

执行以下命令：

```
./hello.py
```

输出结果为：

```
Hello, Python!
```

4，Python3 注释

确保对模块, 函数, 方法和行内注释使用正确的风格

Python中的注释有单行注释和多行注释：

Python中单行注释以 # 开头，例如：：

```
# 这是一个注释
print("Hello, World!")
```

多行注释用三个单引号 ''' 或者三个双引号 """ 将注释括起来，例如:

4-1 单引号（'''）

```
#!/usr/bin/python3
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
print("Hello, World!")
```

4-2 双引号（"""）

```
#!/usr/bin/python3
"""
这是多行注释，用三个双引号
这是多行注释，用三个双引号
这是多行注释，用三个双引号
"""
print("Hello, World!")
```

5，Python运算符

5-1 Python算术运算符

以下假设变量a为10，变量b为21：

运算符	描述	实例
+	加 - 两个对象相加	a + b 输出结果 31
-	减 - 得到负数或是一个数减去另一个数	a - b 输出结果 -11
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 210
/	除 - x 除以 y	b / a 输出结果 2.1
%	取模 - 返回除法的余数	b % a 输出结果 1
**	幂 - 返回x的y次幂	a**b 为10的21次方

//	取整除 - 向下取接近除数的整数	`>>> 9//2`
		`4`
		`>>> -9//2`
		`-5`

以下实例演示了Python所有算术运算符的操作：

```
#!/usr/bin/python3

a = 21
b = 10
c = 0

c = a + b
print ("1 - c 的值为：", c)

c = a - b
print ("2 - c 的值为：", c)

c = a * b
print ("3 - c 的值为：", c)

c = a / b
print ("4 - c 的值为：", c)

c = a % b
print ("5 - c 的值为：", c)

# 修改变量 a 、b 、c
a = 2
b = 3
c = a**b
print ("6 - c 的值为：", c)

a = 10
b = 5
c = a//b
print ("7 - c 的值为：", c)
```

以上实例输出结果：

```
1 - c 的值为： 31
2 - c 的值为： 11
3 - c 的值为： 210
4 - c 的值为： 2.1
5 - c 的值为： 1
6 - c 的值为： 8
7 - c 的值为： 2
```

5-2 Python比较运算符

以下假设变量a为10，变量b为20：

运算符	描述	实例
==	等于 - 比较对象是否相等	(a == b) 返回 False。
!=	不等于 - 比较两个对象是否不相等	(a != b) 返回 True。
>	大于 - 返回x是否大于y	(a > b) 返回 False。
<	小于 - 返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意，这些变量名的大写。	(a < b) 返回 True。
>=	大于等于 - 返回x是否大于等于y。	(a >= b) 返回 False。
<=	小于等于 - 返回x是否小于等于y。	(a <= b) 返回 True。

以下实例演示了Python所有比较运算符的操作：

```
#!/usr/bin/python3

a = 21
b = 10
c = 0

if ( a == b ):
   print ("1 - a 等于 b")
else:
   print ("1 - a 不等于 b")

if ( a != b ):
   print ("2 - a 不等于 b")
else:
   print ("2 - a 等于 b")

if ( a < b ):
   print ("3 - a 小于 b")
else:
   print ("3 - a 大于等于 b")

if ( a > b ):
   print ("4 - a 大于 b")
else:
   print ("4 - a 小于等于 b")
```

```
# 修改变量 a 和 b 的值
a = 5;
b = 20;
if ( a <= b ):
   print ("5 - a 小于等于 b")
else:
   print ("5 - a 大于  b")

if ( b >= a ):
   print ("6 - b 大于等于 a")
else:
   print ("6 - b 小于 a")
```

以上实例输出结果：

```
1 - a 不等于 b
2 - a 不等于 b
3 - a 大于等于 b
4 - a 大于 b
5 - a 小于等于 b
6 - b 大于等于 a
```

5-3 Python赋值运算符

以下假设变量a为10，变量b为20：

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

:=	海象运算符，可在表达式内部为变量赋值。**Python3.8 版本新增运算符。**	在这个示例中，赋值表达式可以避免调用 len() 两次:
		`if (n := len(a)) > 10:`
		`print(f"List is too long ({n} elements, expected <= 10)")`

```
#!/usr/bin/python3

a = 21
b = 10
c = 0

c = a + b
print ("1 - c 的值为: ", c)
```

```
c += a
print ("2 - c 的值为: ", c)

c *= a
print ("3 - c 的值为: ", c)

c /= a
print ("4 - c 的值为: ", c)

c = 2
c %= a
print ("5 - c 的值为: ", c)

c **= a
print ("6 - c 的值为: ", c)

c //= a
print ("7 - c 的值为: ", c)
```

以上实例输出结果：

```
1 - c 的值为：  31
2 - c 的值为：  52
3 - c 的值为：  1092
4 - c 的值为：  52.0
5 - c 的值为：  2
6 - c 的值为：  2097152
7 - c 的值为：  99864
```

5-4 Python位运算符

按位运算符是把数字看作二进制来进行计算的。Python中的按位运算法则如下：

下表中变量 a 为 60，b 为 13二进制格式如下：

```
a = 0011 1100

b = 0000 1101

-----------------

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a  = 1100 0011
```

运算符	描述	实例
&	按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0	(a & b) 输出结果 12 ，二进制解释： 0000 1100
|	按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。	(a | b) 输出结果 61 ，二进制解释： 0011 1101
^	按位异或运算符：当两对应的二进位相异时，结果为1	(a ^ b) 输出结果 49 ，二进制解释： 0011 0001
~	按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1。`~x` 类似于 `-x-1`	(~a) 输出结果 -61 ，二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。
<<	左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。	a << 2 输出结果 240 ，二进制解释： 1111 0000
>>	右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数	a >> 2 输出结果 15 ，二进制解释： 0000 1111

```
#!/usr/bin/python3

a = 60              # 60 = 0011 1100
b = 13              # 13 = 0000 1101
c = 0

c = a & b;          # 12 = 0000 1100
print ("1 - c 的值为：", c)

c = a | b;          # 61 = 0011 1101
print ("2 - c 的值为：", c)

c = a ^ b;          # 49 = 0011 0001
print ("3 - c 的值为：", c)

c = ~a;             # -61 = 1100 0011
print ("4 - c 的值为：", c)

c = a << 2;         # 240 = 1111 0000
print ("5 - c 的值为：", c)

c = a >> 2;         # 15 = 0000 1111
print ("6 - c 的值为：", c)
```

以上实例输出结果：

```
1 - c 的值为： 12
2 - c 的值为： 61
3 - c 的值为： 49
4 - c 的值为： -61
5 - c 的值为： 240
6 - c 的值为： 15
```

5-5 Python逻辑运算符

Python语言支持逻辑运算符，以下假设变量 a 为 10, b为 20:

运算符	逻辑表达式	描述	实例
and	x and y	布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	(a and b) 返回 20。
or	x or y	布尔"或" - 如果 x 是 True，它返回 x 的值，否则它返回 y 的计算值。	(a or b) 返回 10。
not	not x	布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。	not(a and b) 返回 False

```
#!/usr/bin/python3

a = 10
b = 20

if ( a and b ):
   print ("1 - 变量 a 和 b 都为 true")
else:
   print ("1 - 变量 a 和 b 有一个不为 true")

if ( a or b ):
   print ("2 - 变量 a 和 b 都为 true，或其中一个变量为 true")
else:
   print ("2 - 变量 a 和 b 都不为 true")

# 修改变量 a 的值
a = 0
if ( a and b ):
   print ("3 - 变量 a 和 b 都为 true")
else:
   print ("3 - 变量 a 和 b 有一个不为 true")

if ( a or b ):
   print ("4 - 变量 a 和 b 都为 true，或其中一个变量为 true")
else:
   print ("4 - 变量 a 和 b 都不为 true")

if not( a and b ):
   print ("5 - 变量 a 和 b 都为 false，或其中一个变量为 false")
else:
   print ("5 - 变量 a 和 b 都为 true")
```

以上实例输出结果：

```
1 - 变量 a 和 b 都为 true
2 - 变量 a 和 b 都为 true，或其中一个变量为 true
3 - 变量 a 和 b 有一个不为 true
4 - 变量 a 和 b 都为 true，或其中一个变量为 true
5 - 变量 a 和 b 都为 false，或其中一个变量为 false
```

5-6 Python成员运算符

除了以上的一些运算符之外，Python还支持成员运算符，测试实例中包含了一系列的成员，包括字符串，列表或元组。

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False。	x 在 y 序列中 , 如果 x 在 y 序列中返回 True。
not in	如果在指定的序列中没有找到值返回 True，否则返回 False。	x 不在 y 序列中 , 如果 x 不在 y 序列中返回 True。

```
#!/usr/bin/python3

a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print ("1 - 变量 a 在给定的列表中 list 中")
else:
   print ("1 - 变量 a 不在给定的列表中 list 中")

if ( b not in list ):
   print ("2 - 变量 b 不在给定的列表中 list 中")
else:
   print ("2 - 变量 b 在给定的列表中 list 中")

# 修改变量 a 的值
a = 2
if ( a in list ):
   print ("3 - 变量 a 在给定的列表中 list 中")
else:
   print ("3 - 变量 a 不在给定的列表中 list 中")
```

以上实例输出结果：

```
1 - 变量 a 不在给定的列表中 list 中
2 - 变量 b 不在给定的列表中 list 中
3 - 变量 a 在给定的列表中 list 中
```

5-7 Python身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	**x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False
is not	is not 是判断两个标识符是不是引用自不同对象	**x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。

注： id()window.location='https://www.runoob.com/python/extend.html') 函数用于获取对象内存地址。

以下实例演示了Python所有身份运算符的操作：

```
#!/usr/bin/python3

a = 20
b = 20

if ( a is b ):
   print ("1 - a 和 b 有相同的标识")
else:
   print ("1 - a 和 b 没有相同的标识")

if ( id(a) == id(b) ):
   print ("2 - a 和 b 有相同的标识")
```

```
else:
   print ("2 - a 和 b 没有相同的标识")

# 修改变量 b 的值
b = 30
if ( a is b ):
   print ("3 - a 和 b 有相同的标识")
else:
   print ("3 - a 和 b 没有相同的标识")

if ( a is not b ):
   print ("4 - a 和 b 没有相同的标识")
else:
   print ("4 - a 和 b 有相同的标识")
```

