小强字符串处理之正则练习

127 search 第一个匹配串

使用正则模块，`search` 方法，找出子串第一个匹配位置。

复制

```
In [31]: s = 'i love python very much'

In [32]: pat = 'python'

In [33]: r = re.search(pat,s)

In [34]: r.span()
Out[34]: (7, 13)
```

128 match 与 search 不同

正则模块中，`match`，`search` 方法匹配字符串不同

具体不同：

- `match` 在原字符串的开始位置匹配；
- `search` 在字符串的任意位置匹配；

原字符串

复制

```
In [105]: s = 'flourish'
```

寻找模式串 `our` ，使用 `match` 方法

复制

```
In [106]: recom = re.compile('our')

In [107]: recom.match(s) # 返回 None，找不到匹配
```

使用 `search` 方法：

复制

```
In [109]: res = recom.search(s)

In [110]: res.span()
Out[110]: (2, 5) # OK，匹配成功，our 在原字符串的起始索引为 2
```

那么，什么字符串才能 使用 `match` 方法匹配到 `our` ？

比如，字符串 `ourselves`，`ours` 才能 match 到 `our`.

129 finditer 匹配迭代器

使用正则模块，`finditer` 方法，返回所有子串匹配位置的迭代器。

通过返回的对象 `re.Match`，使用它的方法 `span` 找出匹配位置。

复制

```
In [35]: s = '山东省潍坊市青州第1中学高三1班'

In [36]: pat = '1'

In [37]: r = re.finditer(pat,s)

In [38]: for i in r:
    ...:     print(i)

<re.Match object; span=(9, 10), match='1'>
<re.Match object; span=(14, 15), match='1'>
```

130 findall 所有匹配

正则模块，`findall` 方法能查找出子串的所有匹配。

原字符串 `s` ：

复制

```
In [39]: s = '一共20行代码运行时间13.59s'
```

目标查找出所有所有数字

通用字符 `\d` 匹配一位数字[0-9]

`+` 表示匹配数字前面的一个字符1次或多次

复制

```
In [40]: pat = r'\d+'

In [41]: r = re.findall(pat,s)

In [42]: print(r)
['20', '13', '59']
```

返回一个列表，找到三个数字 `20`，`13`，`59`，没有达到预期，期望找到 `20`，`13.59` 。

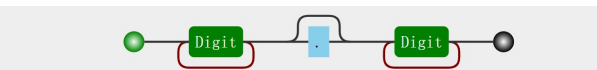
因此，需要修改正则表达式

131 案例：匹配浮点数和整数

`?` 表示前一个字符匹配 0 或 1 次

`.?` 表示匹配小数点（`.`）0 次或 1 次。

匹配浮点数和整数，第一版正则表达式：`r'\d+\.?\d+'` ，图形化演示，此正则表达式的分解演示：



复制

```
In [43]: s = '一共20行代码运行时间13.59s'

In [44]: pat = r'\d+\.?\d+'

In [45]: r = re.findall(pat,s)

In [46]: r
Out[46]: ['20', '13.59']
```

上面的正则表达式 `r'\d+\.?\d+'` ，能适配所有情况吗？

复制

```
In [47]: s = '一共2行代码运行时间1.66s'

In [48]: pat = r'\d+\.?\d+'

In [49]: r = re.findall(pat,s)

In [50]: r
Out[50]: ['1.66']
```

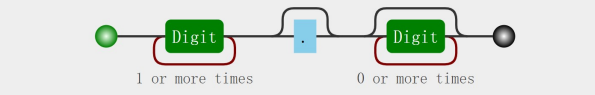
观察结果，没有匹配到数字 2.

正则难点之一，需要考虑全面，足够细心，才可能写出准确无误的正则表达式。

出现问题原因： `r'\d+\.?\d+'` ，后面的 `\d+` 表示至少有一位数字，因此，整个表达式至少会匹配两位数。

修复问题，重新优化正则表达式，将最后的 `+` 后修改为 `*` ，表示匹配前面字符 0 次、1 次或多次。

最终正则表达式为： `r'\d+\.?\d*'` ，正则分解图：



复制

```
In [51]: pat = r'\d+\.?\d*'

In [52]: r = re.findall(pat,s)

In [53]: r
Out[53]: ['2', '1.66']
```

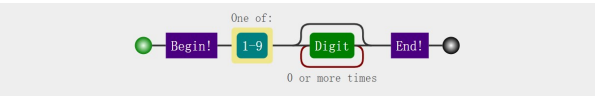
132 案例：匹配正整数

案例：写出匹配所有**正整数**的正则表达式。

如果这样写： `^\d*$` ，会匹配到 0，所以不准确。

如果这样写： `^[1-9]*` ，会匹配 `1.` 串中 `1` ，不是完全匹配，体会 `$` 的作用。

正确写法： `^[1-9]\d*$` ，正则分解图：



复制

```
In [62]: s = [-16, 1.5, 11.43, 10, 5]
In [63]: pat = r'^[1-9]\d*$'
In [70]: [i for i in s if re.match(pat,str(i))]
Out[70]: [10, 5]
```

