

上周作业

- 1. 编写两个函数。第一个函数读入5个数字作为参数，然后将数字从小到大排序后显示在屏幕上，不用编写调用过程，只需要定义函数即可；第二个函数读入一个list作为参数，然后将该list从小到大排序。函数调用结束后，需要编写调用过程，打印这个list在屏幕上应为排序后的效果（因此本题最后应该代码中定义两个函数，然后有一次调用）。

```
def sort_num_5(a,b,c,d,e):  
    #condition 1  
    if a > b: tmp = a; a = b; b = tmp  
    if b > c: tmp = b; b = c; c = tmp  
    if c > d: tmp = c; c = d; d = tmp  
    if d > e: tmp = d; d = e; e = tmp  
    #condition 2  
    if a > b: tmp = a; a = b; b = tmp  
    if b > c: tmp = b; b = c; c = tmp  
    if c > d: tmp = c; c = d; d = tmp  
    #condition 3  
    if a > b: tmp = a; a = b; b = tmp  
    if b > c: tmp = b; b = c; c = tmp  
    #condition 4  
    if a > b: tmp = a; a = b; b = tmp  
    print(a,b,c,d,e)
```

```
def rankUp(a,b,c,d,e):  
    g = [a,b,c,d,e]  
    for i in range(len(g) - 1):  
        m = i  
        for j in range(i + 1,len(g)):  
            if g[j] < g[m]:  
                m = j  
        g[i],g[m] = g[m],g[i]  
    print(g)  
    return(g[0],g[1],g[2],g[3],g[4])
```

```
def sort_list(list_):  
    res = sorted(list_)  
    return res  
  
print(sort_list([19,7,3,12,1]))
```

sort/sorted: 怎么理牌?



- **示例5-19** 编写函数，实现排序

```
from random import randint
```

```
def bubbleSort(lst, reverse=False):  
    length = len(lst)  
    for i in range(0, length):  
        flag = False  
        for j in range(0, length-i-1):  
            #比较相邻两个元素大小，并根据需要进行交换，默认升序排序  
            #这里用字符串记录公式不是必须，只是某种个性化编程风格  
            exp = 'lst[j] > lst[j+1]'  
            #如果reverse=True则降序排序  
            if reverse:  
                exp = 'lst[j] < lst[j+1]'  
            if eval(exp):  
                lst[j], lst[j+1] = lst[j+1], lst[j]  
                #flag=True表示本次扫描发生过元素交换  
                flag = True  
        #如果一次扫描结束后，没有发生过元素交换，说明已经按序排列  
        if not flag:  
            break
```

- **示例5-21** 编写函数，按选择法排序。

```
def selectSort(lst, reverse=False):  
    length = len(lst)  
    for i in range(0, length):  
        #假设剩余元素中第一个最小或最大  
        m = i  
        #扫描剩余元素  
        for j in range(i+1, length):  
            #如果有更小或更大的，就记录下它的位置  
            exp = 'lst[j] < lst[m]'  
            if reverse:  
                exp = 'lst[j] > lst[m]'  
            if eval(exp):  
                m = j  
        #如果发现更小或更大的，就交换值  
        if m != i:  
            lst[i], lst[m] = lst[m], lst[i]
```

- **示例5-23** 编写函数，实现快速排序算法。

```
from random import randint
```

```
def quickSort(lst, reverse=False):  
    if len(lst) <= 1:  
        return lst  
    #默认使用最后一个元素作为枢点  
    pivot = lst.pop()  
    first, second = [], []  
    #默认使用升序排序  
    exp = 'x<=pivot'  
    #reverse=True表示降序排列  
    if reverse == True:  
        exp = 'x>=pivot'  
    for x in lst:  
        first.append(x) if eval(exp) else second.append(x)  
    #递归调用  
    return quickSort(first, reverse) + [pivot] + quickSort(second, reverse)
```

第5部分 字符串和文本处理

字符串

- 在Python中，字符串属于不可变有序序列，使用单引号、双引号、三单引号或三双引号作为定界符，并且不同的定界符之间可以互相嵌套。

'abc'、'123'、'中国'

"Python"

'''Tom said, "Let's go"'''



问1: 程序里，两个单引号会和一个双引号混淆么？
问2: 在什么情况下会需要三引号方式？

字符串

- 除了支持比较大小、计算长度、元素访问、切片、成员测试等操作外，字符串类型还支持一些特有的操作方法，例如字符串格式化、查找、替换、排版等等。
- **但**字符串属于**不可变**序列，**不能**直接对字符串对象进行元素增加、修改与删除等操作，切片操作也只能访问其中的元素而无法使用切片来修改字符串中的字符。



问3：为什么字符串不可变，不可直接修改？

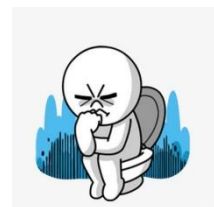
字符串

- 除了支持str类型之外，Python还支持字节串类型bytes
 - str类型字符串可以通过`encode()`方法使用指定的字符串编码格式编码成为bytes对象
 - bytes对象则可以通过`decode()`方法使用指定编码格式解码成为str字符串
- 补充知识
 - 最早字符串编码是美国标准信息交换码ASCII，仅对10个数字、26个大写英文字母、26个小写英文字母及一些其他符号进行了编码。ASCII码采用1个字节来对字符进行编码，也就是最多只能表示256个符号
 - UTF-8对全世界所有国家需要用到的字符进行了编码，以1个字节表示英语字符(兼容ASCII)，以3个字节表示中文，还有些语言的符号使用2个字节（例如俄语和希腊语符号）或4个字节。
 - GB2312是我国制定的中文编码，使用1个字节表示英语，2个字节表示中文；GBK是GB2312的扩充，而CP936是微软在GBK基础上开发的编码方式。GB2312、GBK和CP936都是使用2个字节表示中文（问4：可以表示多少个汉字？）
 -

字符串编码格式简介

- 乱码/报错的由来
 - 不同编码格式之间相差很大，采用不同的编码格式意味着不同的表示和存储形式，把同一字符存入文件时，写入的内容可能会不同，在试图理解其内容时必须了解编码规则并进行正确的解码。如果解码方法不正确就无法还原信息，并报错。

```
>>> '张诚'.encode('utf8')
b'\xe5\xbc\xa0\xe8\xaf\x9a'
>>> '张诚'.encode('cp936')
b'\xd5\xc5\xb3\xcf'
>>> '张诚'.encode('cp936').decode('cp936')
'张诚'
>>> '跟张老师学Python'.encode('utf8').decode('cp936')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    '跟张老师学Python'.encode('utf8').decode('cp936')
UnicodeDecodeError: 'gbk' codec can't decode byte 0xa6 in position 14: illegal multibyte sequence
>>> 'Python程序设计开发宝典'.encode('cp936').decode('utf8')
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    'Python程序设计开发宝典'.encode('cp936').decode('utf8')
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb3 in position 6: invalid start byte
```



问5：如何改对？

字符串编码格式

- Python 3.x完全支持中文字符，默认使用UTF8编码格式，无论是一个数字、英文字母，还是一个汉字，在统计字符串长度时都按一个字符对待和处理。

```
>>> s = '中国山东烟台'
>>> len(s)                                #字符串长度，或者包含的字符个数
6
>>> s = '中国山东烟台ABCDE'             #中文与英文字符同样对待，都算一个字符
>>> len(s)
11
>>> 姓名 = '张三'                         #使用中文作为变量名
>>> print(姓名)                           #输出变量的值
张三
```

再次提醒：在idle界面直接输出显示的语句，在程序里要加上print()

转义字符与原始字符串

转义字符	含义	转义字符	含义
\b	退格，把光标移动到前一列位置	\\	一个斜线\
\f	换页符	\'	单引号'
\n	换行符	\"	双引号"
\r	回车	\ooo	3位八进制数对应的字符
\t	水平制表符	\xhh	2位十六进制数对应的字符
\v	垂直制表符	\uhhhh	4位十六进制数表示的Unicode字符



问6：在什么情况下会需要用到转义？

转义字符与原始字符串

❖转义字符用法

```
>>> print('Hello\nWorld')           #包含转义字符的字符串
Hello
World

>>> print('\101')                   #三位八进制数对应的字符
A

>>> print('\x41')                   #两位十六进制数对应的字符
A

>>> '张诚'.encode('unicode_escape')
b'\\u5f20\\u8bda'

>>> print('我是\u5f20\u8bda')#四位十六进制数表示Unicode字符
我是张诚
```

小知识：Unicode希望能够解决多语言的计算，如不同国家的字符标准，它为每一个符号定义一个数字和名称，并指定字符和它的数值（码位），以及该值的二进制位表示法，通过一个十六进制数字和前缀（U）定义一个16位的数值，如：U+0041 表示 A。它用两个字节表示每个字符的字符编码，可以表示0~65,535的符号¹³

转义字符与原始字符串

- 为了避免对字符串中的转义字符进行转义，可以使用原始字符串，在字符串前面加上字母r或R表示原始字符串，其中的所有字符都表示原始的含义而不会进行任何转义。

```
>>> path = 'C:\Windows\notepad.exe'
```

```
>>> print(path)
```

#字符\n被转义为换行符

```
C:\Windows
```

```
otepad.exe
```

```
>>> path = r'C:\Windows\notepad.exe' #原始字符串，任何字符都不转义
```

```
>>> print(path)
```

```
C:\Windows\notepad.exe
```

