正则

主旨

今天学什么?

- 正则

学习目标

- 学会写正则表达式

正则

什么是正则?

- Python正则表达式，是一种基于字符串模式匹配的语法规则，其功能是在字符串中进行模式匹配和替换。

为什么用正则

- 但当问题变得复杂的时候，比如找出字符串中所有数字，用基本的字符串处理就不行了。

  这不再是一个固定的字符串匹配问题，而是一个模式，一种规则的匹配。为了解决这个问题，有位叫Stephen的大神在1951年提出了正则表达式。

  几乎任何一门通用编程语言都有专门的正则表达式的模块，正则表达式英文是regular expressesion，所以编程语言中的模块名字一般叫re，或者regex等。Python中的正则表达式处理模块是re。

  正则表达式就是为了找到符合某种模式的字符串，这些模式包括：是什么字符，重复多少次，在什么位置，有哪些额外的约束。请注意：这里的每一句话都对应了正则表达式中的一类语法。

  下面我们通过一些例子，来熟悉正则表达式，和写正则表达式的套路和正则表达式的语法。

```
# 假设有一段文字,要确定文本中是否包含数字123456:
text = '张三身高:178，体重：168，学号：123456，密码:9527'
# 我们可以用in运算符，也可以使用index函数:
target = '123456'
if target in text:
 print('找到了')
print(text.index(target))
```

应用场景

- 正则表达式通常用于处理文本数据，比如字符串的搜索、匹配、替换等。以下是一些常见的使用场景：

  1、数据验证：正则表达式可以用于验证输入的数据是否符合规定的格式，比如验证电子邮件地址、手机号码、身份证号码等等。

  2、数据清洗：在处理一些需要抽取特定信息的文本时，正则表达式可以用于清洗文本，去除不需要的字符或者格式，比如去除HTML标签、删除空格、统一格式等等。

  3、数据抽取：正则表达式可以用于从文本中抽取出符合某种格式的信息，比如提取网页中的链接、抽取电子邮件地址等等。

  4、数据转换：正则表达式可以用于将一种格式的文本转换为另一种格式，比如将日期格式转换为另一种日期格式、将金额转换为另一种货币符号等等。

  5、代码开发：正则表达式可以用于代码开发中，比如根据指定的格式匹配某些字符串、对代码进行格式化等等。

re模块

- 导入re正则库
- findall方法：
  在字符串中找到正则表达式所匹配的所有子串，并返回一个列表 ， 如果没有找到匹配的，则返回空列表
  -
```
a = "python123123145java"
print(re.findall("1", a))
```
- match方法
  re.match 尝试从字符串的起始位置匹配一个模式，匹配成功 返回的是一个匹配 对象（这个对象包含了我们匹配的信息），如果不是起始位置匹配成功的话，match()返回的是空，

  注意：match只能匹配到一个
  -
```
a = "python123123java"
print(re.match('python',a))
print(re.match('python',a).group())# 查看匹配的字符
print(re.match('123',a))
print(re.match('python',a).span())# 匹配字符的下标取值区间
```

- search方法
  re.search 扫描整个字符串,匹配成功 返回的是一个匹配对象（这个对象包含了我们匹配的信息）

  注意：search也只能匹配到一个，找到符合规则的就返回，不会一直往后找

  -

```
print(re.search('123',a))# yes
print(re.search('cc',a))# None
print(re.search('python',a).group())
print(re.search('python',a).span())
```

- re.match与re.search的区别： re.match只匹配字符串的开始位置找，如果字符串开始不符合正则表达式，则匹配失败， re.search：匹配整个字符串，如果一直找不到则，返回是空的，没有结果

六部曲

- 固定的字符
  - 代码说明：
    第1行，引入正则表达式模块re
    第3行，使用re的findall()方法找到所有符合模式的字符串，这里的模式就是123456，也就是说找到字符串中所有的123456。
    findall()方法的第1个参数是模式，第2个参数是要查找的字符串。
    模式中会有一些特殊字符，所以用r表示这是一个raw字符串，让Python不要去转义里面的特殊字符。
    上面程序的运行结果是：[123456]，因为整个字符串中就1个123456。

```
# 要求：确定字符串中是否有123456
import re
text = '张三身高:178，体重：168，学号：123456，密码:9527'
print(re.findall(r'123456', text))
```

