05

第 5 章

正则表达式

简答 5-1：请说明正则表达式的模块。
简答 5-2：用正则表达式表达符合格式要求的电话号码。
简答 5-3：简化正则表达式。
简答 5-4：使用括号分组以及简化正则表达式。
简答 5-5：在正则表达式 \d 中，如果使用管道概念 |，应如何表达？
简答 5-6：在正则表达式中，请说明 \d、\D、\s、\S、\w、\W 字符的用法。
面试实例 ch5_1.py：使用正则表达式将电话号码从一段文字抽离出来。
面试实例 ch5_2.py：正则表达式使用 re.search( ) 时，请说明 group( ) 方法。
面试实例 ch5_3.py：在 re 模块中，请说明 groups( ) 的功能。
面试实例 ch5_4.py：请参考面试实例 ch5_3.py，将字符串内的电话号码数据取出。
面试实例 ch5_5.py：搜寻字符串中的 2 个不同的字符串。
面试实例 ch5_6.py：以正则表达式概念搜寻 Johnson、Johnnason、Johnnathan。
面试实例 ch5_7.py：以正则表达式概念搜寻 John、Johnson、Johnnason、Johnnathan。
面试实例 ch5_8.py：以正则表达式概念搜寻 Johnson、Johnnason。
面试实例 ch5_9.py：以正则表达式概念搜寻 Johnson、Johnnason、Johnnanason。
面试实例 ch5_10.py：以正则表达式概念搜寻 Johnnason、Johnnanason。
面试实例 ch5_11.py：请说明 Python 搜寻时 { } 的用法。
面试实例 ch5_12.py：请说明 Python 的 re 模块搜寻是贪婪模式还是非贪婪模式。
面试实例 ch5_13.py：Python 的 re 模块默认是贪婪模式，改为非贪婪模式。

面试实例 ch5_14.py：单字分离程序。
面试实例 ch5_15.py：正则表达式的应用，将标题项目从句子中分离。
面试实例 ch5_16.py：在正则表达式中，请说明中括号 [ ] 的用途。
面试实例 ch5_17.py：在正则表达式中，请说明在中括号 [ ] 内增加 ^ 字符的用途。
面试实例 ch5_18.py：在正则表达式中，请说明 ^ 放在字符串前面的意义。
面试实例 ch5_19.py：在正则表达式中，请说明 $ 字符的意义。
面试实例 ch5_20.py：在正则表达式中，请说明通配符“.”的功能。
面试实例 ch5_21.py：在正则表达式中，请说明通配符“.*”的功能。
面试实例 ch5_22.py：在正则表达式中，请说明 sub( ) 和 subn( ) 的用法。
面试实例 ch5_23.py：字符串取代应用。
面试实例 ch5_24.py：在正则表示法中，请说明如何使用 *** 代替原本的姓名。
面试实例 ch5_25.py：请设计可以判别电子邮件地址的正则表达式。
面试实例 ch5_26.py：找出 @ 前面是 3 ～ 15 个字符，结尾是 @me.com 的电子邮件地址。
面试实例 ch5_27.py：用冒号或是空格切割字符串。
面试实例 ch5_28.py：列出指定手机号码。
面试实例 ch5_29.py：Unicode 基本汉字的应用。
面试实例 ch5_30.py：设计可以筛选匹配 <html><h1>xxxx</h1></html> 的应用。

简答 5-1：请说明正则表达式的模块。

解答：

re 模块。

简答 5-2：用正则表达式表达符合下列格式的电话号码：

xxxx-xxx-xxx

上述 x 代表一个 0 ～ 9 的阿拉伯数字。

解答：

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

简答 5-3：有一个正则表达式如下：

```
r'\d\d\d\d-\d\d\d-\d\d\d'
```

可否简化重复字符串的表达？

解答：

```
r'\d{4}-\d{3}-\d{3}'
```

简答 5-4：有一个正则表达式如下：

```
r'\d\d-\d\d\d\d\d\d\d\d\d'
```

如何使用括号分组以及简化？

解答：

以下是使用括号分组：

```
r'(\d\d)-(\d\d\d\d\d\d\d\d\d')
```

以下是简化：

```
r'(\d{2})-(\d{8})'
```

简答 5-5：在正则表达式 \d 中，如果使用管道概念 |，应如何表达？

解答：

```
(0|1|2|3|4|5|6|7|8|9)
```

简答 5-6：请说明下列字符在正则表达式中的用法：

\d、\D、\s、\S、\w、\W

解答：

字符	使用说明
\d	0 ～ 9 的整数字元
\D	除了 0 ～ 9 的整数字元以外的其他字符
\s	空白、定位、Tab 键、换行、换页字符
\S	除了空白、定位、Tab 键、换行、换页字符以外的其他字符
\w	数字、字母和下画线字符，以及 [A-Za-z0-9_]
\W	除了数字、字母和下画线字符，以及 [A-Za-z0-9_] 以外的其他字符