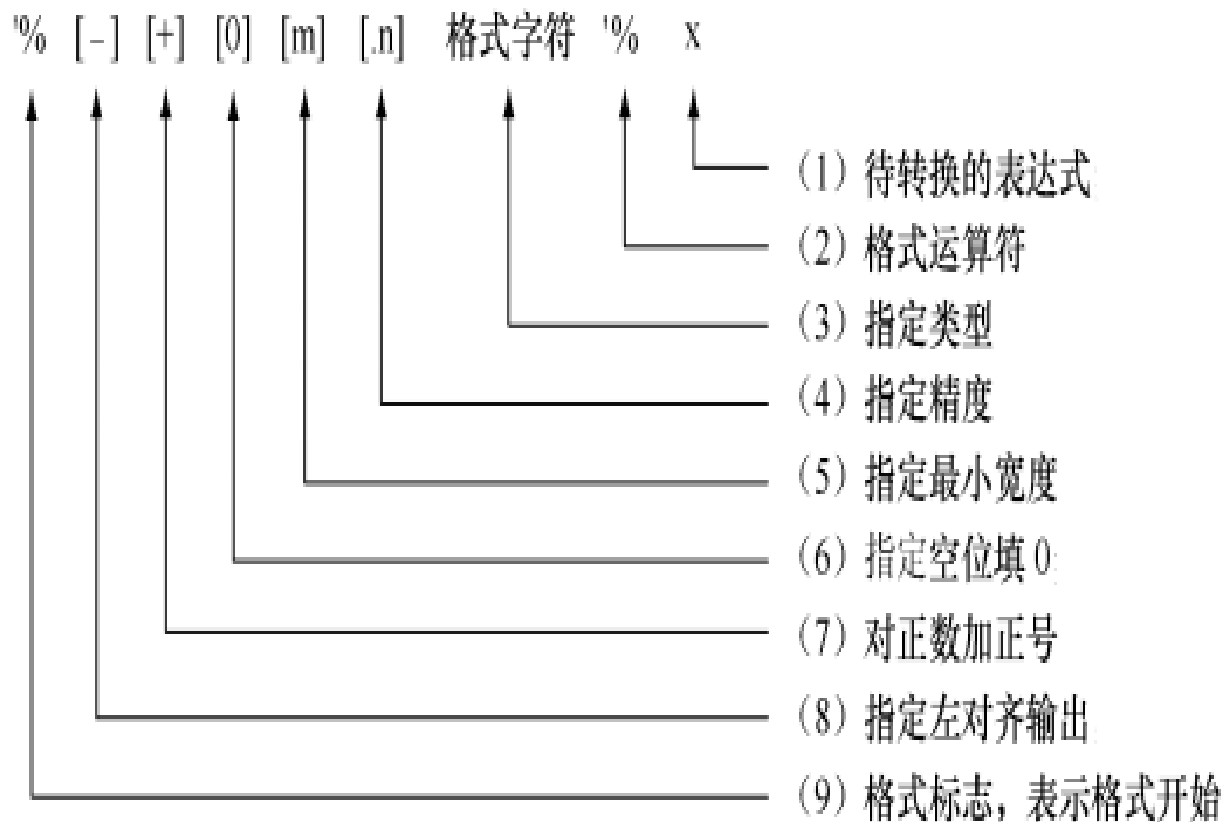
字符串操作： 格式化（输出）

- 1 使用%运算符进行格式化
- 2 使用format方法进行格式化
- 3 格式化的字符串常量



问6： 什么是格式化？

1 使用%运算符进行格式化



格式字符	说明
%s	字符串 (采用str()的显示)
%r	字符串 (采用repr()的显示)
%c	单个字符
%d	十进制整数
%i	十进制整数
%o	八进制整数
%x	十六进制整数
%e	指数 (基底写为e)
%E	指数 (基底写为E)
%f、%F	浮点数
%g	指数(e)或浮点数 (根据显示长度)
%G	指数(E)或浮点数 (根据显示长度)
%%	一个字符“%”

1 使用%运算符进行格式化(在py里面，一定要加print())

```
x = 1235
so = "%o" % x      #对变量来说，%之后有无1个空格都行
print(so)
"2323"
```

```
sh = "%x" % x
print(sh)
"4d3"
```

```
>>> se = "%e" % x
>>> se
"1.235000e+03"
>>> chr(ord("3")+1)
"4"
```

```
>>> "%s" % 65
"65"
>>> "%s" % 65333
"65333"
```

```
>>> "%d" % "555"
TypeError: %d format: a number is required, not str
```

其实也就是说：使用%的话，就可以在print()里面显示输出的样子

2 使用format()方法进行格式化

```
>>> 1/3
```

```
0.3333333333333333
```

```
>>> print('{0:.3f}'.format(1/3))
```

#保留3位小数

```
0.333
```

```
>>> '{0:%}'.format(3.5)
```

#格式化为百分数

```
'350.000000%'
```

```
>>> '{0:_},{0:_x}'.format(1000000)
```

#可以对1个变量同时进行不同方式的显示

```
'1_000_000,f_4240'
```

```
>>> '{0:_},{0:_x},{1:_}'.format(1000000,10) #通过位置顺序编号匹配变量的显示方式
```

```
'1_000_000,f_4240,10'
```

2 使用format()方法进行格式化

```
>>> print("The number {0:.,} in hex is: {0:#x}, the number {1} in oct is  
{1:#o}".format(5555,55))
```

The number 5,555 in hex is: 0x15b3, the number 55 in oct is 0o67

对比 `print("The number %d in hex is: %x, the number %d in oct is %o" %(5555,5555,55,55))`

```
>>> print("The number {1:.,} in hex is: {1:#x}, the number {0} in oct is  
{0:o}".format(5555,55))
```

The number 55 in hex is: 0x37, the number 5555 in oct is 12663

```
>>> print("my name is {name}, my age is {age}, and my QQ is {qq}".format(name  
= "Zhang Cheng",age = 40,qq = "30646****"))
```

my name is Zhang Cheng, my age is 40, and my QQ is 30646****

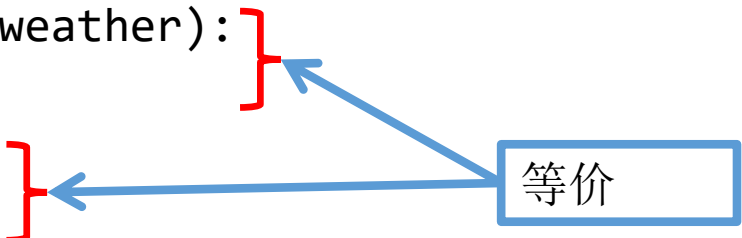
```
>>> position = (5, 8, 13)
```

```
>>> print("X:{0[0]};Y:{0[1]};Z:{0[2]}".format(position))
```

X:5;Y:8;Z:13

2 使用format()方法进行格式化

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



运行结果:

```
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'
```

通过位置顺序编号匹配变量的显示方式，对于列表、元组等结构更得心应手

3 格式化的字符串常量

- 从Python 3.6.x开始支持一种新的字符串格式化方式，官方叫做Formatted String Literals，在字符串前加字母f，含义与字符串对象format()方法类似。

```
>>> name = 'Zhang'
>>> age = 39
>>> f'My name is {name}, and I am {age} years old.'
'My name is Zhang, and I am 39 years old.'
>>> width = 10
>>> precision = 4
>>> value = 11/3
>>> f'result:{value:{width}.{precision}}'
'result:      3.667'
```

等价于

```
f'result:{11/3:{10}.{4}}'
```

4 字符串常用操作

- Python字符串对象提供了大量方法用于字符串的切分、连接、替换和排版等操作，另外还有大量内置函数和运算符也支持对字符串的操作。
- 字符串对象是不可变的（重要的话说到第三遍了），所以字符串对象提供的涉及到字符串“修改”的方法都是返回修改后的新字符串，并不对原始字符串做任何修改，无一例外。

4.1 find()、 rfind()、 index()、 rindex()、 count()

- find()、 rfind()、 index()、 rindex()、 count()
- ✓ find() 和 rfind 方法分别用来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）中首次和最后一次出现的位置，如果不存在则返回-1；
- ✓ index() 和 rindex() 方法用来返回一个字符串在另一个字符串指定范围中首次和最后一次出现的位置，如果不存在则抛出异常；
- ✓ count() 方法用来返回一个字符串在当前字符串中出现的次数。

4.1 find()、rfind()、index()、rindex()、count()

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

```
>>> s.index('pear')
25
>>> s.index('ppp')
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
```


4.2 split()、rsplit()、partition()、rpartition()

- `split()`、`rsplit()`、`partition()`、`rpartition()`
- ✓ `split()` 和 `rsplit()` 方法分别用来以指定字符为分隔符，把当前字符串从左往右或从右往左分隔成多个字符串，并返回包含分隔结果的列表；
- ✓ `partition()` 和 `rpartition()` 用来以指定字符串为分隔符将原字符串分隔为3部分，即分隔符前的字符串、分隔符字符串（第一个遇到的）、分隔符后的字符串，如果指定的分隔符不在原字符串中，则返回原字符串和两个空字符串。

4.2 split()、rsplit()、partition()、rpartition()

```
>>> s = "apple,peach,banana,pear"
>>> s.split(",")
["apple", "peach", "banana", "pear"]
>>> s.partition(',')
('apple', ', ', 'peach,banana,pear')
>>> s.rpartition(',')
('apple,peach,banana', ', ', 'pear')
>>> s.rpartition('banana')
('apple,peach,', 'banana', ',pear')
>>> s = "2017-10-31"
>>> t = s.split("-")
>>> print(t)
['2017', '10', '31']
>>> print(list(map(int, t)))
[2017, 10, 31]
```

分隔符

对每个字符强制转为整数，map第1个参数是个函数，第二个参数是个列表

4.2 split()、rsplit()、partition()、rpartition()

- split() 和 rsplit() 方法还允许指定最大分割次数。

```
>>> s = '\n\nhello\t\t world \n\n\n My name is Zhang    '\n>>> s.split(None, 1)\n['hello', 'world \n\n\n My name is Zhang    ']\n>>> s.rsplit(None, 2)\n['\n\nhello\t\t world \n\n\n My name', 'is', 'Zhang']\n>>> s.split(maxsplit=6)\n['hello', 'world', 'My', 'name', 'is', 'Zhang']\n>>> s.split(maxsplit=100)      #最大分隔次数大于可分隔次数时无效\n['hello', 'world', 'My', 'name', 'is', 'Zhang']
```

