# 实验4. Python字符串的使用

实验目的：

1.掌握Python中字符串的基本使用方法，包括字符串格式化和常见的方法；

2.简单正则表达式的使用。

实验平台：

Windows系列+Python3.X

主要实验内容：

#### 一、字符串

1．Python 字符串

字符串（string）是 Python 中最常用的数据类型，Python使用引号（单引号'或双引号"）作为字符串的定界符（一般为单行字符串，多行字符串使用三引号，稍后介绍）。

要创建一个字符串，只要用成对的引号把若干个字符括起来即可，例如：

var1 = 'Hello World!'

var2 = "Python Runoob"

Python规定，单引号内可以使用双引号，这时双引号被视为一个普通字符，不再作为定界符，反之亦然。例如：

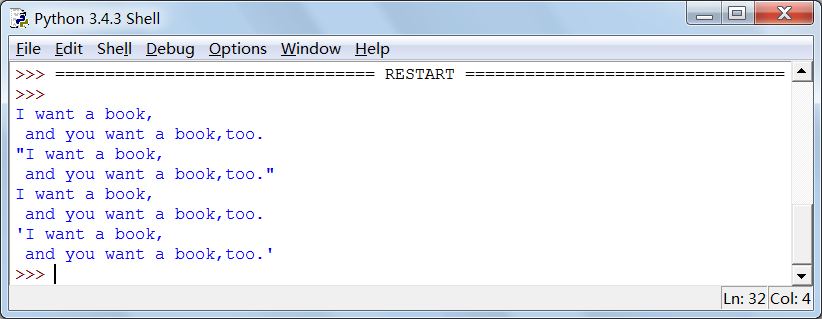
str1='I want a book,\n and you want a book,too.' *#单引号*

str2='"I want a book,\n and you want a book,too."' *#单引号中使用双引号*

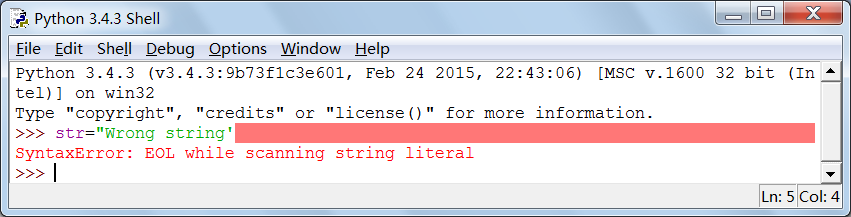
str3="I want a book,\n and you want a book,too." *#双引号*

str4="'I want a book,\n and you want a book,too.'" *#双引号中使用单引号*

print(str1); print(str2); print(str3); print(str4)



一个字符串用什么引号开头，就必须用什么引号结尾，首尾定界符不匹配时将出错，例如：



2．字符串中值的访问（子串截取）

与C语言不同，Python不支持字符类型，即使是单个字符，Python也将其视为一个字符串来使用。

访问子串，实际上就是截取字符串中的子串，有的文献称之为“切片运算”，使用的运算符是方括号（[]或[:]或[::]），例如：

str="0123456789"

print ("str[0:3]:",str[0:3]) *#截取第一位到第三位的字符*

print ("str[:]:",str[:]) *#截取字符串的全部字符*

print ("str[6:]:",str[6:]) *#截取第七个字符到结尾*

print ("str[:-3]:",str[:-3]) *#截取从头开始到倒数第三个字符之前*

print ("str[2]:",str[2]) *#截取第三个字符*

print ("str[-1]:",str[-1]) *#截取倒数第一个字符*

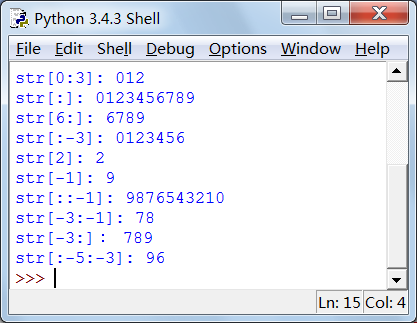
print ("str[::-1]:",str[::-1]) *#创建一个与原字符串顺序相反的字符串*

