Python用" \\\\ "表示正则表达式中的" \ "。原始字符串(raw string) 用'r'作为字符串的前缀，如 r"\n"：表示两个字符"\"和"n"

**元字符：”\”+”A”表示开头（r’\A’==’\\A’）**

**\A Matches only at the start of the string.**

**\Z Matches only at the end of the string.**

**\b Matches the empty string, but only at the start or end of a word.**

**\B Matches the empty string, but not at the start or end of a word.**

**\d Matches any decimal digit; equivalent to the set [0-9]**

**\D Matches any non-digit character; equivalent to [^\d].**

**\s Matches any whitespace character; equivalent to [ \t\n\r\f\v]**

**\S Matches any non-whitespace character; equivalent to [^\s].**

**\w Matches any alphanumeric character; equivalent to [a-zA-Z0-9\_]**

**\W Matches the complement of \w.**

**\\ Matches a literal backslash.**

**(?P<name>...) The substring matched by the group is accessible by name.**

**(?P=name) Matches the text matched earlier by the group named name.**

**(?=...) Matches if ... matches next, but doesn't consume the string. #前向匹配**

**(?!...) Matches if ... doesn't match next.**

**(?<=...) Matches if preceded by ... (must be fixed length). #后向匹配**

**(?<!...) Matches if not preceded by ... (must be fixed length).**

**不贪婪的限定符 \*?、+?、?? 或 {m,n}?，尽可能匹配小的文本**

**compile**

**re.compile(pattern[, flags])**

#将一个字符串编译成 regular pattern object（regex），用于匹配或搜索

re 所定义的 flag 包括：

re.I 忽略大小写（ignorecase）

re.L 表示特殊字符集 \w, \W, \b, \B, \s, \S 依赖于当前环境（locale）

re.M 多行模式（multiline）

re.S 即为’ . ’并且包括换行符在内的任意字符（’ . ’不包括换行符）

re.U 表示特殊字符集 \w, \W, \b, \B, \d, \D, \s, \S 依赖于 Unicode 字符属性数据库

re.X 为了增加可读性，忽略空格和’ # ’后面的注释

**re.search(pattern, string, flags=0)**

**regex.search(string[, pos[, endpos]])**

在**字符串中**查找匹配正则表达式模式的位置，返回 MatchObject 的实例，如果没有找到匹配的位置，则返回 None。

#group()用来提出分组截获的字符串，group(0)指所有匹配的内容，group(1)指第一个()匹配内容

**re.match(pattern, string, flags=0)**

**字符串的开头**是否能匹配正则表达式

>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, physicist")

>>> m.group(0) # The entire match

'Isaac Newton'

>>> m.group(1) # The first parenthesized subgroup.

'Isaac'

>>> m.group(2) # The second parenthesized subgroup.

'Newton'

>>> m.group(1, 2) # Multiple arguments give us a tuple.

('Isaac', 'Newton')

**regex.match(string[, pos[, endpos]])**

>>>pattern = re.compile("ar{1}")

>>>pattern.match("army") # "ar"在开头，匹配成功

<\_sre.SRE\_Match object; span=(0, 2), match='ar'>

>>>pattern.match("mary") # "ar"不在开头，匹配失败

None

>>>pattern.match("mary", 1) # "ar"不在开头，但在子串的开头

<\_sre.SRE\_Match object; span=(1, 3), match='ar'>

**#re.match() 函数只在字符串的开始位置尝试匹配正则表达；re.search() 函数是扫描整个字符串来查找匹配。**

#匹配多个字符：**r'(\[|\]|\')'**

**re.split(pattern, string[, maxsplit=0, flags=0])**

**regex.split(string, maxsplit=0)**

将字符串匹配正则表达式的部分割开并返回一个list。如果用括号将正则表达式括起来，那么匹配的字符串也会被列入到list中返回

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

**regex.findall(string[, pos[, endpos]])**

在字符串中找到正则表达式所匹配的所有子串，并组成一个**列表**返回

**re.finditer(pattern, string, flags=0)**

**regex.finditer(string[, pos[, endpos]])**

在字符串中找到正则表达式所匹配的所有子串，并组成一个**迭代器**返回

**re.sub(pattern, repl, string[, count, flags])**

**regex.sub(repl, string, count=0) #** **count 用于指定最多替换次数**

在字符串 string 中找到匹配正则表达式 pattern 的所有子串，用另一个字符串 repl 进行替换。如果没有找到匹配 pattern 的串，则返回未被修改的 string。

**re.subn(pattern, repl, string[, count, flags])**

**regex.subn(repl, string, count=0)**

该函数的功能和 sub() 相同，但它还**返回新的字符串以及替换的次数**

re.escape(string)

对字符串中的非字母数字进行转义

re.purge()

清空缓存中的正则表达式