正则表达式

概述

正则表达式，Regular Expression，缩写为regex、regexp、RE等。

正则表达式是文本处理极为重要的技术，用它可以对字符串按照某种规则进行检索、替换。

1970年代，Unix之父Ken Thompson将正则表达式引入到Unix中文本编辑器ed和grep命令中，由此正则表达式普及开来。

1980年后，perl语言对Henry Spencer编写的库，扩展了很多新的特性。1997年开始，Philip Hazel开发出了PCRE (Perl Compatible Regular Expressions)，它被PHP和HTTPD等工具采用。

正则表达式应用极其广泛，shell中处理文本的命令、各种高级编程语言都支持正则表达式。

参考 https://www.w3cschool.cn/regex_rmjc/

分类

1. BRE
   基本正则表达式，grep、sed、vi等软件支持。vim有扩展。
2. ERE
   扩展正则表达式，egrep（grep -E）、sed -r等。
3. PCRE
   几乎所有高级语言都是PCRE的方言或者变种。Python从1.6开始使用SRE正则表达式引擎，可以认为是PCRE的子集，见模块re。

基本语法

元字符 metacharacter

代码	说明	举例
.	匹配除换行符外任意一个字符	.
[abc]	字符集合，只能表示一个字符位置。 匹配所包含的任意一个字符	[abc]匹配plain中的'a'

[^abc]	字符集合，只能表示一个字符位置。 匹配除去集合内字符的任意一个字符	[^abc]可以匹配plain中的'p'、'l'、'i'或者'n'
[a-z]	字符范围，也是个集合，表示一个字符位置 匹配所包含的任意一个字符	常用[A-Z] [0-9]
[^a-z]	字符范围，也是个集合，表示一个字符位置 匹配除去集合内字符的任意一个字符	
\b	匹配单词的边界	\bb 在文本中找到单词中b开头的b字符
\B	不匹配单词的边界	t\B 包含t的单词但是不以t结尾的t字符，例如write \Bb不以b开头的含有b的单词，例如able
\d	[0-9]匹配1位数字	\d
\D	[^0-9]匹配1位非数字	
\s	匹配1位空白字符，包括换行符、制表符、空格 [\f\r\n\t\v]	
\S	匹配1位非空白字符	
\w	匹配[a-zA-Z0-9_]，包括中文的字	\w
\W	匹配\w之外的字符	

转义

凡是在正则表达式中有特殊意义的符号，如果想使用它的本意，请使用\转义。反斜杠自身，得使用\\

\r、\n还是转义后代表回车、换行

重复

代码	说明	举例
*	表示前面的正则表达式会重复0次或多次	e\w* 单词中e后面可以有非空白字符
+	表示前面的正则表达式重复至少1次	e\w+ 单词中e后面至少有一个非空白字符
	表示前面的正则表达式会重复0次	

?	或1次	e\w? 单词中e后面至多有一个非空白字符
{n}	重复固定的n次	e\w{1} 单词中e后面只能有一个非空白字符
{n,}	重复至少n次	e\w{1,} 等价 e\w+ e\w{0,} 等价 e\w* e\w{0,1} 等价 e\w?
{n,m}	重复n到m次	e\w{1,10} 单词中e后面至少1个，至多10个非空白字符

练习：
1、匹配手机号码
字符串为"手机号码13851888188。"
2、匹配中国座机
字符串为"号码025-83105736、0543-5467328。"

1、\d{11}
2、\d{3,4}-\d{7,8}

代码	说明	举例
x|y	匹配x或者y	wood took foot food 使用 w|food 或者 (w|f)ood
捕获		
(pattern)	使用小括号指定一个子表达式，也叫分组 捕获后会自动分配组号从1开始 可以改变优先级	
\数字	匹配对应的分组	(very) \1 匹配very very，但捕获的组group是very
(?:pattern)	如果仅仅为了改变优先级，就不需要捕获分组	(?:w|f)ood 'industr(?:y|ies)等价 'industry|industries'
(?<name>exp) (?'name'exp)	分组捕获，但是可以通过name访问分组 Python语法必须是(?P<name>exp)	
零宽断言		wood took foot food