print ("str[-3:-1]:",str[-3:-1]) *#截取倒数第三位与倒数第一位之前的字符*

print ("str[-3:]：",str[-3:]) *#截取倒数第三位到结尾*

print ("str[:-5:-3]:",str[:-5:-3]) *#逆序截取*

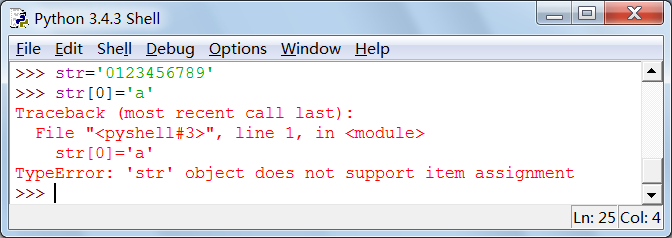
上述程序段的运行结果如下



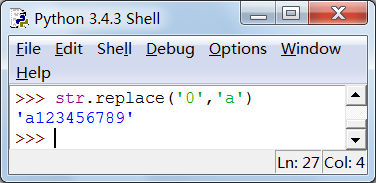
从上面的例子可以看出：（1）要获得单个字符，应使用切片运算符s[i]，但i不能省略；（2）要获得一个连续的子串，应使用切片运算符s[i:j]，它返回字符串s中从索引i（包括i）到j（不包括j）之间的子串，若i被省略，Python就认为i=0，若j被省略，Python就认为j=len(s)-1，其中len(s)表示字符串的长度（稍后介绍）；（3）要每间隔若干字符返回一个字符，应使用切片运算符s[i:j:k]，k为间隔字符的个数，k为正数表示从左向右截取，k为负数表示从右向左截取，k不能为0。不难想象，s[::-1]获得的将是逆序的整个字符串。

3．Python字符串的更新

不能企图通过切片运算更新已存在的字符串，如：



必须使用Python的内建函数replace()（稍后介绍）实现字符串的更新，例如：



4．Python字符串中的转义字符

当字符串中需要使用特殊字符时，可以像C语言那样，在特殊字符前冠以反斜杠(\)，形成转义字符。Python的转义字符如表4-1所示。

表4-1 Python的转义字符

|  |  |
| --- | --- |
| 转义字符 | 描述 |
| \(在行尾时) | 续行符 |
| \\ | 反斜杠符号 |
| \' | 单引号 |
| \" | 双引号 |
| \a | 响铃 |
| \b | 退格(Backspace) |
| \e | 转义 |
| \000 | 空 |
| \n | 换行 |
| \v | 纵向制表符 |
| \t | 横向制表符 |
| \r | 回车 |
| \f | 换页 |
| \0yy | 八进制数yy代表的字符，例如：\012代表换行 |
| \xyy | 十六进制数yy代表的字符，例如：\x0a代表换行 |
| \other | 其它的字符以普通格式输出 |

5．Python的字符串运算符

设a变量的值为字符串"Hello"，b变量的值为字符串"Python"，则各种运算结果如表4-2所示。

表4-2 Python的字符串运算符

|  |  |  |
| --- | --- | --- |
| 操作符 | 描述 | 实例 |
| + | 字符串连接 | a + b 输出结果： HelloPython |
| \* | 重复输出字符串 | a\*2 输出结果：HelloHello |
| [] | 通过索引获取字符串中的字符 | a[1] 输出结果：**e** |
| [ : ] | 截取字符串中的一部分（切片） | a[1:4] 输出结果： ell |
| in | 成员运算符——如果字符串中包含给定的字符，则返回 True | ‘H’ in a 输出结果： 1 |
| not in | 成员运算符——如果字符串中不包含给定的字符，则返回 True | ‘M’ not in a 输出结果： 1 |
| r/R | 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，没有转义为特殊的或不能打印的字符。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') 输出结果： \n  print (R'\n')输出结果： \n |
| % | 格式化字符串 | 见表2-5 |

下面将上述的字符串运算放入一个程序中。

#!/usr/bin/python3

a = "Hello"

b = "Python"

print("a + b 输出结果：", a + b)

print("a \* 2 输出结果：", a \* 2)

print("a[1] 输出结果：", a[1])

print("a[1:4] 输出结果：", a[1:4])

if( "H" in a) :

print("H 在变量 a 中")

else :

print("H 不在变量 a 中")

if( "M" not in a) :

print("M 不在变量 a 中")

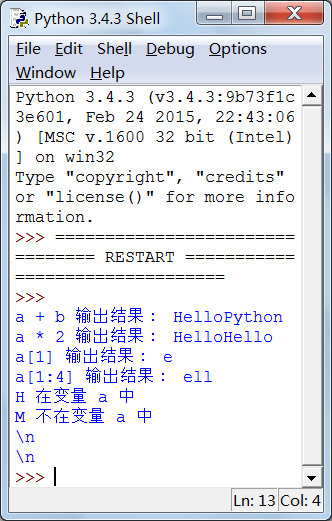
else :

print("M 在变量 a 中")

print (r'\n')

print (R'\n')

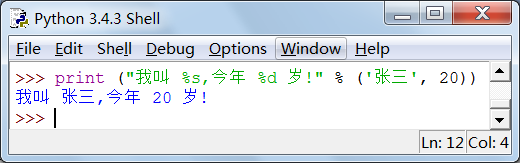
以上程序的输出结果如下：



6．Python字符串的格式化

Python支持字符串的格式化输出。尽管这样做可能会使表达式非常复杂，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，字符串格式化的语法与 C 中 printf 函数相似。我们看下面的例子：



Python字符串格式化用到的符号如表4-3所示。

表4-3 Python字符串格式化用到的符号

|  |  |
| --- | --- |
| 符号 | 描述 |
| %c | 格式化字符及其ASCII码 |
| %s | 格式化字符串 |
| %d | 格式化整型数 |
| %u | 格式化无符号整型数 |
| %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 | 按%f和%e的较短者输出 |
| %G | 按%f和%E的较短者输出 |
| %p | 用十六进制数格式化变量的地址 |