面试实例 ch5_1.py：使用正则表达式将电话号码从一段文字抽离出来。

```
# ch5_1.py
import re

msg1 = 'Please call my secretary using 0930-919-919 or 0952-001-001'
msg2 = '请明天17:30和我一起参加教师节晚餐'
msg3 = '请明天17:30和我一起参加教师节晚餐，可用0933-080-080联络我

def parseString(string):
    """解析字符串是否含有电话号码"""
    phoneRule = re.compile(r'\d\d\d-\d\d\d-\d\d\d')
    phoneNum = phoneRule.findall(string)      # 用列表传回搜寻结果
    print("电话号码是: %s" % phoneNum)        # 列表方式显示电话号码

parseString(msg1)
parseString(msg2)
parseString(msg3)
```

执行结果

```
================= RESTART: D:\Python interveiw\ch5\ch5_1.py =================
电话号码是: ['0930-919-919', '0952-001-001']
电话号码是: []
电话号码是: ['0933-080-080']
```

面试实例 ch5_2.py：正则表达式使用 re.search( ) 时，请说明 group( ) 方法，同时以程序实例说明。

解答：

下列是 group( ) 的相关说明：

group( )：回传第一个比对相符的文字。

group（0）：回传第一个比对相符的文字。

group（1）：回传括号第一组的文字。

group（2）：回传括号第二组的文字。

下列程序会使用小括号分组的概念，将个分组内容输出。

```
# ch5_2.py
import re

msg = 'Please call my secretary using 02-26669999'
pattern = r'(\d{2})-(\d{8})'
phoneNum = re.search(pattern, msg)            # 传回搜寻结果

print("完整号码是: %s" % phoneNum.group())    # 显示完整号码
print("完整号码是: %s" % phoneNum.group(0))   # 显示完整号码
print("区域号码是: %s" % phoneNum.group(1))   # 显示区域号码
print("电话号码是: %s" % phoneNum.group(2))   # 显示电话号码
```

执行结果

```
================= RESTART: D:\Python interveiw\ch5\ch5_2.py =================
完整号码是: 02-26669999
完整号码是: 02-26669999
区域号码是: 02
电话号码是: 26669999
```

面试实例 ch5_3.py：在 re 模块中，请说明 groups( ) 的功能，同时以程序实例说明。

解答：

当我们使用 re.search( ) 搜寻字符串时，可以使用 groups( ) 方法取得分组的内容，下列程序会分别列出区域号码与电话号码。

```
# ch5_3.py
import re

msg = 'Please call my secretary using 02-26669999'
pattern = r'(\d{2})-(\d{8})'
phoneNum = re.search(pattern, msg)      # 传回搜寻结果
areaNum, localNum = phoneNum.groups()   # 留意是groups()
print("区域号码是: %s" % areaNum)        # 显示区域号码
print("电话号码是: %s" % localNum)       # 显示电话号码
```

执行结果

```
================= RESTART: D:\Python interveiw\ch5\ch5_3.py =================
区域号码是: 02
电话号码是: 26669999
```

延续 groups( ) 的应用，在一般电话号码中，区域号码是用小括号括起来，如下所示：

```
(02)-26669999
```

面试实例 ch5_4.py：请参考面试实例 ch5_3.py，将字符串内的电话号码数据取出。

```
# ch5_4.py
import re

msg = 'Please call my secretary using (02)-26669999'
pattern = r'(\(\d{2}\))-(\d{8})'
phoneNum = re.search(pattern, msg)      # 传回搜寻结果
areaNum, localNum = phoneNum.groups()   # 留意是groups()
print("区域号码是: %s" % areaNum)        # 显示区域号码
print("电话号码是: %s" % localNum)       # 显示电话号码
```

执行结果

```
================= RESTART: D:\Python interveiw\ch5\ch5_4.py =================
区域号码是: (02)
电话号码是: 26669999
```

面试实例 ch5_5.py：搜寻字符串中的 2 个不同的字符串，例如：John 和 Tom，同时以程序实例说明。

解答：

处理这类问题时，可以使用 re 模块的管道概念搜寻多个字符串，此例是 John 和 Tom。

```
# ch5_5.py
import re

msg = 'John and Tom will attend my party tonight. John is my best friend.'
pattern = 'John|Tom'                # 搜寻John和Tom
txt = re.findall(pattern, msg)      # 传回搜寻结果
print(txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_5.py ==================
['John', 'Tom', 'John']
```

