



Python语言

第五章 字符串与正则表达式

5.1 字符串基础

5.2 字符串方法

5.3 正则表达式

5.4 实验

5.5 小结

习题

字符串常用的表示方式

- 1、字符串中的字符可以是ASCII字符也可以是其他各种符号。
- 2、它常用英文状态下的单引号（' '）、双引号（" "）或者三单引号（''' '''）、三双引号（""" """）进行表示。

英文字符 ASCII

ASCII表									
American Standard Code for Information Inter									
ASCII控制字符									
十进制	字符	十六进制	十进制	字符	十六进制	十进制	字符	十六进制	十进制
0		00	1		01	2		02	3
4		04	5		05	6		06	7
8		08	9		09	10		0A	11
12		0C	13		0D	14		0E	15
16		10	17		11	18		12	19
20		14	21		15	22		16	23
24		18	25		19	26		1A	27
28		1C	29		1D	30		1E	31
32		20	33		21	34		22	35
36		24	37		25	38		26	39
40		28	41		29	42		2A	43
44		2C	45		2D	46		2E	47
48		30	49		2F	50		20	51
52		34	53		35	54		22	55
56		38	57		39	58		24	59
60		3C	61		3A	62		26	63
64		40	65		3B	66		28	67
68		44	69		3D	6A		2A	6B
72		48	73		3E	6C		2C	6D
76		4C	77		3F	6E		2E	6F
80		50	81		40	68		20	70
84		54	85		41	69		22	71
88		58	89		42	6A		24	72
92		5C	93		43	6B		26	73
96		60	97		44	6C		28	74
100		64	98		45	6D		2A	75
104		68	99		46	6E		2C	76
108		6C	100		47	6F		2E	77
112		70	101		48	70		20	78
116		74	102		49	71		22	79
120		78	103		4A	72		24	7A
124		7C	104		4B	73		26	7B
128		80	105		4C	74		28	7C
132		84	106		4D	75		2A	7D
136		88	107		4E	76		2C	7E
140		8C	108		4F	77		2E	7F
144		90	109		50	78		20	80
148		94	110		51	79		22	81
152		98	111		52	7A		24	82
156		9C	112		53	7B		26	83
160		A0	113		54	7C		28	84
164		A4	114		55	7D		2A	85
168		A8	115		56	7E		2C	86
172		AC	116		57	7F		2E	87
176		B0	117		58	80		20	88
180		B4	118		59	81		22	89
184		B8	119		5A	82		24	8A
188		BC	120		5B	83		26	8B
192		C0	121		5C	84		28	8C
196		C4	122		5D	85		2A	8D
200		C8	123		5E	86		2C	8E
204		CC	124		5F	87		2E	8F
208		D0	125		60	88		20	90
212		D4	126		61	89		22	91
216		D8	127		62	8A		24	92
220		DC	128		63	8B		26	93
224		E0	129		64	8C		28	94
228		E4	130		65	8D		2A	95
232		E8	131		66	8E		2C	96
236		EC	132		67	8F		2E	97
240		F0	133		68	90		20	98
244		F4	134		69	91		22	99
248		F8	135		6A	92		24	9A
252		FC	136		6B	93		26	9B
256		00	137		6C	94		28	9C
260		04	138		6D	95		2A	9D
264		08	139		6E	96		2C	9E
268		0C	140		6F	97		2E	9F
272		10	141		70	98		20	A0
276		14	142		71	99		22	A1
280		18	143		72	9A		24	A2
284		1C	144		73	9B		26	A3
288		20	145		74	9C		28	A4
292		24	146		75	9D		2A	A5
296		28	147		76	9E		2C	A6
300		2C	148		77	9F		2E	A7
304		30	149		78	A0		20	A8
308		34	150		79	A1		22	A9
312		38	151		7A	A2		24	AA
316		3C	152		7B	A3		26	AB
320		40	153		7C	A4		28	AC
324		44	154		7D	A5		2A	AD
328		48	155		7E	A6		2C	AE
332		4C	156		7F	A7		2E	AF
336		50	157		80	A8		20	B0
340		54	158		81	A9		22	B1
344		58	159		82	AA		24	B2
348		5C	160		83	AB		26	B3
352		60	161		84	AC		28	B4
356		64	162		85	AD		2A	B5
360		68	163		86	AE		2C	B6
364		6C	164		87	AF		2E	B7
368		70	165		88	B0		20	B8
372		74	166		89	B1		22	B9
376		78	167		8A	B2		24	BA
380		7C	168		8B	B3		26	BB
384		80	169		8C	B4		28	BC
388		84	170		8D	B5		2A	BD
392		88	171		8E	B6		2C	BE
396		8C	172		8F	B7		2E	BF
400		90	173		90	B8		20	C0
404		94	174		91	B9		22	C1
408		98	175		92	BA		24	C2
412		9C	176		93	BB		26	C3
416		A0	177		94	BC		28	C4
420		A4	178		95	BD		2A	C5
424		A8	179		96	BE		2C	C6
428		AC	180		97	BF		2E	C7
432		B0	181		98	C0		20	C8
436		B4	182		99	C1		22	C9
440		B8	183		9A	C2		24	CA
444		BC	184		9B	C3		26	CB
448		C0	185		9C	C4		28	CC
452		C4	186		9D	C5		2A	CD
456		C8	187		9E	C6		2C	CE
460		CC	188		9F	C7		2E	CF
464		D0	189		A0	C8		20	D0
468		D4	190		A1	C9		22	D1
472		D8	191		A2	CA		24	D2
476		DC	192		A3	CB		26	D3
480		E0	193		A4	CC		28	D4
484		E4	194		A5	CD		2A	D5
488		E8	195		A6	CE		2C	D6
492		EC	196		A7	CF		2E	D7
496		F0	197		A8	D0		20	D8
500		F4	198		A9	D1		22	D9
504		F8	199		AA	D2		24	DA
508		FC	200		AB	D3		26	DB
512		00	201		AC	D4		28	DC
516		04	202		AD	D5		2A	DD
520		08	203		AE	D6		2C	DE
524		0C	204		AF	D7		2E	DF
528		10	205		B0	D8		20	E0
532		14	206		B1	D9		22	E1
536		18	207		B2	DA		24	E2
540		1C	208		B3	DB		26	E3
544		20	209		B4	DC		28	E4
548		24	210		B5	DD		2A	E5
552		28	211		B6	DE		2C	E6
556		2C	212		B7	DF		2E	E7
560		30	213		B8	E0		20	E8
564		34	214		B9	E1		22	E9
568		38	215		BA	E2		24	EA
572		3C	216		BB	E3		26	EB
576		40	217		BC	E4		28	EC
580		44	218		BD	E5		2A	ED
584		48	219		BE	E6		2C	EE
588		4C	220		BF	E7		2E	EF
592		50	221		C0	E8		20	F0
596		54	222		C1	E9		22	F1
600		58	223		C2	EA		24	F2
604		5C	224		C3	EB		26	F3
608		60	225		C4	EC		28	F4
612		64	226		C5	ED		2A	F5
616		68	227		C6	EE		2C	F6
620		6C	228		C7	EF		2E	F7
624		70	229		C8	F0		20	F8
628		74	230		C9	F1		22	F9
632		78	231		CA	F2		24	FA
636		7C	232		CB	F3		26	FB
640		80	233		CC	F4		28	FC
644		84	234		CD	F5		2A	FD
648		88	235		CE	F6		2C	FE
652		8C	236		CF	F7		2E	FF

