



第6章 字符串与正则表达式

刘 卉

huiliu@fudan.edu.cn



6.1 字符串

字符串基本知识

字符串常用方法

字符串格式化

精选案例

字符串

由字符(字母/数字/汉字/其它符号)组成的一个序列.

必须被括在如下的一对符号里:

- 一对单引号''
- 一对双引号""
- 一对三单引号''' '''
- 一对三双引号"""" """"

e.g.

```
>>> a = '上海复旦(200433)'
>>> b = "上海复旦(200433)"
>>> c = """上海复旦(200433)"""
>>> d = """"上海复旦(200433)""""
>>> print(a)
上海复旦(200433)
>>> print(b)
上海复旦(200433)
>>> print(c)
上海复旦(200433)
>>> print(d)
上海复旦(200433)
```

字符编码

ASCII

- 1个字节保存1个字符.
- 128个字符组成: 大小写字母、数字0-9、标点符号、非打印字符(换行符、制表符等4个)以及控制字符(退格、响铃等).

UTF-8

- 1 ~ 6个字节保存1个字符.
- 对世界上所有国家需要用到的字符进行了编码.

GB2312

- 1个字节表示英文, 2个字节表示中文.
- 中国制定的中文编码.

GBK

- 对GB2312的扩充.

CP936

- 微软在GBK基础上完成的编码.

ASCII表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位		ASCII控制字符											ASCII打印字符													
		0000					0001						0010		0011		0100		0101		0110		0111			
		0					1						2		3		4		5		6		7			
		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
低四位																										
0000	0	0		^@	NUL	\0	空字符	16	▶	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH		标题开始	17	◀	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☹	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX		正文结束	19	!!	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK		肯定应答	22	—	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	\a	响铃	23	↕	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◼	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◻	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[BSC	\e	溢出	43	+	59	;	75	K	91	[107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	└	^_	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124		
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分分隔符	45	-	61	=	77	M	93]	109	m	125	}	
1110	E	14	🎵	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	🎵	^O	SI		移入	31	▼	^~	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	△	^Backspace 代码: DEL

^Backspace
代码: DEL

注: 表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入。

制作: NHL QQ:1208980380

2013/08/08

文件编码

- 采用不同编码意味着把同一字符存入文件时, 写入的内容可能不同.
- 在Python3中, 程序源文件默认为UTF-8编码, 全面支持中文.
- 无论是1个数字/字母/汉字, 都按1个字符对待.
- 可使用中文作变量名(不推荐).

```
>>> s = '上海市杨浦区'
>>> len(s)
6
>>> t = 'www.fudan.edu.cn'
>>> len(t)
16
```

```
>>> 姓名 = '匿名'
>>> 年龄 = 19
>>> print(姓名)
匿名
>>> print(年龄*2)
38
```


字符串驻留机制string interning

- 将同一字符串赋值给多个不同对象时, 内存中只有一个副本, 多个对象共享该副本.

- 保护.pyc文件不会因错误代码变得过大.

e.g. 'abc'*10**10

- 数值-5~256也遵循驻留机制
- 关于驻留机制参见:

<http://blog.csdn.net/handsomekang/article/details/41170685>

```
>>> s = "good"
>>> t = "good"
>>> id(s)
49008320
>>> id(t)
49008320
```



6.1 字符串

字符串基本知识

字符串常用方法

字符串格式化

精选案例

判断一个变量是否为字符串

- 使用内置函数 `isinstance(obj, class)` 或 `type(obj)`

```
>>> a = 'good'
>>> isinstance(a, str)
True
>>> b = '汉语'
>>> isinstance(b, str)
True
```

```
>>> a = 'good'
>>> type(a)
<class 'str'>
>>> type(a) is str
True
```



字符串成员判断

□ 成员判断

```
>>> "a" in "abcde"
```

```
True
```

```
>>> "j" in "abcde"
```

```
False
```

❑ 字符串属于不可变序列类型

```
>>> st = "Fudan"
>>> st[2] = "D"
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    st[2] = "D"
TypeError: 'str' object does not support item assignment
```

❑ 支持序列通用方法, 包括切片操作.

```
>>> s = "Fudan University"
>>> x = s[2]
>>> x
'd'
>>> y = s[3:8]
>>> y
'an Un'
>>> z = s[-1:-6:-1]
>>> z
'ytisr'
```

□ 支持字符串特有的操作方法.

```
>>> a = "abc"
>>> b = "abc" * 3
>>> b
'abccabccabc'
>>> c = a + "xyz"
>>> c
'abcxyz'
>>> "x" in c
True
>>> "xy" in c
True
>>> a in b
True
>>> len(c)
6
```

[例] 把一个字符串均分为m行输出, m由用户指定

```
Enter a string: Fudan University Shanghai China
Enter an integer: 3
Fudan Unive
rsity Shang
hai China
```

```
s = input('Enter a string: ')
m = eval(input('Enter an integer: '))

# 求每行的字符数
n = len(s)//m if len(s) % m == 0 else len(s)//m+1
# 方法一：循环输出m行, 每行n个字符
for i in range(m):
    print(s[i*n: (i+1)*n])
# 方法二：以n为步长, 遍历字符串
for i in range(0, len(s), n):
    print(s[i: i+n])
```

字符串str和模块string

- str—a class in module builtins
 - Python内置模块中的一个类
 - 包含大多数字符串处理函数
- string—A collection of string constants
 - 使用前需导入
 - 包含常用的字符串常量



字符串常量

```
>>> import string
>>> string.digits # 数字字符
'0123456789'
>>> string.punctuation # 标点符号
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.ascii_letters # 英文字母
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.printable # 可打印字符
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
>>> string.ascii_lowercase # 小写字母
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_uppercase # 大写字母
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```



检验字符串是否为字母、数字等

`str.isalnum()` 检验字符串是否为数字或字母

`str.isalpha()` -----字母

`str.isdigit()` -----数字字符

`str.isspace()` -----空白字符

`str.isupper()` -----大写字母

`str.islower()` -----小写字母

举例