133 正则匹配时忽略大小写

`re.I` 是 方法的可选参数，表示忽略大小写

如下，找出字符串中所有字符 `t` 或 `T` 的位置，不区分大小写。

复制

```
In [71]: s = 'That'

In [72]: pat = r't'

In [73]: r = re.finditer(pat,s,re.I)

In [79]: for i in r:
    ...:     print(i.span())
    ...:
(0, 1)
(3, 4)
```

134 split 分割单词

正则模块中 `split` 函数强大，能够处理复杂的字符串分割任务。

如果一个规则简单的字符串，直接使用字符串， `split` 函数。

如下字符串，根据分割符 `\t` 分割：

复制

```
In [91]: s = 'id\tname\taddress'

In [92]: s.split('\t')
Out[92]: ['id', 'name', 'address']
```

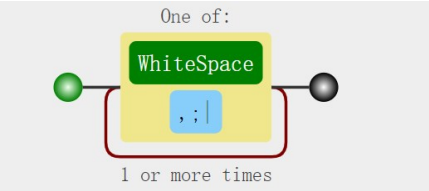
但是，对于分隔符复杂的字符串， `split` 函数就无能为力。

如下字符串，可能的分隔符有 `,` 、 `;` 、 `\t` 、 `|`

复制

```
In [100]: s = 'This,,,   module ; \t   provides|| regular ; '
```

正则字符串为： `[,\s;|]+` ， `\s` 匹配空白字符， 正则分解图，如下：




```
In [102]: words = re.split('[,\s;|]+',s)

In [103]: words
Out[103]: ['This', 'module', 'provides', 'regular', '']
```

135 sub 替换匹配串

正则模块，`sub` 方法，替换匹配到的子串

```
In [113]: content="hello 12345, hello 456321"

In [114]: pat=re.compile(r'\d+') #要替换的部分

In [115]: m=pat.sub("666",content)

In [116]: m
Out[116]: 'hello 666, hello 666'
```

136 compile 预编译

如果要用同一匹配模式，做很多次匹配，可以使用 `compile` 预先编译串。

案例

从一系列字符串中，挑选出所有正浮点数。

正则表达式为：`^[1-9]\d*\.\d*|0\.\d*[1-9]\d*$`

字符 `a|b` 表示 `a` 串匹配失败后，才执行 `b` 串，正则分解图见下：

首先，生成预编译对象 `rec`

```
In [80]: s = [-16,'good',1.5, 0.2, -0.1, '11.43', 10, '5e10']

In [81]: rec = re.compile(r'^[1-9]\d*\.\d*|0\.\d*[1-9]\d*$')
```

下面直接使用 `rec`，匹配列表中的每个元素，不用每次都预编译正则表达式，效率更高。

```
In [84]: [ i for i in s if rec.match(str(i))]
Out[84]: [1.5, 0.2, '11.43']
```

137 贪心捕获

正则模块中，根据某个模式串，匹配到结果。

待爬取网页的部分内容如下所示，现在想要提取 `<div>` 标签中的内容。

```
In [125]: content = '''
     ...:     <h>ddedadsad</h>
     ...:     <div>graph</div>
     ...:    <div>math</div>'''
```

如果正则匹配串写做 `<div>.*</div>`

```
In [118]: result = re.findall(r'<div>.*</div>',content)

In [119]: result
Out[119]: ['<div>graph</div>bb<div>math</div>']
```

返回的结果：`'<div>graph</div>bb<div>math</div>'`

如果我们不想保留字符串最开始的 `<div>` 和结尾的 `</div>` ，那么，就需要使用一对 `()` 去捕获。

正则匹配串修改为：`<div>(.*)</div>` ，只添加一对括号。

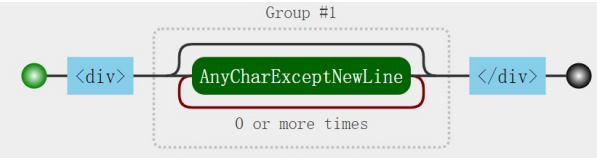
```
In [120]: result = re.findall(r'<div>(.*)</div>',content)

In [121]: result
Out[121]: ['graph</div>bb<div>math']
```