#输出 ASCII

编码 0000-00FF

```
print ('{0:^6}'.format(' '),end = ' ')
for j in range(0,0x10):
    print ('{0:<5X}'.format(j),end = ' ')
print ()
```

#ASCII

0020-00FF

```
for i in range (0X0020,0x00FF,0x10):
    print ('{0:<6X}'.format(i),end = ' ')
    for j in range(0,0x10):
        #print (j)
        print ('{0:<5c}'.format(i+j),end = ' ')
    print ('')
```

转义字符

字符串中还有一种特殊的字符叫做转义字符，转义字符通常用于不能够直接输入的各种特殊字符。Python常用转义字符如表5.1所示：

转义字符	说明
\\	反斜线
\'	单引号
\"	双引号
\a	响铃符
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	水平制表符
\v	垂直制表符
\0	Null，空字符串
\000	以八进制表示的ASCII码对应符
\xhh	以十六进制表示的ASCII码对应符

表5.1用的转义字符

字符串的基础操作包括。

求字符串的长度、字符串的连接、字符串的遍历、字符串的包含判断、字符串的索引和切片等。

1、求字符串的长度

字符串的长度是指字符数组的长度，又可以理解为字符串中的字符个数（空格也算字符），可以用len()函数查看字符串的长度。如：

```
sample_str1 = 'Jack loves Python'
print(len(sample_str1))          #查看字符串长度
```

运行结果如下：

17

2、字符串的连接

字符串的连接是指将多个字符串连接在一起组成一个新的字符串。例如：

```
sample_str2 = ('Jack', 'is', 'a', 'Python', 'fan') #字符串用逗号隔开，组成元组  
print('sample_str2:', sample_str2, type(sample_str2))
```

运行结果如下：

```
sample_str2: ('Jack', 'is', 'a', 'Python', 'fan') <class 'tuple'>
```

当字符串之间**没有任何连接符**时，这些字符串会直接连接在一起，组成新的字符串。

```
sample_str3 = 'Jack"is"a"Python"fan' #字符串间无连接符，默认合并  
print('sample_str3: ', sample_str3)
```

运行结果如下：

```
sample_str3: JackisaPythonfan
```

字符串之间用 '+' 号连接时，也会出现同样的效果，这些字符串将连接在一起，组成一个新的字符串。

```
sample_str4 = 'Jack' + 'is' + 'a' + 'Python' + 'fan'
```

