### 正则表达式 import re :

正则表达式(re模块)是一种小型的,高度专业化的编程语言。(在python中)它内嵌在python中，并通过re模块实现。正则表达式模块被编译成一系列字节码，然后由用c编写的匹配引擎执行。

  正则就是处理字符串的。

1.普通字符：字符串本身的方法。比如字符串的 find(), split().

2. 元字符:   .^$\*+?{}[]|()\

import re

s = "hellopythonworldpydsonds"

print(re.findall("py..on", s)); # .通配符,可以表示任意字符(\n除外)

s = "pythonhelloworldpydsonds"

print(re.findall("^py..on", s)); # ^ 被模糊匹配的字符串必须在字符串的开头位置

s = "pythonhelloworlddspydson"

print(re.findall("py..on$", s)); # $ 被模糊匹配的字符串必须在字符串的末尾位置

s = "pythonheguannnlddspydson"

print(re.findall("guan\*", s)); # \* 匹配0至无穷次的字符, guan\*表示匹配 gua, guan, guann, guannn...(n可以是0个, 0到无穷个,贪婪匹配(尽可能匹配))

s = "pythonheguannnlddspydson"

print(re.findall("guan+", s)); # + 匹配1至无穷次的字符, guan+表示匹配 guan, guann, guannn...(n至少1个, 1到无穷个,贪婪匹配(尽可能匹配))

s = "pythonhegualddspydson"

print(re.findall("guan\*", s)); # \* 匹配0至无穷次的字符, guan\*表示匹配 gua, guan, guann, guannn...(n可以是0个, 0到无穷个,贪婪匹配(尽可能匹配))

s = "pythonhegualddspydson"

print(re.findall("guan+", s)); # + 匹配1至无穷次的字符, guan+表示匹配 guan, guann, guannn...(n至少1个, 1到无穷个,贪婪匹配(尽可能匹配))

s = "pythonhegualddspydson"

print(re.findall("guan?", s)); # ? 匹配0或1个字符, guan?表示只能匹配 gua, guan(只能匹配0个n,或1个n的字符,贪婪匹配(尽可能匹配))

# {0,}==\*, {1,}==+, {0,1}==?, 用{}可以表示,\*,+,?的效果

# 当然{}中可以是任意数字 {0,6}表示可以匹配0-6个字符, {6}表示只能匹配6个字符

s = "pythonheguannlddspydson"

print(re.findall("guan{6}", s)); # {6}表示只能匹配6个字符

s = "pythonheguannlddspydson"

print(re.findall("guan{0,6}", s)); # {0,6}表示可以匹配0-6个字符

s = "pythonheguannnlddspydson"

print(re.findall("guan\*?", s)); # 加?,实现惰性匹配(只匹配最少的)

s = "pythonheguannnlddspydson"

print(re.findall("guan+?", s)); # 加？,实现惰性匹配(只匹配最少的)

s = "pythonheguannnlddspydson"

print(re.findall("guan??", s)); # 加？(第一个？表示元字符,实现模糊匹配,第2个？表示惰性匹配),实现惰性匹配(只匹配最少的)

[] 表示字符集, 在字符集里有功能的符号: -(范围) ^(非) \(转义)

反斜杠后边跟元字符去除特殊功能,比如\.

反斜杠后边跟普通字符实现特殊功能,比如\d

\d 可以匹配任何十进制数,相当于[0-9]

\D 可以匹配任何非数字字符,相当于[^0-9]

\s 可以匹配任何空白字符,相当于[\t\n\r\f\v]

\S 可以匹配任何非空白字符,相当于[^\t\n\r\f\v]

\w 匹配任何字母数字字符,相当于[a-zA-Z0-9]

\W 匹配任何非字母数字字符,相当于[^a-zA-Z0-9]

\b 匹配一个特殊字符边界,比如空格, &, #等

s = "pythonheguanfnlddspydson"

print(re.findall("guan[a-z]", s)); # - [a-z],表示字母a到z之间的所有字母

s = "pythonheguan7nnlddspydson"

print(re.findall("guan[^a-z]", s)); # ^表示非, [^a-z],表示不是字母a到z之间的其他字符

s = "1-90+((888\*4)/2-5)"

print(re.findall("\d+",s))

print(re.findall("\D+",s))

s = "hello world python!"

print(re.findall("\s+",s))

s = "hello world python!"

print(re.findall("\S+",s))

s = "hello wor%ld PYT\*Hon! 43&52"

print(re.findall("\w+",s))

s = "hello wor%ld PYT\*Hon! 43&52&43^87"

print(re.findall("\W+",s))

s = "hello6world"

print(re.findall("hello\.world",s)) # "hello\.world"只能匹配"hello.world"

s = "hello I# I$ world LIST"

print(re.findall("I\\b", s)) # \b在ascii码中本身就有特殊含义,如果写成re.findall("I\b", s),那么最终传给re模块的就是转换过的字符而不是\b

s = "hello I\crld LIST"

print(re.findall("I\\\\c", s))# python解释器会转移一次,变成 "I\\c", 然后re模块再转移一次,变成"I\c"

 分组 "()"

s = "defabdefabdef"

print(re.findall("(def)+", s))

s = "liubei22guanyu21zhangfei20zhaoyun"

print(re.search("(?P<name>[a-z]+)\d+", s).group()) # search()只匹配第一个符合条件的

print(re.search("(?P<name>[a-z]+)(?P<age>\d+)", s).group("name")) # (?P<名称>)为分组起名称

print(re.search("(?P<name>[a-z]+)(?P<age>\d+)", s).group("age")) # (?P<名称>)为分组起名称

s = "567abcdfd545"

print(re.match("\d+", s).group()) # 同search, 不过只在字符串开始处进行匹配

print(re.split("[ |]", s)) # 分割 ['567', 'abcdf', 'd54', '5']

print(re.sub("\d+","a","12345deffun767ction98")) # 替换

print(re.subn("\d","a","12345deffun767ction98")) # 替换

it = re.finditer("\d+","124abd456def789uio") #返回一个迭代器

print(it.\_\_next\_\_().group())

print(it.\_\_next\_\_().group())

print(re.findall("www\.(baidu|souhu)\.com", "rtewww.baidu.comrete")) # 优先获取()匹配的内容, ['baidu']

print(re.findall("www\.(?:baidu|souhu)\.com", "rtewww.baidu.comrete")) # ?:去掉()的优先级