Python正则表达式与re

还差一点 关注

1. 基本匹配

目录

```
re
sentence='python是一种语言，Python是1种很方便的语言'
```

匹配python

```
re.findall('python',sentence)
```

```
In [5]: re.findall('python',sentence)
Out[5]: ['python']
```


匹配Python

正则表达式大小写敏感。

```
re.findall('Python',sentence)
```

```
In [4]: re.findall('Python',sentence)
Out[4]: ['Python']
```


2. 元字符

正则表达式主要依赖于元字符。 元字符不代表他们本身的字面意思，他们都有特殊的含义。一些元字符写在方括号中的时候有一些特殊的意思。以下是一些元字符的介绍：

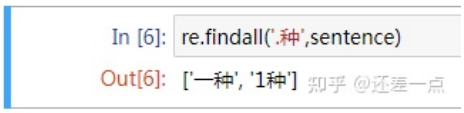
.	句号匹配任意单个字符除了换行符。
[ ]	字符种类。匹配方括号内的任意字符。
[^]	否定的字符种类。匹配除了方括号里的任意字符。
*	匹配*号之前的字符出现0次及以上。
+	匹配+号前的字符出现1次及以上。

(xyz)	字符集，匹配与 xyz 完全相等的字符串。
|	匹配|前或后的字符。
\	转义字符,匹配一些保留的字符 [ ] ( ) { } . * + ? ^ $ \ |
^	从开始行开始匹配。
$	从末端开始匹配。

2.1 点运算符 .

. 匹配任意单个字符，但不匹配换行符。

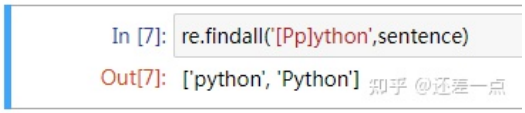
```
re.findall('.种',sentence)
```



2.2 字符集

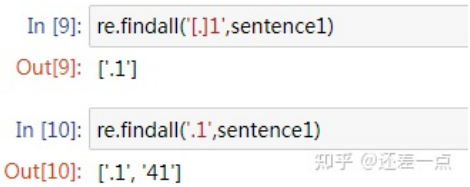
字符集也叫做字符类。 方括号用来指定一个字符集。 在方括号中的字符集不关心顺序。

```
re.findall('[Pp]ython',sentence)
```



[.] 就表示 .

```
sentence1 = '3.1415926'
re.findall('[.]1',sentence1)
re.findall('.1',sentence1)
```



否定字符集

一般来说 ^ 表示一个字符串的开头，但它用在一个方括号的开头的时候，它表示这个字符集是否定的。

```
re.findall('[^.]1',sentence1)
```

2.3 重复次数

后面跟着元字符 + * ? 的，用来指定匹配子模式的次数。 这些元字符在不同的情况下有着不同的意思。

```
re.findall('一*种',sentence)
```

* 字符和 . 字符搭配可以匹配所有的字符 .* 。

2.3.2 + 号

+ 号匹配 + 号之前的字符出现 >=1 次。

```
re.findall('一+种',sentence)
```

2.3.3 ？号

在正则表达式中元字符 ？ 标记在符号前面的字符为可选，即出现 0 或 1 次。

```
re.findall('[P]?ython',sentence)
```

2.4 {} 号

在正则表达式中 {} 是一个量词，常用来限定一个或一组字符可以重复出现的次数。

```
re.findall('[0-9]{1,5}',sentence1)
```

我们可以省略第二个参数。

```
re.findall('[0-9]{2,}',sentence1)
```

如果只有一个数字则表示重复固定的次数。

```
re.findall('[0-9]{3}',sentence1)
```

2.5 （ ） 特征标群

```
re.findall('(py)(thon)',sentence)
```

2.6 | 或运算符

或运算符就表示或，用作判断条件。

```
re.findall('一|1种',sentence)
```

2.7 转码特殊字符

反斜线 \ 在表达式中用于转码紧跟其后的字符。用于指定 { } [ ] / \ + * . $ ^ | ? 这些特殊字符。如果想要匹配这些特殊字符则要在其前面加上反斜线 \ 。

例如 . 是用来匹配除换行符外的所有字符的。如果想要匹配句子中的 . 则要写成 \. 以下这个例子 \.? 是选择性匹配 .

```
re.findall('.\..',sentence1)
```

2.8 锚点

在正则表达式中，想要匹配指定开头或结尾的字符串就要使用到锚点。 ^ 指定开头， $ 指定结尾。

2.8.1 ^ 号

^ 用来检查匹配的字符串是否用匹配字符串开头。

```
re.findall('^python',sentence)
```

```
re.findall('^Python',sentence)
```

2.8.2 $ 号

同理于 ^ 号， $ 号用来匹配字符是否是最后一个

3. 简写字符集

正则表达式提供一些常用的字符集简写。如下:

简写	描述
.	除换行符外的所有字符
\w	匹配所有字母数字，等同于 [a-zA-Z0-9_]
\W	匹配所有非字母数字，即符号，等同于： [^\w]
\d	匹配数字： [0-9]
\D	匹配非数字： [^\d]
\s	匹配所有空格字符，等同于： [\t\n\f\r\p{Z}]
\S	匹配所有非空格字符： [^\s]
\f	匹配一个换页符
\n	匹配一个换行符
\r	匹配一个回车符
\t	匹配一个制表符
\v	匹配一个垂直制表符
\p	匹配 CR/LF（等同于 \r\n），用来匹配 DOS 行终止符

