正则表达式练习 ¶

1. 正则表达式简介

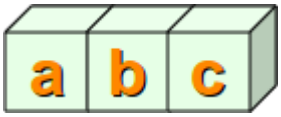
一个**正则表达式**指定了一种**模式**，或者说一种规则，某个字符串或字符串中的一部分只要符合这个模式，那么这个字符串就匹配上了。白话一点说就是正则表达式帮助我们对字符串进行查询匹配或是修改。

正则表达式可以:

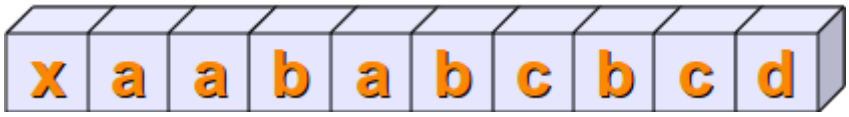
- 测试字符串中的模式；（*例如：测试输入的字符串是否为电话号码*)
- 通过模式匹配从字符串中查找、提取、替换和删除子字符串。

为了帮助大家更形象地了解正则表达式对于字符串的匹配功能，下面通过图片解释来一步一步地解释字符串匹配理念：

我们想查询一下字符串的一部分 string sub = "abc"

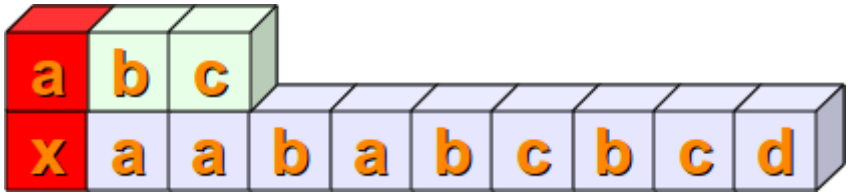


是否存在于字符串 s = "xaababcbcd" 中：

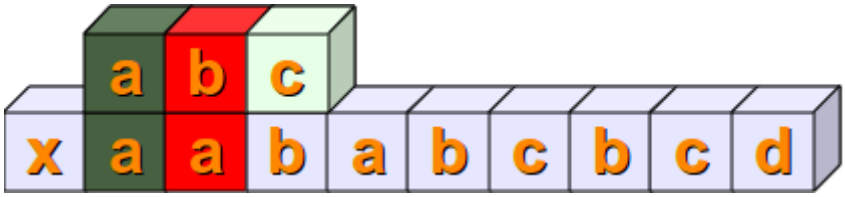


字符串 "abc" 可以被视作一个非常简单的正则表达式.

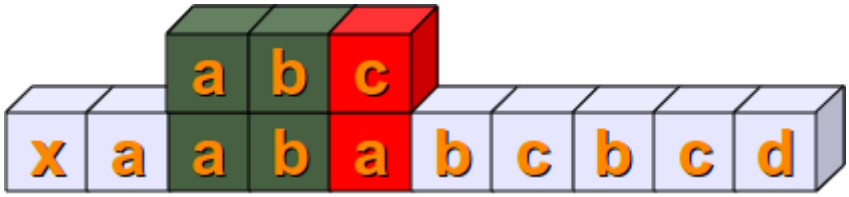
在第一个位置上，我们匹配一下看看，两个字符串是否在第一个位置上匹配（s[0] == sub[0]），不匹配我们标记为红色：



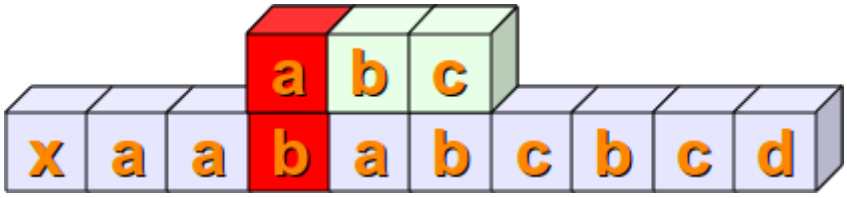
接下来，我们匹配一下s[1:4] == sub.如果sub[0]和字符串s[1]匹配，那么我们将其标记为绿色.接着我们继续匹配其他的位置，但是字符串 s[2] 与 sub[1]不匹配,我们就停止在这个位置的匹配工作



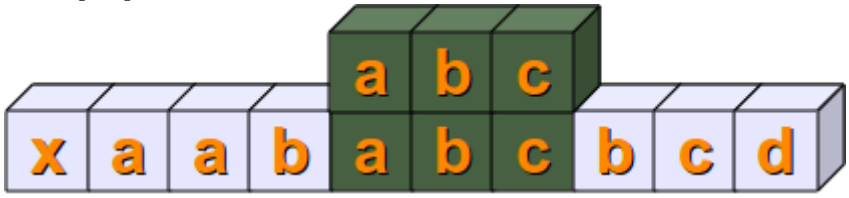
现在，我们匹配一下字符串 s[2:5] 与我的sub是否匹配. 前两个位置是匹配的，但是第三个不匹配:



继续向下一个位置移动，还是不匹配：



直到最后，我们在字符串 s[4:7]找到与sub匹配 :



2. Python 中正则表达式语法简介

正则表达式通常由**普通字符**和**特殊字符（元字符）**组成。

2.1正则表达式语法简介：

字符种类		字符	语法说明	正则表达式例子	匹配字符串
特殊字符		.	匹配出换行符外的任意字符	a.c	abc
	定位符	**^**	**匹配字符串的开头，多行模式中匹配每一行的开头**	**^foo**	**foo**
		$	匹配字符串的结尾，多行模式匹配每一行的结尾	foo$	foo
		\A	**匹配字符串的开头**	**\Aabc**	**abc**
		\Z	匹配字符串的结尾	ple\Z	ple
		\b	**匹配字边界，即字与空格之间的位置**	**\bLucy\b**	**Lucy**
		\B	匹配非字边界	\Bpp\B	pp
	限定符(限定数量)	*****	**匹配前面一个字符的0或无限多次**	**zo***	**z、zoo**
		+	匹配前面一个字符的1或无限多次	zo+	zoo、zooo
		?	**匹配前面一个字符0或1次**	**zo?**	**z、zo**
		{m}	匹配前面一个字符m次	1{2}	11
		{m,n}	**匹配前面一个字符m至n次**	**1{2,3}**	**11、111**
		{,n}	匹配前面一个字符0至n次	1{,2}	1、11
		{m,}	**匹配前面一个字符m至无限次**	**1{2,}**	**11、111**
		*?、+?、{m,n}?、{,n}?、{m,}?	使*、+、{m,n}、{,n}、{m,}尽可能少地匹配	ab*?	ab
	转义字符	****	**匹配的字符串中有特殊字符，你需要在特殊字符前加上转义符\，告诉python你是想匹配这个字符，并不是像使用它的特殊用途。**	****	-
	分枝条件	|	多个条件用|分开，满足任意一种条件均属匹配，从左到右匹配模式，匹配成功即停止	apple|banana	apple、banana
	字符集合	**[abc]**	**字符逐个列出，匹配字符集中任意一个字符,特殊字符在字符集中都会失去其特殊含义，除预定义字符集外**	**[abc]**	**a、b、c**
		[a-z]	两个字符间用-分隔，表示匹配字符范围内任意一个字符	[a-z]	b
		[^xyz]	**匹配未列出的任意字符**	**[^abc]**	**on**
		[^a-z]	匹配任何不在指定范围内的任意字符	[^a-z]	1
	分组	**(_)**	**匹配pattern并获取这一匹配的子字符串，该子字符串用于向后引用**	**gr(a|e)y**	**gray、grey**
		(?P<name>...)	创建分组，并指定别名	(?<last_name>Sam)	Sam
		(?aiLmsux)	**指定正则表达式中其中一部分flags参数**	**(?i)sam**	**sam、Sam**
		(?:...)	匹配模式但不获取匹配的子字符串，匹配的子字符串不能向后引用，用于使用|或后接数量词	(?:Wow!){2}	Wow!Wow!
		(?#_)	**注释**	**(?#这是一条注释)**	
		(?=...)	匹配模式，获取前面的子字符串	(?=a)	a前面的子字符串

		(?!_)	**获取不匹配该模式的前面的子字符串**	**(?!a)**	**不是a前面的子字符串**
		(?<=...)	匹配模式，获取后面的子字符串	(?<=a)	a后面的子字符串
		(?<!_)	**获取不匹配该模式的之后的子字符串**	**(?!=a)**	**不是a后面的子字符串**
		(?(id/name)yes-pattern|no-pattern)	如果编号为id/别名为name的分组匹配到字符，则匹配yes-pattern，如果没有匹配到字符，则匹配no-pattern	(\d)abc(?(1)\d|abc)	1abc1、abcabc
	向后引用	**\<number>**	**引用编号为<number>的分组匹配到的子字符串**	**\dabc\1**	**1abc1**
		(?P=name)	引用别名为name的分组匹配到的子字符串	(?<number>\d)abc(?P=number)	1abc1
		\d	**匹配数字0-9，相当于[0-9]**	**a\db**	**a1b**

