常用正则表达式符号和特殊字符

|  |  |  |
| --- | --- | --- |
| 表示法 | 描述 | 正则表达式示例 |
| 符号 |  |  |
| literal | 匹配文本字符串的字面值 literal | Foo |
| re1|re2 | 匹配正则表达式 re1 或者 re2 | foo|bar |
| . | 匹配任何字符（除了\n 之外） | b.b |
| ^ | 匹配字符串起始部分 | ^Dear |
| $ | 匹配字符串终止部分 | /bin/\*sh$ |
| \* | 匹配 0 次或者多次前面出现的正则表达式 | [A-Za-z0-9]\* |
| + | 匹配 1 次或者多次前面出现的正则表达式 | [a-z]+\.com |
| ? | 匹配 0 次或者 1 次前面出现的正则表达式 | goo? |
| {N} | 匹配 N 次前面出现的正则表达式 | [0-9]{3} |
| {M,N} | 匹配M~N次前面出现的正则表达式 | [0-9]{5,9} |
| […] | 匹配来自字符集中的任意单一字符 | [aeiou] |
| [..x−y..] | 匹配 x ～ y 范围中的任意单一字符 | [0-9], [A-Za-z] |
| [^…] | 不匹配此字符集中出现的任何一个字符，包括某一范围的字符（如果在此字符集中出现） | [^aeiou], [^A-Za-z0-9] |
| (\*|+|?|{})? | 用于匹配上面频繁出现/重复出现符号的非贪婪版本（\*、+、?、{}） | .\*?[a-z] |
| (…) | 匹配封闭的正则表达式，然后另存为子组 | ([0-9]{3})?,f(oo|u)bar |
| 特殊字符 |  |  |
| \d | 匹配任何十进制数字，与[0-9]一致（\D 与\d 相反，不匹配任何非数值型的数字） | data\d+.txt |
| \w | 匹配包括下划线的任何单词字符，与[A-Za-z0-9\_]相同（\W 与之相反） | [A-Za-z\_]\w+ |
| \s | 匹配任何空白字符，与[\n\t\r\v\f]相同（\S 与之相反） | of\sthe |
| \b | 匹配任何单词边界,也就是指单词和空格间的位置（\B 与之相反） | \bThe\b |
| \t | 匹配一个制表符 |  |
| \n | 匹配一个换行符 |  |

Python 3 基本数据类型

Number（数字）

Python3 支持 int、float、bool、complex（复数）

Python 中的变量不需要声明。每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。可为多个对象指定多个变量，例如：

>>> a, b, c, d = 20, 5.5, True, 4+3j

内置的type()函数可以用来查询变量所指的对象类型，此外还可以用isinstance来判断

>>> a = 111

>>> isinstance(a, int)

True

isinstance 和 type 的区别在于：

type()不会认为子类是一种父类类型。

isinstance()会认为子类是一种父类类型。

数值运算：

>>> 5 + 4 # 加法

9

>>> 4.3 - 2 # 减法

2.3

>>> 3 \* 7 # 乘法

21

>>> 2 / 4 # 除法，得到一个浮点数

0.5

>>> 2 // 4 # 除法，得到一个整数

0

>>> 17 % 3 # 取余

2

>>> 2 \*\* 5 # 乘方

32

数值运算注意：

1、Python可以同时为多个变量赋值，如a,b=1,2。

2、一个变量可以通过赋值指向不同类型的对象。

3、数值的除法(/)总是返回一个浮点数，要获取整数使用//操作符。

4、在混合计算时，python会把整形转换为浮点数。

String（字符串） 用单引号括起来

Python中的字符串用单引号(‘)括起来，同时使用反斜杠(\)转义特殊字符

要计算str包含多少个字符，可以用len()函数

字符串的截取的语法格式如下：

变量[头下标：尾下标] 不包括尾下标所指字符 （尾下标-头下标）=字符个数

索引值以0为开始值，-1为末尾的开始位置

加号(+)是字符串的连接符，星号(\*)表示复制当前字符串，紧跟数字为复制的次数

str = 'Runoob'

print (str) # 输出字符串 Runoob

print (str[0:-1]) # 输出第一个个到倒数第二个的所有字符 Runoo

print (str[0]) # 输出字符串第一个字符 R

print (str[2:5]) # 输出从第三个开始到第五个的字符 noo

print (str[2:]) # 输出从第三个开始的后的所有字符 noob

print (str \* 2) # 输出字符串两次

print (str + "TEST") # 连接字符串

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

>>> print('Ru\noob')

Ru

oob

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

Ru\noob

>>>

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

**注意：**

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

List(列表) 写在方括号[]里

List(列表)是python中使用最频繁的数据类型

列表可以完成大多数集合类的数据结构实现。列表中元素的类型可以不相同，它支持数字，字符串甚至可以包含列表（所谓嵌套）。

列表是写在方括号([])之间、用逗号分隔开的元素列表。

和字符串一样，列表同样可以被索引和截取，列表被截取后返回一个包含所需元素的新列表。

列表截取的语法格式如下： 变量[头下标:尾下标] （尾下标-头下标）=元素个数

索引值以 0 为开始值，-1 为从末尾的开始位置。加号+是列表连接运算符，星号\*是重复操作。

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

tinylist = [123, 'runoob']

print (list) # 输出完整列表

print (list[0]) # 输出列表第一个元素 abcd

print (list[1:3]) # 从第二个开始输出到第三个元素 [786,2.23]

print (list[2:]) # 输出从第三个元素开始的所有元素 [2.23, 'runoob', 70.2]

print (tinylist \* 2) # 输出两次列表

print (list + tinylist) # 连接列表

与Python字符串不一样的是，列表中的元素是可以改变的：

>>> a = [1, 2, 3, 4, 5, 6]