#字符串 '+' 连接，默认合并

```
print('sample_str4: ', sample_str4)
```

运行结果如下：

```
sample_str4: JackisaPythonfan
```

用字符串与正整数进行乘法运算时，相当于创建对应次数的字符串，最后组成一个新的字符串。

```
sample_str5 = 'Jack'*3          #重复创建相应的字符串
```

```
print('sample_str5: ', sample_str5)
```

运行结果如下：

```
sample_str5: JackJackJack
```

注意：字符串直接以空格隔开的时候，该字符串会组成元组类型。

操作符及使用	描述
$x + y$	连接两个字符串x和y
$n * x$ 或 $x * n$	复制n次字符串x
$x \text{ in } s$	如果x是s的子串，返回True，否则返回False

获取星期字符串

```
#WeekNamePrintV2.py
```

```
weekStr = "一二三四五六日"
```

```
weekId = eval(input("请输入星期数字(1-7) : "))
```

```
print("星期" + weekStr[weekId-1])
```

3、字符串的遍历

通常使用for循环对字符串进行遍历。例如：

```
sample_str6 = 'Python'      #遍历字符串
for a in sample_str6:
    print(a)
```

运行结果如下：

P
y
t
h
o
n

其中变量a，每次循环按顺序代指字符串里面的一个字符。

4、字符串的包含判断

字符串是字符的有序集合，因此用in操作来判断指定的字符是否存在包含关系。如：

```
sample_str7 = 'Python'
```

```
print('a' in sample_str7)
```

#字符串中不存在包含关系

```
print('Py' in sample_str7)
```

#字符串中存在包含关系

运行结果如下：

False

True

5、索引和切片

字符串是一个有序集合，因此可以通过偏移量实现索引和切片的操作。在字符串中字符从左到右的字符索引依次为0, 1, 2, 3, ..., len()-1, 字符从右到左的索引依次为-1, -2, -3, ..., -len()。索引其实简单来说是指字符串的排列顺序，可以通过索引来查找该顺序上的字符。例如：

```
sample_str8 = 'Python'
```

```
print(sample_str8[0])
```

#字符串对应的第一个字符

```
print(sample_str8[1])
```

#字符串对应的第二个字符

```
print(sample_str8[-1])
```

#字符串对应的最后一个字符

```
print(sample_str8[-2])
```

#字符串对应的倒数第二个字符

运行结果如下：

P

y

n

o

注意：虽然索引可以获得该顺序上的字符，但是不能够通过该索引去修改对应的字符。例如：

```
sample_str8[0] = 'b'           #修改字符串的第一个字符
```

Traceback (most recent call last): #系统正常报错

File "<pyshell#4>", line 9, in <module>

```
sample_str8[0] = 'b'
```

TypeError: 'str' object does not support item assignment

切片，也叫分片，和元组与列表相似，是指从某一个索引范围中获取连续的多个字符（又称为子字符）。常用格式如下：

stringname[start:end]

这里的stringname是指被切片的字符串，start和end分别指开始和结束时字符的索引，其中切片的最后一个字符的索引是end-1，这里有一个诀窍叫：包左不包右。例如：

```
sample_str9 = 'abcdefghijkl'
```

```
print(sample_str9[0:4]) #获取索引为0-4之间的字符串，从索引0开始到3为止，不包括索引为4的字符  
print(sample_str9[0:10:2]) ??
```

运行结果如下：

```
abcd
```

字符串切片高级用法

使用[M: N: K]根据步长对字符串切片

- <字符串>[M: N] , M缺失表示**至开头** , N缺失表示**至结尾**

"〇一二三四五六七八九十"[:3] 结果是 "〇一二"

- <字符串>[M: N: K] , 根据步长K对字符串切片

"〇一二三四五六七八九十"[1:8:2] 结果是 "一三五七"

"〇一二三四五六七八九十"[::-1] 结果是 "十九八七六五四三二一〇"

若不指定起始切片的索引位置，默认是从0开始；若不指定结束切片的顺序，默认是字符串的长度-1。例如：

```
sample_str10 = 'abcdefg'
```

```
print("起始不指定", sample_str10[:3]) #获取索引为0-3之间的字符串，不包括3
```

```
print("结束不指定", sample_str10[3:]) # 从索引3到最后一个字符，不包括len
```

运行结果如下：

起始不指定 abc

结束不指定 defg

默认切片的字符串是连续的，但是也可以通过指定步进数（step）来跳过中间的字符，其中默认的step是1。例如指定步进数为2：

```
sample_str11 = '012345678'
```

```
print('跳2个字符', sample_str11[1:7:2]) #索引1~7，每2个字符截取
```

运行结果如下：

跳2个字符 135

Unicode编码

Python字符串的编码方式

- 统一字符编码，即覆盖几乎所有字符的编码方式
- 从0到1114111 (0x10FFFF)空间，每个编码对应一个字符
- Python字符串中每个字符都是Unicode编码字符

