# re模块操作

在Python中需要通过正则表达式对字符串进行匹配的时候，可以使用一个模块，名字为re

## 1. re模块的使用过程

#coding=utf-8

# 导入re模块

import re

# 使用match方法进行匹配操作

result = re.match(正则表达式,要匹配的字符串)

# 如果上一步匹配到数据的话，可以使用group方法来提取数据

result.group()

## 2. re模块示例(匹配以itcast开头的语句)

#coding=utf-8

import re

result = re.match("itcast","itcast.cn")

result.group()

运行结果为：

itcast

## 3. 说明

* re.match() 能够匹配出以xxx开头的字符串

# 匹配单个字符

在上一小节中，了解到通过re模块能够完成使用正则表达式来匹配字符串

本小节，将要讲解正则表达式的单字符匹配

| **字符** | **功能** |
| --- | --- |
| . | 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 | 匹配单词字符，即a-z、A-Z、0-9、\_ |
| \W | 匹配非单词字符 |

### 示例1： .

#coding=utf-8

import re

ret = re.match(".","M")

print(ret.group())

ret = re.match("t.o","too")

print(ret.group())

ret = re.match("t.o","two")

print(ret.group())

运行结果：

M

too

two

### 示例2：[ ]

#coding=utf-8

import re

# 如果hello的首字符小写，那么正则表达式需要小写的h

ret = re.match("h","hello Python")

print(ret.group())

# 如果hello的首字符大写，那么正则表达式需要大写的H

ret = re.match("H","Hello Python")

print(ret.group())

# 大小写h都可以的情况

ret = re.match("[hH]","hello Python")

print(ret.group())

ret = re.match("[hH]","Hello Python")

print(ret.group())

ret = re.match("[hH]ello Python","Hello Python")

print(ret.group())

# 匹配0到9第一种写法

ret = re.match("[0123456789]Hello Python","7Hello Python")

print(ret.group())

# 匹配0到9第二种写法

ret = re.match("[0-9]Hello Python","7Hello Python")

print(ret.group())

ret = re.match("[0-35-9]Hello Python","7Hello Python")

print(ret.group())

# 下面这个正则不能够匹配到数字4，因此ret为None

ret = re.match("[0-35-9]Hello Python","4Hello Python")

# print(ret.group())

运行结果：

h

H

h

H

Hello Python

7Hello Python

7Hello Python

7Hello Python

### 示例3：\d

#coding=utf-8

import re

# 普通的匹配方式

ret = re.match("嫦娥1号","嫦娥1号发射成功")

print(ret.group())

ret = re.match("嫦娥2号","嫦娥2号发射成功")

print(ret.group())

ret = re.match("嫦娥3号","嫦娥3号发射成功")

print(ret.group())

# 使用\d进行匹配

ret = re.match("嫦娥\d号","嫦娥1号发射成功")

print(ret.group())

ret = re.match("嫦娥\d号","嫦娥2号发射成功")

print(ret.group())

ret = re.match("嫦娥\d号","嫦娥3号发射成功")

print(ret.group())

运行结果：

嫦娥1号

嫦娥2号

嫦娥3号

嫦娥1号

嫦娥2号

嫦娥3号

**匹配多个字符**

匹配多个字符的相关格式

| **字符** | **功能** |
| --- | --- |
| \* | 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1次或者0次，即要么有1次，要么没有 |
| {m} | 匹配前一个字符出现m次 |
| {m,n} | 匹配前一个字符出现从m到n次 |

**示例1：\***

需求：匹配出，一个字符串第一个字母为大小字符，后面都是小写字母并且这些小写字母可有可无

#coding=utf-8

import re

ret = re.match("[A-Z][a-z]\*","M")

print(ret.group())

ret = re.match("[A-Z][a-z]\*","MnnM")

print(ret.group())

ret = re.match("[A-Z][a-z]\*","Aabcdef")

print(ret.group())

运行结果：

M

Mnn

Aabcdef

**示例2：+**

需求：匹配出，变量名是否有效

#coding=utf-8

import re

names = ["name1", "\_name", "2\_name", "\_\_name\_\_"]

for name in names:

ret = re.match("[a-zA-Z\_]+[\w]\*",name)

if ret:

print("变量名 %s 符合要求" % ret.group())

else:

print("变量名 %s 非法" % name)