4.2 split()、rsplit()、partition()、rpartition()

- 对于split()和rsplit()方法，如果不指定分隔符，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，把连续多个空白字符看作一个分隔符。

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

4.2 split()、rsplit()、partition()、rpartition()

◆然而，明确传递参数指定split()使用的分隔符时（又遇到多个这些分隔符），情况是不一样的：第一个分隔符后的当成1个为空的分隔符

```
>>> 'a,,,bb,,ccc'.split(',')
['a', '', '', 'bb', '', 'ccc']
>>> 'a\t\t\tbb\t\tccc'.split('\t')
['a', '', '', 'bb', '', 'ccc']
>>> 'a\t\t\tbb\t\tccc'.split()
['a', 'bb', 'ccc']
```

4.3 join()

■ 字符串连接join()

连接符

```
>>> li = ["apple", "peach", "banana", "pear"]
```

```
>>> ','.join(li)
```

```
'apple,peach,banana,pear'
```

```
>>> '.'.join(li)
```

```
'apple.peach.banana.pear'
```

```
>>> '::'.join(li)
```

```
'apple::peach::banana::pear'
```

4.3 join()

- **组合应用：**使用split()和join()方法删除字符串中多余的空白字符，连续多个空白字符只保留一个。

```
>>> x = 'aaa      bb      c d e   fff   '
>>> ' '.join(x.split())           #使用空格作为连接符
'aaa bb c d e fff'
def equavilent(s1, s2):           #判断两个字符串在Python意义上是否等价
    if s1 == s2:
        return True
    elif ' '.join(s1.split()) == ' '.join(s2.split()):
        return True
    elif ''.join(s1.split()) == ''.join(s2.split()):
        return True
    else:
        return False
print(equavilent('pip list', 'pip    list'))
True
```

4.4 lower()、upper()、capitalize()、title()、swapcase()

■ lower()、upper()、capitalize()、title()、swapcase()

```
>>> s = "What is Your Name?"
>>> s.lower()                #返回小写字符串
'what is your name?'
>>> s.upper()                #返回大写字符串
'WHAT IS YOUR NAME?'
>>> s.capitalize()          #字符串首字符大写
'What is your name?'
>>> s.title()                #每个单词的首字母大写
'What Is Your Name?'
>>> s.swapcase()             #大小写互换
'wHAT IS yOUR nAME?'
```


4.5 replace()、maketrans()、translate()

- 查找替换replace()，类似于Word中的“全部替换”功能。

```
>>> s = "中国，中国"
```

```
>>> print(s)
```

```
中国，中国
```

```
>>> s2 = s.replace("中国", "中华人民共和国") #两个参数都作为一个整理
```

```
>>> print(s2)
```

```
中华人民共和国，中华人民共和国
```

4.5 replace()、maketrans()、translate()

- 问题解决：测试用户输入中是否有敏感词，如果有的话就把敏感词替换为3个星号***。

```
words = ('测试', '非法', '暴力', '话')
text = '这句话里含有非法内容'
for word in words:
    if word in text:
        text = text.replace(word, '***')
text
'这句***里含有***内容'
```

4.5 replace()、maketrans()、translate()

- 字符串对象的`maketrans()`方法用来生成字符映射表，而`translate()`方法用来根据映射表中定义的对应关系转换字符串并替换其中的字符，使用这两个方法的组合可以同时处理多个字符。

这两个参数不是作为整体进行处理的

#创建映射表，将字符"abcdef123"——对应地转换为"uvwxyz@# \$"

```
>>> table = ''.maketrans('abcdef123', 'uvwxyz@#$')
```

```
>>> s = "Python is a greate programming language. I like it!"
```

#按映射表进行替换

```
>>> s.translate(table)
```

```
'Python is u gryuty progrumming lunguugy. I liky it!'
```

4.5 replace()、maketrans()、translate()

- 应用：凯撒加密，每个字母替换为后面第k个。

```
>>> import string
>>> def kaisa(s, k):
    lower = string.ascii_lowercase           #小写字母
    upper = string.ascii_uppercase          #大写字母
    before = string.ascii_letters
    after = lower[k:] + lower[:k] + upper[k:] + upper[:k]
    table = ''.maketrans(before, after)      #创建映射表
    return s.translate(table)

>>> s = "Python is a greate programming language. I like it!"
>>> kaisa(s, 3)
'Sbwkrq lv d juhduh surjudpplqj odqjxdjh. L olnh lw!'
```

之前字符的encode/decode与之类似，只不过映射表是事先定义好的

4.6 strip()、rstrip()、lstrip()

■ strip()、rstrip()、lstrip()

```
>>> s = " abc "
```

```
>>> s.strip()
```

```
'abc'
```

#删除空白字符

```
>>> '\n\nhello world \n\n'.strip()
```

```
'hello world'
```

#删除空白字符

```
>>> "aaaassddf".strip("a")
```

```
'ssddf'
```

#删除指定字符

```
>>> "aaaassddf".strip("af")
```

```
'ssdd'
```

```
>>> "aaaassddfaaa".rstrip("a")
```

```
'aaaassddf'
```

#删除字符串右端指定字符

```
>>> "aaaassddfaaa".lstrip("a")
```

```
'ssddfaaa'
```

#删除字符串左端指定字符

4.6 strip()、rstrip()、lstrip()

- 这三个函数的参数指定的字符串并不作为一个整体对待，而是在原字符串的两侧、右侧、左侧删除参数字符串中包含的所有字符，一层一层地从外往里扒。

```
>>> 'aabbccddeeffg'.strip('af')  #字母f不在字符串两侧，所以不删除
'bbccddeeffg'
>>> 'aabbccddeeffg'.strip('gaf')
'bbccddeee'
>>> 'aabbccddeeffg'.strip('gaef')
'bbccdd'
>>> 'aabbccddeeffg'.strip('gbaef')
'ccdd'
>>> 'aabbccddeeffg'.strip('gbaefcd')
''
```

4.7 startswith()、endswith()

■ `s.startswith(t)`、`s.endswith(t)`，判断字符串是否以指定字符串开始或结束

```
>>> s = 'Beautiful is better than ugly.'
```

```
>>> s.startswith('Be')      #检测整个字符串
```

```
True
```

```
>>> s.startswith('Be', 5)    #指定检测范围起始位置
```

```
False
```

```
>>> s.startswith('Be', 0, 5) #指定检测范围起始和结束位置
```

```
True
```

```
>>> import os
```

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

相当于查找匹配，只不过对于特定问题更方便

4.8 isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()

- isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()，用来测试字符串是否为数字或字母、是否为字母、是否为数字字符、是否为空白字符、是否为大写字母以及是否为小写字母。

```
>>> '1234abcd'.isalnum()
```

```
True
```

```
>>> '1234abcd'.isalpha()
```

#全部为英文字母时返回True

```
False
```

```
>>> '1234abcd'.isdigit()
```

#全部为数字时返回True

```
False
```

```
>>> 'abcd'.isalpha()
```

```
True
```

```
>>> '1234.0'.isdigit()
```

```
False
```


4.8 isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()

```
>>> '1234'.isdigit()
```

```
True
```

```
>>> '九'.isnumeric()
```

#isnumeric()方法支持汉字数字

```
True
```

```
>>> '九'.isdigit()
```

```
False
```

```
>>> '九'.isdecimal()
```

```
False
```

```
>>> 'IVⅢX'.isdecimal()
```

```
False
```

```
>>> 'IVⅢX'.isdigit()
```

```
False
```

```
>>> 'IVⅢX'.isnumeric()
```

#支持罗马数字

```
True
```

4.9 center()、ljust()、rjust()、zfill()

- center()、ljust()、rjust()，返回指定宽度的新字符串，原字符串居中、左对齐或右对齐出现在新字符串中，如果指定宽度大于字符串长度，则使用指定的字符（默认为空格）进行填充。zfill()返回指定宽度的字符串，在左侧以字符0进行填充。

```
>>> 'Hello world!'.center(20)          #居中对齐，以空格进行填充
'    Hello world!    '
>>> 'Hello world!'.center(20, '=')     #居中对齐，以字符=进行填充
'====Hello world!===='
>>> 'Hello world!'.ljust(20, '=')      #左对齐
'Hello world!===== '
>>> 'Hello world!'.rjust(20, '=')     #右对齐
'=====Hello world!'
```

4.10 字符串对象支持的运算符

- Python字符串支持加法运算符，表示两个字符串连接，生成新字符串。

```
>>> 'hello ' + 'world'  
'hello world'
```

4.10 字符串对象支持的运算符

■ 成员判断，关键字in

>>> "a" in "abcde" #测试一个字符中是否存在于另一个字符串中

True

>>> 'ab' in 'abcde'

True

>>> 'ac' in 'abcde' #关键字in左边的字符串作为一个整体对待

False

>>> "j" in "abcde"

False

4.10 字符串对象支持的运算符

- Python字符串支持与**整数**的乘法运算，表示序列重复，也就是**字符串内容的重复**，得到新字符串。

```
>>> 'abcd' * 3  
'abcdabcdabcd'
```

4.11 适用于字符串对象的内置函数

```
>>> x = 'Hello world.'
>>> len(x)                                #字符串长度
12
>>> max(x)                                #最大字符
'w'
>>> min(x)
'.'

>>> list(zip(x,x))                        #zip()也可以作用于字符串
[('H', 'H'), ('e', 'e'), ('l', 'l'), ('l', 'l'), ('o', 'o'), (' ', ' '), ('w', 'w'), ('o', 'o'), ('r', 'r'), ('l', 'l'), ('d', 'd'), ('.', '.')]
>>> sorted(x)
[' ', '.', 'H', 'd', 'e', 'l', 'l', 'l', 'o', 'o', 'r', 'w']
>>> list(reversed(x))
['.', 'd', 'l', 'r', 'o', 'w', ' ', 'o', 'l', 'l', 'e', 'H']
>>> list(enumerate(x))
[(0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'), (5, ' '), (6, 'w'), (7, 'o'), (8, 'r'), (9, 'l'), (10, 'd'), (11, '.')]
>>> list(map(add, x, x))
['HH', 'ee', 'll', 'll', 'oo', ' ', 'ww', 'oo', 'rr', 'll', 'dd', '..']
```

