正则表达的作用是**处理文本、提取期望的信息**，在其他的很多语言中也是类似的表达，而不只是Python。在Python中提供了很多种字符串对象的内置处理方法，
如**.split()**、**.find()**、**.index()**等。

## 一个例子

```
Python3 高级开发工程师 上海互教教育科技有限公司上海 - 浦东新区2万/月 02-18满员
测试开发工程师（C++/python）上海墨鹍数码科技有限公司上海 - 浦东新区2.5万/每月02-18未满员
Python3 开发工程师 上海德拓信息技术股份有限公司上海 - 徐汇区1.3万/每月02-18剩余11人
测试开发工程师（Python） 赫利普（上海）信息科技有限公司上海 - 浦东新区1.1万/每月02-18剩余5人
Python 高级开发工程师 上海行动教育科技股份有限公司上海 - 闵行区2.8万/月02-18剩余255人
python 开发工程师 上海优似腾软件开发有限公司上海 - 浦东新区2.5万/每月02-18满员
```

期望输出：

```
2.5
1.3
1.1
2.8
2.5
```

## 通过Python内置函数及方法：

```
### 要求：提取每个职位的薪资 ###
content = ''' '''

# 将文本内容按照行放入列表
lines = content.splitlines()
```

```
# 对每一行文本进行分析：按照规律可知，期望薪资的单位是“万/每月”或者“万/月”，那么这两个字符串前的数字即为期望输出
for line in lines:
    # 用匿名函数 lambda x: ... 其中x为形参
    # lambda x: 当if为真时，运行第一个函数，否则则为第二个
    pos2 = (lambda x: line.find(x) if line.find(x) != -1 else None)('万/每月') or (
        lambda x: line.find(x) if line.find(x) != -1 else None)('万/月')

    if isinstance(pos2, int):
        idx = pos2 - 1
        # 只要是数字或者小数点，那么就继续向前查找
        while line[idx].isdigit() or line[idx] == '.':
            idx -= 1
            # 循环完成后，idx指向的下标就是数字前面的字符
            # 那么数字的起始索引值就是 idx+1
        pos1 = idx + 1
        print(line[pos1:pos2])
```

## 通过正则表达式：

```
import re
p = re.compile(r'([\d.]+万/每{0,1}月)')
for one in p.findall(content):
    print(one)
```

其中，`.../每{0,1}月` 也可以写为 `.../每?月` 。

`re.compile(r'_')` 函数的参数就是正则表达式的字符串。上面的例子指定的搜索子串的特征为 `([\d.]+万/每{0,1}月)` ，该函数会返回一个*compile*对象，*compile*对象的 `.findall` 方法返回**所有返回的子串**，并放在一个**列表**里。

正则表达式验证工具：RegEx验证

# RegEx语法

正则表达式中包含两种字符，一种是直接进行相同匹配的字符**普通字符**；一种是特殊的字符，也被成为**元字符（metacharacters）**，如：

```
. * + ? \ [ ] ^ $ {} | ()
```

## 句号 .

表示要匹配**除了换行符以外的**任何单个字符

```
苹果是绿色的
香蕉是黄色的
橙子是橙色的
乌鸦是黑色的
```

找出上面文本中的所有颜色：

```
import re
text = '''
     苹果是绿色的
     橙子是橙色的
     香蕉是黄色的
     乌鸦是黑色的'''
char_match = re.compile(r'(.色)')
for element in char_match.findall(text):
 print(element)
```

运行结果如下：

```
绿色
黄色
橙色
黑色


Process finished with exit code 0
```

## 星号 *

表示匹配 * **前面的子表达式**任意次，包括0次

```
苹果，是绿色的
橙子，是橙色的
香蕉，是黄色的
乌鸦，是黑色的
```

从以上文本中，找出每行“，”之后的内容，包括“，”本身。

```
import re
 text = '''
     苹果，是绿色的
     橙子，是橙色的
     香蕉，是黄色的
     乌鸦，是黑色的'''
str_match = re.compile(r'(，.*)')
for element in str_match.findall(text):
 print(element)
```

(，.*) 表示：从 “，” 开始，任意次数 “*” 的任意字符 “.”

运行结果如下：

```
，是绿色的
，是橙色的
，是黄色的
，是黑色的
```

## 加号 +

表示匹配 + **前面的子表达式**一次或多次，不包括0次

## 花括号 { }

表示匹配**前面**的**子表达式**指定次数

```
text = '''
        张飒，158 4665 7365，43
        李嗣，178 6543 6789，26
        杨武，134 7965 7321，36
        王陆，165 7865 1354，34
        刘琦，173 6543 8654，45'''
```

如从以上文本中提取电话号码信息：

```
info_match = re.compile(r'(\d{3}.\d{4}.\d{4})')
for number in info_match.findall(text):
 print(number)
```

输出为：

```
158 4665 7365
178 6543 6789
134 7965 7321
165 7865 1354
173 6543 8654
```

## 贪婪模式和非贪婪模式

例如要把下面字符串中的所有 `html` 标签提取出来：

```
source = '<html><head><title>Title</title>'
```

得到如下列表：

```
['<html>', '<head>', '<title>', '</title>']
```

利用 `<.*>` 表示在 `<>` 中出现的任意次数的任意字符来表示一个 `html` 标签，理论上是可行的，下面是实际运行的结果：

```
<html><head><title>Title</title>
```

原因是，在正则表达式中 `'*', '+', '?'` 都是"贪婪的"，它们会尽可能多地匹配内容，所以用 `<.*>` 就表示从第一个 `<` 一直匹配到了最后一个 `>`。要得到期望输出，需采用**非贪婪模式**，在 `*` 后面加 `?`，变成 `<.*?>`。（控制"尽可能多"还是"尽可能少"）

```
source = '<html><head><title>Title</title>'
    tag_match = re.compile(r'(<.*?>)')
    print(tag_match.findall(source))
    for tag in tag_match.findall(source):
        print(tag)
```

输出变为：

```
['<html>', '<head>', '<title>', '</title>']
<html>
<head>
<title>
</title>
```

## 元字符的转义

对元字符本身进行匹配

```
text = '''
        苹果.是绿色的
        橙子.是橙色的
        香蕉.是黄色的'''
```

提取上面文本中所有在 `.` 之前的字符

```
extract_info = re.compile(r'(.*\.)')
 for element in extract_info.findall(text):
     print(element)
```

输出如下结果：

```
        苹果.
        橙子.
        香蕉.
```

> *注意在文本中如果包括tab缩进和空格，在用`.*`也能够被匹配。（见`正则表达：.`）