# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*基础教程\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

参考资料：<https://www.runoob.com/python/python-tutorial.html>

# 基础语法

1.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。

2.Python语句中一般以新行作为语句的结束符。

但是我们可以使用斜杠（ \）将一行的语句分为多行显示，如下所示：

total = item\_one + \

item\_two + \

item\_three

语句中包含 [], {} 或 () 括号就不需要使用多行连接符。如下实例：

days = ['Monday', 'Tuesday', 'Wednesday',

'Thursday', 'Friday']

3.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"""

4.Python注释

python中单行注释采用 # 开头。

python 中多行注释使用三个单引号(''')或三个双引号(""")。

5.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

6.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 **,**。（Python2）

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=""**：（Python3）

# 数据类型

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。

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）。

## Numbers（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。

***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。*

数字数据类型用于存储数值。

他们是不可改变的数据类型，这意味着改变数字数据类型会分配一个新的对象。

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支持四种不同的数字类型：

* int（有符号整型）
* long（长整型[也可以代表八进制和十六进制]）
* float（浮点型）
* complex（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。

***注意：****long 类型只存在于 Python2.X 版本中，在 2.2 以后的版本中，int 类型数据溢出后会自动转为long类型。在 Python3.X 版本中 long 类型被移除，使用 int 替代。*

Python Number 类型转换

int(x [,base ]) 将x转换为一个整数

long(x [,base ]) 将x转换为一个长整数

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

chr(x ) 将一个整数转换为一个字符

unichr(x ) 将一个整数转换为Unicode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

Python math 模块、cmath 模块

Python 中数学运算常用的函数基本都在 math 模块、cmath 模块中。

Python math 模块提供了许多对浮点数的数学运算函数。

Python cmath 模块包含了一些用于复数运算的函数。

cmath 模块的函数跟 math 模块函数基本一致，区别是 cmath 模块运算的是复数，math 模块运算的是数学运算。

要使用 math 或 cmath 函数必须先导入：（Python3不用导入）

import math

|  |  |
| --- | --- |
| **函数** | **返回值 ( 描述 )** |
| [abs(x)](https://www.runoob.com/python3/python3-func-number-abs.html" \t "/home/zgj/文档\\x/_blank) | 返回数字的绝对值，如abs(-10) 返回 10 |
| [ceil(x)](https://www.runoob.com/python3/python3-func-number-ceil.html" \t "/home/zgj/文档\\x/_blank) | 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)](https://www.runoob.com/python3/python3-func-number-exp.html" \t "/home/zgj/文档\\x/_blank) | 返回e的x次幂(ex),如math.exp(1) 返回2.718281828459045 |
| [fabs(x)](https://www.runoob.com/python3/python3-func-number-fabs.html" \t "/home/zgj/文档\\x/_blank) | 返回数字的绝对值，如math.fabs(-10) 返回10.0 |
| [floor(x)](https://www.runoob.com/python3/python3-func-number-floor.html" \t "/home/zgj/文档\\x/_blank) | 返回数字的下舍整数，如math.floor(4.9)返回 4 |
| [log(x)](https://www.runoob.com/python3/python3-func-number-log.html" \t "/home/zgj/文档\\x/_blank) | 如math.log(math.e)返回1.0,math.log(100,10)返回2.0 |
| [log10(x)](https://www.runoob.com/python3/python3-func-number-log10.html" \t "/home/zgj/文档\\x/_blank) | 返回以10为基数的x的对数，如math.log10(100)返回 2.0 |
| [max(x1, x2,...)](https://www.runoob.com/python3/python3-func-number-max.html" \t "/home/zgj/文档\\x/_blank) | 返回给定参数的最大值，参数可以为序列。 |
| [min(x1, x2,...)](https://www.runoob.com/python3/python3-func-number-min.html" \t "/home/zgj/文档\\x/_blank) | 返回给定参数的最小值，参数可以为序列。 |
| [modf(x)](https://www.runoob.com/python3/python3-func-number-modf.html" \t "/home/zgj/文档\\x/_blank) | 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)](https://www.runoob.com/python3/python3-func-number-pow.html" \t "/home/zgj/文档\\x/_blank) | x\*\*y 运算后的值。 |
| [round(x [,n])](https://www.runoob.com/python3/python3-func-number-round.html" \t "/home/zgj/文档\\x/_blank) | 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)](https://www.runoob.com/python3/python3-func-number-sqrt.html" \t "/home/zgj/文档\\x/_blank) | 返回数字x的平方根。 |

Python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)](https://www.runoob.com/python/func-number-choice.html" \t "/home/zgj/文档\\x/_blank) | 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])](https://www.runoob.com/python/func-number-randrange.html" \t "/home/zgj/文档\\x/_blank) | 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()](https://www.runoob.com/python/func-number-random.html" \t "/home/zgj/文档\\x/_blank) | 随机生成下一个实数，它在[0,1)范围内。 |
| [seed([x])](https://www.runoob.com/python/func-number-seed.html" \t "/home/zgj/文档\\x/_blank) | 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)](https://www.runoob.com/python/func-number-shuffle.html" \t "/home/zgj/文档\\x/_blank) | 将序列的所有元素随机排序 |
| [uniform(x, y)](https://www.runoob.com/python/func-number-uniform.html" \t "/home/zgj/文档\\x/_blank) | 随机生成下一个实数，它在[x,y]范围内。 |

Python三角函数

Python包括以下三角函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [acos(x)](https://www.runoob.com/python/func-number-acos.html" \t "/home/zgj/文档\\x/_blank) | 返回x的反余弦弧度值。 |
| [asin(x)](https://www.runoob.com/python/func-number-asin.html" \t "/home/zgj/文档\\x/_blank) | 返回x的反正弦弧度值。 |
| [atan(x)](https://www.runoob.com/python/func-number-atan.html" \t "/home/zgj/文档\\x/_blank) | 返回x的反正切弧度值。 |
| [atan2(y, x)](https://www.runoob.com/python/func-number-atan2.html" \t "/home/zgj/文档\\x/_blank) | 返回给定的 X 及 Y 坐标值的反正切值。 |
| [cos(x)](https://www.runoob.com/python/func-number-cos.html" \t "/home/zgj/文档\\x/_blank) | 返回x的弧度的余弦值。 |
| [hypot(x, y)](https://www.runoob.com/python/func-number-hypot.html" \t "/home/zgj/文档\\x/_blank) | 返回欧几里德范数 sqrt(x\*x + y\*y)。 |
| [sin(x)](https://www.runoob.com/python/func-number-sin.html" \t "/home/zgj/文档\\x/_blank) | 返回的x弧度的正弦值。 |
| [tan(x)](https://www.runoob.com/python/func-number-tan.html" \t "/home/zgj/文档\\x/_blank) | 返回x弧度的正切值。 |
| [degrees(x)](https://www.runoob.com/python/func-number-degrees.html" \t "/home/zgj/文档\\x/_blank) | 将弧度转换为角度,如degrees(math.pi/2) ， 返回90.0 |
| [radians(x)](https://www.runoob.com/python/func-number-radians.html" \t "/home/zgj/文档\\x/_blank) | 将角度转换为弧度 |

Python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）。 |

## String（字符串）

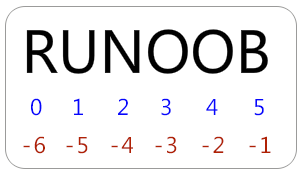
Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。

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。

Python 访问子字符串，可以使用方括号来截取字符串，

python的字串列表有2种取值顺序:

* 从左到右索引默认0开始的，最大范围是字符串长度少1
* 从右到左索引默认-1开始的，最大范围是字符串开头



如果你要实现从字符串中获取一段子字符串的话，可以使用 **[头下标:尾下标]** 来截取相应的字符串，其中下标是从 0 开始算起，可以是正数或负数，下标可以为空表示取到头或尾。

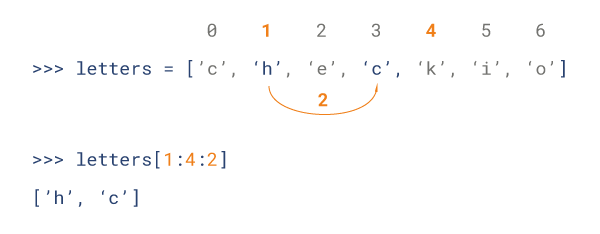
**[头下标:尾下标]** 获取的子字符串包含头下标的字符，但不包含尾下标的字符。

加号（+）是字符串连接运算符，星号（\*）是重复操作。

print str \* 2 # 输出字符串两次

print str + "TEST" # 输出连接的字符串

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：



Python字符串运算符

|  |  |  |
| --- | --- | --- |
| 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' \n >>> print R'\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。

如下实例（Python2）：

#!/usr/bin/python

print "My name is %s and weight is %d kg!" % ('Zara', 21)

以上实例输出结果：

My name is Zara and weight is 21 kg!

如下实例（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

|  |  |
| --- | --- |
| **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) |

### 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()](https://www.runoob.com/python3/python3-string-capitalize.html) 将字符串的第一个字符转换为大写 |
| 2 | [center(width, fillchar)](https://www.runoob.com/python3/python3-string-center.html)  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))](https://www.runoob.com/python3/python3-string-count.html)  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")](https://www.runoob.com/python3/python3-string-decode.html)  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')](https://www.runoob.com/python3/python3-string-encode.html)  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))](https://www.runoob.com/python3/python3-string-endswith.html) 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)](https://www.runoob.com/python3/python3-string-expandtabs.html)  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))](https://www.runoob.com/python3/python3-string-find.html)  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))](https://www.runoob.com/python3/python3-string-index.html)  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()](https://www.runoob.com/python3/python3-string-isalnum.html)  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()](https://www.runoob.com/python3/python3-string-isalpha.html)  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()](https://www.runoob.com/python3/python3-string-isdigit.html)  如果字符串只包含数字则返回 True 否则返回 False.. |
| 13 | [islower()](https://www.runoob.com/python3/python3-string-islower.html)  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()](https://www.runoob.com/python3/python3-string-isnumeric.html)  如果字符串中只包含数字字符，则返回 True，否则返回 False |
| 15 | [isspace()](https://www.runoob.com/python3/python3-string-isspace.html)  如果字符串中只包含空白，则返回 True，否则返回 False. |
| 16 | [istitle()](https://www.runoob.com/python3/python3-string-istitle.html)  如果字符串是标题化的(见 title())则返回 True，否则返回 False |
| 17 | [isupper()](https://www.runoob.com/python3/python3-string-isupper.html)  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)](https://www.runoob.com/python3/python3-string-join.html)  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)](https://www.runoob.com/python3/python3-string-len.html)  返回字符串长度 |
| 20 | [ljust(width[, fillchar])](https://www.runoob.com/python3/python3-string-ljust.html)  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()](https://www.runoob.com/python3/python3-string-lower.html)  转换字符串中所有大写字符为小写. |
| 22 | [lstrip()](https://www.runoob.com/python3/python3-string-lstrip.html)  截掉字符串左边的空格或指定字符。 |
| 23 | [maketrans()](https://www.runoob.com/python3/python3-string-maketrans.html)  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)](https://www.runoob.com/python3/python3-string-max.html)  返回字符串 str 中最大的字母。 |
| 25 | [min(str)](https://www.runoob.com/python3/python3-string-min.html)  返回字符串 str 中最小的字母。 |
| 26 | [replace(old, new [, max])](https://www.runoob.com/python3/python3-string-replace.html)  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))](https://www.runoob.com/python3/python3-string-rfind.html)  类似于 find()函数，不过是从右边开始查找. |
| 28 | [rindex( str, beg=0, end=len(string))](https://www.runoob.com/python3/python3-string-rindex.html)  类似于 index()，不过是从右边开始. |
| 29 | [rjust(width,[, fillchar])](https://www.runoob.com/python3/python3-string-rjust.html)  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()](https://www.runoob.com/python3/python3-string-rstrip.html)  删除字符串字符串末尾的空格. |
| 31 | [split(str="", num=string.count(str))](https://www.runoob.com/python3/python3-string-split.html)  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])](https://www.runoob.com/python3/python3-string-splitlines.html)  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))](https://www.runoob.com/python3/python3-string-startswith.html)  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])](https://www.runoob.com/python3/python3-string-strip.html)  在字符串上执行 lstrip()和 rstrip() |
| 35 | [swapcase()](https://www.runoob.com/python3/python3-string-swapcase.html)  将字符串中大写转换为小写，小写转换为大写 |
| 36 | [title()](https://www.runoob.com/python3/python3-string-title.html)  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="")](https://www.runoob.com/python3/python3-string-translate.html)  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()](https://www.runoob.com/python3/python3-string-upper.html)  转换字符串中的小写字母为大写 |
| 39 | [zfill (width)](https://www.runoob.com/python3/python3-string-zfill.html)  返回长度为 width 的字符串，原字符串右对齐，前面填充0 |
| 40 | [isdecimal()](https://www.runoob.com/python3/python3-string-isdecimal.html)  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。 |

## 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。它支持字符，数字，字符串甚至可以包含列表（即嵌套）。

列表用 **[ ]** 标识，是 python 最通用的复合数据类型。

列表中值的切割也可以用到变量 **[头下标:尾下标]** ，就可以截取相应的列表，从左到右索引默认 0 开始，从右到左索引默认 -1 开始，下标可以为空表示取到头或尾。

加号 **+** 是列表连接运算符，星号 **\*** 是重复操作。

Python列表函数&方法

Python包含以下函数:

|  |  |
| --- | --- |
| **序号** | **函数** |
| 1 | [len(list)](https://www.runoob.com/python3/python3-att-list-len.html) 列表元素个数 |
| 2 | [max(list)](https://www.runoob.com/python3/python3-att-list-max.html) 返回列表元素最大值 |
| 3 | [min(list)](https://www.runoob.com/python3/python3-att-list-min.html) 返回列表元素最小值 |
| 4 | [list(seq)](https://www.runoob.com/python3/python3-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)](https://www.runoob.com/python3/python3-att-list-append.html) 在列表末尾添加新的对象 |
| 2 | [list.count(obj)](https://www.runoob.com/python3/python3-att-list-count.html) 统计某个元素在列表中出现的次数 |
| 3 | [list.extend(seq)](https://www.runoob.com/python3/python3-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)](https://www.runoob.com/python3/python3-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)](https://www.runoob.com/python3/python3-att-list-insert.html) 将对象插入列表 |
| 6 | [list.pop([index=-1])](https://www.runoob.com/python3/python3-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)](https://www.runoob.com/python3/python3-att-list-remove.html) 移除列表中某个值的第一个匹配项 |
| 8 | [list.reverse()](https://www.runoob.com/python3/python3-att-list-reverse.html) 反向列表中元素 |
| 9 | [list.sort( key=None, reverse=False)](https://www.runoob.com/python3/python3-att-list-sort.html) 对原列表进行排序 |
| 10 | [list.clear()](https://www.runoob.com/python3/python3-att-list-clear.html) 清空列表 |
| 11 | [list.copy()](https://www.runoob.com/python3/python3-att-list-copy.html) 复制列表 |

## Tuple（元组）

元组是另一个数据类型，类似于 List（列表）。

元组用 **()** 标识。内部元素用逗号隔开。但是元组不能二次赋值，相当于只读列表。

加号 **+** 是元组连接运算符，星号 **\*** 是重复操作

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

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') |

## Dictionary（字典）

字典(dictionary)是除列表以外python之中最灵活的内置数据结构类型。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。

字典用"{ }"标识。字典由索引(key)和它对应的值value组成。

**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 **{ }**。

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()](https://www.runoob.com/python3/python3-att-dictionary-clear.html) 删除字典内所有元素 |
| 2 | [radiansdict.copy()](https://www.runoob.com/python3/python3-att-dictionary-copy.html) 返回一个字典的浅复制 |
| 3 | [radiansdict.fromkeys()](https://www.runoob.com/python3/python3-att-dictionary-fromkeys.html) 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)](https://www.runoob.com/python3/python3-att-dictionary-get.html) 返回指定键的值，如果值不在字典中返回default值 |
| 5 | [key in dict](https://www.runoob.com/python3/python3-att-dictionary-in.html) 如果键在字典dict里返回true，否则返回false |
| 6 | [radiansdict.items()](https://www.runoob.com/python3/python3-att-dictionary-items.html) 以列表返回可遍历的(键, 值) 元组数组 |
| 7 | [radiansdict.keys()](https://www.runoob.com/python3/python3-att-dictionary-keys.html) 返回一个迭代器，可以使用 list() 来转换为列表 |
| 8 | [radiansdict.setdefault(key, default=None)](https://www.runoob.com/python3/python3-att-dictionary-setdefault.html) 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)](https://www.runoob.com/python3/python3-att-dictionary-update.html) 把字典dict2的键/值对更新到dict里 |
| 10 | [radiansdict.values()](https://www.runoob.com/python3/python3-att-dictionary-values.html) 返回一个迭代器，可以使用 list() 来转换为列表 |
| 11 | [pop(key[,default])](https://www.runoob.com/python3/python3-att-dictionary-pop.html) 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()](https://www.runoob.com/python3/python3-att-dictionary-popitem.html) 随机返回并删除字典中的最后一对键和值。 |

