## 导入和查看正则表达式模块

import re

查看正则表达式模块方法

dir(re)

[‘DEBUG’, ‘DOTALL’, ‘I’, ‘IGNORECASE’, ‘L’, ‘LOCALE’, ‘M’, ‘MULTILINE’, ‘S’, ‘Scanner’, ‘T’,’TEMPLATE’, ‘U’, ‘UNICODE’, ‘VERBOSE’, ‘X’, ‘\_MAXCACHE’, ‘all‘, ‘builtins‘, ‘doc‘,’file‘, ‘name‘, ‘package‘, ‘version‘, ‘\_alphanum’, ‘\_cache’, ‘\_cache\_repl’,’\_compile’, ‘\_compile\_repl’, ‘\_expand’, ‘\_pattern\_type’, ‘\_pickle’, ‘\_subx’, ‘compile’,’copy\_reg’, ‘error’, ‘escape’, ‘findall’, ‘finditer’, ‘match’, ‘purge’, ‘search’, ‘split’,’sre\_compile’, ‘sre\_parse’, ‘sub’, ‘subn’, ‘sys’, ‘template’]

提示：

1. 当我们不会用模块方法的时候用help

2. py2中pattern中的字符串要和string的编码一致，不然会找不到，这个经常出现。

## 正则表达式语法

基本规则

‘[‘ ‘]’ 字符集合设定符

由一对方括号括起来的字符，表明一个字符集合，能够匹配包含在其中的任意一个字符。比如 [abc123]，表明字符’a’ ‘b’ ‘c’ ‘1’ ‘2’ ‘3’都符合它的要求。可以被匹配。

在’[‘ ‘]’中还可以通过 ’-‘ 减号来指定一个字符集合的范围，比如可以用[a-zA-Z]来指定所以英文字母的大小写，不可以把大小的顺序颠倒了。

如果在’[‘ ‘]’里面的开头写一个 ‘^’ 号，则表示取非，即在括号里的字符都不匹配。如[^a-zA-Z]表明不匹配所有英文字母。但是如果 ‘^’不在开头，则它就不再是表示取非，而表示其本身，如[a-z^A-Z]表明匹配所有的英文字母和字符’^’。

‘|’ 或规则

将两个规则并列起来，以‘|’连接，表示只要满足其中之一就可以匹配。比如 [a-zA-Z]|[0-9] 表示满足数字或字母就可以匹配，这个规则等价于 [a-zA-Z0-9]

’|’要注意两点：

第一， 它在’[‘ ‘]’之中不再表示或，而表示他本身的字符。如果要在’[‘ ‘]’外面表示一个’|’字符，必须用反斜杠引导，即 ’\|’ ;

第二， 它的有效范围是它两边的整条规则，比如‘dog|cat’匹配的是‘dog’和’cat’，而不是’g’和’c’。如果想限定它的有效范围，必需使用一个无捕获组 ‘(?: )’包起来。比如要匹配 ‘I have a dog’或’I have a cat’，需要写成r’I have a (?:dog|cat)’ ，而不能写成 r’I have a dog|cat’，其中dog和cat是字符串，不能是变量。

例:

s = ‘I have a dog , I have a cat’

re.findall( r’I have a (?:dog|cat)’ , s )

[‘I have a dog’, ‘I have a cat’] #正如我们所要的

下面再看看不用无捕获组会是什么后果：

re.findall( r’I have a dog|cat’ , s )

[‘I have a dog’, ‘cat’] #它将’I have a dog’ 和’cat’当成两个规则了

至于无捕获组的使用，后面将仔细说明。这里先跳过。

‘.’ 匹配所有字符匹配除换行符’\n’外的所有字符

如果使用了=re.S选项，匹配包括’\n’的所有字符。

例：

s=’123 \n456 \n789’

findall(r‘.+’,s)

[‘123’, ‘456’, ‘789’]

re.findall(r‘.+’ , s , re.S)

