2017.10.9

Python标准语法与使用

BlogRunner团队（第八组）

前端开发工程师：郭莹婷

**Python 简介**

Python是一种解释型、面向对象、动态数据类型的高级程序设计语言。

Python由Guido van Rossum于1989年底发明，第一个公开发行版发行于1991年。像Perl语言一样, Python 源代码同样遵循 GPL(GNU General Public License)协议。

Python 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。 Python 的设计具有很强的可读性，相比其他语言经常使用英文关键字，其他语言的一些标点符号，它具有比其他语言更有特色语法结构。

* Python 是一种解释型语言： 这意味着开发过程中没有了编译这个环节。类似于PHP和Perl语言。
* Python 是交互式语言： 这意味着，您可以在一个Python提示符，直接互动执行写你的程序。
* Python 是面向对象语言: 这意味着Python支持面向对象的风格或代码封装在对象的编程技术。
* Python 是初学者的语言：Python 对初级程序员而言，是一种伟大的语言，它支持广泛的应用程序开发，从简单的文字处理到 WWW 浏览器再到游戏。

**Python 发展历史**

Python 是由 Guido van Rossum 在八十年代末和九十年代初，在荷兰国家数学和计算机科学研究所设计出来的。Python 本身也是由诸多其他语言发展而来的,这包括 ABC、Modula-3、C、C++、Algol-68、SmallTalk、Unix shell 和其他的脚本语言等等。像 Perl 语言一样，Python 源代码同样遵循 GPL(GNU General Public License)协议。现在 Python 是由一个核心开发团队在维护，Guido van Rossum 仍然占据着至关重要的作用，指导其进展。

**Python 特点**

* 易于学习：Python有相对较少的关键字，结构简单，和一个明确定义的语法，学习起来更加简单。
* 易于阅读：Python代码定义的更清晰。
* 易于维护：Python的成功在于它的源代码是相当容易维护的。
* 一个广泛的标准库：Python的最大的优势之一是丰富的库，跨平台的，在UNIX，Windows和Macintosh兼容很好。
* 互动模式：互动模式的支持，您可以从终端输入执行代码并获得结果的语言，互动的测试和调试代码片断。
* 可移植：基于其开放源代码的特性，Python已经被移植（也就是使其工作）到许多平台。
* 可扩展：如果你需要一段运行很快的关键代码，或者是想要编写一些不愿开放的算法，你可以使用C或C++完成那部分程序，然后从你的Python程序中调用。
* 数据库：Python提供所有主要的商业数据库的接口。
* GUI编程：Python支持GUI可以创建和移植到许多系统调用。
* 可嵌入: 你可以将Python嵌入到C/C++程序，让你的程序的用户获得"脚本化"的能力。

**Python 标识符**

在 Python 里，标识符由字母、数字、下划线组成。

在 Python 中，所有标识符可以包括英文、数字以及下划线(\_)，但不能以数字开头。

Python 中的标识符是区分大小写的。

以下划线开头的标识符是有特殊意义的。以单下划线开头 \_foo 的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用 from xxx import \* 而导入；以双下划线开头的 \_\_foo 代表类的私有成员；以双下划线开头和结尾的 \_\_foo\_\_ 代表 Python 里特殊方法专用的标识，如 \_\_init\_\_() 代表类的构造函数。

Python 可以同一行显示多条语句，方法是用分号 ; 分开，如：

>>> print 'hello';print 'runoob';

hello

runoob

**Python 保留字符**

下面的列表显示了在Python中的保留字。这些保留字不能用作常数或变数，或任何其他标识符名称。

所有 Python 的关键字只包含小写字母。

|  |  |  |
| --- | --- | --- |
| and | exec | not |
| assert | finally | or |
| break | for | pass |
| class | from | print |
| continue | global | raise |
| def | if | return |
| del | import | try |
| elif | in | while |
| else | is | with |
| except | lambda | yield |

**行和缩进**

学习 Python 与其他语言最大的区别就是，Python 的代码块不使用大括号 {} 来控制类，函数以及其他逻辑判断。python 最具特色的就是用缩进来写模块。缩进的空白数量是可变的，但是所有代码块语句必须包含相同的缩进空白数量，这个必须严格执行。