格式化操作符前面可以加入辅助操作符，这一点也与C语言极为相似。Python的格式化辅助操作符如表4-4所示。

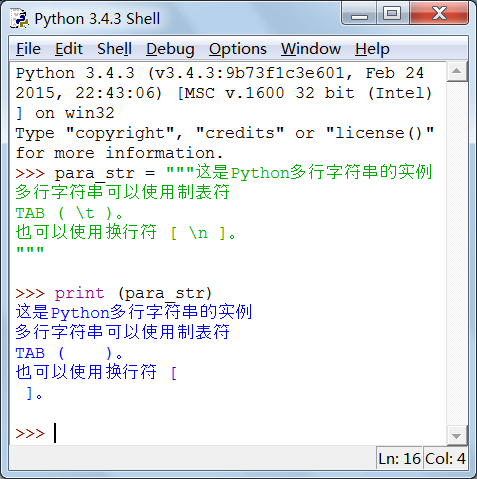
表4-4 Python格式化辅助操作符

|  |  |
| --- | --- |
| 符号 | 功能 |
| \* | 定义宽度或者精度 |
| - | 输出左对齐 |
| + | 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')，在十六进制数前面显示'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) |

尽管使用格式化操作符输出字符串很方便，而且与C语言兼容，但Python并不推荐使用这种方法，具体原因请参见本书4.1.2小节。

7．Python的三引号

Python的三引号（三个单引号或三个双引号均可）用于字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符。举例如下：



三引号把程序员从引号和特殊字符串的泥潭里解脱出来，自始至终保持一小块字符串的格式，符合WYSIWYG（What you see is what you get，所见即所得）。

一个典型的应用是，当需要一块HTML或者SQL时，如果用字符串组合转义字符，将会非常的繁琐，而使用三引号不失为一种绝佳的选择，例如：

errHTML = '''

<HTML><HEAD><TITLE>

Friends CGI Demo</TITLE></HEAD>

<BODY><H3>ERROR</H3>

<B>%s</B><P>

<FORM><INPUT TYPE=button VALUE=Back

ONCLICK="window.history.back()"></FORM>

</BODY></HTML>

'''

cursor.execute('''

CREATE TABLE users (

login VARCHAR(8),

uid INTEGER,

prid INTEGER)

''')

三引号中可以使用成对出现的字符串定界符，如

str5='''I want a book,and you want a book,\n too.''' *#三单引号*

str6='''"I want a book,and you want a book,\n too."''' *#三单引号中间使用双引号*

str7='''I want a book,

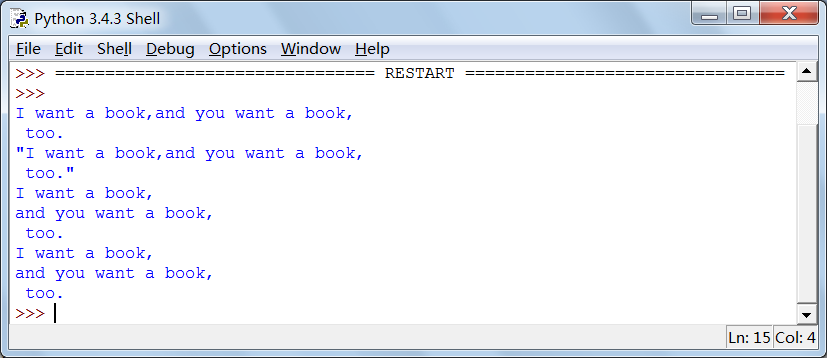
and you want a book,\n too.''' *#三单引号中有换行符*

str8="""I want a book,

and you want a book,\n too.""" *#三双引号中有换行符*

print(str5); print(str6); print(str7); print(str8)

上面程序段的运行结果如下：



注意：Python字符串的几种定界符均已介绍完毕，现对其归纳如下：

（1）单引号中可以使用双引号，中间的会当作字符串输出。

（2）双引号中可以使用单引号，中间的会当作字符串输出。

（3）三单引号和三双引号中间的字符串在输出时保持原来的格式。

（4）单引号和双引号不能搭配使用，必须成对使用。

8．Unicode 字符串

在Python 2中，普通字符串是以8位ASCII码形式进行存储的，而Unicode字符串则存储为16位Unicode字符串。使用Unicode字符串，旨在表示更多的字符。其语法非常简单，只要在字符串前面加上前缀u即可。

在Python 3中，默认所有的字符串都是Unicode字符串，因而不必在字符串前面加前缀u。

9．原始字符串

前面讲解字符串运算符时已提及原始字符串并举例（表2-4），此处再强调一下。在字符串前面加字母r或R，就是告诉Python解释器，其后的字符串是原始字符串（raw string）。原始字符串指的是，字符串内的所有字符均保持其原有的含义，不做转义处理。换言之，“\n”在原始串中是两个字符，即“\”和“n”，而不会被转义为换行符。由于正则表达式（本章最后介绍）经常与“\”冲突，所以，当一个字符串中使用了正则表达式之后，往往在前面加字母r，具体例子请参见2.8.2节。此外，在描述文件路径时往往使用“\”，为避免歧义，也往往在其前冠以字母r，具体例子请见5.1节。