4.11 适用于字符串对象的内置函数

- 内置函数`eval()`用来把任意字符串转化为Python表达式并进行求值。

```
>>> eval("3+4")
```

#计算表达式的值

```
7
```

```
>>> a = 3
```

```
>>> b = 5
```

```
>>> eval('a+b')
```

#这时候要求变量a和b已存在

```
8
```

```
>>> import math
```

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

```
1.7320508075688772
```

4.12 字符串对象的切片操作

- 切片也适用于字符串，但仅限于读取其中的元素，不支持字符串修改。

```
>>> 'Explicit is better than implicit.'[:8]
'Explicit'
>>> 'Explicit is better than implicit.'[9:23]
'is better than'
>>> path = 'C:\\Python35\\test.bmp'
>>> path[:-4] + '_new' + path[-4:]
'C:\\Python35\\test_new.bmp'
```


5 字符串常量

- Python标准库string中定义数字字符、标点符号、英文字母、大写字母、小写字母等常量。

```
>>> import string
>>> string.digits
'0123456789'
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

5 字符串常量

- 问题解决：生成指定长度的随机密码。


```
>>> import string
>>> characters = string.digits + string.ascii_letters #常量组成一组字符
>>> print(characters)
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
>>> import random
>>> ''.join([random.choice(characters) for i in range(8)])
'J5Cuofhy'
>>> ''.join([random.choice(characters) for i in range(10)])
'RkHA3K3tNl'
>>> ''.join([random.choice(characters) for i in range(16)])
'zSabpGltJ0X4CCjh'
```

正则表达式

第8章 正则表达式

- 正则表达式是字符串处理的有力工具，正则表达式使用预定义的模式去匹配一类具有共同特征的字符串，可以快速、准确地完成复杂的查找、替换等处理要求，比字符串自身提供的方法提供了更强大的处理功能。例如
 - 使用字符串对象的`split()`方法只能指定一个分隔符，而使用正则表达式可以很方便地指定多个符号作为分隔符；
 - 使用字符串对象的`split()`并指定分隔符时，很难处理分隔符连续多次出现的情况，而正则表达式让这一切都变成非常轻松。
- 正则表达式在文本编辑与处理、网页爬虫之类的场合中有重要应用。

举例: https://v.taobao.com/




阿里巴巴旗下内容服务平台

[首页](#) [内容推广](#) [官方活动](#) [行业解决方案](#) [直播通](#) [品牌营销](#) [淘榜单](#) [更多](#) [请登录](#)

[主播](#) [机构](#) [直播综艺栏目](#) [商家直播代运营](#) [淘女郎](#)

垂直领域: [不限](#) [美搭](#) [美妆个护](#) [居家](#) [美食](#) [母婴](#) [型男](#) [数码科技](#) [运动](#) [汽车](#) [文化娱乐](#) [萌宠](#)
[园艺](#) [动漫](#) [星座](#) [摄影](#) [游戏](#) [旅游](#) [其他](#)

综合能力 ↓ 粉丝数 ↓ 合作任务数 ↓ 服务评分 ↓ 500个服务方



切糕

粉丝 25.62万

¥1000

合作任务数: 710

服务评分: 5.0

任务完成率: 72%

垂直领域: 美食

所属机构: 新逻辑

淘女郎代播 | 切糕直播

100.00



合作任务数: 791

服务评分: 4.8

任务完成率: 95%

垂直领域: 美搭

所属机构: 远致文化

淘女郎代播 | 主播代播代运营

200.00

热门服务



美妆个护混播
¥ 101 已售出: 0



查看器 控制台 调试器 网络 样式编辑器 性能 内存 存储 无障碍环境

user:1

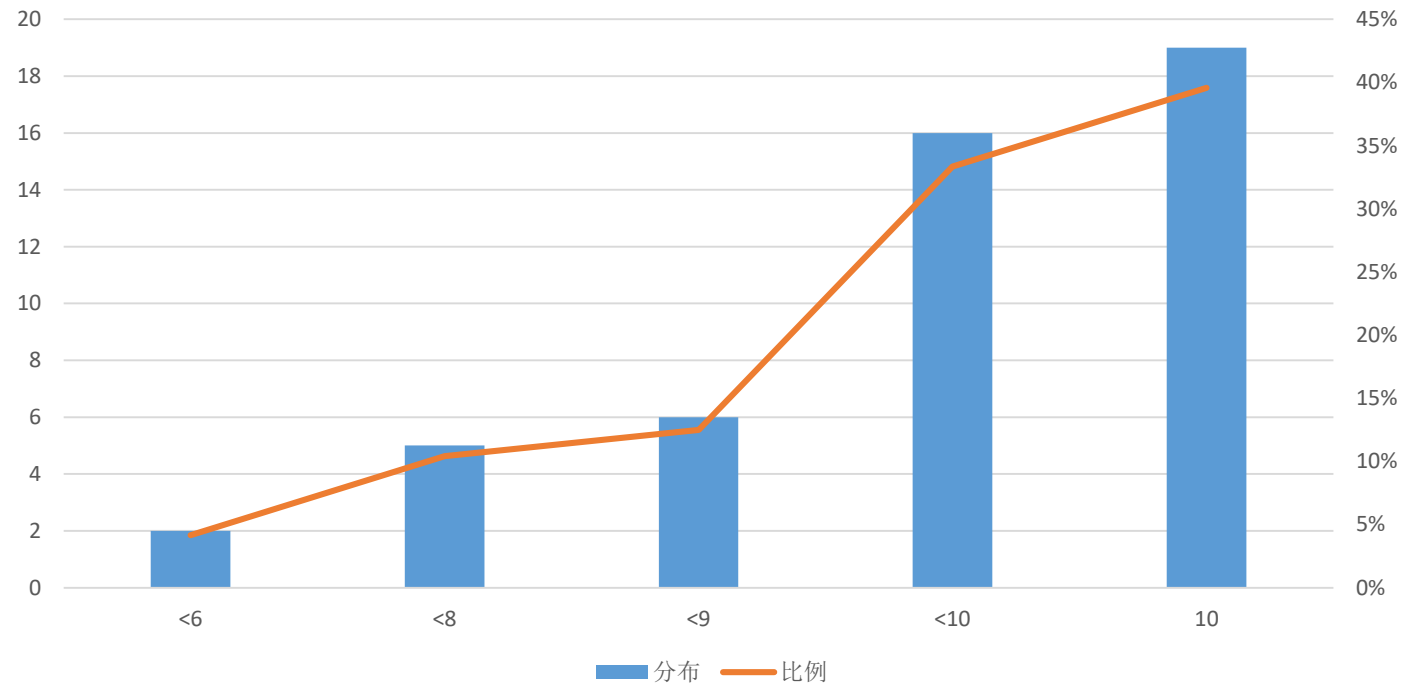
```
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<title>阿里V任务</title>
<meta name="description" content="阿里V任务? 阿里巴巴集团官方内容服务平台, 提供包括图文、直播、短视频、品牌营销、整合推广等全方位内容服务, 让您的店铺、宝贝、品牌在内容化潮流中脱颖而出。">
<meta name="keyword" content="阿里巴巴, 内容服务, 阿里V任务, 淘宝V任务, 天猫V任务, 1688V任务, 找创作者, 找机构, 找媒体, 淘宝内容化, 天猫众测, 阿里众测, 淘宝众测">
<!--[if lt IE 9]> <script>location.href = "http://www.taobao.com/markets/tbhome/ali-page-updater"; </script> <![endif-->
<link rel="stylesheet" type="text/css" charset="utf-8" href="//g.alicdn.com/mission/mission-assets/1.1.16/index.css">
<link rel="shortcut icon" type="image/x-icon" href="//gw.alicdn.com/tfs/TB1AzoQ5XXXXXXbXVXXXXXXXXXX-32-32.png">
<!--plus相关meta-->
<meta name="data-spm" content="a21xh">
<meta name="anplus.atav" content="rbksum">
```

html:ks-gecko69.ks-gecko.ks-firefox69.ks... > head > meta

过滤样式

```
元素 { 内联
}
*, ::after, ::before {
  box-sizing: border-box;
}
继承自 html
html {
  -webkit-text-size-adjust: 100%;
  font-weight: 400;
  font-size: 100%;
}
body, html {
```

期中考核分布



	分布	比例
<6	2	4%
<8	5	10%
<9	6	13%
<10	16	33%
10	19	40%

共48位同学

""" 输入区间 [a,b], 对于其中不是5的整数的数字, 按照 三角形的形式 print

input: 1 12

output:

1

2 3

4 6 7

8 9 11 12

input: 5 15

6

7 8

9 11 12

13 14

"""

#容易出错的点

打印的数值超出上界

回行不正确

```
a,b = map(int, input('输入两个整数 ').split())
```

```
nums = list(range(a,b+1))
```

```
nums = [i for i in nums if i%5] #重新构建列表, 去掉5整除的数
```

```
idx = 0 # 当前行的开始元素
```

```
ll = 1 # 当前行的长度
```

```
while idx < len(nums):
```

```
    print(" ".join(map(str, nums[idx:idx+ll])))
```

```
    idx += ll
```

```
    ll += 1
```

""" 对于一个整数, 翻转其中的数字部分 (去除前导零)

input: 0

0

input: 123

321

input: -120

-21

input: 9876543210

123456789

"""

易错点:

0的额外处理

代码不具有可扩展性

$b = (a // 100) + (((a \% 100) // 10) * 10) + ((a \% 10) * 100)$