2.2正则表达式语法例子

字符种类	字符	语法说明	正则表达式例子	匹配字符串
普通字符	26个字母(包括大小写)以及数字	匹配本身	a	a

In [7]:

```
# 找到句子中的单词dog
p = re.search(r'dog', 'This is a dog!')
print (p.group())
```

dog

字符种类	字符	语法说明	正则表达式例子	匹配字符串
	.	匹配出换行符外的任意字符	a.c	abc

In [8]:

```
# 匹配两个w中间有任意字符的三个字符
print (re.search(r'w.w', 'wow').group())
```

wow

字符种类	字符	语法说明	正则表达式例子	匹配字符串
定位符	^	匹配字符串的开头，多行模式中匹配每一行的开头	^foo	foo
	$	匹配字符串的结尾，多行模式匹配每一行的结尾	foo$	foo
	\A	匹配字符串的开头	\Aabc	abc
	\Z	匹配字符串的结尾	ple\Z	ple
	\b	匹配字边界，即字与空格之间的位置	\bLucy\b	Lucy
	\B	匹配非字边界	\Bpp\B	pp

In [9]:

```
# 匹配字符串开头的数字
print ('特殊字符^ :', re.search(r'^\d', '2018/05/21').group())
print ('特殊字符\A :', re.search(r'\A\d', '2018/05/21').group())

# 匹配字符串结尾的数字
print ('特殊字符$ :', re.search(r'\d$', '2018/05/21').group())
print ('特殊字符\Z :', re.search(r'\d\Z', '2018/05/21').group())

# 匹配单词apple
print ('特殊字符\\b :', re.search(r'\bapple\b', 'apple-pie apple').group())

# 匹配不是在单词开头或结尾的pp
print ('特殊字符\B :', re.search(r'\Bpp\B', 'apple-pie').group())
```

```
特殊字符^ : 2
特殊字符\A : 2
特殊字符$ : 1
特殊字符\Z : 1
特殊字符\b : apple
特殊字符\B : pp
```

字符种类		字符	语法说明	正则表达式例子	匹配字符串
	限定符(限定数量)	*	匹配前面一个字符的0或无限多次	zo*	z、zoo
		+	匹配前面一个字符的1或无限多次	zo+	zoo、zooo
		?	匹配前面一个字符0或1次	zo?	z、zo
		{m}	匹配前面一个字符m次	1{2}	11
		{m,n}	匹配前面一个字符m至n次	1{2,3}	11、111
		{,n}	匹配前面一个字符0至n次	1{,2}	1、11
		{m,}	匹配前面一个字符m至无限次	1{2,}	11、111
		*?、+?、{m,n}?、{,n}?、{m,}?	使*、+、{m,n}、{,n}、{m,}尽可能少地匹配	ab*?	ab

In [10]:

```
text = 'Hello World!'
# e后面有0个或以上的l
print (re.search(r'el*', text).group())

# e后有0个或1个l
print (re.search(r'el?', text).group())

# e后面有1个或以上的l
print (re.search(r'el+', text).group())

# e后面有1个或2个的l
print (re.search(r'el{1,2}', text).group())

# 懒惰匹配
print (re.search(r'el*?', text).group())
print (re.search(r'el+?', text).group())
```

```
ell
el
ell
ell
e
el
```

字符种类	字符	语法说明	正则表达式例子	匹配字符串
转义字符	\	匹配的字符串中有特殊字符，你需要在特殊字符前加上转义符\，告诉python你是想匹配这个字符，并不是像使用它的特殊用途。	\.	.

In [11]:

```
# 找出句子中的句号
print (re.search(r'\.', 'Nice to meet you. How do you do for living?').group())
```

```
.
```

字符种类	字符	语法说明	正则表达式例子	匹配字符串
分枝条件	|	多个条件用|分开，满足任意一种条件均属匹配，从左到右匹配模式，匹配成功即停止	apple|banana	apple、banana

In [12]:

```
# 分枝条件
print (re.search(r'center|centre', ('center')).group())
```

```
center
```

字符种类	字符	语法说明	正则表达式例子	匹配字符串
字符集合	[abc]	字符逐个列出，匹配字符集中任意一个字符,特殊字符在字符集中都会失去其特殊含义，除预定义字符集外	[abc]	a、b、c
	[a-z]	两个字符间用-分隔，表示匹配字符范围内任意一个字符	[a-z]	b
	[^xyz]	匹配未列出的任意字符	[^abc]	on
	[^a-z]	匹配任何不在指定范围内的任意字符	[^a-z]	1

In [13]:

```
# 匹配指定字符集的任一字符
print (re.search(r'[a-z]+', 'The phone is 8889999.').group())

# 匹配不包含在指定字符集的任一字符
print (re.search(r'[^a-z]+', 'The phone is 8889999.').group())
```

```
he
T
```