Unicode编码

一些有趣的例子

```
>>> "1 + 1 = 2 " + chr(10004)
```

```
'1 + 1 = 2 ✓'
```

```
>>> "这个字符𐄀的Unicode值是：" + str(ord("𐄀"))
```

```
'这个字符𐄀的Unicode值是： 9801'
```

```
>>> for i in range(12):
```

```
    print(chr(9800 + i), end="")
```

```
𐄀𐄁𐄂𐄃𐄄𐄅𐄆𐄇𐄈𐄉𐄊𐄋
```

```
#汉字笔画      36字      31C0-31E3
print ('{0:^6}'.format(' '),end = '')
for j in range(0,0x10):
    print ('{0:<5X}'.format(j),end = '')
print ()

#基本汉字      20902字      4E00-9FA5
#for i in range (0x4E00,0x5E00,0x10):
#汉字笔画      36字      31C0-31E3
for i in range (0x31C0 ,0x31F0,0x10):
    print ('{0:<6X}'.format(i),end = '')
    for j in range(0,0x10):
        #print (j)
        print ('{0:<4c}'.format(i+j),end = '')
    print ('')
```

字符集	字数	Unicode 编码
基本汉字	20902字	4E00-9FA5
基本汉字补充	74字	9FA6-9FEF
扩展A	6582字	3400-4DB5
扩展B	42711字	20000-2A6D6
扩展C	4149字	2A700-2B734
扩展D	222字	2B740-2B81D
扩展E	5762字	2B820-2CEA1
扩展F	7473字	2CEB0-2EBE0
扩展G	4939字	30000-3134A
康熙字典部首	214字	2F00-2FD5
部首扩展	115字	2E80-2EF3
兼容汉字	477字	F900-FAD9
兼容扩展	542字	2F800-2FA1D
PUA(GBK)部件	81字	E815-E86F
部件扩展	452字	E400-E5E8
PUA增补	207字	E600-E6CF
汉字笔画	36字	31C0-31E3
汉字结构	12字	2FF0-2FFB
汉语注音	43字	3105-312F
注音扩展	22字	31A0-31BA

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
31C0	ノ	ㄣ	ㄥ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ
31D0	一	丨	ノ	ㄣ	ㄥ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ	ㄩ	ㄣ
31E0	乙	ㄣ	ノ	ㄣ	ㄥ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ	ㄩ	ㄣ	ㄤ	ㄨ	ㄩ	ㄣ

第五章 字符串与正则表达式

5.1 字符串基础

5.2 字符串方法

5.3 正则表达式

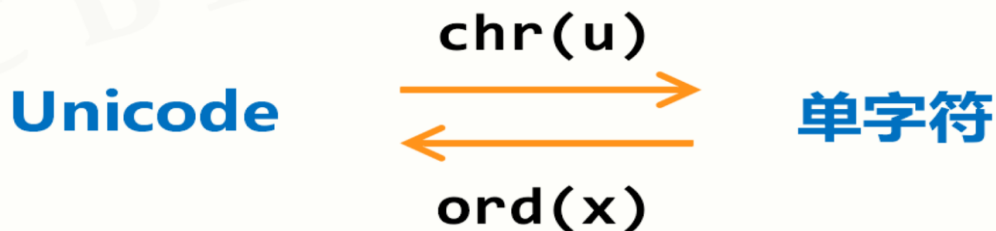
5.4 实验

5.5 小结

习题

一些以函数形式提供的字符串处理功能

函数及使用	描述
<code>len(x)</code>	长度，返回字符串x的长度 <code>len("一二三456")</code> 结果为 6
<code>str(x)</code>	任意类型x所对应的字符串形式 <code>str(1.23)</code> 结果为"1.23" <code>str([1,2])</code> 结果为"[1,2]"
<code>hex(x)</code> 或 <code>oct(x)</code>	整数x的十六进制或八进制小写形式字符串 <code>hex(425)</code> 结果为"0x1a9" <code>oct(425)</code> 结果为"0o651"
<code>chr(u)</code>	x为Unicode编码，返回其对应的字符
<code>ord(x)</code>	x为字符，返回其对应的Unicode编码



字符串是str类型对象，所以Python内置了一系列操作字符串的方法。其中常用的方法如下：

方法及使用 2/3	描述
<code>str.replace(old, new)</code>	返回字符串str副本，所有old子串被替换为new <code>"python".replace("n", "n123.io")</code> 结果为 <code>"python123.io"</code>
<code>str.center(width[, fillchar])</code>	字符串str根据宽度width居中，fillchar可选 <code>"python".center(20, "=")</code> 结果为 <code>'=====python====='</code>
<code>str.strip(chars)</code>	从str中去掉在其左侧和右侧chars中列出的字符 <code>"= python= ".strip(" =np")</code> 结果为 <code>"ytho"</code>
<code>str.join(iter)</code>	在iter变量除最后元素外每个元素后增加一个str <code>", ".join("12345")</code> 结果为 <code>"1,2,3,4,5"</code> #主要用于字符串分隔等

字符串是str类型对象，所以Python内置了一系列操作字符串的方法。其中常用的方法如下：

方法及使用 2/3	描述
<code>str.replace(old, new)</code>	返回字符串str副本，所有old子串被替换为new <code>"python".replace("n", "n123.io")</code> 结果为 <code>"python123.io"</code>
<code>str.center(width[, fillchar])</code>	字符串str根据宽度width居中，fillchar可选 <code>"python".center(20, "=")</code> 结果为 <code>'=====python====='</code>