面试实例 ch5_6.py：以正则表达式概念，应如何搜寻下列字符串？以程序实例说明。

Johnson

Johnnason

Johnnathan

解答：

可以使用下列正则表达式格式：

```
pattern = 'John(son|nason|nathan)'
```

程序会搜寻 Johnson、Johnnason 或 Johnnathan 任一字符串，然后列出结果，这个程序将列出第一个搜寻比对到的字符串。

```
1  # ch5_6.py
2  import re
3
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = 'John(son|nason|nathan)'
6  txts = re.findall(pattern,msg)      # 传回搜寻结果
7  print(txts)
8  for txt in txts:                    # 将搜寻到内容加上John
9      print('John'+txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_6.py ==================
['son', 'nason', 'nathan']
Johnson
Johnnason
Johnnathan
```

面试实例 ch5_7.py：以正则表达式概念，应如何搜寻下列字符串？以程序实例说明。

John

Johnson

Johnnason

Johnnathan

解答：

可以使用下列正则表达式格式：

```
pattern = 'John(son|nason|nathan)？ '
```

？符号表示可有可无。

程序会搜寻 Johnson、Johnnason 或 Johnnathan 任一字符串，然后列出结果，这个程序将列出第一个搜寻比对到的字符串。

```
# ch5_7.py
import re

msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
pattern = 'John(son|nason|nathan)?'
txts = re.findall(pattern,msg)      # 传回搜寻结果
print(txts)
for txt in txts:                    # 将搜寻到内容加上John
    print('John'+txt)
```

执行结果

```
================ RESTART: D:/Python interveiw/ch5/ch5_7.py ================
['', 'son', 'nason', 'nathan']
John
Johnson
Johnnason
Johnnathan
```

面试实例 ch5_8.py：以正则表达式概念，应如何搜寻下列字符串？以程序实例说明。

Johnson

Johnnason

解答：

可以使用下列正则表达式格式：

```
pattern = 'John((na)？ son)'
```

上述 na 字符串可有可无，下列程序是使用？做搜寻的应用。

```
# ch5_8.py
import re
# 测试1
msg = 'Johnson will attend my party tonight.'
pattern = 'John((na)?son)'
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
# 测试2
msg = 'Johnnason will attend my party tonight.'
pattern = 'John((na)?son)'
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
```

执行结果

```
================ RESTART: D:/Python interveiw/ch5/ch5_8.py ================
Johnson
Johnnason
```

面试实例 ch5_9.py：以正则表达式概念，应如何搜寻下列字符串？以程序实例说明。

Johnson

Johnnason

Johnnanason

解答：

可以使用下列正则表达式格式：

```
pattern = 'John((na)*son)'
```

上述 na 字符串可有可无，下列程序会使用 * 做搜寻的应用。

```
# ch5_9.py
import re
# 测试1
msg = 'Johnson will attend my party tonight.'
pattern = 'John((na)*son)'        # 字符串na可以0到多次
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
# 测试2
msg = 'Johnnason will attend my party tonight.'
pattern = 'John((na)*son)'        # 字符串na可以0到多次
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
# 测试3
msg = 'Johnnananason will attend my party tonight.'
pattern = 'John((na)*son)'        # 字符串na可以0到多次
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_9.py ==================
Johnson
Johnnason
Johnnananason
```

面试实例 ch5_10.py：以正则表达式概念，应如何搜寻下列字符串？以程序实例说明。

```
Johnnason
Johnnanason
```

解答：

可以使用下列正则表达式格式：

```
pattern = 'John((na)+son)'
```

上述 na 字符串可一次到多次，下列程序会使用 + 做搜寻的应用。

```
# ch5_10.py
import re
# 测试1
msg = 'Johnson will attend my party tonight.'
pattern = 'John((na)+son)'        # 字符串na可以1到多次
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt)                        # 请注意是直接打印对象
# 测试2
msg = 'Johnnason will attend my party tonight.'
pattern = 'John((na)+son)'        # 字符串na可以1到多次
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
# 测试3
msg = 'Johnnananason will attend my party tonight.'
pattern = 'John((na)+son)'        # 字符串na可以1到多次
txt = re.search(pattern,msg)      # 传回搜寻结果
print(txt.group())
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_10.py ==================
None
Johnnason
Johnnananason
```

面试实例 ch5_11.py：请说明 Python 搜寻时 { } 的用法，同时以程序实例说明。

解答：

大括号除了可以设定重复次数，也可以设定指定范围，例如：(son) {3，5} 代表所搜寻的字符串可以是‘sonsonson’、‘sonsonsonson’或‘sonsonsonsonson’。(son) {3，5} 正则表达式相当于下列表达式：

((son)(son)(son))|((son)(son)(son)(son))|((son)(son)(son)(son)(son))

下列程序会设定搜寻 son 字符串，重复 3 ～ 5 次皆算搜寻成功。

```
# ch5_11.py
import re

def searchStr(pattern, msg):
    txt = re.search(pattern, msg)
    if txt == None:          # 搜寻失败
        print("搜寻失败 ",txt)
    else:                    # 搜寻成功
        print("搜寻成功 ",txt.group())

msg1 = 'son'
msg2 = 'sonson'
msg3 = 'sonsonson'
msg4 = 'sonsonsonson'
msg5 = 'sonsonsonsonson'
pattern = '(son){3,5}'
searchStr(pattern,msg1)
searchStr(pattern,msg2)
searchStr(pattern,msg3)
searchStr(pattern,msg4)
searchStr(pattern,msg5)
```