## 6.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)

集合内置方法完整列表

|  |  |
| --- | --- |
| **方法** | **描述** |
| [add()](https://www.runoob.com/python3/ref-set-add.html" \t "/home/zgj/文档\\x/_blank) | 为集合添加元素 |
| [clear()](https://www.runoob.com/python3/ref-set-clear.html" \t "/home/zgj/文档\\x/_blank) | 移除集合中的所有元素 |
| [copy()](https://www.runoob.com/python3/ref-set-copy.html" \t "/home/zgj/文档\\x/_blank) | 拷贝一个集合 |
| [difference()](https://www.runoob.com/python3/ref-set-difference.html" \t "/home/zgj/文档\\x/_blank) | 返回多个集合的差集 |
| [difference\_update()](https://www.runoob.com/python3/ref-set-difference_update.html" \t "/home/zgj/文档\\x/_blank) | 移除集合中的元素，该元素在指定的集合也存在。 |
| [discard()](https://www.runoob.com/python3/ref-set-discard.html" \t "/home/zgj/文档\\x/_blank) | 删除集合中指定的元素 |
| [intersection()](https://www.runoob.com/python3/ref-set-intersection.html" \t "/home/zgj/文档\\x/_blank) | 返回集合的交集 |
| [intersection\_update()](https://www.runoob.com/python3/ref-set-intersection_update.html" \t "/home/zgj/文档\\x/_blank) | 返回集合的交集。 |
| [isdisjoint()](https://www.runoob.com/python3/ref-set-isdisjoint.html" \t "/home/zgj/文档\\x/_blank) | 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()](https://www.runoob.com/python3/ref-set-issubset.html" \t "/home/zgj/文档\\x/_blank) | 判断指定集合是否为该方法参数集合的子集。 |
| [issuperset()](https://www.runoob.com/python3/ref-set-issuperset.html" \t "/home/zgj/文档\\x/_blank) | 判断该方法的参数集合是否为指定集合的子集 |
| [pop()](https://www.runoob.com/python3/ref-set-pop.html" \t "/home/zgj/文档\\x/_blank) | 随机移除元素 |
| [remove()](https://www.runoob.com/python3/ref-set-remove.html" \t "/home/zgj/文档\\x/_blank) | 移除指定元素 |
| [symmetric\_difference()](https://www.runoob.com/python3/ref-set-symmetric_difference.html" \t "/home/zgj/文档\\x/_blank) | 返回两个集合中不重复的元素集合。 |
| [symmetric\_difference\_update()](https://www.runoob.com/python3/ref-set-symmetric_difference_update.html" \t "/home/zgj/文档\\x/_blank) | 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()](https://www.runoob.com/python3/ref-set-union.html" \t "/home/zgj/文档\\x/_blank) | 返回两个集合的并集 |
| [update()](https://www.runoob.com/python3/ref-set-update.html" \t "/home/zgj/文档\\x/_blank) | 给集合添加元素 |

## 7.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])](https://www.runoob.com/python/python-func-int.html" \t "/home/zgj/文档\\x/_blank) | 将x转换为一个整数 |
| [long(x [,base] )](https://www.runoob.com/python/python-func-long.html" \t "/home/zgj/文档\\x/_blank) | 将x转换为一个长整数 |
| [float(x)](https://www.runoob.com/python/python-func-float.html" \t "/home/zgj/文档\\x/_blank) | 将x转换到一个浮点数 |
| [complex(real [,imag])](https://www.runoob.com/python/python-func-complex.html" \t "/home/zgj/文档\\x/_blank) | 创建一个复数 |
| [str(x)](https://www.runoob.com/python/python-func-str.html" \t "/home/zgj/文档\\x/_blank) | 将对象 x 转换为字符串 |
| [repr(x)](https://www.runoob.com/python/python-func-repr.html" \t "/home/zgj/文档\\x/_blank) | 将对象 x 转换为表达式字符串 |
| [eval(str)](https://www.runoob.com/python/python-func-eval.html" \t "/home/zgj/文档\\x/_blank) | 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)](https://www.runoob.com/python/att-tuple-tuple.html" \t "/home/zgj/文档\\x/_blank) | 将序列 s 转换为一个元组 |
| [list(s)](https://www.runoob.com/python/att-list-list.html" \t "/home/zgj/文档\\x/_blank) | 将序列 s 转换为一个列表 |
| [set(s)](https://www.runoob.com/python/python-func-set.html" \t "/home/zgj/文档\\x/_blank) | 转换为可变集合 |
| [dict(d)](https://www.runoob.com/python/python-func-dict.html" \t "/home/zgj/文档\\x/_blank) | 创建一个字典。d 必须是一个序列 (key,value)元组。 |
| [frozenset(s)](https://www.runoob.com/python/python-func-frozenset.html" \t "/home/zgj/文档\\x/_blank) | 转换为不可变集合 |
| [chr(x)](https://www.runoob.com/python/python-func-chr.html" \t "/home/zgj/文档\\x/_blank) | 将一个整数转换为一个字符 |
| [unichr(x)](https://www.runoob.com/python/python-func-unichr.html" \t "/home/zgj/文档\\x/_blank) | 将一个整数转换为Unicode字符 |
| [ord(x)](https://www.runoob.com/python/python-func-ord.html" \t "/home/zgj/文档\\x/_blank) | 将一个字符转换为它的整数值 |
| [hex(x)](https://www.runoob.com/python/python-func-hex.html" \t "/home/zgj/文档\\x/_blank) | 将一个整数转换为一个十六进制字符串 |
| [oct(x)](https://www.runoob.com/python/python-func-oct.html" \t "/home/zgj/文档\\x/_blank) | 将一个整数转换为一个八进制字符串 |

# 3.Python 运算符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \*\* | 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（**向下取整**） | | >>> 9//24>>> -9//2-5 | |
| \*\*= | 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 | |
| **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 |

|  |  |  |
| --- | --- | --- |
| **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。 |

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()](https://www.runoob.com/python/python-func-id.html" \t "/home/zgj/文档\\x/_blank) 函数用于获取对象内存地址。

# 4.结构语句

## 1.if语句

基本形式为：

if 判断条件：

执行语句……else：

执行语句……

if name == 'python': # 判断变量是否为 python

flag = True # 条件成立时设置标志为真

print 'welcome boss' # 并输出欢迎信息

else: print name # 条件不成立时输出变量名称

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……

也可以在同一行的位置上使用if条件判断语句，如下实例：

if ( var == 100 ) : print "变量 var 的值为100"

**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。

## 2.while语句

while 判断条件：

执行语句……

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

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：

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：

while (flag): print 'Given flag is really true!'

在 Python 中没有 do..while 循环。

## 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。

**语法：**

for循环的语法格式如下：

for iterating\_var in sequence:

statements(s)

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。

## 4.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。

Python 语言 pass 语句语法格式如下：

pass

## 5.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

# 5.Python3 迭代器与生成器

## 1.迭代器

迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。

迭代器有两个基本的方法：**iter()** 和 **next()**。

字符串，列表或元组对象都可用于创建迭代器：

实例(Python 3.0+)

>>> list=[1,2,3,4]  
>>> it = iter(list)    # 创建迭代器对象  
>>> **print** (next(it))   # 输出迭代器的下一个元素  
1  
>>> **print** (next(it))  
2  
>>>

迭代器对象可以使用常规for语句进行遍历：

实例(Python 3.0+)

#!/usr/bin/python3

list=[1,2,3,4]

it = iter(list) # 创建迭代器对象

for x in it:

print (x, end=" ")

执行以上程序，输出结果如下：

1 2 3 4

也可以使用 next() 函数：

实例(Python 3.0+)

#!/usr/bin/python3

import sys # 引入 sys 模块

list=[1,2,3,4]

it = iter(list) # 创建迭代器对象

while True:

try:

print (next(it))

except StopIteration:

sys.exit()

执行以上程序，输出结果如下：

1234

创建一个迭代器

把一个类作为一个迭代器使用需要在类中实现两个方法 \_\_iter\_\_() 与 \_\_next\_\_() 。

如果你已经了解的面向对象编程，就知道类都有一个构造函数，Python 的构造函数为 \_\_init\_\_(), 它会在对象初始化的时候执行。

更多内容查阅：[Python3 面向对象](https://www.runoob.com/python3/python3-class.html" \t "/home/zgj/文档\\x/_blank)

\_\_iter\_\_() 方法返回一个特殊的迭代器对象， 这个迭代器对象实现了 \_\_next\_\_() 方法并通过 StopIteration 异常标识迭代的完成。

\_\_next\_\_() 方法（Python 2 里是 next()）会返回下一个迭代器对象。

创建一个返回数字的迭代器，初始值为 1，逐步递增 1：

实例(Python 3.0+)

class MyNumbers:

def \_\_iter\_\_(self):

self.a = 1

return self

def \_\_next\_\_(self):

x = self.a

self.a += 1

return x

myclass = MyNumbers()

myiter = iter(myclass)

print(next(myiter))

print(next(myiter))

print(next(myiter))

print(next(myiter))

print(next(myiter))

执行输出结果为：

12345

StopIteration

StopIteration 异常用于标识迭代的完成，防止出现无限循环的情况，在 \_\_next\_\_() 方法中我们可以设置在完成指定循环次数后触发 StopIteration 异常来结束迭代。

在 20 次迭代后停止执行：

实例(Python 3.0+)

class MyNumbers:

def \_\_iter\_\_(self):

self.a = 1

return self

def \_\_next\_\_(self):

if self.a <= 20:

x = self.a

self.a += 1

return x

else:

raise StopIteration

myclass = MyNumbers()

myiter = iter(myclass)

for x in myiter:

print(x)

执行输出结果为：

1234567891011121314151617181920

## 2.生成器

在 Python 中，使用了 yield 的函数被称为生成器（generator）。

跟普通函数不同的是，生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。

在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。

调用一个生成器函数，返回的是一个迭代器对象。

以下实例使用 yield 实现斐波那契数列：

实例(Python 3.0+)

#!/usr/bin/python3

import sys

def fibonacci(n): # 生成器函数 - 斐波那契

a, b, counter = 0, 1, 0

while True:

if (counter > n):

return

yield a

a, b = b, a + b

counter += 1

f = fibonacci(10) # f 是一个迭代器，由生成器返回生成

while True:

try:

print (next(f), end=" ")

except StopIteration:

sys.exit()

执行以上程序，输出结果如下：

0 1 1 2 3 5 8 13 21 34 55

# Python3 函数

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

参数

以下是调用函数时可使用的正式参数类型：

必需参数

关键字参数

实例(Python 3.0+)

#!/usr/bin/python3

#可写函数说明

def printinfo( name, age ):

"打印任何传入的字符串"

print ("名字: ", name)

print ("年龄: ", age)

return

#调用printinfo函数

printinfo( age=50, name="runoob" )

此时可以不用按照函数参数声明的顺序传递参数

默认参数

调用函数时，如果没有传递参数，则会使用默认参数。以下实例中如果没有传入 age 参数，则使用默认值：

实例(Python 3.0+)

#!/usr/bin/python3

#可写函数说明

def printinfo( name, age = 35 ):

"打印任何传入的字符串"

print ("名字: ", name)

print ("年龄: ", age)

return

#调用printinfo函数

printinfo( age=50, name="runoob" )

print ("------------------------")

printinfo( name="runoob" )

不定长参数

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 **\*** 的参数会以元组(tuple)的形式导入，存放所有未命名的变量参数。

实例(Python 3.0+)

#!/usr/bin/python3

# 可写函数说明

def printinfo( arg1, \*vartuple ):

"打印任何传入的参数"

print ("输出: ")

print (arg1)

print (vartuple)

# 调用printinfo 函数

printinfo( 70, 60, 50 )

以上实例输出结果：

输出: 70(60, 50)

如果在函数调用时没有指定参数，它就是一个空元组。

还有一种就是参数带两个星号 **\*\***基本语法如下：

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

"函数\_文档字符串"

function\_suite

return [expression]

加了两个星号 **\*\*** 的参数会以字典的形式导入。

实例(Python 3.0+)

#!/usr/bin/python3

# 可写函数说明

def printinfo( arg1, \*\*vardict ):

"打印任何传入的参数"

print ("输出: ")

print (arg1)

print (vardict)

# 调用printinfo 函数

printinfo(1, a=2,b=3)

以上实例输出结果：

输出: 1{'a': 2, 'b': 3}

声明函数时，参数中星号 **\*** 可以单独出现，例如:

def f(a,b,\*,c):

return a+b+c

如果单独出现星号 **\*** 后的参数必须用关键字传入。

>>> def f(a,b,\*,c):

... return a+b+c

... >>> f(1,2,3)

# 报错Traceback (most recent call last):

File "<stdin>", line 1, in <module>TypeError: f() takes 2 positional arguments but 3 were given

>>> f(1,2,c=3) # 正常

>>>6

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。

## 匿名函数

python 使用 lambda 来创建匿名函数。

所谓匿名，意即不再使用 def 语句这样标准的形式定义一个函数。

* 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，且不能访问自己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。

语法

lambda 函数的语法只包含一个语句，如下：

lambda [arg1 [,arg2,.....argn]]:expression

如下实例：

实例(Python 3.0+)

#!/usr/bin/python3

# 可写函数说明

sum = lambda arg1, arg2: arg1 + arg2

# 调用sum函数

print ("相加后的值为 : ", sum( 10, 20 ))

print ("相加后的值为 : ", sum( 20, 20 ))

以上实例输出结果：

相加后的值为 : 30相加后的值为 : 40

# 7.Python3 数据结构

## 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。

## 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

# 8.Python3 模块

## import 语句

想使用 Python 源文件，只需在另一个源文件里执行 import 语句，语法如下：

import module1[, module2[,... moduleN]

当解释器遇到 import 语句，如果模块在当前的搜索路径就会被导入。

一个模块只会被导入一次，不管你执行了多少次import。

## from … import 语句

Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中，语法如下：

from modname import name1[, name2[, ... nameN]]

例如，要导入模块 fibo 的 fib 函数，使用如下语句：

>>> from fibo import fib, fib2>>> fib(500)1 1 2 3 5 8 13 21 34 55 89 144 233 377

这个声明不会把整个fibo模块导入到当前的命名空间中，它只会将fibo里的fib函数引入进来。

## from … import \* 语句

把一个模块的所有内容全都导入到当前的命名空间也是可行的，只需使用如下声明：

from modname import \*

这提供了一个简单的方法来导入一个模块中的所有项目。然而这种声明不该被过多地使用。

## \_\_name\_\_属性

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

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

说明：**\_\_name\_\_** 与 **\_\_main\_\_** 底下是双下划线， **\_ \_** 是这样去掉中间的那个空格。

## dir() 函数

内置的函数 dir() 可以找到模块内定义的所有名称。以一个字符串列表的形式返回:

# Python3 输入和输出

## 输出格式美化

Python两种输出值的方式: 表达式语句和 print() 函数。

第三种方式是使用文件对象的 write() 方法，标准输出文件可以用 sys.stdout 引用。

如果你希望输出的形式更加多样，可以使用 str.format() 函数来格式化输出值。

如果你希望将输出的值转成字符串，可以使用 repr() 或 str() 函数来实现。

* **str()：** 函数返回一个用户易读的表达形式。
* **repr()：** 产生一个解释器易读的表达形式。

例如

>>> s = 'Hello, Runoob'  
>>> str(s)  
'Hello, Runoob'  
>>> repr(s)  
"'Hello, Runoob'"

str.format() 的基本使用如下:

>>> **print**('{}网址： "{}!"'.format('菜鸟教程', 'www.runoob.com'))  
菜鸟教程网址： "www.runoob.com!"

括号及其里面的字符 (称作格式化字段) 将会被 format() 中的参数替换。

在括号中的数字用于指向传入对象在 format() 中的位置，如下所示：

>>> **print**('{0} 和 {1}'.format('Google', 'Runoob'))  
Google 和 Runoob  
>>> **print**('{1} 和 {0}'.format('Google', 'Runoob'))  
Runoob 和 Google

如果在 format() 中使用了关键字参数, 那么它们的值会指向使用该名字的参数。

>>> **print**('{name}网址： {site}'.format(name='菜鸟教程', site='www.runoob.com'))  
菜鸟教程网址： www.runoob.com

位置及关键字参数可以任意的结合:

>>> **print**('站点列表 {0}, {1}, 和 {other}。'.format('Google', 'Runoob', other='Taobao'))  
站点列表 Google, Runoob, 和 Taobao。

**!a** (使用 **ascii()**), **!s** (使用 **str()**) 和 **!r** (使用**repr()**) 可以用于在格式化某个值之前对其进行转化:

>>> **import** math  
>>> **print**('常量 PI 的值近似为： {}。'.format(math.pi))  
常量 PI 的值近似为： 3.141592653589793。  
>>> **print**('常量 PI 的值近似为： {!r}。'.format(math.pi))  
常量 PI 的值近似为： 3.141592653589793。

可选项 **:** 和格式标识符可以跟着字段名。 这就允许对值进行更好的格式化。 下面的例子将 Pi 保留到小数点后三位：

>>> **import** math  
>>> **print**('常量 PI 的值近似为 {0:.3f}。'.format(math.pi))  
常量 PI 的值近似为 3.142。

在 **:** 后传入一个整数, 可以保证该域至少有这么多的宽度。 用于美化表格时很有用。

>>> table = {'Google': 1, 'Runoob': 2, 'Taobao': 3}  
>>> **for** name, number **in** table.items():  
...     **print**('{0:10} ==> {1:10d}'.format(name, number))  
...  
Google     ==>          1  
Runoob     ==>          2  
Taobao     ==>          3

如果你有一个很长的格式化字符串, 而你不想将它们分开, 那么在格式化时通过变量名而非位置会是很好的事情。

最简单的就是传入一个字典, 然后使用方括号 **[]** 来访问键值 :

>>> table = {'Google': 1, 'Runoob': 2, 'Taobao': 3}  
>>> **print**('Runoob: {0[Runoob]:d}; Google: {0[Google]:d}; Taobao: {0[Taobao]:d}'.format(table))  
Runoob: 2; Google: 1; Taobao: 3

也可以通过在 table 变量前使用 **\*\*** 来实现相同的功能：

>>> table = {'Google': 1, 'Runoob': 2, 'Taobao': 3}  
>>> **print**('Runoob: {Runoob:d}; Google: {Google:d}; Taobao: {Taobao:d}'.format(\*\*table))  
Runoob: 2; Google: 1; Taobao: 3

旧式字符串格式化

**%** 操作符也可以实现字符串格式化。 它将左边的参数作为类似 **sprintf()** 式的格式化字符串, 而将右边的代入, 然后返回格式化后的字符串. 例如:

>>> **import** math  
>>> **print**('常量 PI 的值近似为：%5.3f。' % math.pi)  
常量 PI 的值近似为：3.142。

因为 str.format() 比较新的函数， 大多数的 Python 代码仍然使用 % 操作符。但是因为这种旧式的格式化最终会从该语言中移除, 应该更多的使用 str.format().