字符串是str类型对象，所以Python内置了一系列操作字符串的方法。其中常用的方法如下：

1. str.strip([chars])

若方法里面的chars不指定默认去掉字符串的首、尾空格或者换行符，但是如果指定了chars，那么会删除首尾的chars例如：

```
>>> sample_fun1 = ' Hello world^#'
```

```
>>> print(sample_fun1.strip())
```

#默认去掉首尾空格

```
>>> print(sample_fun1.strip('#'))
```

#指定首尾需要删除的字符

```
>>> print(sample_fun1.strip('^#'))
```

运行结果如下：

```
Hello world^#
```

```
Hello world^
```

```
Hello world
```


2. str.count('chars',start,end)

统计chars字符串或者字符在str中出现的次数，从start顺序开始查找一直到end顺序范围结束，默认是从顺序0开始。例如：

```
>>> sample_fun2 = 'abcdabfabbcd'
```

```
>>> print(sample_fun2.count('ab',2,9))      #统计字符串出现的次数
```

运行结果如下：

```
2
```

3. str.capitalize()

将字符串的首字母大写。例如：

```
>>> sample_fun3 = 'abc'
```

```
>>> print(sample_fun3.capitalize())      #首字母大写
```

运行结果如下：

```
Abc
```

4. str.replace(oldstr, newstr, count)

用旧的子字符串替换新的子字符串，若不指定count默认全部替换。例如：

```
>>> sample_fun4 = 'ab12cd3412cd'
```

```
>>> print(sample_fun4.replace('12','21'))      #不指定替换次数count
```

```
>>> print(sample_fun4.replace('12','21',1))     #指定替换次数count
```

运行结果如下：

```
ab21cd3421cd
```

```
ab21cd3412cd
```

5. str.find('str',start,end)

查找并返回子字符在start到end范围内的顺序，默认范围是从父字符串的头开始到尾结束，例如：

```
>>> sample_fun5 = '0123156'
```

```
>>> print(sample_fun5.find('5'))    #查看子字符串的顺序
```

```
>>> print(sample_fun5.find('5',1,4)) #指定范围内没有该字符串默认  
返回-1
```

```
>>> print(sample_fun5.find('1'))    #多个字符串返回第一次出现时候  
的顺序
```

运行结果如下：

```
5
```

```
-1
```

```
1
```

6. `str.index('str',start,end)`

该函数与`find`函数一样，但是如果在某一个范围内没有找到该字符串的时候，不再返回-1而是直接报错。例如：

```
>>> sample_fun6 = '0123156'
```

```
>>> print(sample_fun6.index(7))  #指定范围内没有找到该字符串会  
报错
```

运行结果如下：

Traceback (most recent call last):

File "D:/python/space/demo05-02-03.py", line 2, in <module>

```
    print(sample_fun6.index(7))  #指定范围内没有找到该字符串会报  
错
```

TypeError: must be str, not int

7. str.isalnum()

字符串是由字母或数字组成则返回true否则返回false。例如：

```
>>> sample_fun7 = 'abc123'    #字符串由字母和数字组成
>>> sample_fun8 = 'abc'       #字符串由字母组成
>>> sample_fun9 = '123'       #字符串由数字组成
>>> sample_fun10 = 'abc12%'   #字符串由除了数字字母以为的字符组成
```

```
print(sample_fun7.isalnum())
print(sample_fun8.isalnum())
print(sample_fun9.isalnum())
print(sample_fun10.isalnum())
```

运行结果如下：

```
True
True
True
False
```

8. str.isalpha()

字符串是否全是由字母组成的，是返回true，否则返回false。例如

```
>>> sample_fun11 = 'abc123'           #字符串中不只是有字母
```

```
>>> sample_fun12 = 'abc'             #字符串中只是有字母
```

```
print(sample_fun11.isalpha())
```

```
print(sample_fun12.isalpha())
```

运行结果如下：

False

True

9. str.isdigit()

字符串是否全是由数字组成，是则返回true，否则返回false。例如：

```
>>> sample_fun13 = 'abc12'           #字符串中不只是有数字
>>> sample_fun14 = '12'              #字符串中只是有数字
```

```
print(sample_fun13.isdigit())
```

```
print(sample_fun14.isdigit())
```

运行结果如下：

False

True

10. str.isspace()

字符串是否全是由空格组成的，是则返回true，否则返回false。例如：

```
>>> sample_fun15 = ' abc'           #字符串中不只有空格
```

```
>>> sample_fun16 = ' '             #字符串中只有空格
```

```
>>> print(sample_fun15.isspace())
```

```
>>> print(sample_fun16.isspace())
```

运行结果如下：

False

True

11. str.islower()

字符串是否全是小写，是则返回true，否则返回false。例如：

```
>>> sample_fun17 = 'abc'           #字符串中的字母全是小写
>>> sample_fun18 = 'Abcd'         #字符串中的字母不只有小写
>>> print(sample_fun17.islower())
>>> print(sample_fun18.islower())
```