执行结果

```
================== RESTART: D:\Python interveiw\ch5\ch5_11.py ==================
搜寻失败  None
搜寻失败  None
搜寻成功  sonsonson
搜寻成功  sonsonsonson
搜寻成功  sonsonsonsonson
```

面试实例 ch5_12.py：请说明 Python 的 re 模块搜寻是贪婪模式还是非贪婪模式？同时以程序实例说明。

解答：

贪婪模式。

下列程序使用搜寻模式‘(son) {3，5}’搜寻字符串‘sonsonsonsonson’。

```
# ch5_12.py
import re

def searchStr(pattern, msg):
    txt = re.search(pattern, msg)
    if txt == None:          # 搜寻失败
        print("搜寻失败 ",txt)
    else:                    # 搜寻成功
        print("搜寻成功 ",txt.group())

msg = 'sonsonsonsonson'
pattern = '(son){3,5}'
searchStr(pattern,msg)
```

执行结果

```
================== RESTART: D:\Python interveiw\ch5\ch5_12.py ==================
搜寻成功  sonsonsonsonson
```

其实由上述程序所设定的搜寻模式可知，3、4 或 5 个 son 重复就算找到了，可是 Python 执行结果是列出重复最多的字符串，即 5 次重复，这是 Python 的默认模式，这种模式又称贪婪（greedy）模式。

面试实例 ch5_13.py：Python 的 re 模块默认搜寻模式是贪婪模式，请说明如何改为非贪婪模式，同时以程序实例说明。

解答：

在正则表达式的搜寻模式最右边加上？符号。

下列程序会以非贪婪模式设计字符串‘sonsonsonsonson’的搜寻。

```
# ch5_13.py
import re

def searchStr(pattern, msg):
    txt = re.search(pattern, msg)
    if txt == None:          # 搜寻失败
        print("搜寻失败 ",txt)
    else:                    # 搜寻成功
        print("搜寻成功 ",txt.group())

msg = 'sonsonsonsonson'
pattern = '(son){3,5}?'      # 非贪婪模式
searchStr(pattern,msg)
```

执行结果

```
================== RESTART: D:\Python interveiw\ch5\ch5_13.py ==================
搜寻成功  sonsonson
```

面试实例 ch5_14.py：单词分离程序，有一段句子如下：

```
John,Johnson,Johnason and Johnnathan will attend my party tonight.
```

将一段英文句子的单词分离，同时将英文单词前 4 个字母是 John 的单词分离。笔者设定如下：

```
pattern = '\w+'          # 意义是不限长度的数字、字母和下画线字符当作符合搜寻
pattern = 'John\w*'      # John 开头后面接 0 或多个数字、字母和下画线字符
```

```
1  # ch5_14.py
2  import re
3  # 测试1将字符串从句子分离
4  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
5  pattern = '\w+'                    # 不限长度的单词
6  txt = re.findall(pattern,msg)      # 传回搜寻结果
7  print(txt)
8  # 测试2将John开始的字符串分离
9  msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
10 pattern = 'John\w*'                # John开头的单词
11 txt = re.findall(pattern,msg)      # 传回搜寻结果
12 print(txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_14.py ==================
['John', 'Johnson', 'Johnnason', 'and', 'Johnnathan', 'will', 'attend', 'my', 'p
arty', 'tonight']
['John', 'Johnson', 'Johnnason', 'Johnnathan']
```

面试实例 ch5_15.py：正则表达式的应用，将下列标题项目从句子中分离。

```
'1 cat,2 dogs,3 pigs,4 swans'
```

下列程序重点是第 5 行。

\d+：表示不限长度的数字。

\s：表示空格。

\w+：表示不限长度的数字、字母和下画线连续字符。

```
1  # ch5_15.py
2  import re
3
4  msg = '1 cat, 2 dogs, 3 pigs, 4 swans'
5  pattern = '\d+\s\w+'
6  txt = re.findall(pattern,msg)      # 传回搜寻结果
7  print(txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_15.py ==================
['1 cat', '2 dogs', '3 pigs', '4 swans']
```

面试实例 ch5_16.py：在正则表达式中，请说明中括号 [ ] 的用途。

解答：

Python 可以使用中括号来设定字符，可参考下列范例。

[a-z]：代表 a ～ z 的小写字符。

[A-Z]：代表 A ～ Z 的大写字符。

[aeiouAEIOU]：代表英文发音的元音字符。

[2-5]：代表 2 ～ 5 的数字。

在字符分类中，中括号内可以不用放上正则表示法的反斜杠 \ 执行，”.、？ 、*、（、）等字符的转义。例如：[2-5.] 会搜寻 2 ～ 5 的数字和句点，这个语法不用写成 [2-5\.]。