(?=exp)	零宽度正预测先行断言 断言exp一定在匹配的右边出现，也就是说断言后面一定跟个exp	f(?=oo) f后面一定有oo出现
(?<=exp)	零宽度正回顾后发断言 断言exp一定出现在匹配的左边出现，也就是说前面一定有个exp前缀	(?<=f)ood、(?<=t)ook分别匹配ood、ook，ook前一定有t出现
负向零宽断言		
(?!exp)	零宽度负预测先行断言 断言exp一定不会出现在右侧，也就是说断言后面一定不是exp	\d{3}(?!\d)匹配3位数字，断言3位数字后面一定不能是数字 foo(?!d)foo后面一定不是d
(?<!exp)	零宽度负回顾后发断言 断言exp一定不能出现在左侧，也就是说断言前面一定不能是exp	(?<!f)ood ood的左边一定不是f
注释		
(?#comment)	注释	f(?=oo)(?#这个后断言不捕获)

注意：
断言会不会捕获呢？也就是断言占不占分组号呢？
断言不占分组号。断言如同条件，只是要求匹配必须满足断言的条件。

分组和捕获是同一个意思
使用正则表达式时，能用简单表达式，就不要复杂的表达式

贪婪与非贪婪
默认是贪婪模式，也就是说尽量多匹配更长的字符串。

非贪婪很简单，在重复的符号后面加上一个?问号，就尽量的少匹配了。

代码	说明	举例
*?	匹配任意次，但尽可能少重复	
+?	匹配至少1次，，但尽可能少重复	
??	匹配0次或1次，，但尽可能少重复	
{n,}?	匹配至少n次，但尽可能少重复	
{n,m}?	匹配至少n次，至多m次，但尽可能少重复	

very very happy 使用**v.*y**和**v.*?y**

引擎选项

代码	说明	Python
IgnoreCase	匹配时忽略大小写	re.I re.IGNORECASE
Singleline	单行模式 . 可以匹配所有字符，包括\n	re.S re.DOTALL
Multiline	多行模式 ^ 行首、$ 行尾	re.M re.MULTILINE
IgnorePatternWhitespace	忽略表达式中的空白字符，如果要使用空白字符用转义，#可以用来做注释	re.X re.VERBOSE

单行模式：
. 可以匹配所有字符，包括换行符。
^ 表示整个字符串的开头，**$**整个字符串的结尾

多行模式：
. 可以匹配除了换行符之外的字符。
^ 表示行首，**$**行尾
^ 表示整个字符串的开始，**$** 表示整个字符串的结尾。开始指的是**\n**后紧接着下一个字符，结束指的是**/n**前的字符

可以认为，单行模式就如同看穿了换行符，所有文本就是一个长长的只有一行的字符串，所有^就是这一行字符串的行首，**$**就是这一行的行尾。
多行模式，无法穿透换行符，^和**$**还是行首行尾的意思，只不过限于每一行

注意：注意字符串中看不见的换行符,**\r\n**会影响**e$**的测试，**e$**只能匹配**e\n**

举例
very very happy
harry key

上面2行happy之后，有可能是\r\n结尾。
y$ 单行匹配key的y，多行匹配happy和key的y。

练习

1、匹配一个0~999之间的任意数字

```
1
12
995
9999
102
02
003
4d
```

2、IP地址

匹配合法的IP地址

```
192.168.1.150
0.0.0.0
255.255.255.255
17.16.52.100
172.16.0.100
400.400.999.888
001.022.003.000
257.257.255.256
```