字符种类	字符	语法说明	正则表达式例子	匹配字符串
特殊字符	(...)	匹配pattern并获取这一匹配的子字符串，该子字符串用于向后引用	gr(a|e)y	gray、grey
分组	(?P<name>...)	创建分组，并指定别名	(?<last_name>Sam)	Sam
	(?aiLmsux)	指定正则表达式中其中一部分flags参数	(?i)sam	sam、Sam
	(?:...)	匹配模式但不获取匹配的子字符串，匹配的子字符串不能向后引用，用于使用|或后接数量词	(?:Wow!){2}	Wow!Wow!
	(?#...)	注释	(?#这是一条注释)	
	(?=...)	匹配模式，获取前面的子字符串	(?=a)	a前面的子字符串
	(?!...)	获取不匹配该模式的前面的子字符串	(?!a)	不是a前面的子字符串
	(?<=...)	匹配模式，获取后面的子字符串	(?<=a)	a后面的子字符串
	(?<!...)	获取不匹配该模式的之后的子字符串	(?!=a)	不是a后面的子字符串
	(?(id/name)yes-pattern|no-pattern)	如果编号为id/别名为name的分组匹配到字符，则匹配yes-pattern，如果没有匹配到字符，则匹配no-pattern	(\d)abc(?(1)\d|abc)	1abc1、abcabc

In [14]:

```
print (re.search(r'(Wow!)\1\1', 'Wow!Wow!Wow!').group())

# 匹配名字
print (re.search(r'(?P<first_name>\w+)', 'Malcolm Reynolds').group())

# 匹配sam,忽略大小写
print (re.findall(r'(?i)sam', 'sam Sam'))

# 添加注释
print (re.search(r'(?#匹配名字)\w+ \w+', 'Malcolm Reynolds').group())

# 匹配def前的abc
print (re.search('abc(?=def)', 'abcdef').group())

# 匹配不在def前的abc
print (re.search('abc(?!def)', 'abcdef'))

# 匹配abc后的def
print (re.search('(?<=abc)def', 'abcdef').group())

# 匹配不在abc后的def
print (re.search('(?<!abc)def', 'abcdef'))

# 匹配'数字abc'后面是数字或abc的字符串
print (re.search(r'(\d)abc(?(1)\d|abc)', '1abc1 abcabc').group())
```

```
Wow!Wow!Wow!
Malcolm
['sam', 'Sam']
Malcolm Reynolds
abc
None
def
None
1abc1
```

字符种类	字符	语法说明	正则表达式例子	匹配字符串
向后引用	\<number>	引用编号为<number>的分组匹配到的子字符串	\dabc\1	1abc1
	(?P=name)	引用别名为name的分组匹配到的子字符串	(?<number>\d)abc(?P=number)	1abc1

In [15]:

```
print (re.search(r'(?P<number>\d)abc(?P=number)', '1abc1').group())
```

```
1abc1
```

字符种类		字符	语法说明	正则表达式例子	匹配字符串
特殊字符	预定义字符	\d	匹配数字0-9，相当于[0-9]	a\db	a1b
		\D	匹配除数字0-9之外的字符，相当于[^0-9]	a\Db	aab
		\s	匹配空格，相当于[\t\n\r\f\v]	Hello\sworld	Hello world
		\S	匹配除空格外的任意字符，相当于[^ \t\n\r\f\v]	Hello\S	Hello!
		\w	[a-zA-Z0-9_]	h\w	he
		\W	匹配除数字、大小写字母和下划线的任意字符，相当于[^a-zA-Z0-9_]	Hi\W	Hi!

In [16]:

```
# 找到电话号码
print (re.search(r'\d+', 'My phone is 1984924898.Please call me.').group())

# 匹配电话号码前的字符串
print (re.search(r'\D+', 'My phone is 1984924898.Please call me.').group())

# 匹配My phone
print (re.search(r'My\sphone', 'My phone is 1984924898.Please call me.').group())

# 匹配me.
print (re.search(r'me\S', 'My phone is 1984924898.Please call me.').group())

# 匹配第一个词
print (re.search(r'\w+', 'My phone is 1984924898.Please call me.').group())

# 匹配句号
print (re.search(r'\W$', 'My phone is 1984924898.Please call me.').group())
```

```
1984924898
My phone is
My phone
me.
My
.
```

3. Python 中re模块函数

在Python中想应用正则表达式，我们需要引入 **re 模块。**

Python通过re模块提供对正则表达式的支持。

使用re的一般步骤是

- 1.将正则表达式的字符串形式编译为Pattern实例
- 2.使用Pattern实例处理文本并获得匹配结果（一个Match实例）
- 3.使用Match实例获得信息，进行其他的操作。