4. 前后预查

先行断言和后发断言都属于**非捕获簇**（不捕获文本，也不针对组合计进行计数）。

符号	描述
?=	正先行断言-存在
?!	负先行断言-排除
?<=	正后发断言-存在
?<!	负后发断言-排除

4.1 ?=... 正先行断言

`?=...` 正先行断言，表示第一部分表达式之后必须跟着 `?=...` 定义的表达式。

返回结果只包含满足匹配条件的第一部分表达式。 定义一个正先行断言要使用 `()`。在括号内部使用一个问号和等号： `(?=...)`。

```
re.findall('[Pp]ython(?=是一种)',sentence)
```

4.2 ?!... 负先行断言

负先行断言 `?!` 用于筛选所有匹配结果，筛选条件为 其后不跟随着断言中定义的格式。 正先行断言 定义和 负先行断言 一样，区别就是 = 替换成 ！ 也就是 `(?!...)`。

4.3 ?<= ... 正后发断言

正后发断言 记作（?<=...）用于筛选所有匹配结果，筛选条件为 其前跟随着断言中定义的格式。

```
re.findall('(?<=[Pp]ython)...',sentence)
```

4.4 ?<!... 负后发断言

负后发断言 记作 （?<!...）用于筛选所有匹配结果，筛选条件为 其前不跟随着断言中定义的格式。

```
re.findall('(?<!python)是.种',sentence)
```

5. 标志

标志也叫模式修正符，因为它可以用来修改表达式的搜索结果。 这些标志可以任意的组合使用，它也是整个正则表达式的一部分。

标志	描述
i	忽略大小写。
g	全局搜索。
m	多行修饰符：锚点元字符 ^ $ 工作范围在每行的起始。

5.1 忽略大小写 (Case Insensitive)

修饰语 i 用于忽略大小写。

```
re.findall('Python',sentence,re.I)
```

5.2 全局搜索 (Global search)

修饰符 g 常用于执行一个全局搜索匹配，即（不仅仅返回第一个匹配的，而是返回全部）。

5.3 多行修饰符 (Multiline)

6. 贪婪匹配与惰性匹配 (Greedy vs lazy matching)

正则表达式默认采用贪婪匹配模式，在该模式下意味着会匹配尽可能长的子串。我们可以使用 ? 将贪婪匹配模式转化为惰性匹配模式。

```
re.findall('.*thon',sentence)
```

```
re.findall('.*?thon',sentence)
```

常用函数

> findall(pattern,string,flags=0)

```
import re
sen1='python是一种语言，Python是一种很方便的语言'
```

匹配python

```
re.findall('python',sen1)
```

```
re.findall('[Pp]ython',sen1) #大写或小写p
```

> sub(pattern,repl,string,count=0,flags=0)

```
phone = "010-88888888 # 这是一个北京电话号码"
re.sub(' #.*','',phone)
```

```
re.sub('\D','',phone) #替换非数字的字符串
```

```
re.sub('\d','',phone) #替换数字的字符串
```

> split(pattern,string,maxsplit=0,flags=0)

在实际操作中，还要对空格进行处理。

```
[i.strip() for i in re.split('\|',sen2) ]
```

常用模式

原字符 就是写什么，匹什么

```
sen1='python是一种语言，Python是一种很方便的语言'
re.findall('python',sen1)
```

. 代表一位字符，但除了换行

```
re.findall('..thon',sen1)
```

如果匹配 . 则用\.表示

```
re.findall('\d\.\d?','3.1415')
```

\n 匹配一个换行符 **\t** 匹配一个制表符

\d 匹配任意数字 **\D** 非数字

\s 空白 **\S** 非空白

\w 数字，字母，下划线 **\W** 非数字，字母，下划线

[] 匹配[]里的任何一个字符

```
number = "固定电话 01088888888 Mobilephone 13888888888 "
re.findall('1[356789]\d*',number) #匹配1开头，第二位是356789,剩下是数字的号码
```

```
re.findall('0[^356789]\d*',number)#匹配0开头，第二位不是356789,剩下是数字的号码
```

```
re.findall('[a-zA-Z]',number) #匹配任意一个字母
```

() 匹配（）里的部分

```
re.findall('\d(\d*\d)',number)
```

+ 前一个字符1次以上

```
re.findall('\d+',number)
```

* 0次以上

```
re.findall('\d\d*',number)
```

{} 表示字符个数的范围

```
re.findall('\d{5}',number) #数字5个及以上
```

```
re.findall('\d{5,}',number) #大于等于5个  {,5}小于等于
```

```
re.findall('\d{1,5}',number) #大于等于1，小于等于5
```

编辑于 2021-12-21 17:51

Python 正则表达式

评论千万条，友善第一条

还没有评论，发表第一个评论吧

推荐阅读

python小结10：python正则表达式详解

正则表达式的用法较多，也比较灵

python正则表达式以及re库的使用

本文写作目的是为了方便自己在编

Python正则表达式由浅入深（一）

CDA数据分析师 出品数据分析工作
文本处理相关，比
客户的信息，从文
等都是比较常见的