```
>>> '1234abcd'.isalnum()  
True  
>>> '1234abcd'.isalpha()  
False  
>>> '1234abcd'.isdigit()  
False  
>>> 'abcd'.isalpha()  
True  
>>> '1234.0'.isdigit()  
False  
>>> '1234'.isdigit()  
True  
>>>
```

[例] 删除字符串中的非字母和非数字符号

```
import string # 方法一：利用string模块中定义的常量

def delchar(s):
    charSet = string.ascii_letters + string.digits
    result = ''
    for c in s:
        if c in charSet:
            result += c # 只保留字母和数字
    return result

if __name__ == '__main__':
    s = input('Enter something: ')
    result = delchar(s)
    print(result) # print(delchar(s))
```



□ 方法二: 利用str.isalnum()

```
def delchar(s):
```

```
    result = ''  
    for c in s:  
        if c.isalnum():  
            result += c  
    return result
```

```
if __name__ == '__main__':  
    s = input('Enter something: ')  
    print(delchar(s))
```

字符串转换函数



仅返回结果, 均**不修改**原字符串

str.lower(): 得到小写格式字符串

str.upper(): 得到大写格式字符串

str.capitalize(): 得到首字母大写格式

str.title(): 得到每个单词首字母大写格式

str.swapcase(): 得到大小写互换格式

```
>>> s = "What is YoUr NaMe?"
>>> t = s.lower()
>>> t
'what is your name?'
>>> s.upper()
'WHAT IS YOUR NAME?'
>>> s.capitalize()
'What is your name?'
>>> s.title()
'What Is Your Name?'
>>> s.swapcase()
'wHAT IS yOuR nAmE?'
>>> s
'What is YoUr NaMe?'
>>>
```



互逆的两个BIF ord(.)与chr(.)

ord(字符) → ASCII码

```
>>> ord('8')
56
>>> ord('H')
72
>>> ord('y')
121
>>> print(ord('k'))
107
```

chr(ASCII码) → 字符

```
>>> chr(55)
'7'
>>> chr(100)
'd'
>>> chr(0)
'\x00'
>>> print(chr(0))
□
```



随机生成小写英文字母

□ 方法一

```
>>> import random
>>> import string # 随机选择1个小写英文字母
>>> print(random.choice(string.ascii_lowercase))
```

□ 方法二

```
>>> r = random.randint(0, 25) # 返回[0,25]区间的一个随机整数
>>> print(chr(r+ord('a'))) # 随机生成1个小写英文字母
# ord('a') <= r+ord('a') <= ord('z')
# 'a' <= chr(r+ord('a')) <= 'z'
```


编程实现大小写字母转换

```
def swap_case(s): # 转换1个字符串
    result = ''
    for c in s:
        if 'a' <= c <= 'z': # if x in string.ascii_lowercase:
            result += chr(ord(c) + ord('A') - ord('a'))
        elif 'A' <= c <= 'Z': # if x in string.ascii_uppercase:
            result += chr(ord(c) + ord('a') - ord('A'))
        else:
            result += c
    return result

if __name__ == '__main__':
    s = input('Enter a string: ')
    print(swap_case(s)) # print(s.swapcase())
```



swapcase1.py

Copy&Run

方法二

```
def swap_case(s):  
    result = ''  
    for c in s:  
        y = c  
        if 'a' <= c <= 'z':  
            y = chr(ord(c) + ord('A') - ord('a'))  
        if 'A' <= c <= 'Z':  
            y = chr(ord(c) + ord('a') - ord('A'))  
        result += y # 如果c不是字母,则无需转换  
    return result  
  
if __name__ == '__main__':  
    s = input('Enter a string: ')  
    print(swap_case(s))
```



swapcase.py

Copy&Run

方法三

```
def swap_case(x): # 转换1个字符
    y = x
    if 'a' <= x <= 'z':
        y = chr(ord(x) + ord('A') - ord('a'))
    if 'A' <= x <= 'Z':
        y = chr(ord(x) + ord('a') - ord('A'))
    return y

if __name__ == '__main__':
    s = input('Enter a string: ')
    result = ''
    for c in s:
        result += swap_case(c)
    print(result)
```





字符串联接函数——join(.)

- `S.join(iterable) -> str`
 - Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

join(.)举例

```
>>> s,t = "ABC","xyz"
>>> "".join([s,t])
'ABCxyz'
>>> "-".join([s,t,s,t])
'ABC-xyz-ABC-xyz'
>>> ' '.join(s)
'A B C'
>>> '*'.join(t)
'x*y*z'
>>> s.join(t)
'xABCyABCz'
>>> t.join(s)
'AxyzBxyzC'
>>>
```

```
>>> '*'.join(('a','b','c','d'))
'a*b*c*d'
>>>
```

注意:join函数只接收一个参数,该参数是一个可迭代对象;返回一个字符串.

[例] 输入一个字符串, 输出以空格间隔的字符序列

```
Enter a string: FudanUniversity  
F u d a n U n i v e r s i t y
```

方法一: for循环

```
s = input('Enter a string: ')  
for each in s:  
    print(each, end=' ')  
print()
```

方法二: join函数

```
s = input('Enter a string: ')  
print(' '.join(s))
```

方法三: 简化方法二

```
print(' '.join(input('Enter a string: ')))
```



❑ 不推荐使用 + 连接字符串, 优先使用join()方法

- 效率原因: 每做一次+, 需要新建一个字符串, 而join()一次性计算出字符串的总长度.

```
def swap_case(s): # 转换1个字符串
    result = [] # 转换的中间结果保存在列表中
    for c in s:
        y = c
        if 'a' <= c <= 'z':
            y = chr(ord(c) + ord('A') - ord('a'))
        elif 'A' <= c <= 'Z':
            y = chr(ord(c) + ord('a') - ord('A'))
        result.append(y)
    return ''.join(result) # 将列表转换为字符串返回

if __name__ == '__main__':
    s = input('Enter a string: ')
    print(swap_case(s))
```





获取字符串表达式的值

□ 内置函数eval()

```
>>> eval("3+4")
7
>>> a = 3
>>> b = 5
>>> eval('a+b')
8
>>> import math
>>> eval('help(math.sqrt)')
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.

