.匹配除换行符之外的任意字符。

\d匹配数字。

\D匹配非数字。

\s 匹配空白。

\S匹配非空白。

\w匹配单词字符，a-z A-Z 0-9

\W匹配非单词字符。

\*可以匹配0个或一个以上的字符。

+可以匹配1个及以上的字符，比如bevel+ed可以匹配beveled beveled bevellled。

tassel\*ed 和tassel+ed都可以匹配tasselld。

？表示匹配是非贪婪的，它既表示{0,1}(最少匹配0个，最多匹配1个)也表示量词是非贪婪的。\d+?表示只匹配一个数字。

^在起始处匹配。

$在结尾处匹配。

\A在起始处匹配。

\b在单词边界匹配，在其之前的字符必须是一个单词，而跟随其后的字符必须是一个单词，或者相反，前面是非单词，后面是单词。

\B 在非单词边界匹配。

\Z 在结尾处匹配。

(?=e)//如果表达式e在此断言处匹配，但没有超越此处，称为正前瞻，则匹配。

(?!e)//如果表达式e在此断言处不匹配，也没有超越此处，称为负前瞻，则匹配。

(?<=e)//如果表达式e恰在本断言之前匹配，称为正回顾，则匹配。

(?<!e)//如果表达式e恰在本断言之前不匹配，称为负回顾，则匹配。

我们需要可以匹配两个或更多方案中的任意一种的正则表达式，通常还需要捕获匹配的内容以备进一步处理。此外我们还需要某个量词可以应用于几个表达式。所有这些目的都有可以使用()进行组合完成。在多种方案中选择一个的情况下则使用交替字符|。

s1="air craft aircraft airplane jet"

print (re.findall(r"(air(craft|plane)|jet)",s1))//输出[('aircraft', 'craft'), ('airplane', 'plane'), ('jet', '')]

print (re.findall(r"(air(?:craft|plane)|jet)",s1))//关闭第二次捕获的craft plane等。

print (re.findall(r"(aircraft|airplane)",s1))// ['aircraft', 'airplane']

对前面的捕获，可以通过反向引用对其进行引用，也就是说，对前面的捕获组进行引用。反向引用的语法格式是在正则表达式本身内部使用\i,这里i是前面的捕获组号，每次捕获都会进行编号，从1开始，以1为单位递增，方向从左至右，每次新捕获到一个左圆括号时都进行一次新的捕获编号。(\w+)\s+\1表示捕获一个单词后，再捕获相同的一个。

在较长的或复杂的正则表达式中，对捕获进行命名而不是使用编号通常会更加方便，而且也使得维护更加方便简单。为捕获命名可以在左圆括号后面跟随?P<name>，比如：(?P<key>\w+)=(?P<value>+)包含两次捕获，一次名为”key”,一次名为”value”在正则表达式中，用于对命名的捕获进行反向引用的语法是(?P=name).

正则表达式模块的标记：

re.A//使\b\B\s\S\w\W都假定字符串为ASCII。

re.I//使正则表达式以大小写不敏感的方式匹配。

re.M//使^在起始处并在每个换行符后匹配，使$在结尾处但在每个换行符之前匹配。

re.S//使.匹配每个字符，包括换行符。

re.X//使空白与注释包含在匹配中。

new\_name=[]//将最后一个名字放在前面，并用逗号隔开，输出：['bryant,kobe', 'kevin,duncan tim derik nowitzki', 'rose,durant kevin nowitzki']

names=["kobe bryant","duncan tim derik nowitzki kevin","durant kevin nowitzki rose"]

for name in names:

#name=re.sub(r"(\w+(?:\s+\w+)\*)\s+(\w+)",r"\2,\1",name)

name=re.sub(r"(?P<forename>\w+(?:\s+\w+)\*)"

r"\s+(?P<surname>\w+)",

r"\g<surname>,\g<forename>",name)//两种方式都可以

new\_name.append(name)

print(new\_name)

def comment\_check(line1):

if(re.search(r"^[\*][\*][/]",line1)!=None)://匹配\*\*/

return True

else:

return False