```
a = int(input("输入:"))
```

```
if a==0: # 对输入0做特别处理
```

```
    print(0); exit()
```

```
flag = a>0
```

```
a = abs(a) #按绝对值计算, 最后加符号
```

```
ans = 0
```

```
while a:
```

```
    a, b = divmod(a, 10)
```

```
    ans = ans*10 + b
```

```
print(ans if flag else -ans)
```



```
""" 要求写一个函数 list_shifting(l) 实现列表循环左移一位.
"""
```

```
def list_shifting(l:list):
    return l[1:] + [l[0]]
```

```
""" 对于输入的底边长度奇数 x, 输出三角形, 每行的长度分别为 1,3,...x
```

```
input: 5
```

```
*
```

```
***
```

```
*****
```

```
"""
```

```
# 易错点
```

```
#对（正）奇数有无检验
```

```
for i in range(1, x+1,2):
```

```
    a = (x-i)//2
```

```
    print(" "*a + "*" * i + " "*a)
```

```
for i in range(1,n+1,2):
```

```
    print(("*" * i).center(n,))
```

""" 利用泰勒展开计算 $\sin(x)$ 的近似值, 需要手动实现 `frac` 函数, 要求精度 $1e-6$

计算公式: $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots$

通项公式: $(-1)^{(n+1)} * x^{(2n-1)} / (2n-1)!$

input: 0, $\pi/2$, 1, $-\pi/2$...

$\sin(0.00) = 0.000000$

$\sin(1.57) = 1.000000$

$\sin(2.00) = 0.909297$

$\sin(-1.57) = -1.000000$

"""

易错点

#习惯for循环并缺少条件判断语句

```
def poly_sin(x):
```

```
    s=0
```

```
    for i in range(1,100):
```

```
        a= $(-1)^{(i+1)} * (x^{(2*i-1)}) / \text{frac}(2*i-1)$ 
```

```
        s=s+a
```

```
    return s
```

```
print(poly_sin(1))
```

题意理解瑕疵, 语法完全正确, 算对

```
def poly_sin(a):
```

```
    b = 0
```

```
    i = 1
```

```
    while abs( $b - \text{math.sin}(a)$ ) >= math.pow(0.1,6):
```

```
        b += math.pow(a,i)*math.pow(-1,(i-1)/2)/frac(i)
```

```
        i += 2
```

```
    return b
```

#更好点的做法

```
eps = 1e-6
```

```
def poly_sin(x):
```

```
    acc = 0
```

```
    n = 1
```

```
    while True:
```

```
         $\text{delta} = (-1)^{(n+1)} * x^{(2*n-1)} / \text{frac}(2*n-1)$ 
```

```
        if abs(delta) < eps: break
```

```
        acc += delta
```

```
        n += 1
```

```
    return acc
```



新闻聚焦



十年，复旦科研强劲动能的秘密是…… | 见证十年

2021年11月8日，北京人民大会堂，2020



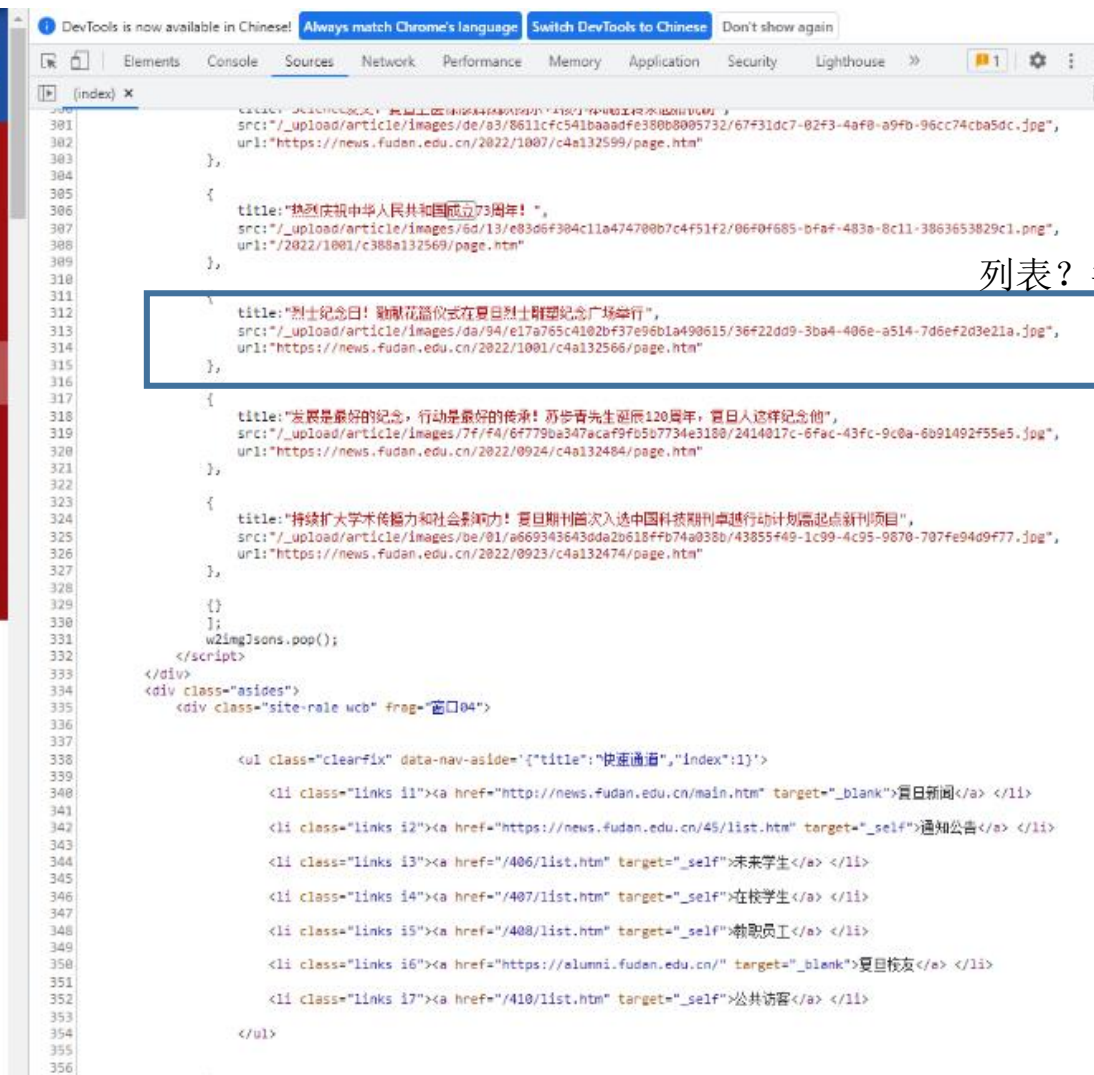
让学生探索属于自己的无限可能

坚守一份育人初心，计算机科学技术学院教授黄萱菁躬耕讲台二十余载。她……



凝心聚力、服务大局，建设“第一个复旦”！复旦大学统战工……

10月8日，复旦大学统战工作会议召开。



列表？字典？



八爪鱼教程视频，陆续更新上抖音

[关注送模板](#)[产品](#)[价格](#)[解决方案](#)[采集模板](#)[软件下载](#)[教程与帮助](#)[关于我们](#)[登录](#)[注册](#)

不懂网络爬虫技术，也可轻松采集数据

第一步

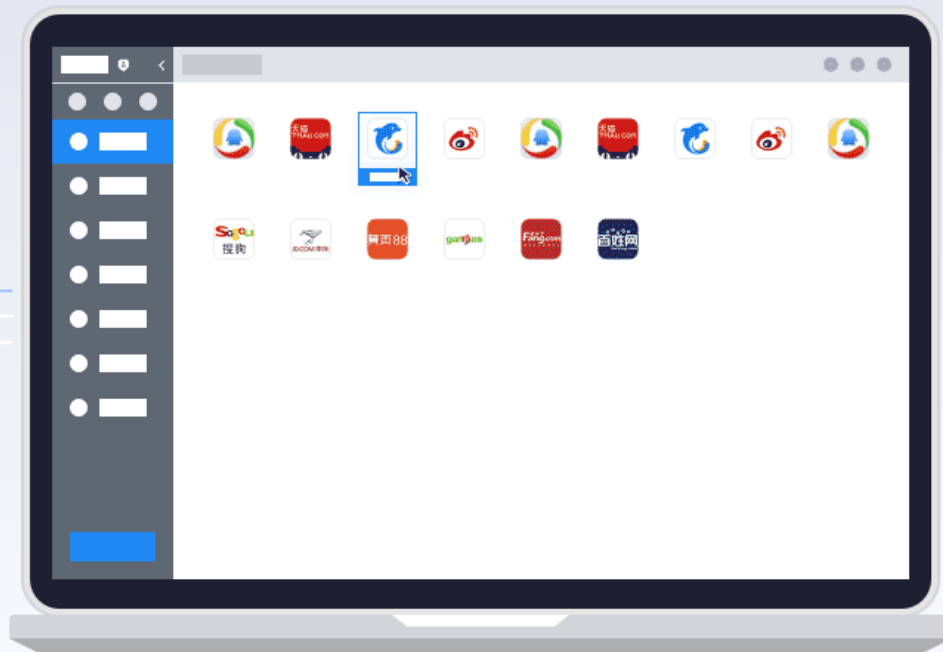
打开客户端，选择简易模式和相应的网站模板

第二步

预览模板的采集字段、参数设置和示例数据

第三步

设置对应的参数，保存运行完成数据采集

[试用简易模式](#)[5分钟DEMO演示](#)

操作简单·功能强大·满足你的所有需求

人人可用的数据采集器

