Python3 基础语法

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、白俄罗斯语、马其顿语、俄语、塞尔维亚语。

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。

---

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']
```

注释

Python 中单行注释以 # 开头，实例如下:

实例(Python 3.0+)

```
#!/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 """:

实例(Python 3.0+)

```
#!/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!

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：

实例(Python 3.0+)

```
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：

实例

```
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
```

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']
```

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) - 复数由实部和虚部组成，形式为 a + bj，其中 a 是实部，b 是虚部，j 表示虚数单位。如 1 + 2j、 1.1 + 2.2j

---

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。
- 字符串切片 **str[start:end]**，其中 start（包含）是切片开始的索引，end（不包含）是切片结束的索引。
- 字符串的切片可以加上步长参数 step，语法格式如下：**str[start:end:step]**

```
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"""
```

实例(Python 3.0+)

```
#!/usr/bin/python3
str='123456789'
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六个的字符（不包含）
print(str[2:]) # 输出从第三个开始后的所有字符
print(str[1:5:2]) # 输出从第二个开始到第五个且每隔一个的字符（步长为 2）
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，会自动将反斜杠转义，例如：

```
>>> print('\n')      # 输出空行

>>> print(r'\n')     # 输出 \n
\n
>>>
```

以上实例输出结果：

```
123456789
12345678
1
345
3456789
24
123456789123456789
123456789 你好
------------------------------
```

hello
runoob
hello\nrunoob

注:

转义字符	输出
\'	'
\"	"
\a	‘bi’响一声
\b	退格
\f	换页（在打印时）
\n	回车，光标在下一行
\r	换行，光标在上一行
\t	八个空格，横向制表符
\v	纵向制表符
\\	\
\000	空

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。

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:

实例(Python 3.0+)

```
#!/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。

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:

实例(Python 3.0+)

```
#!/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，**runoob** 有 6 个字符，**\n** 表示一个字符，加起来 **7** 个字符。

```
>>> import sys
>>> sys.stdout.write(" hi ")   # hi 前后各有 1 个空格
 hi 4
```

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
```

---

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=""**：

实例(Python 3.0+)

```
#!/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
```

更多内容参考：*Python2 与 Python3 print 不换行*

---

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
```

注：python 标准库 sys：
https://www.ityouknow.com/python/2019/10/09/python-sys-demonstration-028.html
更多内容可以参考：Python import 和 from … import 的主要区别

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。

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:

实例(Python 3.0+)

```
#!/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)
```

运行实例 »

执行以上程序会输出如下结果:

```
100
1000.0
runoob
```

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。

标准数据类型

Python3 中常见的数据类型有:

- Number（数字）
- String（字符串）
- bool（布尔类型）
- 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）。

此外还有一些高级的数据类型，如: 字节数组类型(bytes)。

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：

实例

```
>>> 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：*Python3 中，bool 是 int 的子类，True 和 False 可以和数字相加，* ***True==1、False==0*** *会返回* ***True****，但可以通过* ***is*** *来判断类型。*

```
>>> issubclass(bool, int)
True
>>> True==1
True
>>> False==0
True
>>> True+1
2
>>> False+1
1
>>> 1 is True
False
>>> 0 is False
False
```

在 Python2 中是没有布尔型的，它用数字 0 表示 False，用 1 表示 True。

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

数值运算
实例
```
>>> 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。

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

Python 还支持复数，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。

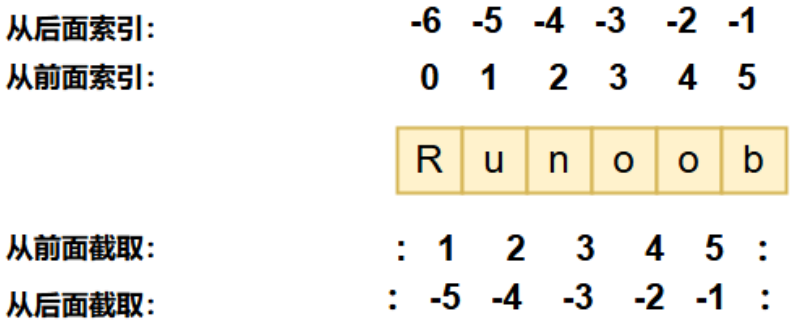
String（字符串）

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]

索引值以 0 为开始值，-1 为从末尾的开始位置。



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

实例

```
#!/usr/bin/python3

str = 'Runoob'  # 定义一个字符串变量

print(str)          # 打印整个字符串
print(str[0:-1])    # 打印字符串第一个到倒数第二个字符（不包含倒数第一个字符）
print(str[0])       # 打印字符串的第一个字符
print(str[2:5])     # 打印字符串第三到第五个字符（包含第五个字符）
print(str[2:])      # 打印字符串从第三个字符开始到末尾
print(str * 2)      # 打印字符串两次
print(str + "TEST")  # 打印字符串和"TEST"拼接在一起
```

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：

实例

```
>>> 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。

实例

```
>>> 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。

---

bool（布尔类型）

布尔类型即 True 或 False。

在 Python 中，True 和 False 都是关键字，表示布尔值。

布尔类型可以用来控制程序的流程，比如判断某个条件是否成立，或者在某个条件满足时执行某段代码。

布尔类型特点：

- 布尔类型只有两个值：True 和 False。
- bool 是 int 的子类，因此布尔值可以被看作整数来使用，其中 True 等价于 1。
- 布尔类型可以和其他数据类型进行比较，比如数字、字符串等。在比较时，Python 会将 True 视为 1，False 视为 0。
- 布尔类型可以和逻辑运算符一起使用，包括 and、or 和 not。这些运算符可以用来组合多个布尔表达式，生成一个新的布尔值。
- 布尔类型也可以被转换成其他数据类型，比如整数、浮点数和字符串。在转换时，True 会被转换成 1，False 会被转换成 0。
- 可以使用 bool() 函数将其他类型的值转换为布尔值。以下值在转换为布尔值时为 False：None、False、零（0、0.0、0j）、空序列（如 ''、()、[]）和空映射（如 {}）。其他所有值转换为布尔值时均为 True。

实例

```
# 布尔类型的值和类型
a = True
b = False
print(type(a))  # <class 'bool'>
print(type(b))  # <class 'bool'>

# 布尔类型的整数表现
print(int(True))   # 1
print(int(False))  # 0

# 使用 bool() 函数进行转换
print(bool(0))        # False
print(bool(42))       # True
print(bool(''))       # False
print(bool('Python'))  # True
print(bool([]))       # False
print(bool([1, 2, 3])) # True
```

```
# 布尔逻辑运算
print(True and False)  # False
print(True or False)   # True
print(not True)        # False

# 布尔比较运算
print(5 > 3)  # True
print(2 == 2) # True
print(7 < 4)  # False

# 布尔值在控制流中的应用
if True:
    print("This will always print")

if not False:
    print("This will also always print")

x = 10
if x:
    print("x is non-zero and thus True in a boolean context")
```

注意: 在 Python 中，所有非零的数字和非空的字符串、列表、元组等数据类型都被视为 True，只有 **0、空字符串、空列表、空元组**等被视为 False。因此，在进行布尔类型转换时，需要注意数据类型的真假性。