10．Python的字符串内建函数

Python 的字符串有很多内建函数，常用的如表4-5所示。

表4-5 常用的Python 字符串内建函数

|  |  |
| --- | --- |
| 序号 | 方法及描述 |
| 1 | capitalize()  将字符串的第一个字符转换为大写。 |
| 2 | center(width, fillchar)  返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。 |
| 3 | count(str, beg= 0,end=len(string))  返回 str 在 string 里面出现的次数，如果指定 beg 或者 end，则返回指定范围内 str 出现的次数。 |
| 4 | bytes.decode(encoding="utf-8", errors="strict")  Python 3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。 |
| 5 | encode(encoding='utf-8',errors='strict')  以 encoding 指定的编码格式编码字符串，如果出错，则默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'。 |
| 6 | endswith(suffix, beg=0, end=len(string))  检查字符串是否以suffix结束，如果指定beg 或者 end，则检查指定的范围内是否以 suffix 结束，如果是，则返回 True,否则返回 False。 |
| 7 | expandtabs(tabsize=8)  把字符串中的 tab 符号转为空格，tab 符号默认的空格数是 8。 |
| 8 | find(str, beg=0 end=len(string))  检测 str 是否包含在字符串中，如果用 beg 和 end 指定范围，则检查是否包含在指定的范围内，如果是，则返回开始的索引值，否则返回-1。 |
| 9 | index(str, beg=0, end=len(string))  跟find()方法一样，只不过如果str不在字符串中，则会报一个异常。 |
| 10 | isalnum()  如果字符串至少有一个字符，并且所有字符都是字母或数字，则返回 True，否则返回 False。 |
| 11 | isalpha()  如果字符串至少有一个字符，并且所有字符都是字母，则返回 True，否则返回 False。 |
| 12 | isdigit()  如果字符串只包含数字，则返回 True，否则返回 False。 |
| 13 | islower()  如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False。 |
| 14 | isnumeric()  如果字符串中只包含数字字符，则返回 True，否则返回 False。 |
| 15 | isspace()  如果字符串中只包含空格，则返回 True，否则返回 False。 |
| 16 | istitle()  如果字符串是标题化的(见 title())，则返回 True，否则返回 False。 |
| 17 | isupper()  如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False。 |
| 18 | join(seq)  以指定字符串作为分隔符，将 seq 中所有的元素(字符串表示)合并为一个新的字符串。 |
| 19 | len(string)  返回字符串长度，即字符串中字符的个数。 |
| 20 | ljust(width[, fillchar])  返回一个原字符串左对齐，并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格。 |
| 21 | lower()  转换字符串中所有大写字母为小写。 |
| 22 | lstrip()  截掉字符串左边的空格。 |
| 23 | maketrans(intab,outtab)  创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串，表示转换的目标。 |
| 24 | max(str)  返回字符串 str 中最大的字母。 |
| 25 | min(str)  返回字符串 str 中最小的字母。 |
| 26 | replace(old, new [, max])  将字符串中的 old 替换成 new。如果 max 指定，则替换不超过 max 次。 |
| 27 | rfind(str, beg=0,end=len(string))  类似于 find()函数，不过是从右边开始查找。 |
| 28 | rindex( str, beg=0, end=len(string))  类似于 index()，不过是从右边开始。 |
| 29 | rjust(width,[, fillchar])  返回一个原字符串右对齐，并使用fillchar(默认空格）填充至长度 width 的新字符串。 |
| 30 | rstrip()  删除字符串末尾的空格。 |
| 31 | split(str="", num=string.count(str))  num=string.count(str)) 以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num 个子字符串。 |
| 32 | splitlines([keepends])  按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。 |
| 33 | startswith(str, beg=0,end=len(string))  检查字符串是否是以str开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查。 |
| 34 | strip([chars])  在字符串上执行 lstrip()和 rstrip()。 |
| 35 | swapcase()  将字符串中大写转换为小写，小写转换为大写。 |
| 36 | title()  返回“标题化”的字符串，就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())。 |
| 37 | translate(table, deletechars="")  根据 table 给出的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 deletechars 参数中。 |
| 38 | upper()  转换字符串中的小写字母为大写。 |
| 39 | zfill(width)  返回长度为 width 的字符串，原字符串右对齐，前面填充0。 |
| 40 | isdecimal()  检查字符串是否只包含十进制字符，如果是则返回True，否则返回 False。 |

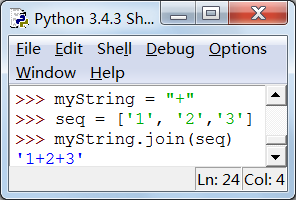
这些内建函数为我们处理字符串带来了极大的方便。读者学习Python语言时，不要试图自己编写程序处理字符串，应当充分利用这些内建函数完成相应的功能。