>>> eval('math.sqrt(3)')
1.7320508075688772
```




将数字转换成字符串 `str()`

```
>>> s = str(3.4)
>>> s
'3.4'
>>> s = str(3)
>>> s
'3'
>>> s = str(2.3E5)
>>> s
'230000.0'
>>>
```

[例] 列出1000以内的所有回文数(如: 121是回文数, 122不是)

```
def isPalindrome(n):  
    s = str(n) # 将整数转换为字符串  
    return s == s[::-1] # 如果正向/反向字符串相同,说明是回文  
  
if __name__ == '__main__':  
    pa_list = [i for i in range(1000) if isPalindrome(i)]  
    print(pa_list)
```

字符串分割

□ split(...), rsplit(...)

`S.split(sep=None, maxsplit=-1) -> list of strings`

- 以指定字符sep为分隔符, 从字符串左端/右端开始, 将其分割为多个字符串, 返回包含分割结果的列表.
- `split()`与`join()`作用相反.
- `split()` vs `rsplit()`: 第2个参数指定的分隔次数n小于实际分隔符个数时, 才能看出差别.

字符串分割(续)

partition(), rpartition()

- 以指定字符串为分隔符, 将原字符串分割为3部分元组: 分隔符前的字符串、分隔符字符串、分隔符后的字符串.
- 若指定分隔符不在原字符串中, 则返回原字符串和两个空字符串.

```
>>> s="apple,peach,banana,pear"
>>> li=s.split(",")
>>> li
['apple', 'peach', 'banana', 'pear']
>>> s.partition(',')
('apple', ', ', 'peach,banana,pear')
>>> s.rpartition(',')
('apple, peach, banana', ', ', 'pear')
>>> s.rpartition('banana')
('apple,peach,', 'banana', ',pear')
```

```
>>>s="2016-10-31"
>>>t=s.split("-")
>>>print(t)
['2016', '10', '31']
>>>list(map(int, t))
[2016, 10, 31]
>>> s = 'a, b, c,, d, '
>>> s.split(',')
['a', ' b', ' c', ', ', ' d', ' ']
```

split(), rsplit()方法的第一个参数sep: 若不指定分隔符, 则字符串中的任何空白符号(空格/换行符/制表符等)都将被认为是分隔符, 返回包含最终分割结果的列表.

```
>>> s = 'hello world \n\n My name is Dong '
>>> s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>> s = '\n\nhello world \n\n\n My name is Dong '
>>> s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
>>> s = '\n\nhello\t\t world \n\n\n My name\t is Dong '
>>> s.split()
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

split(), rsplit()方法的第二个参数maxsplit: 允许指定最大分割次数.

```
>>> s = '\n\nhello\t\t world \n\n\n My name is Dong '
>>> s.split(None, 1)    # None表示不指定分割符
['hello', 'world \n\n\n My name is Dong ']
>>> s.rsplit(None, 1)
['\n\nhello\t\t world \n\n\n My name is', 'Dong']
>>> s.split(None, 2)
['hello', 'world', 'My name is Dong ']
>>> s.rsplit(None, 2)
['\n\nhello\t\t world \n\n\n My name', 'is', 'Dong']
>>> s.split(None, 5)
['hello', 'world', 'My', 'name', 'is', 'Dong ']
>>> s.split(None, 6)
['hello', 'world', 'My', 'name', 'is', 'Dong']
```

字符串消减

□ strip(), rstrip(), lstrip()

- 删除两端/右端/左端的空格/连续的指定字符.

```
>>> s=" abc "  
>>> s2=s.strip( ) #主要用法：去除字符串两侧的空白符  
>>> s2  
'abc'  
>>> "aaaassddf".strip("a")  
'ssddf'  
>>> "aaaassddf".strip("af")  
'ssdd'  
>>> "aaaassddfaaa".rstrip("a")  
'aaaassddf'  
>>> "aaaassddfaaa".lstrip("a")  
'ssddfaaa'
```

```
>>> x="aabbaxyz"  
>>> x.strip("ab")  
'xyz'  
>>>
```


字符串对齐

center()

- 返回指定宽度的新字符串: 原字符串居中, 并使用指定字符(默认空格)填充.

ljust()/rjust()

- 返回指定宽度的新字符串: 原字符串左/右对齐, 并使用指定字符(默认空格)填充.

```
>>> 'Hello world!'.center(20)
'    Hello world!    '
>>> 'Hello world!'.center(20, '=')
'====Hello world!===='
>>> 'Hello world!'.ljust(20, '=')
'Hello world!===== '
>>> 'Hello world!'.rjust(20, '=')
'=====Hello world! '
>>>
```



打印三角形图案(1)

```
lines = eval(input('Enter an integer: '))
```

左对齐

```
for i in range(lines):
```

```
    stars = '*' * (i+1) # 字符串与整数相乘，构成由'*'组成的字符串
    print(stars)
```

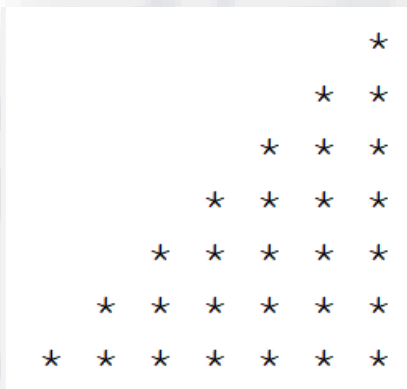
```
Enter an integer: 7
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
```

打印三角形图案(2)

```
lines = eval(input('Enter an integer: '))
```

右对齐

```
for i in range(lines):  
    stars = '*' * (i+1)  
    print(stars.rjust(2 * lines)) # 指定宽度
```



Copy&Run

打印三角形图案(3)

```
n = eval(input('Enter an odd number: '))  
half = n//2+1
```

输出三角形

```
for i in range(half): # i递增  
    stars = '* ' * (i+1)  
    line = stars.rjust(2*half)  
    print(line)  
for i in range(half-1, 0, -1): # i递减  
    stars = '* ' * i  
    line = stars.rjust(2*half)  
    print(line)
```

Enter an odd number: 9

```
          *  
        * *  
      * * *  
    * * * *  
* * * * *  
  * * * *  
    * * *  
      * *  
        *
```

打印三角形图案(4)

输出菱形

```
for i in range(half):  
    stars = '* ' * (i+1)  
    line = stars.center(2*half)  
    print(line)  
for i in range(half-1, 0, -1):  
    stars = '* ' * i  
    line = stars.center(2*half)  
    print(line)
```



Copy&Run

报数出圈

设有编号为 $1 \sim n$ 的 n 个人按顺时针站成一个圆圈.