以上实例输出结果：

```
1 - a 和 b 有相同的标识
2 - a 和 b 有相同的标识
3 - a 和 b 没有相同的标识
4 - a 和 b 没有相同的标识
```

is 与 == 区别：

is 用于判断两个变量引用对象是否为同一个， == 用于判断引用变量的值是否相等。

```
>>>a = [1, 2, 3]
>>> b = a
>>> b is a
True
>>> b == a
True
>>> b = a[:]
>>> b is a
False
>>> b == a
True
```

5-8 Python运算符优先级

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘，除，求余数和取整除
+ -	加法减法
>> <<	右移，左移运算符
&	位 'AND'
^ |	位运算符
<= < > >=	比较运算符
== !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not and or	逻辑运算符

以下实例演示了Python所有运算符优先级的操作：

```
#!/usr/bin/python3

a = 20
b = 10
c = 15
d = 5
e = 0

e = (a + b) * c / d       #( 30 * 15 ) / 5
print ("(a + b) * c / d 运算结果为：",  e)

e = ((a + b) * c) / d     # (30 * 15 ) / 5
print ("((a + b) * c) / d 运算结果为：",  e)

e = (a + b) * (c / d);    # (30) * (15/5)
print ("(a + b) * (c / d) 运算结果为：",  e)

e = a + (b * c) / d;      #  20 + (150/5)
print ("a + (b * c) / d 运算结果为：",  e)
```

以上实例输出结果：

```
(a + b) * c / d 运算结果为：  90.0
((a + b) * c) / d 运算结果为：  90.0
(a + b) * (c / d) 运算结果为：  90.0
a + (b * c) / d 运算结果为：  50.0
```

注意：Pyhton3 已不支持 <> 运算符，可以使用 != 代替，如果你一定要使用这种比较运算符，可以使用以下的方式：

```
>>> from __future__ import barry_as_FLUFL
>>> 1 <> 2
True
```

6， Python3 数字(Number)

Python 数字数据类型用于存储数值。

数据类型是不允许改变的,这就意味着如果改变数字数据类型的值，将重新分配内存空间。

以下实例在变量赋值时 Number 对象将被创建：

```
var1 = 1
var2 = 10
```

您也可以使用del语句删除一些数字对象的引用。

del语句的语法是：

```
del var1[,var2[,var3[....,varN]]]
```

您可以通过使用del语句删除单个或多个对象的引用，例如：

```
del var
del var_a, var_b
```

Python 支持三种不同的数值类型：

- **整型(Int)** - 通常被称为是整型或整数，是正或负整数，不带小数点。Python3 整型是没有限制大小的，可以当作 Long 类型使用，所以 Python3 没有 Python2 的 Long 类型。
- **浮点型(float)** - 浮点型由整数部分与小数部分组成，浮点型也可以使用科学计数法表示（2.5e2 = 2.5 x 102 = 250）
- **复数((complex))** - 复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示， 复数的实部a和虚部b都是浮点型。

我们可以使用十六进制和八进制来代表整数：

```
>>> number = 0xA0F # 十六进制
>>> number
2575

>>> number=0o37 # 八进制
>>> number
31
```

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3e+18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2E-12	4.53e-7j

6-1 Python 数字类型转换

有时候，我们需要对数据内置的类型进行转换，数据类型的转换，你只需要将数据类型作为函数名即可。

- **int(x)** 将x转换为一个整数。
- **float(x)** 将x转换到一个浮点数。
- **complex(x)** 将x转换到一个复数，实数部分为 x，虚数部分为 0。
- **complex(x, y)** 将 x 和 y 转换到一个复数，实数部分为 x，虚数部分为 y。x 和 y 是数字表达式。

以下实例将浮点数变量 a 转换为整数：

```
>>> a = 1.0
>>> int(a)
1
```

6-2 Python 数字运算

Python 解释器可以作为一个简单的计算器，您可以在解释器里输入一个表达式，它将输出表达式的值。

表达式的语法很直白： **+**, **-**, ***** 和 **/**, 和其它语言（如Pascal或C）里一样。例如：

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # 总是返回一个浮点数
1.6
```

注意：在不同的机器上浮点运算的结果可能会不一样。

在整数除法中，除法 **/** 总是返回一个浮点数，如果只想得到整数的结果，丢弃可能的分数部分，可以使用运算符 **//** ：

```
>>> 17 / 3  # 整数除法返回浮点型
5.666666666666667
>>>
>>> 17 // 3  # 整数除法返回向下取整后的结果
5
>>> 17 % 3  # %操作符返回除法的余数
2
>>> 5 * 3 + 2
17
```

注意：//** 得到的并不一定是整数类型的数，它与分母分子的数据类型有关系。

```
>>> 7//2
3
>>> 7.0//2
3.0
>>> 7//2.0
3.0
>>>
```

等号 = 用于给变量赋值。赋值之后，除了下一个提示符，解释器不会显示任何结果。

```
>>> width = 20
>>> height = 5*9
>>> width * height
900
```

Python 可以使用 ** 操作来进行幂运算：

```
>>> 5 ** 2  # 5 的平方
25
>>> 2 ** 7  # 2的7次方
128
```

变量在使用前必须先"定义"（即赋予变量一个值），否则会出现错误：

```
>>> n   # 尝试访问一个未定义的变量
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

不同类型的数混合运算时会将整数转换为浮点数：

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

在交互模式中，最后被输出的表达式结果被赋值给变量 _ 。例如：

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

此处， _ 变量应被用户视为只读变量。

6-3 数学函数

函数	返回值 (描述)
abs(x)	返回数字的绝对值，如abs(-10) 返回 10
ceil(x)	返回数字的上入整数，如math.ceil(4.1) 返回 5
cmp(x, y)	如果 x < y 返回 -1, 如果 x == y 返回 0, 如果 x > y 返回 1。 **Python 3 已废弃，使用 (x>y)-(x<y) 替换。**
exp(x)	返回e的x次幂(e^x),如math.exp(1) 返回2.718281828459045
fabs(x)	返回数字的绝对值，如math.fabs(-10) 返回10.0
floor(x)	返回数字的下舍整数，如math.floor(4.9)返回 4
log(x)	如math.log(math.e)返回1.0,math.log(100,10)返回2.0
log10(x)	返回以10为基数的x的对数，如math.log10(100)返回 2.0
max(x1, x2,...)	返回给定参数的最大值，参数可以为序列。
min(x1, x2,...)	返回给定参数的最小值，参数可以为序列。
modf(x)	返回x的整数部分与小数部分，两部分的数值符号与x相同，整数部分以浮点型表示。
pow(x, y)	x**y 运算后的值。
round(x [,n])	返回浮点数x的四舍五入值，如给出n值，则代表舍入到小数点后的位数。
sqrt(x)	返回数字x的平方根。

6-4 随机数函数

随机数可以用于数学，游戏，安全等领域中，还经常被嵌入到算法中，用以提高算法效率，并提高程序的安全性。

Python包含以下常用随机数函数：

函数	描述
choice(seq)	从序列的元素中随机挑选一个元素，比如random.choice(range(10))，从0到9中随机挑选一个整数。
randrange ([start,] stop [,step])	从指定范围内，按指定基数递增的集合中获取一个随机数，基数默认值为 1
random()	随机生成下一个实数，它在[0,1)范围内。
seed([x])	改变随机数生成器的种子seed。如果你不了解其原理，你不必特别去设定seed，Python会帮你选择seed。
shuffle(lst)	将序列的所有元素随机排序
uniform(x, y)	随机生成下一个实数，它在[x,y]范围内。

6-5 三角函数

Python包括以下三角函数：

函数	描述
acos(x)	返回x的反余弦弧度值。
asin(x)	返回x的反正弦弧度值。
atan(x)	返回x的反正切弧度值。
atan2(y, x)	返回给定的 X 及 Y 坐标值的反正切值。
cos(x)	返回x的弧度的余弦值。
hypot(x, y)	返回欧几里德范数 sqrt(x*x + y*y)。
sin(x)	返回的x弧度的正弦值。
tan(x)	返回x弧度的正切值。
degrees(x)	将弧度转换为角度,如degrees(math.pi/2)， 返回90.0
radians(x)	将角度转换为弧度

6-6 数学常量

常量	描述
pi	数学常量 pi（圆周率，一般以π来表示）
e	数学常量 e，e即自然常数（自然常数）。

7，Python3 字符串

字符串是 Python 中最常用的数据类型。我们可以使用引号(' 或 ")来创建字符串。

创建字符串很简单，只要为变量分配一个值即可。例如：

```
var1 = 'Hello World!'
var2 = "Runoob"
```

7-1 Python 访问字符串中的值

Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。

Python 访问子字符串，可以使用方括号来截取字符串，如下实例：

```
#!/usr/bin/python3

var1 = 'Hello World!'
var2 = "Runoob"

print ("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5])
```

以上实例执行结果：

```
var1[0]:  H
var2[1:5]:  unoo
```

7-2 Python 字符串更新

你可以截取字符串的一部分并与其他字段拼接，如下实例：

```
#!/usr/bin/python3

var1 = 'Hello World!'

print ("已更新字符串 : ", var1[:6] + 'Runoob!')
```

以上实例执行结果

```
已更新字符串 :  Hello Runoob!
```

7-3 Python转义字符

在需要在字符中使用特殊字符时，python用反斜杠()转义字符。如下表：

转义字符	描述
\(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，**yy** 代表的字符，例如：**\o12** 代表换行，其中 o 是字母，不是数字 0。
\xyy	十六进制数，yy代表的字符，例如：\x0a代表换行
\other	其它的字符以普通格式输出

7-4 Python字符串运算符

下表实例变量a值为字符串 "Hello"，b变量值为 "Python"：

操作符	描述	实例
+	字符串连接	a + b 输出结果：HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[]	通过索引获取字符串中字符	a[1] 输出结果 **e**
[:]	截取字符串中的一部分，遵循**左闭右开**原则，str[0,2] 是不包含第 3 个字符的。	a[1:4] 输出结果 **ell**
in	成员运算符 - 如果字符串中包含给定的字符返回 True	**'H' in a** 输出结果 True
not in	成员运算符 - 如果字符串中不包含给定的字符返回 True	**'M' not in a** 输出结果 True
r/R	原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母 `r`（可以大小写）以外，与普通字符串有着几乎完全相同的语法。	`print( r'\n' )` `print( R'\n' )`
%	格式字符串	请看下一节内容。

```
#!/usr/bin/python3

a = "Hello"
b = "Python"

print("a + b 输出结果：", a + b)
print("a * 2 输出结果：", a * 2)
print("a[1] 输出结果：", a[1])
print("a[1:4] 输出结果：", a[1:4])

if( "H" in a) :
    print("H 在变量 a 中")
else :
    print("H 不在变量 a 中")

if( "M" not in a) :
    print("M 不在变量 a 中")
else :
    print("M 在变量 a 中")

print (r'\n')
print (R'\n')
```