运行结果如下：

True

False

12. str.isupper()

字符串是否全是大写，是则返回true，否则返回false。例如：

```
>>> sample_fun19 = 'abCa'          #字符串中的字母不全是大写字母
>>> sample_fun20 = 'ABCA'         #字符串中的字母全是大写字母
>>> print(sample_fun19.isupper())
>>> print(sample_fun20.isupper())
```

运行结果如下：

False

True

13. str.istitle()

字符串首字母是否是大写，是则返回true，否则返回false。例如：

```
>>> sample_fun21 = 'Abc'           #字符串首字母大写
>>> sample_fun22 = 'aAbc'         #字符串首字母不是大写
>>> print(sample_fun21s.istitle())
>>> print(sample_fun22.istitle())
```

运行结果如下：

True

False

14. str.lower()

将字符串中的字母全部转换成小写字母。例如：

```
>>> sample_fun23 = 'aAbB'      #将字符串中的字母全部转为小写字母
```

```
>>> print(sample_fun23.lower())
```

运行结果如下：

```
aabb
```

15. str.upper()

将字符串中的字母全部转换成大写字母。例如：

```
>>> sample_fun24 = 'abcD'      #将字符串中的字母全部转为
```

```
>>> print(sample_fun24.upper())
```

运行结果如下：

```
ABCD
```

16. str.split(sep,maxsplit)

将字符串按照指定的sep字符进行分割，maxsplit是指定需要分割的次数，若不指定sep默认是分割空格。例如：

```
>>> sample_fun25 = 'abacdaef'
```

```
>>> print(sample_fun25.split('a'))
```

#指定分割字符串

```
>>> print(sample_fun25.split())
```

#不指定分割字符串

```
>>> print(sample_fun25.split('a',1))
```

#指定分割次数

运行结果如下：

```
['', 'b', 'cd', 'ef']
```

```
['abacdaef']
```

```
['', 'bacdaef']
```

17. `str.startswith(sub[,start[,end]])`

判断字符串在指定范围内是否以sub开头，默认范围是整个字符串。例如：

```
>>> sample_fun26 = '12abcdef'
```

```
>>> print(sample_fun26.startswith('12',0,5)) #范围内是否是以该字符开头
```

运行结果如下：

```
True
```

18. `str.endswith(sub[,start[,end]])`

判断字符串在指定范围内是否是以sub结尾，默认范围是整个字符串。例如：

```
>>> sample_fun27 = 'abcdef12'
```

```
>>> print(sample_fun27.endswith('12')) #指定范围内是否是以该字符结尾
```

运行结果如下：

```
True
```

19. str.partition(sep)

将字符串从sep第一次出现的位置开始分隔成三部分：sep顺序前、sep、sep顺序后。最后会返回出一个三元数组，如果没有找到sep的时候，返回字符本身和两个空格组成的三元数组。例如：

```
>>> sample_fun28 = '123456'
```

```
>>> print(sample_fun28.partition('34')) #指定字符分割，能够找到该字符
```

```
>>> print(sample_fun28.partition('78')) #指定字符分割，不能够找到该字符
```

运行结果如下：

```
('12', '34', '56')
```

```
('123456', '', '')
```

20. str.rpartition(sep)

该函数与partition(sep)函数一致，但是sep不再是第一次出现的顺序，而是最后一次出现的顺序。例如：

```
>>> sample_fun29 = '12345634'
```

```
>>> print(sample_fun29.rpartition('34')) #指定字符最后一次的位置进行分割
```

运行结果如下：

```
('123456', '34', '')
```


字符串的格式化通常有两种方式，

1、字符串格式化表达式

2、函数形式进行格式化

字符串格式化表达式

常用%进行表示，其中%前面是需要**格式控制符**，而%后面就是需要填充的实际参数，这个实际参数其本质就是元组。%也可以理解为占位符。例如：

```
print('My name is %s, and I am %d'%(Jack, 9)) #表达式格式化
```

运行结果如下：

My name is Jack, and I am 9

注意：如果想要将后面填充的浮点数保留两位小数，可以用%f2表示，同时会对第三位小数进行四舍五入。例如：

```
print('你花了%.2f元钱'%(20.45978)) #浮点数保留两个小数
```

运行结果如下：

你花了20.46元钱

字符串常见的格式化符号如表5. 2

格式控制符	说明
%s	字符串（采用str()的显示）或其他任何对象
%r	与%s相似（采用repr()的显示）
%c	单个字符
%b	参数转换成二进制整数
%d	参数转换成十进制整数
%i	参数转换成十进制整数
%o	参数转换成八进制整数
%u	参数转换成十进制整数
%x	参数转换成十六进制整数，字母小写
%X	参数转换成十六进制整数，字母大写
%e. E	按科学计数法格式转换成浮点数
%f. F	按定点小数格式转换成浮点数
%g. G	按定点小数格式转换成浮点数，与%f. F不同