因此，在 Python 的代码块中必须使用相同数目的行首缩进空格数。

建议你在每个缩进层次使用 单个制表符 或 两个空格 或 四个空格 , 切记不能混用

**多行语句**

Python语句中一般以新行作为为语句的结束符。

但是我们可以使用斜杠（ \）将一行的语句分为多行显示。

语句中包含 [], {} 或 () 括号就不需要使用多行连接符。

**Python 引号**

Python 可以使用引号( ' )、双引号( " )、三引号( ''' 或 """ ) 来表示字符串，引号的开始与结束必须的相同类型的。

其中三引号可以由多行组成，编写多行文本的快捷语法，常用于文档字符串，在文件的特定地点，被当做注释。

word = 'word'

sentence = "这是一个句子。"

paragraph = """这是一个段落。

包含了多个语句"""

**Python注释**

python中单行注释采用 # 开头。

注释可以在语句或表达式行末：

python 中多行注释使用三个单引号(''')或三个双引号(""")。

**Python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。

**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逗号：

**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)。

**运算符**

Python语言支持以下类型的运算符:

* [算术运算符](http://www.runoob.com/python/python-operators.html#ysf1)
* [比较（关系）运算符](http://www.runoob.com/python/python-operators.html#ysf2)
* [赋值运算符](http://www.runoob.com/python/python-operators.html#ysf3)
* [逻辑运算符](http://www.runoob.com/python/python-operators.html#ysf4)
* [位运算符](http://www.runoob.com/python/python-operators.html#ysf5)
* [成员运算符](http://www.runoob.com/python/python-operators.html#ysf6)
* [身份运算符](http://www.runoob.com/python/python-operators.html#ysf7)
* [运算符优先级](http://www.runoob.com/python/python-operators.html#ysf8)

**Python算术运算符**

以下假设变量： a=10，b=20：

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| + | 加 - 两个对象相加 | a + b 输出结果 30 |
| - | 减 - 得到负数或是一个数减去另一个数 | a - b 输出结果 -10 |
| \* | 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 200 |
| / | 除 - x除以y | b / a 输出结果 2 |
| % | 取模 - 返回除法的余数 | b % a 输出结果 0 |
| \*\* | 幂 - 返回x的y次幂 | a\*\*b 为10的20次方， 输出结果 100000000000000000000 |
| // | 取整除 - 返回商的整数部分 | 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0 |

**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. |
| <> | 不等于 - 比较两个对象是否不相等 | (a <> b) 返回 true。这个运算符类似 != 。 |
| > | 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。 |

**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 |

**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：

|  |  |  |
| --- | --- | --- |
| 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 |

**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 是非 0，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 |

**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。 |

**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()](http://www.runoob.com/python/python-func-id.html) 函数用于获取对象内存地址。

**Python运算符优先级**

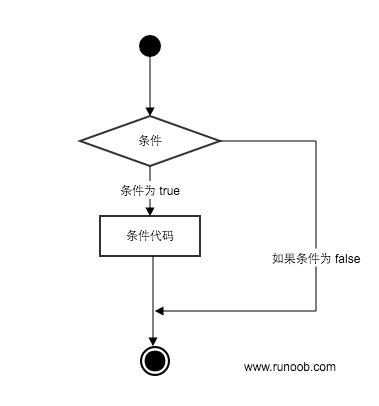
以下表格列出了从最高到最低优先级的所有运算符：

|  |  |
| --- | --- |
| 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，取模和取整除 |
| + - | 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 or and | 逻辑运算符 |

**Python条件语句**

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:



Python程序语言指定任何非0和非空（null）值为true，0 或者 null为false。

Python 编程中 if 语句用于控制程序的执行，基本形式为：

if 判断条件：

执行语句……

else：

执行语句……

其中"判断条件"成立时（非零），则执行后面的语句，而执行内容可以多行，以缩进来区分表示同一范围。

else 为可选语句，当需要在条件不成立时执行内容则可以执行相关语句，具体例子如下：

if 语句的判断条件可以用>（大于）、<(小于)、==（等于）、>=（大于等于）、<=（小于等于）来表示其关系。

当判断条件为多个值时，可以使用以下形式：

if 判断条件1:

执行语句1……

elif 判断条件2:

执行语句2……

elif 判断条件3:

执行语句3……

else:

执行语句4……

由于 python 并不支持 switch 语句，所以多个条件判断，只能用 elif 来实现，如果判断需要多个条件需同时判断时，可以使用 or （或），表示两个条件有一个成立时判断条件成功；使用 and （与）时，表示只有两个条件同时成立的情况下，判断条件才成功。

