正则表达式

动机

1. 文本处理已经成为计算机常见工作之一

2. 对文本的搜索，定位，提取的逻辑往往比较复杂

3. 为了解决上述问题，产生正则表达式技术

定义 ： 正则表达式即文本的高级匹配模式，提供搜索，替 代，获取等功能。本质是由一系列特属符号和字符 构成的字串，这个字串就是正则表达式。

特点

\* 方便进行检索和修改等文本操作

\* 支持语言众多

\* 灵活多样

目标

1.熟练掌握所有正则表达式符号

2.能够编写简单的正则表达式，读懂较复杂的正则表达式

3.能够使用python re模块操作正则表达式

正则表达式匹配手段

通过设定有特殊意义的符号，描述符号和字符的重复行为及位置特征来表示一类特定规则的字符串

python ---> re模块 处理正则表达式

re.findall(pattern,string)

pattern ： 以字符串形式传入一个正则表达式

string ： 要匹配的目标字符串

返回值 ： 得到列表，将目标字串中能用正则匹配的内容放 入列表

正则表达式元字符

1. 普通字符匹配

（除了后续讲解的特殊字符全是普通字符）

可以用普通字符来匹配对应的字符

In [11]: re.findall("abc",'abcdefghabi')

Out[11]: ['abc']

In [12]: re.findall("成都",'成都的街头走一走')

Out[12]: ['成都']

2. 或

元字符: |

匹配规则： 匹配符号两侧的正则表达式均可

In [14]: re.findall("ab|cd",'abcdefghabi')

Out[14]: ['ab', 'cd', 'ab']

3. 匹配单个字符

元字符： .

匹配规则： 匹配除 \n 外任意一个字符

f.o ---> foo fao f@o

In [18]: re.findall("w.o",'woo,wao is not wbo')

Out[18]: ['woo', 'wao', 'wbo']

4. 匹配开始位置

元字符： ^

匹配规则： 匹配目标字符串的开头位置

In [20]: re.findall("^Jame","Jame,how are you")

Out[20]: ['Jame']

5. 匹配结束位置

元字符 ： $

匹配规则： 匹配目标字符串的结束位置

In [23]: re.findall("py$","hello.py")

Out[23]: ['py']

6. 匹配重复

元字符 ： \*

匹配规则： 匹配前面出现的正则表达式0次或多次

fo\* ----> f fo fooooooooooooooooooo

In [31]: re.findall("ab\*","abcdefae&65abbbbbbbb")

Out[31]: ['ab', 'a', 'abbbbbbbb']

7. 匹配重复

元字符 ： +

匹配规则： 匹配前面出现的正则表达式1次或多次

ab+ ---》 ab abbbbb

In [33]: re.findall(".+py$","hello.py")

Out[33]: ['hello.py']

8. 匹配重复

元字符 ： ？

匹配规则： 匹配前面出现的正则表达式0次或1次

ab? ---> a ab

In [36]: re.findall("ab?","abcea,adsfabbbbbb")

Out[36]: ['ab', 'a', 'a', 'ab']

9. 匹配重复

元字符： {n}

匹配规则： 匹配前面的正则出现n次

ab{3} ---> abbb

In [39]: re.findall("ab{3}","abcea,adsfabbbbbb")Out[39]: ['abbb']

10. 匹配重复

元字符 ： {m,n}

匹配规则 ： 匹配前面的正则m-n次

ab{3,5} ---> abbb abbbb abbbbb

In [45]: re.findall("ab{3,5}","ab abbb abbbbabbbbbb")Out[45]: ['abbb', 'abbbb', 'abbbbb']

11. 匹配字符集合

元字符： [字符集]

匹配规则: 匹配字符集中任意一个字符

[abc123] --> a b c 1 2 3

In [46]: re.findall("[aeiou]","hello world")

Out[46]: ['e', 'o', 'o']

[0-9] [a-z] [A-Z] [0-9a-z]

In [47]: re.findall("^[A-Z][a-z]\*","Hello world")

Out[47]: ['Hello']

[\_abc0-9]

12. 匹配字符集

元字符： [^...]