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）。

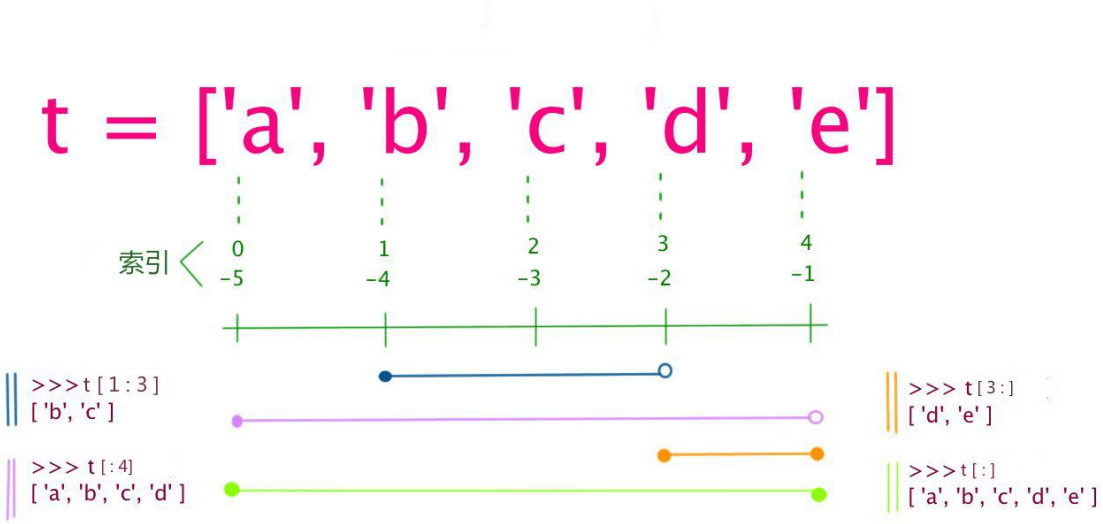
列表是写在方括号 **[]** 之间、用逗号分隔开的元素列表。

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：

```
变量[头下标:尾下标]
```

索引值以 **0** 为开始值，**-1** 为从末尾的开始位置。



加号 + 是列表连接运算符，星号 * 是重复操作。如下实例：

实例

```
#!/usr/bin/python3

list = [ 'abcd', 786 , 2.23, 'runoob', 70.2 ]  # 定义一个列表
tinylist = [123, 'runoob']

print (list)            # 打印整个列表
print (list[0])         # 打印列表的第一个元素
print (list[1:3])       # 打印列表第二到第四个元素（不包含第四个元素）
print (list[2:])        # 打印列表从第三个元素开始到末尾
print (tinylist * 2)    # 打印 tinylist 列表两次
print (list + tinylist) # 打印两个列表拼接在一起的结果
```

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：

实例

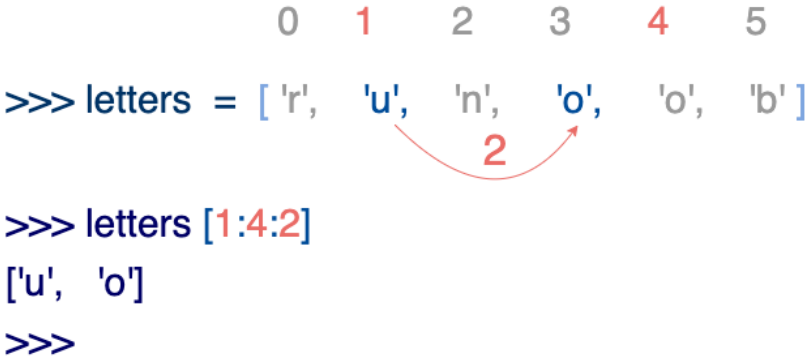
```
>>> a = [1, 2, 3, 4, 5, 6]
>>> a[0] = 9
>>> a[2:5] = [13, 14, 15]
>>> a
[9, 2, 13, 14, 15, 6]
>>> a[2:5] = []   # 将对应的元素值设置为 []
>>> a
[9, 2, 6]
```

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、列表写在方括号之间，元素用逗号隔开。
- 2、和字符串一样，列表可以被索引和切片。
- 3、列表可以使用 + 操作符进行拼接。
- 4、列表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：

实例

```
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
```

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：

实例

```
#!/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。

实例

```
>>> 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
```

如果你想创建只有一个元素的元组，需要注意在元素后面添加一个逗号，以区分它是一个元组而不是一个普通的值，这是因为在没有逗号的情况下，Python 会将括号解释为数学运算中的括号，而不是元组的表示。

如果不添加逗号，如下所示，它将被解释为一个普通的值而不是元组：

```
not_a_tuple = (42)
```

这样的话，not_a_tuple 将是整数类型而不是元组类型。

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。

Set（集合）

Python 中的集合（Set）是一种无序、可变的数据类型，用于存储唯一的元素。

集合中的元素不会重复，并且可以进行交集、并集、差集等常见的集合操作。

在 Python 中，集合使用大括号 **{}** 表示，元素之间用逗号 **,** 分隔。

另外，也可以使用 **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)

实例

```
#!/usr/bin/python3

sites = {'Google', 'Taobao', 'Runoob', 'Facebook', 'Zhihu', 'Baidu'}

print(sites)  # 输出集合，重复的元素被自动去掉

# 成员测试
if 'Runoob' in sites :
    print('Runoob 在集合中')
else :
    print('Runoob 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：

```
{'Zhihu', 'Baidu', 'Taobao', 'Runoob', 'Google', 'Facebook'}
Runoob 在集合中
{'b', 'c', 'a', 'r', 'd'}
{'r', 'b', 'd'}
{'b', 'c', 'a', 'z', 'm', 'r', 'l', 'd'}
{'c', 'a'}
```

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

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。

实例

```
#!/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：

实例

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

{x: x2 for x in (2, 4, 6)}** 该代码使用的是字典推导式，更多推导式内容可以参考：Python 推导式。
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 **{ }**。

---

bytes 类型

在 Python3 中，bytes 类型表示的是不可变的二进制序列（byte sequence）。
与字符串类型不同的是，bytes 类型中的元素是整数值（0 到 255 之间的整数），而不是 Unicode 字符。
bytes 类型通常用于处理二进制数据，比如图像文件、音频文件、视频文件等等。在网络编程中，也经常使用 bytes 类型来传输二进制数据。

创建 bytes 对象的方式有多种，最常见的方式是使用 b 前缀：
此外，也可以使用 bytes() 函数将其他类型的对象转换为 bytes 类型。bytes() 函数的第一个参数是要转换的对象，第二个参数是编码方式，如果省略第二个参数，则默认使用 UTF-8 编码：

```
x = bytes("hello", encoding="utf-8")
```

与字符串类型类似，bytes 类型也支持许多操作和方法，如切片、拼接、查找、替换等等。同时，由于 bytes 类型是不可变的，因此在进行修改操作时需要创建一个新的 bytes 对象。例如：
实例

```
x = b"hello"
y = x[1:3]  # 切片操作，得到 b"el"
z = x + b"world"  # 拼接操作，得到 b"helloworld"
```

需要注意的是，bytes 类型中的元素是整数值，因此在进行比较操作时需要使用相应的整数值。例如：
实例

```
x = b"hello"
if x[0] == ord("h"):
    print("The first element is 'h'")
```

其中 ord() 函数用于将字符转换为相应的整数值。

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，在下一章节 Python3 数据类型转换 会具体介绍。
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

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
```

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!
```

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!
```

Python3 注释

在 Python3 中，注释不会影响程序的执行，但是会使代码更易于阅读和理解。

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!")
```

符号后面的所有文本都被视为注释，不会被解释器执行。

多行注释

在 Python 中，多行字符串（由三个单引号 ''' 或三个双引号 """ 包围的文本块）在某些情况下可以被视为一种实现多行注释的技巧。

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：

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!")
```

2、双引号（"""）

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

***注意**：虽然多行字符串在这里被当作多行注释使用，但它实际上是一个字符串，我们只要不使用它，它不会影响程序的运行。*

这些字符串在代码中可以被放置在一些位置，而不引起实际的执行，从而达到注释的效果。

拓展说明

在 Python 中，多行注释是由三个单引号 ''' 或三个双引号 """ 来定义的，而且这种注释方式并不能嵌套使用。当你开始一个多行注释块时，Python 会一直将后续的行都当作注释，直到遇到另一组三个单引号或三个双引号。

嵌套多行注释会导致语法错误。

例如，下面的示例是不合法的：

实例

```
'''
这是外部的多行注释
可以包含一些描述性的内容

    '''
        这是尝试嵌套的多行注释
        会导致语法错误
```

'''
'''
在这个例子中，内部的三个单引号并没有被正确识别为多行注释的结束，而是被解释为普通的字符串。
这将导致代码结构不正确，最终可能导致语法错误。
如果你需要在注释中包含嵌套结构，推荐使用单行注释（以#开头）而不是多行注释。
单行注释可以嵌套在多行注释中，而且不会引起语法错误。例如：
实例
'''
这是外部的多行注释
可以包含一些描述性的内容

这是内部的单行注释
可以嵌套在多行注释中
'''

这样的结构是合法的，并且通常能够满足文档化和注释的需求。

Python3 运算符

什么是运算符?