看到结果中已经没有开始的 `<div>`，结尾的 `</div>`

仅使用一对括号，便成功捕获到我们想要的部分。

`(.*)` 表示捕获任意多个字符，尽可能多的匹配字符，也被称为 `贪心捕获`

`(.*)` 的正则分解图如下所示，`.` 表示匹配除换行符外的任意字符。



138 非贪心捕获

观察上面返回的结果 `['graph</div>bb<div>math']` ，如果只想要得到两个 `<div></div>` 间

的内容，该怎么写正则表达式？

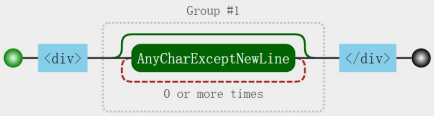
相比 `(.*)`，仅多添加一个 `?`，匹配串为：`(.*?)`

复制

```
In [125]: content = '''
     ...:     <h>ddedadsad</h>
     ...:     <div>graph</div>
     ...:    <div>math</div>'''

In [126]: result = re.findall(r'<div>(.*?)</div>',content)

In [127]: result
Out[127]: ['graph', 'math']
```

终于得到 2 个 `<div>` 对间的内容

这种匹配模式 串 `(.*?)`，被称为非贪心捕获。正则图中，红色虚线表示非贪心匹配。



139 正则匹配负整数

匹配所有负整数，不包括 0

正则表达式：`^-[1-9]\d*$`

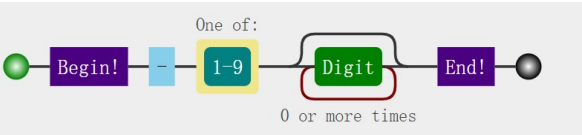
`^-` 表示字符串以 `-` 开头

`[1-9]` 表示数字 1 到 9，**注意不要写成 `\d`，因为负整数没有以 -0 开头的。**

`\d*` 表示数字0到9出现0次，1次或多次

`$` 表示字符串以数字结尾

以上分布讲解的示意图，如下所示：



测试字符串：

复制

```
import re
s = ['-1','-15756','9','-01','10','-']
pat = r'^-[1-9]\d*$'
rec = re.compile(pat)
rs = [i for i in s if rec.match(i)]
print(rs)
# 结果
# ['-1', '-15756']
```

140 正则匹配负浮点数

正确写出匹配负浮点数的正则表达式，要先思考分析。

考虑到两个实例：-0.12, -111.234 ，就必须要分为两种情况。

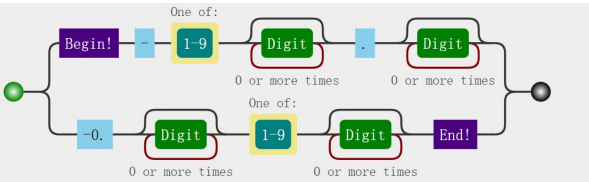
适应实例 -0.12 的正则表达式：`^-0.\d*[1-9]\d*$`，注意要考虑到 -0.0000 这种非浮点数，因此正则表达式必须这样写。

不要想当然的写为：`^-0.\d*$`，或者 `^-0.\d*[1-9]*$`，或者 `^-0.\d*[0-9]*$`，这些都是错误的！

适应实例 -111.234 的正则表达式：`^-[1-9]\d*.\d*$`

使用 `|`，综合两种情况，故正则表达式为：

`^-[1-9]\d*\.\d*|-0\.\d*[1-9]\d*$`



测试字符串：

复制

```
import re
s = ['-1','-1.5756','-0.00','-000.1','-1000.10']
pat = r'^-[1-9]\d*\.\d*|-0\.\d*[1-9]\d*$'
rec = re.compile(pat)
rs = [i for i in s if rec.match(i)]
print(rs)
# 结果
# ['-1.5756', '-1000.10']
```

141 案例：使用正则表达式判断密码是否安全

密码要求为 6 到 20 位，只包含英文字母和数字。

复制

```
import re
pat = re.compile(r'\w{6,20}') # 这是错误的，因为 \w 通配符匹配的是字母，数字和下划线，题目要求不能含有下划线
# 使用最稳的方法：\da-zA-Z 满足“密码只包含英文字母和数字”
# \d匹配数字 0-9
# a-z 匹配所有小写字符；A-Z 匹配所有大写字符
```

```
pat = re.compile(r'[\da-zA-Z]{6,20}')
```

选用 fullmatch 方法，查看是否整个字符串都匹配。

以下测试例子都返回 None，原因都在解释里。

复制

```
pat.fullmatch('qaz12') # 返回 None，长度小于 6
pat.fullmatch('qaz12wsxedcrfvtgb67890942234343434') # None 长度大于 22
pat.fullmatch('qaz_231') # None 含有下划线
```

下一章

还没有评论

说点什么

评论

存