[‘123\n456\n789’]

‘^’和’$’ 匹配字符串开头和结尾

注意’^’不能在‘[ ]’中，否则请看上面的’[‘ ‘]’说明。

在多行模式下，它们可以匹配每一行的行首和行尾。具体请看后面compile函数说明的’M’选项部分

‘\A’ 匹配字符串开头

匹配字符串的开头。它和’^’的区别是，’\A’只匹配整个字符串的开头（相当于单行模式），即使在’M’模式下，它也不会匹配其它行的行首。

‘\Z’ 匹配字符串结尾

匹配字符串的结尾。它和’$’的区别是，’\Z’只匹配整个字符串的结尾，即使在’M’模式下，它也不会匹配其它各行的行尾。

例：

s= ‘12 34\n56 78\n90’

re.findall( r’^\d+’ , s , re.M ) #匹配位于行首的数字

[‘12’, ‘56’, ‘90’]

re.findall( r’\A\d+’, s , re.M ) #匹配位于字符串开头的数字

[‘12’]

re.findall( r’\d+$’ , s , re.M ) #匹配位于行尾的数字

[‘34’, ‘78’, ‘90’]

re.findall( r’\d+\Z’ , s , re.M ) #匹配位于字符串尾的数字

[‘90’]

‘\d’ 匹配数字

’\d’表示匹配一个数字，即等价于[0-9]

‘\D’ 匹配非数字

匹配一个非数字的字符，等价于[^0-9]

‘\w’ 匹配字母和数字

匹配所有的英文字母和数字，即等价于[a-zA-Z0-9]。

‘\W’ 匹配非英文字母和数字

即’\w’的补集，等价于[^a-zA-Z0-9]。

‘\s’ 匹配间隔符

即匹配空格符、制表符、回车符等表示分隔意义的字符，它等价于[ \t\r\n\f\v]。（注意最前面有个空格)

‘\S’ 匹配非间隔符

即间隔符的补集，等价于[^ \t\r\n\f\v]

‘\b’ 匹配单词边界(相当于前向界定和后向界定)

它匹配一个单词的边界，比如空格等，不过它是一个‘0’长度字符，它匹配完的字符串不会包括那个分界的字符。而如果用’\s’来匹配的话，则匹配出的字符串中会包含那个分界符。

例：

s = ‘abc abcde bc bcd’

re.findall( r’\bbc\b’ , s ) #匹配一个单独的单词 ‘bc’ ，而当它是其它单词的一部分的时候不匹配

[‘bc’] #只找到了那个单独的’bc’

re.findall( r’\sbc\s’ , s ) #匹配一个单独的单词 ‘bc’

[’ bc ‘] #只找到那个单独的’bc’，不过注意前后有两个空格，可能有点看不清楚

‘\B’ 匹配非边界

和’\b’相反，它只匹配非边界的字符。它同样是个0长度字符。

re.findall( r’\Bbc\w+’ , s ) #匹配包含’bc’但不以’bc’为开头的单词

[‘bcde’] #成功匹配了’abcde’中的’bcde’，而没有匹配’bcd’

## 精确匹配和最小匹配

Python正则式还可以精确指定匹配的次数 ：

1. ‘{m}’ 精确匹配m次; ‘{m,n}’ 匹配最少m次，最多n次。(n>m)

2. 如果你只想指定一个最少次数或只指定一个最多次数，你可以把另外一个参数空起来。比如你想指定最少3次，可以写成 {3,} （注意那个逗号），同样如果只想指定最大为5次，可以写成{，5}，也可以写成{0,5}。

例 寻找下面字符串中

a：3位数

b: 2位数到4位数

c: 5位数以上的数

d: 4位数以下的数

>>> s= ‘ 1 22 333 4444 55555 666666 ‘

>>> re.findall( r’\b\d{3}\b’ , s ) # a：3位数

['333']