搜寻字符的应用，这个程序首先将搜寻 [aeiouAEIOU]，然后将搜寻 [2-5.]。

```
# ch5_16.py
import re
# 测试1搜寻[aeiouAEIOU]字符
msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
pattern = '[aeiouAEIOU]'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
# 测试2搜寻[2-5.]字符
msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
pattern = '[2-5.]'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch5/ch5_16.py =================
['o', 'o', 'o', 'o', 'a', 'o', 'a', 'o', 'a', 'a', 'i', 'a', 'e', 'a', 'o', 'i']
['.', '2', '.', '3', '.', '4', '.']
```

面试实例 ch5_17.py：在正则表达式中，若是在中括号 [ ] 内增加 ^ 字符，请说明此用途，同时以程序实例说明。

解答：

在中括号内的左方加上 ^ 字符，意义是搜寻不在这些字符内的所有字符。

```
# ch5_17.py
import re
# 测试1搜寻不在[aeiouAEIOU]的字符
msg = 'John, Johnson, Johnnason and Johnnathan will attend my party tonight.'
pattern = '[^aeiouAEIOU]'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
# 测试2搜寻不在[2-5.]的字符
msg = '1. cat, 2. dogs, 3. pigs, 4. swans'
pattern = '[^2-5.]'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
```

执行结果

```
================= RESTART: D:/Python interveiw/ch5/ch5_17.py =================
['J', 'h', 'n', ',', ' ', 'J', 'h', 'n', 's', 'n', ',', ' ', 'J', 'h', 'n', 'n',
 's', 'n', ' ', 'n', 'd', ' ', 'J', 'h', 'n', 'n', 't', 'h', 'n', ' ', 'w', 'l',
 'l', ' ', 't', 't', 'n', 'd', ' ', 'm', 'y', ' ', 'p', 'r', 't', 'y', ' ', 't',
 'n', 'g', 'h', 't', '.']
['1', '.', 'c', 'a', 't', ',', ' ', ' ', 'd', 'o', 'g', 's', ',', ' ', ' ', 'p',
 'i', 'g', 's', ',', ' ', ' ', 's', 'w', 'a', 'n', 's']
```

面试实例 ch5_18.py：在正则表达式中，请说明 ^ 放在字符串前面的意义，同时用程序实例解说。

解答：

在正则表示法中起始位置加上 ^ 字符，表示是正则表示法的字符串必须出现在被搜寻字符串的起始位置，如果搜寻成功才算成功。

下列程序是正则表示法 ^ 字符的应用，测试 1 字符串 John 是在最前面，所以可以得到搜寻结果；测试 2 字符串 John 不是在最前面，所以搜寻结果失败回传空字符串。

```
# ch5_18.py
import re
# 测试1搜寻John字符串在最前面
msg = 'John will attend my party tonight.'
pattern = '^John'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
# 测试2搜寻John字符串不是在最前面
msg = 'My best friend is John'
pattern = '^John'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_18.py ==================
['John']
[]
```

面试实例 ch5_19.py：在正则表达式中，请说明 $ 字符的意义，同时用程序实例解说。

解答：

正则表示法的末端放置 $ 字符时，表示正则表示法的字符串必须出现在被搜寻字符串的最后位置，如果搜寻成功才算成功。

下列是正则表示法 $ 字符的应用，测试 1 是搜寻字符串结尾是非英文字符、数字和下画线字符，由于结尾字符是“.”，所以回传所搜寻到的字符。测试 2 是搜寻字符串结尾是非英文字符、数字和下画线字符，由于结尾字符是“8”，所以回传搜寻结果是空字符串。测试 3 是搜寻字符串结尾是数字字符，由于结尾字符是“8”，所以回传搜寻结果是“8”。测试 4 是搜寻字符串结尾是数字字符，由于结尾字符是“.”，所以回传搜寻结果是空字符串。

```
# ch5_19.py
import re
# 测试1搜寻最后字符是非英文字母数字和底线字符
msg = 'John will attend my party 28 tonight.'
pattern = '\W$'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
# 测试2搜寻最后字符是非英文字母数字和下画线字符
msg = 'I am 28'
pattern = '\W$'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
# 测试3搜寻最后字符是数字
msg = 'I am 28'
pattern = '\d$'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
# 测试4搜寻最后字符是数字
msg = 'I am 28 year old.'
pattern = '\d$'
txt = re.findall(pattern,msg)      # 传回搜寻结果
print(txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_19.py ==================
['.']
[]
['8']
[]
```