本章节主要说明 Python 的运算符。

举个简单的例子:

4 + 5 = 9

例子中，**4** 和 **5** 被称为**操作数**，**+** 称为**运算符**。

Python 语言支持以下类型的运算符:

- 算术运算符
- 比较（关系）运算符
- 赋值运算符
- 逻辑运算符
- 位运算符
- 成员运算符
- 身份运算符
- 运算符优先级

接下来让我们一个个来学习 Python 的运算符。

Python 算术运算符

以下假设变量 **a=10**，变量 **b=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 - 往小的方向取整数	>>> 9//2
		4
		>>> -9//2
		-5

以下实例演示了 Python 所有算术运算符的操作:

实例(Python 3.0+)

```
#!/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
```

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：

实例(Python 3.0+)

```
#!/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
```

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，这个运算符的主要目的是在表达式中同时进行赋值和返回赋值的值。**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)")

以下实例演示了 Python 所有赋值运算符的操作:

实例(Python 3.0+)

```
#!/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

在 Python 3.8 及更高版本中，引入了一种新的语法特性，称为"海象运算符"（Walrus Operator），它使用 **:=** 符号。这个运算符的主要目的是在表达式中同时进行赋值和返回赋值的值。
使用海象运算符可以在一些情况下简化代码，尤其是在需要在表达式中使用赋值结果的情况下。这对于简化循环条件或表达式中的重复计算很有用。
下面是一个简单的实例，演示了海象运算符的使用：
实例

```
# 传统写法
n = 10
if n > 5:
    print(n)

# 使用海象运算符
if (n := 10) > 5:
    print(n)
```

1. if (n := 10) > 5:：这是使用海象运算符（:=）的写法。海象运算符在表达式中进行赋值操作。
   - (n := 10)：将变量 n 赋值为 10，同时返回这个赋值结果。
   - >5：检查赋值后的 n 是否大于 5。如果条件为真，则执行接下来的代码块。
2. print(n)：如果条件为真，打印变量 n 的值（即 10）。

海象运算符的优点：

- 海象运算符（:=）允许在表达式内部进行赋值，这可以减少代码的重复，提高代码的可读性和简洁性。
- 在上述例子中，传统写法需要单独一行来赋值 n，然后在 if 语句中进行条件检查。而使用海象运算符的写法可以在 if 语句中直接进行赋值和条件检查。

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

以下实例演示了 Python 所有位运算符的操作：

实例(Python 3.0+)

```
#!/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
```

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 x 的值，否则返回 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

以上实例输出结果:

实例(Python 3.0+)

```
#!/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

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。

以下实例演示了 Python 所有成员运算符的操作：

实例(Python 3.0+)

```
#!/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
```

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	**x is not y**， 类似 **id(x) != id(y)**。如果引用的不是同一个对象

	用自不同对象	则返回结果 True，否则返回 False。

注： id() 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：

实例(Python 3.0+)

```
#!/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

Python 运算符优先级

以下表格列出了从最高到最低优先级的所有运算符， 相同单元格内的运算符具有相同优先级。 运算符均指二元运算，除非特别指出。 相同单元格内的运算符从左至右分组（除了幂运算是从右至左分组）：

运算符	描述
(expressions...), [expressions...], {key: value...}, {expressions...}	圆括号的表达式
x[index], x[index:index], x(arguments...), x.attribute	读取，切片，调用，属性引用

运算符	描述
await x	await 表达式
**	乘方(指数)
+x, -x, ~x	正，负，按位非 NOT
*, @, /, //, %	乘，矩阵乘，除，整除，取余
+, -	加和减
<<, >>	移位
&	按位与 AND
^	按位异或 XOR
|	按位或 OR
in,not in, is,is not, <, <=, >, >=, !=, ==	比较运算，包括成员检测和标识号检测
not x	逻辑非 NOT
and	逻辑与 AND
or	逻辑或 OR
if -- else	条件表达式
lambda	lambda 表达式
:=	赋值表达式

以下实例演示了 Python 所有运算符优先级的操作：

实例(Python 3.0+)

```
#!/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

and 拥有更高优先级：
实例
```
x = True
y = False
z = False

if x or y and z:
   print("yes")
else:
   print("no")
```

以上实例先计算 **y and z** 并返回 False， 然后 **x or False** 返回 True，输出结果：
Yes

注意：*Python3 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
```

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。布尔(bool)是整型的子类型。
- **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 = $2.5 \times 10^2 = 250$）
- **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

- Python 支持复数，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。

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
```

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。

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)	以浮点数形式返回数字的绝对值，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。 **其实准确的说是保留值将保留到离上一位更近的一端。**
sqrt(x)	返回数字 x 的平方根。

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。

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

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）。

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"

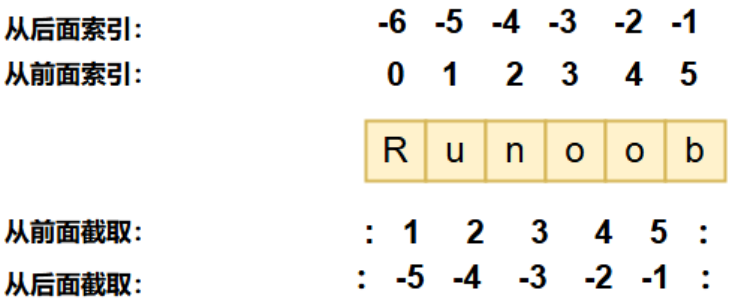
Python 访问字符串中的值

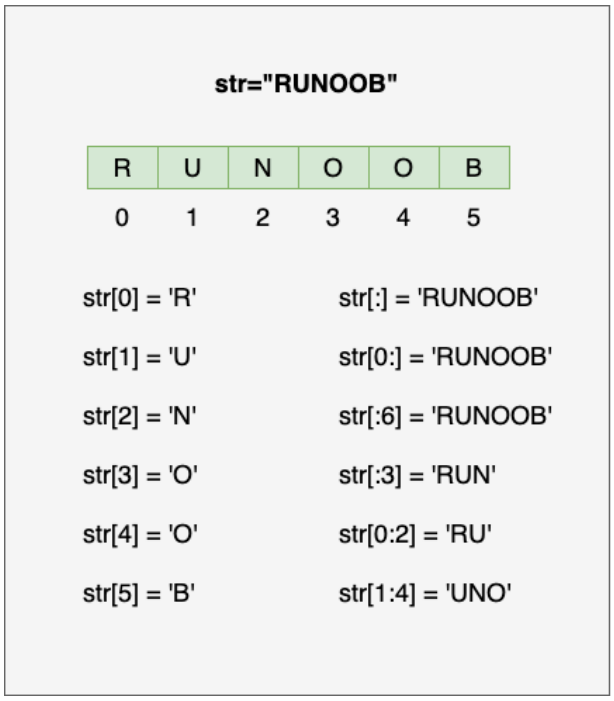
Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。

Python 访问子字符串，可以使用方括号 **[]** 来截取字符串，字符串的截取的语法格式如下：

变量[头下标:尾下标]

索引值以 **0** 为开始值，**-1** 为从末尾的开始位置。





如下实例：

实例(Python 3.0+)

```
#!/usr/bin/python3 v
ar1 = '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
```

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:

实例(Python 3.0+)

```
#!/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!
```

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	实例
\(在行尾时)	续行符	>>> print("line1 \ ... line2 \ ... line3") line1 line2 line3 >>>
\\	反斜杠符号	>>> print("\\") \
\'	单引号	>>> print('\'') '
\"	双引号	>>> print("\"") "
\a	响铃	>>> print("\a") 执行后电脑有响声。
\b	退格(Backspace)	>>> print("Hello \b World!") Hello World!
\000	空	>>> print("\000") >>>
\n	换行	>>> print("\n") >>>
\v	纵向制表符	>>> print("Hello \v World!") Hello World! >>>
\t	横向制表符	>>> print("Hello \t World!") Hello World! >>>

转义字符	描述	实例

\r	回车，将 **\r** 后面的内容移到字符串开头，并逐一替换开头部分的字符，直至将 **\r** 后面的内容完全替换完成。	>>> print("Hello\rWorld!")
		World!
		>>> print('google runoob taobao\r123456')
		123456 runoob taobao