>>> re.findall( r’\b\d{2,4}\b’ , s ) # b: 2位数到4位数

['22', '333', '4444']

>>> re.findall( r’\b\d{5,}\b’, s ) # c: 5位数以上的数

['55555', '666666']

>>> re.findall( r’\b\d{1,4}\b’ , s ) # 4位数以下的数

['1', '22', '333', '4444']

数量词的贪婪模式与非贪婪模式 ‘\*?’ ‘+?’ ‘??’ 最小匹配

正则表达式通常用于在文本中查找匹配的字符串。Python里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；非贪婪的则相反，总是尝试匹配尽可能少的字符。例如：正则表达式"ab\*"如果用于查找"abbbc"，将找到"abbb"。而如果使用非贪婪的数量词"ab\*?"，将找到"a"。

‘\*’ ‘+’ ‘?’通常都是尽可能多的匹配字符。有时候我们希望它尽可能少的匹配。比如一个c语言的注释 ‘/\* part 1 \*/ /\* part 2 \*/’，如果使用最大规则：

>>> s =r ‘/\* part 1 \*/ code /\* part 2 \*/’

>>> re.findall( r’/\\*.\*\\*/’ , s )

[‘/\* part 1 \*/ code /\* part 2 \*/’]

结果把整个字符串都包括进去了。如果把规则改写成

>>> re.findall( r’/\\*.\*?\\*/’ , s ) #在\*后面加上?，表示尽可能少的匹配

['/\* part 1 \*/', '/\* part 2 \*/']

结果正确的匹配出了注释里的内容

‘(?:)’ 无捕获组

当你要将一部分规则作为一个整体对它进行某些操作，比如指定其重复次数时，你需要将这部分规则用’(?:’ ‘)’把它包围起来，而不能仅仅只用一对括号。

例：匹配字符串中重复的’ab’

s=’ababab abbabb aabaab’

re.findall( r’\b(?:ab)+\b’ , s )

[‘ababab’]

如果仅使用一对括号： re.findall( r’\b(ab)+\b’ , s )

[‘ab’]

这是因为如果只使用一对括号，那么这就成为了一个组(group)。

## Python正则库API及正则选项

re选项

re库API中，一般都有flags参数，通过该参数指定正则表达式选项。传递时一般使用简写，比如开启DOTALL和MULTILINE使用re.I|re.M

A ASCII 使\w\W\b\B\d\D匹配ASCII字符

I IGNORECASE 忽略大小写

L LOCALE 使\w\W\b\B匹配本地字符集

M MULTILINE 多行模式，"^" 匹配每行开头，"$"匹配每行结尾

S DOTALL "." 匹配所有字符，包括"\n"

X VERBOSE 详细模式，忽略空白可以加入注释

U UNICODE 使\w\W\b\B\d\D匹配unicode字符集

API速查

这里只是列出API，便于查阅，后面会详细介绍API的使用。建议先跳过这一段，直接看后面的Sample，再回过头来看这一段。

API分为三组，第一组是模块API(Module Contents),通过re.xx()使用；第二组是表达式API(Regular Expression Objects)，re.complie()函数会返回一个表达式对象，通过该对象使用的函数；第三组是匹配对象API(Match Objects)，像search这些函数都会返回一个匹配结果，这组API用于操作结果集。

re库对于很多函数，例如match，都提供了两种调用方式，一是直接通过re库调用，将正则表达式作为参数，二是先用complie编译表达式，通过返回的对象调用，方法二在正则表达式会被多次使用时会减少重复编译花费的时间。