面试实例 ch5_20.py：在正则表达式中，请说明通配符“.”的功能，同时用程序实例解说。

解答：

通配符（wildcard）“.”表示可以搜寻除了换行字符以外的所有字符，但是只限定一个字符。

下列程序是通配符的应用，搜寻一个通配符加上 at，下列输出中的第 4 个数据由于 at 符合，故 Python 自动加上空格符。第 6 个数据是因为只能加上一个字符，所以搜寻结果是 lat。

```
1  # ch5_20.py
2  import re
3  msg = 'cat hat sat at matter flat'
4  pattern = '.at'
5  txt = re.findall(pattern,msg)      # 传回搜寻结果
6  print(txt)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_20.py ==================
['cat', 'hat', 'sat', ' at', 'mat', 'lat']
```

面试实例 ch5_21.py：在正则表达式中，请说明通配符“.*”的功能，同时用程序实例解说。

解答：

将“.”字符与“*”组合，可以搜寻所有字符，意义是搜寻 0 到多个通配符（换行字符除外）。下列程序是搜寻所有字符“.*”的组合应用。

```
1  # ch5_21.py
2  import re
3
4  msg = 'Name: Jiin-Kwei Hung Address: 8F, Nan-Jing E. Rd, Taipei'
5  pattern = 'Name: (.*) Address: (.*)'
6  txt = re.search(pattern,msg)      # 传回搜寻结果
7  Name, Address = txt.groups()
8  print("Name:    ", Name)
9  print("Address: ", Address)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_21.py ==================
Name:     Jiin-Kwei Hung
Address:  8F, Nan-Jing E. Rd, Taipei
```

面试实例 ch5_22.py：在正则表达式中，请说明 sub( ) 和 subn( ) 的用法，同时用程序实例解说。

解答：

Python re 模块内的 sub( ) 方法可以用新的字符串取代原本字符串的内容，此方法的基本使用语法如下：

```
result = re.sub(pattern,newstr,msg)  # msg 是整个欲处理的字符串或句子
```

pattern 是欲搜寻的字符串，如果搜寻成功则用 newstr 取代，同时成功取代的结果回传给 result 变量。如果搜寻到多笔相同字符串，这些字符串将全部被取代，须留意原先 msg 内容将不会改变。如果搜寻失败则将 msg 内容回传给 result 变量，当然 msg 内容也不会改变。

下列程序是字符串取代的应用，测试 1 是发现 2 个字符串被成功取代（Eli Nan 被 Kevin Thomson 取代），同时列出取代结果。测试 2 是取代失败，所以 txt 与原 msg 内容相同。

```
# ch5_22.py
import re
#测试1取代使用re.sub()结果成功
msg = 'Eli Nan will attend my party tonight. My best friend is Eli Nan'
pattern = 'Eli Nan'                    # 欲搜寻字符串
newstr = 'Kevin Thomson'               # 新字符串
txt = re.sub(pattern,newstr,msg)       # 如果找到则取代
if txt != msg:                         # 如果txt与msg内容不同表示取代成功
    print("取代成功: ", txt)           # 列出成功取代结果
else:
    print("取代失败: ", txt)           # 列出失败取代结果
#测试2取代使用re.sub()结果失败
pattern = 'Eli Thomson'                # 欲搜寻字符串
txt = re.sub(pattern,newstr,msg)       # 如果找到则取代
if txt != msg:                         # 如果txt与msg内容不同表示取代成功
    print("取代成功: ", txt)           # 列出成功取代结果
else:
    print("取代失败: ", txt)           # 列出失败取代结果
```

执行结果

```
================== RESTART: D:\Python interveiw\ch5\ch5_22.py ==================
取代成功:  Kevin Thomson will attend my party tonight. My best friend is Kevin T
homson
取代失败:  Eli Nan will attend my party tonight. My best friend is Eli Nan
```

subn( ) 的用法和 sub( ) 用法相同，不过回传值是元组，此元组内容如下：

```
result = re.subn(pattern,newstr,msg)
```

上述 result 的内容是（newstr，number_of_subs_made），如果取代成功第 1 个参数是新字符串，第 2 个参数是取代次数。如果取代失败第 1 个参数是原字符串，第 2 个参数是 0。

面试实例 ch5_23.py：这是字符串取代的应用，测试 1 是发现 2 个字符串被成功取代（Eli Nan 被 Kevin Thomson 取代），同时列出取代结果和取代次数。测试 2 是取代失败，所以 txt 与原 msg 内容相同。

```
# ch5_23.py
import re
#测试1取代使用re.sub()结果成功
msg = 'Eli Nan will attend my party tonight. My best friend is Eli Nan'
pattern = 'Eli Nan'                    # 欲搜寻字符串
newstr = 'Kevin Thomson'               # 新字符串
txt = re.subn(pattern,newstr,msg)      # 如果找到则取代
if txt != msg:                         # 如果txt与msg内容不同表示取代成功
    print("取代次数: ", txt[1])        # 列出成功取代次数
    print("取代成功: ", txt[0])        # 列出成功取代结果
else:
    print("取代次数: ", txt[1])        # 列出成功取代次数
    print("取代失败: ", txt[0])        # 列出失败取代结果
#测试2取代使用re.sub()结果失败
pattern = 'Eli Thomson'                # 欲搜寻字符串
txt = re.subn(pattern,newstr,msg)      # 如果找到则取代
if txt != msg:                         # 如果txt与msg内容不同表示取代成功
    print("取代次数: ", txt[1])        # 列出成功取代次数
    print("取代成功: ", txt[0])        # 列出成功取代结果
else:
    print("取代次数: ", txt[1])        # 列出成功取代次数
    print("取代失败: ", txt[0])        # 列出失败取代结果
```