- 某一类字符

- 单字符匹配 2

字符	描述
.	匹配任意1个字符(除了\n)
[]	匹配 [] 中列举的字符
\d	匹配数字，即0-9
\D	匹配非数字，即不是数字
\s	匹配空白，即 空格，tab键
\S	匹配非空白
\w	匹配单词符，即a-z、A-Z、0-9、_
\W	匹配非单词字符

- 重复某一类字符
  - 数量元字符

字符	描述
*	匹配前一个字符出现0次或者无限次，即可有可无
+	匹配前一个字符出现1次或者无限次，即至少有1次
?	匹配前一个字符出现0次或者1次，即要么有1次，要么没有
{m}	匹配前一个字符出现m次
{m,}	匹配前一个字符至少出现m次
{m,n}	匹配前一个字符出现从m到n次

    - 这个模式\d+在\d的后面增加了+号，表示数字可以出现1到多次，所以178等都符合它的要求。

```
#要求：找所有的数字，比如178，168，123456，1024等。
text = '张三身高:178，体重：168，学号：123456，密码:1024'
print(re.findall(r'\d+', text))
```

    -

```
# 组合其它数量元字符匹配应用
print(re.findall(r'\d*', text))
print(re.findall(r'\d?', text))
print(re.findall(r'\d{3}', text))
```

- 组合某一类字符

- \d{4}-\d{8}这是一个组合的模式，表示前面4个数字，中间一个横杠，后面8个数字。这里会匹配一个结果：0511-52152166

```
# 要求：找出座机号码
text = '''张三电话是18812345678，
          他还有一个电话号码是18887654321，
          他爱好的数字是01234567891，
          他的座机是：0511-52152166'''
print(re.findall(r'\d{4}-\d{8}', text))
```

- 多种情况分组字符
  - 分组匹配

字符	描述
竖线	匹配左右任意一个表达式
(ab)	将括号中的字符作为一个分组

    - 比上面有复杂了点，因为使用竖线(|)来表示"或"的关系，就是手机号码和电话号码都可以。这里会匹配三个结果：18812345678，18887654321，0511-52152166。

```
# 要求：找出手机号码或者座机号码
text = '''张三电话是18812345678，
          他还有一个电话号码是18887654321，
          他爱好的数字是01234567891，
          他的座机是：0511-52152166'''
print(re.findall(r'\d{4}-\d{8}|1\d{10}', text))
```

- 限定位置
  - 边界元字符

字符	描述
^	匹配字符串开头
$	匹配字符串结尾

- 在表达式的最开始使用了^符号，表示一定要在句子的开头才行。这时候只有18812345678能匹配上。

```
# 要求：在句子开头的手机号码,或座机
text = '''张三电话是18812345678，
          他还有一个电话号码是18887654321，
          他爱好的数字是01234567891，
          他的座机是：0511-52152166'''
print(re.findall(r'^1\d{10}|^\d{4}-\d{8}', text))
```

▸ 其他实例 1

如何写?

- 写正则表达式的步骤：
  如何写正则表达式呢？我总结了几个步骤。不管多复杂，基本上都百试不爽。
  我们仍然以包含分机号码的座机电话号码为例，比如0571-88776655-9527，演示下面的步骤：
  1,确定模式包含几个子模式(确定数据格式)
  它包含3个子模式：0571-88776655-9527。这3个子模式用固定字符连接。
  2,各个部分的字符分类是什么(确定数据格式有需要那些元字符)
  这3个子模式都是数字类型，可以用\d。现在可以写出模式为：\d-\d-\d
  3,各个子模式如何重复（确定元字符的修饰次数）
  第1个子模式重复3到4次，因为有010和021等直辖市
  第2个子模式重复7到8次，有的地区只有7位电话号码
  第3个子模式重复3-4次
  加上次数限制后，模式成为：\d{3,4}-\d{7,8}-\d{3,4}
  但有的座机没有分机号，所以我们用或运算符让它支持两者：
  \d{3,4}-\d{7,8}-\d{3,4}|\d{3,4}-\d{7,8}
  经过一通分析，最后的正则就写成了，测试一下：

```
text = '随机数字：01234567891，座机1：0571-52152166，座机2：0571-52152188-1234'
print(re.findall(r'\d{3,4}-\d{7,8}-\d{3,4}|\d{3,4}-\d{7,8}', text))
# 最后的结果是：两个座机都可以找出来。
```