\f	换页	>>> print("Hello \f World!") Hello World! >>>
\yyy	八进制数，y 代表 0~7 的字符，例如：\012 代表换行。	>>> print("\110\145\154\154\157\40\127\157\162\154\144\41") Hello World!
\xyy	十六进制数，以 \x 开头，y 代表的字符，例如：\x0a 代表换行	>>> print("\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64\x21") Hello World!
\other	其它的字符以普通格式输出	

使用 **\r** 实现百分比进度：

实例

```
import time

for i in range(101):
    print("\r{:3}%".format(i),end=' ')
    time.sleep(0.05)
```

以下实例，我们使用了不同的转义字符来演示单引号、换行符、制表符、退格符、换页符、ASCII、二进制、八进制数和十六进制数的效果：

实例

```
print('\'Hello, world!\'')  # 输出：'Hello, world!'

print("Hello, world!\nHow are you?")  # 输出：Hello, world!
                                      #       How are you?

print("Hello, world!\tHow are you?")  # 输出：Hello, world!    How are you?

print("Hello,\b world!")  # 输出：Hello world!

print("Hello,\f world!")  # 输出：
                          # Hello,
                          #  world!

print("A 对应的 ASCII 值为：", ord('A'))  # 输出：A 对应的 ASCII 值为：  65

print("\x41 为 A 的 ASCII 代码")  # 输出：A 为 A 的 ASCII 代码
```

```
decimal_number = 42
binary_number = bin(decimal_number)  # 十进制转换为二进制
print('转换为二进制:', binary_number)  # 转换为二进制: 0b101010

octal_number = oct(decimal_number)  # 十进制转换为八进制
print('转换为八进制:', octal_number)  # 转换为八进制: 0o52

hexadecimal_number = hex(decimal_number)  # 十进制转换为十六进制
print('转换为十六进制:', hexadecimal_number) # 转换为十六进制: 0x2a
```

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。

实例(Python 3.0+)

```
#!/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
```

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。

实例(Python 3.0+)

```
#!/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:

<tbody</tbody

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)

Python2.6 开始，新增了一种格式化字符串的函数 str.format()，它增强了字符串格式化的功能。

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

实例(Python 3.0+)

```
#!/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)
''')
```

---

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 (%):

实例

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

f-string 格式化字符串以 **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:

实例

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

实例

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

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

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。

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 suffix 结束，如果 beg 或者 end 指定则检查指定的范围内是否以 suffix 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() 检查字符串是否由字母和数字组成，即字符串中的所有字符都是字母或数字。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或中文字则返回 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 old 替换成 new,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() 删除字符串末尾的空格或指定字符。
31	split(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 table 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。

Python3 列表

序列是 Python 中最基本的数据结构。

序列中的每个值都有对应的位置值，称之为索引，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。

列表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

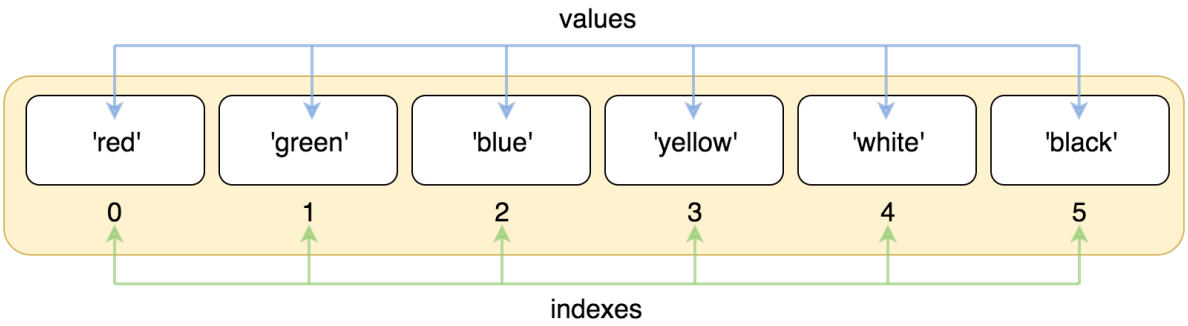
创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。如下所示：

```
list1 = ['Google', 'Runoob', 1997, 2000]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]
list4 = ['red', 'green', 'blue', 'yellow', 'white', 'black']
```

访问列表中的值

与字符串的索引一样，列表索引从 **0** 开始，第二个索引是 **1**，依此类推。

通过索引列表可以进行截取、组合等操作。



实例

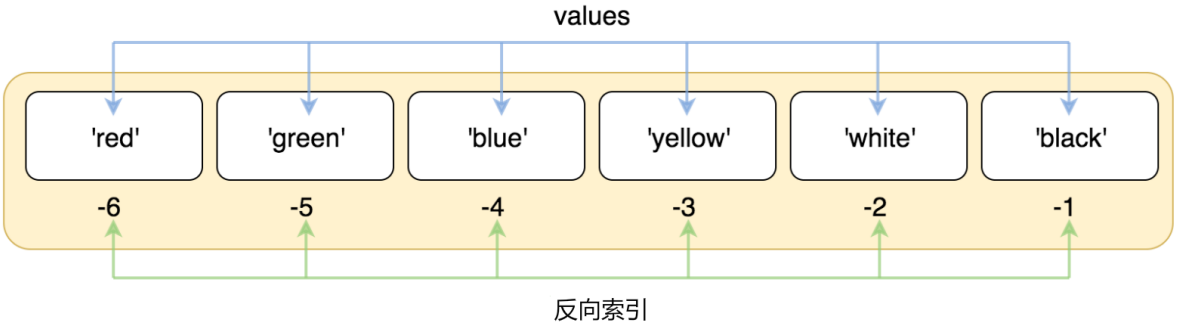
```
#!/usr/bin/python3

list = ['red', 'green', 'blue', 'yellow', 'white', 'black']
print( list[0] )
print( list[1] )
print( list[2] )
```

以上实例输出结果：

```
red
green
blue
```

索引也可以从尾部开始，最后一个元素的索引为 **-1**，往前一位为 **-2**，以此类推。



实例

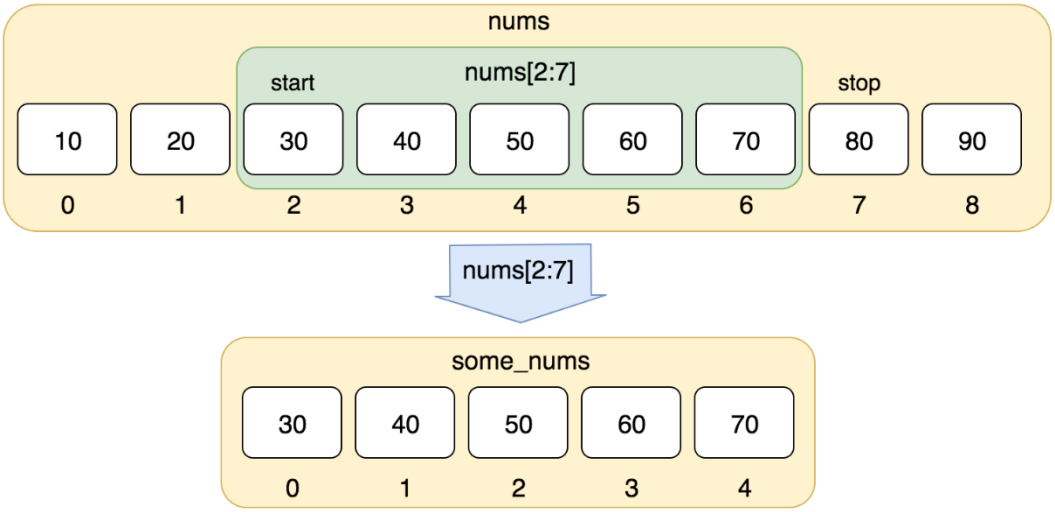
```
#!/usr/bin/python3

list = ['red', 'green', 'blue', 'yellow', 'white', 'black']
print( list[-1] )
print( list[-2] )
print( list[-3] )
```

以上实例输出结果：

```
black
white
yellow
```

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：



实例

```
#!/usr/bin/python3

nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
print(nums[0:4])
```

以上实例输出结果：