执行结果

```
================= RESTART: D:\Python interveiw\ch5\ch5_23.py =================
取代次数:  2
取代成功:  Kevin Thomson will attend my party tonight. My best friend is Kevin T
homson
取代次数:  0
取代成功:  Eli Nan will attend my party tonight. My best friend is Eli Nan
```

面试实例 ch5_24.py：在正则表示法中，请说明如何使用 *** 代替原本的姓名。

解答：

可以使用 re.sub( ) 方法，现在先用程序说明，然后再解析此程序。

下列程序将 CIA 情报员名字用名字首字母和 *** 取代，同时用程序实例解说。

```
# ch5_24.py
import re
# 使用隐藏文字执行取代
msg = 'CIA Mark told CIA Linda that secret USB had given to CIA Peter.'
pattern = r'CIA (\w)\w*'               # 欲搜寻CIA + 空一格后的名字
newstr = r'\1***'                      # 新字符串使用隐藏文字
txt = re.sub(pattern,newstr,msg)       # 执行取代
print("取代成功: ", txt)               # 列出取代结果
```

执行结果

```
================= RESTART: D:/Python interveiw/ch5/ch5_24.py =================
取代成功:  M*** told L*** that secret USB had given to P***.
```

上述程序的关键是第 5 行，这一行将搜寻 CIA 字符串外加空一格后出现不限长度的字符串（可以是英文大小写或数字或下画线所组成）。概念是括号内的（\w）代表必须只有一个字符，同时小括号代表这是一个分组（group），由于整行只有一个括号所以知道这是第一分组，同时只有一个分

组，括号外的 \w* 表示可以有 0 到多个字符。所以（\w）\w* 相当于是一到多个字符组成的单词，同时存在分组 1。

上述程序第 6 行的 **\1** 代表用分组 **1** 找到的第 1 个字母当作字符串开头，后面 ******* 则是接在第 1 个字母后的字符。对 CIA Mark 而言所找到的第一个字母是 M，所以取代的结果是 **M*****。对 CIA Linda 而言所找到的第一个字母是 L，所以取代的结果是 **L*****。对 CIA Peter 而言所找到的第一个字母是 P，所以取代的结果是 **P*****。

面试实例 ch5_25.py：请设计可以判别电子邮件地址的正则表达式，同时使用实例说明。

解答：

```
pattern = r'''(
  [a-zA-Z0-9_.]+                # 使用者账号
  @                             # @符号
  [a-zA-Z0-9-.]+                # 主机域名domain
  [\.]                          # .符号
  [a-zA-Z]{2,4}                 # 可能是com或edu或其他
  ([\.])?                       # .符号，也可能无
  ([a-zA-Z]{2,4})?              # 地区
  )'''
```

第 1 行使用者账号常用的有字符 a ～ z、字符 A ～ Z、数字 0 ～ 9、下画线 _、点 .。第 2 行是 @ 符号。第 3 行是主机域名，常用的有字符 a ～ z、字符 A ～ Z、数字 0 ～ 9、分隔符 -、点 .。第 4 行是点 . 符号。第 5 行最常见的是 com 或 edu，也可能是 cc 或其他，这通常由 2 ～ 4 个字符组成，常用的有字符 a ～ z、字符 A ～ Z。第 6 行是点 . 符号，在美国通常只要有前 5 行就够了，但是在其他国家则常常需要此字段，所以此字段后面是？字符。第 7 行通常是地区，例如：中国是 cn、日本是 ja，常用的有字符 a ～ z、字符 A ～ Z。

下列是电子邮件地址的搜寻。

```
1  # ch5_25.py
2  import re
3
4  msg = '''txt@deepstone.com.tw kkk@gmail.com yyyttt'''
5  pattern = r'''(
6      [a-zA-Z0-9_.]+                # 使用者账号
7      @                             # @符号
8      [a-zA-Z0-9-.]+                # 主机域名domain
9      [\.]                          # .符号
10     [a-zA-Z]{2,4}                 # 可能是com或edu或其他
11     ([\.])?                       # .符号，也可能无
12     ([a-zA-Z]{2,4})?              # 地区
13     )'''
14 eMail = re.findall(pattern, msg, re.VERBOSE)     # 传回搜寻结果
15 for e in eMail:
16     print(e[0])
```