## 读取键盘输入

Python提供了 input() 内置函数从标准输入读入一行文本，默认的标准输入是键盘。

input 可以接收一个Python表达式作为输入，并将运算结果返回。

实例

#!/usr/bin/python3  
  
str = input("请输入：");  
**print** ("你输入的内容是: ", str)

这会产生如下的对应着输入的结果：

请输入：菜鸟教程你输入的内容是: 菜鸟教程

## 读和写文件

open() 将会返回一个 file 对象，基本语法格式如下:

open(filename, mode)

* filename：包含了你要访问的文件名称的字符串值。
* mode：决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

不同模式打开文件的完全列表：

|  |  |
| --- | --- |
| **模式** | **描述** |
| r | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 |
| rb | 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。 |
| r+ | 打开一个文件用于读写。文件指针将会放在文件的开头。 |
| rb+ | 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。 |
| w | 打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| wb | 以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| w+ | 打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| wb+ | 以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| a | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| ab | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| a+ | 打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。 |
| ab+ | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。 |

## 文件对象的方法

本节中剩下的例子假设已经创建了一个称为 f 的文件对象。

### **f.read()**

为了读取一个文件的内容，调用 f.read(size), 这将读取一定数目的数据, 然后作为字符串或字节对象返回。

size 是一个可选的数字类型的参数。 当 size 被忽略了或者为负, 那么该文件的所有内容都将被读取并且返回。

### **f.readline()**

f.readline() 会从文件中读取单独的一行。换行符为 '\n'。f.readline() 如果返回一个空字符串, 说明已经已经读取到最后一行。

### **f.readlines()**

f.readlines() 将返回该文件中包含的所有行。

如果设置可选参数 sizehint, 则读取指定长度的字节, 并且将这些字节按行分割。

另一种方式是迭代一个文件对象然后读取每行:

实例

#!/usr/bin/python3  
  
# 打开一个文件  
f = open("/tmp/foo.txt", "r")  
  
**for** line **in** f:  
    **print**(line, end='')  
  
# 关闭打开的文件  
f.close()

执行以上程序，输出结果为：

Python 是一个非常好的语言。是的，的确非常好!!

这个方法很简单, 但是并没有提供一个很好的控制。 因为两者的处理机制不同, 最好不要混用。

### **f.write()**

f.write(string) 将 string 写入到文件中, 然后返回写入的字符数。

### **f.tell()**

f.tell() 返回文件对象当前所处的位置, 它是从文件开头开始算起的字节数。

### **f.seek()**

如果要改变文件当前的位置, 可以使用 f.seek(offset, from\_what) 函数。

from\_what 的值, 如果是 0 表示开头, 如果是 1 表示当前位置, 2 表示文件的结尾，

### **f.close()**

在文本文件中 (那些打开文件的模式下没有 b 的), 只会相对于文件起始位置进行定位。

当你处理完一个文件后, 调用 f.close() 来关闭文件并释放系统的资源，如果尝试再调用该文件，则会抛出异常。

当处理一个文件对象时, 使用 with 关键字是非常好的方式。在结束后, 它会帮你正确的关闭文件。 而且写起来也比 try - finally 语句块要简短:

>>> **with** open('/tmp/foo.txt', 'r') **as** f:  
...     read\_data = f.read()  
>>> f.closed  
True

## pickle 模块

python的pickle模块实现了基本的数据序列和反序列化。

通过pickle模块的序列化操作我们能够将程序中运行的对象信息保存到文件中去，永久存储。

通过pickle模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

基本接口：

pickle.dump(obj, file, [,protocol])

有了 pickle 这个对象, 就能对 file 以读取的形式打开:

x = pickle.load(file)

**注解：**从 file 中读取一个字符串，并将它重构为原来的python对象。

**file:** 类文件对象，有read()和readline()接口。

一、dump()方法

pickle.dump(obj, file, [,protocol])

注释：序列化对象，将对象obj保存到文件file中去。参数protocol是序列化模式，默认是0（ASCII协议，表示以文本的形式进行序列化），protocol的值还可以是1和2（1和2表示以二进制的形式进行序列化。其中，1是老式的二进制协议；2是新二进制协议）。file表示保存到的类文件对象，file必须有write()接口，file可以是一个以'w'打开的文件或者是一个StringIO对象，也可以是任何可以实现write()接口的对象。

二、load()方法

pickle.load(file)

注释：反序列化对象，将文件中的数据解析为一个python对象。file中有read()接口和readline()接口

实例 1

#!/usr/bin/python3  
**import** pickle  
  
# 使用pickle模块将数据对象保存到文件  
data1 = {'a': [1, 2.0, 3, 4+6j],  
         'b': ('string', u'Unicode string'),  
         'c': None}  
  
selfref\_list = [1, 2, 3]  
selfref\_list.append(selfref\_list)  
  
output = open('data.pkl', 'wb')  
  
# Pickle dictionary using protocol 0.  
pickle.dump(data1, output)  
  
# Pickle the list using the highest protocol available.  
pickle.dump(selfref\_list, output, 2)  
  
output.close()

实例 2

#!/usr/bin/python3  
**import** pickle  
  
#使用pickle模块从文件中重构python对象  
pkl\_file = open('data.pkl', 'rb')  
  
data1 = pickle.load(pkl\_file)  
print(data1)  
  
data2 = pickle.load(pkl\_file)  
print(data2)  
  
pkl\_file.close()

# 10.Python3 File(文件) 方法

### **open() 方法**

Python open() 方法用于打开一个文件，并返回文件对象，在对文件进行处理过程都需要使用到这个函数，如果该文件无法被打开，会抛出 OSError。

**注意：**使用 open() 方法一定要保证关闭文件对象，即调用 close() 方法。

open() 函数常用形式是接收两个参数：文件名(file)和模式(mode)。

open(file, mode='r')

完整的语法格式为：

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

参数说明:

* file: 必需，文件路径（相对或者绝对路径）。
* mode: 可选，文件打开模式
* buffering: 设置缓冲
* encoding: 一般使用utf8
* errors: 报错级别
* newline: 区分换行符
* closefd: 传入的file参数类型
* opener:

### **file 对象**

file 对象使用 open 函数来创建，下表列出了 file 对象常用的函数：

|  |  |
| --- | --- |
| **序号** | **方法及描述** |
| 1 | [file.close()](https://www.runoob.com/python3/python3-file-close.html)  关闭文件。关闭后文件不能再进行读写操作。 |
| 2 | [file.flush()](https://www.runoob.com/python3/python3-file-flush.html)  刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件, 而不是被动的等待输出缓冲区写入。 |
| 3 | [file.fileno()](https://www.runoob.com/python3/python3-file-fileno.html)  返回一个整型的文件描述符(file descriptor FD 整型), 可以用在如os模块的read方法等一些底层操作上。 |
| 4 | [file.isatty()](https://www.runoob.com/python3/python3-file-isatty.html)  如果文件连接到一个终端设备返回 True，否则返回 False。 |
| 5 | [file.next()](https://www.runoob.com/python3/python3-file-next.html)  **Python 3 中的 File 对象不支持 next() 方法。**  返回文件下一行。 |
| 6 | [file.read([size])](https://www.runoob.com/python3/python3-file-read.html)  从文件读取指定的字节数，如果未给定或为负则读取所有。 |
| 7 | [file.readline([size])](https://www.runoob.com/python3/python3-file-readline.html)  读取整行，包括 "\n" 字符。 |
| 8 | [file.readlines([sizeint])](https://www.runoob.com/python3/python3-file-readlines.html)  读取所有行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比 sizeint 较大, 因为需要填充缓冲区。 |
| 9 | [file.seek(offset[, whence])](https://www.runoob.com/python3/python3-file-seek.html)  移动文件读取指针到指定位置 |
| 10 | [file.tell()](https://www.runoob.com/python3/python3-file-tell.html)  返回文件当前位置。 |
| 11 | [file.truncate([size])](https://www.runoob.com/python3/python3-file-truncate.html)  从文件的首行首字符开始截断，截断文件为 size 个字符，无 size 表示从当前位置截断；截断之后后面的所有字符被删除，其中 Widnows 系统下的换行代表2个字符大小。 |
| 12 | [file.write(str)](https://www.runoob.com/python3/python3-file-write.html)  将字符串写入文件，返回的是写入的字符长度。 |
| 13 | [file.writelines(sequence)](https://www.runoob.com/python3/python3-file-writelines.html)  向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。 |

# 11.Python3 OS 文件/目录方法

**os** 模块提供了非常丰富的方法用来处理文件和目录。常用的方法如下表所示：

|  |  |
| --- | --- |
| **序号** | **方法及描述** |
| 1 | [os.access(path, mode)](https://www.runoob.com/python3/python3-os-access.html)  检验权限模式 |
| 2 | [os.chdir(path)](https://www.runoob.com/python3/python3-os-chdir.html)  改变当前工作目录 |
| 3 | [os.chflags(path, flags)](https://www.runoob.com/python3/python3-os-chflags.html)  设置路径的标记为数字标记。 |
| 4 | [os.chmod(path, mode)](https://www.runoob.com/python3/python3-os-chmod.html)  更改权限 |
| 5 | [os.chown(path, uid, gid)](https://www.runoob.com/python3/python3-os-chown.html)  更改文件所有者 |
| 6 | [os.chroot(path)](https://www.runoob.com/python3/python3-os-chroot.html)  改变当前进程的根目录 |
| 7 | [os.close(fd)](https://www.runoob.com/python3/python3-os-close.html)  关闭文件描述符 fd |
| 8 | [os.closerange(fd\_low, fd\_high)](https://www.runoob.com/python3/python3-os-closerange.html)  关闭所有文件描述符，从 fd\_low (包含) 到 fd\_high (不包含), 错误会忽略 |
| 9 | [os.dup(fd)](https://www.runoob.com/python3/python3-os-dup.html)  复制文件描述符 fd |
| 10 | [os.dup2(fd, fd2)](https://www.runoob.com/python3/python3-os-dup2.html)  将一个文件描述符 fd 复制到另一个 fd2 |
| 11 | [os.fchdir(fd)](https://www.runoob.com/python3/python3-os-fchdir.html)  通过文件描述符改变当前工作目录 |
| 12 | [os.fchmod(fd, mode)](https://www.runoob.com/python3/python3-os-fchmod.html)  改变一个文件的访问权限，该文件由参数fd指定，参数mode是Unix下的文件访问权限。 |
| 13 | [os.fchown(fd, uid, gid)](https://www.runoob.com/python3/python3-os-fchown.html)  修改一个文件的所有权，这个函数修改一个文件的用户ID和用户组ID，该文件由文件描述符fd指定。 |
| 14 | [os.fdatasync(fd)](https://www.runoob.com/python3/python3-os-fdatasync.html)  强制将文件写入磁盘，该文件由文件描述符fd指定，但是不强制更新文件的状态信息。 |
| 15 | [os.fdopen(fd[, mode[, bufsize]])](https://www.runoob.com/python3/python3-os-fdopen.html)  通过文件描述符 fd 创建一个文件对象，并返回这个文件对象 |
| 16 | [os.fpathconf(fd, name)](https://www.runoob.com/python3/python3-os-fpathconf.html)  返回一个打开的文件的系统配置信息。name为检索的系统配置的值，它也许是一个定义系统值的字符串，这些名字在很多标准中指定（POSIX.1, Unix 95, Unix 98, 和其它）。 |
| 17 | [os.fstat(fd)](https://www.runoob.com/python3/python3-os-fstat.html)  返回文件描述符fd的状态，像stat()。 |
| 18 | [os.fstatvfs(fd)](https://www.runoob.com/python3/python3-os-fstatvfs.html)  返回包含文件描述符fd的文件的文件系统的信息，Python 3.3 相等于 statvfs()。 |
| 19 | [os.fsync(fd)](https://www.runoob.com/python3/python3-os-fsync.html)  强制将文件描述符为fd的文件写入硬盘。 |
| 20 | [os.ftruncate(fd, length)](https://www.runoob.com/python3/python3-os-ftruncate.html)  裁剪文件描述符fd对应的文件, 所以它最大不能超过文件大小。 |
| 21 | [os.getcwd()](https://www.runoob.com/python3/python3-os-getcwd.html)  返回当前工作目录 |
| 22 | [os.getcwdu()](https://www.runoob.com/python3/python3-os-getcwdu.html)  返回一个当前工作目录的Unicode对象 |
| 23 | [os.isatty(fd)](https://www.runoob.com/python3/python3-os-isatty.html)  如果文件描述符fd是打开的，同时与tty(-like)设备相连，则返回true, 否则False。 |
| 24 | [os.lchflags(path, flags)](https://www.runoob.com/python3/python3-os-lchflags.html)  设置路径的标记为数字标记，类似 chflags()，但是没有软链接 |
| 25 | [os.lchmod(path, mode)](https://www.runoob.com/python3/python3-os-lchmod.html)  修改连接文件权限 |
| 26 | [os.lchown(path, uid, gid)](https://www.runoob.com/python3/python3-os-lchown.html)  更改文件所有者，类似 chown，但是不追踪链接。 |
| 27 | [os.link(src, dst)](https://www.runoob.com/python3/python3-os-link.html)  创建硬链接，名为参数 dst，指向参数 src |
| 28 | [os.listdir(path)](https://www.runoob.com/python3/python3-os-listdir.html)  返回path指定的文件夹包含的文件或文件夹的名字的列表。 |
| 29 | [os.lseek(fd, pos, how)](https://www.runoob.com/python3/python3-os-lseek.html)  设置文件描述符 fd当前位置为pos, how方式修改: SEEK\_SET 或者 0 设置从文件开始的计算的pos; SEEK\_CUR或者 1 则从当前位置计算; os.SEEK\_END或者2则从文件尾部开始. 在unix，Windows中有效 |
| 30 | [os.lstat(path)](https://www.runoob.com/python3/python3-os-lstat.html)  像stat(),但是没有软链接 |
| 31 | [os.major(device)](https://www.runoob.com/python3/python3-os-major.html)  从原始的设备号中提取设备major号码 (使用stat中的st\_dev或者st\_rdev field)。 |
| 32 | [os.makedev(major, minor)](https://www.runoob.com/python3/python3-os-makedev.html)  以major和minor设备号组成一个原始设备号 |
| 33 | [os.makedirs(path[, mode])](https://www.runoob.com/python3/python3-os-makedirs.html)  递归文件夹创建函数。像mkdir(), 但创建的所有intermediate-level文件夹需要包含子文件夹。 |
| 34 | [os.minor(device)](https://www.runoob.com/python3/python3-os-minor.html)  从原始的设备号中提取设备minor号码 (使用stat中的st\_dev或者st\_rdev field )。 |
| 35 | [os.mkdir(path[, mode])](https://www.runoob.com/python3/python3-os-mkdir.html)  以数字mode的mode创建一个名为path的文件夹.默认的 mode 是 0777 (八进制)。 |
| 36 | [os.mkfifo(path[, mode])](https://www.runoob.com/python3/python3-os-mkfifo.html)  创建命名管道，mode 为数字，默认为 0666 (八进制) |
| 37 | [os.mknod(filename[, mode=0600, device])](https://www.runoob.com/python3/python3-os-mknod.html) 创建一个名为filename文件系统节点（文件，设备特别文件或者命名pipe）。 |
| 38 | [os.open(file, flags[, mode])](https://www.runoob.com/python3/python3-os-open.html)  打开一个文件，并且设置需要的打开选项，mode参数是可选的 |
| 39 | [os.openpty()](https://www.runoob.com/python3/python3-os-openpty.html)  打开一个新的伪终端对。返回 pty 和 tty的文件描述符。 |
| 40 | [os.pathconf(path, name)](https://www.runoob.com/python3/python3-os-pathconf.html)  返回相关文件的系统配置信息。 |
| 41 | [os.pipe()](https://www.runoob.com/python3/python3-os-pipe.html)  创建一个管道. 返回一对文件描述符(r, w) 分别为读和写 |
| 42 | [os.popen(command[, mode[, bufsize]])](https://www.runoob.com/python3/python3-os-popen.html)  从一个 command 打开一个管道 |
| 43 | [os.read(fd, n)](https://www.runoob.com/python3/python3-os-read.html)  从文件描述符 fd 中读取最多 n 个字节，返回包含读取字节的字符串，文件描述符 fd对应文件已达到结尾, 返回一个空字符串。 |
| 44 | [os.readlink(path)](https://www.runoob.com/python3/python3-os-readlink.html)  返回软链接所指向的文件 |
| 45 | [os.remove(path)](https://www.runoob.com/python3/python3-os-remove.html)  删除路径为path的文件。如果path 是一个文件夹，将抛出OSError; 查看下面的rmdir()删除一个 directory。 |
| 46 | [os.removedirs(path)](https://www.runoob.com/python3/python3-os-removedirs.html)  递归删除目录。 |
| 47 | [os.rename(src, dst)](https://www.runoob.com/python3/python3-os-rename.html)  重命名文件或目录，从 src 到 dst |
| 48 | [os.renames(old, new)](https://www.runoob.com/python3/python3-os-renames.html)  递归地对目录进行更名，也可以对文件进行更名。 |
| 49 | [os.rmdir(path)](https://www.runoob.com/python3/python3-os-rmdir.html)  删除path指定的空目录，如果目录非空，则抛出一个OSError异常。 |
| 50 | [os.stat(path)](https://www.runoob.com/python3/python3-os-stat.html)  获取path指定的路径的信息，功能等同于C API中的stat()系统调用。 |
| 51 | [os.stat\_float\_times([newvalue])](https://www.runoob.com/python3/python3-os-stat_float_times.html) 决定stat\_result是否以float对象显示时间戳 |
| 52 | [os.statvfs(path)](https://www.runoob.com/python3/python3-os-statvfs.html)  获取指定路径的文件系统统计信息 |
| 53 | [os.symlink(src, dst)](https://www.runoob.com/python3/python3-os-symlink.html)  创建一个软链接 |
| 54 | [os.tcgetpgrp(fd)](https://www.runoob.com/python3/python3-os-tcgetpgrp.html)  返回与终端fd（一个由os.open()返回的打开的文件描述符）关联的进程组 |
| 55 | [os.tcsetpgrp(fd, pg)](https://www.runoob.com/python3/python3-os-tcsetpgrp.html)  设置与终端fd（一个由os.open()返回的打开的文件描述符）关联的进程组为pg。 |
| 56 | os.tempnam([dir[, prefix]])  **Python3 中已删除。**返回唯一的路径名用于创建临时文件。 |
| 57 | os.tmpfile()  **Python3 中已删除。**返回一个打开的模式为(w+b)的文件对象 .这文件对象没有文件夹入口，没有文件描述符，将会自动删除。 |
| 58 | os.tmpnam()  **Python3 中已删除。**为创建一个临时文件返回一个唯一的路径 |
| 59 | [os.ttyname(fd)](https://www.runoob.com/python3/python3-os-ttyname.html)  返回一个字符串，它表示与文件描述符fd 关联的终端设备。如果fd 没有与终端设备关联，则引发一个异常。 |
| 60 | [os.unlink(path)](https://www.runoob.com/python3/python3-os-unlink.html)  删除文件路径 |
| 61 | [os.utime(path, times)](https://www.runoob.com/python3/python3-os-utime.html)  返回指定的path文件的访问和修改的时间。 |
| 62 | [os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])](https://www.runoob.com/python3/python3-os-walk.html)  输出在文件夹中的文件名通过在树中游走，向上或者向下。 |
| 63 | [os.write(fd, str)](https://www.runoob.com/python3/python3-os-write.html)  写入字符串到文件描述符 fd中. 返回实际写入的字符串长度 |
| 64 | [os.path 模块](https://www.runoob.com/python3/python3-os-path.html" \t "/home/zgj/文档\\x/_blank)  获取文件的属性信息。 |