以上实例输出结果为：

```
a + b 输出结果： HelloPython
a * 2 输出结果： HelloHello
a[1] 输出结果： e
a[1:4] 输出结果： ell
H 在变量 a 中
M 不在变量 a 中
\n
\n
```

7-5 Python字符串格式化

Python 支持格式化字符串的输出 。尽管这样可能会用到非常复杂的表达式，但最基本的用法是将一个值插入到一个有字符串格式符 %s 的字符串中。

在 Python 中，字符串格式化使用与 C 中 sprintf 函数一样的语法。

```
#!/usr/bin/python3

print ("我叫 %s 今年 %d 岁!" % ('小明', 10))
```

以上实例输出结果：

```
我叫 小明 今年 10 岁!
```

python字符串格式化符号:

符 号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%f 和 %E 的简写
%p	用十六进制数格式化变量的地址

格式化操作符辅助指令:

符号	功能
*	定义宽度或者小数点精度
-	用做左对齐
+	在正数前面显示加号(+)
<sp>	在正数前面显示空格
#	在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话)

7-6 Python三引号

python三引号允许一个字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符。实例如下

```
#!/usr/bin/python3

para_str = """这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [ \n ]。
"""
print (para_str)
```

以上实例执行结果为：

```
这是一个多行字符串的实例
多行字符串可以使用制表符
TAB (    )。
也可以使用换行符 [
 ]。
```

三引号让程序员从引号和特殊字符串的泥潭里面解脱出来，自始至终保持一小块字符串的格式是所谓的WYSIWYG（所见即所得）格式的。

一个典型的用例是，当你需要一块HTML或者SQL时，这时用字符串组合，特殊字符串转义将会非常的繁琐。

```
errHTML = '''
<HTML><HEAD><TITLE>
Friends CGI Demo</TITLE></HEAD>
<BODY><H3>ERROR</H3>
<B>%s</B><P>
<FORM><INPUT TYPE=button VALUE=Back
ONCLICK="window.history.back()"></FORM>
</BODY></HTML>
'''
cursor.execute('''
CREATE TABLE users (
login VARCHAR(8),
uid INTEGER,
prid INTEGER)
''')
```

7-7 f-string

f-string 是 python3.6 之后版本添加的，称之为字面量格式化字符串，是新的格式化字符串的语法。

之前我们习惯用百分号 (%):

```
>>> name = 'Runoob'
>>> 'Hello %s' % name
'Hello Runoob'
```

f-string 格式话字符串以 f 开头，后面跟着字符串，字符串中的表达式用大括号 {} 包起来，它会将变量或表达式计算后的值替换进去，实例如下：

```
>>> name = 'Runoob'
>>> f'Hello {name}'  # 替换变量

>>> f'{1+2}'         # 使用表达式
'3'

>>> w = {'name': 'Runoob', 'url': 'www.runoob.com'}
>>> f'{w["name"]}: {w["url"]}'
'Runoob: www.runoob.com'
```

用了这种方式明显更简单了，不用再去判断使用 %s，还是 %d。

在 Python 3.8 的版本中可以使用 = 符号来拼接运算表达式与结果：

```
>>> x = 1
>>> print(f'{x+1}')   # Python 3.6
2

>>> x = 1
>>> print(f'{x+1=}')   # Python 3.8
'x+1=2'
```

7-8 Unicode 字符串

在Python2中，普通字符串是以8位ASCII码进行存储的，而Unicode字符串则存储为16位unicode字符串，这样能够表示更多的字符集。使用的语法是在字符串前面加上前缀 **u**。

在Python3中，所有的字符串都是Unicode字符串。

7-9 Python 的字符串内建函数

Python 的字符串常用内建函数如下

序号	方法及描述
1	capitalize() 将字符串的第一个字符转换为大写
2	center(width, fillchar) 返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。
3	count(str, beg= 0,end=len(string)) 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
4	bytes.decode(encoding="utf-8", errors="strict") Python3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。
5	encode(encoding='UTF-8',errors='strict') 以 encoding 指定的编码格式编码字符串，如果出错默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'
6	endswith(suffix, beg=0, end=len(string)) 检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.
7	expandtabs(tabsize=8) 把字符串 string 中的 tab 符号转为空格，tab 符号默认的空格数是 8 。
8	find(str, beg=0, end=len(string)) 检测 str 是否包含在字符串中，如果指定范围 beg 和 end ，则检查是否包含在指定范围内，如果包含返回开始的索引值，否则返回-1
9	index(str, beg=0, end=len(string)) 跟find()方法一样，只不过如果str不在字符串中会报一个异常.
10	isalnum() 如果字符串至少有一个字符并且所有字符都是字母或数字则返 回 True,否则返回 False
11	isalpha() 如果字符串至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False
12	isdigit() 如果字符串只包含数字则返回 True 否则返回 False..
13	islower() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
14	isnumeric() 如果字符串中只包含数字字符，则返回 True，否则返回 False
15	isspace() 如果字符串中只包含空白，则返回 True，否则返回 False.
16	istitle() 如果字符串是标题化的(见 title())则返回 True，否则返回 False

17	isupper() 如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
18	join(seq) 以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
19	len(string) 返回字符串长度
20	ljust(width[, fillchar]) 返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格。
21	lower() 转换字符串中所有大写字符为小写.
22	lstrip() 截掉字符串左边的空格或指定字符。
23	maketrans() 创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。
24	max(str) 返回字符串 str 中最大的字母。
25	min(str) 返回字符串 str 中最小的字母。
26	replace(old, new [, max]) 把 将字符串中的 str1 替换成 str2,如果 max 指定，则替换不超过 max 次。
27	rfind(str, beg=0,end=len(string)) 类似于 find()函数，不过是从右边开始查找.
28	rindex(str, beg=0, end=len(string)) 类似于 index()，不过是从右边开始.
29	rjust(width,[, fillchar]) 返回一个原字符串右对齐,并使用fillchar(默认空格) 填充至长度 width 的新字符串
30	rstrip() 删除字符串字符串末尾的空格.
31	split(str="", num=string.count(str)) num=string.count(str)) 以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num+1 个子字符串
32	splitlines([keepends])

	按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。
33	startswith(substr, beg=0,end=len(string)) 检查字符串是否是以指定子字符串 substr 开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查。
34	strip([chars]) 在字符串上执行 lstrip()和 rstrip()
35	swapcase() 将字符串中大写转换为小写，小写转换为大写
36	title() 返回"标题化"的字符串,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())
37	translate(table, deletechars="") 根据 str 给出的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 deletechars 参数中
38	upper() 转换字符串中的小写字母为大写
39	zfill (width) 返回长度为 width 的字符串，原字符串右对齐，前面填充0
40	isdecimal() 检查字符串是否只包含十进制字符，如果是返回 true，否则返回 false。

8，Python3 列表

- 序列是Python中最基本的数据结构。序列中的每个元素都分配一个数字 - 它的位置，或索引，第一个索引是0，第二个索引是1，依此类推。
- Python有6个序列的内置类型，但最常见的是列表和元组。
- 序列都可以进行的操作包括索引，切片，加，乘，检查成员。
- 此外，Python已经内置确定序列的长度以及确定最大和最小的元素的方法。
- 列表是最常用的Python数据类型，它可以作为一个方括号内的逗号分隔值出现。
- 列表的数据项不需要具有相同的类型
- 创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。如下所示：

```
list1 = ['Google', 'Runoob', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

与字符串的索引一样，列表索引从0开始。列表可以进行截取、组合等。

8-1 访问列表中的值

使用下标索引来访问列表中的值，同样你也可以使用方括号的形式截取字符，如下所示：

```
#!/usr/bin/python3

list1 = ['Google', 'Runoob', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];

print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

以上实例输出结果：

```
list1[0]:  Google
list2[1:5]:  [2, 3, 4, 5]
```

8-2 更新列表

你可以对列表的数据项进行修改或更新，你也可以使用append()方法来添加列表项，如下所示：

```
#!/usr/bin/python3

list = ['Google', 'Runoob', 1997, 2000]

print ("第三个元素为 : ", list[2])
list[2] = 2001
print ("更新后的第三个元素为 : ", list[2])
```

注意：我们会在接下来的章节讨论append()方法的使用

以上实例输出结果：

```
第三个元素为 :  1997
更新后的第三个元素为 :  2001
```

8-3 删除列表元素

可以使用 del 语句来删除列表的的元素，如下实例：

```
#!/usr/bin/python3

list = ['Google', 'Runoob', 1997, 2000]

print ("原始列表 : ", list)
del list[2]
print ("删除第三个元素 : ", list)
```

以上实例输出结果：

```
原始列表 :  ['Google', 'Runoob', 1997, 2000]
删除第三个元素 :  ['Google', 'Runoob', 2000]
```

注意：我们会在接下来的章节讨论 remove() 方法的使用

8-4 Python列表脚本操作符

列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。

如下所示：

Python 表达式	结果	描述
len([1, 2, 3])	3	长度
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	组合
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复
3 in [1, 2, 3]	True	元素是否存在于列表中
for x in [1, 2, 3]: print(x, end=" ")	1 2 3	迭代

8-5 Python列表截取与拼接

Python的列表截取与字符串操作类型，如下所示：

```
L=['Google', 'Runoob', 'Taobao']
```

Python 表达式	结果	描述
L[2]	'Taobao'	读取第三个元素
L[-2]	'Runoob'	从右侧开始读取倒数第二个元素: count from the right
L[1:]	['Runoob', 'Taobao']	输出从第二个元素开始后的所有元素

```
>>>L=['Google', 'Runoob', 'Taobao']
>>> L[2]
'Taobao'
>>> L[-2]
'Runoob'
>>> L[1:]
['Runoob', 'Taobao']
>>>
```

列表还支持拼接操作：

```
>>>squares = [1, 4, 9, 16, 25]
>>> squares += [36, 49, 64, 81, 100]
>>> squares
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

8-6 嵌套列表

使用嵌套列表即在列表里创建其它列表，例如：

```
>>>a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

8-7 Python列表函数&方法

Python包含以下函数:

序号	函数
1	len(list) 列表元素个数
2	max(list) 返回列表元素最大值
3	min(list) 返回列表元素最小值
4	list(seq) 将元组转换为列表

Python包含以下方法:

序号	方法
1	list.append(obj) 在列表末尾添加新的对象
2	list.count(obj) 统计某个元素在列表中出现的次数
3	list.extend(seq) 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
4	list.index(obj) 从列表中找出某个值第一个匹配项的索引位置
5	list.insert(index, obj) 将对象插入列表
6	list.pop([index=-1]) 移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹配项
8	list.reverse() 反向列表中元素
9	list.sort(key=None, reverse=False) 对原列表进行排序
10	list.clear() 清空列表
11	list.copy() 复制列表

9，Python3 元组

- Python 的元组与列表类似，不同之处在于元组的元素不能修改。
- 元组使用小括号，列表使用方括号。
- 元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。

```
>>>tup1 = ('Google', 'Runoob', 1997, 2000);
>>> tup2 = (1, 2, 3, 4, 5 );
>>> tup3 = "a", "b", "c", "d";   #  不需要括号也可以
>>> type(tup3)
<class 'tuple'>
```

创建空元组

```
tup1 = ();
```

元组中只包含一个元素时，需要在元素后面添加逗号，否则括号会被当作运算符使用：

```
>>>tup1 = (50)
>>> type(tup1)     # 不加逗号，类型为整型
<class 'int'>

>>> tup1 = (50,)
>>> type(tup1)     # 加上逗号，类型为元组
<class 'tuple'>
```

元组与字符串类似，下标索引从0开始，可以进行截取，组合等。

9-1 访问元组

元组可以使用下标索引来访问元组中的值，如下实例:

```
#!/usr/bin/python3

tup1 = ('Google', 'Runoob', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )

print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

以上实例输出结果：

```
tup1[0]:  Google
tup2[1:5]:  (2, 3, 4, 5)
```

9-2 修改元组

元组中的元素值是不允许修改的，但我们可以对元组进行连接组合，如下实例:

```
#!/usr/bin/python3

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz')

# 以下修改元组元素操作是非法的。
# tup1[0] = 100

# 创建一个新的元组
tup3 = tup1 + tup2;
print (tup3)
```

以上实例输出结果：

```
(12, 34.56, 'abc', 'xyz')
```

9-3 删除元组

元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组，如下实例:

```
#!/usr/bin/python3

tup = ('Google', 'Runoob', 1997, 2000)

print (tup)
del tup;
print ("删除后的元组 tup : ")
print (tup)
```

以上实例元组被删除后，输出变量会有异常信息，输出如下所示：

```
删除后的元组 tup :
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    print (tup)
NameError: name 'tup' is not defined
```

9-4 元组运算符

与字符串一样，元组之间可以使用 + 号和 * 号进行运算。这就意味着他们可以组合和复制，运算后会生成一个新的元组。

Python 表达式	结果	描述
len((1, 2, 3))	3	计算元素个数
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	连接
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	复制
3 in (1, 2, 3)	True	元素是否存在
for x in (1, 2, 3): print (x,)	1 2 3	迭代

9-5 元组索引，截取

因为元组也是一个序列，所以我们可以访问元组中的指定位置的元素，也可以截取索引中的一段元素，如下所示：

元组：

```
L = ('Google', 'Taobao', 'Runoob')
```

Python 表达式	结果	描述
L[2]	'Runoob'	读取第三个元素
L[-2]	'Taobao'	反向读取；读取倒数第二个元素
L[1:]	('Taobao', 'Runoob')	截取元素，从第二个开始后的所有元素。

运行实例如下：

```
>>> L = ('Google', 'Taobao', 'Runoob')
>>> L[2]
'Runoob'
>>> L[-2]
'Taobao'
>>> L[1:]
('Taobao', 'Runoob')
```

9-6 元组内置函数

Python元组包含了以下内置函数

序号	方法及描述	实例

1	len(tuple)
	计算元组元素个数。
		`>>> tuple1 = ('Google', 'Runoob', 'Taobao')`
		`>>> len(tuple1)`
		`3`
		`>>>`

2	max(tuple)
	返回元组中元素最大值。
		`>>> tuple2 = ('5', '4', '8')`
		`>>> max(tuple2)`
		`'8'`
		`>>>`

3	min(tuple)
	返回元组中元素最小值。
		`>>> tuple2 = ('5', '4', '8')`
		`>>> min(tuple2)`
		`'4'`
		`>>>`

4	tuple(seq)
	将列表转换为元组。
		`>>> list1= ['Google', 'Taobao', 'Runoob', 'Baidu']`
		`>>> tuple1=tuple(list1)`
		`>>> tuple1`
		`('Google', 'Taobao', 'Runoob', 'Baidu')`

9-7 关于元组是不可变的

所谓元组的不可变指的是元组所指向的内存中的内同不可变。

```
>>> tup = ('r', 'u', 'n', 'o', 'o', 'b')
>>> tup[0] = 'g'     # 不支持修改元素
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> id(tup)     # 查看内存地址
4440687904
>>> tup = (1,2,3)
>>> id(tup)
4441088800    # 内存地址不一样了
```

从以上实例可以看出，重新赋值的元组 tup，绑定到新的对象了，不是修改了原来的对象。

10，Python3 字典

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值(key=>value)对用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号({})中，格式如下所示：

```
d = {key1 : value1, key2 : value2 }
```

键必须是唯一的，但值则不必。

值可以取任何数据类型，但键必须是不可变的，如字符串，数字或元组。

一个简单的字典实例：

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

也可如此创建字典：

```
dict1 = { 'abc': 456 }
dict2 = { 'abc': 123, 98.6: 37 }
```

10-1 访问字典里的值

把相应的键放入到方括号中，如下实例:

```
#!/usr/bin/python3

dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])
```

以上实例输出结果：

```
dict['Name']:  Runoob
dict['Age']:  7
```

如果用字典里没有的键访问数据，会输出错误如下：

```
#!/usr/bin/python3

dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

print ("dict['Alice']: ", dict['Alice'])
```

以上实例输出结果：

```
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    print ("dict['Alice']: ", dict['Alice'])
KeyError: 'Alice'
```

10-2 修改字典

向字典添加新内容的方法是增加新的键/值对，修改或删除已有键/值对如下实例:

```
#!/usr/bin/python3

dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8               # 更新 Age
dict['School'] = "菜鸟教程"  # 添加信息


print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

以上实例输出结果：

```
dict['Age']:  8
dict['School']:  菜鸟教程
```

10-3 删除字典元素

能删单一的元素也能清空字典，清空只需一项操作。

显示删除一个字典用del命令，如下实例：

```
#!/usr/bin/python3

dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

del dict['Name'] # 删除键 'Name'
dict.clear()     # 清空字典
del dict         # 删除字典

print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

但这会引发一个异常，因为用执行 del 操作后字典不再存在：

```
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print ("dict['Age']: ", dict['Age'])
TypeError: 'type' object is not subscriptable
```

注：del() 方法后面也会讨论。

10-4 字典键的特性

字典值可以是任何的 python 对象，既可以是标准的对象，也可以是用户定义的，但键不行。

两个重要的点需要记住：

1）不允许同一个键出现两次。创建时如果同一个键被赋值两次，后一个值会被记住，如下实例：

```
#!/usr/bin/python3

dict = {'Name': 'Runoob', 'Age': 7, 'Name': '小菜鸟'}

print ("dict['Name']: ", dict['Name'])
```

以上实例输出结果：

```
dict['Name']:  小菜鸟
```

2）键必须不可变，所以可以用数字，字符串或元组充当，而用列表就不行，如下实例：

```
#!/usr/bin/python3

dict = {['Name']: 'Runoob', 'Age': 7}

print ("dict['Name']: ", dict['Name'])
```

以上实例输出结果：

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Runoob', 'Age': 7}
TypeError: unhashable type: 'list'
```

10-5 字典内置函数&方法

Python字典包含了以下内置函数：

序号	函数及描述	实例

1	len(dict)
	计算字典元素个数，即键的总数。
		`>>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}`
		`>>> len(dict)`
		`3`

2	str(dict)
	输出字典，以可打印的字符串表示。
		`>>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}`
		`>>> str(dict)`
		`"{'Name': 'Runoob', 'Class': 'First', 'Age': 7}"`

3	type(variable)
	返回输入的变量类型，如果变量是字典就返回字典类型。
		`>>> dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}`
		`>>> type(dict)`
		`<class 'dict'>`

Python字典包含了以下内置方法：

序号	函数及描述
1	radiansdict.clear() 删除字典内所有元素
2	radiansdict.copy() 返回一个字典的浅复制
3	radiansdict.fromkeys() 创建一个新字典，以序列seq中元素做字典的键，val为字典所有键对应的初始值
4	radiansdict.get(key, default=None) 返回指定键的值，如果值不在字典中返回default值
5	key in dict 如果键在字典dict里返回true，否则返回false
6	radiansdict.items() 以列表返回可遍历的(键, 值) 元组数组
7	radiansdict.keys() 返回一个迭代器，可以使用 list() 来转换为列表
8	radiansdict.setdefault(key, default=None) 和get()类似, 但如果键不存在于字典中，将会添加键并将值设为default
9	radiansdict.update(dict2) 把字典dict2的键/值对更新到dict里
10	radiansdict.values() 返回一个迭代器，可以使用 list() 来转换为列表
11	pop(key[,default]) 删除字典给定键 key 所对应的值，返回值为被删除的值。key值必须给出。 否则，返回default值。
12	popitem() 随机返回并删除字典中的最后一对键和值。

11，Python3 集合

集合（set）是一个无序的不重复元素序列。

可以使用大括号 **{ }** 或者 **set()** 函数创建集合，注意：创建一个空集合必须用 **set()** 而不是 **{ }**，因为 **{ }** 是用来创建一个空字典。

创建格式：

```
parame = {value01,value02,...}
或者
set(value)
```

```
>>>basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # 这里演示的是去重功能
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # 快速判断元素是否在集合内
True
>>> 'crabgrass' in basket
False

>>> # 下面展示两个集合间的运算.
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # 集合a中包含而集合b中不包含的元素
{'r', 'd', 'b'}
>>> a | b                              # 集合a或b中包含的所有元素
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # 集合a和b中都包含了的元素
{'a', 'c'}
>>> a ^ b                              # 不同时包含于a和b的元素
{'r', 'd', 'b', 'm', 'z', 'l'}
```

类似列表推导式，同样集合支持集合推导式(Set comprehension):

```
>>>a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

11-1 集合的基本操作

11-1-1 添加元素

语法格式如下：