从第1个人开始, 按顺时针方向从1开始报数, 报到 m 的人出列;



再从出列的下一个开始从1报数, 报到 m 的人出列,, 直到所有人出列.

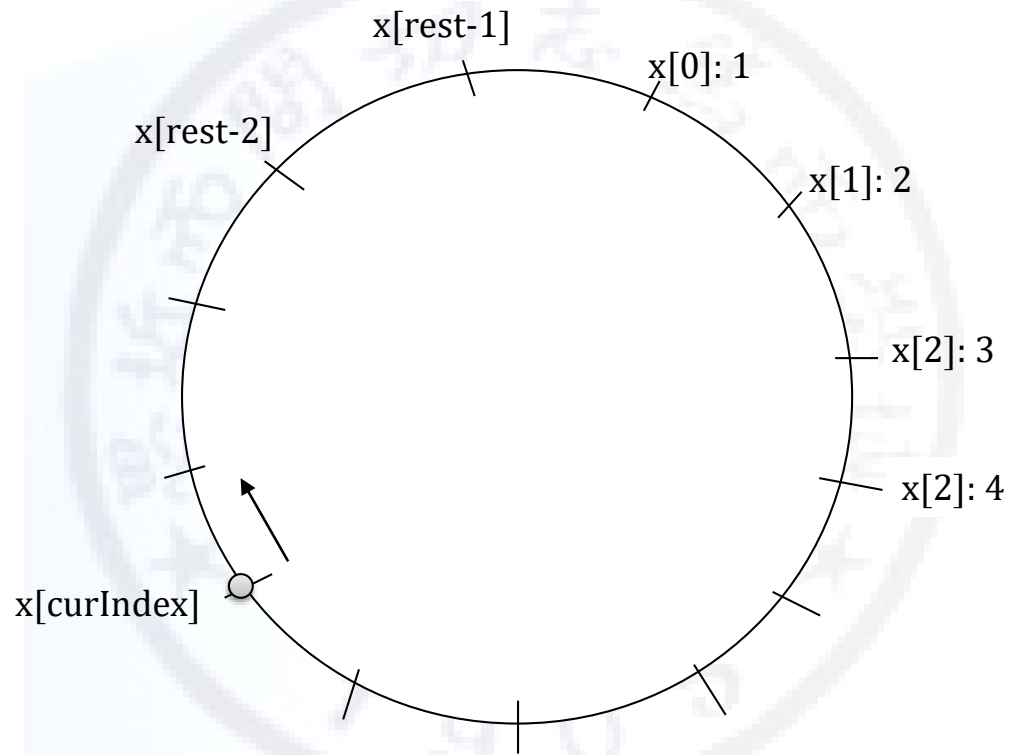


求出列顺序.

解题思路

例: 10个人(编号为1~10)报到3出列, 则出列顺序为: 3 6 9 2 7 1 8 5
10 4

- 某一轮报数开始时, 假设圈里还有 $rest$ 个人, 他们的编号为 $x[0]$, $x[1]$, $x[2]$, ..., $x[rest-1]$. $rest$ 初始为 n .
- 假设本轮报1的人对应的元素下标为 cur ($0 \leq cur < rest$), 则报 m 的人对应的元素下标为 $cur+m-1$, 将其从列表中删除(导致 $rest$ 自动减1).
- 将 cur 更新为 $cur+m-1$, 然后进行下一轮报数.
- 问题: 若 $cur+m-1 \geq rest$, 怎么办?



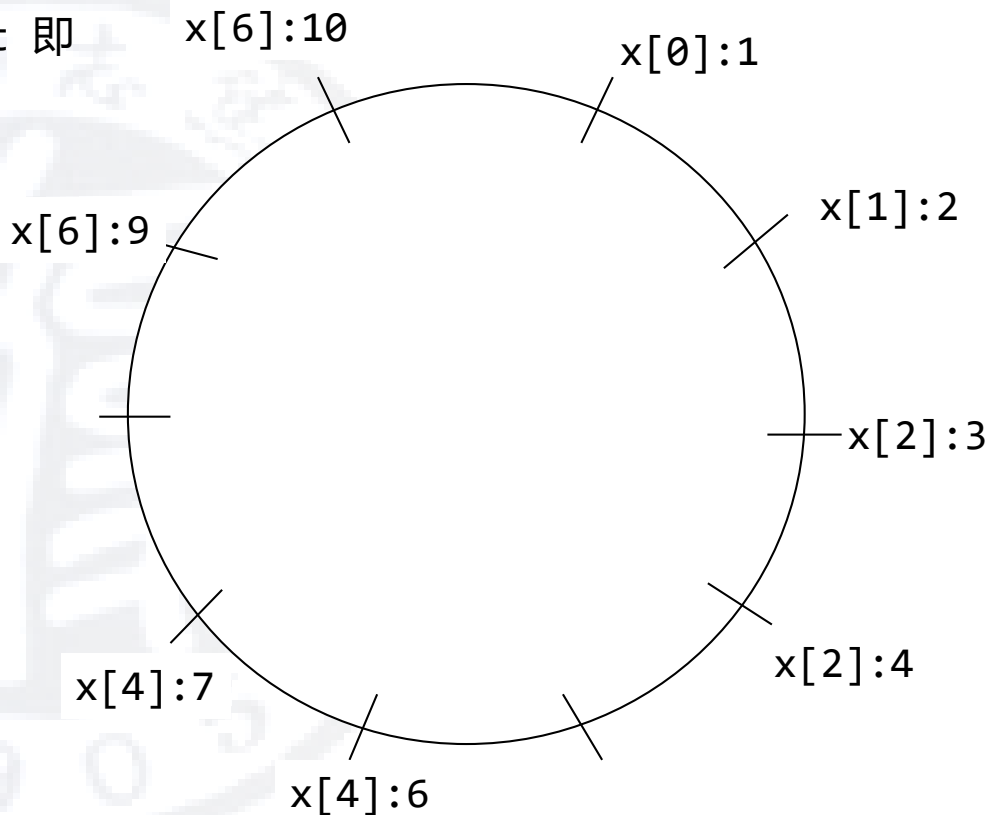

```
tmp = curIndex+m-1
```

```
cur = tmp if tmp < rest else tmp-rest 即
```

```
cur = tmp % rest
```