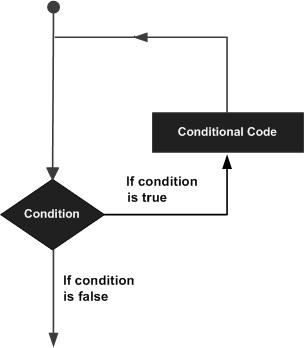
当if有多个条件时可使用括号来区分判断的先后顺序，括号中的判断优先执行，此外 and 和 or 的优先级低于>（大于）、<（小于）等判断符号，即大于和小于在没有括号的情况下会比与或要优先判断。

**Python循环语句**

Python的循环语句，程序在一般情况下是按顺序执行的。

编程语言提供了各种控制结构，允许更复杂的执行路径。

循环语句允许我们执行一个语句或语句组多次，下面是在大多数编程语言中的循环语句的一般形式：



Python提供了for循环和while循环（在Python中没有do..while循环）:

|  |  |
| --- | --- |
| 循环类型 | 描述 |
| [while 循环](http://www.runoob.com/python/python-while-loop.html) | 在给定的判断条件为 true 时执行循环体，否则退出循环体。 |
| [for 循环](http://www.runoob.com/python/python-for-loop.html) | 重复执行语句 |
| [嵌套循环](http://www.runoob.com/python/python-nested-loops.html) | 你可以在while循环体中嵌套for循环 |

**循环控制语句**

循环控制语句可以更改语句执行的顺序。Python支持以下循环控制语句：

|  |  |
| --- | --- |
| 控制语句 | 描述 |
| [break 语句](http://www.runoob.com/python/python-break-statement.html) | 在语句块执行过程中终止循环，并且跳出整个循环 |
| [continue 语句](http://www.runoob.com/python/python-continue-statement.html) | 在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。 |
| [pass 语句](http://www.runoob.com/python/python-pass-statement.html) | pass是空语句，是为了保持程序结构的完整性。 |

Python While 循环语句

Python 编程中 while 语句用于循环执行程序，即在某条件下，循环执行某段程序，以处理需要重复处理的相同任务。其基本形式为：

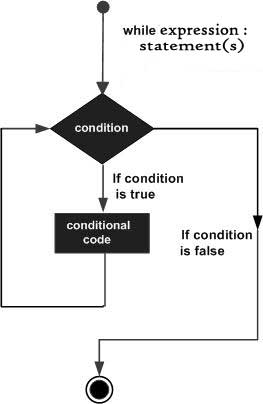
while 判断条件：

执行语句……

执行语句可以是单个语句或语句块。判断条件可以是任何表达式，任何非零、或非空（null）的值均为true。

当判断条件假false时，循环结束。

执行流程图如下：



while 语句时还有另外两个重要的命令 continue，break 来跳过循环，continue 用于跳过该次循环，break 则是用于退出循环，此外"判断条件"还可以是个常值，表示循环必定成立。

**无限循环**

如果条件判断语句永远为 true，循环将会无限的执行下去。

**循环使用 else 语句**

在 python 中，while … else 在循环条件为 false 时执行 else 语句块。

**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。

注意：以上的无限循环你可以使用 CTRL+C 来中断循环。

Python for 循环语句

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。

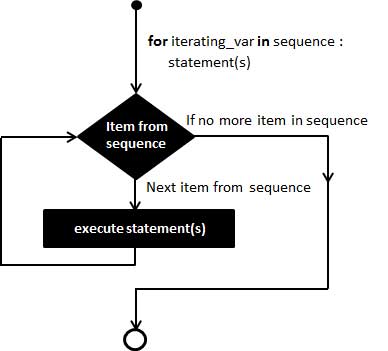
语法：

for循环的语法格式如下：

for iterating\_var in sequence:

statements(s)

**流程图：**



**通过序列索引迭代**

另外一种执行循环的遍历方式是通过索引。内置函数 len() 和 range(),函数 len() 返回列表的长度，即元素的个数。 range返回一个序列的数。

**循环使用 else 语句**

在 python 中，for … else 表示这样的意思，for 中的语句和普通的没有区别，else 中的语句会在循环正常执行完（即 for 不是通过 break 跳出而中断的）的情况下执行，while … else 也是一样。