re.compile(pattern, flags=0) 预编译一个正则表达式，返回一个表达式对象

re.search(pattern, string, flags = 0) 在字符串中找匹配的串，返回第一个匹配到的匹配对象

re.match(pattern, string, flags=0) 从头开始匹配，返回匹配对象

re.split(pattern, string, maxsplit=0, flags=0) 使用pattern分割字符串，返回一个结果list

re.findall(pattern, string, flags=0) search加强版，返回所有的匹配对象的list

re.finditer(pattern, string, flags=0) 返回一个迭代器，用户可以使用迭代器查看所有匹配对象

re.sub(pattern, repl, string, count=0, flags=0) 使用repl替换string中pattern匹配到的部分；

这里repl可以是一个函数，参数是匹配对象，返回要替代的串

re.subn(pattern, repl, string, count=0, flags=0) 类似sub，返回元组(new\_string, number\_of\_subs\_made)

re.escape(string) 将所有的非字母数字字符前加"\"后返回

re.purge() 清空正则表达式缓存

表达式API(Regular Expression Objects)

flags 编译时的flag

groups 表达式中分组的数量

groupindex 以有别名的组别名为键、编号为值的字典

pattern 编译时用的表达式字符串

search(string[, pos[, endpos]]) 从Pos处开始查找字符串，返回匹配对象

match(string[, pos[, endpos]]) 从Pos处匹配字符串，返回匹配对象

split(string, maxsplit=0) 同re.split

findall(string[, pos[, endpos]]) 从Pos处查找所有匹配的字符串，返回所有匹配对象的list

finditer(string[, pos[, endpos]]) 从Pos处查找所有的字符串，返回一个迭代器

sub(repl, string, count=0) 同re.sub

subn(repl, string, count=0) 同re.subn

匹配对象API(Match Objects)

pos 传递给函数的pos

endpos 传递给函数的endpos

lastindex 最后一个捕获的group的下标

lastgroup 最后一个捕获的group的名字

re 调用match或者search的表达式对象

string match或者search的字符串

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<id>、\g<name>引用分组

注意0不能使用，另外\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。

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，0代表整个匹配串

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,…n))

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

start([group]) 返回指定组的串在原串中的起始索引

end([group]) 返回指定组的串在原串中的结束索引

span([group]) 返回(start(group), end(group))

## Match Object对象拥有的方法

1.group([group1,…])

　 返回匹配到的一个或者多个子组。如果是一个参数，那么结果就是一个字符串，如果是多个参数，那么结果就是一个参数一个item的元组。group1的默认值为0(将返回所有的匹配值).如果groupN参数为0，相对应的返回值就是全部匹配的字符串，如果group1的值是[1…99]范围之内的,那么将匹配对应括号组的字符串。如果组号是负的或者比pattern中定义的组号大，那么将抛出IndexError异常。如果pattern没有匹配到，但是group匹配到了，那么group的值也为None。如果一个pattern可以匹配多个，那么组对应的是样式匹配的最后一个。另外，子组是根据括号从左向右来进行区分的。

　>m=re.match(“(\w+) (\w+)”,”abcd efgh, chaj”)

　>m.group() # 匹配全部

　‘abcd efgh’

　>m.group(1) # 第一个括号的子组.

　‘abcd’

　>m.group(2)

　‘efgh’

　>m.group(1,2) # 多个参数返回一个元组

　(‘abcd’, ‘efgh’)

　>m=re.match("(?P<first\_name>\w+) (?P<last\_name>\w+)","sam lee")

m.group(“first\_name”) #使用group获取含有name的子组

‘sam’

m.group(“last\_name”)

‘lee’

　下面把括号去掉

　>m=re.match(“\w+ \w+”,”abcd efgh, chaj”)

　>m.group()

　‘abcd efgh’

　>m.group(1)

　Traceback (most recent call last):

File “pyshell#32>”, line 1, in

m.group(1)

　IndexError: no such group

　If a group matches multiple times, only the last match is accessible:

如果一个组匹配多个，那么仅仅返回匹配的最后一个的。

　>m=re.match(r”(..)+”,”a1b2c3”)

　>m.group(1)

　‘c3’

　>m.group()

　‘a1b2c3’

　Group的默认值为0，返回正则表达式pattern匹配到的字符串

　>s=”afkak1aafal12345adadsfa”