```
import requests
```

```
#Open the website
```

```
url = "https://news.baidu.com/"
```

```
page = requests.get(url)
```

```
page_content = page.content
```

```
filename="d:\\baidu.txt"
```

```
with open(filename, 'wb') as fp:
```

```
    fp.write(page_content)
```

```
fp.close()
```

Baidu 新闻

百度一下 帮助

热点要闻

- 习近平：弘扬伟大建党精神和延安精神
- 鉴往知来，跟着总书记学党史 走进延安革命纪念馆
- 党的二十大报告，提到这些战略
- 坚持和发展马克思主义必须做好“两个结合”
- 商务部：我国消费市场总体延续恢复增长态势
- 广西南玉铁路全线41座隧道全部贯通
- 高手相逢，谁更胜一筹？且看中国“科技群英谱”
- 多国资深媒体人：中国发展将惠及世界
- 美国通货膨胀持续飙升 10月消费者信心指数下降
- 苗圃腌鱼 稻香鱼肥说丰年
- 越南、德国高层访华你追我赶，阮富仲为什么比朔尔茨快？
- 北京网络辟谣 互联网联合辟谣平台 网络辟谣标签工作专区
- 收藏！30个要点带您速览二十大党章的重要修改
- 石泰峰已任中央统战部副部长
- 国际航班总量将增加一倍 入境机票价格大幅回落
- 无症状轻症多、或能自愈，新冠治疗还需口服药吗？专家解答
- 将“反台独”写入党章，体现怎样的对台方略？
- 国防部：敦促美方停止售台武器和美台军事联系
- 他带着一个媒体团到访，却没等来记者会
- 俄罗斯：北约原子弹已在乌克兰，已做好核环境作战准备

故宫国博首次联袂办展 吸引游客参观领略“和合”文化

理上网来·理论新境界

热搜新闻词 HOT WORDS

中国式现代化 为他国发展提供借鉴	一组数字看中国航天的 圆梦故事	人民币汇率为何 大幅上涨	31省份新增本土 193+924
广州海关查获新 型毒品“听话水”	汶川地震被救 少年14年后救火 牺牲	金门大桥将于10 月30日通车	民宿回应透明底 浴缸不会泄露 隐私
长沙宁乡巨响系 飞机音爆导致	日本梅毒确诊者 年度累计首超 1万例		

百家号 BAIJIA

谷歌：衰退逼近，广告一哥已卧倒


```

<!doctype html>
<html class="expanded">
<head>

<!--STATUS OK-->
<meta http-equiv=Content-Type content="text/html;charset=utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">
<link rel="icon" href="//gss0.bdstatic.com/5folcy0a2gl2n2jgoY3K/static/fisp_static/common/img/favicon.ico" mce_href="../../static/img/favicon.ico"
type="image/x-icon">

<title>百度新闻——海量中文资讯平台</title>
<meta name="description" content="百度新闻是包含海量资讯的新闻服务平台，真实反映每时每刻的新闻热点。您可以搜索新闻事件、热点话题、人物动态、产品资讯等，
快速了解它们的最新进展。" >
<script type="text/javascript">
        document.write("<script type='text/javascript' src='//news-bos.cdn.bcebos.com/mvideo/pccconf_2019.js?' + new Date().getTime() + '><
\\script>");
    </script>
<script type="text/javascript"> window.NEWSLOGURL = 'https://log.news.baidu.com/v.gif'; window.HUNTERLOGURL = '//log.news.baidu.com/u.gif';
window._hmt = window._hmt || [];</script>
<script type="text/javascript" src="//gss0.bdstatic.com/5folcy0a2gl2n2jgoY3K/static/fisp_static/common/resource/js/usermonitor_88a158c.js?v=1.2"></script>
<script defer async type="text/javascript" src="//gss0.bdstatic.com/5folcy0a2gl2n2jgoY3K/static/fisp_static/wza/aria.js?
appid=c890648bf4dd00d05eb9751dd0548c30" charset="utf-8"></script>

<script src="//gss0.bdstatic.com/5folcy0a2gl2n2jgoY3K/static/fisp_static/news/js/jquery-1.8.3.min_a6ffa58.js" type="text/javascript"></script>
<script src="https://efe-h2.cdn.bcebos.com/cliresource/ubc-report-sdk/2.0.8/ubc-web-sdk.umd.min.js"></script>

<link rel="stylesheet" type="text/css"
href="//gss0.bdstatic.com/5folcy0a2gl2n2jgoY3K/static/fisp_static/common/module_static_include/module_static_include_130fb43.css"/> <link rel="stylesheet"
type="text/css" href="//gss0.bdstatic.com/5folcy0a2gl2n2jgoY3K/static/fisp_static/news/focustop/focustop_415cfee.css"/> </head>
<body>

```

1 正则表达式语法

- 正则表达式由元字符及其不同组合来构成，通过巧妙地构造正则表达式可以匹配任意字符串，并完成查找、替换、分隔等复杂的字符串处理任务。

元字符	功能说明
.	匹配除换行符以外的任意单个字符
*	匹配位于*之前的字符或子模式的0次或多次出现
+	匹配位于+之前的字符或子模式的1次或多次出现
-	在[]之内用来表示范围
	匹配位于 之前或之后的字符
^	匹配行首，匹配以^后面的字符开头的字符串
\$	匹配行尾，匹配以\$之前的字符结束的字符串
?	匹配位于?之前的0个或1个字符。当此字符紧随任何其他限定符（*、+、?、{n}、{n,}、{n,m}）之后时，匹配模式是“非贪心的”。“非贪心的”模式匹配搜索到的、尽可能短的字符串，而默认的“贪心的”模式匹配搜索到的、尽可能长的字符串。例如，在字符串“oooo”中，“o+?”只匹配单个“o”，而“o+”匹配所有“o”
\	表示位于\之后的为转义字符
\num	此处的num是一个正整数，表示子模式编号。 例如，“(.)\1”匹配两个连续的相同字符
\f	换页符匹配
\n	换行符匹配

1.1 正则表达式基本语法

元字符	功能说明
\r	匹配一个回车符
\b	匹配单词头或单词尾
\B	与\b含义相反
\d	匹配任何数字，相当于[0-9]
\D	与\d含义相反，等效于[^0-9]
\s	匹配任何空白字符，包括空格、制表符、换页符，与 [\f\n\r\t\v] 等效
\S	与\s含义相反
\w	匹配任何字母、数字以及下划线，相当于[a-zA-Z0-9_]
\W	与\w含义相反\w含义相反，与“[^A-Za-z0-9_]”等效
()	将位于()内的内容作为一个整体来对待
{m, n}	{ }前的字符或子模式重复至少m次，至多n次
[]	表示范围，匹配位于[]中的任意一个字符
[^xyz]	反向字符集，匹配除x、y、z之外的任何字符
[a-z]	字符范围，匹配指定范围内的任何字符
[^a-z]	反向范围字符，匹配除小写英文字母之外的任何字符

正则表达式`^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$`可以检查字符串是否为IP地址



问7：和之前学的字符串查找有什么区别？

1.1 正则表达式基本语法

- 如果以“\”开头的元字符与转义字符相同，则需要使用“\\”，或者使用原始字符串。
- 在字符串前加上字符r或R之后表示原始字符串，字符串中任意字符都不再进行转义。原始字符串可以减少用户的输入，主要用于正则表达式和文件路径字符串的情况，但如果字符串以一个斜线“\”结束的话，则需要多写一个斜线，即以“\\”结束。

1.2 正则表达式扩展语法

- 正则表达式使用圆括号 “()” 表示一个子模式，圆括号内的内容作为一个整体对待，例如 '(red)+' 可以匹配 'redred'、'redredred' 等一个或多个重复 'red' 的情况。
- 使用子模式扩展语法可以实现更加复杂的字符串处理功能。
 - ✓ (pattern)*: 允许模式重复0次或多次
 - ✓ (pattern)+: 允许模式重复1次或多次
 - ✓ (pattern){m,n}: 允许模式重复m~n次

1.3 正则表达式集锦

- ✓ 最简单的正则表达式是普通字符串，可以匹配自身
- ✓ '[pjc]ython'可以匹配'python'、'jython'、'cython'
- ✓ '[a-zA-Z0-9]'可以匹配一个任意大小写字母或数字
- ✓ '[^abc]'可以一个匹配任意除'a'、'b'、'c'之外的字符
- ✓ 'python|perl'或'p(ython|erl)'都可以匹配'python'或'perl'
- ✓ 子模式后面加上问号表示可选。r'(http://)?(www\.)?python\.org'只能匹配'
'http://www.python.org'、'http://python.org'、'www.python.org'和'python.org'
- ✓ '^http'只能匹配所有以'http'开头的字符串

1.3 正则表达式集锦

- ✓ `'(a|b)*c'`: 匹配多个（包含0个）a或b，后面紧跟一个字母c。
- ✓ `'ab{1,}'`: 等价于`'ab+'`，匹配以字母a开头后面带1个至多个字母b的字符串。
- ✓ `'^[a-zA-Z]{1}([a-zA-Z0-9._]){4,19}$'`: 匹配长度为5-20的字符串，**必须以字母开头**并且可带字母、数字、“_”、“.”的字符串。
- ✓ `'^(\w){6,20}$'`: 匹配长度为6-20的字符串，可以包含字母、数字、下划线。
- ✓ `'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$'`: 检查给定字符串是否为合法IP地址。
- ✓ `'^(13[4-9]\d{8})|(15[01289]\d{8})$'`: 检查给定字符串是否为手机号码。
- ✓ `'^[a-zA-Z]+$'`: 检查给定字符串是否只包含英文字母大小写。
- ✓ `'^\w+@(\w+\.)+\w+$'`: 检查给定字符串是否为合法电子邮件地址。
- ✓ `r'(\w)(?!.*\1)'`: 查找字符串中每个字符的最后一次出现。
- ✓ `r'(\w)(?=.*\1)'`: 查找字符串中所有重复出现的字符。

1.3 正则表达式集锦

- ✓ `'^(\-)?\d+(\.\d{1,2})?$',` 检查给定字符串是否为最多带有2位小数的正数或负数。
- ✓ `'[\u4e00-\u9fa5]'` 匹配给定字符串中所有汉字。
- ✓ `'^\d{18}|\d{15}$'` 检查给定字符串是否为合法身份证格式。
- ✓ `'\d{4}-\d{1,2}-\d{1,2}'` 匹配指定格式的日期，例如2016-1-31。
- ✓ `'(.+)\1+'` 匹配任意字符的一次或多次重复出现。

1.2 正则表达式扩展语法

语法	功能说明
(?P<groupname>)	为子模式命名
(?iLmsux)	设置匹配标志，可以是几个字母的组合，每个字母含义与编译标志相同
(?:...)	匹配但不捕获该匹配的子表达式
(?P=groupname)	表示在此之前的命名为groupname的子模式
(?#...)	表示注释
(?<=...)	用于正则表达式之前，表示如果<=后的内容在字符串中不出现则匹配，但不返回<=之后的内容
(?=...)	用于正则表达式之后，表示如果=后的内容在字符串中出现则匹配，但不返回=之后的内容
(?<!=...)	用于正则表达式之前，表示如果<!=后的内容在字符串中不出现则匹配，但不返回<!=之后的内容
(?!...)	用于正则表达式之后，表示如果!后的内容在字符串中不出现则匹配，但不返回!之后的内容

- ✓ '((?P<f>\b\w+\b)s+(?P=f))': 匹配连续出现两次的单词。
- ✓ '((?P<f>.)?(?P=f)(?P<g>.)?(?P=g))': 匹配AABB形式的成语或字母组合。
- ✓ '^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[.,_]).{8,}\$': 检查给定字符串是否为强密码，必须同时包含英语字母大写字母、英文小写字母、数字或特殊符号（如英文逗号、英文句号、下划线），并且长度必须至少8位。
- ✓ "(?!.*[\"'\\/;=%?]).+": 如果给定字符串中包含'、”、/、;、=、%、?则匹配失败。

1.3 正则表达式集锦

- 使用时要注意的是，正则表达式只是进行形式上的检查，并不保证内容一定正确。
- 例如上面的例子中，正则表达式`'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$'`可以检查字符串是否为IP地址，字符串'888.888.888.888'这样的也能通过检查，但实际上并不是有效的IP地址。
- 同样的道理，正则表达式`'^\d{18}|\d{15}$'`也只负责检查字符串是否为18位或15位数字，并不保证一定是合法的身份证号。

2 直接使用正则表达式模块re处理字符串

- Python标准库re模块提供了正则表达式操作所需要的功能。

`import re`

方法	功能说明
<code>compile(pattern[, flags])</code>	创建模式对象
<code>escape(string)</code>	将字符串中所有特殊正则表达式字符转义
<code>findall(pattern, string[, flags])</code>	返回包含字符串中所有与给定模式匹配的项的列表
<code>finditer(pattern, string, flags=0)</code>	返回包含所有匹配项的迭代对象，其中每个匹配项都是match对象
<code>fullmatch(pattern, string, flags=0)</code>	尝试把模式作用于整个字符串，返回match对象或None
<code>match(pattern, string[, flags])</code>	从字符串的 开始处 匹配模式，返回match对象或None
<code>purge()</code>	清空正则表达式缓存
<code>search(pattern, string[, flags])</code>	在 整个字符串 中寻找模式，返回match对象或None
<code>split(pattern, string[, maxsplit=0])</code>	根据模式匹配项分隔字符串
<code>sub(pat, repl, string[, count=0])</code>	将字符串中所有与pat匹配的项用repl替换，返回新字符串，repl可以是字符串或返回字符串的可调用对象，作用于每个匹配的match对象
<code>subn(pat, repl, string[, count=0])</code>	将字符串中所有pat的匹配项用repl替换，返回包含新字符串和替换次数的二元元组，repl可以是字符串或返回字符串的可调用对象，作用于每个匹配的match对象

2 直接使用正则表达式模块re处理字符串

- 其中函数参数“**flags**”的值可以是下面几个的不同组合（使用“|”进行组合）：
 - ✓ **re.I**（注意是大写字母I，不是数字1，表示忽略大小写）
 - ✓ **re.L**（支持本地字符集的字符）
 - ✓ **re.M**（多行匹配模式）
 - ✓ **re.S**（使元字符“.”匹配任意字符，包括换行符）
 - ✓ **re.U**（匹配Unicode字符）
 - ✓ **re.X**（忽略模式中的空格，并可以使用#注释）

2 直接使用正则表达式模块re处理字符串