```
[10, 20, 30, 40]
```

使用负数索引值截取：

实例

```
#!/usr/bin/python3

list = ['Google', 'Runoob', "Zhihu", "Taobao", "Wiki"]

# 读取第二位
print ("list[1]: ", list[1])
# 从第二位开始（包含）截取到倒数第二位（不包含）
print ("list[1:-2]: ", list[1:-2])
```

以上实例输出结果：

```
list[1]:  Runoob
list[1:-2]:  ['Runoob', 'Zhihu']
```

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：

实例(Python 3.0+)

```
#!/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])

list1 = ['Google', 'Runoob', 'Taobao']
list1.append('Baidu')
print ("更新后的列表 : ", list1)
```

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
更新后的列表 :  ['Google', 'Runoob', 'Taobao', 'Baidu']
```

删除列表元素

可以使用 del 语句来删除列表中的元素，如下实例：

实例(Python 3.0+)

```
#!/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

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

Python 列表截取与拼接

Python 的列表截取与字符串操作类似，如下所示：

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

操作：

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]
>>>
```

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'
```

列表比较

列表比较需要引入 **operator** 模块的 **eq** 方法（详见：Python operator 模块）：

实例

```
# 导入 operator 模块
import operator
```

```
a = [1, 2]
b = [2, 3]
c = [2, 3]
print("operator.eq(a,b): ", operator.eq(a,b))
print("operator.eq(c,b): ", operator.eq(c,b))
```

以上代码输出结果为:

```
operator.eq(a,b):  False
operator.eq(c,b):  True
```

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

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。

实例(Python 3.0+)

```
>>> 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 = ()
```

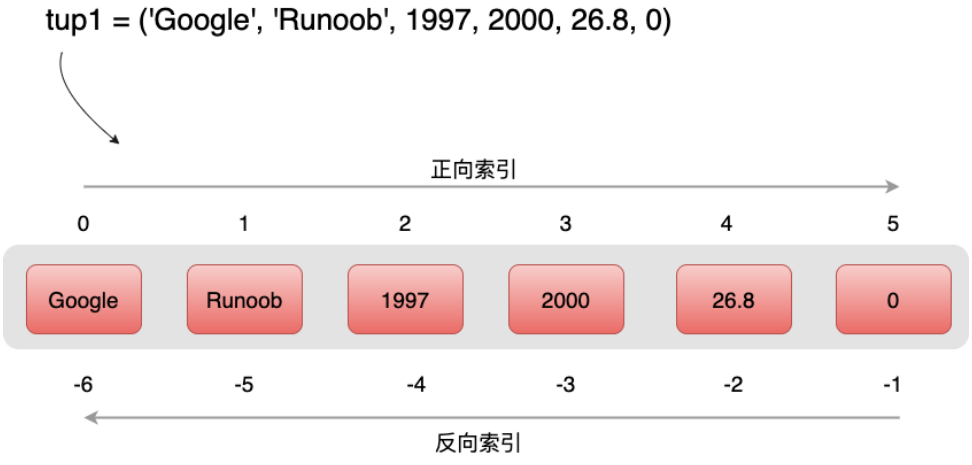
元组中只包含一个元素时，需要在元素后面添加逗号 **,** ，否则括号会被当作运算符使用:

实例(Python 3.0+)

```
>>> 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。



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:

实例(Python 3.0+)

```
#!/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)
```

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：

实例(Python 3.0+)

```
#!/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')
```

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：

实例(Python 3.0+)

```
#!/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
```

元组运算符

与字符串一样，元组之间可以使用 **+**、**+=**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

>>> a = (1, 2, 3)
>>> b = (4, 5, 6)
>>> c = a+b
>>> c
(1, 2, 3, 4, 5, 6)
	(1, 2, 3, 4, 5, 6)	连接，c 就是一个新的元组，它包含了 a 和 b 中的所有元素。

>>> a = (1, 2, 3)
>>> b = (4, 5, 6)
>>> a += b
>>> a
(1, 2, 3, 4, 5, 6)
	(1, 2, 3, 4, 5, 6)	连接，a 就变成了一个新的元组，它包含了 a 和 b 中的所有元素。

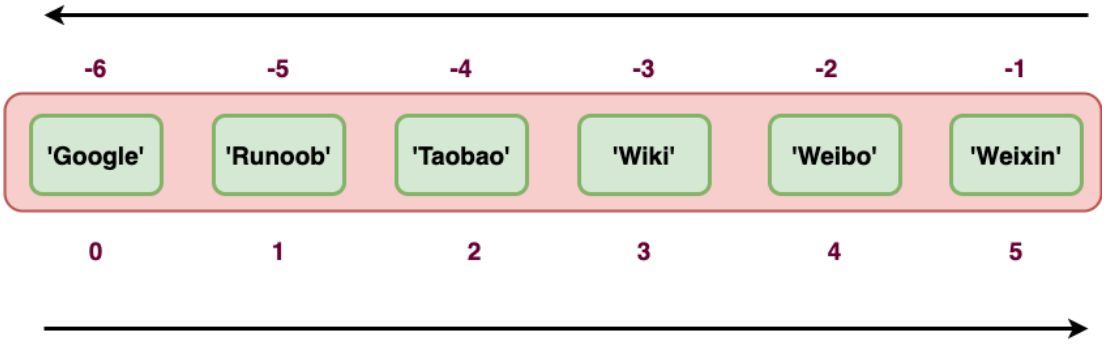
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	复制
3 in (1, 2, 3)	True	元素是否存在
for x in (1, 2, 3): print (x, end=" ")	1 2 3	迭代

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：

tup = ('Google', 'Runoob', 'Taobao', 'Wiki', 'Weibo','Weixin')



Python 表达式	结果	描述
tup[1]	'Runoob'	读取第二个元素
tup[-2]	'Weibo'	反向读取，读取倒数第二个元素
tup[1:]	('Runoob', 'Taobao', 'Wiki', 'Weibo', 'Weixin')	截取元素，从第二个开始后的所有元素。
tup[1:4]	('Runoob', 'Taobao', 'Wiki')	截取元素，从第二个开始到第四个元素（索引为 3）。

运行实例如下：

实例

```
>>> tup = ('Google', 'Runoob', 'Taobao', 'Wiki', 'Weibo','Weixin')
>>> tup[1]
'Runoob'
>>> tup[-2]
```

```
'Weibo'
>>> tup[1:]
('Runoob', 'Taobao', 'Wiki', 'Weibo', 'Weixin')
>>> tup[1:4]
('Runoob', 'Taobao', 'Wiki')
>>>
```

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(iterable)
	将可迭代系列转换为元组。
		>>> list1= ['Google', 'Taobao', 'Runoob', 'Baidu']
		>>> tuple1=tuple(list1)
		>>> tuple1
		('Google', 'Taobao', 'Runoob', 'Baidu')

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容不可变。

```
>>> 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。

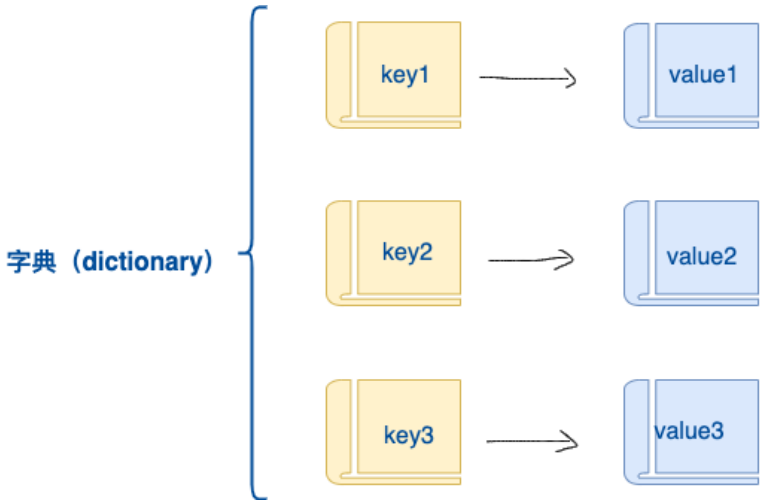
Python3 字典

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, key3 : value3 }

注意：dict 作为 Python 的关键字和内置函数，变量名不建议命名为 **dict**。

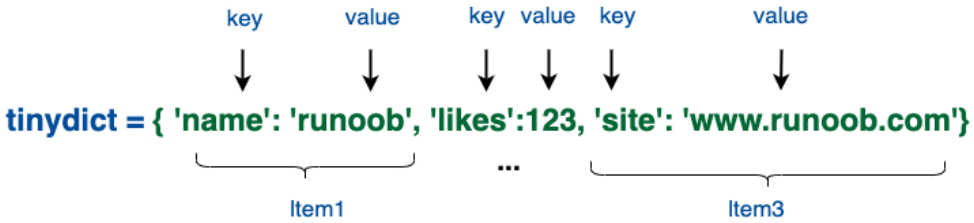


键必须是唯一的，但值则不必。

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。

一个简单的字典实例：

tinydict = {'name': 'runoob', 'likes': 123, 'url': 'www.runoob.com'}



也可如此创建字典：

tinydict1 = { 'abc': 456 }

tinydict2 = { 'abc': 123, 98.6: 37 }

创建空字典

使用大括号 **{ }** 创建空字典：

实例