# 12.Python3 错误和异常

## 异常处理

以下例子中，让用户输入一个合法的整数，但是允许用户中断这个程序（使用 Control-C 或者操作系统提供的方法）。用户中断的信息会引发一个 KeyboardInterrupt 异常。

>>>while True:

try:

x = int(input("Please enter a number: "))

break

except ValueError:

print("Oops! That was no valid number. Try again ")

try语句按照如下方式工作；

* 首先，执行try子句（在关键字try和关键字except之间的语句）
* 如果没有异常发生，忽略except子句，try子句执行后结束。
* 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的except子句将被执行。最后执行 try 语句之后的代码。
* 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

处理程序将只针对对应的try子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

一个except子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组，例如:

except (RuntimeError, TypeError, NameError): pass

最后一个except子句可以忽略异常的名称，它将被当作通配符使用

try except 语句还有一个可选的else子句，如果使用这个子句，那么必须放在所有的except子句之后。这个子句将在try子句没有发生任何异常的时候执行。

## 抛出异常

Python 使用 raise 语句抛出一个指定的异常。例如:

>>>raise NameError('HiThere') Traceback (most recent call last): File "<stdin>", line 1, in ? NameError: HiThere

raise 唯一的一个参数指定了要被抛出的异常。它必须是一个异常的实例或者是异常的类（也就是 Exception 的子类）。

## Finally字句

try:

... raise KeyboardInterrupt

finally:

... print('Goodbye, world!')

以上例子不管 try 子句里面有没有发生异常，finally 子句都会执行。

如果一个异常在 try 子句里（或者在 except 和 else 子句里）被抛出，而又没有任何的 except 把它截住，那么这个异常会在 finally 子句执行后被抛出。

## 关键字with

关键词 with 语句就可以保证诸如文件之类的对象在使用完之后一定会正确的执行他的清理方法:

with open("myfile.txt") as f:

for line in f:

print(line, end="")

以上这段代码执行完毕后，就算在处理过程中出问题了，文件 f 总是会关闭。

# 13.面向对象

## 类定义

语法格式如下：

class ClassName:

<statement-1>

<statement-N>

类实例化后，可以使用其属性，实际上，创建一个类之后，可以通过类名访问其属性。

实例(Python 3.0+)

#!/usr/bin/python3

class MyClass:

"""一个简单的类实例"""

i = 12345

def f(self):

return 'hello world'

# 实例化类

x = MyClass()

# 访问类的属性和方法

print("MyClass 类的属性 i 为：", x.i)

print("MyClass 类的方法 f 输出为：", x.f())

## 构造方法

类有一个名为 \_\_init\_\_() 的特殊方法（**构造方法**），该方法在类实例化时会自动调用，像下面这样：

def \_\_init\_\_(self):

self.data = []

类定义了 \_\_init\_\_() 方法，类的实例化操作会自动调用 \_\_init\_\_() 方法。

当然， \_\_init\_\_() 方法可以有参数，参数通过 \_\_init\_\_() 传递到类的实例化操作上。例如:

实例(Python 3.0+)

#!/usr/bin/python3

class Complex:

def \_\_init\_\_(self, realpart, imagpart):

self.r = realpart

self.i = imagpart

x = Complex(3.0, -4.5)

print(x.r, x.i) # 输出结果：3.0 -4.5

## 类的方法

在类的内部，使用 **def** 关键字来定义一个方法，与一般函数定义不同，类方法必须包含参数 self, 且为第一个参数，self 代表的是类的实例。

**self代表类的实例，而非类**

类的方法与普通的函数只有一个特别的区别——它们必须有一个额外的**第一个参数名称**, 按照惯例它的名称是 self。

class Test:

def prt(self)

print(self)

print(self.\_\_class\_\_)

t = Test()

t.prt()

从执行结果可以很明显的看出，self 代表的是类的实例，代表当前对象的地址，而 self.class 则指向类。

self 不是 python 关键字，我们把他换成 runoob 也是可以正常执行的:

## 继承

Python 同样支持类的继承，如果一种语言不支持继承，类就没有什么意义。派生类的定义如下所示:

class DerivedClassName(BaseClassName1):

<statement-1>

<statement-N>

需要注意圆括号中基类的顺序，若是基类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找基类中是否包含方法。

BaseClassName（示例中的基类名）必须与派生类定义在一个作用域内。除了类，还可以用表达式，基类定义在另一个模块中时这一点非常有用:

class DerivedClassName(modname.BaseClassName):

实例(Python 3.0+)

#!/usr/bin/python3

#类定义

class people:

#定义基本属性

name = ''

age = 0

#定义私有属性,私有属性在类外部无法直接进行访问

\_\_weight = 0

#定义构造方法

def \_\_init\_\_(self,n,a,w):

self.name = n

self.age = a

self.\_\_weight = w

def speak(self):

print("%s 说: 我 %d 岁。" %(self.name,self.age))

#单继承示例

class student(people):

grade = ''

def \_\_init\_\_(self,n,a,w,g):

#调用父类的构函

people.\_\_init\_\_(self,n,a,w)

self.grade = g

#覆写父类的方法

def speak(self):

print("%s 说: 我 %d 岁了，我在读 %d 年级"%(self.name,self.age,self.grade))

s = student('ken',10,60,3)

s.speak()

## 多继承

Python同样有限的支持多继承形式。多继承的类定义形如下例:

class DerivedClassName(Base1, Base2, Base3):

<statement-1>

<statement-N>

需要注意圆括号中父类的顺序，若是父类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找父类中是否包含方法。

情况一：**子类需要自动调用父类的方法：**子类不重写\_\_init\_\_()方法，实例化子类后，会自动调用父类的\_\_init\_\_()的方法。

情况二：**子类不需要自动调用父类的方法：**子类重写\_\_init\_\_()方法，实例化子类后，将不会自动调用父类的\_\_init\_\_()的方法。

情况三：**子类重写\_\_init\_\_()方法又需要调用父类的方法：**使用super关键词，或者直接用父类的名字加以调用：

## 类属性与方法

### **类的私有属性**

**\_\_private\_attrs**：两个下划线开头，声明该属性为私有，不能在类的外部被使用或直接访问。在类内部的方法中使用时 **self.\_\_private\_attrs**。

### **类的方法**

在类的内部，使用 def 关键字来定义一个方法，与一般函数定义不同，类方法必须包含参数 **self**，且为第一个参数，**self** 代表的是类的实例。

**self** 的名字并不是规定死的，也可以使用 **this**，但是最好还是按照约定是用 **self**。

### **类的私有方法**

**\_\_private\_method**：两个下划线开头，声明该方法为私有方法，只能在类的内部调用 ，不能在类的外部调用。**self.\_\_private\_methods**。

### **类的专有方法：**

* **\_\_init\_\_ :** 构造函数，在生成对象时调用
* **\_\_del\_\_ :** 析构函数，释放对象时使用
* **\_\_repr\_\_ :** 打印，转换
* **\_\_setitem\_\_ :** 按照索引赋值
* **\_\_getitem\_\_:** 按照索引获取值
* **\_\_len\_\_:** 获得长度
* **\_\_cmp\_\_:** 比较运算
* **\_\_call\_\_:** 函数调用
* **\_\_add\_\_:** 加运算
* **\_\_sub\_\_:** 减运算
* **\_\_mul\_\_:** 乘运算
* **\_\_truediv\_\_:** 除运算
* **\_\_mod\_\_:** 求余运算
* **\_\_pow\_\_:** 乘方

## 方法的重载

 Python 并不支持

但是可以通过一些小技巧近似地达到重载的效果

class ADD:  
 def add(self,\*args):  
 if len(args) > 1:  
 print("多个参数")  
 else:  
 print("只有一个参数")  
add = ADD()  
add.add(2)  
add.add(1,2,3)

如果参数类型不同可以使用functools.singledispatch 装饰器。（此种方法无法在类的内部重载方法）

from functools import singledispatch  
@singledispatch  
def add(a):  
 print("int")  
@add.register(float)  
def \_(a):  
 print("float")  
@add.register(str)  
def \_(a):  
 print("string")  
add(1)  
add(2.3)  
add("zgj")

# 14.Python3 命名空间和作用域

## 命名空间：

* **内置名称（built-in names**）， Python 语言内置的名称，比如函数名 abs、char 和异常名称 BaseException、Exception 等等。
* **全局名称（global names）**，模块中定义的名称，记录了模块的变量，包括函数、类、其它导入的模块、模块级的变量和常量。
* **局部名称（local names）**，函数中定义的名称，记录了函数的变量，包括函数的参数和局部定义的变量。（类中定义的也是）

命名空间查找顺序:

假设我们要使用变量 runoob，则 Python 的查找顺序为：**局部的命名空间去 -> 全局命名空间 -> 内置命名空间**。

如果找不到变量 runoob，它将放弃查找并引发一个 NameError 异常:

## 作用域

Python的作用域一共有4种，分别是：

有四种作用域：

* **L（Local）**：最内层，包含局部变量，比如一个函数/方法内部。
* **E（Enclosing）**：包含了非局部(non-local)也非全局(non-global)的变量。比如两个嵌套函数，一个函数（或类） A 里面又包含了一个函数 B ，那么对于 B 中的名称来说 A 中的作用域就为 nonlocal。
* **G（Global）**：当前脚本的最外层，比如当前模块的全局变量。
* **B（Built-in）**： 包含了内建的变量/关键字等。，最后被搜索

规则顺序： **L –> E –> G –>gt; B**。

在局部找不到，便会去局部外的局部找（例如闭包），再找不到就会去全局找，再者去内置中找。

## 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。

### **global 和 nonlocal关键字**

当内部作用域想修改外部作用域的变量时，就要用到global和nonlocal关键字了。

以下实例修改全局变量 num：

实例(Python 3.0+)

#!/usr/bin/python3

num = 1

def fun1():

global num # 需要使用 global 关键字声明

print(num)

num = 123

print(num)

fun1()

print(num)

以上实例输出结果：

1

123

123

如果要修改嵌套作用域（enclosing 作用域，外层非全局作用域）中的变量则需要 nonlocal 关键字了，如下实例：

实例(Python 3.0+)

#!/usr/bin/python3

def outer():

num = 10

def inner():

nonlocal num # nonlocal关键字声明

num = 100

print(num)

inner()

print(num)

outer()

以上实例输出结果：

100

100

# 15.Python3 标准库概览

## 操作系统接口

os模块提供了不少与操作系统相关联的函数。

>>> import os

>>> os.getcwd() # 返回当前的工作目录'C:\\Python34'

>>> os.chdir('/server/accesslogs') # 修改当前的工作目录

>>> os.system('mkdir today') # 执行系统命令 mkdir 0

建议使用 "import os" 风格而非 "from os import \*"。这样可以保证随操作系统不同而有所变化的 os.open() 不会覆盖内置函数 open()。

在使用 os 这样的大型模块时内置的 dir() 和 help() 函数非常有用:

## 文件通配符

glob模块提供了一个函数用于从目录通配符搜索中生成文件列表:

>>> import glob

>>> glob.glob('\*.py')

['primes.py', 'random.py', 'quote.py']

## 命令行参数

通用工具脚本经常调用命令行参数。这些命令行参数以链表形式存储于 sys 模块的 argv 变量。例如在命令行中执行 "python demo.py one two three" 后可以得到以下输出结果:

>>> import sys>>> print(sys.argv)['demo.py', 'one', 'two', 'three']

## 错误输出重定向和程序终止

sys 还有 stdin，stdout 和 stderr 属性，即使在 stdout 被重定向时，后者也可以用于显示警告和错误信息。

>>> sys.stderr.write('Warning, log file not found starting a new one\n')

Warning, log file not found starting a new one

大多脚本的定向终止都使用 "sys.exit()"。

## 字符串正则匹配

re模块为高级字符串处理提供了正则表达式工具。对于复杂的匹配和处理，正则表达式提供了简洁、优化的解决方案:

## 数学

math模块为浮点运算提供了对底层C函数库的访问:

## 随机数

Random模块提供了生成随机数的工具。

import random

## 访问 互联网

有几个模块用于访问互联网以及处理网络通信协议。其中最简单的两个是用于处理从 urls 接收的数据的 urllib.request 以及用于发送电子邮件的 smtplib:

from urllib.request import urlopen

import smtplib

第二个例子需要本地有一个在运行的邮件服务器。

## 日期和时间

datetime模块为日期和时间处理同时提供了简单和复杂的方法。

支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出。

该模块还支持时区处理:

## 数据压缩

以下模块直接支持通用的数据打包和压缩格式：zlib，gzip，bz2，zipfile，以及 tarfile。

import zlib

## 性能度量

有些用户对了解解决同一问题的不同方法之间的性能差异很感兴趣。Python 提供了一个度量工具，为这些问题提供了直接答案。

例如，使用元组封装和拆封来交换元素看起来要比使用传统的方法要诱人的多,timeit 证明了现代的方法更快一些。

from timeit import Timer

相对于 timeit 的细粒度，:mod:profile 和 pstats 模块提供了针对更大代码块的时间度量工具。

## 测试模块

开发高质量软件的方法之一是为每一个函数开发测试代码，并且在开发过程中经常进行测试

doctest模块提供了一个工具，扫描模块并根据程序中内嵌的文档字符串执行测试。

import doctest

doctest.testmod() # 自动验证嵌入测试

unittest模块不像 doctest模块那么容易使用，不过它可以在一个独立的文件里提供一个更全面的测试集:

import unittest

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*高级教程\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 1.Python3 正则表达式

import re

## re.match函数

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。

**函数语法**：

re.match(pattern, string, flags=0)

函数参数说明：

|  |  |
| --- | --- |
| **参数** | **描述** |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。参见：[正则表达式修饰符 - 可选标志](https://www.runoob.com/python3/python3-reg-expressions.html" \l "flags) |

匹配成功re.match方法返回一个匹配的对象，否则返回None。

我们可以使用group(num) 或 groups() 匹配对象函数来获取匹配表达式。

|  |  |
| --- | --- |
| **匹配对象方法** | **描述** |
| group(num=0) | 匹配的整个表达式的字符串，group() 可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组。 |
| groups() | 返回一个包含所有小组字符串的元组，从 1 到 所含的小组号。 |

## re.search方法

re.search 扫描整个字符串并返回第一个成功的匹配。

函数语法：

re.search(pattern, string, flags=0)

函数参数说明：

|  |  |
| --- | --- |
| **参数** | **描述** |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。参见：[正则表达式修饰符 - 可选标志](https://www.runoob.com/python3/python3-reg-expressions.html" \l "flags) |