In [17]:

```
# encoding: UTF-8
import re  # 引入Python中正则表达式模块 — re 模块。

# 将正则表达式编译成Pattern对象
pattern = re.compile(r'hello.*!')

# 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
match = pattern.match('hello, hanxiaoyang! How are you?')

if match:
    # 使用Match获得分组信息
    print (match.group())
```

```
hello, hanxiaoyang!
```

- **re模块** 中提供很多函数，这些函数和方法可以帮助你使用正则表达式对字符串进行匹配、提取和替换等操作，你也可以将正则表达式编译为对象后，通过对象的方法对字符串进行相同的操作。

以下为将要介绍的re模块里的函数和方法：

函数	功能	函数	功能
re.compile ( )	将字符串形式的正则表达式编译为Pattern对象	**re.match ( )**	匹配字符串开头
re.search ( )	从左到右扫描字符串，返回对应第一个匹配对象	**re.finall ( )**	匹配的子字符串以列表形式返回
re.finditer ( )	返回匹配访问顺序	**re.split ( )**	按照能够匹配的子串将string分割后返回列表
re.sub ( )	输出修改替换后的字符串	**re.subn ( )**	返回一个元组（新字符串，替换次数)
re.escape ( )	转义除ASCII字母、数字和下划线的所有字符	**re.purge ( )**	清除所有正则表达式缓存

- **函数句法**

【例】 **re.match(pattern, string, flags=0)**

pattern：需要匹配的正则表达式

string：需要被匹配的字符串

flags：匹配模式，取值可以使用按位或运算符'|'表示同时生效，比如 re.compile('pattern', re.I | re.M)

flags	功能
re.I(re.IGNORECASE)	忽略大小写（括号内是完整写法，下同）
re.M(MULTILINE)	多行模式，'^'和'$'匹配每行的开头和结尾
re.S(DOTALL)	点任意匹配模式，'.'匹配任意字符，包括换行符
re.L(LOCALE)	使预定字符类 \w \W \b \B \s \S 取决于当前区域设定
re.U(UNICODE)	使预定字符类 \w \W \b \B \s \S \d \D 取决于unicode定义的字符属性
re.X(VERBOSE)	详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释
re.A(ASCII)	在unicode字符串模式的情况下影响\w, \W, \b, \B, \d, \D, \s, \S，让它们只匹配ASCII码,在bytes字符串模式下会被忽略

3.1. 函数 re.compile ( )

re.compile(pattern,flags = 0) 将正则表达式的字符串形式编译成正则表达式对象，该对象可以使用match()等方法进行操作。

【例 3.1】将带字母o的字符串编译成一个正则对象

In [18]:

```
import re
test="Hi, nice to meet you where are you from?" #需要被匹配的字符串
k=re.compile(r'\w*o\w*')     #将匹配带o的字符串编译成一个正则对象
print (k.findall(test))      #显示所有包含o的字符串
```

['to', 'you', 'you', 'from']

3.2. 函数 re.match ( )

re.match(pattern, string, flags=0) 如果字符串开头处有匹配模式的子字符串，则返回匹配对象。即使在多行模式下，也只匹配字符串开头，而非每行开头

re.match( )函数的属性：

- string: 匹配时使用的文本。
- re: 匹配时使用的Pattern对象。
- pos: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
- endpos: 文本中正则表达式结束搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
- lastindex: 最后一个被捕获的分组的索引。如果没有被捕获的分组，将为None。
- lastgroup: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为None。

group( ) 格式	功能
group([group1, ...])	获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。group1可以使用编号也可以使用别名；编号0代表整个匹配的子串；不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。
groups([default])	以元组形式返回全部分组截获的字符串。相当于调用group(1,2,…last)。default表示没有截获字符串的组以这个值替代，默认为None。
groupdict([default])	返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。default含义同上。
start([group])	返回指定的组截获的子串在string中的起始索引（子串第一个字符的索引）。group默认值为0。
end([group])	返回指定的组截获的子串在string中的结束索引（子串最后一个字符的索引+1）。group默认值为0。
span([group])	返回(start(group), end(group))。
expand(template)	将匹配到的分组代入template中然后返回。template中可以使用\id或\g、\g引用分组，但不能使用编号0。\id与\g是等价的；但\10将被认为是第10个分组，如果你想表达\1之后是字符'0'，只能使用\g<1>0。

【例 3.2.1】匹配字符串 'dog cat dog'中的‘dog’：

In [19]:

```
import re
match = re.match(r'dog', 'dog cat dog')
match.group(0) # group()method
```

Out[19]:

'dog'