```
s.add( x )
```

将元素 x 添加到集合 s 中，如果元素已存在，则不进行任何操作。

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.add("Facebook")
>>> print(thisset)
{'Taobao', 'Facebook', 'Google', 'Runoob'}
```

还有一个方法，也可以添加元素，且参数可以是列表，元组，字典等，语法格式如下：

```
s.update( x )
```

x 可以有多个，用逗号分开。

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.update({1,3})
>>> print(thisset)
{1, 3, 'Google', 'Taobao', 'Runoob'}
>>> thisset.update([1,4],[5,6])
>>> print(thisset)
{1, 3, 4, 5, 6, 'Google', 'Taobao', 'Runoob'}
>>>
```

11-1-2 移除元素

语法格式如下：

```
s.remove( x )
```

将元素 x 从集合 s 中移除，如果元素不存在，则会发生错误。

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.remove("Taobao")
>>> print(thisset)
{'Google', 'Runoob'}
>>> thisset.remove("Facebook")   # 不存在会发生错误
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Facebook'
>>>
```

此外还有一个方法也是移除集合中的元素，且如果元素不存在，不会发生错误。格式如下所示：

```
s.discard( x )
```

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.discard("Facebook")  # 不存在不会发生错误
>>> print(thisset)
{'Taobao', 'Google', 'Runoob'}
```

我们也可以设置随机删除集合中的一个元素，语法格式如下：

```
s.pop()
```

```
thisset = set(("Google", "Runoob", "Taobao", "Facebook"))
x = thisset.pop()

print(x)
```

输出结果：

```
$ python3 test.py
Runoob
```

多次执行测试结果都不一样。

set 集合的 pop 方法会对集合进行无序的排列，然后将这个无序排列集合的左面第一个元素进行删除。

11-1-3 计算集合元素个数

语法格式如下：

```
len(s)
```

计算集合 s 元素个数。

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> len(thisset)
3
```

11-1-4 清空集合

语法格式如下：

```
s.clear()
```

清空集合 s。

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.clear()
>>> print(thisset)
set()
```

11-1-5 判断元素是否在集合中存在

语法格式如下：

```
x in s
```

判断元素 x 是否在集合 s 中，存在返回 True，不存在返回 False。

```
>>>thisset = set(("Google", "Runoob", "Taobao"))
>>> "Runoob" in thisset
True
>>> "Facebook" in thisset
False
>>>
```

11-2 集合内置方法完整列表

方法	描述
add()	为集合添加元素
clear()	移除集合中的所有元素
copy()	拷贝一个集合
difference()	返回多个集合的差集
difference_update()	移除集合中的元素，该元素在指定的集合也存在。
discard()	删除集合中指定的元素
intersection()	返回集合的交集
intersection_update()	返回集合的交集。
isdisjoint()	判断两个集合是否包含相同的元素，如果没有返回 True，否则返回 False。
issubset()	判断指定集合是否为该方法参数集合的子集。
issuperset()	判断该方法的参数集合是否为指定集合的子集
pop()	随机移除元素
remove()	移除指定元素
symmetric_difference()	返回两个集合中不重复的元素集合。
symmetric_difference_update()	移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。
union()	返回两个集合的并集
update()	给集合添加元素

12，Python3 编程第一步

在前面的教程中我们已经学习了一些 Python3 的基本语法知识，下面我们尝试来写一个斐波纳契数列。

```
#!/usr/bin/python3

# Fibonacci series: 斐波纳契数列
# 两个元素的总和确定了下一个数
a, b = 0, 1
while b < 10:
    print(b)
    a, b = b, a+b
```

其中代码 **a, b = b, a+b** 的计算方式为先计算右边表达式，然后同时赋值给左边，等价于：

```
n=b
m=a+b
a=n
b=m
```

执行以上程序，输出结果为：

```
1
1
2
3
5
8
```

这个例子介绍了几个新特征。

第一行包含了一个复合赋值：变量 a 和 b 同时得到新值 0 和 1。最后一行再次使用了同样的方法，可以看到，右边的表达式会在赋值变动之前执行。右边表达式的执行顺序是从左往右的。

输出变量值:

```
>>> i = 256*256
>>> print('i 的值为: ', i)
i 的值为:  65536
```

end 关键字

关键字end可以用于将结果输出到同一行，或者在输出的末尾添加不同的字符，实例如下：

```
#!/usr/bin/python3

# Fibonacci series: 斐波纳契数列
# 两个元素的总和确定了下一个数
a, b = 0, 1
while b < 1000:
    print(b, end=',')
    a, b = b, a+b
```

执行以上程序，输出结果为：

```
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

13，Python3 条件控制

13-1 if 语句

Python中if语句的一般形式如下所示：

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

- 如果 "condition_1" 为 True 将执行 "statement_block_1" 块语句
- 如果 "condition_1" 为False，将判断 "condition_2"
- 如果"condition_2" 为 True 将执行 "statement_block_2" 块语句
- 如果 "condition_2" 为False，将执行"statement_block_3"块语句

Python 中用 **elif** 代替了 **else if**，所以if语句的关键字为：**if – elif – else**。

注意：

- 1、每个条件后面要使用冒号 :，表示接下来是满足条件后要执行的语句块。
- 2、使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。
- 3、在Python中没有switch – case语句。

以下是一个简单的 if 实例：

```
#!/usr/bin/python3

var1 = 100
if var1:
    print ("1 - if 表达式条件为 true")
    print (var1)

var2 = 0
if var2:
    print ("2 - if 表达式条件为 true")
    print (var2)
print ("Good bye!")
```

执行以上代码，输出结果为：

```
1 - if 表达式条件为 true
100
Good bye!
```

从结果可以看到由于变量 var2 为 0，所以对应的 if 内的语句没有执行。

以下实例演示了狗的年龄计算判断：

```
#!/usr/bin/python3

age = int(input("请输入你家狗狗的年龄: "))
print("")
if age <= 0:
    print("你是在逗我吧!")
elif age == 1:
    print("相当于 14 岁的人。")
elif age == 2:
    print("相当于 22 岁的人。")
elif age > 2:
    human = 22 + (age -2)*5
    print("对应人类年龄: ", human)

### 退出提示
input("点击 enter 键退出")
```

将以上脚本保存在dog.py文件中，并执行该脚本：

```
$ python3 dog.py
请输入你家狗狗的年龄: 1

相当于 14 岁的人。
点击 enter 键退出
```

以下为if中常用的操作运算符:

操作符	描述
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于，比较两个值是否相等
!=	不等于

```
#!/usr/bin/python3

# 程序演示了 == 操作符
# 使用数字
print(5 == 6)
# 使用变量
x = 5
y = 8
print(x == y)
```

以上实例输出结果：

```
False
False
```

high_low.py文件演示了数字的比较运算：

```
#!/usr/bin/python3

# 该实例演示了数字猜谜游戏
number = 7
guess = -1
print("数字猜谜游戏!")
while guess != number:
    guess = int(input("请输入你猜的数字："))

    if guess == number:
        print("恭喜，你猜对了！")
    elif guess < number:
        print("猜的数字小了...")
    elif guess > number:
        print("猜的数字大了...")
```

执行以上脚本，实例输出结果如下：

```
$ python3 high_low.py
数字猜谜游戏!
请输入你猜的数字：1
猜的数字小了...
请输入你猜的数字：9
猜的数字大了...
请输入你猜的数字：7
恭喜，你猜对了!
```

13-2 if 嵌套

在嵌套 if 语句中，可以把 if...elif...else 结构放在另外一个 if...elif...else 结构中。

```
if 表达式1:
    语句
    if 表达式2:
        语句
    elif 表达式3:
        语句
    else:
        语句
elif 表达式4:
    语句
else:
    语句
```

```
# !/usr/bin/python3

num=int(input("输入一个数字："))
if num%2==0:
    if num%3==0:
        print ("你输入的数字可以整除 2 和 3")
    else:
        print ("你输入的数字可以整除 2，但不能整除 3")
else:
    if num%3==0:
        print ("你输入的数字可以整除 3，但不能整除 2")
    else:
        print  ("你输入的数字不能整除 2 和 3")
```

将以上程序保存到 test_if.py 文件中，执行后输出结果为：

```
$ python3 test.py
输入一个数字：6
你输入的数字可以整除 2 和 3
```

14，Python3 循环语句

14-1 while 循环

Python 中 while 语句的一般形式：

```
while 判断条件(condition)：
    执行语句(statements)……
```

同样需要注意冒号和缩进。另外，在 Python 中没有 do..while 循环。

以下实例使用了 while 来计算 1 到 100 的总和：

```
#!/usr/bin/env python3

n = 100

sum = 0
counter = 1
while counter <= n:
    sum = sum + counter
    counter += 1

print("1 到 %d 之和为: %d" % (n,sum))
```

执行结果如下：

```
1 到 100 之和为: 5050
```

14-2 无限循环

我们可以通过设置条件表达式永远不为 false 来实现无限循环，实例如下

```
#!/usr/bin/python3

var = 1
while var == 1 :  # 表达式永远为 true
   num = int(input("输入一个数字  :"))
   print ("你输入的数字是: ", num)

print ("Good bye!")
```

执行以上脚本，输出结果如下：

```
输入一个数字  :5
你输入的数字是:  5
输入一个数字  :
```

你可以使用 **CTRL+C** 来退出当前的无限循环。

无限循环在服务器上客户端的实时请求非常有用。

14-3 while 循环使用 else 语句

在 while ... else 在条件语句为 false 时执行 else 的语句块。

语法格式如下：

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

循环输出数字，并判断大小：

```
#!/usr/bin/python3

count = 0
while count < 5:
   print (count, " 小于 5")
   count = count + 1
else:
   print (count, " 大于或等于 5")
```

执行以上脚本，输出结果如下：

```
0  小于 5
1  小于 5
2  小于 5
3  小于 5
4  小于 5
5  大于或等于 5
```

14-4 简单语句组

类似if语句的语法，如果你的while循环体中只有一条语句，你可以将该语句与while写在同一行中， 如下所示：

```
#!/usr/bin/python

flag = 1

while (flag): print ('欢迎访问菜鸟教程!')

print ("Good bye!")
```