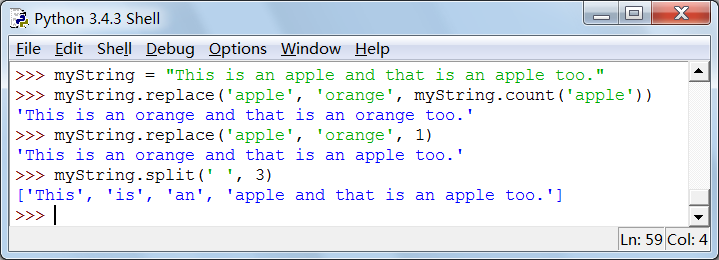
细心的读者可能已经注意到，表4-5的标题上写的是“函数”，而表格栏目中谓之“方法”。其实，函数与方法并无严格的区别。在面向过程的语言中，一个模块主要强调的是数据处理，就像数学上的函数一样，故称之为函数；在面向对象的语言中，一般把类中定义的函数称作方法、服务或操作，因为它主要强调这个类的对象封装了一些属性和方法（变量和函数）并向外提供服务。表4-5中所列函数均是定义在某个类里面的，故称之为方法。说到底，方法就是类函数。

有鉴于此，在不产生混淆的前提下，本书的后续章节拟不严格区分函数与方法。

有关类与面向对象的概念，请读者参阅第六章中的相关内容。

下面看几个关于字符串操作的例子（其中使用了下一节将要介绍的列表）。





#### 二、正则表达式

首先须指出，正则表达式并不是Python的一部分。正则表达式是用于处理字符串的强大工具，它拥有自己独特的语法以及一个独立的处理引擎，效率上可能不如str自带的方法，但功能十分强大。Python中内建了一个re模块以支持正则表达式。

正则表达式有两种基本操作，分别是匹配和替换。匹配指的是在一个文本字符串中搜索并匹配一个特殊表达式；替换指的是在一个字符串中查找并替换匹配一个特殊表达式的字符串。

#### 1. 基本元素

正则表达式定义了一系列的特殊字符元素，以执行匹配操作。表4-6列示了正则表达式的基本字符。

表4-6正则表达式的基本字符

|  |  |
| --- | --- |
| 字符 | 描述 |
| text | 匹配text字符串。 |
| . | 匹配除换行符之外的任意一个单个字符。 |
| ^ | 匹配一个字符串的开头。 |
| $ | 匹配一个字符串的末尾。 |

在正则表达式中，还可用匹配限定符来约束匹配的次数。表4-7列示了正则表达式的匹配限定符。

表4-7 正则表达式的匹配限定符

|  |  |  |
| --- | --- | --- |
| 最大匹配 | 最小匹配 | 描述 |
| \* | \*? | 重复匹配前表达式零次或多次。 |
| + | +? | 重复匹配前表达式一次或多次。 |
| ? | ?? | 重复匹配前表达式零次或一次。 |
| {m} | {m}? | 精确重复匹配前表达式m次。 |
| {m,} | {m,}? | 至少重复匹配前表达式m次。 |
| {m,n} | {m,n}? | 至少重复匹配前表达式m次，至多重复匹配前表达式n次。 |

由表2-20和表2-21可知，".\*"为最大匹配，能匹配源字符串中所有能匹配的字符串。".\*?"为最小匹配，只匹配第一次出现的字符串。例如，d.\*g能匹配任意以d开头、以g结尾的字符串，如"debug"和"debugging"，甚至"dog is walking"都能匹配。而d.\*?g只能匹配"debug"，在"dog is walking"字符串中，则只能匹配到"dog"。

如果要实现更为复杂的匹配，可以用组合运算符，如表4-8所示。

表4-8 组合运算符

|  |  |
| --- | --- |
| 组 | 描述 |
| [...] | 匹配集合内的字符，如[a-z],[1-9]或[,./;']。 |
| [^...] | 匹配除集合外的所有字符，相当于取反操作。 |
| A|B | 匹配表达式A或B，相当于or操作。 |
| (...) | 表达式分组，每对括号为一组，如([a-b]+)([A-Z]+)([1-9]+)。 |
| \number | 匹配在number表达式组内的文本。 |

有一组特殊的字符序列，用来匹配具体的字符类型或字符环境。如\b匹配字符边界，food\b匹配"food"、"zoofood"，而和"foodies"不匹配。这些特殊的字符序列示于表4-9。

表4-9 特殊字符序列

|  |  |
| --- | --- |
| 字符 | 描述 |
| \A | 只匹配字符串的开始。 |
| \b | 匹配一个单词边界。 |
| \B | 匹配一个单词的非边界。 |
| \d | 匹配任意十进制数字字符，等价于r'[0-9]'。 |
| \D | 匹配任意非十进制数字字符，等价于r'[^0-9]'。 |
| \s | 匹配任意空白字符（空格符、tab制表符、换行符、回车、换页符、垂直线符号）。 |
| \S | 匹配任意非空白字符。 |
| \w | 匹配任意字母数字字符。 |
| \W | 匹配任意非字母数字字符。 |
| \Z | 仅匹配字符串的尾部。 |
| \\ | 匹配反斜线字符。 |