但是，我们却匹配不了不在开头的‘cat’：

In [20]:

```
re.match(r'cat', 'dog cat dog')
```

【例 3.2.2】理解re.match( )函数的属性

In [21]:

```
import re
p = re.compile('[a-z]+') # 用re.compile()函数将含小写字母的字符串编译成一个正则对象
m = p.match('tempo')     # 将匹配的字符串存储在变量m中

print(m.group())         # 返回匹配的字符串
print(m.start(),m.end()) # 返回字符串匹配的开始位置和结尾位置
print(m.span())          # 返回一个元组，包含字符串匹配的首尾位置
```

```
tempo
0 5
(0, 5)
```

【例 3.2.3】理解group()的用法

In [22]:

```
import re
m = re.match(r'(\w+) (\w+)(?P<sign>.*)', 'hello hanxiaoyang!')

print ("m.group(1,2):", m.group(1, 2)) #返回匹配的两个分组（词）
print ("m.groups():", m.groups())      #返回所有分组（词）
print ("m.groupdict():", m.groupdict())
print ("m.start(2):", m.start(2))      # 返回第二个分组（词）的开头位置
print ("m.end(2):", m.end(2))          # 返回第二个分组（词）的结尾位置
print ("m.span(2):", m.span(2))        # 返回一个元祖包含第二个分组（词）的首尾位置
print (r"m.expand(r'\2 \1'):", m.expand(r'\2 \1')) #将匹配到的分组代入template中然后返回了相反位置
```

```
m.group(1,2): ('hello', 'hanxiaoyang')
m.groups(): ('hello', 'hanxiaoyang', '!')
m.groupdict(): {'sign': '!'}
m.start(2): 6
m.end(2): 17
m.span(2): (6, 17)
m.expand(r'\2 \1'): hanxiaoyang hello
```

3.3. 函数 re.search ( )

re.search(pattern, string, flags=0) search() 函数和 match()函数近似，但是search()函数不局限于查询匹配字符串开头。

【例 3.3】同样例子3.2.1，这一次我们匹配字符串 'dog cat dog'中的'cat'

In [23]:

```
import re
search = re.search(r'cat', 'dog cat dog cat')
search.group(0)
```

Out[23]:

```
'cat'
```

3.4. 函数 re.findall ( )

re.findall(pattern, string, flags=0) 比起返回匹配的内容，findall()会给我们返回一个匹配内容的列表

我们仍然使用上面相同的例子:

【例 3.4.1】利用 findall()函数匹配字符串 'dog cat dog'中的‘dog’和‘cat’

In [24]:

```
import re
find_dog = re.findall(r'dog', 'dog cat dog')
find_cat = re.findall(r'cat', 'dog cat dog')
print (find_dog, find_cat)
```

['dog', 'dog'] ['cat']

【例 3.4.2】找出下列这段话的冠词the

In [25]:

```
import re
speech = "I just spoke with Governor Romney and I congratulated him and Paul Ryan on a hard-foug
ht campaign. (Cheers, applause.) We may have battled fiercely, but it's only because we love thi
s country deeply and we care so strongly about its future. From George to Lenore to their son Mi
tt, the Romney family has chosen to give back to America through public service. And that is a l
egacy that we honour and applaud tonight. (Cheers, applause.) In the weeks ahead, I also look fo
rward to sitting down with Governor Romney to talk about where we can work together to move this
country forward."
len(re.findall(r'the',speech))
```

Out[25]:

4

3.5. 函数 re.finditer ( )

re.finditer(pattern, string, flags=0) 搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

【例 3.5.1】找寻每个匹配的字符串分组的匹配位置区间

In [26]:

```
matches = re.finditer(r'([a-zA-Z]+) \d+', 'June 24, August 9, Dec 12')
for match in matches:
    print ("Match at index: %s, %s" % (match.start(), match.end()))
```

```
Match at index: 0, 7
Match at index: 9, 17
Match at index: 19, 25
```

【例 3.5.2】找寻字符串中所有数字

In [27]:

```
import re
p = re.compile(r'\d+')
for m in p.finditer('one1two2three3four4'):
    print (m.group())
```

```
1
2
3
4
```

3.6. 函数 re.split ( )

re.split(pattern,string,maxsplit=0,flags=0) 按照能够匹配的子串将string分割后返回列表。maxsplit用于指定最大分割次数，不指定将全部分割。

【例 3.6.1】匹配并分割字符串中所有数字

In [28]:

```
import re
p = re.compile(r'(\d+)')
print (p.split('one111112two245568three3873984723four4'))
```

```
['one', '111112', 'two', '245568', 'three', '3873984723', 'four', '4', '']
```