3、选出含有ftp的链接，且文件类型是gz或者xz的文件名

```
ftp://ftp.astron.com/pub/file/file-5.14.tar.gz
ftp://ftp.gmplib.org/pub/gmp-5.1.2/gmp-5.1.2.tar.xz
ftp://ftp.vim.org/pub/vim/unix/vim-7.3.tar.bz2
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration//iana-et
c/iana-etc-2.30.tar.bz2
http://anduin.linuxfromscratch.org/sources/other/udev-lfs-205-1.tar.bz2
http://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.2.4.tar.gz
http://download.savannah.gnu.org/releases/man-db/man-db-2.6.5.tar.xz
http://download.savannah.gnu.org/releases/sysvinit/sysvinit-2.88dsf.tar.bz2
http://ftp.altlinux.org/pub/people/legion/kbd/kbd-1.15.5.tar.gz
http://mirror.hust.edu.cn/gnu/autoconf/autoconf-2.69.tar.xz
http://mirror.hust.edu.cn/gnu/automake/automake-1.14.tar.xz
```

参考

1、匹配一个0~999之间的任意数字

```
1
12
995
9999
102
02
003
4d
```

\d	1位数
[1-9]?\d	1-2位数
^([1-9]\d\d?|\d)	1-3位数
^([1-9]\d\d?|\d)$	1-3位数的行
^([1-9]\d\d?|\d)\r?$	
^([1-9]\d\d?|\d)(?!\d)	数字开头1-3位数且之后不能是数字

2、IP地址

匹配合法的IP地址

```
192.168.1.150
0.0.0.0
255.255.255.255
17.16.52.100
172.16.0.100
400.400.999.888
001.022.003.000
257.257.255.256
```

((\d{1,3}).){3}(\d{1,3})
(?:(\d{1,3}).){3}(\d{1,3}) # 400.400.999.888

对于ip地址验证的问题

- 可以把数据提出来后，交给IP地址解析库 处理，如果解析异常，就说明有问题，正则的验证只是一个初步的筛选，把明显错误过滤掉。
- 可以使用复杂的正则表达式验证地址正确性
- 前导0是可以的

```
import socket
nw = socket.inet_aton('192.168.05.001')
print(nw, socket.inet_ntoa(nw))
```

分析：
每一段上可以写的数字有1、01、001、000、23、023、230、100，也就说1位就是0-9,2位每一位也是0-9,3位第一位只能0-2，其余2位都可以0-9
(?:([0-2]\d{2}|\d{1,2})\.){3}([0-2]\d{2}|\d{1,2}) # 解决超出200的问题，但是256呢？

200是特殊的，要再单独分情况处理
25[0-5]|2[0-4]\d|[01]?\d\d? 这就是每一段的逻辑
(?:(25[0-5]|2[0-4]\d|[01]?\d\d?)\.){3}(25[0-5]|2[0-4]\d|[01]?\d\d?)

3、选出含有ftp的链接，且文件类型是gz或者xz的文件名

```
ftp://ftp.astron.com/pub/file/file-5.14.tar.gz
ftp://ftp.gmplib.org/pub/gmp-5.1.2/gmp-5.1.2.tar.xz
ftp://ftp.vim.org/pub/vim/unix/vim-7.3.tar.bz2
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration//iana-et
c/iana-etc-2.30.tar.bz2
http://anduin.linuxfromscratch.org/sources/other/udev-lfs-205-1.tar.bz2
http://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.2.4.tar.gz
http://download.savannah.gnu.org/releases/man-db/man-db-2.6.5.tar.xz
http://download.savannah.gnu.org/releases/sysvinit/sysvinit-2.88dsf.tar.bz2
http://ftp.altlinux.org/pub/people/legion/kbd/kbd-1.15.5.tar.gz
http://mirror.hust.edu.cn/gnu/autoconf/autoconf-2.69.tar.xz
http://mirror.hust.edu.cn/gnu/automake/automake-1.14.tar.xz
```

```
.*ftp.*\.(?:gz|xz) #
ftp.*/(.*(?:gz|xz))
.*ftp.*/([^/]*\.(?:gz|xz)) # 捕获文件名分组
(?<=.*ftp.*/)[^/]*\.(?:gz|xz) # 断言文件名前一定还有ftp
```

Python的正则表达式

Python使用re模块提供了正则表达式处理的能力。

常量

常量	说明
re.M re.MULTILINE	多行模式
re.S re.DOTALL	单行模式
re.I re.IGNORECASE	忽略大小写
re.X re.VERBOSE	忽略表达式中的空白字符