假设本轮报1的人对应的元素下标为 curIndex ($0 \leq \text{curIndex} < \text{rest}$), 则报 m 的人对应的元素下标为 $(\text{curIndex} + m - 1) \% \text{rest}$, 将其从列表中删除(导致 rest 自动减1).

将 curIndex 更新为 $(\text{curIndex} + m - 1) \% \text{rest}$, 然后进行下一轮报数.



报数出圈

```
n = eval(input('Enter the number of people: '))
```

```
m = eval(input('Enter the key (<%d): ' % n))
```

以格式字符串作input函数参数

```
circle = list(range(1, n+1)) # 保存编号(从1开始)
```

```
result = [] # 保存出列顺序
```

```
cur = 0 # 从第一个人开始报数
```

```
while circle: # 圈不为空时,循环
```

```
    cur = (cur+m-1) % len(circle) # 出列者下标,也是下一轮报1的元素
```

```
    result.append(circle.pop(cur)) ''' pop将下标为cur的元素删除,且返回被删除的元素(即编号),将其添加到结果列表中'''
```

```
print(result)
```



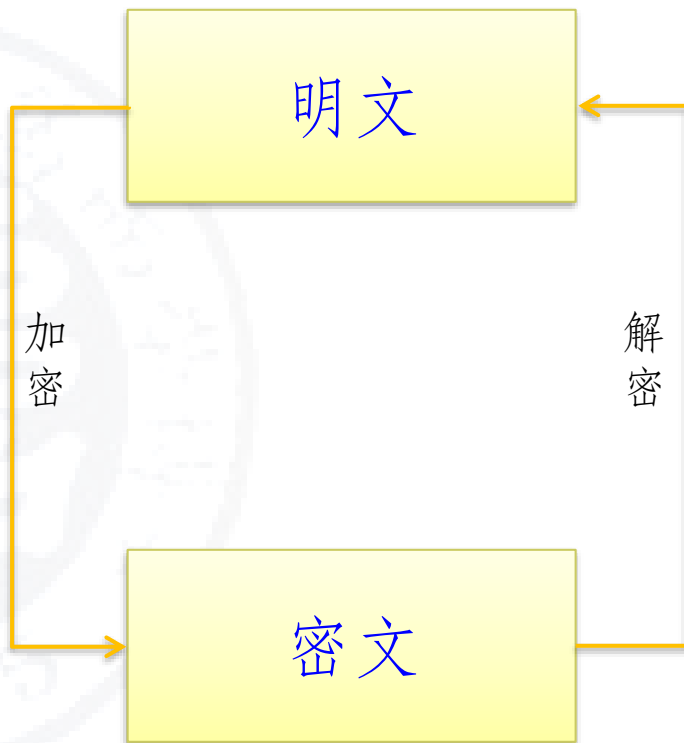
一种简单的加密和解密算法

加密

- 对明文每 m 位提取其字符, 构成密文;
- 到达末尾则回到头部;
- 已经提取的不再提取.
 - 如: "abcde", $m=3$
 - 密文为: "caebd"

解密

- 已知密文和 m , 如何恢复明文?



明文加密

```
original = list(input('Enter a string: ')) # 不再保存编号,而是保存明文
key = eval(input('Enter the key (<%d): ' % (len(original))))
```

```
encryption = [] # 保存密文
```

```
cur = 0
```

```
while original:
```

```
    cur = (cur+key-1) % len(original)
```

```
    encryption.append(original.pop(cur))
```

```
encryption = ''.join(encryption) # 由字符列表构成字符串
```

```
print('After encrypting, the string is: ', encryption)
```

密文解密

假定已知密文字符串 `encryption`.

难点: 寻找密文字符在原文字符串中的位置(下标).

```
original = [None]*len(encryption)    # 定义与密文同样长度的明文列表
```

```
temp = list(range(len(encryption)))    # 密文->明文的'下标环'
```

```
cur = 0
```

```
for c in encryption:
```

```
    cur = (cur+key-1) % len(temp)    # 寻找c在明文中的下标
```

```
    original[temp.pop(cur)] = c    # 将已解密元素下标从"下标环"中删除
```

```
original = ''.join(original)
```

```
print('After deciphering, the string is: ', original)
```