运行结果：

变量名 name1 符合要求

变量名 \_name 符合要求

变量名 2\_name 非法

变量名 \_\_name\_\_ 符合要求

**示例3：?**

需求：匹配出，0到99之间的数字

#coding=utf-8

import re

ret = re.match("[1-9]?[0-9]","7")

print(ret.group())

ret = re.match("[1-9]?\d","33")

print(ret.group())

ret = re.match("[1-9]?\d","09")

print(ret.group())

运行结果：

7

33

0 # 这个结果并不是想要的，利用$才能解决

**示例4：{m}**

需求：匹配出，8到20位的密码，可以是大小写英文字母、数字、下划线

#coding=utf-8

import re

ret = re.match("[a-zA-Z0-9\_]{6}","12a3g45678")

print(ret.group())

ret = re.match("[a-zA-Z0-9\_]{8,20}","1ad12f23s34455ff66")

print(ret.group())

运行结果：

12a3g4

1ad12f23s34455ff66

# 匹配分组

| **字符** | **功能** |
| --- | --- |
| | | 匹配左右任意一个表达式 |
| (ab) | 将括号中字符作为一个分组 |
| \num | 引用分组num匹配到的字符串 |
| (?P<name>) | 分组起别名 |
| (?P=name) | 引用别名为name分组匹配到的字符串 |

## 示例1：|

需求：匹配出0-100之间的数字

#coding=utf-8

import re

ret = re.match("[1-9]?\d","8")

print(ret.group()) # 8

ret = re.match("[1-9]?\d","78")

print(ret.group()) # 78

# 不正确的情况

ret = re.match("[1-9]?\d","08")

print(ret.group()) # 0

# 修正之后的

ret = re.match("[1-9]?\d$","08")

if ret:

print(ret.group())

else:

print("不在0-100之间")

# 添加|

ret = re.match("[1-9]?\d$|100","8")

print(ret.group()) # 8

ret = re.match("[1-9]?\d$|100","78")

print(ret.group()) # 78

ret = re.match("[1-9]?\d$|100","08")

# print(ret.group()) # 不是0-100之间

ret = re.match("[1-9]?\d$|100","100")

print(ret.group()) # 100

## 示例2：( )

需求：匹配出163、126、qq邮箱

#coding=utf-8

import re

ret = re.match("\w{4,20}@163\.com", "test@163.com")

print(ret.group()) # test@163.com

ret = re.match("\w{4,20}@(163|126|qq)\.com", "test@126.com")

print(ret.group()) # test@126.com

ret = re.match("\w{4,20}@(163|126|qq)\.com", "test@qq.com")

print(ret.group()) # test@qq.com

ret = re.match("\w{4,20}@(163|126|qq)\.com", "test@gmail.com")

if ret:

print(ret.group())

else:

print("不是163、126、qq邮箱") # 不是163、126、qq邮箱

### 不是以4、7结尾的手机号码(11位)

import re

tels = ["13100001234", "18912344321", "10086", "18800007777"]

for tel in tels:

ret = re.match("1\d{9}[0-35-68-9]", tel)

if ret:

print(ret.group())

else:

print("%s 不是想要的手机号" % tel)

### 提取区号和电话号码

>>> ret = re.match("([^-]\*)-(\d+)","010-12345678")

>>> ret.group()

'010-12345678'

>>> ret.group(1)

'010'

>>> ret.group(2)

'12345678'

## 示例3：\

需求：匹配出<html>hh</html>

#coding=utf-8

import re

# 能够完成对正确的字符串的匹配

ret = re.match("<[a-zA-Z]\*>\w\*</[a-zA-Z]\*>", "<html>hh</html>")

print(ret.group())

# 如果遇到非正常的html格式字符串，匹配出错

ret = re.match("<[a-zA-Z]\*>\w\*</[a-zA-Z]\*>", "<html>hh</htmlbalabala>")

print(ret.group())

# 正确的理解思路：如果在第一对<>中是什么，按理说在后面的那对<>中就应该是什么

# 通过引用分组中匹配到的数据即可，但是要注意是元字符串，即类似 r""这种格式

ret = re.match(r"<([a-zA-Z]\*)>\w\*</\1>", "<html>hh</html>")

print(ret.group())