使用 `| 位或` 运算开启多种选项

方法

编译

re.compile(pattern, flags=0)
设定flags，编译模式，返回正则表达式对象regex。
pattern就是正则表达式字符串，flags是选项。正则表达式需要被编译，为了提高效率，这些编译后的结果被保存，下次使用同样的pattern的时候，就不需要再次编译。

re的其它方法为了提高效率都调用了编译方法，就是为了提速。

单次匹配

re.match(pattern, string, flags=0)
regex.match(string[, pos[, endpos]])
match匹配从字符串的开头匹配，regex对象match方法可以重设定开始位置和结束位置。返回match对象

re.search(pattern, string, flags=0)
regex.search(string[, pos[, endpos]])
从头搜索直到第一个匹配，regex对象search方法可以重设定开始位置和结束位置，返回match对象

re.fullmatch(pattern, string, flags=0)

regex.fullmatch(string[, pos[, endpos]])
整个字符串和正则表达式匹配

```
import re

s = '''bottle\nbag\nbig\napple'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%8==0 else ' ')
print()

(0, 'b') (1, 'o') (2, 't') (3, 't') (4, 'l') (5, 'e') (6, '\n') (7, 'b')
(8, 'a') (9, 'g') (10, '\n') (11, 'b') (12, 'i') (13, 'g') (14, '\n') (15, 'a')
(16, 'p') (17, 'p') (18, 'l') (19, 'e')

# match方法
print('--match--')
result = re.match('b', s) # 找到一个就不找了
print(1, result) # bottle
result = re.match('a', s) # 没找到，返回None
print(2, result)
result = re.match('^a', s, re.M) # 依然从头开始找，多行模式没有用
print(3, result)
result = re.match('^a', s, re.S) # 依然从头开始找
print(4, result)
# 先编译，然后使用正则表达式对象
regex = re.compile('a')
result = regex.match(s) # 依然从头开始找
print(5, result)
result = regex.match(s, 15) # 把索引15作为开始找
print(6, result) # apple
print()

# search方法
print('--search--')
result = re.search('a', s) # 扫描找到匹配的第一个位置
print(7, result) # apple
regex = re.compile('b')
result = regex.search(s, 1)
print(8, result) # bag
regex = re.compile('^b', re.M)
```

```
result = regex.search(s) # 不管是不是多行，找到就返回
print(8.5, result) # bottle
result = regex.search(s, 8)
print(9, result) # big

# fullmatch方法
result = re.fullmatch('bag', s)
print(10, result)
regex = re.compile('bag')
result = regex.fullmatch(s)
print(11, result)
result = regex.fullmatch(s, 7)
print(12, result)
result = regex.fullmatch(s, 7, 10)
print(13, result) # 要完全匹配，多了少了都不行，[7, 10)
```

全文搜索

re.findall(pattern, string, flags=0)
regex.findall(string[, pos[, endpos]])
对整个字符串，从左至右匹配，返回所有匹配项的列表

re.finditer(pattern, string, flags=0)
regex.finditer(string[, pos[, endpos]])
对整个字符串，从左至右匹配，返回所有匹配项，返回迭代器。
注意每次迭代返回的是match对象。

```
import re

s = '''bottle\nbag\nbig\nable'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%8==0 else ' ')
print()

(0, 'b') (1, 'o') (2, 't') (3, 't') (4, 'l') (5, 'e') (6, '\n') (7, 'b')
(8, 'a') (9, 'g') (10, '\n') (11, 'b') (12, 'i') (13, 'g') (14, '\n') (15, 'a')
(16, 'b') (17, 'l') (18, 'e')

# findall方法
```

```
result = re.findall('b', s)
print(1, result)
regex = re.compile('^b')
result = regex.findall(s)
print(2, result)
regex = re.compile('^b', re.M)
result = regex.findall(s, 7)
print(3, result) # bag big
regex = re.compile('^b', re.S)
result = regex.findall(s)
print(4, result) # bottle
regex = re.compile('^b', re.M)
result = regex.findall(s, 7, 10)
print(5, result) # bag

# finditer方法
result = regex.finditer(s)
print(type(result))
print(next(result))
print(next(result))
```