6.1 字符串

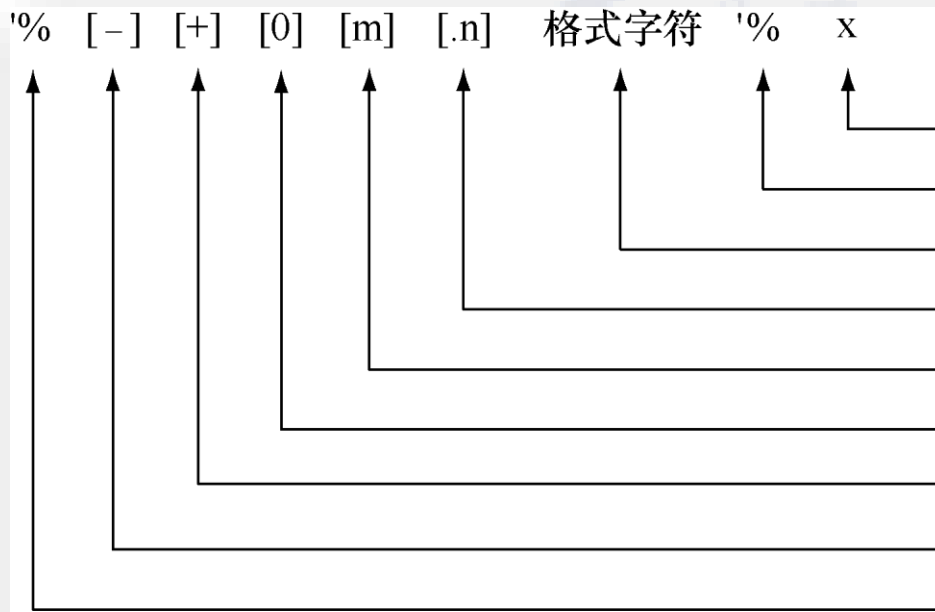
字符串基本知识

字符串常用方法

字符串格式化

精选案例

字符串格式化



- (1) 待转换的表达式;
- (2) 格式运算符;
- (3) 指定类型, 见表5-1;
- (4) 指定精度;
- (5) 指定最小宽度;
- (6) 指定空位填0;
- (7) 对正数加正号;
- (8) 指定左对齐输出;
- (9) 格式标志, 表示格式开始。

□ 常用格式字符——与待格式化的表达式类型匹配

匹配
'%' [-] [+] [0] [m] [.n] 格式字符 '%' x

格式字符	说明	格式字符	说明
%s	字符串(采用str()的显示)	%e	指数(基底写为e)
%r	字符串(采用repr()的显示)	%E	指数(基底写为E)
%c	单个字符	%f、%F	浮点数
%d	十进制整数	%g	指数(e)或浮点数(根据显示长度)
%i	十进制整数(与C语言兼容)	%G	指数(E)或浮点数(根据显示长度)
%o	八进制整数	%%	字符"%"
%x	十六进制整数		


```
>>> x = 1235
>>> "%o" % x
"2323"
>>> "%x" % x
"4d3"
>>> "%e" % x
"1.235000e+03"
>>> "%d" % x
'1235'
>>> "%s" % x
'1235'
```

```
>>> '%f' % 1.23
```

```
'1.230000'
```

```
>>> '%s' % 1.23
```

```
'1.23'
```

```
>>> "%d" % "555"
```

```
Traceback (most recent call last):
```

```
File "<pyshell#19>", line 1, in
<module>
```

```
"%d"% "555"
```

```
TypeError: %d format: a number is
required, not str
```

%常用格式举例

格式	说明
10.2f	格式化浮点数，总宽度为 10 ，四舍五入到小数点后第 2 位
10.2e	以科学计数法表示浮点数，总宽度为 10 ，保留小数点后 2 位
5d	将整数格式化为总宽度为 5 的十进制形式，超出忽略宽度 5 ，不足则补上空格
5o	-----八进制形式，超出忽略宽度 5 ，不足则补上空格
5x	-----十六进制形式，超出忽略宽度 5 ，不足则补上空格
50s	将字符串格式化为总宽度为 50 的字符串，不足则补上空格
50.2s	截取字符串的前 2 位，将其格式化为总宽度为 50 的字符串，不足补空格
-10.2f	向左对齐格式化对象，若无负号(即默认)则是右对齐
+10.2f	格式化浮点数，如果是正数，则自动加上正号+(默认不加正号)



字符串格式化—str.format()

1. 按照参数顺序格式化

一对{}对应一个参数

```
>>> print('His name is {}. He is {}'.format('Lucas', 14))  
His name is Lucas. He is 14.
```

2. 指定参数格式化顺序, 且可以多次使用

```
>>> print('His name is {1}. {1} is {0}'.format(14, 'Lucas'))  
His name is Lucas. Lucas is 14.
```

3. 按照关键参数方式格式化

```
>>> print('His name is {name}. He is {age}'.format(name='Lucas', age=14))  
His name is Lucas. He is 14.
```

4. 使用字典/列表格式化

```
>>> person = {'name': 'Lucas', 'age': 14}
>>> print('His name is {name}. He is {age}.'.format(**person))
His name is Lucas. He is 14.
>>> person = ['Lucas', 14]
>>> print('His name is {}. He is {}'.format(*person))
His name is Lucas. He is 14.
```

5. 通过序列格式化

```
>>> person = ('Lucas', 14)
>>> print('His name is {0[0]}. He is {0[1]}.'.format(person))
His name is Lucas. He is 14.
>>> person = {'name': 'Lucas', 'age': 14}
>>> print('His name is {0[name]}. He is {0[age]}.'.format(person))
His name is Lucas. He is 14.
```

数据格式化 {[填充字符][对齐方式][宽度][.小数位数][修饰符]}

- 由:引导, 后跟填充字符, 只能是一个字符, 默认用空格填充.
- ^、<、> 分别代表居中、左对齐、右对齐, 后跟宽度.
- 修饰符: , % e

数字	格式	结果	描述
3.1415926	{:.2}	'3.14'	保留小数点后两位
3.1415926	{:+.2}	'+3.14'	带符号保留小数点后两位
5	{:0>2}	'05'	补零 (填充左边, 宽度为2)
5	{:x<4}	'5xxx'	补x (填充右边, 宽度为4)
3.1415926	{:x^10.2f}	'xxx3.14xxx'	居中对齐 (宽度为10)
1000000	{:,}	'1,000,000'	以逗号分隔的数字格式
0.25	{:.2%}	'25.00%'	百分比格式
1000000000	{:.2e}	'1.00e+09'	指数记法