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。

Python 语言 pass 语句语法格式如下：

pass

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。

**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

def functionname( parameters ):

"函数\_文档字符串"

function\_suite

return [expression]

默认情况下，参数值和参数名称是按函数声明中定义的的顺序匹配起来的。

**函数调用**

定义一个函数只给了函数一个名称，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提示符执行。

**参数传递**

在 python 中，类型属于对象，变量是没有类型的。

**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。

**参数**

以下是调用函数时可使用的正式参数类型：

* 必备参数
* 关键字参数
* 默认参数
* 不定长参数

**必备参数**

必备参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

调用printme()函数，你必须传入一个参数，不然会出现语法错误。

**关键字参数**

关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。

使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值

**缺省参数**

调用函数时，缺省参数的值如果没有传入，则被认为是默认值。下例会打印默认的age，如果age没有被传入。

**不定长参数**

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述2种参数不同，声明时不会命名。基本语法如下：

def functionname([formal\_args,] \*var\_args\_tuple ):

"函数\_文档字符串"

function\_suite

return [expression]

加了星号（\*）的变量名会存放所有未命名的变量参数。选择不多传参数也可。

**匿名函数**

python 使用 lambda 来创建匿名函数。

* lambda只是一个表达式，函数体比def简单很多。
* lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
* lambda函数拥有自己的命名空间，且不能访问自有参数列表之外或全局命名空间里的参数。
* 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

**return 语句**

return语句[表达式]退出函数，选择性地向调用方返回一个表达式。不带参数值的return语句返回None。

**变量作用域**

一个程序的所有的变量并不是在哪个位置都可以访问的。访问权限决定于这个变量是在哪里赋值的。

变量的作用域决定了在哪一部分程序你可以访问哪个特定的变量名称。两种最基本的变量作用域如下：

* 全局变量
* 局部变量

**全局变量和局部变量**

定义在函数内部的变量拥有一个局部作用域，定义在函数外的拥有全局作用域。

局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内声明的变量名称都将被加入到作用域中。

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。

**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。

**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。

**删除列表元素**

可以使用 del 语句来删除列表的的元素。

**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, | 1 2 3 | 迭代 |

**Python列表截取**

描述：

|  |  |  |
| --- | --- | --- |
| Python 表达式 | 结果 | 描述 |
| L[2] | 'Taobao' | 读取列表中第三个元素 |
| L[-2] | 'Runoob' | 读取列表中倒数第二个元素 |
| L[1:] | ['Runoob', 'Taobao'] | 从第二个元素开始截取列表 |

**Python列表函数&方法**

Python包含以下函数:

|  |  |
| --- | --- |
| 序号 | 函数 |
| 1 | [cmp(list1, list2)](http://www.runoob.com/python/att-list-cmp.html) 比较两个列表的元素 |
| 2 | [len(list)](http://www.runoob.com/python/att-list-len.html) 列表元素个数 |
| 3 | [max(list)](http://www.runoob.com/python/att-list-max.html) 返回列表元素最大值 |
| 4 | [min(list)](http://www.runoob.com/python/att-list-min.html) 返回列表元素最小值 |
| 5 | [list(seq)](http://www.runoob.com/python/att-list-list.html) 将元组转换为列表 |

Python包含以下方法:

|  |  |
| --- | --- |
| 序号 | 方法 |
| 1 | [list.append(obj)](http://www.runoob.com/python/att-list-append.html) 在列表末尾添加新的对象 |
| 2 | [list.count(obj)](http://www.runoob.com/python/att-list-count.html) 统计某个元素在列表中出现的次数 |
| 3 | [list.extend(seq)](http://www.runoob.com/python/att-list-extend.html) 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)](http://www.runoob.com/python/att-list-index.html) 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)](http://www.runoob.com/python/att-list-insert.html) 将对象插入列表 |
| 6 | [list.pop(obj=list[-1])](http://www.runoob.com/python/att-list-pop.html) 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)](http://www.runoob.com/python/att-list-remove.html) 移除列表中某个值的第一个匹配项 |
| 8 | [list.reverse()](http://www.runoob.com/python/att-list-reverse.html) 反向列表中元素 |
| 9 | [list.sort([func])](http://www.runoob.com/python/att-list-sort.html) |

**参考链接：**

**http://www.runoob.com/python/python-tutorial.html**