匹配成功re.search方法返回一个匹配的对象，否则返回None。

我们可以使用group(num) 或 groups() 匹配对象函数来获取匹配表达式。

|  |  |
| --- | --- |
| **匹配对象方法** | **描述** |
| group(num=0) | 匹配的整个表达式的字符串，group() 可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组。 |
| groups() | 返回一个包含所有小组字符串的元组，从 1 到 所含的小组号。 |

## 检索和替换

Python 的re模块提供了re.sub用于替换字符串中的匹配项。

语法：

re.sub(pattern, repl, string, count=0, flags=0)

参数：

* pattern : 正则中的模式字符串。
* repl : 替换的字符串，也可为一个函数。
* string : 要被查找替换的原始字符串。
* count : 模式匹配后替换的最大次数，默认 0 表示替换所有的匹配。
* flags : 编译时用的匹配模式，数字形式。

前三个为必选参数，后两个为可选参数。

### **compile 函数**

compile 函数用于编译正则表达式，生成一个正则表达式（ Pattern ）对象，供 match() 和 search() 这两个函数使用。

语法格式为：

re.compile(pattern[, flags])

参数：

* pattern : 一个字符串形式的正则表达式
* flags 可选，表示匹配模式，比如忽略大小写，多行模式等，具体参数为：

re.I 忽略大小写

* + re.L 表示特殊字符集 \w, \W, \b, \B, \s, \S 依赖于当前环境
  + re.M 多行模式
  + re.S 即为' . '并且包括换行符在内的任意字符（' . '不包括换行符）
  + re.U 表示特殊字符集 \w, \W, \b, \B, \d, \D, \s, \S 依赖于 Unicode 字符属性数据库
  + re.X 为了增加可读性，忽略空格和' # '后面的注释

当匹配成功时返回一个 Match 对象，其中：

* group([group1, …]) 方法用于获得一个或多个分组匹配的字符串，当要获得整个匹配的子串时，可直接使用 group() 或 group(0)；
* start([group]) 方法用于获取分组匹配的子串在整个字符串中的起始位置（子串第一个字符的索引），参数默认值为 0；
* end([group]) 方法用于获取分组匹配的子串在整个字符串中的结束位置（子串最后一个字符的索引+1），参数默认值为 0；
* span([group]) 方法返回 (start(group), end(group))。

### **findall**

在字符串中找到正则表达式所匹配的所有子串，并返回一个列表，如果没有找到匹配的，则返回空列表。

**注意：** match 和 search 是匹配一次 findall 匹配所有。

语法格式为：

re.findall(string[, pos[, endpos]])

参数：

* **string** 待匹配的字符串。
* **pos** 可选参数，指定字符串的起始位置，默认为 0。
* **endpos** 可选参数，指定字符串的结束位置，默认为字符串的长度。

### **re.finditer**

和 findall 类似，在字符串中找到正则表达式所匹配的所有子串，并把它们作为一个迭代器返回。

re.finditer(pattern, string, flags=0)

参数：

|  |  |
| --- | --- |
| **参数** | **描述** |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。参见：[正则表达式修饰符 - 可选标志](https://www.runoob.com/python3/python3-reg-expressions.html" \l "flags) |

### **re.split**

split 方法按照能够匹配的子串将字符串分割后返回列表，它的使用形式如下：

re.split(pattern, string[, maxsplit=0, flags=0])

参数：

|  |  |
| --- | --- |
| **参数** | **描述** |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| maxsplit | 分隔次数，maxsplit=1 分隔一次，默认为 0，不限制次数。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。参见：[正则表达式修饰符 - 可选标志](https://www.runoob.com/python3/python3-reg-expressions.html" \l "flags) |

## 正则表达式对象

### **re.RegexObject**

re.compile() 返回 RegexObject 对象。

### **re.MatchObject**

group() 返回被 RE 匹配的字符串。

* **start()** 返回匹配开始的位置
* **end()** 返回匹配结束的位置
* **span()** 返回一个元组包含匹配 (开始,结束) 的位置

## 正则表达式修饰符 - 可选标志

正则表达式可以包含一些可选标志修饰符来控制匹配的模式。修饰符被指定为一个可选的标志。多个标志可以通过按位 OR(|) 它们来指定。如 re.I | re.M 被设置成 I 和 M 标志：

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| re.I | 使匹配对大小写不敏感 |
| re.L | 做本地化识别（locale-aware）匹配 |
| re.M | 多行匹配，影响 ^ 和 $ |
| re.S | 使 . 匹配包括换行在内的所有字符 |
| re.U | 根据Unicode字符集解析字符。这个标志影响 \w, \W, \b, \B. |
| re.X | 该标志通过给予你更灵活的格式以便你将正则表达式写得更易于理解。 |

## 正则表达式模式

下表列出了正则表达式模式语法中的特殊元素。如果你使用模式的同时提供了可选的标志参数，某些模式元素的含义会改变。

|  |  |
| --- | --- |
| **模式** | **描述** |
| ^ | 匹配字符串的开头 |
| $ | 匹配字符串的末尾。 |
| . | 匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符。 |
| [...] | 用来表示一组字符,单独列出：[amk] 匹配 'a'，'m'或'k' |
| [^...] | 不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。 |
| re\* | 匹配0个或多个的表达式。 |
| re+ | 匹配1个或多个的表达式。 |
| re? | 匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式 |
| re{ n} | 匹配n个前面表达式。例如，"o{2}"不能匹配"Bob"中的"o"，但是能匹配"food"中的两个o。 |
| re{ n,} | 精确匹配n个前面表达式。例如，"o{2,}"不能匹配"Bob"中的"o"，但能匹配"foooood"中的所有o。"o{1,}"等价于"o+"。"o{0,}"则等价于"o\*"。 |
| re{ n, m} | 匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式 |
| a| b | 匹配a或b |
| (re) | 匹配括号内的表达式，也表示一个组 |
| (?imx) | 正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。 |
| (?-imx) | 正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。 |
| (?: re) | 类似 (...), 但是不表示一个组 |
| (?imx: re) | 在括号中使用i, m, 或 x 可选标志 |
| (?-imx: re) | 在括号中不使用i, m, 或 x 可选标志 |
| (?#...) | 注释. |
| (?= re) | 前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，否则失败。但一旦所含表达式已经尝试，匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。 |
| (?! re) | 前向否定界定符。与肯定界定符相反；当所含表达式不能在字符串当前位置匹配时成功。 |
| (?> re) | 匹配的独立模式，省去回溯。 |
| \w | 匹配数字字母下划线 |
| \W | 匹配非数字字母下划线 |
| \s | 匹配任意空白字符，等价于 [\t\n\r\f]。 |
| \S | 匹配任意非空字符 |
| \d | 匹配任意数字，等价于 [0-9]。 |
| \D | 匹配任意非数字 |
| \A | 匹配字符串开始 |
| \Z | 匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。 |
| \z | 匹配字符串结束 |
| \G | 匹配最后匹配完成的位置。 |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。 |
| \B | 匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。 |
| \n, \t, 等。 | 匹配一个换行符。匹配一个制表符, 等 |
| \1...\9 | 匹配第n个分组的内容。 |
| \10 | 匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。 |

#### 特殊字符类

|  |  |
| --- | --- |
| **实例** | **描述** |
| . | 匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。 |
| \d | 匹配一个数字字符。等价于 [0-9]。 |
| \D | 匹配一个非数字字符。等价于 [^0-9]。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。 |
| \w | 匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9\_]'。 |
| \W | 匹配任何非单词字符。等价于 '[^A-Za-z0-9\_]'。 |

# 2.Python CGI编程

CGI(Common Gateway Interface),通用网关接口,它是一段程序,运行在服务器上如：HTTP服务器，提供同客户端HTML页面的接口。

运行CGI程序之前需要在自己使用的web服务器上打开运行CGI程序的开关，不同的服务器有不同的打开方式，具体可以百度参考；

然后写完程序放到服务器配置时指定的文件夹下，然后通过URL访问即可；

**注意：默认情况下CGI程序不能有中文，否则会导致访问页面出现空白页面的现象**

## 第一个CGI程序

我们使用Python创建第一个CGI程序，文件名为hello.py，文件位于/var/www/cgi-bin目录中，内容如下：

#!/usr/bin/python3

print ("Content-type:text/html")

print () # 空行，告诉服务器结束头部

print ('<html>')

print ('<head>')

print ('<meta charset="utf-8">')

print ('<title>Hello Word - 我的第一个 CGI 程序！</title>')

print ('</head>')

print ('<body>')

print ('<h2>Hello Word! 我是来自菜鸟教程的第一CGI程序</h2>')

print ('</body>')

print ('</html>')

文件保存后修改 hello.py，修改文件权限为 755：

chmod 755 hello.py

以上程序在浏览器访问显示结果如下：



这个的hello.py脚本是一个简单的Python脚本，脚本第一行的输出内容"Content-type:text/html"发送到浏览器并告知浏览器显示的内容类型为"text/html"。

用 print 输出一个空行用于告诉服务器结束头部信息。

## HTTP头部

hello.py文件内容中的" Content-type:text/html"即为HTTP头部的一部分，它会发送给浏览器告诉浏览器文件的内容类型。

HTTP头部的格式如下：

HTTP 字段名: 字段内容

例如：

Content-type: text/html

以下表格介绍了CGI程序中HTTP头部经常使用的信息：

|  |  |
| --- | --- |
| **头** | **描述** |
| Content-type: | 请求的与实体对应的MIME信息。例如: Content-type:text/html |
| Expires: Date | 响应过期的日期和时间 |
| Location: URL | 用来重定向接收方到非请求URL的位置来完成请求或标识新的资源 |
| Last-modified: Date | 请求资源的最后修改时间 |
| Content-length: N | 请求的内容长度 |
| Set-Cookie: String | 设置Http Cookie |

## CGI环境变量

所有的CGI程序都接收以下的环境变量，这些变量在CGI程序中发挥了重要的作用：

|  |  |
| --- | --- |
| **变量名** | **描述** |
| CONTENT\_TYPE | 这个环境变量的值指示所传递来的信息的MIME类型。目前，环境变量CONTENT\_TYPE一般都是：application/x-www-form-urlencoded,他表示数据来自于HTML表单。 |
| CONTENT\_LENGTH | 如果服务器与CGI程序信息的传递方式是POST，这个环境变量即使从标准输入STDIN中可以读到的有效数据的字节数。这个环境变量在读取所输入的数据时必须使用。 |
| HTTP\_COOKIE | 客户机内的 COOKIE 内容。 |
| HTTP\_USER\_AGENT | 提供包含了版本数或其他专有数据的客户浏览器信息。 |
| PATH\_INFO | 这个环境变量的值表示紧接在CGI程序名之后的其他路径信息。它常常作为CGI程序的参数出现。 |
| QUERY\_STRING | 如果服务器与CGI程序信息的传递方式是GET，这个环境变量的值即使所传递的信息。这个信息经跟在CGI程序名的后面，两者中间用一个问号'?'分隔。 |
| REMOTE\_ADDR | 这个环境变量的值是发送请求的客户机的IP地址，例如上面的192.168.1.67。这个值总是存在的。而且它是Web客户机需要提供给Web服务器的唯一标识，可以在CGI程序中用它来区分不同的Web客户机。 |
| REMOTE\_HOST | 这个环境变量的值包含发送CGI请求的客户机的主机名。如果不支持你想查询，则无需定义此环境变量。 |
| REQUEST\_METHOD | 提供脚本被调用的方法。对于使用 HTTP/1.0 协议的脚本，仅 GET 和 POST 有意义。 |
| SCRIPT\_FILENAME | CGI脚本的完整路径 |
| SCRIPT\_NAME | CGI脚本的的名称 |
| SERVER\_NAME | 这是你的 WEB 服务器的主机名、别名或IP地址。 |
| SERVER\_SOFTWARE | 这个环境变量的值包含了调用CGI程序的HTTP服务器的名称和版本号。例如，上面的值为Apache/2.2.14(Unix) |

以下是一个简单的CGI脚本输出CGI的环境变量：

#!/usr/bin/python3

import os

print ("Content-type: text/html")

print ()

print ("<meta charset=\"utf-8\">")

print ("<b>环境变量</b><br>")

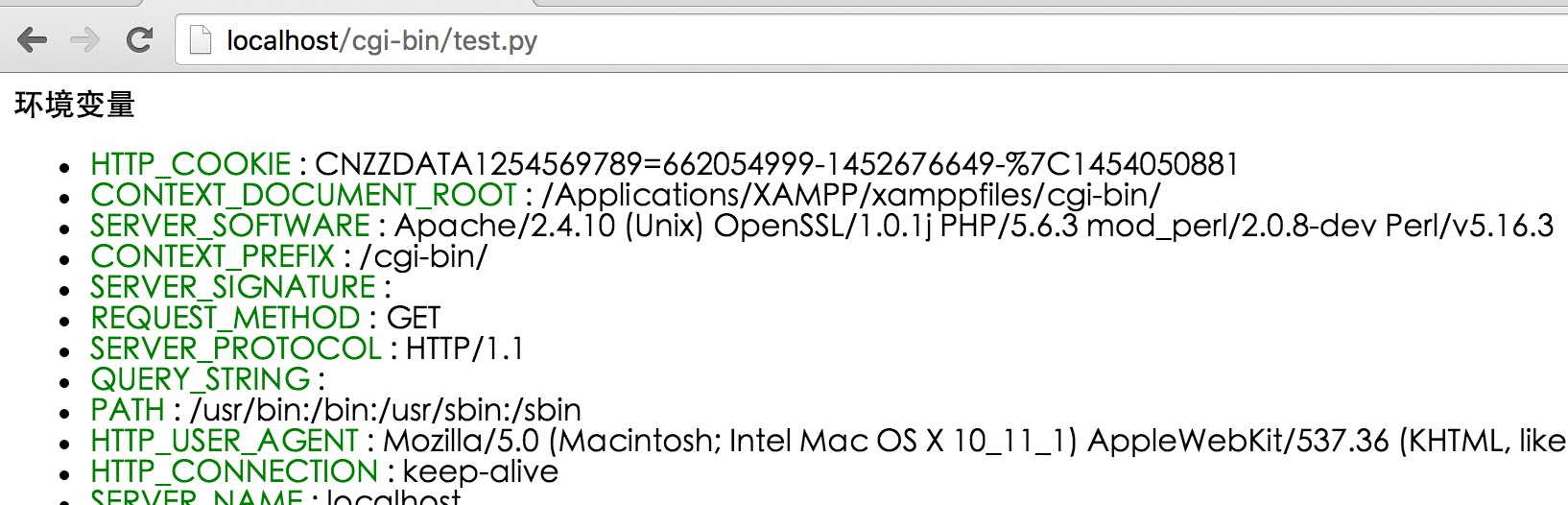
print ("<ul>")

for key in os.environ.keys():

print ("<li><span style='color:green'>%30s </span> : %s </li>" % (key,os.environ[key]))

print ("</ul>")

将以上点保存为 test.py ,并修改文件权限为 755，执行结果如下：



## 使用GET方法传输数据

GET方法发送编码后的用户信息到服务端，数据信息包含在请求页面的URL上，以"?"号分割,

#!/usr/bin/python3

# CGI处理模块

import cgi, cgitb

# 创建 FieldStorage 的实例化

form = cgi.FieldStorage()

# 获取数据

site\_name = form.getvalue('name')

site\_url = form.getvalue('url')

## 使用POST方法传递数据

使用POST方法向服务器传递数据是更安全可靠的，像一些敏感信息如用户密码等需要使用POST传输数据。

# CGI处理模块

import cgi, cgitb

# 创建 FieldStorage 的实例化

form = cgi.FieldStorage()

# 获取数据

site\_name = form.getvalue('name')

site\_url = form.getvalue('url')

## 通过CGI程序传递checkbox数据

# 引入 CGI 处理模块 import cgi, cgitb

# 创建 FieldStorage的实例

form = cgi.FieldStorage()

# 接收字段数据,如果选中了则不为假

if form.getvalue('google'):

google\_flag = "是"

else:

google\_flag = "否"

## 通过CGI程序传递Radio数据

# 引入 CGI 处理模块

import cgi, cgitb

# 创建 FieldStorage的实例

form = cgi.FieldStorage()

# 接收字段数据

if form.getvalue('site'):

site = form.getvalue('site')

else:

site = "提交数据为空"

## 通过CGI程序传递 Textarea 数据

# 接收字段数据

if form.getvalue('textcontent'):

text\_content = form.getvalue('textcontent')

else:

text\_content = "没有内容"

## 通过CGI程序传递下拉数据。

# 接收字段数据

if form.getvalue('dropdown'):

dropdown\_value = form.getvalue('dropdown')

else:

dropdown\_value = "没有内容"

## CGI中使用Cookie

### **cookie的语法**

http cookie的发送是通过http头部来实现的，他早于文件的传递，头部set-cookie的语法如下：

Set-cookie:name=name;expires=date;path=path;domain=domain;secure

* **name=name:** 需要设置cookie的值(name不能使用"**;**"和"**,**"号),有多个name值时用 "**;**" 分隔，例如：**name1=name1;name2=name2;name3=name3**。
* **expires=date:** cookie的有效期限,格式： expires="Wdy,DD-Mon-YYYY HH:MM:SS"
* **path=path:**设置cookie支持的路径,如果path是一个路径，则cookie对这个目录下的所有文件及子目录生效，例如： path="/cgi-bin/"，如果path是一个文件，则cookie指对这个文件生效，例如：path="/cgi-bin/cookie.cgi"。
* **domain=domain:** 对cookie生效的域名，例如：domain="www.runoob.com"
* **secure:** 如果给出此标志，表示cookie只能通过SSL协议的https服务器来传递。
* cookie的接收是通过设置环境变量HTTP\_COOKIE来实现的，CGI程序可以通过检索该变量获取cookie信息。

### Cookie设置

Cookie的设置非常简单，cookie会在http头部单独发送。以下实例在cookie中设置了name 和 expires：

#!/usr/bin/python3

print ('Content-Type: text/html')

print ('Set-Cookie: name="菜鸟教程";expires=Wed, 28 Aug 2016 18:30:00 GMT')

print ()

### 检索Cookie信息

Cookie信息检索页非常简单，Cookie信息存储在CGI的环境变量HTTP\_COOKIE中，存储格式如下：

key1=value1;key2=value2;key3=value3....