注意：以上的无限循环你可以使用 CTRL+C 来中断循环。

执行以上脚本，输出结果如下：

```
欢迎访问菜鸟教程!
欢迎访问菜鸟教程!
欢迎访问菜鸟教程!
欢迎访问菜鸟教程!
欢迎访问菜鸟教程!
.....
```

14-5 for 语句

Python for循环可以遍历任何序列的项目，如一个列表或者一个字符串。

for循环的一般格式如下：

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>
```

```
>>>languages = ["C", "C++", "Perl", "Python"]
>>> for x in languages:
...     print (x)
...
C
C++
Perl
Python
>>>
```

以下 for 实例中使用了 break 语句，break 语句用于跳出当前循环体：

```
#!/usr/bin/python3

sites = ["Baidu", "Google","Runoob","Taobao"]
for site in sites:
    if site == "Runoob":
        print("菜鸟教程!")
        break
    print("循环数据 " + site)
else:
    print("没有循环数据!")
print("完成循环!")
```

执行脚本后，在循环到 "Runoob"时会跳出循环体：

```
循环数据 Baidu
循环数据 Google
菜鸟教程!
完成循环!
```

14-6 range()函数

如果你需要遍历数字序列，可以使用内置range()函数。它会生成数列，例如:

```
>>>for i in range(5):
...     print(i)
...
0
1
2
3
4
```

你也可以使用range指定区间的值：

```
>>>for i in range(5,9) :
    print(i)


5
6
7
8
>>>
```

也可以使range以指定数字开始并指定不同的增量(甚至可以是负数，有时这也叫做'步长')：

```
>>>for i in range(0, 10, 3) :
    print(i)


0
3
6
9
>>>
```

负数：

```
>>>for i in range(-10, -100, -30) :
    print(i)


-10
-40
-70
>>>
```

您可以结合range()和len()函数以遍历一个序列的索引,如下所示:

```
>>>a = ['Google', 'Baidu', 'Runoob', 'Taobao', 'QQ']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Google
1 Baidu
2 Runoob
3 Taobao
4 QQ
>>>
```

还可以使用range()函数来创建一个列表：

```
>>>list(range(5))
[0, 1, 2, 3, 4]
>>>
```

14-7 break 和 continue 语句及循环中的 else 子句

- **break** 语句可以跳出 for 和 while 的循环体。如果你从 for 或 while 循环中终止，任何对应的循环 else 块将不执行。
- **continue** 语句被用来告诉 Python 跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

while 中使用 break：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('循环结束。')
```

输出结果为：

```
4
3
循环结束。
```

while 中使用 continue：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        continue
    print(n)
print('循环结束。')
```

输出结果为：

```
4
3
1
0
循环结束。
```

更多实例如下：

```
#!/usr/bin/python3

for letter in 'Runoob':     # 第一个实例
   if letter == 'b':
      break
   print ('当前字母为 :', letter)

var = 10                    # 第二个实例
while var > 0:
   print ('当期变量值为 :', var)
   var = var -1
   if var == 5:
```

```
      break

print ("Good bye!")
```

执行以上脚本输出结果为：

```
当前字母为 : R
当前字母为 : u
当前字母为 : n
当前字母为 : o
当前字母为 : o
当期变量值为 : 10
当期变量值为 : 9
当期变量值为 : 8
当期变量值为 : 7
当期变量值为 : 6
Good bye!
```

以下实例循环字符串 Runoob，碰到字母 o 跳过输出：

```
#!/usr/bin/python3

for letter in 'Runoob':     # 第一个实例
   if letter == 'o':        # 字母为 o 时跳过输出
      continue
   print ('当前字母 :', letter)

var = 10                    # 第二个实例
while var > 0:
   var = var -1
   if var == 5:             # 变量为 5 时跳过输出
      continue
   print ('当前变量值 :', var)
print ("Good bye!")
```

执行以上脚本输出结果为：

```
当前字母 : R
当前字母 : u
当前字母 : n
当前字母 : b
当前变量值 : 9
当前变量值 : 8
当前变量值 : 7
当前变量值 : 6
当前变量值 : 4
当前变量值 : 3
当前变量值 : 2
当前变量值 : 1
当前变量值 : 0
Good bye!
```

循环语句可以有 else 子句，它在穷尽列表(以for循环)或条件变为 false (以while循环)导致循环终止时被执行，但循环被 break 终止时不执行。

如下实例用于查询质数的循环例子:

```
#!/usr/bin/python3

for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, '等于', x, '*', n//x)
            break
    else:
        # 循环中没有找到元素
        print(n, ' 是质数')
```

执行以上脚本输出结果为:

```
2  是质数
3  是质数
4 等于 2 * 2
5  是质数
6 等于 2 * 3
7  是质数
8 等于 2 * 4
9 等于 3 * 3
```

14-8 pass 语句

Python pass是空语句，是为了保持程序结构的完整性。

pass 不做任何事情，一般用做占位语句，如下实例

```
>>>while True:
...     pass  # 等待键盘中断 (Ctrl+C)
```

最小的类:

```
>>>class MyEmptyClass:
...     pass
```

以下实例在字母为 o 时 执行 pass 语句块:

```
#!/usr/bin/python3

for letter in 'Runoob':
   if letter == 'o':
      pass
      print ('执行 pass 块')
   print ('当前字母 :', letter)

print ("Good bye!")
```

执行以上脚本输出结果为:

```
当前字母 : R
当前字母 : u
当前字母 : n
执行 pass 块
当前字母 : o
执行 pass 块
当前字母 : o
当前字母 : b
Good bye!
```

15，Python3 迭代器与生成器

15-1 迭代器

15-1-1 基础知识

1，迭代器：迭代取值的工具，迭代是重复的过程，每一次重复都是基于上次的结果而继续的，单纯的重复不是迭代

```
# 可迭代对象： 但凡内置有__iter__()方法的都称之为可迭代对象
# 字符串---列表---元祖---字典---集合---文件操作  都是可迭代对象

# 调用可迭代对象下的__iter__方法将其转换为可迭代对象
d = {'a':1, 'b':2, 'c':3}

d_iter = d.__iter__() # 把字典d转换成了可迭代对象

#   d_iter.__next__()      # 通过__next__()方法可以取值

print(d_iter.__next__()) # a
print(d_iter.__next__()) # b
print(d_iter.__next__()) # c

# 没值了以后就会报错， 抛出异常StopIteration
#-------------------------------------------------
d = {'a':1, 'b':2, 'c':3}
d_iter = d.__iter__()
while True:
    try:
        print(d_iter.__next__())
    except StopIteration:
        break
# 对同一个迭代器对象，取值取干净的情况下第二次取值的时候去不了，没值，只能造新的迭代器
```

15-1-2 迭代器与for循环工作原理

```
#可迭代对象与迭代器详解
    #可迭代对象：内置有__iter__() 方法对象
        # 可迭代对象.__iter__(): 得到可迭代对象

    #迭代器对象：内置有__next__() 方法
        # 迭代器对象.__next__()：得到迭代器的下一个值
        # 迭代器对象.__iter__(): 得到的值迭代器对象的本身（调跟没调一个样）-----------> 为了保证for循环的工作
```

```
# for循环工作原理
    d = {'a':1, 'b':2, 'c':3}
    d_iter = d.__iter__()

    # 1，d.__iter__() 方法得到一个跌倒器对象
    # 2,迭代器对象的__next__()方法拿到返回值，将该返回值赋值给k
    # 3,循环往复步骤2，直到抛出异常，for循环会捕捉异常并结束循环

    for k in d:
        print(k)

    # 可迭代器对象不一定是迭代器对象----------迭代器对象一定是可迭代对象
    # 字符串---列表---元祖---字典---集合只是可迭代对象，不是迭代器对象、
    # 文件操作时迭代器对象也是可迭代对象
```

15-2 生成器（本质就是迭代器）

```
# 函数里包含yield,并且调用函数以后就能得到一个可迭代对象
def test():
    print('第一次')
    yield 1
    print('第二次')
    yield 2
    print('第三次')
    yield 3
    print('第四次')

g = test()
print(g) # <generator object test at 0x0000014C809A27A0>
g_iter = g.__iter__()
res1 = g_iter.__next__() # 第一次
print(res1) # 1
res2 = g_iter.__next__() # 第二次
print(res2) # 2
res3 = g_iter.__next__() # 第三次
print(res3) # 3

# 补充
len(s) -------> s.__len__()
next(s) ------> s.__next__()
iter(d) -------> d.__iter__()
```

15-2-1 yield 表达式

```
def person(name):
    print("%s吃东西啦！！"%name)
    while True:
        x = yield None
        print('%s吃东西啦---%s'%(name,x))


g = person('aini')
# next(g) ================ g.send(None)
next(g)
next(g)
# send()方法可以给yield传值
```

```
# 不能在第一次运行时用g.send()来传值，需要用g.send(None)或者next(g) 来初始化，第二次开始可以用g.send("值")来传值
g.send("雪糕")  # aini吃东西啦---雪糕
g.send("西瓜")  # aini吃东西啦---西瓜
```

15-2-2 三元表达式

```
x = 10
y = 20
res = x if x > y else y
# 格式
条件成立时返回的值 if 条件 else 条件不成立时返回的值
```

15-2-3 列表生成式

```
l = ['aini_aaa','dilnur_aaa','donghua_aaa','egon']
res = [name for name in l if name.endswith('aaa')]
print(res)

# 语法：  [结果 for 元素 in 可迭代对象 if 条件]


l = ['aini_aaa','dilnur_aaa','donghua_aaa','egon']
l = [name.upper() for name in l]
print(l)

l = ['aini_aaa','dilnur_aaa','donghua_aaa','egon']
l = [name.replace('_aaa','') for name in l if name.endswith('_aaa')]
print(l)
```

15-2-4 其他生成器（——没有元祖生成式——）

```
### 字典生成器
keys = ['name','age','gender']
res = {key: None for key in keys}
print(res)  # {'name': None, 'age': None, 'gender': None}

items = [('name','aini'),('age',22),('gender','man')]
res = {k:v for k,v in items}
print(res)

## 集合生成器
keys = ['name','age','gender']
set1 = {key for key in keys}

## 没有元祖生成器
g = (i for i in range(10) if i % 4 == 0 )  ## 得到的是一个迭代器


#### 统计文件字符个数
with open('aini.txt', mode='rt', encoding= 'utf-8') as f:
    res = sum(len(line) for line in f)
    print(res)
```

15-2-5 二分法

```
l = [-10,-6,-3,0,1,10,56,134,222,234,532,642,743,852,1431]

def search_num(num,list):
    mid_index = len(list) // 2
    if len(list) == 0:
        print("没找到")
        return False
    if num > list[mid_index]:
        list = list[mid_index + 1 :]
        search_num(num,list)
    elif num < list[mid_index]:
        list = list[:mid_index]
        search_num(num, list)
    else:
        print('找到了' , list[mid_index])

search_num(743,l)
```

15-2-6 匿名函数与lambdaj

```
## 定义
res = lambda x,y : x+y
## 调用
(lambda x,y : x+y)(10,20)  # 第一种方法
res(10,20)    ## 第二种方法

##应用场景
salary = {
    'aini':20000,
    'aili':50000,
    'dilnur':15000,
    'hahhaha':42568,
    'fdafdaf':7854
}

res = max(salary ,key= lambda x : salary[x])
print(res)
```

16，Python3 函数

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。你已经知道Python提供了许多内建函数，比如print()。但你也可以自己创建函数，这被叫做用户自定义函数。

16-1 定义一个函数

你可以定义一个由自己想要功能的函数，以下是简单的规则：

- 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 **()**。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
- 函数内容以冒号起始，并且缩进。

- **return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回None。

语法

Python 定义函数使用 def 关键字，一般格式如下：

```
def 函数名（参数列表）:
    函数体
```

默认情况下，参数值和参数名称是按函数声明中定义的顺序匹配起来的。

让我们使用函数来输出"Hello World！"：

```
>>>def hello() :
   print("Hello World!")