```
>>> import re                                #导入re模块
>>> text = 'alpha. beta....gamma delta'      #测试用的字符串
>>> re.split('[\. ]+', text)                  #使用指定字符作为分隔符进行分隔
['alpha', 'beta', 'gamma', 'delta']
>>> re.split('[\. ]+', text, maxsplit=2)      #最多分隔2次
['alpha', 'beta', 'gamma delta']
>>> re.split('[\. ]+', text, maxsplit=1)      #最多分隔1次
['alpha', 'beta....gamma delta']
>>> pat = '[a-zA-Z]+'
>>> re.findall(pat, text)                      #查找所有单词
['alpha', 'beta', 'gamma', 'delta']
```

2 直接使用正则表达式模块re处理字符串

```
>>> pat = '{name}'
>>> text = 'Dear {name}...'
>>> re.sub(pat, 'Mr.Zhang', text)          #字符串替换
'Dear Mr.Zhang...'
>>> s = 'a s d'
>>> re.sub('a|s|d', 'good', s)            #字符串替换
'good good good'
>>> s = "It's a very good good idea"
>>> re.sub(r'(\b\w+) \1', r'\1', s)        #处理连续的重复单词
"It's a very good idea"
>>> re.sub(r'((\w+) )\1', r'\2', s)
"It's a very goodidea"
>>> re.sub('a', lambda x:x.group(0).upper(), 'aaa abc abde')
#repl为可调用对象
'AAA Abc Abde'
```

2 直接使用正则表达式模块re处理字符串

```
>>> re.sub('[a-z]', lambda x:x.group(0).upper(), 'aaa abc abde')
'AAA ABC ABDE'
>>> re.sub('[a-zA-z]', lambda x:chr(ord(x.group(0))^32), 'aaa aBc abde')
#英文字母大小写互换
'AAA AbC ABDE'
>>> re.subn('a', 'dfg', 'aaa abc abde') #返回新字符串和替换次数
('dfgdfgdfg dfghbc dfghbde', 5)
>>> re.sub('a', 'dfg', 'aaa abc abde')
'dfgdfgdfg dfghbc dfghbde'
>>> re.escape('http://www.python.org') #字符串转义
'http\\:\\\\\\/\\www\\.python\\.org'
```

2 直接使用正则表达式模块re处理字符串

```
>>> print(re.match('done|quit', 'done'))           #匹配成功，返回match对象
<_sre.SRE_Match object at 0x00B121A8>
>>> print(re.match('done|quit', 'done!'))           #匹配成功
<_sre.SRE_Match object at 0x00B121A8>
>>> print(re.match('done|quit', 'doe!'))           #匹配不成功，返回空值None
None
>>> print(re.match('done|quit', 'd!one!'))          #匹配不成功
None
>>> print(re.search('done|quit', 'd!one!done'))     #匹配成功
<_sre.SRE_Match object at 0x0000000002D03D98>
```

2 直接使用正则表达式模块re处理字符串

- 下面的代码使用不同的方法删除字符串中多余的空格，如果遇到连续多个空格则只保留一个，同时删除字符串两侧的所有空白字符。

```
>>> import re
>>> s = 'aaa      bb      c d e   fff   '
>>> ' '.join(s.split()) #直接使用字符串对象的方法
'aaa bb c d e fff'
>>> ' '.join(re.split('[\s]+', s.strip())) #同时使用re中的函数和字符串对象的方法
'aaa bb c d e fff'
>>> ' '.join(re.split('\s+', s.strip())) #与上一行代码等价
'aaa bb c d e fff'
>>> re.sub('\s+', ' ', s.strip()) #直接使用re模块的字符串替换方法
'aaa bb c d e fff'
```

2 直接使用正则表达式模块re处理字符串

- 下面的代码使用几种不同的方法来删除字符串中指定内容:

```
>>> email = "tony@tiremove_thisger.net"
>>> m = re.search("remove_this", email)           #使用search()方法返回的match对象
>>> email[:m.start()] + email[m.end():]          #字符串切片
'tony@tiger.net'
>>> re.sub('remove_this', '', email)             #直接使用re模块的sub()方法
'tony@tiger.net'
>>> email.replace('remove_this', '')              #直接使用字符串替换方法
'tony@tiger.net'
```


2 直接使用正则表达式模块re处理字符串

- 下面的代码使用以“\”开头的元字符来实现字符串的特定搜索。

```
>>> import re
>>> example = 'Beautiful is better than ugly.'
>>> re.findall('\bb.+?\b', example)      #以字母b开头的完整单词
                                           #此处问号?表示非贪心模式
['better']
>>> re.findall('\bb.+?\b', example)      #贪心模式的匹配结果
['better than ugly']
>>> re.findall('\bb\w*\b', example)
['better']
>>> re.findall('\Bh.+?\b', example)      #不以h开头且含有h字母的单词剩余部分
['han']
```

2 直接使用正则表达式模块re处理字符串