正则表达式中还有一套声明(assertion)，可用于对具体事件进行声明，如表4-10所示。

表4-10正则表达式声明

|  |  |
| --- | --- |
| 声明 | 描述 |
| (?iLmsux) | 匹配空字符串，iLmsux字符对应下表的正则表达式修饰符。 |
| (?:...) | 匹配圆括号内定义的表达式，但不填充字符组表。 |
| (?P) | 匹配圆括号内定义的表达式，但匹配的表达式还可用作name标识的符号组。 |
| (?P=name) | 匹配所有与前面命名的字符组相匹配的文本。 |
| (?#...) | 引入注释，忽略圆括号内的内容。 |
| (?=...) | 如果所提供的文本与下一个正则表达式元素匹配，这之间没有多余的文本就匹配。这允许在一个表达式中进行超前操作，而不影响正则表达式其余部分的分析。如"Martin"其后紧跟"Brown"，则"Martin(?=Brown)"就只与"Martin"匹配。 |
| (?!...) | 仅当指定表达式与下一个正则表达式元素不匹配时匹配，是(?=...)的反操作。 |
| (?<=...) | 如果字符串当前位置的前缀字符串是给定文本，就匹配，整个表达式就在当前位置终止。如(?<=abc)def表达式与"abcdef"匹配。这种匹配是对前缀字符数量的精确匹配。 |
| (?<!...) | 如果字符串当前位置的前缀字符串不是给定的正文，就匹配，是(?<=...)的反操作。 |

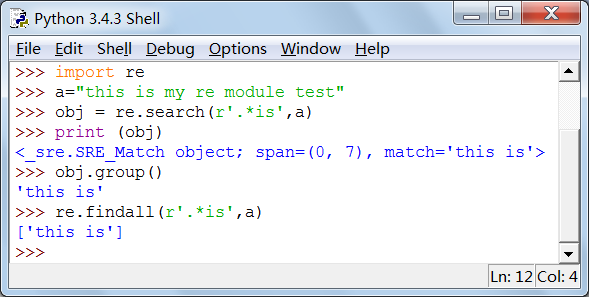
正则表达式还支持一些修饰符（见表4-11），它会影响正则表达式的执行方法。

表4-11 正则表达式的常用修饰符

|  |  |
| --- | --- |
| 修饰符 | 描述 |
| I | 忽略表达式的大小写来匹配文本。 |
| L | 根据当前语言环境解释单词。这种解释影响字母组(\w和\W)以及字边界行为(\b和\B)。 |
| M | 多行匹配。就是匹配换行符两端的潜在匹配。影响正则中的^$符号 |
| S | 使一个句点(.)匹配任何字符，包括换行符。 |
| U | 根据Unicode字符集解释字母。此标志影响\w，\W，\b，\B的行为。 |

#### 2.正则表达式的操作举例

通过Python内建的re模块，我们就可以在Python中利用正则表达式对字符串进行搜索、抽取和替换操作。例如：使用re.search()函数能够执行一个基本的搜索操作，它能返回一个MatchObject对象；使用re.findall()函数能够返回一个匹配列表。例如：



搜索操作返回的MatchObject对象有很多方法可供使用，其功能如表4-12所示。

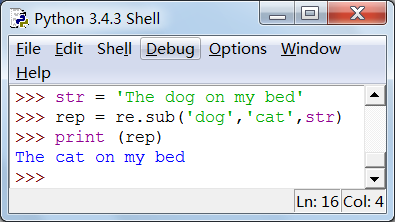
表4-12 MatchObject对象的常用方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| expand(template) | 展开模板中用反斜线定义的内容。 |
| m.group([group,...]) | 返回匹配的文本，是个元组。此文本是与给定group或由其索引数字定义的组匹配的文本，如果没有给定组名，则返回所有匹配项。 |
| m.groups([default]) | 返回一个元组，该元组包含模式中与所有组匹配的文本。如果给出default参数，default参数值就是与给定表达式不匹配的组的返回值。default参数的默认取值为None。 |
| m.groupdict([default]) | 返回一个字典，该字典包含匹配的所有子组。如果给出default参数，其值就是那些不匹配组的返回值。default参数的默认取值为None。 |
| m.start([group]) | 返回指定group的开始位置，或返回全部匹配的开始位置。 |
| m.end([group]) | 返回指定group的结束位置，或返回全部匹配的结束位置。 |
| m.span([group]) | 返回两元素组，此元组等价于关于一给定组或一个完整匹配表达式的(m.start(group),m.end(group)))列表。 |
| m.pos | 传递给match()或search()函数的pos值。 |
| m.endpos | 传递给match()或search()函数的endpos值。 |
| m.re | 创建这个MatchObject对象的正则式对象 |
| m.string | 提供给match()或search()函数的字符串。 |