匹配规则：匹配除了中括号中字符集字符之外的任意一个字符

In [50]: re.findall("[^0-9]+","hello1")

Out[50]: ['hello']

13. 匹配任意（非）数字字符

元字符 ： \d \D

匹配规则： \d 匹配任意一个数字字符 [0-9]

\D 匹配任意一个非数字字符 [^0-9]

In [52]: re.findall("1\d{10}","13717776561")

Out[52]: ['13717776561']

In [53]: re.findall("\D+","hello world")

Out[53]: ['hello world']

14. 匹配任意（非）普通字符

（数字字母下划线 普通utf-8字符）

元字符 ： \w \W

匹配规则： \w 匹配一个普通字符

\W 匹配一个非普通字符

In [54]: re.findall("\w+","hello$1")

Out[54]: ['hello', '1']

In [55]: re.findall("\W+","hello$1")

Out[55]: ['$']

15. 匹配(非)空字符

（空格， \r \n \t \v \f）

元字符： \s \S

匹配规则: \s 匹配任意空字符

\S 匹配任意非空字符

In [59]: re.findall("\w+\s+\w+","hello world")

Out[59]: ['hello world']

In [61]: re.findall("\S+","hello world")

Out[61]: ['hello', 'world']

16. 匹配起止位置

元字符： \A \Z

匹配规则： \A 匹配字符串开头位置 ^

\Z 匹配字符串结尾位置 $

In [63]: re.findall("\Ahi","hi,Tom")

Out[63]: ['hi']

In [2]: re.findall("is\Z",'This')

Out[2]: ['is']

完全匹配 ： 使用一个正则表达式可以匹配目标字符串的全 部内容

In [6]: re.findall("\A\w{5}\d{3}\Z",'abcde123')

Out[6]: ['abcde123']

17. 匹配（非）单词边界

（普通字符和非普通字符的交接位置认为是单词边界）

元字符 ： \b \B

匹配规则 ： \b 匹配单词边界位置

\B 匹配非单词边界位置

In [17]: re.findall(r"\bis\b",'This is a test')

Out[17]: ['is']

元字符总结：

匹配单个字符 ： a . [...] [^...] \d \D \w \W \s \S

匹配重复： \* + ？ {n} {m,n}

匹配位置： ^ $ \A \Z \b \B

其他 ： | （） \

正则表达式的转义

正则表达式特殊字符

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

在正则表达式中如果想匹配这些特殊字符需要进行转义

e.g.

In [23]: re.findall("\[\d+\]",'abc[123]')

Out[23]: ['[123]']

raw 字符串 ---》 原始字符串

特点 ： 对字符串中的内容不进行转义，即表达原始含义

r"\b" ---> \b

"\\b" ---> \b

e.g.

In [39]: re.findall("\\w+@\\w+\\.cn",'lvze@tedu.cn')

Out[39]: ['lvze@tedu.cn']

In [40]: re.findall(r"\w+@\w+\.cn",'lvze@tedu.cn')

Out[40]: ['lvze@tedu.cn']

贪婪和非贪婪

贪婪模式 ： 正则表达式的重复匹配，总是尽可能多的向后匹配内容。

\* + ？ {m,n}

贪婪 ---》 非贪婪（懒惰） 尽可能少的匹配内容

\*？ +? ?? {m,n}?