执行结果

```
================= RESTART: D:/Python interveiw/ch5/ch5_25.py =================
txt@deepstone.com.tw
kkk@gmail.com
```

面试实例 ch5_26.py：有一列表内有一系列电子邮件地址，如下所示：

```
emails = ['abcd@me.com',
          'ab@me.com',
          'abc@me.com',
          'defg@me.com.tw',
          'kkkk@kk.edu'
         ]
```

这个程序会找出 @ 前面是 3 ～ 15 个字符，然后结尾是 @me.com 的电子邮件地址。

```
# ch5_26.py
import re
emails = ['abcd@me.com',
          'ab@me.com',
          'abc@me.com',
          'defg@me.com.tw',
          'kkkk@kk.edu'
         ]

for email in emails:
    result = re.match('[\w]{3,15}@me.com$', email)
    if result:
        print('找到了 {}'.format(email))
    else:
        print('不相符 {}'.format(email))
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_26.py ==================
找到了 abcd@me.com
不相符 ab@me.com
找到了 abc@me.com
不相符 defg@me.com.tw
不相符 kkkk@kk.edu
```

上述程序第 11 行的 $ 字符，表示正则表达式必须出现在被搜寻字符串的最后位置。

面试实例 ch5_27.py：用冒号或是空格切割字符串，例如：有一个字符串如下所示。

```
'city:Taipei Taiwan 300'
```

请将上述字符串转成下列列表。

```
['city','Taipei','Taiwan','300']
```

```
# ch5_27.py
import re
string = 'city:Taipei Taiwan 300'
result = re.split(r':| ', string)
print(result)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_27.py ==================
['city', 'Taipei', 'Taiwan', '300']
```

面试实例 ch5_28.py：有一系列电话号码如下：

```
tels = ['0952333344',
        '1111122222',
        '0998833999',
        '12345',
        '0940000111',
        '0912222333',
        '02-29212918'
       ]
```

列出不是 1 与 4 结尾的台湾手机号码，非 10 个数字的号码以及第 1 个数字不是 0 的号码也不是台湾手机号码。

```
# ch5_28.py
import re
tels = ['0952333344',
        '1111122222',
        '0998833999',
        '12345',
        '0940000111',
        '0912222333',
        '02-29212918'
       ]

for t in tels:
    #result = re.match('0\d{9}[0,2-3,5-9]', t)
    result = re.match('0\d{8}[0,2-3,5-9]', t)
    if result:
        print('是符合的台湾手机号码    : {}'.format(t))
    else:
        print('不是符合的台湾手机号码 : {}'.format(t))
```

执行结果

```
================== RESTART: D:\Python interveiw\ch5\ch5_28.py ==================
不是符合的台湾手机号码 : 0952333344
不是符合的台湾手机号码 : 1111122222
是符合的台湾手机号码    : 0998833999
不是符合的台湾手机号码 : 12345
不是符合的台湾手机号码 : 0940000111
是符合的台湾手机号码    : 0912222333
不是符合的台湾手机号码 : 02-29212918
```

面试实例 ch5_29.py：Unicode 基本汉字的编码范围如下：

4E00 - 9FA5

有一个字符串如下：

string = '中文,abc,真好'

请筛选中文字，可以获得下列结果。

['中文','真好']

```
# ch5_29.py
import re

string = '中文, abc, 真好'
pattern = re.compile(r'[\u4E00-\u9FA5]+')
result = pattern.findall(string)
print(result)
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_29.py ==================
['中文', '真好']
```

面试实例 ch5_30.py：有一个列表内容如下：

```
webpage = ['<html><h1>www.abc.com</h1></html>',
           '<html><h2>www.espn.com</h2></html>',
           '<html><h1>www.nbc.com</h2></html>',
           '<html><h2>www.cnn.com</h3></html>'
          ]
```

设计可以筛选下列匹配：

`<html><h1>xxxx</h1></html>`

相当于前面的 <> 和后面的 <> 是对应的。

```
1  # ch5_30.py
2  import re
3
4  webpage = ['<html><h1>www.abc.com</h1></html>',
5             '<html><h2>www.espn.com</h2></html>',
6             '<html><h1>www.nbc.com</h2></html>',
7             '<html><h2>www.cnn.com</h3></html>'
8            ]
9
10 for page in webpage:
11     result = re.match(r'<(\w*)><(\w*)>.*?</\2></\1>', page)
12     if result:
13         print('{} 符合匹配 '.format(page))
14     else:
15         print('{} 不符合匹配 '.format(page))
```

执行结果

```
================== RESTART: D:/Python interveiw/ch5/ch5_30.py ==================
<html><h1>www.abc.com</h1></html> 符合匹配
<html><h2>www.espn.com</h2></html> 符合匹配
<html><h1>www.nbc.com</h2></html> 不符合匹配
<html><h2>www.cnn.com</h3></html> 不符合匹配
```