>>> hello()
Hello World!
>>>
```

更复杂点的应用，函数中带上参数变量:

```
#!/usr/bin/python3

# 计算面积函数
def area(width, height):
    return width * height

def print_welcome(name):
    print("Welcome", name)

print_welcome("Runoob")
w = 4
h = 5
print("width =", w, " height =", h, " area =", area(w, h))
```

以上实例输出结果：

```
Welcome Runoob
width = 4  height = 5  area = 20
```

16-2 函数调用

定义一个函数：给了函数一个名称，指定了函数里包含的参数，和代码块结构。

这个函数的基本结构完成以后，你可以通过另一个函数调用执行，也可以直接从 Python 命令提示符执行。

如下实例调用了 **printme()** 函数：

```
#!/usr/bin/python3

# 定义函数
def printme( str ):
   # 打印任何传入的字符串
   print (str)
   return

# 调用函数
printme("我要调用用户自定义函数!")
printme("再次调用同一函数")
```

以上实例输出结果：

```
我要调用用户自定义函数!
再次调用同一函数
```

16-3 参数传递

在 python 中，类型属于对象，变量是没有类型的：

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

a="Runoob"
```

以上代码中，**[1,2,3]** 是 List 类型，**"Runoob"** 是 String 类型，而变量 a 是没有类型，她仅仅是一个对象的引用（一个指针），可以是指向 List 类型对象，也可以是指向 String 类型对象。

16-4 可更改(mutable)与不可更改(immutable)对象

在 python 中，strings, tuples, 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

- **不可变类型：**变量赋值 **a=5** 后再赋值 **a=10**，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变a的值，相当于新生成了a。
- **可变类型：**变量赋值 **la=[1,2,3,4]** 后再赋值 **la[2]=5** 则是将 list la 的第三个元素值更改，本身la没有动，只是其内部的一部分值被修改了。

python 函数的参数传递：

- **不可变类型：**类似 c++ 的值传递，如 整数、字符串、元组。如fun（a），传递的只是a的值，没有影响a对象本身。比如在 fun（a）内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。
- **可变类型：**类似 c++ 的引用传递，如 列表，字典。如 fun（la），则是将 la 真正的传过去，修改后fun外部的la也会受影响

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说传不可变对象和传可变对象。

16-5 python 传不可变对象实例

```
#!/usr/bin/python3

def ChangeInt( a ):
    a = 10

b = 2
ChangeInt(b)
print( b ) # 结果是 2
```

实例中有 int 对象 2，指向它的变量是 b，在传递给 ChangeInt 函数时，按传值的方式复制了变量 b，a 和 b 都指向了同一个 Int 对象，在 a=10 时，则新生成一个 int 值对象 10，并让 a 指向它。

16-6 传可变对象实例

可变对象在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。例如：

```
#!/usr/bin/python3

# 可写函数说明
def changeme( mylist ):
   "修改传入的列表"
   mylist.append([1,2,3,4])
   print ("函数内取值: ", mylist)
   return

# 调用changeme函数
mylist = [10,20,30]
changeme( mylist )
print ("函数外取值: ", mylist)
```

传入函数的和在末尾添加新内容的对象用的是同一个引用。故输出结果如下：

```
函数内取值:  [10, 20, 30, [1, 2, 3, 4]]
函数外取值:  [10, 20, 30, [1, 2, 3, 4]]
```

16-7 函数参数详解

16-7-1 位置参数--------关键字参数---------混合使用

```
1，位置实参:在函数调用阶段，  按照从左到有的顺序依次传入的值
# 特点：按照顺序与形参一一对应

2 关键字参数
# 关键字实参：在函数调用阶段，按照key=value的形式传入的值
# 特点：指名道姓给某个形参传值，可以完全不参照顺序
def func(x,y):
    print(x,y)

func(y=2,x=1) # 关键字参数
func(1,2)  # 位置参数

3，混合使用，强调
    # 1、位置实参必须放在关键字实参前
        def func(x,y):
            print(x,y)
```

```
        func(1,y=2)
        func(y=2,1)

    # 2、不能能为同一个形参重复传值
        def func(x,y):
            print(x,y)
        func(1,y=2,x=3)
        func(1,2,x=3,y=4)
```

16-7-2 默认参数------位置参数与默认参数混用

```
4，默认参数
    # 默认形参：在定义函数阶段，就已经被赋值的形参，称之为默认参数
    # 特点：在定义阶段就已经被赋值，意味着在调用阶段可以不用为其赋值
        def func(x,y=3):
            print(x,y)

        func(x=1)
        func(x=1,y=44444)


        def register(name,age,gender='男'):
             print(name,age,gender)

        register('三炮',18)
        register('二炮',19)
        register('大炮',19)
        register('没炮',19,'女')


5，位置形参与默认形参混用，强调：
    # 1、位置形参必须在默认形参的左边
          def func(y=2,x):  # 错误写法
                pass

    # 2、默认参数的值是在函数定义阶段被赋值的，准确地说被赋予的是值的内存地址
    # 示范1：
        m=2
        def func(x,y=m): # y=>2的内存地址
            print(x,y）
        m=3333333333333333333
        func(1)

    # 3、虽然默认值可以被指定为任意数据类型，但是不推荐使用可变类型
    # 函数最理想的状态：函数的调用只跟函数本身有关系，不外界代码的影响
        m = [111111, ]

        def func(x, y=m):
        print(x, y)

        m.append(3333333)
        m.append(444444)
        m.append(5555)

        func(1)
        func(2)
        func(3)
```

```
    def func(x,y,z,l=None):
        if l is None:
            l=[]
            l.append(x)
            l.append(y)
            l.append(z)
         print(l)

    func(1,2,3)
    func(4,5,6)

    new_l=[111,222]
    func(1,2,3,new_l)
```

16-7-3 可变长度的参数

```
6，可变长度的参数（*与**的用法）
    # 可变长度指的是在调用函数时，传入的值（实参）的个数不固定
    # 而实参是用来为形参赋值的，所以对应着，针对溢出的实参必须有对应的形参来接收

6.1 可变长度的位置参数
    # I：*形参名：用来接收溢出的位置实参，溢出的位置实参会被*保存成元组的格式然后赋值紧跟其后的形参名
        # *后跟的可以是任意名字，但是约定俗成应该是args

        def func(x,y,*z): # z =（3,4,5,6）
            print(x,y,z)

        func(1,2,3,4,5,6)

        def my_sum(*args):
               res=0
               for item in args:
                    res+=item
                return res

            res=my_sum(1,2,3,4,)
            print(res)

    # II：*可以用在实参中，实参中带*，先*后的值打散成位置实参
        def func(x,y,z):
            print(x,y,z)

        func(*[11,22,33]) # func(11，22，33)
        func(*[11,22]) # func(11，22)

        l=[11,22,33]
        func(*l)

    # III：形参与实参中都带*
        def func(x,y,*args): # args=(3,4,5,6)
            print(x,y,args)

        func(1,2,[3,4,5,6])
        func(1,2,*[3,4,5,6]) # func(1,2,3,4,5,6)
        func(*'hello') # func('h','e','l','l','o')
```

```
6.2 可变长度的关键字参数
    # I：**形参名：用来接收溢出的关键字实参，**会将溢出的关键字实参保存成字典格式，然后赋值给紧跟其后的形参名
        # **后跟的可以是任意名字，但是约定俗成应该是kwargs
        def func(x,y,**kwargs):
            print(x,y,kwargs)

        func(1,y=2,a=1,b=2,c=3)

    # II: **可以用在实参中(**后跟的只能是字典)，实参中带**，先**后的值打散成关键字实参
        def func(x,y,z):
            print(x,y,z)

        func(*{'x':1,'y':2,'z':3}) # func('x','y','z')
        func(**{'x':1,'y':2,'z':3}) # func(x=1,y=2,z=3)

    # 错误
        func(**{'x':1,'y':2,}) # func(x=1,y=2)
        func(**{'x':1,'a':2,'z':3}) # func(x=1,a=2,z=3)


    # III: 形参与实参中都带**
        def func(x,y,**kwargs):
           print(x,y,kwargs)

        func(y=222,x=111,a=333,b=444)
        func(**{'y':222,'x':111,'a':333,'b':4444})

    # 混用*与**：*args必须在**kwargs之前
        def func(x,*args,**kwargs):
            print(args)
            print(kwargs)

        func(1,2,3,4,5,6,7,8,x=1,y=2,z=3)


    def index(x,y,z):
        print('index=>>> ',x,y,z)

    def wrapper(*args,**kwargs): #args=(1,) kwargs={'z':3,'y':2}
        index(*args,**kwargs)
        # index(*(1,),**{'z':3,'y':2})
        # index(1,z=3,y=2)

    wrapper(1,z=3,y=2) # 为wrapper传递的参数是给index用的
```

16-7-4 函数的类型提示

```
## : 后面是提示信息，可以随意写
def regidter(name:"不能写艾尼",age:"至少18岁"):
    print(name)
    print(age)

def register(name:str,age:int,hobbies:tuple)->int:  #  返回值类型为 int
    print(name)
```

```
    print(age)
    print(hobbies)

# 添加提示功能的同时，再添加默认值
def register(name:str = 'aini',age:int = 18 ,hobbies:tuple)->int:  #  返回值类型为
int
    print(name)
    print(age)
    print(hobbies)
```