>>> a[0] = 9 a=[9 ,2 ,3 ,4 ,5 ,6] 改变a[0]的值

>>> a[2:5] = [] # 删除

>>> a 输出[9, 2, 6] 从下标2开始删除三个元素

**注意：**

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

Tuple(元组) 写在小括号()里

元组(tuple)与列表类似，不同之处在于元组的元素不能修改。元组写在小括号()里，元素之前用逗号隔开。元组中的元素类型也可以不相同：

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

tinytuple = (123, 'runoob')

print (tuple) # 输出完整元组

print (tuple[0]) # 输出元组的第一个元素 abcd

print (tuple[1:3]) # 输出从第二个元素开始到第三个元素 (786,2.23)

print (tuple[2:]) # 输出从第三个元素开始的所有元素

print (tinytuple \* 2) # 输出两次元组

print (tuple + tinytuple) # 连接元组

元组与字符串类似，可以被索引且下标索引从0开始，-1 为从末尾开始的位置。也可以进行截取（看上面，这里不再赘述）。其实，可以把字符串看作一种特殊的元组。

虽然tuple的元素不可改变，但它可以包含可变的对象，比如list列表。

构造包含 0 个或 1 个元素的元组比较特殊，所以有一些额外的语法规则：

tup1 = () # 空元组

tup2 = (20,) # 一个元素，需要在元素后添加逗号

**注意：**

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

Set(集合) 写在大括号{ }里 或者set()函数创建集合

集合(Set)是一个无序不重复元素的序列。

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，质疑：一个空集合必须用set()而不是{ }，因为{ }是用来创建一个空字典。

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

print(student) # 输出集合，重复的元素被自动去掉

{'Jack', 'Rose', 'Tom', 'Mary', 'Jim'}

# 成员测试

if(*'Rose'* in student) :

print(*'Rose 在集合中'*)

else :

print(*'Rose 不在集合中'*)

# set可以进行集合运算

a = set(*'abracadabra'*)

b = set(*'alacazam'*)

print(a) #{'b','r','d','c','a'}

print(a - b) # a和b的差集 {'b','r','d'}

print(a | b) # a和b的并集 {'l','z','a','r','c','d','m','b'}

print(a & b) # a和b的交集 {'c','a'}

print(a ^ b) # 不同时存在的元素 {'l' ,'z','r','m','d','b'}

Dictionary(字典) 字典用"{ }"标识，

字典(dictionary)是python中另一个非常有用的内置数据类型。

列表是有序的对象结合，字典是无序的对象集合，两者之前的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典是一种映射类型，字典用”{}”标识，它是一个无序的键(Key):值(value)对集合。

键(key)必须使用不可变类型。在同一个字典中，键(key)必须是唯一的。

dict = {}

dict[*'one'*] = *"1 - 菜鸟教程"*

dict[2] = *"2 - 菜鸟工具"*

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

print (dict[*'one'*]) # 输出键为 'one' 的值 1 - 菜鸟教程

print (dict[2]) # 输出键为 2 的值 2 - 菜鸟工具

print (tinydict) # 输出完整的字典 {'name': 'runoob', 'site': 'www.runoob.com', 'code': 1}

print (tinydict.keys()) # 输出所有键 dict\_keys(['code', 'name', 'site'])

print (tinydict.values()) # 输出所有值 dict\_values([1, 'runoob', 'www.runoob.com'])

Python运算符

Python逻辑运算符：

and X and Y 布尔"与"

or X or Y 布尔”或”

not not X 布尔”非”

python 成员运算符：

in 如果在指定的序列中找到值返回True，否则返回False

not in 如果在指定的序列中没有找到值返回True，否则返回False

Python3 条件控制

Python条件语句是通过一条或多条语句的执行结果来决定执行的代码块。

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

注意：

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

If嵌套

if 表达式1:

语句

if 表达式2:

语句

elif 表达式3:

语句

else

语句

elif 表达式4:

语句

else:

语句

Python3 循环语句

Python中的循环语句有 for 和 while。

While循环：

while 判断条件：

语句

While循环使用else语句：

在while…else，在条件语句为false时执行else的语句块

count = 0

while count < 5:

print (count, " 小于 5")

count = count + 1

else:

print (count, " 大于或等于 5")

For语句，Python for循环可以遍历任何序列的项目，如一个列表或者一个字符串。

For循环的一般格式如下：

for <variable> in <sequence>:

<statements>

else:

<statements>

实例如下：

>>> languages = ["C", "C++", "Perl", "Python"]

>>> for x in languages:

... print (x)

C

C++

Perl

Python

>>>

break 语句可以跳出 for 和 while 的循环体，如果你从for或while循环中终止，任何对应的循环else块将不执行。

continue语句告诉Python跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

Python3 迭代器与生成器

迭代是Python最强大的功能之一，是访问集合元素的一种方式。。

迭代器是一个可以记住遍历的位置的对象。

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：

list=[1,2,3,4,5]

it = iter(list) # 创建迭代器对象

for x in it:

print (x, end=" ") #输出结果： 1 2 3 4 5

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。

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()

输出：0 1 1 2 3 5 8 13 21 34 55