```
# 使用大括号 {} 来创建空字典
emptyDict = {}

# 打印字典
print(emptyDict)

# 查看字典的数量
print("Length:", len(emptyDict))

# 查看类型
print(type(emptyDict))
```

以上实例输出结果：
```
{}
Length: 0
<class 'dict'>
```

使用内建函数 **dict()** 创建字典：

实例

```
emptyDict = dict()

# 打印字典
print(emptyDict)

# 查看字典的数量
print("Length:",len(emptyDict))

# 查看类型
print(type(emptyDict))
```

以上实例输出结果：
```
{}
Length: 0
<class 'dict'>
```

访问字典里的值

把相应的键放入到方括号中，如下实例：

实例

```
#!/usr/bin/python3
tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}
print ("tinydict['Name']: ", tinydict['Name'])
print ("tinydict['Age']: ", tinydict['Age'])
```

以上实例输出结果：
```
tinydict['Name']:  Runoob
tinydict['Age']:  7
```

如果用字典里没有的键访问数据，会输出错误如下：

实例

```
#!/usr/bin/python3
tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}
print ("tinydict['Alice']: ", tinydict['Alice'])
```

以上实例输出结果：
```
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    print ("tinydict['Alice']: ", tinydict['Alice'])
KeyError: 'Alice'
```

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:

实例

```
#!/usr/bin/python3
tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

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

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

以上实例输出结果:

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

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。

显式删除一个字典用 del 命令，如下实例:

实例

```
#!/usr/bin/python3
tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

del tinydict['Name'] # 删除键 'Name'
tinydict.clear() # 清空字典
del tinydict # 删除字典
print ("tinydict['Age']: ", tinydict['Age'])
print ("tinydict['School']: ", tinydict['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 "/runoob-test/test.py", line 9, in <module>
    print ("tinydict['Age']: ", tinydict['Age'])
NameError: name 'tinydict' is not defined
```

注：del() 方法后面也会讨论。

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:

实例

```
#!/usr/bin/python3
tinydict = {'Name': 'Runoob', 'Age': 7, 'Name': '小菜鸟'}
print ("tinydict['Name']: ", tinydict['Name'])
```

以上实例输出结果:

```
tinydict['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：

实例

```
#!/usr/bin/python3
tinydict = {['Name']: 'Runoob', 'Age': 7}
print ("tinydict['Name']: ", tinydict['Name'])
```

以上实例输出结果：

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

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。
		>>> tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}
		>>> len(tinydict)
		3

2	str(dict)
	输出字典，可以打印的字符串表示。
		>>> tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}
		>>> str(tinydict)
		"{'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。
		>>> tinydict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}
		>>> type(tinydict)
		<class 'dict'>

Python 字典包含了以下内置方法：

序号	函数及描述
1	dict.clear() 删除字典内所有元素
2	dict.copy() 返回一个字典的浅复制
3	dict.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	dict.get(key, default=None) 返回指定键的值，如果键不在字典中返回 default 设置的默认值
5	key in dict 如果键在字典 dict 里返回 true，否则返回 false
6	dict.items() 以列表返回一个视图对象

7	dict.keys() 返回一个视图对象
8	dict.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	dict.update(dict2) 把字典 dict2 的键/值对更新到 dict 里
10	dict.values() 返回一个视图对象
11	pop(key[,default]) 删除字典 key（键）所对应的值，返回被删除的值。
12	popitem() 返回并删除字典中的最后一对键和值。

Python3 集合

集合（set）是一个无序的不重复元素序列。

集合中的元素不会重复，并且可以进行交集、并集、差集等常见的集合操作。

可以使用大括号 **{ }** 创建集合，元素之间用逗号 **,** 分隔， 或者也可以使用 **set()** 函数创建集合。

创建格式：

parame = {value01,value02,...}

或者

set(value)

以下是一个简单实例：

```
set1 = {1, 2, 3, 4}          # 直接使用大括号创建集合
set2 = set([4, 5, 6, 7])     # 使用 set() 函数从列表创建集合
```

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。

更多实例演示：

实例(Python 3.0+)

```
>>> 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):

实例(Python 3.0+)

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

集合的基本操作

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。

实例(Python 3.0+)

```
>>> 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。

实例(Python 3.0+)

```
>>> 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'}
>>>
```

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。

实例(Python 3.0+)

```
>>> 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 )
```

实例(Python 3.0+)

```
>>> 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()
```

脚本模式实例(Python 3.0+)

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

输出结果:
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。

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。
实例(Python 3.0+)
```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> len(thisset)
3
```

4、清空集合
语法格式如下：
s.clear()
清空集合 s。
实例(Python 3.0+)
```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.clear()
>>> print(thisset)
set()
```

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。
实例(Python 3.0+)
```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> "Runoob" in thisset
True
>>> "Facebook" in thisset
False
>>>
```

集合内置方法完整列表

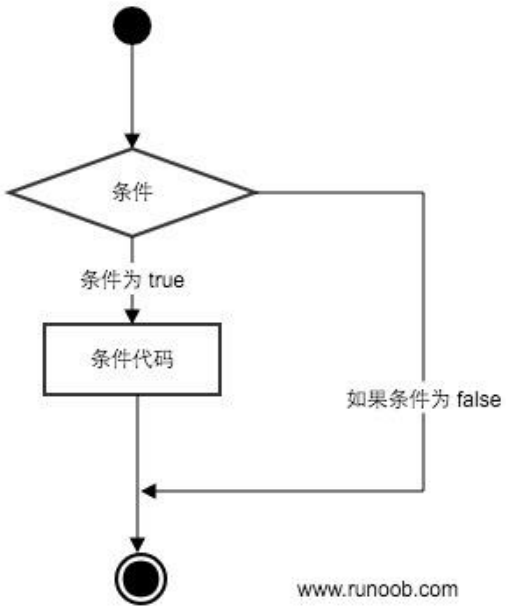
方法	描述
add()	为集合添加元素
clear()	移除集合中的所有元素
copy()	拷贝一个集合
difference()	返回多个集合的差集
difference_update()	移除集合中的元素，该元素在指定的集合也存在。
discard()	删除集合中指定的元素

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
	中。
union()	返回两个集合的并集
update()	给集合添加元素
len()	计算集合元素个数

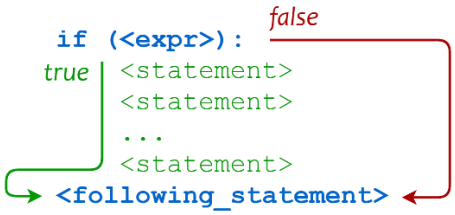
Python3 条件控制

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。

可以通过下图来简单了解条件语句的执行过程:



代码执行过程:



---

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
```