进制转换

格式	结果	说明
<code>'{:b}'.format(11)</code>	<code>'1011'</code>	二进制
<code>'{:#b}'.format(11)</code>	<code>'0b1011'</code>	
<code>'{:o}'.format(11)</code>	<code>'13'</code>	八进制
<code>'{:#o}'.format(11)</code>	<code>'0o13'</code>	
<code>'{:x}'.format(11)</code>	<code>'b'</code>	十六进制
<code>'{:#x}'.format(11)</code>	<code>'0xb'</code>	
<code>'{:#X}'.format(11)</code>	<code>0XB</code>	

```
weather = [("Monday", "rainy"), ("Tuesday", "sunny"),  
           ("Wednesday", "sunny"), ("Thursday", "rainy"),  
           ("Friday", "Cloudy")]  
formatter = "Weather of '{0[0]}' is '{0[1]}'. ".format # 方法对象  
for item in weather:  
    print(formatter(item))  
for item in map(formatter, weather):  
    print(item)
```

```
Weather of 'Monday' is 'rainy'.  
Weather of 'Tuesday' is 'sunny'.  
Weather of 'Wednesday' is 'sunny'.  
Weather of 'Thursday' is 'rainy'.  
Weather of 'Friday' is 'Cloudy'.
```



str.format()的优点

在%方法中必须指定数据项对应的格式字符, 而在format中不需要理会数据类型.

format的参数可以多次输出, 且参数顺序可指定.

format的填充方式十分灵活, 对齐方式十分强大.

推荐使用format, %方式将会在后面的版本被淘汰.

查找方法

find(), rfind()

- 查找一个字符串在另一个字符串指定范围(默认是整个字符串)中首次/最后一次出现的位置, 若不存在则返回-1.

index(), rindex()

- 返回一个字符串在另一个字符串指定范围中首次/最后一次出现的位置, 若不存在则抛出异常.

count()

- 返回一个字符串在另一个字符串中出现的次数.

```
>>> s="apple,peach,banana,peach,  
    pear"  
>>> s.find("peach")  
6  
>>> s.find("peach", 7)  
19  
>>> s.find("peach", 7, 20)  
-1 # 没找到返回-1  
>>> s.rfind('p')  
25  
>>> s.index('p')  
1  
>>> s.index('pe')  
6
```

```
>>> s.index('pear')  
25  
>>> s.index('ppp')#没找到抛出异常  
Traceback (most recent call  
    last):  
  File "<pyshell#11>", line 1, in  
    <module>  
    s.index('ppp')  
ValueError: substring not found  
>>> s.count('p')  
5  
>>> s.count('pp')  
1  
>>> s.count('ppp')  
0
```

字符串查找替换 `replace()`

□ `S.replace(old, new[, count])` -> str

- Return a copy of `S` with all occurrences of substring *old* replaced by *new*.
- If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
>>> prog = '''a, b = 0, 1
for i in range(n):
    a, b = b, a+b
    print(a, end = ' ')
...'''
```

```
>>> prog = prog.replace('a', 'first')
>>> prog = prog.replace('b', 'second')
>>> print(prog)
first, second = 0, 1
for i in rfirstinge(n):
    first, second = second, first+second
    print(first, end = ' ')
```



判断字符串是否以指定字符串开始或结束

□ `s.startswith(t)`, `s.endswith(t)`

```
>>> import os
```

```
>>> [filename for filename in os.listdir('c: \\') if  
    filename.endswith(('.bmp', '.jpg', '.gif'))]
```

字符串转换

□ 生成映射表函数maketrans(), 按映射表关系转换字符串函数translate()

- Python3.4以上版本将maketrans()从string的方法改为str的方法。

```
>>> table=str.maketrans("abcdef123", "uvwxyz@#$")
>>> s="Python is a great programming language. I like it!"
>>> s.translate(table)
'Python is u gryut programming lunguugy. I liky it!'
>>> table=str.maketrans("abcdef123", "uvwxyz@#$", "gtm")
>>> s.translate(table)           #第3个参数表示要删除的字符
'Pyhon is u ryu proruin lunuuy. I liky i!'
```

4.2 正则表达式

正则表达式元字符

re模块主要方法

直接使用re模块方法

使用正则表达式对象

子模式与match对象

应用案例

正则表达式

- 字符串处理的有力工具和技术, 为文本处理提供了功能强大、灵活而又高效的方法.
 - 快速分析大量文本以找到特定的、符合某些复杂规则(亦称模式)的字符串
- e.g. 1) 从英文小说中查找hi, 但him, high, history,...不算
- 2) 查找后面不远处跟着一个Lucy的hi
- 提取/编辑/替换/删除文本子字符串
- 将提取的字符串添加到集合以生成报告



正则表达式就是用于描述这些规则(模式)的语言.

广泛地应用于各种字符串处理应用程序: 网页处理, 日志文件分析等.

正则表达式引擎

□ re模块提供了正则表达式操作所需要的功能

e.g. `re.findall(pattern, string)`: 返回列表, 列出string中匹配pattern的所有项.

```
>>> import re
>>> re.findall('d', 'godness')
['d']
>>> re.sub('d', 'od', 'godness')
'goodness'
>>>
```

```
>>> 'godness'.find('d')
2
>>> 'godness'.replace('d', 'od')
'goodness'
```

```
>>> prog = '''a, b = 0, 1
for i in range(n):
    a, b = b, a+b
    print(a, end = ' ')
'''
```

```
>>> print(prog)
a, b = 0, 1
for i in range(n):
    a, b = b, a+b
    print(a, end = ' ')
```

```
>>> prog = prog.replace('a', 'first')
>>> prog = prog.replace('b', 'second')
>>> print(prog)
first, second = 0, 1
for i in range(n):
    first, second = second, first+second
    print(first, end = ' ')
```

```
>>> prog = '''a, b = 0, 1
for i in range(n):
    a, b = b, a+b
    print(a, end = ' ')
'''
```

```
>>> print(prog)
a, b = 0, 1
for i in range(n):
    a, b = b, a+b
    print(a, end = ' ')
```

仅替换作为完整单词的'a'