In [46]: re.findall(r"ab\*?",'abbbbbbbbbb')

Out[46]: ['a']

In [47]: re.findall(r"ab+?",'abbbbbbbbbb')

Out[47]: ['ab']

正则表达式分组

使用()可以为正则表达式建立子组，子组不会影响正则表达式整体的匹配内容，可以被看做是一个内部单元。

子组的作用：

1. 形成内部整体，该表某些元字符的行为

In [52]: re.search(r"(ab)+",'abababab').group()

Out[52]: 'abababab'

re.search(r"\w+@\w+\.(com|cn)",'lvze@tedu.cn').group()

2. 子组匹配内容可以被单独获取

re.search(r"\w+@\w+\.(com|cn)",'lvze@tedu.com').group(1)

Out[59]: 'com'

子组注意事项

\* 一个正则表达式中可以有多个子组，区分第一，第二。。。子组

\* 子组不要出现重叠的情况，尽量简单

捕获组和非捕获组 （命令组，未命名组）

格式 ： （?P<name>pattern）

re.search(r"(?P<dog>ab)+",'abababab').group()

作用 ： 1 方便通过名字区分每个子组

2 捕获组可以重复调用

格式：(?P=name)

e.g. (?P<dog>ab)cd(?P=dog) ===>

abcdab

In [69]: re.search(r"(?P<dog>ab)cdef(?P=dog)",'abcdefab').group()

Out[69]: 'abcdefab'

正则表达式的匹配原则

1. 正确性 能够正确匹配目标字符串

2. 唯一性 除了匹配的目标内容，尽可能不会有不需要的 内容

3. 全面性 对目标字符串可能的情况要考虑全面不漏

re模块的使用

regex = re.compile(pattern,flags = 0)

功能 ： 生成正则表达式对象

参数 ： pattern 正则表达式

flags 功能标志位，丰富正则表达式的匹配

返回值： 返回一个正则表达式对象

re.findall(pattern,string,flags = 0)

功能 ： 根据正则表达式匹配目标字串内容

参数 ： pattern 正则表达式

string 目标字符串

返回值： 列表 里面是匹配到的内容

如果正则表达式有子组，则只返回子组中的内容

regex.findall(string,pos,endpos)

功能 ： 根据正则表达式匹配目标字串内容

参数 ： string 目标字符串

pos,endpos : 截取目标字符串的起止位置进行匹 配，默认是整个字符串

返回值： 列表 里面是匹配到的内容

如果正则表达式有子组，则只返回子组中的内容

re.split(pattern,string,flags = 0)

功能 ： 通过正则表达式切割目标字符串

参数 ： pattern 正则

string 目标字串

返回值 ： 以列表形式返回切割后的内容

re.sub(pattern,replace,string,max,flags)

功能: 替换正则表达式匹配内容

参数： pattern 正则

replace 要替换的内容

string 目标字符串

max 设定最多替换几处

返回值 ： 替换后的字符串

re.subn(pattern,replace,string,max,flags)返回元组

功能和参数同sub

返回值多一个实际替换了几处

re.finditer(pattern,string,flags)

功能： 使用正则匹配目标字串

参数： pattern 正则

string 目标字串

返回值： 迭代对象 ----》 迭代内容为match对象

re.fullmatch(pattern,string,flags)

功能 ： 完全匹配一个字符串

参数： pattern 正则

string 目标字串

返回值：match对象，匹配到的内容

re.match(pattern,string,flags)

功能 ： 匹配一个字符串起始内容

参数： pattern 正则

string 目标字串

返回值：match对象，匹配到的内容

re.search(pattern,string,flags)

功能 ： 匹配第一个符合条件的字符串

参数： pattern 正则

string 目标字串

返回值：match对象，匹配到的内容

regex 对象的属性

flags 标志位数值

pattern 正则表达式

groups 子组个数

groupindex 获取捕获组字典，键为组名值是第几组

作业 ：

1. 读取一个文本

匹配文本中所有以大写字母开头的单词

匹配文本中所有数字

数字类型：234 -12 1.23 40% 1/2

2. 熟练元字符的使用规则

3. 将finditer match search sub split 使用regex对 象调用练习

4. 复习tftp文件服务器项目 （项目思路和项目代码）

end', 'endpos', 'expand', 'group', 'groupdict', 'groups', 'lastgroup', 'lastindex', 'pos', 're', 'regs', 'span', 'start', 'string'