# 因为2对<>中的数据不一致，所以没有匹配出来

test\_label = "<html>hh</htmlbalabala>"

ret = re.match(r"<([a-zA-Z]\*)>\w\*</\1>", test\_label)

if ret:

print(ret.group())

else:

print("%s 这是一对不正确的标签" % test\_label)

运行结果：

<html>hh</html>

<html>hh</htmlbalabala>

<html>hh</html>

<html>hh</htmlbalabala> 这是一对不正确的标签

## 示例4：\number

需求：匹配出<html><h1>www.itcast.cn</h1></html>

#coding=utf-8

import re

labels = ["<html><h1>www.itcast.cn</h1></html>", "<html><h1>www.itcast.cn</h2></html>"]

for label in labels:

ret = re.match(r"<(\w\*)><(\w\*)>.\*</\2></\1>", label)

if ret:

print("%s 是符合要求的标签" % ret.group())

else:

print("%s 不符合要求" % label)

运行结果：

<html><h1>www.itcast.cn</h1></html> 是符合要求的标签

<html><h1>www.itcast.cn</h2></html> 不符合要求

## 示例5：(?P<name>) (?P=name)

需求：匹配出<html><h1>www.itcast.cn</h1></html>

#coding=utf-8

import re

ret = re.match(r"<(?P<name1>\w\*)><(?P<name2>\w\*)>.\*</(?P=name2)></(?P=name1)>", "<html><h1>www.itcast.cn</h1></html>")

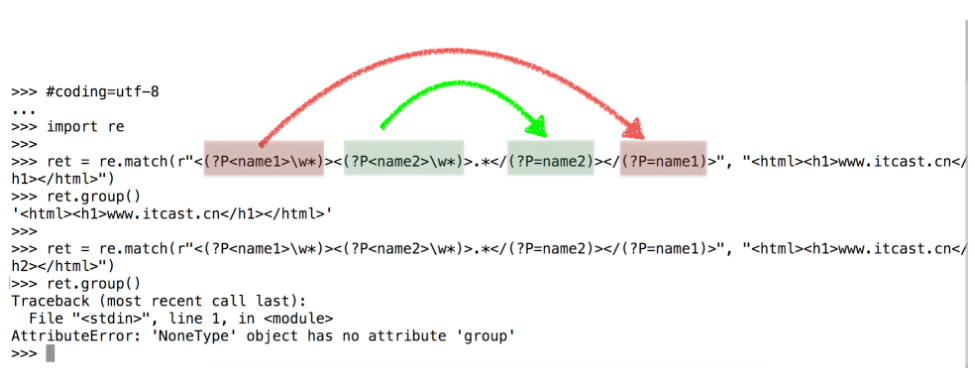
ret.group()

ret = re.match(r"<(?P<name1>\w\*)><(?P<name2>\w\*)>.\*</(?P=name2)></(?P=name1)>", "<html><h1>www.itcast.cn</h2></html>")

ret.group()

### 注意：(?P<name>)和(?P=name)中的字母p大写

运行结果：



# re模块的高级用法

## search

需求：匹配出文章阅读的次数

#coding=utf-8

import re

ret = re.search(r"\d+", "阅读次数为 9999")

ret.group()

运行结果：

'9999'

## findall

需求：统计出python、c、c++相应文章阅读的次数

#coding=utf-8

import re

ret = re.findall(r"\d+", "python = 9999, c = 7890, c++ = 12345")

print(ret)

运行结果：

['9999', '7890', '12345']

## sub 将匹配到的数据进行替换

需求：将匹配到的阅读次数加1

方法1：

#coding=utf-8

import re

ret = re.sub(r"\d+", '998', "python = 997")

print(ret)

运行结果：

python = 998

方法2：

#coding=utf-8

import re

def add(temp):

strNum = temp.group()

num = int(strNum) + 1

return str(num)

ret = re.sub(r"\d+", add, "python = 997")

print(ret)

ret = re.sub(r"\d+", add, "python = 99")

print(ret)