```
>>> prog = re.sub(r'\ba\b', 'first', prog)
>>> prog = re.sub(r'\bb\b', 'second', prog)
>>> print(prog)
first, second = 0, 1
for i in range(n):
    first, second = second, first+second
    print(first, end = ' ')
```

正则表达式组成

- 由普通字符(包括转义字符)、特殊字符(称为**元字符**)及其不同组合构成.

e.g. 1) 从英文小说中查找hi, 但him, high, history, ...不算

正则表达式: `\bhi\b`

元字符

2) 查找后面不远处跟着一个Lucy的hi

正则表达式: `\bhi\b.*\bLucy\b`

普通字符

普通字符(包括转义字符)

- 基本的正则表达式由单个/多个普通字符组成, 用以匹配字符串中对应的单个/多个普通字符.
- 普通字符包括ASCII字符, Unicode字符和转义字符
- 由于'^\$. *?+-\{\}[]| ()'被正则表达式用作元字符, 如作为普通字符使用, 需要转义.

正则表达式	字符串	说明
'fo'	'The quick brown fox jumps for food'	匹配其中含有'fo'的3个字符串
'1+1=2'	'1+1=2'	+为元字符, 无法匹配
'1\+1=2'	'1+1=2'	['1+1=2']
'(note)'	'please(note)'	()为元字符, 匹配"note"
'\ (note\)'	'please(note)'	匹配"(note)"

Python 语 句	匹 配 结 果
<code>re.findall('fo', 'The quick brown fox jumps for food')</code>	<code>['fo', 'fo', 'fo']</code>
<code>re.findall('1+1=2', '1+1=2')</code>	<code>[]</code>
<code>re.findall('1\\+1=2', '1+1=2')</code>	<code>['1+1=2']</code>
<code>re.findall('(note)', 'please(note)')</code>	<code>['note']</code>
<code>re.findall('\\(note\\)', 'please(note)')</code>	<code>['(note)']</code>

□ 一个正则表达式的应用示例

```
>>> import re
```

`r'`: 原始字符串

`\d`: 匹配任何数字字符

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

```
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
```

```
>>> print('Phone number found: ' + mo.group())
```

Phone number found: 415-555-4242

❑ Review of Regular Expression Matching

While there are several steps to using regular expressions in Python, each step is fairly simple.

- Import the regex module with *import re*. `import re`
- Create a Regex object with the *re.compile()* function. (Remember to use a raw string.) `phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')`
- Pass the string you want to search into the **Regex object's** *search()* method. This returns a Match object.

```
mo = phoneNumRegex.search('My number is 415-555-4242.')
```

- Call the **Match object's** *group()* method to return a string of the actual matched text. `print('Phone number found: ' + mo.group())`

4.2.1 正则表达式元字符

正则表达式元字符分类:

- 字符类 `[] - ^`
- 预定义字符类 `.\d\D\s\S\w\W`
- 边界匹配符 `^ $\b\B`
- 重复限定符 `{ } + * ?`
- 分组符 `()`
- 选择符 `|`

字符类[] - ^

□ 由一对[]括起来的字符集合

元字符	说明
[]	匹配位于[]中的任意一个字符
-	用在[]之内表示范围
^	用在[]之内表示否定

- [xyz]: 枚举字符集, 匹配括号中任意字符

e.g. '[pjc]ython'可匹配'python'、'jython'、'cython'

- [^xyz]: 否定枚举字符集, 匹配不在括号中的任意字符

e.g. '[^abc]'可匹配除'a'、'b'、'c'之外的一个任意字符

- [a-z]: 指定范围的字符, 匹配指定范围的任意字符

e.g. '[a-zA-Z0-9]'可匹配任意一个字母或数字

- [^m-z]: 匹配指定范围以外的任意字符

```
e.g. >>> re.findall('[0-9]', 'a12b34c56')
['1', '2', '3', '4', '5', '6']
>>> re.findall('[^0-9]', 'a12b34c56')
['a', 'b', 'c']
```

预定义字符类 . \d \D \s \S \w \W

□ 经常用到的一些特定字符类


元字符	说明
.	匹配除换行符以外的任意单个字符
\d	匹配单个数字, 相当于[0-9]
\D	与\d含义相反, 匹配单个非数字, 相当于[^0-9]
\s	匹配单个空白字符, 相当于[\t\n\r\f\v]
\S	与\s含义相反, 相当于[^ \t\n\r\f\v]
\w	匹配单个字母/数字/下划线, 相当于[a-zA-Z0-9_]
\W	与\w含义相反, 相当于[^a-zA-Z0-9_]

```
>>> re.findall('\d', 'a12b34c56')    # 找出所有数字
['1', '2', '3', '4', '5', '6']
>>> re.findall('.', 'pi=3.14')        # 找出所有字符
['p', 'i', '=', '3', '.', '1', '4']
>>> re.findall('\.', 'pi=3.14')       # 找出所有小数点
['.']
>>> re.findall('[\d.]', 'pi=3.14')    # 找出所有数字和小数点
['3', '.', '1', '4']
>>> re.findall('\d.', 'pi=3.14')     # 找出由1个数字和1个字符构成的字符序列
['3.', '14']
>>> re.findall('\d\.', 'pi=3.14')    # 找出由1个数字和1个小数点构成的字符序列
['3.']
```

- Note that inside the square brackets, the normal regular expression symbols are not interpreted as such. This means you do not need to escape the '\$.*?+{}|()' characters with a preceding backslash.

边界匹配符 ^ \$ \b \B

- 字符串匹配往往涉及从某个位置开始匹配, 如: 行的开头或结尾、单词的开头或结尾等, 边界匹配符用于位置的匹配。

元字符	说明
^	匹配行首以^之后字符开头的字符串, 如: "^a" 匹配"abc"中的"a", 不匹配"bat"中的"a"
\$	匹配行尾以\$之前字符结束的字符串, 如: "c\$" 匹配"abc"中的"c", 不匹配"acb"中的"c"
\b	匹配单词头或单词尾, 如: "\\bfoo\\b" 匹配"foo", "foo.", "(foo)", "bar foo baz" 但不匹配"foobar", "foo3"  '\b'在正则表达式中表示单词边界, 而在字符串中表示退格字符⇨与标准转义字符重复的元字符必须使用\\, 也可使用原始字符串r""或r", 即r"\bfoo\b" 或 r'\bfoo\b'
\B	与\b含义相反, 如: 'py\B' 匹配 "python", "py3", "py2", 但不匹配"happy", "sleepy", "py!"