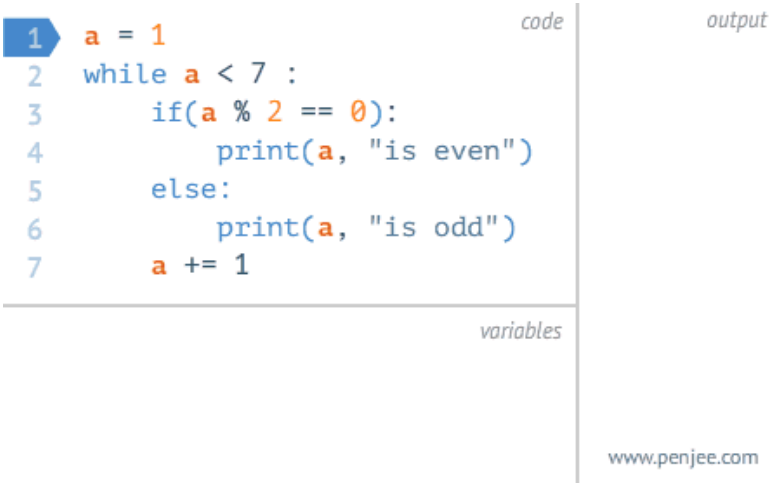
- 如果 "condition_1" 为 True 将执行 "statement_block_1" 块语句
- 如果 "condition_1" 为 False，将判断 "condition_2"
- 如果"condition_2" 为 True 将执行 "statement_block_2" 块语句
- 如果 "condition_2" 为 False，将执行"statement_block_3"块语句

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，但在 Python3.10 版本添加了 **match…case**，功能也类似，详见下文。

Gif 演示:



实例

以下是一个简单的 if 实例:

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:

实例

```
#!/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

实例

```
#!/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：

实例

```
#!/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！
```

---

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
```

实例

```
# !/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

---

match...case

Python 3.10 增加了 **match...case** 的条件判断，不需要再使用一连串的 **if-else** 来判断了。

match 后的对象会依次与 case 后的内容进行匹配，如果匹配成功，则执行匹配到的表达式，否则直接跳过，_ 可以匹配一切。

语法格式如下：

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

case _: 类似于 C 和 Java 中的 **default:**，当其他 case 都无法匹配时，匹配这条，保证永远会匹配成功。

实例

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"

mystatus=400
print(http_error(400))
```

以上是一个输出 HTTP 状态码的实例，输出结果为：

```
Bad request
```

一个 case 也可以设置多个匹配条件，条件使用 | 隔开，例如：

```
...
    case 401|403|404:
        return "Not allowed"
```

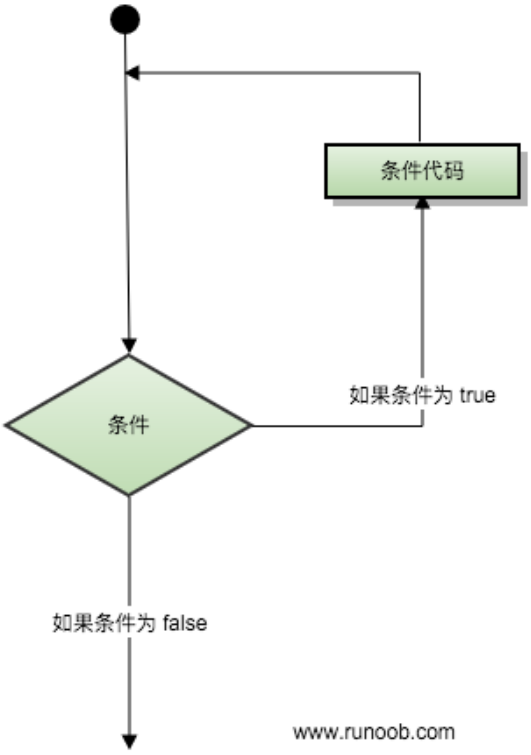
match...case 更多内容参考：Python match-case 语句

Python3 循环语句

本章节将为大家介绍 Python 循环语句的使用。

Python 中的循环语句有 for 和 while。

Python 循环语句的控制结构图如下所示：



---

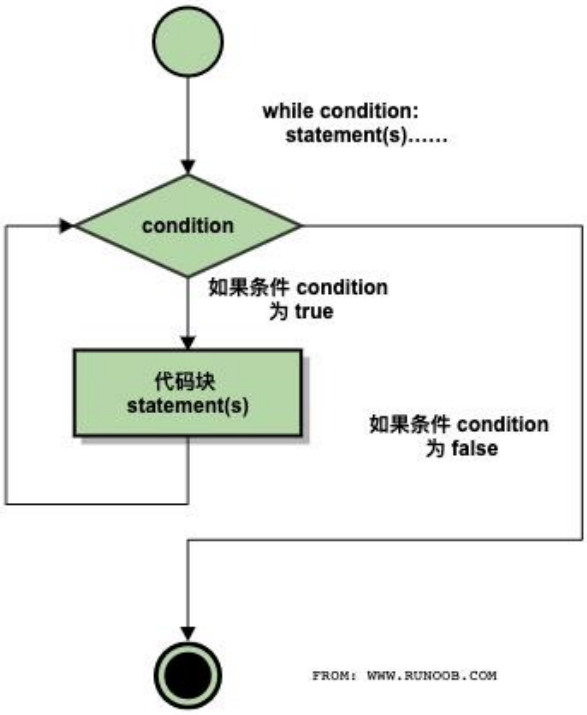
while 循环

Python 中 while 语句的一般形式：

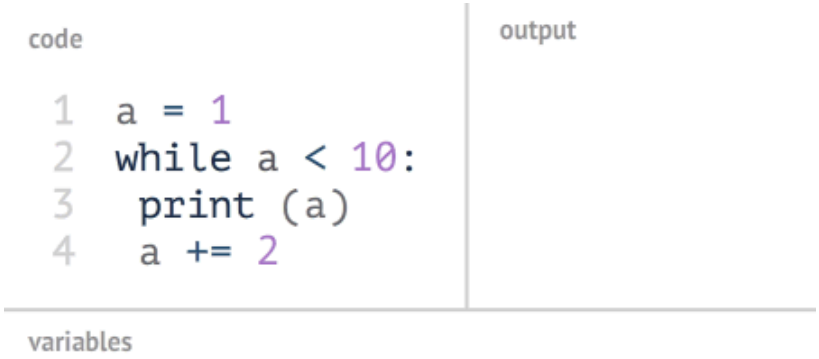
while 判断条件(condition)：

　　执行语句(statements)……

执行流程图如下：



执行 Gif 演示：



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：

实例

```
#!/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
```

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：

实例

```
#!/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。

while 循环使用 else 语句

如果 while 后面的条件语句为 false 时，则执行 else 的语句块。

语法格式如下：

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

expr 条件语句为 true 则执行 statement(s) 语句块，如果为 false，则执行 additional_statement(s)。

循环输出数字，并判断大小：

实例

```
#!/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
```

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：

实例

```
#!/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!
.....
```

---

for 语句

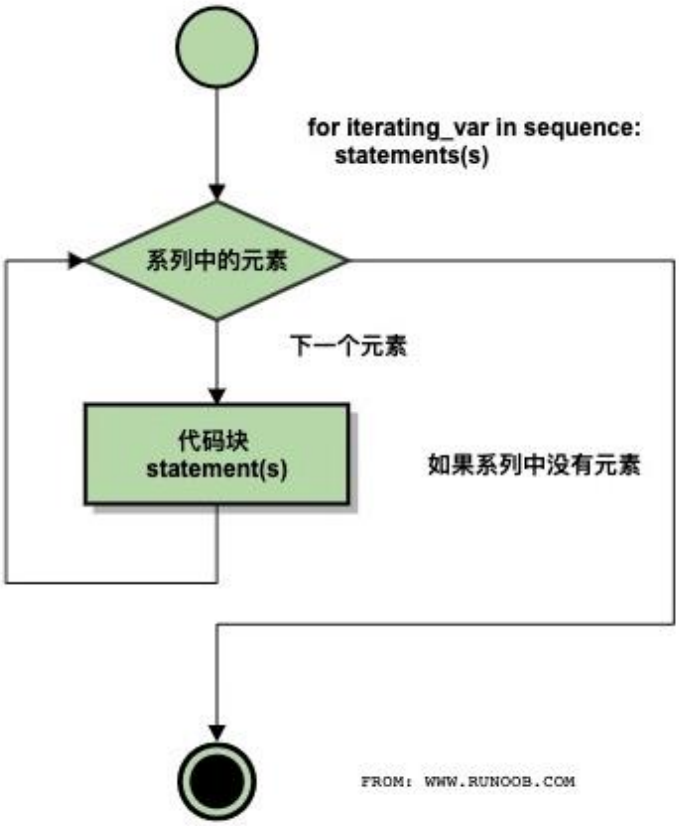
Python for 循环可以遍历任何可迭代对象，如一个列表或者一个字符串。

for 循环的一般格式如下：

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

```
else:
   <statements>
```