17，Python3 数据结构

17-1 列表

Python中列表是可变的，这是它区别于字符串和元组的最重要的特点，一句话概括即：列表可以修改，而字符串和元组不能。

以下是 Python 中列表的方法：

方法	描述
list.append(x)	把一个元素添加到列表的结尾，相当于 a[len(a):] = [x]。
list.extend(L)	通过添加指定列表的所有元素来扩充列表，相当于 a[len(a):] = L。
list.insert(i, x)	在指定位置插入一个元素。第一个参数是准备插入到其前面的那个元素的索引，例如 a.insert(0, x) 会插入到整个列表之前，而 a.insert(len(a), x) 相当于 a.append(x) 。
list.remove(x)	删除列表中值为 x 的第一个元素。如果没有这样的元素，就会返回一个错误。
list.pop([i])	从列表的指定位置移除元素，并将其返回。如果没有指定索引，a.pop()返回最后一个元素。元素随即从列表中被移除。（方法中 i 两边的方括号表示这个参数是可选的，而不是要求你输入一对方括号，你会经常在 Python 库参考手册中遇到这样的标记。）
list.clear()	移除列表中的所有项，等于del a[:]。
list.index(x)	返回列表中第一个值为 x 的元素的索引。如果没有匹配的元素就会返回一个错误。
list.count(x)	返回 x 在列表中出现的次数。
list.sort()	对列表中的元素进行排序。
list.reverse()	倒排列表中的元素。
list.copy()	返回列表的浅复制，等于a[:]。

下面示例演示了列表的大部分方法：

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
```

```
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

注意：类似 insert, remove 或 sort 等修改列表的方法没有返回值。

17-2 将列表当做堆栈使用

列表方法使得列表可以很方便的作为一个堆栈来使用，堆栈作为特定的数据结构，最先进入的元素最后一个被释放（后进先出）。用 append() 方法可以把一个元素添加到堆栈顶。用不指定索引的 pop() 方法可以把一个元素从堆栈顶释放出来。例如：

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

17-3 将列表当作队列使用

也可以把列表当做队列用，只是在队列里第一加入的元素，第一个取出来；但是拿列表用作这样的目的效率不高。在列表的最后添加或者弹出元素速度快，然而在列表里插入或者从头部弹出速度却不快（因为所有其他的元素都得一个一个地移动）。

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

17-4 列表推导式

- 列表推导式提供了从序列创建列表的简单途径。通常应用程序将一些操作应用于某个序列的每个元素，用其获得的结果作为生成新列表的元素，或者根据确定的判定条件创建子序列。
- 每个列表推导式都在 for 之后跟一个表达式，然后有零到多个 for 或 if 子句。返回结果是一个根据表达从其后的 for 和 if 上下文环境中生成出来的列表。如果希望表达式推导出一个元组，就必须使用括号。
- 这里我们将列表中每个数值乘三，获得一个新的列表：

```
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]
```

现在我们玩一点小花样：

```
>>> [[x, x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]
```

这里我们对序列里每一个元素逐个调用某方法：

```
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
```

我们可以用 if 子句作为过滤器：

```
>>> [3*x for x in vec if x > 3]
[12, 18]
>>> [3*x for x in vec if x < 2]
[]
```

以下是一些关于循环和其它技巧的演示：

```
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [x*y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]
>>> [x+y for x in vec1 for y in vec2]
[6, 5, -7, 8, 7, -5, 10, 9, -3]
>>> [vec1[i]*vec2[i] for i in range(len(vec1))]
[8, 12, -54]
```

列表推导式可以使用复杂表达式或嵌套函数：

```
>>> [str(round(355/113, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

17-5 嵌套列表解析

Python的列表还可以嵌套。

以下实例展示了3X4的矩阵列表：

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

以下实例将3X4的矩阵列表转换为4X3列表：

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

以下实例也可以使用以下方法来实现：

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

另外一种实现方法：

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested listcomp
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

17-6 del 语句

使用 del 语句可以从一个列表中依索引而不是值来删除一个元素。这与使用 pop() 返回一个值不同。可以用 del 语句从列表中删除一个切割，或清空整个列表（我们以前介绍的方法是给该切割赋一个空列表）。例如：

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

也可以用 del 删除实体变量：

```
>>> del a
```

17-7 元组和序列

元组由若干逗号分隔的值组成，例如：

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

如你所见，元组在输出时总是有括号的，以便于正确表达嵌套结构。在输入时可能有或没有括号， 不过括号通常是必须的（如果元组是更大的表达式的一部分）

17-8 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。
- 可以用大括号({})创建集合。注意：如果要创建一个空集合，你必须用 set() 而不是 {}；后者创建一个空的字典，下一节我们会介绍这个数据结构。

以下是一个简单的演示：

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # 删除重复的
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # 检测成员
True
>>> 'crabgrass' in basket
False

>>> # 以下演示了两个集合的操作
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # a 中唯一的字母
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # 在 a 中的字母，但不在 b 中
{'r', 'd', 'b'}
>>> a | b                              # 在 a 或 b 中的字母
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # 在 a 和 b 中都有的字母
{'a', 'c'}
>>> a ^ b                              # 在 a 或 b 中的字母，但不同时在 a 和 b 中
{'r', 'd', 'b', 'm', 'z', 'l'}
```

集合也支持推导式：

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

17-9 字典

- 另一个非常有用的 Python 内建数据类型是字典。
- 序列是以连续的整数为索引，与此不同的是，字典以关键字为索引，关键字可以是任意不可变类型，通常用字符串或数值。
- 理解字典的最佳方式是把它看做无序的键=>值对集合。在同一个字典之内，关键字必须是互不相同。
- 一对大括号创建一个空的字典：{}。

这是一个字典运用的简单例子：

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

构造函数 dict() 直接从键值对元组列表中构建字典。如果有固定的模式，列表推导式指定特定的键值对：

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

此外，字典推导可以用来创建任意键和值的表达式词典：

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

如果关键字只是简单的字符串，使用关键字参数指定键值对有时候更方便：

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

17-10 遍历技巧

在字典中遍历时，关键字和对应的值可以使用 items() 方法同时解读出来：

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

在序列中遍历时，索引位置和对应值可以使用 enumerate() 函数同时得到：

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

同时遍历两个或更多的序列，可以使用 zip() 组合：

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

要反向遍历一个序列，首先指定这个序列，然后调用 reversed() 函数：

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

要按顺序遍历一个序列，使用 sorted() 函数返回一个已排序的序列，并不修改原值

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for f in sorted(set(basket)):
...     print(f)
...
apple
banana
orange
pear
```

18，Python3 模块

18-1 模块

```
## 内置模块
## 第三方模块
## 自定义模块

## 模块的四种形式
1，  使用Python编写的py文件
2，  已被编译为共享库或DLL的C或C++扩展
3，  把一系列模块组织到一起的文件夹（文件夹下面有个__init__.py 该文件夹称为包）
3，  使用C编写并链接到Python解释器的内置模块

import foo
## 首次导入模块会发生什么？
1，执行foo.py
2，产生foo.py的命名空间
3，在当前文件中产生的有一个名字foo,改名字指向2中产生的命名空间

## 无论是调用还是修改与源模块为准，与调用位置无关

## 导入模块规范
1 Python内置模块
2，Python第三方模块
3，自定义模块

## 起别名

import foo as f

## 自定义模块命名应该纯小写+下划线

## 可以在函数内导入模块
```

18-2 写模块时测试

```
# 每个Python文件内置了__name__,指向Python文件名

# 当foo.py 被运行时，
__name__  =  "__main__"

# 当foo.py 被当做模块导入时，
__name__ != "__main__"

##### 测试时可以if判断,在foo.py文件中写以下判断
if __name__ == "__main__" :
    ##  你的测试代码
```

18-3 from xxx import xxx

```
# from foo import x 发生什么事情
1，  产生一个模块的命名空间
2，  运行foo.py 产生，将运行过程中产生的名字都丢到命名空间去
3，  在当前命名空间拿到一个名字，改名字指向模块命名空间
```

18-4 从一个模块导入所有

```
#不太推荐使用
form foo import *
# 被导入模块有个 __all__ = []
__all__ = []    # 存放导入模块里的所有变量和函数，  默认放所有的变量和函数，也可以手动修改


foo.py
    __all__ = ['x','change']
    x = 10
    def change():
        global x
        x = 20
    a = 20
    b = 30

run.py
    from foo import *  ## * 导入的是foo.py里的 __all__ 列表里的变量和函数
    print(x)
    change()
    print(a)  # 会报错，因为foo.py 里的 __all__ 列表里没有a变量
```

18-5 sys.path 模块搜索路径优先级

```
1，  内存（内置模块）
2，  从硬盘查找

import sys
# 值为一个列表，存放了一系列的文件夹
# 其中第一个文件夹是当前执行所在的文件夹
# 第二个文件夹当不存在，因为这不是解释器存放的，是pycharm添加的
print(sys.path)
# sys.path 里放的就是模块的存放路径查找顺序
[
'E:\\Desktop\\python全栈\\模块', 'E:\\Desktop\\python全栈', 'D:\\软件
\\pycharm\\PyCharm 2021.3.1\\plugins\\python\\helpers\\pycharm_display', 'D:\\软
件\\python\\python310.zip', 'D:\\软件\\python\\DLLs', 'D:\\软件\\python\\lib',
'D:\\软件\\python', 'C:\\Users\\艾尼-
aini\\AppData\\Roaming\\Python\\Python310\\site-packages', 'D:\\软件
\\python\\lib\\site-packages', 'D:\\软件\\python\\lib\\site-packages\\win32',
'D:\\软件\\python\\lib\\site-packages\\win32\\lib', 'D:\\软件\\python\\lib\\site-
packages\\Pythonwin', 'D:\\软件\\pycharm\\PyCharm
2021.3.1\\plugins\\python\\helpers\\pycharm_matplotlib_backend'
]
```

18-6 sys.modules 查看内存中的模块

```
import sys
print(sys.module)    # 是一个字典，存放导入的模块

## 可以判断一个模块是否已经在内存中
print('foo' in sys.module)
```

18-7 编写规范的模块

```
"this module is used to ......"    # 第一行文档注释
import sys  # 导入需要用到的包
x = 1  # 定义全局变量
class foo:    # 定义类
    pass
def test():  #定义函数
    pass

if __name__ == "__main__":
    pass
```