匹配替换

re.sub(pattern, replacement, string, count=0, flags=0)
regex.sub(replacement, string, count=0)
使用pattern对字符串string进行匹配，对匹配项使用repl替换。
replacement可以是string、bytes、function。

re.subn(pattern, replacement, string, count=0, flags=0)
regex.subn(replacement, string, count=0)
同sub返回一个元组（new_string，number_of_subs_made）

```
import re

s = '''bottle\nbag\nbig\napple'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%8==0 else ' ')
print()

(0, 'b') (1, 'o') (2, 't') (3, 't') (4, 'l') (5, 'e') (6, '\n') (7, 'b')
(8, 'a') (9, 'g') (10, '\n') (11, 'b') (12, 'i') (13, 'g') (14, '\n') (15, 'a')
```

```
(16, 'p') (17, 'p') (18, 'l') (19, 'e')

# 替换方法
regex = re.compile('b\wg')
result = regex.sub('magedu', s)
print(1, result) # 被替换后的字符串
result = regex.sub('magedu', s, 1) # 替换1次
print(2, result) # 被替换后的字符串

regex = re.compile('\s+')
result = regex.subn('\t', s)
print(3, result) # 被替换后的字符串及替换次数的元组
```

分割字符串

字符串的分割函数，太难用，不能指定多个字符进行分割。

re.split(pattern, string, maxsplit=0, flags=0)
re.split分割字符串

```
import re

s = '''01 bottle
02 bag
03      big1
100       able'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%8==0 else ' ')
print()

# 把每行单词提取出来
print(s.split())  # 做不到['01', 'bottle', '02', 'bag', '03', 'big1', '100', 'able'
]
result = re.split('[\s\d]+', s)
print(1, result) # ['', 'bottle', 'bag', 'big', 'able']
regex = re.compile('^[\s\d]+') # 字符串首
result = regex.split(s)
print(2, result) # ['', 'bottle\n02 bag\n03      big1\n100       able']
regex = re.compile('^[\s\d]+', re.M) # 行首
result = regex.split(s)
print(3, result) # ['', 'bottle\n', 'bag\n', 'big1\n', 'able']
```

```
regex = re.compile('\s+\d+\s+')
result = regex.split(' ' + s)
print(4, result)
```

分组

使用小括号的pattern捕获的数据被放到了组group中。

match、search函数可以返回match对象；findall返回字符串列表；finditer返回一个个match对象

如果pattern中使用了分组，如果有匹配的结果，会在match对象中
1、使用group(N)方式返回对应分组，1-N是对应的分组，0返回整个匹配的字符串
2、如果使用了命名分组，可以使用group('name')的方式取分组
3、也可以使用groups()返回所有组
4、使用groupdict() 返回所有命名的分组

```
import re

s = '''bottle\nbag\nbig\napple'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%8==0 else ' ')
print()

# 分组
regex = re.compile('(b\w+)')
result = regex.match(s)
print(type(result))
print(1, 'match', result.groups())

result = regex.search(s, 1)
print(2, 'search', result.groups()) #

# 命名分组
regex = re.compile('(b\w+)\n(?P<name2>b\w+)\n(?P<name3>b\w+)')
result = regex.match(s)
print(3, 'match', result)
print(4, result.group(3), result.group(2), result.group(1))
print(5, result.group(0).encode()) # 0 返回整个匹配字符串
print(6, result.group('name2'), result.group('name3'))
print(6, result.groups())
```

```
print(7, result.groupdict())

result = regex.findall(s)
for x in result: # 字符串列表
    print(type(x), x)

regex = re.compile('(?P<head>b\w+)')
result = regex.finditer(s)
for x in result:
    print(type(x), x, x.group(), x.group('head'))
```

练习

匹配邮箱地址

```
test@hot-mail.com
v-ip@magedu.com
web.manager@magedu.com.cn
super.user@google.com
a@w-a-com
```