以下是一个简单的CGI检索cookie信息的程序：

#!/usr/bin/python3

# 导入模块

import os

import http.cookies

print ("Content-type: text/html")

print ()

print ("""

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h1>读取cookie信息</h1>

""")

if 'HTTP\_COOKIE' in os.environ:

cookie\_string=os.environ.get('HTTP\_COOKIE')

c= http.cookies.SimpleCookie()

# c=Cookie.SimpleCookie()

c.load(cookie\_string)

try:

data=c['name'].value

print ("cookie data: "+data+"<br>")

except KeyError:

print ("cookie 没有设置或者已过去<br>")

print ("""

</body>

</html>

""")

## 文件上传实例

HTML设置上传文件的表单需要设置 **enctype** 属性为 **multipart/form-data**，

save\_file.py脚本文件代码如下：

#!/usr/bin/python3

import cgi, os

import cgitb; cgitb.enable()

form = cgi.FieldStorage()

# 获取文件名

fileitem = form['filename']

# 检测文件是否上传

if fileitem.filename:

# 设置文件路径

fn = os.path.basename(fileitem.filename)

open('/tmp/' + fn, 'wb').write(fileitem.file.read())

message = '文件 "' + fn + '" 上传成功'

else:

message = '文件没有上传'

如果你使用的系统是Unix/Linux，你必须替换文件分隔符，在window下只需要使用open()语句即可：

fn = os.path.basename(fileitem.filename.replace("\\", "/" ))

## 文件下载对话框

我们先在当前目录下创建 foo.txt 文件，用于程序的下载。

文件下载通过设置HTTP头信息来实现，功能代码如下：

#!/usr/bin/python3

# HTTP 头部

print ("Content-Disposition: attachment; filename=\"foo.txt\"")

print ()# 打开文件

fo = open("foo.txt", "rb")

str = fo.read();

print (str)

# 关闭文件

fo.close()

# 3.Python MySQL - mysql-connector 驱动

使用 **pip** 命令来安装 **mysql-connector**：

python -m pip install mysql-connector

使用以下代码测试 mysql-connector 是否安装成功：

demo\_mysql\_test.py:

import mysql.connector

执行以上代码，如果没有产生错误，表明安装成功。

## 创建数据库连接

可以使用以下代码来连接数据库：

demo\_mysql\_test.py:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost", # 数据库主机地址

user="yourusername", # 数据库用户名

passwd="yourpassword" # 数据库密码

database=”databasesName” #需要连接的数据库名称)

mycursor = mydb.cursor() #获取数据库操作游标

print(mydb)

## 插入、更新、删除数据

sql = "INSERT INTO sites (name, url) VALUES (%s, %s)"

val = ("RUNOOB", "https://www.runoob.com")

mycursor.execute(sql, val)

mydb.commit() # 数据表内容有更新，必须使用到该语句

## 批量插入

批量插入使用 **executemany()** 方法，该方法的第二个参数是一个元组列表，包含了我们要插入的数据：

sql = "INSERT INTO sites (name, url) VALUES (%s, %s)"

val = [ ('Google', 'https://www.google.com'), ('Github', 'https://www.github.com'), ('Taobao', 'https://www.taobao.com'), ('stackoverflow', 'https://www.stackoverflow.com/') ] mycursor.executemany(sql, val)

mydb.commit() # 数据表内容有更新，必须使用到该语句

## 查询数据

mycursor.execute("SELECT \* FROM sites")

myresult = mycursor.fetchall() # fetchall() 获取所有记录

for x in myresult:

print(x)

如果我们只想读取一条数据，可以使用 **fetchone()** 方法：

为了防止数据库查询发生 SQL 注入的攻击，我们可以使用 **%s** 占位符来转义查询的条件：

# 4.Python3 MySQL 数据库连接 - PyMySQL 驱动

PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库，Python2中则使用mysqldb。

如果还未安装，我们可以使用以下命令安装最新版的 PyMySQL：

$ pip3 install PyMySQL

以下实例链接 Mysql 的 TESTDB 数据库：

## 实例(Python 3.0+)

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用 cursor() 方法创建一个游标对象 cursor

cursor = db.cursor()

# 使用 execute() 方法执行 SQL 查询

cursor.execute("SELECT VERSION()")

# 使用 fetchone() 方法获取单条数据.

data = cursor.fetchone()

print ("Database version : %s " % data)

# 关闭数据库连接

db.close()

try:

# 执行sql语句

cursor.execute(sql)

# 执行sql语句

db.commit()

except:

# 发生错误时回滚

db.rollback()

## 数据库查询操作

Python查询Mysql使用 fetchone() 方法获取单条数据, 使用fetchall() 方法获取多条数据。

* **fetchone():** 该方法获取下一个查询结果集。结果集是一个对象
* **fetchall():**接收全部的返回结果行.
* **rowcount:** 这是一个只读属性，并返回执行execute()方法后影响的行数。

# 5.Python3 网络编程

Python 提供了两个级别访问的网络服务。：

* 低级别的网络服务支持基本的 Socket，它提供了标准的 BSD Sockets API，可以访问底层操作系统Socket接口的全部方法。
* 高级别的网络服务模块 SocketServer， 它提供了服务器中心类，可以简化网络服务器的开发。

## socket()函数

Python 中，我们用 socket（）函数来创建套接字，语法格式如下：

socket.socket([family[, type[, proto]]])

参数

* family: 套接字家族可以使AF\_UNIX或者AF\_INET
* type: 套接字类型可以根据是面向连接的还是非连接分为SOCK\_STREAM或SOCK\_DGRAM
* protocol: 一般不填默认为0.

## Socket 对象(内建)方法

| **函数** | **描述** |
| --- | --- |
| 服务器端套接字 | |
| s.bind() | 绑定地址（host,port）到套接字， 在AF\_INET下,以元组（host,port）的形式表示地址。 |
| s.listen() | 开始TCP监听。backlog指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为1，大部分应用程序设为5就可以了。 |
| s.accept() | 被动接受TCP客户端连接,(阻塞式)等待连接的到来 |
| 客户端套接字 | |
| s.connect() | 主动初始化TCP服务器连接，。一般address的格式为元组（hostname,port），如果连接出错，返回socket.error错误。 |
| s.connect\_ex() | connect()函数的扩展版本,出错时返回出错码,而不是抛出异常 |
| 公共用途的套接字函数 | |
| s.recv() | 接收TCP数据，数据以字符串形式返回，bufsize指定要接收的最大数据量。flag提供有关消息的其他信息，通常可以忽略。 |
| s.send() | 发送TCP数据，将string中的数据发送到连接的套接字。返回值是要发送的字节数量，该数量可能小于string的字节大小。 |
| s.sendall() | 完整发送TCP数据，完整发送TCP数据。将string中的数据发送到连接的套接字，但在返回之前会尝试发送所有数据。成功返回None，失败则抛出异常。 |
| s.recvfrom() | 接收UDP数据，与recv()类似，但返回值是（data,address）。其中data是包含接收数据的字符串，address是发送数据的套接字地址。 |
| s.sendto() | 发送UDP数据，将数据发送到套接字，address是形式为（ipaddr，port）的元组，指定远程地址。返回值是发送的字节数。 |
| s.close() | 关闭套接字 |
| s.getpeername() | 返回连接套接字的远程地址。返回值通常是元组（ipaddr,port）。 |
| s.getsockname() | 返回套接字自己的地址。通常是一个元组(ipaddr,port) |
| s.setsockopt(level,optname,value) | 设置给定套接字选项的值。 |
| s.getsockopt(level,optname[.buflen]) | 返回套接字选项的值。 |
| s.settimeout(timeout) | 设置套接字操作的超时期，timeout是一个浮点数，单位是秒。值为None表示没有超时期。一般，超时期应该在刚创建套接字时设置，因为它们可能用于连接的操作（如connect()） |
| s.gettimeout() | 返回当前超时期的值，单位是秒，如果没有设置超时期，则返回None。 |
| s.fileno() | 返回套接字的文件描述符。 |
| s.setblocking(flag) | 如果flag为0，则将套接字设为非阻塞模式，否则将套接字设为阻塞模式（默认值）。非阻塞模式下，如果调用recv()没有发现任何数据，或send()调用无法立即发送数据，那么将引起socket.error异常。 |
| s.makefile() | 创建一个与该套接字相关连的文件 |

## 简单实例

### **服务端**

我们使用 socket 模块的 **socket** 函数来创建一个 socket 对象。socket 对象可以通过调用其他函数来设置一个 socket 服务。

现在我们可以通过调用 **bind(hostname, port)** 函数来指定服务的 *port(端口)*。

接着，我们调用 socket 对象的 *accept* 方法。该方法等待客户端的连接，并返回 *connection* 对象，表示已连接到客户端。

完整代码如下：

#!/usr/bin/python3# 文件名：server.py

# 导入 socket、sys 模块import socketimport sys

# 创建 socket 对象

serversocket = socket.socket(

socket.AF\_INET, socket.SOCK\_STREAM)

# 获取本地主机名

host = socket.gethostname()

port = 9999

# 绑定端口号

serversocket.bind((host, port))

# 设置最大连接数，超过后排队

serversocket.listen(5)

while True:

# 建立客户端连接

clientsocket,addr = serversocket.accept()

print("连接地址: %s" % str(addr))

msg='欢迎访问菜鸟教程！'+ "\r\n"

clientsocket.send(msg.encode('utf-8'))

clientsocket.close()

### **客户端**

接下来我们写一个简单的客户端实例连接到以上创建的服务。端口号为 9999。

**socket.connect(hosname, port )** 方法打开一个 TCP 连接到主机为 *hostname* 端口为 *port* 的服务商。连接后我们就可以从服务端获取数据，记住，操作完成后需要关闭连接。

完整代码如下：

#!/usr/bin/python3# 文件名：client.py

# 导入 socket、sys 模块import socketimport sys

# 创建 socket 对象

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# 获取本地主机名

host = socket.gethostname()

# 设置端口号

port = 9999

# 连接服务，指定主机和端口

s.connect((host, port))

# 接收小于 1024 字节的数据

msg = s.recv(1024)

s.close()

print (msg.decode('utf-8'))

现在我们打开两个终端，第一个终端执行 server.py 文件：

$ python3 server.py

第二个终端执行 client.py 文件：

$ python3 client.py 欢迎访问菜鸟教程！

这时我们再打开第一个终端，就会看到有以下信息输出：

连接地址： ('192.168.0.118', 33397)

## Python Internet 模块

以下列出了 Python 网络编程的一些重要模块：

|  |  |  |  |
| --- | --- | --- | --- |
| **协议** | **功能用处** | **端口号** | **Python 模块** |
| HTTP | 网页访问 | 80 | httplib, urllib, xmlrpclib |
| NNTP | 阅读和张贴新闻文章，俗称为"帖子" | 119 | nntplib |
| FTP | 文件传输 | 20 | ftplib, urllib |
| SMTP | 发送邮件 | 25 | smtplib |
| POP3 | 接收邮件 | 110 | poplib |
| IMAP4 | 获取邮件 | 143 | imaplib |
| Telnet | 命令行 | 23 | telnetlib |
| Gopher | 信息查找 | 70 | gopherlib, urllib |

# 6.Python3 SMTP发送邮件

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式。

python的smtplib提供了一种很方便的途径发送电子邮件。它对smtp协议进行了简单的封装。

Python创建 SMTP 对象语法如下：

import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local\_hostname]]] )

参数说明：

* host: SMTP 服务器主机。 你可以指定主机的ip地址或者域名如:runoob.com，这个是可选参数。
* port: 如果你提供了 host 参数, 你需要指定 SMTP 服务使用的端口号，一般情况下SMTP端口号为25。
* local\_hostname: 如果SMTP在你的本机上，你只需要指定服务器地址为 localhost 即可。

Python SMTP对象使用sendmail方法发送邮件，语法如下：

SMTP.sendmail(from\_addr, to\_addrs, msg[, mail\_options, rcpt\_options]

参数说明：

* from\_addr: 邮件发送者地址。
* to\_addrs: 字符串列表，邮件发送地址。
* msg: 发送消息

这里要注意一下第三个参数，msg是字符串，表示邮件。我们知道邮件一般由标题，发信人，收件人，邮件内容，附件等构成，发送邮件的时候，要注意msg的格式。这个格式就是smtp协议中定义的格式。

### **实例**

以下是一个使用Python发送邮件简单的实例：

实例

#!/usr/bin/python3

import smtplib from email.mime.text

import MIMEText from email.header

import Header

sender = 'from@runoob.com'

receivers = ['429240967@qq.com'] # 接收邮件，可设置为你的QQ邮箱或者其他邮箱

# 三个参数：第一个为文本内容，第二个 plain 设置文本格式，第三个 utf-8 设置编码

message = MIMEText('Python 邮件发送测试...', 'plain', 'utf-8')

message['From'] = Header("菜鸟教程", 'utf-8') # 发送者

message['To'] = Header("测试", 'utf-8') # 接收者

subject = 'Python SMTP 邮件测试'

message['Subject'] = Header(subject, 'utf-8')

try: smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, receivers, message.as\_string())

print ("邮件发送成功")

except smtplib.SMTPException:

print ("Error: 无法发送邮件")

我们使用三个引号来设置邮件信息，标准邮件需要三个头部信息： **From**, **To**, 和 **Subject** ，每个信息直接使用空行分割。

我们通过实例化 smtplib 模块的 SMTP 对象 *smtpObj* 来连接到 SMTP 访问，并使用 *sendmail* 方法来发送信息。

执行以上程序，如果你本机安装sendmail，就会输出：

$ python3 test.py 邮件发送成功

如果我们本机没有 sendmail 访问，也可以使用其他服务商的 SMTP 访问（QQ、网易、Google等）。

实例

#!/usr/bin/python3

import smtplib from email.mime.text

import MIMEText from email.header

import Header

# 第三方 SMTP 服务

mail\_host="smtp.XXX.com" #设置服务器

mail\_user="XXXX" #用户名

mail\_pass="XXXXXX" #口令

sender = 'from@runoob.com'

receivers = ['429240967@qq.com'] # 接收邮件，可设置为你的QQ邮箱或者其他邮箱

message = MIMEText('Python 邮件发送测试...', 'plain', 'utf-8')

message['From'] = Header("菜鸟教程", 'utf-8')

message['To'] = Header("测试", 'utf-8')

subject = 'Python SMTP 邮件测试'

message['Subject'] = Header(subject, 'utf-8')

try:

smtpObj = smtplib.SMTP()

smtpObj.connect(mail\_host, 25) # 25 为 SMTP 端口号

smtpObj.login(mail\_user,mail\_pass)

smtpObj.sendmail(sender, receivers, message.as\_string())

print ("邮件发送成功")

except smtplib.SMTPException:

print ("Error: 无法发送邮件")

## 使用Python发送HTML格式的邮件

Python发送HTML格式的邮件与发送纯文本消息的邮件不同之处就是将MIMEText中\_subtype设置为html。具体代码如下：

实例

#!/usr/bin/python3

import smtplib from email.mime.text

import MIMEText from email.header

import Header sender = 'from@runoob.com'

receivers = ['429240967@qq.com'] # 接收邮件，可设置为你的QQ邮箱或者其他邮箱

mail\_msg = """ <p>Python 邮件发送测试...</p> <p><a href="http://www.runoob.com">这是一个链接</a></p> """

message = MIMEText(mail\_msg, 'html', 'utf-8')

message['From'] = Header("菜鸟教程", 'utf-8')

message['To'] = Header("测试", 'utf-8')

subject = 'Python SMTP 邮件测试'

message['Subject'] = Header(subject, 'utf-8')

try:

smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, receivers, message.as\_string())

print ("邮件发送成功")

except smtplib.SMTPException:

print ("Error: 无法发送邮件")

执行以上程序，如果你本机安装sendmail，就会输出：

$ python3 test.py 邮件发送成功

## Python 发送带附件的邮件

发送带附件的邮件，首先要创建MIMEMultipart()实例，然后构造附件，如果有多个附件，可依次构造，最后利用smtplib.smtp发送。

实例

#!/usr/bin/python3

import smtplib from email.mime.text

import MIMEText from email.mime.multipart

import MIMEMultipart from email.header

import Header

sender = 'from@runoob.com'

receivers = ['429240967@qq.com'] # 接收邮件，可设置为你的QQ邮箱或者其他邮箱

#创建一个带附件的实例

message = MIMEMultipart()

message['From'] = Header("菜鸟教程", 'utf-8')

message['To'] = Header("测试", 'utf-8')

subject = 'Python SMTP 邮件测试'

message['Subject'] = Header(subject, 'utf-8')

#邮件正文内容

message.attach(MIMEText('这是菜鸟教程Python 邮件发送测试……', 'plain', 'utf-8'))