```
>>> re.findall('\b\w.+?\b', example)           #所有单词
['Beautiful', 'is', 'better', 'than', 'ugly']
>>> re.findall('\w+', example)                 #所有单词
['Beautiful', 'is', 'better', 'than', 'ugly']
>>> re.findall(r'\b\w.+?\b', example)          #使用原始字符串
['Beautiful', 'is', 'better', 'than', 'ugly']
>>> re.split('\s', example)                    #使用任何空白字符分隔字符串
['Beautiful', 'is', 'better', 'than', 'ugly.']
>>> re.findall('\d+\.?\d+\.?\d+', 'Python 2.7.13') #查找并返回x.x.x形式的数字
['2.7.13']
>>> re.findall('\d+\.?\d+\.?\d+', 'Python 2.7.13,Python 3.6.0')
['2.7.13', '3.6.0']
>>> s = '<html><head>This is head.</head><body>This is body.</body></html>'
>>> pattern = r'<html><head>(.)</head><body>(.)</body></html>'
>>> result = re.search(pattern, s)
>>> result.group(1)                            #第一个子模式
'This is head.'
>>> result.group(2)                            #第二个子模式
'This is body.'
```

3 使用正则表达式对象处理字符串

- 首先使用re模块的`compile()`方法将正则表达式编译生成正则表达式对象，然后再使用正则表达式对象提供的方法进行字符串处理。
- 使用编译后的正则表达式对象可以**提高字符串处理速度**，**也提供了更强大的文本处理功能**。

3 使用正则表达式对象处理字符串

- `match()`、`search()`、`findall()`
- ✓ `match(string[, pos[, endpos]])`方法在字符串开头或指定位置进行搜索，**模式必须出现在字符串开头或指定位置**；
- ✓ `search(string[, pos[, endpos]])`方法在**整个字符串或指定范围**中进行搜索；
- ✓ `findall(string[, pos[, endpos]])`方法在字符串**指定范围**中**查找所有**符合正则表达式的字符串并以列表形式返回。

3 使用正则表达式对象处理字符串

```
>>> import re
>>> example = 'Fudan University'
>>> pattern = re.compile(r'\bU\w+\b')      #查找以U开头的单词
>>> pattern.findall(example)                #使用正则表达式对象的findall()方法
['University']
>>> pattern = re.compile(r'\w+n\b')        #查找以字母n结尾的单词
>>> pattern.findall(example)
['Fudan']
>>> pattern = re.compile(r'\b[a-zA-Z]{3}\b') #查找3个字母长的单词
>>> pattern.findall(example)
['and']
>>> s = 'ab134ab98723jafjweoruiagab'
>>> m = re.search(r'((ab).*){2}.*(ab)', s) #在s中查找ab的第3次出现
>>> m.group(3)
'ab'
>>> m.span(3)
(24, 26)
>>> s[24:]
'ab'
```

3 使用正则表达式对象处理字符串

```
>>> pattern.match(example)           #从字符串开头开始匹配, 失败返回空值
>>> pattern.search(example)           #在整个字符串中搜索, 成功
<_sre.SRE_Match object; span=(31, 34), match='and'>
>>> pattern = re.compile(r'\b\w*a\w*\b') #查找所有含有字母a的单词
>>> pattern.findall(example)
['Fudan']
>>> text = "He was carefully disguised but captured quickly by police."
>>> re.findall(r"\w+ly", text)         #查找所有以字母组合ly结尾的单词
['carefully', 'quickly']
```

3 使用正则表达式对象处理字符串

■ sub()、subn()

- ✓ 正则表达式对象的sub(repl, string[, count = 0])和subn(repl, string[, count = 0])方法用来实现字符串替换功能，其中参数repl可以为字符串或返回字符串的可调用对象。

```
>>> example = '''Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.'''
```

3 使用正则表达式对象处理字符串

```
>>> pattern = re.compile(r'\bb\w*\b', re.I) #匹配以b或B开头的单词  
>>> print(pattern.sub('*', example))      #将符合条件的单词替换为*
```

```
* is * than ugly.
```

```
Explicit is * than implicit.
```

```
Simple is * than complex.
```

```
Complex is * than complicated.
```

```
Flat is * than nested.
```

```
Sparse is * than dense.
```

```
Readability counts.
```


3 使用正则表达式对象处理字符串

```
>>> print(pattern.sub(lambda x: x.group(0).upper(), example))
```

#把所有匹配项都改为大写

BEAUTIFUL is BETTER than ugly.

Explicit is BETTER than implicit.

Simple is BETTER than complex.

Complex is BETTER than complicated.

Flat is BETTER than nested.

Sparse is BETTER than dense.

Readability counts.

3 使用正则表达式对象处理字符串

```
>>> print(pattern.sub('*', example, 1))
```

 #只替换1次

```
* is better than ugly.
```

```
Explicit is better than implicit.
```

```
Simple is better than complex.
```

```
Complex is better than complicated.
```

```
Flat is better than nested.
```

```
Sparse is better than dense.
```

```
Readability counts.
```

3 使用正则表达式对象处理字符串

```
>>> pattern = re.compile(r'\bb\w*\b')    #匹配以字母b开头的单词  
>>> print(pattern.sub('*', example, 1)) #将符合条件的单词替换为*  
                                         #只替换1次
```

```
Beautiful is * than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.
```

3 使用正则表达式对象处理字符串

- 正则表达式对象的split(string[, maxsplit = 0])方法用来实现字符串分隔。

```
>>> example = r'one,two,three.four/five\six?seven[eight]nine|ten'
>>> pattern = re.compile(r'[.,/\|\?[\]\|]')      #指定多个可能的分隔符
>>> pattern.split(example)
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
>>> example = r'one1two2three3four4five5six6seven7eight8nine9ten'
>>> pattern = re.compile(r'\d+')                  #使用数字作为分隔符
>>> pattern.split(example)
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
>>> example = r'one two      three  four,five.six.seven,eight,nine9ten'
>>> pattern = re.compile(r'[\s,.\d]+')            #允许分隔符重复
>>> pattern.split(example)
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

4 match对象

- 正则表达式对象的match方法和search方法匹配成功后返回match对象。

match对象的主要方法有：

- ✓ group()：返回匹配的一个或多个子模式内容
- ✓ groups()：返回一个包含匹配的所有子模式内容的元组
- ✓ groupdict()：返回包含匹配的所有命名子模式内容的字典
- ✓ start()：返回指定子模式内容的起始位置
- ✓ end()：返回指定子模式内容的结束位置的前一个位置
- ✓ span()：返回一个包含指定子模式内容起始位置和结束位置前一个位置的元组。

4 match对象

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, physicist")
>>> m.group(0)                #返回整个模式内容
'Isaac Newton'
>>> m.group(1)                #返回第1个子模式内容
'Isaac'
>>> m.group(2)                #返回第2个子模式内容.
'Newton'
>>> m.group(1, 2)             #返回指定的多个子模式内容
('Isaac', 'Newton')
```

4 match对象

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.group('first_name')          #使用命名的子模式
'Malcolm'
>>> m.group('last_name')
'Reynolds'
>>> m = re.match(r"(\d+)\.(\d+)", "24.1632")
>>> m.groups()                    #返回所有匹配的子模式（不包括第0个）
('24', '1632')
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.groupdict()                 #以字典形式返回匹配的结果
{'first_name': 'Malcolm', 'last_name': 'Reynolds'}
```

4 match对象

```
>>> exampleString = '''There should be one-- and preferably only one --  
obvious way to do it.
```

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than right now.'''

```
>>> pattern = re.compile(r'(?<=\w\s)never(?:=\s\w)')
```

#查找不在句子开头和结尾的

never

```
>>> matchResult = pattern.search(exampleString)
```

```
>>> matchResult.span()
```

```
(172, 177)
```


4 match对象

```
>>> pattern = re.compile(r'(?i)\bn\w+\b')    #查找以n或N字母开头的所有单词
>>> index = 0
>>> while True:
    matchResult = pattern.search(exampleString, index)
    if not matchResult:
        break
    print(matchResult.group(0), ': ', matchResult.span(0))
    index = matchResult.end(0)

not : (92, 95)
Now : (137, 140)
never : (156, 161)
never : (172, 177)
now : (205, 208)
```

4 match对象

```
>>> pattern = re.compile(r'(?<!not\s)be\b')  
                                     #查找前面没有单词not的单词be  
>>> index = 0  
>>> while True:  
    matchResult = pattern.search(exampleString, index)  
    if not matchResult:  
        break  
    print(matchResult.group(0), ': ', matchResult.span(0))  
    index = matchResult.end(0)  
  
be : (13, 15)  
>>> exampleString[13:20]  
'be one-'
```

#验证一下结果是否正确

4 match对象

```
>>> pattern = re.compile(r'(\b\w*(?P<f>\w+)(?P=f)\w*\b)')    #有连续相同字母的单词
>>> index = 0
>>> while True:
    matchResult = pattern.search(exampleString, index)
    if not matchResult:
        break
    print(matchResult.group(0), ': ', matchResult.group(2))
    index = matchResult.end(0) + 1
unless : s
better : t
better : t
>>> s = 'aabc abcd abbcd abccd abcd'
>>> pattern.findall(s)
[('aabc', 'a'), ('abbcd', 'b'), ('abccd', 'c'), ('abcd', 'd')]
```

5 应用举例

- 示例8-1 使用正则表达式提取字符串中的电话号码。

```
import re
```

```
telNumber = '''Suppose my Phone No. is 0535-1234567,  
             yours is 010-12345678,  
             his is 025-87654321.'''
```

```
pattern = re.compile(r'(\d{3,4})-(\d{7,8})')
```

#注意，逗号后面不能有空格

```
index = 0
```

```
while True:
```

```
    matchResult = pattern.search(telNumber, index)
```

#从指定位置开始匹配

```
    if not matchResult:
```

```
        break
```

```
    print('-'*30)
```

```
    print('Success:')
```

```
    for i in range(3):
```

```
        print('Searched content:', matchResult.group(i),\
```

```
            ' Start from:', matchResult.start(i), 'End at:', matchResult.end(i),\
```

```
            ' Its span is:', matchResult.span(i))
```

```
    index = matchResult.end(2)
```

#指定下次匹配的起始位置