　>pattern=r”(\d)\w+(\d{2})\w”

　>m=re.match(pattern,s)

　>print m

　None

　>m=re.search(pattern,s)

　>m

　<\_sre.SRE\_Match object at 0x00C2FDA0>

　>m.group()

　‘1aafal12345a’

　>m.group(1)

　‘1’

　>m.group(2)

　‘45’

　>m.group(1,2,0)

　(‘1’, ‘45’, ‘1aafal12345a’)

2.groups([default])

　返回一个包含所有子组的元组。Default是用来设置没有匹配到组的默认值的。Default默认是”None”,

　>m=re.match(“(\d+).(\d+)”,”23.123”)

　>m.groups()

　(‘23’, ‘123’)

　>m=re.match(“(\d+).?(\d+)?”,”24”) #这里的第二个\d没有匹配到，使用默认值”None”

　>m.groups()

　(‘24’, None)

　>m.groups(“0”)

　(‘24’, ‘0’)

3.groupdict([default])

　返回匹配到的所有命名子组的字典。Key是name值，value是匹配到的值。参数default是没有匹配到的子组的默认值。这里与groups()方法的参数是一样的。默认值为None

　>m=re.match(“(\w+) (\w+)”,”hello world”)

　>m.groupdict()

　{}

　>m=re.match(“(?P\w+) (?P\w+)”,”hello world”)

　>m.groupdict()

　{‘secode’: ‘world’, ‘first’: ‘hello’}

通过上例可以看出，groupdict()对没有name的子组不起作用

## re常用模块

re.compile(pattern, flags=0)

re.compile 可以把正则表达式编译成一个正则对象。可以把那些经常使用的正则表达式编译成正则表达式对象，这样可以提高一定的效率。

help(re.compile)

compile(pattern, flags=0)

第一个参数：规则

第二个参数：标志位

实例：

test=”Hi, nice to meet you where are you from?”

k=re.compile(r’\w\*o\w\*’) #匹配带o的字符串

dir(k)

[‘copy‘, ‘deepcopy‘, ‘findall’, ‘finditer’, ‘match’, ‘scanner’, ‘search’, ‘split’,’sub’, ‘subn’]

print k.findall(test) #显示所有包涵o的字符串

[‘to’, ‘you’, ‘you’, ‘from’]

print k.sub(lambdam: ‘[‘+m.group(0) +’]’,test) # 将字符串中含有o的单词用[]括起来

Hi, nice [to] meet [you] where are [you] [from]?

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

re.search 函数会在字符串内查找模式匹配，直到找到第一个匹配然后返回一个对象k。

k.group(0)代表整个匹配模式对应的字符串，k.group(1)代表匹配模式中的组对应的字符串

如果字符串没有匹配，则返回None。

help(re.search)

search(pattern, string, flags=0)

第一个参数：规则

第二个参数：表示要匹配的字符串

第三个参数：标致位，用于控制正则表达式的匹配方式

实例：下面的例子kuangl

name=”Hello,My name is kuangl,nice to meet you…”

k=re.search(r’k(uan)gl’,name)

if k:

… print k.group(0),k.group(1)

… else:

… print ”Sorry,not search!”

…

kuangl uan

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

re.match 尝试从字符串的开始匹配一个模式，也等于说是匹配第一个单词。

help(re.match)

match(pattern, string, flags=0)

第一个参数：规则

第二个参数：表示要匹配的字符串

第三个参数：标致位，用于控制正则表达式的匹配方式

实例1：下面的例子匹配Hello单词

name=”Hello,My name is kuangl,nice to meet you…”

k=re.match(r”(\H….)”,name)

if k:

… print k.group(0),’\n’,k.group(1)

… else:

… print “Sorry,not match!”