【例 3.6.2】体验不同分割条件来分割字符串

In [29]:

```
import re
num = re.split(r'-', '0376-2233-445') # 以-分割字符串，返回列表中不包含-
num1 = re.split(r'(-)','0376-2233-445') # 以(-)分组分割字符串，返回列表中包含-
num2 = re.split(r'-','0376-2233-445',maxsplit = 1) # 指定分割次数为1，剩下的子串作为一个整体
num3 = re.split(r'(\d+)','0376-2233-445') # 分组从字符串开头匹配，列表首尾为一个空字符串

print('only numebr: ', num)
print('numebr and sign: ', num1)
print('numebr split once: ', num2)
print('split with space: ', num3)
```

```
only numebr:  ['0376', '2233', '445']
numebr and sign:  ['0376', '-', '2233', '-', '445']
numebr split once:  ['0376', '2233-445']
split with space:  ['', '0376', '-', '2233', '-', '445', '']
```

3.7. 函数 re.sub( )

re.sub(pattern, string, flags=0) sub()函数会返回按要求修改过的字符串内容。

sub(repl, string[, count]) | re.sub(pattern, repl, string[, count]):

- 使用repl替换string中每一个匹配的子串后返回替换后的字符串。
  - 当repl是一个字符串时，可以使用\id或\g、\g引用分组，但不能使用编号0。
  - 当repl是一个方法时，这个方法应当只接受一个参数（Match对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。 count用于指定最多替换次数，不指定时全部替换。

【例 3.7.1】修改电话号码格式

In [30]:

```
import re
phone = "2004-959-559 # This is Phone Number"

# 删除字符串中python的注释
num = re.sub(r'#.*$', "", phone)
print ("Phone Num : "+ num)

# 删除除数字以外的所有元素
num = re.sub(r'\D', "", phone)
print ("Phone Num : "+ num)
```

```
Phone Num : 2004-959-559
Phone Num : 2004959559
```

【例 3.7.2】变更输出字符串顺序

In [31]:

```
import re
p = re.compile(r'(\w+) (\w+)')    #用re.compile()函数将匹配的词编译成一个正则对象
s = 'i say, hello hanxiaoyang!'   # 被匹配的字符串
print (p.sub(r'\2 \1', s))         # 更改了匹配的字符串两个分组的位置

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print (p.sub(func, s))
```

```
say i, hanxiaoyang hello!
I Say, Hello Hanxiaoyang!
```

3.8. 函数 re.subn( )

re.subn(pattern,repl,string,count=0,flags=0) 返回一个元组（新字符串，替换次数）

【例 3.8.1】变更输出新字符串及替换次数

In [32]:

```
import re
p = re.compile(r'(\w+) (\w+)')
s = 'i say, hello hanxiaoyang!'
print (p.subn(r'\2 \1', s))

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print (p.subn(func, s))
```

```
('say i, hanxiaoyang hello!', 2)
('I Say, Hello Hanxiaoyang!', 2)
```

【例 3.8.2】对比re.sub()和re.subn()

In [33]:

```
import re
words = 'Thank you. Thank you. Thank you so much.'
result1 = re.sub(r'Thank you',r'Thanks',words) # 返回替换后的新字符串
result2 = re.subn(r'Thank you',r'Thanks',words) # 返回一个元组（新字符串，替换次数）
print (result1,'\n',result2)
```

```
Thanks. Thanks. Thanks so much.
 ('Thanks. Thanks. Thanks so much.', 3)
```

3.9. 函数 re.escape( )

re.escape(pattern) 转义除ASCII字母、数字和下划线的所有字符。

Return string with all non-alphanumerics backslashed; this is useful if you want to match an arbitrary literal string that may have regular expression metacharacters in it.

In [34]:

```
help(re.escape) # help re.escape（）函数的用法
```

```
Help on function escape in module re:

escape(pattern)
    Escape all the characters in pattern except ASCII letters, numbers and '_'.
```

【例 3.9.1】

In [35]:

```
import re
print(re.escape('^a.*$'))
print (re.escape('The pattern [0-9A-Fa-f] is equal to the pattern \w'))
```

```
\^a\.\*\$
The\ pattern\ \[0\-9A\-Fa\-f\]\ is\ equal\ to\ the\ pattern\ \\w
```

3.10. 函数 re.purge( )

re.purge() 清除所有正则表达式缓存

In [36]:

```
help(re.purge) # help re.purge（）函数的用法
```

```
Help on function purge in module re:

purge()
    Clear the regular expression caches
```

4. 正则表达式练习题

4.1 基础题

1、编写一个函数匹配包含字母'z'的单词

In [37]:

```
# 参考答案
import re
def match_z(strings):
    result = re.findall(r'\b\w*z\w*\b',strings) #[a-zA-Z]
    return result

match_z('zoo')
```

Out[37]:

```
['zoo']
```

2、编写一个函数提取字符串中的第一个单词

In [38]:

```
# 参考答案
import re
def first_word(strings):
    f_word = re.match(r'\b\w+\b',strings)
    return f_word.group()

first_word('hello world')
```