匹配html标记内的内容

```
<a href='http://www.magedu.com/index.html' target='_blank'>马哥教育</a>
```

匹配URL

```
http://www.magedu.com/index.html
https://login.magedu.com
file:///ect/sysconfig/network
```

匹配二代中国身份证ID

```
321105700101003
321105197001010030
11210020170101054X
17位数字+1位校验码组成
前6位地址码，8位出生年月，3位数字，1位校验位（0-9或X）
```

判断密码强弱

要求密码必须由 **10-15**位 指定字符组成：
十进制数字
大写字母
小写字母
下划线
要求四种类型的字符都要出现才算合法的强密码

例如：**Aatb32_67mnq**，其中包含大写字母、小写字母、数字和下划线，是合格的强密码

单词统计word count

对**sample**文件进行单词统计，要求使用正则表达式

参考

邮箱
\w+[-.\w]*@[\w-]+(\.[\w-]+)+

html提取
<[^<>]+>(.*)<[^<>]+>
如果要匹配标记**a**
<(\w+)\s+[^<>]+>(.*)(</\1>)

URL提取
(\w+)://([^\s]+)

身份证验证
身份证验证需要使用公式计算，最严格的应该实名验证。
\d{17}[0-9xX]|\d{15}

强密码

Aatb32_67mnq
Aatb32_67m.nq
中国是一个伟大的国家**aA_8**

10-15位，其中包含大写字母、小写字母、数字和下划线
^\w{10,15}$
如果测试有不可见字符干扰使用**^\w{10,15}\r?$**
看似正确，但是，如果密码有中文呢？
^[a-zA-Z0-9_]{10,15}$

但是还是没有解决类似于 **111111111112** 这种密码的问题，如何解决？
需要用到一些非正则表达式的手段。
利用判断来解决，思路如下：
1、可以判断当前密码字符串中是否有**\W**，如果出现就说明一定不是合法的，如果不出现说明合法
2、对合法继续判断，如果出现过**_**下划线，说明有可能是强密码，但是没有下划线说明一定不是强密码。
3、对包含下划线的合法密码字符串继续判断，如果出现过**\d**的，说明有可能是强密码，没有出现**\d**的一定不是强密码
4、对上一次的包含下划线、数字的合法的密码字符串继续判断，如果出现了**[A-Z]**说明有可能是强密码，没有出现**[A-Z]**说明一定不是强密码
5、对上一次包含下划线、数字、大写字母的合法密码字符串继续判断，如果出现了**[a-z]**说明就是强密码，找到了，没有出现小写字母就一定不是强密码。
请注意上面的判断的顺序，应该是概率上最可能不出在密码字符串的先判断。

单词统计

```
from collections import defaultdict
import re


def makekey2(line:str, chars=set("""!'"#./\()[],*- \r\n""")):
    start = 0

    for i, c in enumerate(line):
        if c in chars:
            if start == i: # 如果紧挨着还是特殊字符，start一定等于i
                start += 1 # 加1并continue
                continue
            yield line[start:i]
            start = i + 1 # 加1是跳过这个不需要的特殊字符c
    else:
        if start < len(line): # 小于，说明还有有效的字符，而且一直到末尾
            yield line[start:]
```

```
# '''\\host\mount splitext('.cshrc splitdrive("//host/computer/dir . abc path`s\r\n
'''
regex = re.compile('[^\w-]+')

def makekey3(line:str):
    for word in regex.split(line):
        if len(word):
            yield word

def wordcount(filename, encoding='utf8', ignore=set()):
    d = defaultdict(lambda:0)
    with open(filename, encoding=encoding) as f:
        for line in f:
            for word in map(str.lower, makekey2(line)):
                if word not in ignore:
                    d[word] += 1
    return d

def top(d:dict, n=10):
    for i, (k,v) in enumerate(sorted(d.items(), key=lambda item: item[1], reverse=T
rue)):
        if i > n:
            break
        print(k,v)

# 单词统计前几名
top(wordcount('sample', ignore={'the', 'a'}))
```