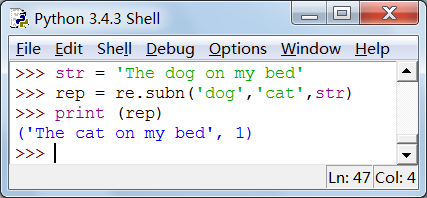
使用sub()或subn()函数可以在字符串上执行替换操作。sub()函数的基本格式如下：

sub(pattern,replace,string[,count])

举例如下：



其中的replace参数可接受函数。要获得替换的次数，可使用subn()函数（基本格式同sub()函数），该函数返回一个元组，此元组包含替换了的文本及替换的次数。



若需要用同一个正则式进行多次匹配操作，则可以把正则表达式编译成内部语言，这样可以提高处理速度。编译正则表达式使用compile()函数，其基本格式为：

compile(str[,flags])

其中，str表示需要编译的正则表达式串，flags是一个修饰标志符。正则表达式被编译后生成一个对象，该对象拥有多种方法和属性，如表4-13所示。

表4-13正则表达式编译后对象的方法/属性

|  |  |
| --- | --- |
| 方法/属性 | 描述 |
| r.search(string[,pos[,endpos]]) | 同search()函数，但此函数允许指定搜索的起点和终点。 |
| r.match(string[,pos[,endpos]]) | 同match()函数，但此函数允许指定搜索的起点和终点。 |
| r.split(string[,max]) | 同split()函数。 |
| r.findall(string) | 同findall()函数。 |
| r.sub(replace,string[,count]) | 同sub()函数。 |
| r.subn(replace,string[,count]) | 同subn()函数。 |
| r.flags | 创建对象时定义的标志。 |
| r.groupindex | 将r'(?Pid)'定义的符号组名字映射为组序号的字典。 |
| r.pattern | 在创建对象时使用的模式。 |

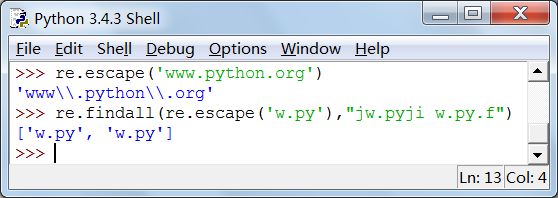
有两个函数在此值得一提，一是re.escape()函数，二是re.getattr()函数。

1．re.escape()函数用于转义字符串

在使用Python的过程中，对转义字符的使用也有苦恼之时。因为有时候我们需要使用一些特殊符号，如“$ \* . ^”等的原意，有时候需要的是被转义后的功能，并且转义字符的使用很繁琐时，很容易出错，re.escape()是解决这一问题的灵丹妙药。

re.escape(pattern)可以对字符串中所有可能被解释为正则运算符的字符进行转义。如果字符串很长，且包含很多特殊字符，而又不想输入一大堆反斜杠，或者字符串来自于用户（比如通过input函数获取输入的内容），且要用作正则表达式的一部分的时候，就可以使用这个函数。

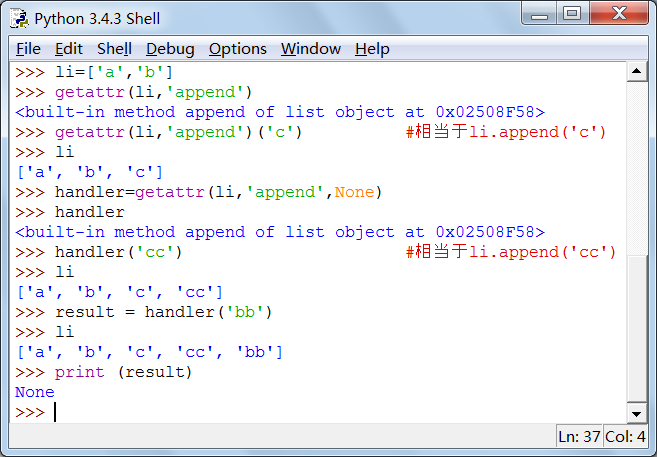
现举一例说明之：



这里的re.escape(‘w.py’)用作re.findall函数的正则表达式部分。

2．re.getattr()函数用于获取对象的引用

现举例说明如下：



#### 三、练习

1．字符串练习(一)：写出各条命令的执行结果。

a = 'Hello World'

c = a[0:5]

d = a[6:]

e = a[3:8]

print (c,d,e)

print (a[6:-1])

b=’Python’

print (a+b)

print (a\*2)

2．字符串练习（二）：写出各条命令的执行结果。

str="0123456789"

print ("str[0:3]:",str[0:3])

print ("str[:]:",str[:])

print ("str[6:]:",str[6:])

print ("str[:-3]:",str[:-3])

print ("str[2]:",str[2])

print ("str[-1]:",str[-1])

print ("str[::-1]:",str[::-1])

print ("str[-3:-1]:",str[-3:-1])

print ("str[-3:]：",str[-3:])

print ("str[:-5:-3]:",str[:-5:-3])