…

Hello

Hello

re.match与re.search的区别：re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；而re.search匹配整个字符串，直到找到一个匹配。

实例2：判断字符串i是否是一个数字串

if re.match('\d+', i)

re.split(pattern, string, maxsplit=0, flags=0)

re.split 用于来分割字符串

help(re.split)

split(pattern, string, maxsplit=0)

第一个参数：规则

第二个参数：字符串

第三个参数：最大分割字符串，默认为0，表示每个匹配项都分割

实例：分割所有的字符串

test=”Hi, nice to meet you where are you from?”

re.split(r”\s+”,test)

[‘Hi,’, ‘nice’, ‘to’, ‘meet’, ‘you’, ‘where’, ‘are’, ‘you’, ‘from?’]

re.split(r”\s+”,test,3) #分割前三个

[‘Hi,’, ‘nice’, ‘to’, ‘meet you where are you from?’]

python使用多个分隔符分割字符串

下面是使用汉语标点符号[。！？，]和英文标点符号[!?,]以及微博转发符号“//@”作为断句的分隔符的示例

line = '地丁 一。雪！地陪？霜 ，阿!地?地址,鄄鄄//@村鄄李'

print(re.split(r'[。！？，!?,]|//@', line))

['地丁 一', '雪', '地陪', '霜 ', '阿', '地', '地址', '鄄鄄', '村鄄李']

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

re.findall 在目标字符串查找符合规则的字符串

help(re.findall)

findall(pattern, string, flags=0)

第一个参数：规则

第二个参数：目标字符串

但三个参数：后面还可以跟一个规则选择项

返回的结果是一个列表(如果有括号()分组，则只返回所有分组的列表)，列表中存放的是符合规则的字符串，如果没有符合规则的字符串找到，就会返回一个空值。

实例：查找邮件账号

mail=’user01@mail.comuser02@mail.com user04@mail.com’#第3个故意没有尖括号

re.findall(r’(\w+@m….[a-z]{3})’,mail)

[‘user01@mail.com’, ‘user02@mail.com’, ‘user04@mail.com’]

实例2：

post\_ori\_text=r'sinaSSOController.preloginCallBack({"retcode":0,"servertime":1441512360,"pcid":"gz-da3c627b7b8260ba5cfd453f4eb3347ca01f","nonce":"9OZ4S7","pubkey":"EB2A385686618\*\*\*\*","rsakv":"1330428213","exectime":3})'

re.findall('{"retcode":(.\*?),"servertime":(.\*?),"pcid":"(.\*?)","nonce":"(.\*?)","pubkey":"(.\*?)","rsakv":"(.\*?)","exectime":(.\*?)}', post\_ori\_text, re.I)[0]

上面 的结果为(u'0', u'1441512470', u'gz-eae72a8b0f6be469f4ac627ca38f7069d5c5', u'D2HXK0', u'EB2A38568661887FA18\*\*\*\*\*\*', u'1330428213', u'2')

re.sub(pattern, repl, string, count=0, flags=0)

re.sub 用于替换字符串的匹配项。If the pattern isn’t found,string is returned unchanged.

第一个参数：规则

第二个参数：替换后的字符串

第三个参数：字符串

第四个参数：替换个数。默认为0，表示每个匹配项都替换

第二个参数：repl

repl，就是replacement，被替换的字符串的意思。repl可以是字符串，也可以是函数。

repl是字符串

如果repl是字符串的话，其中的任何反斜杠转义字符，都会被处理的。

\n：会被处理为对应的换行符；

\r：会被处理为回车符；

其他不能识别的转移字符，则只是被识别为普通的字符：

比如\j，会被处理为j这个字母本身；

反斜杠加g以及中括号内一个名字，即：\g<name>，对应着命了名的组，named group

repl是函数

import re;

def pythonReSubDemo():

inputStr = "hello 123 world 456";

def \_add111(matched):

intValue = int(matched.group("number"))

return str(intValue + 111)

replacedStr = re.sub("(?P<number>\d+)", \_add111, inputStr);

print replacedStr