Out[38]:

```
'hello'
```

3、调出字符串'AV is largest Analytics community of India'中每个字母Extract each character (提示：用“\w“)

In [39]:

```
# 答案如下：
import re
result=re.findall(r'\w','AV is largest Analytics community of India')
print(result)
```

```
['A', 'V', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't', 'A', 'n', 'a', 'l', 'y',
't', 'i', 'c', 's', 'c', 'o', 'm', 'm', 'u', 'n', 'i', 't', 'y', 'o', 'f', 'I',
'n', 'd', 'i', 'a']
```

4、提取出字符串'AV is largest Analytics community of India'中每个单词的前两个字母(提示：用“\b“)

In [40]:

```
# 答案如下：
import re
result=re.findall(r'\b\w.','AV is largest Analytics community of India')
print (result)
```

```
['AV', 'is', 'la', 'An', 'co', 'of', 'In']
```

4.2 中级题

1、编写一个函数将字符串中的'Road'简写为'Rd.'

In [41]:

```
# 参考答案
import re
def abb(strings):
    result = re.sub(r'\bRoad\b','Rd.',strings)
    return result

abb('Zhongshan Road')
```

Out[41]:

```
'Zhongshan Rd.'
```

2、编写一个函数字符串是否为带2-3位小数的正数或负数

In [42]:

```
# 参考答案
import re
def decimals(strings):
    if re.match(r'-?\d+.\d{2,3}',str(strings)) != None:
        print (True)
    else:
        print (False)

decimals(-2.35)
```

```
True
```

3、提取出字符串'Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009'中的年份(提示：'\d'提取数字)

In [43]:

```
# 答案如下：
result=re.findall(r'\d{2}-\d{2}-(\d{4})','Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC
 67-8945 12-01-2009')
print(result)
```

['2007', '2011', '2009']

In [44]:

```
re.findall(r'\d{2}-\d{2}-\d{4}','Amit 34-3456 12-05-2007')
```

Out[44]:

['12-05-2007']

4、你想在字符串 'xyz alice-b@google.com purple monkey'中找到邮箱地址.(提示：用r'\w+@\w+'模式)

In [45]:

```
# 答案如下：
str = 'purple alice-b@google.com monkey dishwasher'
match = re.search(r'\w+@\w+', str)
if match:
    print (match.group())
```

b@google

5、打开一个txt文档，利用 findall()函数，返回特定pattern的list

In [46]:

```
# 答案提示：
f = open('test.txt', 'r')  # Open file
strings = re.findall(r'some pattern', f.read())# Feed the file text into findall(); it returns a
list of all the found strings
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-46-1453d65db54a> in <module>
      1 # 答案提示：
----> 2 f = open('test.txt', 'r')  # Open file
      3 strings = re.findall(r'some pattern', f.read())# Feed the file text
 into findall(); it returns a list of all the found strings

FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

4.2 挑战题

1、国内电话号码(0511-4405222、021-87888822)，编写一个函数检查电话号码是否合格

In [ ]:

```
# 参考答案
import re
def check_num(strings):
    if re.match(r'\d{3}-\d{8}|\d{4}-\d{7}',str(strings)) != None:
        print ('strings is a phone number')
    else:
        print ('strings is not a phone number')

check_num('87888822')
```

2、密码(以字母开头，长度在6~18之间，只能包含字母、数字和下划线)，编写一个函数检查密码是否合格

In [ ]:

```
# 参考答案
import re
def password(strings):
    if re.match(r'^[a-zA-Z]\w{5,17}$',strings) != None:
        print (True)
    else:
        print (False)

password('ydfnasdfowiejjkfkjfioe')
```

3、提取一个HTML数据中在 <td> 和</td>之间除了第一个数字指数以外的其他所有信息。

In [ ]:

```
# 参考答案:
result=re.findall(r'<td>\w+</td>\s<td>(\w+)</td>\s<td>(\w+)</td>',str)  #假设HTML代码存在str中
print (result)
```

5. 正则表达式相关学习资料链接

5.1 验证工具

我们最喜爱的正则表达式在线验证工具之一： http://regexr.com/ (http://regexr.com/)

5.2 练习工具

正则表达式进阶练习 (https://alf.nu/RegexGolf)

5.3 最常用的正则表达式

最常用的正则表达式总结于此：http://www.cnblogs.com/zxin/archive/2013/01/26/2877765.html (http://www.cnblogs.com/zxin/archive/2013/01/26/2877765.html)

5.4 50道正则表达式练习题

https://www.w3resource.com/python-exercises/re/ (https://www.w3resource.com/python-exercises/re/)