3．正则表达式练习：根据要求写出相应的正则表达式。

（1）匹配正整数

（2）匹配负整数

（3）匹配整数

（4）匹配由26个大写英文字母组成的字符串

（5）匹配由26个小写英文字母组成的字符串

（6）匹配由26个英文字母组成的字符串

（7）匹配由数字和26个英文字母组成的字符串

（8）匹配中国邮政编码

（9）匹配身份证（15位或18位）

# 实验5.Python控制结构

实验目的：

1.理解选择结构和循环结构的执行过程

2.掌握选择结构和循环结构的编程方法

3.理解循环嵌套的执行过程

4.运用流程控制编程解决一些实际问题

实验平台：

Windows系列+Python3.X

主要实验内容：

熟读教材第三章内容。编写以下程序：

1.编写输出10以内素数的循环程序。

for n in range(2, 10):

for x in range(2, n):

if n % x == 0:

print(n, '等于', x, '\*', n//x)

break

else:

*# 循环中没有找到元素*

print(n, ' 是素数')

========================== RESTART =========================

2 是素数

3 是素数

4 等于 2 \* 2

5 是素数

6 等于 2 \* 3

7 是素数

8 等于 2 \* 4

9 等于 3 \* 3

2.使用if-elif-else语句判断输入的数字是正数、负数还是零。使用嵌套的if语句实现同样的功能。

方法一：

num = float(input("输入一个数字: "))

if num < 0:

print("负数")

elif num == 0:

print("零")

else:

print("正数")

========================== RESTART =========================

输入一个数字: 5

正数

方法二：

num = float(input("输入一个数字: "))

if num <= 0:

if num == 0:

print("零")

else:

print("负数")

else:

print("正数")

========================== RESTART =========================

输入一个数字: -5

负数

3.用if语句判断输入的一个数字是奇数还是偶数。

*# Python判断奇数偶数*

*# 如果是偶数，除以2后余数为0*

*# 如果是余数，除以2后余数为1*

num = int(input("输入一个数字: "))

if (num % 2) != 0:

print("{0} 是奇数".format(num))

else:

print("{0} 是偶数".format(num))

========================== RESTART =========================

输入一个数字: -7

-7 是奇数

========================== RESTART =========================

输入一个数字: 8

8 是偶数

4.用if语句判断用户输入的年份是否为闰年。

year = int(input("输入一个年份: "))

if (year % 4) == 0:

if (year % 100) == 0:

if (year % 400) == 0:

print("{0} 是闰年".format(year)) *# 整百年能被400整除的是闰年*

else:

print("{0} 不是闰年".format(year))

else:

print("{0} 是闰年".format(year))  *# 非整百年能被4整除的为闰年*

else:

print("{0} 不是闰年".format(year))

========================== RESTART =========================

输入一个年份: 1982

1982 不是闰年

5.使用标准格式输出阶乘（factorial）。整数的阶乘是所有小于及等于该数的正整数的积，即：n!=1×2×3×... ×n。0的阶乘定义为1。

*# 获取用户输入的数字*

num = int(input("请输入一个数字: "))

factorial = 1

*# 查看数字是负数，0 或 正数*

if num < 0:

print("抱歉，负数没有阶乘")

elif num == 0:

print("0 的阶乘为 1")

else:

for i in range(1,num + 1):

factorial \*= i

print("%d 的阶乘为 %d" %(num,factorial))

========================== RESTART =========================

请输入一个数字: 5

5 的阶乘为 120

6.改进九九乘法表，用for语句和range()函数实现。建议使用end换行。

*# 九九乘法表*

for i in range(1, 10):

for j in range(1, i+1):

print('{}x{}={}\t'.format(i, j, i\*j), end='')

print()

========================== RESTART =========================

1x1=1

2x1=2 2x2=4

3x1=3 3x2=6 3x3=9

4x1=4 4x2=8 4x3=12 4x4=16

5x1=5 5x2=10 5x3=15 5x4=20 5x5=25

6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36

7x1=7 7x2=14 7x3=21 7x4=28 7x5=35 7x6=42 7x7=49

8x1=8 8x2=16 8x3=24 8x4=32 8x5=40 8x6=48 8x7=56 8x8=64

9x1=9 9x2=18 9x3=27 9x4=36 9x5=45 9x6=54 9x7=63 9x8=72 9x9=81

7.求指定区间内的水仙花数（亦称阿姆斯特朗数），要求使用循环语句和判断语句。

如果一个n位正整数等于其各位数字的立方之和，则称该数为水仙花数或阿姆斯特朗数。例如3^3 + 7^3 + 0^3 = 370。1000以内的水仙花数有： 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407。

*# 获取用户输入数字*

lower = int(input("最小值: "))

upper = int(input("最大值: "))

for num in range(lower,upper + 1):

sum = 0 *#初始化*

n = len(str(num)) *#指数*

*#检测*

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* n

temp //= 10

if num == sum:

print(num,end=' ')

========================== RESTART =========================

最小值: 1

最大值: 1000

1 2 3 4 5 6 7 8 9 153 370 371 407