表5.2 Python格式控制符号

格式化是对字符串进行格式表达的方式

- 字符串格式化使用.format()方法，用法如下：

<模板字符串>.format(<逗号分隔的参数>)

字符串格式化方法

```
print('My name is %s, and I am %d'%( 'Jack', 9)) #表达式格式化
```

运行结果如下：

My name is Jack, and I am 9

想要进行字符串格式化使用函数形式**format()**方法。例如：

```
print('My name is {0}, and I am {1} '.format('Jack', 9)) #函数格式化
```

运行结果如下：

My name is Jack, and I am 9

字符串类型的格式化

槽

"{ } : 计算机{ } 的CPU占用率为{ } %".format("2018-10-10", "C", 10)

↑ ↑ ↑
0 1 2

字符串中槽{}的默认顺序

↑ ↑ ↑
0 1 2

format()中参数的顺序

"{1} : 计算机{0} 的CPU占用率为{2} %".format("2018-10-10", "C", 10)

槽内部对格式化的配置方式

{ <参数序号> : <格式控制标记> }

:	<填充>	<对齐>	<宽度>	<,>	<.精度>	<类型>
引导 符号	用于填充的 单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽设定的输 出宽度	数字的千位 分隔符	浮点数小数 精度 或 字 符串最大输 出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %
引导 符号	用于填充的 单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽设定的输 出宽度	<pre> >>> "{0:=^20}".format("PYTHON") ' =====PYTHON===== ' >>> "{0:*>20}".format("BIT") ' *****BIT' >>> "{:10}".format("BIT") ' BIT </pre>		

:	<填充>	<对齐>	<宽度>	<,>	<.精度>	<类型>
<pre>>>> "{0:,.2f}".format(12345.6789) '12,345.68'</pre>				数字的千位 分隔符	浮点数小数 精度或字 符串最大输 出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %
<pre>>>> "{0:b},{0:c},{0:d},{0:o},{0:x},{0:X}".format(425) '110101001,Σ,425,651,1a9,1A9'</pre>						
<pre>>>> "{0:e},{0:E},{0:f},{0:%}".format(3.14) '3.140000e+00,3.140000E+00,3.140000,314.000000%'</pre>						

第五章 字符串与正则表达式

5.1 字符串基础

5.2 字符串方法

5.3 正则表达式

5.4 实验

5.5 小结

习题

5.3.1 认识正则表达式

正则表达式 (Regular Expression)，此处的“Regular”即是“规则”、“规律”的意思，Regular Expression即“描述某种规则的表达式”，因此它又可称为正规表示式、正规表示法、正规表达式、规则表达式、常规表示法等，在代码中常常被简写为regex、regexp或RE。正则表达式使用某些单个字符串，来描述或匹配某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索或替换那些符合某个模式的文本，如下面的表5.3、5.4、5.5、5.6所示。

字符	说明
.	匹配任意1个字符 (除了\n)
[]	匹配[]中列举的字符\d
\d	匹配数字，即0-9
\D	匹配非数字，即不是数字
\s	匹配空白，即空格，Tab键
\S	匹配非空白
\w	匹配单词字符，即a-z. A-Z. 0-9. _
\W	匹配非单词字符

表5.3单个字符匹配

5.3.1 认识正则表达式

表5.4表示数量的匹配

字符	说明
*	匹配前一个字符出现0次或者无限次，即可有可无
+	匹配前一个字符出现1次或者无限次，即至少有1次
?	匹配前一个字符出现1次或者0次，即要么有1次，要么没有
{m}	匹配前一个字符出现m次
{m,}	匹配前一个字符至少出现m次
{m,n}	匹配前一个字符出现从m到n次

表5.5 表示边界的匹配

字符	说明
^	匹配字符串开头
\$	匹配字符串结尾
\b	匹配一个单词的边界
\B	匹配非单词边界

5.3.1 认识正则表达式

字符	说明
	匹配左右任意一个表达式
(ab)	将括号中字符作为一个分组
\num	引用分组num匹配到的字符串
(?P<name>)	分组起别名
(?P=name)	引用别名为name分组匹配到的字符串

表5.6 匹配分组

5.3.2re模块

在Python中需要通过正则表达式对字符串进行匹配的时候，可以导入一个库（模块），名字为re，它提供了对正则表达式操作所需的方法，如表5.7。

方法	说明
<code>re.match(pattern,string flags)</code>	从字符串的开始匹配一个匹配对象，例如匹配第一个单词
<code>re.search(pattern,string flags)</code>	在字符串中查找匹配的对象,找到第一个后就返回,如果没有找到就返回None
<code>re.sub(pattern,repl,string count)</code>	替换掉字符中的匹配项
<code>re.split(r',',text)</code>	分割字符
<code>re.findall(pattern,string flags)</code>	获取字符串中所有匹配的对象
<code>re.compile(pattern,flags)</code>	创建模式对象

表5.7 re模块常见的方法

5.3.3 re.match()方法

re.match()是用来进行正则匹配检查的方法，若字符串匹配正则表达式，则match()方法返回匹配对象（Match Object），否则返回None（注意不是空字符串""）。