# 构造附件1，传送当前目录下的 test.txt 文件

att1 = MIMEText(open('test.txt', 'rb').read(), 'base64', 'utf-8')

att1["Content-Type"] = 'application/octet-stream'

# 这里的filename可以任意写，写什么名字，邮件中显示什么名字

att1["Content-Disposition"] = 'attachment; filename="test.txt"'

message.attach(att1)

# 构造附件2，传送当前目录下的 runoob.txt 文件

att2 = MIMEText(open('runoob.txt', 'rb').read(), 'base64', 'utf-8')

att2["Content-Type"] = 'application/octet-stream'

att2["Content-Disposition"] = 'attachment; filename="runoob.txt"'

message.attach(att2)

try:

smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, receivers, message.as\_string())

print ("邮件发送成功")

except smtplib.SMTPException:

print ("Error: 无法发送邮件")

$ python3 test.py 邮件发送成功

## 在 HTML 文本中添加图片

邮件的 HTML 文本中一般邮件服务商添加外链是无效的，正确添加图片的实例如下所示：

实例

#!/usr/bin/python3

import smtplib from email.mime.image

import MIMEImage from email.mime.multipart

import MIMEMultipart from email.mime.text

import MIMEText from email.header

import Header

sender = 'from@runoob.com'

receivers = ['429240967@qq.com'] # 接收邮件，可设置为你的QQ邮箱或者其他邮箱

msgRoot = MIMEMultipart('related')

msgRoot['From'] = Header("菜鸟教程", 'utf-8')

msgRoot['To'] = Header("测试", 'utf-8')

subject = 'Python SMTP 邮件测试'

msgRoot['Subject'] = Header(subject, 'utf-8')

msgAlternative = MIMEMultipart('alternative')

msgRoot.attach(msgAlternative)

mail\_msg = """ <p>Python 邮件发送测试...</p> <p><a href="http://www.runoob.com">菜鸟教程链接</a></p> <p>图片演示：</p> <p><img src="cid:image1"></p> """ msgAlternative.attach(MIMEText(mail\_msg, 'html', 'utf-8'))

# 指定图片为当前目录

fp = open('test.png', 'rb')

msgImage = MIMEImage(fp.read())

fp.close()

# 定义图片 ID，在 HTML 文本中引用

msgImage.add\_header('Content-ID', '<image1>')

msgRoot.attach(msgImage)

try:

smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, receivers, msgRoot.as\_string())

print ("邮件发送成功")

except smtplib.SMTPException:

print ("Error: 无法发送邮件")

$ python3 test.py 邮件发送成功

## 使用第三方 SMTP 服务发送(推荐)

这里使用了 QQ 邮箱(你也可以使用 163，Gmail等)的 SMTP 服务，需要做以下配置：



QQ 邮箱通过生成授权码来设置密码：



QQ 邮箱 SMTP 服务器地址：smtp.qq.com，ssl 端口：465。

以下实例你需要修改：发件人邮箱（你的QQ邮箱），密码，收件人邮箱（可发给自己）。

QQ SMTP

#!/usr/bin/python3

import smtplib from email.mime.text

import MIMEText from email.utils

import formataddr my\_sender='429240967@qq.com' # 发件人邮箱账号

my\_pass = 'xxxxxxxxxx' # 发件人邮箱密码

my\_user='429240967@qq.com' # 收件人邮箱账号，我这边发送给自己

def mail():

ret=True

try:

msg=MIMEText('填写邮件内容','plain','utf-8')

msg['From']=formataddr(["FromRunoob",my\_sender]) # 括号里的对应发件人邮箱昵称、发件人邮箱账号

msg['To']=formataddr(["FK",my\_user]) # 括号里的对应收件人邮箱昵称、收件人邮箱账号

msg['Subject']="菜鸟教程发送邮件测试" # 邮件的主题，也可以说是标题

server=smtplib.SMTP\_SSL("smtp.qq.com", 465) # 发件人邮箱中的SMTP服务器，端口是25

server.login(my\_sender, my\_pass) # 括号中对应的是发件人邮箱账号、邮箱密码 server.sendmail(my\_sender,[my\_user,],msg.as\_string()) # 括号中对应的是发件人邮箱账号、收件人邮箱账号、发送邮件

server.quit() # 关闭连接

except Exception: # 如果 try 中的语句没有执行，则会执行下面的 ret=False

ret=False

return ret

ret=mail()

if ret:

print("邮件发送成功")

else:

print("邮件发送失败")

$ python test.py 邮件发送成功

# 7.Python3 多线程

Python3 线程中常用的两个模块为：

* **\_thread**
* **threading(推荐使用)**

thread 模块已被废弃。用户可以使用 threading 模块代替。所以，在 Python3 中不能再使用"thread" 模块。为了兼容性，Python3 将 thread 重命名为 "\_thread"。

## 创建Python线程

Python中使用线程有两种方式：函数或者用类来包装线程对象。

函数式：调用 \_thread 模块中的start\_new\_thread()函数来产生新线程。语法如下:

\_thread.start\_new\_thread ( function, args[, kwargs] )

参数说明:

* function - 线程函数。
* args - 传递给线程函数的参数,他必须是个tuple类型。
* kwargs - 可选参数。

实例：

#!/usr/bin/python3

import \_threadimport time

# 为线程定义一个函数def print\_time( threadName, delay):

count = 0

while count < 5:

time.sleep(delay)

count += 1

print ("%s: %s" % ( threadName, time.ctime(time.time()) ))

# 创建两个线程try:

\_thread.start\_new\_thread( print\_time, ("Thread-1", 2, ) )

\_thread.start\_new\_thread( print\_time, ("Thread-2", 4, ) )except:

print ("Error: 无法启动线程")

while 1:

pass

## 线程模块

Python3 通过两个标准库 \_thread 和 threading 提供对线程的支持。

\_thread 提供了低级别的、原始的线程以及一个简单的锁，它相比于 threading 模块的功能还是比较有限的。

threading 模块除了包含 \_thread 模块中的所有方法外，还提供的其他方法：

* threading.currentThread(): 返回当前的线程变量。
* threading.enumerate(): 返回一个包含正在运行的线程的list。正在运行指线程启动后、结束前，不包括启动前和终止后的线程。
* threading.activeCount(): 返回正在运行的线程数量，与len(threading.enumerate())有相同的结果。

除了使用方法外，线程模块同样提供了Thread类来处理线程，Thread类提供了以下方法:

* **run():** 用以表示线程活动的方法。
* **start():**启动线程活动。
* **join([time]):** 等待至线程中止。这阻塞调用线程直至线程的join() 方法被调用中止-正常退出或者抛出未处理的异常-或者是可选的超时发生。
* **isAlive():** 返回线程是否活动的。
* **getName():** 返回线程名。
* **setName():** 设置线程名。

## 使用 threading 模块创建线程

我们可以通过直接从 threading.Thread 继承创建一个新的子类，并实例化后调用 start() 方法启动新线程，即它调用了线程的 run() 方法：

#!/usr/bin/python3

import threadingimport time

exitFlag = 0

class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print ("开始线程：" + self.name)

print\_time(self.name, self.counter, 5)

print ("退出线程：" + self.name)

def print\_time(threadName, delay, counter):

while counter:

if exitFlag:

threadName.exit()

time.sleep(delay)

print ("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

# 创建新线程

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

# 开启新线程

thread1.start()

thread2.start()

thread1.join()

thread2.join()print ("退出主线程")

## 线程同步(线程锁)

如果多个线程共同对某个数据修改，则可能出现不可预料的结果，为了保证数据的正确性，需要对多个线程进行同步。

使用 Thread 对象的 Lock 和 Rlock 可以实现简单的线程同步，这两个对象都有 acquire 方法和 release 方法，对于那些需要每次只允许一个线程操作的数据，可以将其操作放到 acquire 和 release 方法之间。如下：

实例：

#!/usr/bin/python3

import threadingimport time

class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print ("开启线程： " + self.name)

# 获取锁，用于线程同步

threadLock.acquire()

print\_time(self.name, self.counter, 3)

# 释放锁，开启下一个线程

threadLock.release()

def print\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print ("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

threadLock = threading.Lock()

threads = []

# 创建新线程

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

# 开启新线程

thread1.start()

thread2.start()

# 添加线程到线程列表

threads.append(thread1)

threads.append(thread2)

# 等待所有线程完成for t in threads:

t.join()print ("退出主线程")

执行以上程序，输出结果为：

开启线程： Thread-1

开启线程： Thread-2

Thread-1: Wed Apr 6 11:52:57 2016

Thread-1: Wed Apr 6 11:52:58 2016

Thread-1: Wed Apr 6 11:52:59 2016

Thread-2: Wed Apr 6 11:53:01 2016

Thread-2: Wed Apr 6 11:53:03 2016

Thread-2: Wed Apr 6 11:53:05 2016

退出主线程

## 线程优先级队列（ Queue）

Python 的 Queue 模块中提供了同步的、线程安全的队列类，包括FIFO（先入先出)队列Queue，LIFO（后入先出）队列LifoQueue，和优先级队列 PriorityQueue。

这些队列都实现了锁原语，能够在多线程中直接使用，可以使用队列来实现线程间的同步。

Queue 模块中的常用方法:

* Queue.qsize() 返回队列的大小
* Queue.empty() 如果队列为空，返回True,反之False
* Queue.full() 如果队列满了，返回True,反之False
* Queue.full 与 maxsize 大小对应
* Queue.get([block[, timeout]])获取队列，timeout等待时间
* Queue.get\_nowait() 相当Queue.get(False)
* Queue.put(item) 写入队列，timeout等待时间
* Queue.put\_nowait(item) 相当Queue.put(item, False)
* Queue.task\_done() 在完成一项工作之后，Queue.task\_done()函数向任务已经完成的队列发送一个信号
* Queue.join() 实际上意味着等到队列为空，再执行别的操作

实例:

#!/usr/bin/python3

import queueimport threadingimport time

exitFlag = 0

class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, q):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.q = q

def run(self):

print ("开启线程：" + self.name)

process\_data(self.name, self.q)

print ("退出线程：" + self.name)

def process\_data(threadName, q):

while not exitFlag:

queueLock.acquire()

if not workQueue.empty():

data = q.get()

queueLock.release()

print ("%s processing %s" % (threadName, data))

else:

queueLock.release()

time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]

nameList = ["One", "Two", "Three", "Four", "Five"]

queueLock = threading.Lock()

workQueue = queue.Queue(10)

threads = []

threadID = 1

# 创建新线程for tName in threadList:

thread = myThread(threadID, tName, workQueue)

thread.start()

threads.append(thread)

threadID += 1

# 填充队列

queueLock.acquire()for word in nameList:

workQueue.put(word)

queueLock.release()

# 等待队列清空while not workQueue.empty():

pass

# 通知线程是时候退出

exitFlag = 1

# 等待所有线程完成for t in threads:

t.join()print ("退出主线程")

以上程序执行结果：

开启线程：Thread-1

开启线程：Thread-2

开启线程：Thread-3

Thread-3 processing One

Thread-1 processing Two

Thread-2 processing Three

Thread-3 processing Four

Thread-1 processing Five

退出线程：Thread-3

退出线程：Thread-2

退出线程：Thread-1

退出主线程

# 8.Python3 JSON 数据解析

JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于ECMAScript的一个子集。

Python3 中可以使用 json 模块来对 JSON 数据进行编解码，它包含了两个函数：

* **json.dumps():**对数据进行编码。
* **json.loads():**对数据进行解码。

在json的编解码过程中，python 的原始类型与json类型会相互转换，具体的转化对照如下：

## Python 编码为 JSON 类型转换对应表：

| **Python** | **JSON** |
| --- | --- |
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |

## JSON 解码为 Python 类型转换对应表：

| **JSON** | **Python** |
| --- | --- |
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

## json.dumps 与 json.loads 实例

实例(Python 3.0+)

#!/usr/bin/python3

import json

# Python 字典类型转换为 JSON 对象

data1 = { 'no' : 1, 'name' : 'Runoob', 'url' : 'http://www.runoob.com' }

json\_str = json.dumps(data1)

print ("Python 原始数据：", repr(data1))

print ("JSON 对象：", json\_str)

# 将 JSON 对象转换为 Python 字典

data2 = json.loads(json\_str)

print ("data2['name']: ", data2['name'])

print ("data2['url']: ", data2['url'])

执行以上代码输出结果为：

Python 原始数据： {'name': 'Runoob', 'no': 1, 'url': 'http://www.runoob.com'}

JSON 对象： {"name": "Runoob", "no": 1, "url": "http://www.runoob.com"}

data2['name']: Runoob

data2['url']: http://www.runoob.com

## 处理文件

如果你要处理的是文件而不是字符串，你可以使用 **json.dump()** 和 **json.load()** 来编码和解码JSON数据。例如：

实例(Python 3.0+)

# 写入 JSON 数据

with open('data.json', 'w') as f:

json.dump(data, f)

# 读取数据

with open('data.json', 'r') as f:

data = json.load(f)

# 9.Python3 日期和时间

Python 提供了一个 time 和 calendar 模块可以用于格式化日期和时间。

## 时间元组？

很多Python函数用一个元组装起来的9组数字处理时间:

|  |  |  |
| --- | --- | --- |
| **序号** | **字段** | **值** |
| 0 | 4位数年 | 2008 |
| 1 | 月 | 1 到 12 |
| 2 | 日 | 1到31 |
| 3 | 小时 | 0到23 |
| 4 | 分钟 | 0到59 |
| 5 | 秒 | 0到61 (60或61 是闰秒) |
| 6 | 一周的第几日 | 0到6 (0是周一) |
| 7 | 一年的第几日 | 1到366 (儒略历) |
| 8 | 夏令时 | -1, 0, 1, -1是决定是否为夏令时的旗帜 |

上述也就是struct\_time元组。这种结构具有如下属性：

|  |  |  |
| --- | --- | --- |
| **序号** | **属性** | **值** |
| 0 | tm\_year | 2008 |
| 1 | tm\_mon | 1 到 12 |
| 2 | tm\_mday | 1 到 31 |
| 3 | tm\_hour | 0 到 23 |
| 4 | tm\_min | 0 到 59 |
| 5 | tm\_sec | 0 到 61 (60或61 是闰秒) |
| 6 | tm\_wday | 0到6 (0是周一) |
| 7 | tm\_yday | 一年中的第几天，1 到 366 |
| 8 | tm\_isdst | 是否为夏令时，值有：1(夏令时)、0(不是夏令时)、-1(未知)，默认 -1 |

## 获取当前时间

从返回浮点数的时间戳方式向时间元组转换，只要将浮点数传递给如localtime之类的函数。

#!/usr/bin/python3

import time

localtime = time.localtime(time.time())

print ("本地时间为 :", localtime)

以上实例输出结果：

本地时间为 : time.struct\_time(tm\_year=2016, tm\_mon=4, tm\_mday=7, tm\_hour=10, tm\_min=28, tm\_sec=49, tm\_wday=3, tm\_yday=98, tm\_isdst=0)

## 获取格式化的时间

你可以根据需求选取各种格式，但是最简单的获取可读的时间模式的函数是asctime():

#!/usr/bin/python3

import time

localtime = time.asctime( time.localtime(time.time()) )

print ("本地时间为 :", localtime)

以上实例输出结果：

本地时间为 : Thu Apr 7 10:29:13 2016

## 格式化日期

我们可以使用 time 模块的 strftime 方法来格式化日期，：

#!/usr/bin/python3

import time

# 格式化成2016-03-20 11:45:39形式

print (time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))

# 格式化成Sat Mar 28 22:24:24 2016形式

print (time.strftime("%a %b %d %H:%M:%S %Y", time.localtime()))

# 将格式字符串转换为时间戳

a = "Sat Mar 28 22:24:24 2016"

print (time.mktime(time.strptime(a,"%a %b %d %H:%M:%S %Y")))

以上实例输出结果：

2016-04-07 10:29:46

Thu Apr 07 10:29:46

20161459175064.0

## python中时间日期格式化符号：

* %y 两位数的年份表示（00-99）
* %Y 四位数的年份表示（000-9999）
* %m 月份（01-12）
* %d 月内中的一天（0-31）
* %H 24小时制小时数（0-23）
* %I 12小时制小时数（01-12）
* %M 分钟数（00=59）
* %S 秒（00-59）
* %a 本地简化星期名称
* %A 本地完整星期名称
* %b 本地简化的月份名称
* %B 本地完整的月份名称
* %c 本地相应的日期表示和时间表示
* %j 年内的一天（001-366）
* %p 本地A.M.或P.M.的等价符
* %U 一年中的星期数（00-53）星期天为星期的开始
* %w 星期（0-6），星期天为星期的开始
* %W 一年中的星期数（00-53）星期一为星期的开始
* %x 本地相应的日期表示
* %X 本地相应的时间表示
* %Z 当前时区的名称
* %% %号本身

## 获取某月日历

Calendar模块有很广泛的方法用来处理年历和月历，例如打印某月的月历：

#!/usr/bin/python3

import calendar

cal = calendar.month(2016, 1)print ("以下输出2016年1月份的日历:")print (cal)