流程图：



Python for 循环实例：

实例

```
#!/usr/bin/python3

sites = ["Baidu", "Google","Runoob","Taobao"]
for site in sites:
   print(site)
```

以上代码执行输出结果为：

```
Baidu
Google
Runoob
Taobao
```

也可用于打印字符串中的每个字符：

实例

```
#!/usr/bin/python3
word = 'runoob'

for letter in word:
   print(letter)
```

以上代码执行输出结果为：

```
r
```

u
n
o
o
b

整数范围值可以配合 range() 函数使用：

实例

```
#!/usr/bin/python3

# 1 到 5 的所有数字：
for number in range(1, 6):
    print(number)
```

以上代码执行输出结果为：

1
2
3
4
5

---

for...else

在 Python 中，for...else 语句用于在循环结束后执行一段代码。

语法格式如下：

```
for item in iterable:
    # 循环主体
else:
    # 循环结束后执行的代码
```

当循环执行完毕（即遍历完 iterable 中的所有元素）后，会执行 else 子句中的代码，如果在循环过程中遇到了 break 语句，则会中断循环，此时不会执行 else 子句。

实例

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

执行脚本后，输出结果为：

0
1
2
3
4
5
Finally finished!

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

实例

```
#!/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!
```

---

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:

实例

```
>>>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：

实例

```
>>>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'):

实例

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

0
3
6
9
>>>
```

负数:

实例

```
>>>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:

实例

```
>>>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:

实例

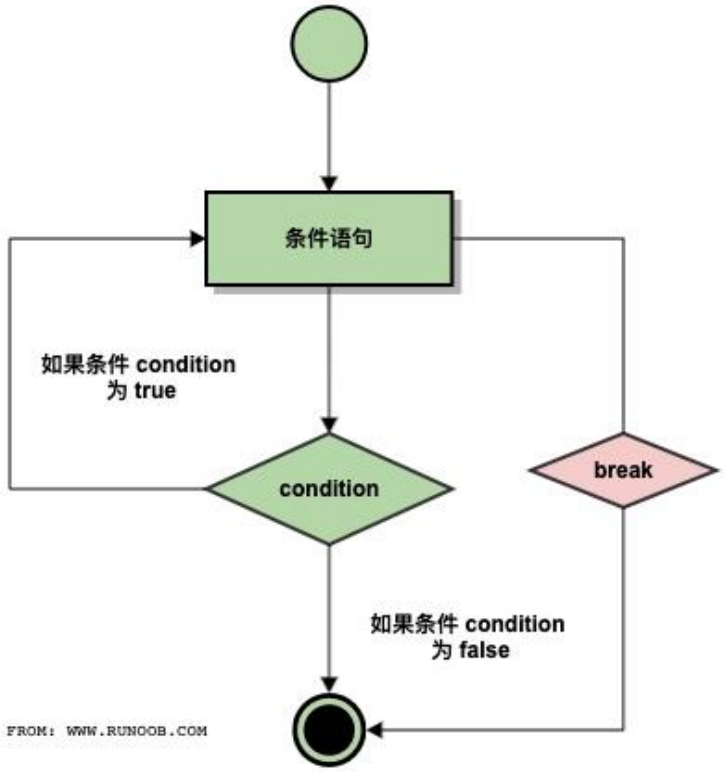
```
>>>list(range(5))
[0, 1, 2, 3, 4]
>>>
```

更多关于 range() 函数用法参考: https://www.runoob.com/python3/python3-func-range.html

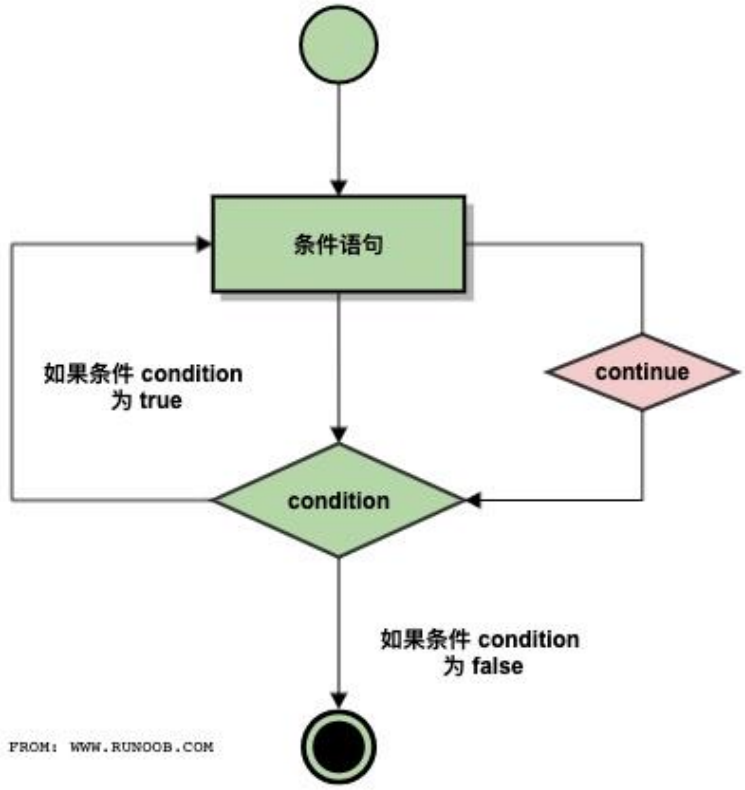
---

break 和 continue 语句及循环中的 else 子句

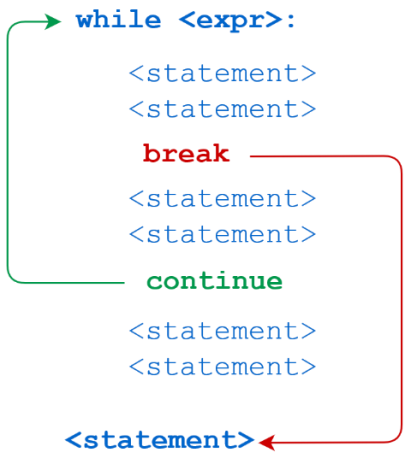
break 执行流程图:



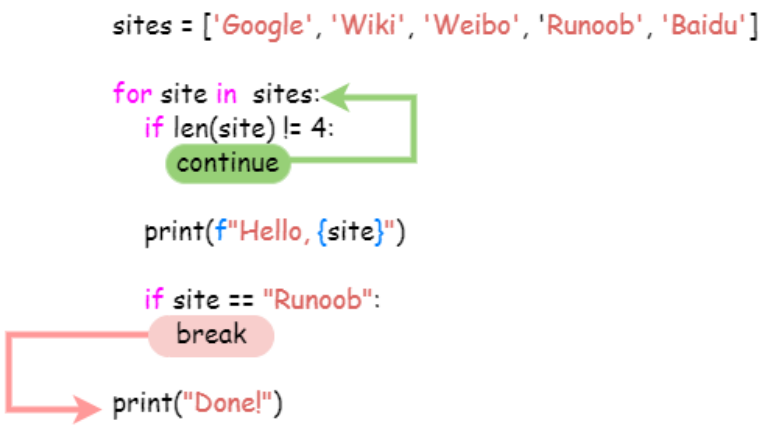
continue 执行流程图：



while 语句代码执行过程：



for 语句代码执行过程:



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。

实例

while 中使用 break:

实例

```
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:

实例

```
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:

实例

```
#!/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 ('当前变量值为 :', 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
当前变量值为 : 10
当前变量值为 : 9
当前变量值为 : 8
当前变量值为 : 7
当前变量值为 : 6
Good bye!
```

以下实例循环字符串 Runoob，碰到字母 o 跳过输出:

实例

```
#!/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:

实例

```
#!/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

---

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

实例

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

最小的类:

实例

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

以下实例在字母为 o 时 执行 pass 语句块:

实例

```
#!/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!
```