匹配对象Match Object具有group()方法，用来返回字符串的匹配部分。常用格式为：

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

这里的pattern格式为('正则表达式', '匹配的字符串')例如：

```
>>> import re                                #导入re包
>>> sample_result1 = re.match('Python','Python12') #从头查找匹配字符串
>>> print(sample_result1.group())             #输出匹配的字符串
```

运行结果如下：

```
Python
```

5.3.4 re.search()方法

re.search()方法和re.match()方法相似，也是用来对正则匹配检查的方法但不同的是search()方法是在字符串的头开始一直到尾进行查找，若正则表达式与字符串匹配成功，那么就返回匹配对象，否则返回None。例如：

```
>>> import re
```

```
>>> sample_result2 = re.search('Python','354Python12') #依次匹配字符串
```

```
>>> print(sample_result2.group())
```

运行结果如下：

```
Python
```

5.3.4 re.search()方法与re.match()方法的区别

虽然re.match()和re.search()方法都是指定的正则表达式与字符串进行匹配，但是 re.match()是从字符串的开始位置进行匹配，若匹配成功，则返回匹配对象，否则返回None。而re.search()方法却是从字符串的全局进行扫描，若匹配成功就返回匹配对象，否则返回None。例如：

```
>>> import re
>>> sample_result3 = re.match('abc','abcdef1234') #match只能够匹配头
>>> sample_result4 = re.match('1234','abcdef1234')
>>> print(sample_result3.group())
>>> print(sample_result4)
```

5.3.4 re.search()方法与re.match()方法的区别

```
>>> sample_result5 = re.search('abc','abcdef1234') #search匹配全体字符
>>> sample_result6 = re.search('1234','abcdef1234')
>>> print(sample_result5.group())
>>> print(sample_result6.group())
```

运行结果如下：

abc

None

abc

1234

第五章 字符串与正则表达式

5.1 字符串基础

5.2 字符串方法

5.3 正则表达式

5.4 实验

5.5 小结

习题

5.4.1 使用字符串处理函数

1. 我们常看到自己电脑上的文件路径如' C:\Windows\Logs\dosvc', 请将该路径分割为不同的文件夹。

```
>>> sample_str1 = 'C:\Windows\Logs\dosvc'
>>> sample_slipstr = sample_str1.split('\\') #\转义字符要转一次才是本意
>>> print(sample_slipstr)
```

运行结果如下：

```
['C:', 'Windows', 'Logs', 'dosvc']
```

2. Python的官网是<https://www.python.org>判断该网址是否是以org结尾。

```
>>> sample_str2 = 'https://www.python.org'
>>> print(sample_str2.endswith('org'))      #从字符串末尾开始查找
```

运行结果如下：

```
True
```

5.4.2 正则表达式的使用

写出一个正则表达式来匹配是否是手机号。

```
>>> import re                                #定义一个正则表达式
>>> phone_rule = re.compile('1\d{10}')
>>> phone_num = input('请输入一个手机号')    #通过规则去匹配字符串
>>> sample_result3 = phone_rule.search(phone_num)
>>> if sample_result3 != None:
print('这是一个手机号')
else:
```

```
print('这不是一个手机号')
```

运行结果如下：

请输入一个手机号12312345678

这是一个手机号

Process finished with exit code 0

请输入一个手机号24781131451

这不是一个手机号

Process finished with exit code 0

5.4.3 使用re模块

用两种方式写出一个正则表达式匹配字符'Python123'中的'Python'并输出字符串'Python'。

```
>>> import re                #导入re包
>>> sample_regu = re.compile('Python') #定义正则表达式规则
>>> sample_result4 = sample_regu.match('Python123') #用match方式匹配字符串
>>> print(sample_result4.group())      #用search方式匹配字符串
>>> sample_result5 = sample_regu.search('Python123')
>>> print(sample_result5.group())
```

第五章 字符串与正则表达式

5.1 字符串基础

5.2 字符串方法

5.3 正则表达式

5.4 实验

5.5 小结

习题

本章首先讲解了Python字符串概念，字符串的基本操作；其次是字符串的格式化，主要的格式化符号、格式化元组；还有操作字符串的基本方法，这些符号和方法在Python的开发中会被经常使用到。之后，我们学习了正则表达式，re模块和正则表达式的基本表示符号，这些符号可以帮助简化正则表达式。

正则表达式的用途非常广泛，几乎任何编程语言都可以使用到它，所以学好正则表达式，对于提高自己的编程能力有非常重要的作用。

第五章 字符串与正则表达式

5.1 字符串基础

5.2 字符串方法

5.3 正则表达式

5.4 实验

5.5 小结

习题

习题:

1. 将字符串'abcdefg' 使用函数的方式进行倒叙输出。
2. 在我们生活中节假日的问候是必不可少的，请使用字符串格式化的方式写一个新年问候语模板。
3. 写出能够匹配163邮箱（@163.com）的正则表达式。
4. 简述re模块中re.match()与re.search()的区别。

感谢聆听