以上实例输出结果：



## Time 模块

Time 模块包含了以下内置函数，既有时间处理的，也有转换时间格式的：

|  |  |
| --- | --- |
| **序号** | **函数及描述** |
| 1 | time.altzone 返回格林威治西部的夏令时地区的偏移秒数。如果该地区在格林威治东部会返回负值（如西欧，包括英国）。对夏令时启用地区才能使用。 |
| 2 | time.asctime([tupletime]) 接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2008"（2008年12月11日 周二18时07分14秒）的24个字符的字符串。 |
| 3 | [time.clock()](https://www.runoob.com/python3/python3-att-time-clock.html" \t "/home/zgj/文档\\x/_blank) 用以浮点数计算的秒数返回当前的CPU时间。用来衡量不同程序的耗时，比time.time()更有用。 |
| 4 | time.ctime([secs]) 作用相当于asctime(localtime(secs))，未给参数相当于asctime() |
| 5 | time.gmtime([secs]) 接收时间戳（1970纪元后经过的浮点秒数）并返回格林威治天文时间下的时间元组t。注：t.tm\_isdst始终为0 |
| 6 | time.localtime([secs] 接收时间戳（1970纪元后经过的浮点秒数）并返回当地时间下的时间元组t（t.tm\_isdst可取0或1，取决于当地当时是不是夏令时）。 |
| 7 | [time.mktime(tupletime)](https://www.runoob.com/python3/python3-att-time-mktime.html" \t "/home/zgj/文档\\x/_blank) 接受时间元组并返回时间戳（1970纪元后经过的浮点秒数）。 |
| 8 | time.sleep(secs) 推迟调用线程的运行，secs指秒数。 |
| 9 | time.strftime(fmt[,tupletime]) 接收以时间元组，并返回以可读字符串表示的当地时间，格式由fmt决定。 |
| 10 | time.strptime(str,fmt='%a %b %d %H:%M:%S %Y') 根据fmt的格式把一个时间字符串解析为时间元组。 |
| 11 | time.time( ) 返回当前时间的时间戳（1970纪元后经过的浮点秒数）。 |
| 12 | [time.tzset()](https://www.runoob.com/python3/python3-att-time-tzset.html" \t "/home/zgj/文档\\x/_blank) 根据环境变量TZ重新初始化时间相关设置。 |
| 13 | **time.perf\_counter()** 返回计时器的精准时间（系统的运行时间），包含整个系统的睡眠时间。由于返回值的基准点是未定义的，所以，只有连续调用的结果之间的差才是有效的。 |
| 14 | **time.process\_time()** 返回当前进程执行 CPU 的时间总和，不包含睡眠时间。由于返回值的基准点是未定义的，所以，只有连续调用的结果之间的差才是有效的。 |

Time模块包含了以下2个非常重要的属性：

|  |  |
| --- | --- |
| **序号** | **属性及描述** |
| 1 | **time.timezone** 属性time.timezone是当地时区（未启动夏令时）距离格林威治的偏移秒数（>0，美洲;<=0大部分欧洲，亚洲，非洲）。 |
| 2 | **time.tzname** 属性time.tzname包含一对根据情况的不同而不同的字符串，分别是带夏令时的本地时区名称，和不带的。 |

## 日历（Calendar）模块

此模块的函数都是日历相关的，例如打印某月的字符月历。

星期一是默认的每周第一天，星期天是默认的最后一天。更改设置需调用calendar.setfirstweekday()函数。模块包含了以下内置函数：

|  |  |
| --- | --- |
| **序号** | **函数及描述** |
| 1 | **calendar.calendar(year,w=2,l=1,c=6)** 返回一个多行字符串格式的year年年历，3个月一行，间隔距离为c。 每日宽度间隔为w字符。每行长度为21\* W+18+2\* C。l是每星期行数。 |
| 2 | **calendar.firstweekday( )** 返回当前每周起始日期的设置。默认情况下，首次载入caendar模块时返回0，即星期一。 |
| 3 | **calendar.isleap(year)**  是闰年返回 True，否则为 false。  >>> import calendar>>> print(calendar.isleap(2000))True>>> print(calendar.isleap(1900))False |
| 4 | **calendar.leapdays(y1,y2)** 返回在Y1，Y2两年之间的闰年总数。 |
| 5 | **calendar.month(year,month,w=2,l=1)** 返回一个多行字符串格式的year年month月日历，两行标题，一周一行。每日宽度间隔为w字符。每行的长度为7\* w+6。l是每星期的行数。 |
| 6 | **calendar.monthcalendar(year,month)** 返回一个整数的单层嵌套列表。每个子列表装载代表一个星期的整数。Year年month月外的日期都设为0;范围内的日子都由该月第几日表示，从1开始。 |
| 7 | **calendar.monthrange(year,month)**  返回两个整数。第一个是该月的星期几，第二个是该月有几天。星期几是从0（星期一）到 6（星期日）。  >>> import calendar>>> calendar.monthrange(2014, 11)(5, 30)  (5, 30)解释：5 表示 2014 年 11 月份的第一天是周六，30 表示 2014 年 11 月份总共有 30 天。 |
| 8 | **calendar.prcal(year,w=2,l=1,c=6)** 相当于 print calendar.calendar(year,w,l,c). |
| 9 | **calendar.prmonth(year,month,w=2,l=1)** 相当于 print calendar.calendar（year，w，l，c）。 |
| 10 | **calendar.setfirstweekday(weekday)** 设置每周的起始日期码。0（星期一）到6（星期日）。 |
| 11 | **calendar.timegm(tupletime)** 和time.gmtime相反：接受一个时间元组形式，返回该时刻的时间戳（1970纪元后经过的浮点秒数）。 |
| 12 | **calendar.weekday(year,month,day)** 返回给定日期的日期码。0（星期一）到6（星期日）。月份为 1（一月） 到 12（12月）。 |

## 其他相关模块和函数

在Python中，其他处理日期和时间的模块还有：

* [time 模块](https://docs.python.org/3/library/time.html" \t "/home/zgj/文档\\x/_blank)
* [datetime模块](https://docs.python.org/3/library/datetime.html" \t "/home/zgj/文档\\x/_blank)

# Python3 内置函数

|  |  |  |
| --- | --- | --- |
| 函数名 | 简介 | 语法 |
| abs() | 求绝对值 | abs(x) |
| dict() | 创建字典 |  |
| help() | 用于查看函数或模块用途的详细说明。 | help([object])  object -- 对象； |
| min() | 返回给定参数的最小值，参数可以为序列。 | min(x,y,z) |
| setattr() | 对应函数 getattr()，用于设置属性值，该属性不一定是存在的。 | setattr(object, name, value)  object -- 对象。  name -- 字符串，对象属性。  value -- 属性值。 |
| all() | 用于判断给定的可迭代参数 iterable 中的所有元素是否都为 TRUE，如果是返回 True，否则返回 False。 | all(iterable)  iterable -- 元组或列表。 |
| dir() | 不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。 | dir([object])  object -- 对象、变量、类型。 |
| hex() | 用于将一个指定数字转换为 16 进制数。 | hex(x) |
| next() | 返回迭代器的下一个项目。 | next(iterator[, default])  iterator -- 可迭代对象  default -- 可选，用于设置在没有下一个元素时返回该默认值，如果不设置，又没有下一个元素则会触发 StopIteration 异常。 |
| slice() | 实现切片对象，主要用在切片操作函数里的参数传递。 | slice(start, stop[, step])  start -- 起始位置  stop -- 结束位置  step -- 间距 |
| any() | 判断给定的可迭代参数 iterable 是否全部为 False，则返回 False，如果有一个为 True，则返回 True。 | any(iterable)  iterable -- 元组或列表 |
| divmod() | 接收两个数字类型（非复数）参数，返回一个包含商和余数的元组(a // b, a % b)。 |  |
| id() | 用于获取对象的内存地址。 | id([object]) |
| sorted() | 对所有可迭代的对象进行排序操作。 | sorted(iterable, key=None, reverse=False)  iterable -- 可迭代对象。  key -- 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。  reverse -- 排序规则，reverse = True 降序 ， reverse = False 升序（默认）。 |
| ascii() | ascii() 函数类似 repr() 函数, 返回一个表示对象的字符串, 但是对于字符串中的非 ASCII 字符则返回通过 repr() 函数使用 \x, \u 或 \U 编码的字符。 | ascii(object) |
| enumerate() | 用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。 | enumerate(sequence, [start=0]) |
| input() | 接受一个标准输入数据，返回为 string 类型。 | input([prompt])  prompt: 提示信息 |
| oct() | 将一个整数转换成8进制字符串 | oct(x) |
| staticmethod | staticmethod 返回函数的静态方法。 | @staticmethod  def f(arg1, arg2, ...): |
| bin() | 返回一个整数 int 或者长整数 long int 的二进制表示。 | bin(x) |
| eval() | 用来执行一个字符串表达式，并返回表达式的值。 | eval(expression[, globals[, locals]])  expression -- 表达式。  globals -- 变量作用域，全局命名空间，如果被提供，则必须是一个字典对象。  locals -- 变量作用域，局部命名空间，如果被提供，可以是任何映射对象。 |
| int() | 用于将一个字符串或数字转换为整型。 | int(x, base=10)  x -- 字符串或数字。  base -- 进制数，默认十进制。  #如果是带参数base的话，x要以字符串的形式进行输入 |
| open() | 用于打开一个文件，并返回文件对象，在对文件进行处理过程都需要使用到这个函数，如果该文件无法被打开，会抛出 OSError。  注意：使用 open() 函数一定要保证关闭文件对象，即调用 close() 函数。 | open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)  file: 必需，文件路径（相对或者绝对路径）。  mode: 可选，文件打开模式  buffering: 设置缓冲  encoding: 一般使用utf8  errors: 报错级别  newline: 区分换行符  closefd: 传入的file参数类型 |
| str() | 将对象转化为适于人阅读的形式。 | str(object) |
| bool() | 用于将给定参数转换为布尔类型，如果没有参数，返回 False。 | bool(x) |
| exec() | 执行储存在字符串或文件中的 Python 语句，相比于 eval，exec可以执行更复杂的 Python 代码。 | exec(object[, globals[, locals]]) |
| isinstance() | 判断一个对象是否是一个已知的类型，类似 type()。  isinstance() 与 type() 别：  type() 不会认为子类是一种父类类型，不考虑继承关系。  isinstance() 会认为子类是一种父类类型，考虑继承。  如果要判断两个类型是否相同推荐使用 isinstance()。 | isinstance(object, classinfo)  object -- 实例对象。  classinfo -- 可以是直接或间接类名、基本类型或者由它们组成的元组。 |
| ord() | 它以一个字符串（Unicode 字符）作为参数，返回对应的 ASCII 数值，或者 Unicode 数值。 | ord(c)  c -- 字符。 |
| sum() | 对系列进行求和计算。 | sum(iterable[, start])  iterable -- 可迭代对象，如：列表、元组、集合。  start -- 指定相加的参数，如果没有设置这个值，默认为0。 |
| bytearray() | bytearray() 方法返回一个新字节数组。这个数组里的元素是可变的，并且每个元素的值范围: 0 <= x < 256。 |  |
| filter() | filter() 函数用于过滤序列，过滤掉不符合条件的元素，返回一个迭代器对象，如果要转换为列表，可以使用 list() 来转换。  该接收两个参数，第一个为函数，第二个为序列，序列的每个元素作为参数传递给函数进行判，然后返回 True 或 False，最后将返回 True 的元素放到新列表中。 | filter(function, iterable)  function -- 判断函数。  iterable -- 可迭代对象。 |
| issubclass() | 用于判断参数 class 是否是类型参数 classinfo 的子类。 | issubclass(class, classinfo)  class -- 类。  classinfo -- 类。 |
| pow() | 返回 xy（x的y次方） 的值。 | 以下是 math 模块 pow() 方法的语法:  import math  math.pow( x, y )  内置的 pow() 方法  pow(x, y[, z])  函数是计算x的y次方，如果z在存在，则再对结果进行取模，其结果等效于pow(x,y) %z  **注意：**pow() 通过内置的方法直接调用，内置方法会把参数作为整型，而 math 模块则会把参数转换为 float。 |
| super() | **super()** 函数是用于调用父类(超类)的一个方法。 |  |
| bytes() | 返回一个新的 bytes 对象，该对象是一个 0 <= x < 256 区间内的整数不可变序列。它是 bytearray 的不可变版本。 |  |
| float() | 用于将整数和字符串转换成浮点数。 | float(x) |
| iter() | 用来生成迭代器。 | iter(object[, sentinel])  object -- 支持迭代的集合对象。  sentinel -- 如果传递了第二个参数，则参数 object 必须是一个可调用的对象（如，函数），此时，iter 创建了一个迭代器对象，每次调用这个迭代器对象的\_\_next\_\_()方法时，都会调用 object。 |
| print() | 用于打印输出，最常见的一个函数。 | print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)  objects -- 复数，表示可以一次输出多个对象。输出多个对象时，需要用 , 分隔。  sep -- 用来间隔多个对象，默认值是一个空格。  end -- 用来设定以什么结尾。默认值是换行符 \n，我们可以换成其他字符串。  file -- 要写入的文件对象。  flush -- 输出是否被缓存通常决定于 file，但如果 flush 关键字参数为 True，流会被强制刷新。 |
| tuple() | 将列表转换为元组。 | tuple( seq )  seq -- 要转换为元组的序列。 |
| callable() | 用于检查一个对象是否是可调用的。如果返回 True，object 仍然可能调用失败；但如果返回 False，调用对象 object 绝对不会成功。  对于函数、方法、lambda 函式、 类以及实现了 **\_\_call\_\_** 方法的类实例, 它都返回 True。 | callable(object)  object -- 对象 |
| format() | 格式化字符串的函数 | 参见：  <https://www.runoob.com/python/att-string-format.html> |
| len() | 方法返回对象（字符、列表、元组等）长度或项目个数。 | len( s ) |
| property() | property() 函数的作用是在新式类中返回属性值。 |  |
| type() | type() 函数如果你只有第一个参数则返回对象的类型，三个参数返回新的类型对象。 |  |
| chr() | chr() 用一个整数作参数，返回一个对应的字符。 | chr(x) |
| frozenset() | frozenset() 返回一个冻结的集合，冻结后集合不能再添加或删除任何元素。 | class frozenset([iterable])  iterable -- 可迭代的对象，比如列表、字典、元组等等。 |
| list() | list() 方法用于将元组或字符串转换为列表。 | list(seq) |
| range() | Python3 range() 函数返回的是一个可迭代对象（类型是对象），而不是列表类型， 所以打印的时候不会打印列表。  Python3 list() 函数是对象迭代器，可以把range()返回的可迭代对象转为一个列表，返回的变量类型为列表。  Python2 range() 函数返回的是列表。 | range(stop)  range(start, stop[, step]) |
| vars() | 返回对象object的属性和属性值的字典对象。 | vars([object])  参数  object -- 对象 |
| classmethod | classmethod 修饰符对应的函数不需要实例化，不需要 self 参数，但第一个参数需要是表示自身类的 cls 参数，可以来调用类的属性，类的方法，实例化对象等。 | @classmethod  def func2(cls): |
| getattr() | getattr() 函数用于返回一个对象属性值。 | getattr(object, name[, default]) |
| locals() | locals() 函数会以字典类型返回当前位置的全部局部变量。  对于函数, 方法, lambda 函式, 类, 以及实现了 \_\_call\_\_ 方法的类实例, 它都返回 True。 | locals() |
| repr() | repr() 函数将对象转化为供解释器读取的形式。 | repr(object) |
| zip() | zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的对象，这样做的好处是节约了不少的内存。 |  |
| compile() | compile() 函数将一个字符串编译为字节代码。 | compile(source, filename, mode[, flags[, dont\_inherit]])  source -- 字符串或者AST（Abstract Syntax Trees）对象。。  filename -- 代码文件名称，如果不是从文件读取代码则传递一些可辨认的值。  mode -- 指定编译代码的种类。可以指定为 exec, eval, single。  flags -- 变量作用域，局部命名空间，如果被提供，可以是任何映射对象。。  flags和dont\_inherit是用来控制编译源码时的标志 |
| globals() | globals() 函数会以字典类型返回当前位置的全部全局变量。 | globals() |
| map() | map() 会根据提供的函数对指定序列做映射。  第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。 | map(function, iterable, ...)  参数  function -- 函数  iterable -- 一个或多个序列  返回值  Python 2.x 返回列表。  Python 3.x 返回迭代器。 |
| reversed() | reversed 函数返回一个反转的迭代器。 | reversed(seq) |
| \_\_import\_\_() | \_\_import\_\_() 函数用于动态加载类和函数 。  如果一个模块经常变化就可以使用 \_\_import\_\_() 来动态载入。 | \_\_import\_\_(name[, globals[, locals[, fromlist[, level]]]])  参数说明：  name -- 模块名 |
| complex() | complex() 函数用于创建一个值为 real + imag \* j 的复数或者转化一个字符串或数为复数。如果第一个参数为字符串，则不需要指定第二个参数。。 | class complex([real[, imag]])  参数说明：  real -- int, long, float或字符串；  imag -- int, long, float； |
| hasattr() | hasattr() 函数用于判断对象是否包含对应的属性。 | hasattr(object, name) |
| max() | max() 方法返回给定参数的最大值，参数可以为序列。 | max( x, y, z, .... ) |
| round() | round() 方法返回浮点数x的四舍五入值。 | round( x [, n] )  x -- 数字表达式。  n -- 表示从小数点位数，其中 x 需要四舍五入，默认值为 0。 |
| delattr(object, name) | delattr 函数用于删除属性。 | delattr(object, name)  object -- 对象。  name -- 必须是对象的属性。 |
| hash() | hash() 用于获取取一个对象（字符串或者数值等）的哈希值。 | hash(object) |
| memoryview() | memoryview() 函数返回给定参数的内存查看对象(Momory view)。  所谓内存查看对象，是指对支持缓冲区协议的数据进行包装，在不需要复制对象基础上允许Python代码访问。 | memoryview(obj) |
| set() | set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等。 | class set([iterable])  参数说明：  iterable -- 可迭代对象对象； |