运行结果：

python = 998

python = 100

**从下面的字符串中取出文本**

<div>

<p>岗位职责：</p>

<p>完成推荐算法、数据统计、接口、后台等服务器端相关工作</p>

<p><br></p>

<p>必备要求：</p>

<p>良好的自我驱动力和职业素养，工作积极主动、结果导向</p>

<p>&nbsp;<br></p>

<p>技术要求：</p>

<p>1、一年以上 Python 开发经验，掌握面向对象分析和设计，了解设计模式</p>

<p>2、掌握HTTP协议，熟悉MVC、MVVM等概念以及相关WEB开发框架</p>

<p>3、掌握关系数据库开发设计，掌握 SQL，熟练使用 MySQL/PostgreSQL 中的一种<br></p>

<p>4、掌握NoSQL、MQ，熟练使用对应技术解决方案</p>

<p>5、熟悉 Javascript/CSS/HTML5，JQuery、React、Vue.js</p>

<p>&nbsp;<br></p>

<p>加分项：</p>

<p>大数据，数理统计，机器学习，sklearn，高性能，大并发。</p>

</div>

参考答案:

re.sub(r"<[^>]\*>|&nbsp;|\n", "", test\_str)

## split 根据匹配进行切割字符串，并返回一个列表

需求：切割字符串“info:xiaoZhang 33 shandong”

#coding=utf-8

import re

ret = re.split(r":| ","info:xiaoZhang 33 shandong")

print(ret)

运行结果：

['info', 'xiaoZhang', '33', 'shandong']

# python贪婪和非贪婪

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；

非贪婪则相反，总是尝试匹配尽可能少的字符。

在"\*","?","+","{m,n}"后面加上？，使贪婪变成非贪婪。

>>> s="This is a number 234-235-22-423"

>>> r=re.match(".+(\d+-\d+-\d+-\d+)",s)

>>> r.group(1)

'4-235-22-423'

>>> r=re.match(".+?(\d+-\d+-\d+-\d+)",s)

>>> r.group(1)

'234-235-22-423'

>>>

正则表达式模式中使用到通配字，那它在从左到右的顺序求值时，会尽量“抓取”满足匹配最长字符串，在我们上面的例子里面，“.+”会从字符串的启始处抓取满足模式的最长字符，其中包括我们想得到的第一个整型字段的中的大部分，“\d+”只需一位字符就可以匹配，所以它匹配了数字“4”，而“.+”则匹配了从字符串起始到这个第一位数字4之前的所有字符。

解决方式：非贪婪操作符“？”，这个操作符可以用在"\*","+","?"的后面，要求正则匹配的越少越好。

>>> re.match(r"aa(\d+)","aa2343ddd").group(1)

'2343'

>>> re.match(r"aa(\d+?)","aa2343ddd").group(1)

'2'

>>> re.match(r"aa(\d+)ddd","aa2343ddd").group(1)

'2343'

>>> re.match(r"aa(\d+?)ddd","aa2343ddd").group(1)

'2343'

>>>

字符串为:

<img data-original="https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973\_201611131917\_small.jpg" src="https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973\_201611131917\_small.jpg" style="display: inline;">

请提取url地址

参考答案

re.search(r"https://.\*?\.jpg", test\_str)

# r的作用

>>> mm = "c:\\a\\b\\c"

>>> mm

'c:\\a\\b\\c'

>>> print(mm)

c:\a\b\c

>>> re.match("c:\\\\",mm).group()

'c:\\'

>>> ret = re.match("c:\\\\",mm).group()

>>> print(ret)

c:\

>>> ret = re.match("c:\\\\a",mm).group()

>>> print(ret)

c:\a

>>> ret = re.match(r"c:\\a",mm).group()

>>> print(ret)

c:\a

>>> ret = re.match(r"c:\a",mm).group()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'NoneType' object has no attribute 'group'

>>>

说明

Python中字符串前面加上 r 表示原生字符串，

与大多数编程语言相同，正则表达式里使用"\"作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符"\"，那么使用编程语言表示的正则表达式里将需要4个反斜杠"\\"：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。

Python里的原生字符串很好地解决了这个问题，有了原生字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

>>> ret = re.match(r"c:\\a",mm).group()

>>> print(ret)

c:\a