Python 正規表示式

中介字元	說明
.	除了新行符號外的任何字元，例如 '.' 配對除了 '\n' 之外的任何字元。
^	字串開頭的子字串或排除指定字元或群組，例如 'a[^b]c' 配對除了 'abc' 之外的任何 a 開頭 c 結尾的三字元組合。
$	字串結尾的子字串，例如 'abc$' 配對以 'abc' 結尾的字串。
*	單一字元或群組出現任意次數，例如 'ab*' 配對 'a' 、 'ab' 或 'abb' 等等。
+	單一字元或群組出現至少一次，例如 'ab+' 配對 'ab' 或 'abb' 等等。
?	單一字元或群組 0 或 1 次，例如 'ab+' 配對 'a' 或 'ab' 。
{m,n}	單一字元或群組的 m 到 n 倍數，例如 'a{6}' 為連續六個 'a' ， 'a{3,6}' 為三到六個 'a' 。
[]	對中括弧內的字元形成集合，例如 '[a-z]' 為所有英文小寫字母。
\	特別序列的起始字元。
|	單一字元或群組的或，例如 'a|b' 為 'a' 或 'b' 。
()	對小括弧內的字元形成群組。

特別序列	說明
\number	群組的序數
\A	字串的開頭字元。
\b	作為單字的界線字元，例如 r'\bfoo\b' 配對 'foo' 或 'bar foo baz' 。
\B	作為字元的界線字元，例如 r'py\B' 配對 'python' 或 'py3' 。
\d	數字，從 0 到 9 。
\D	非數字。
\s	各種空白符號，包括新行符號 \n 。
\S	非空白符號。
\w	任意文字字元，包括數字。
\W	非文字字元，包括空白符號。
\Z	字串的結尾字元。

```
import re
txt = "The rain in Spain"
x = re.findall("[a-m]", txt)
print(x)
```

```
import re
txt = "That will be 59 dollars"
x = re.findall("\d", txt)
print(x)
```

```
import re
txt = "hello world"
x = re.findall("he..o", txt)
print(x)
```

```
import re
txt = "hello world"
x = re.findall("^hello", txt)
if (x):
  print("Yes, the string starts with 'hello'")
else:
  print("No match")
```

```
import re
txt = "The rain in Spain falls mainly in the plain!"
x = re.findall("aix+", txt)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
import re
txt = "The rain in Spain falls mainly in the plain!"
x = re.findall("al{2}", txt)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
import re
txt = "The rain in Spain falls mainly in the plain!"
x = re.findall("falls|stays", txt)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

```
import re
txt = "hello world"
x = re.findall("world$", txt)
if (x):
  print("Yes, the string ends with 'world'")
else:
  print("No match")
```

```
import re
txt = "The rain in Spain falls mainly in the plain!"
x = re.findall("aix*", txt)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
```

```
  print("No match")
```

```
import re
txt = "The rain in Spain falls mainly in the plain!"
x = re.findall("aix+", txt)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

.	表示任意字元，如果說指定了 DOTALL 的標識，就表示包括新行在內的所有字元。	‘abc’ >>>’a.c’ >>>結果為:’abc’
^	表示字串開頭。	‘abc’ >>>’^abc’ >>>結果為:’abc’
$	表示字串結尾。	‘abc’ >>>’abc$’ >>>結果為:’abc’
*, , ?	‘*’ 表示匹配前一個字元重複 0 次到無限次，’ ‘表示匹配前一個字元重複 1 次到無限次，’ ?’ 表示匹配前一個字元重複 0 次到 1 次	‘abcccd’ >>>’abc*’ >>>結果為:’abccc’ ‘abcccd’ >>>’abc ’ >>>結果為:’abccc’ ‘abcccd’ >>>’abc?’ >>>結果為:’abc’
*?, ?, ??	前面的*, ,?等都是貪婪匹配，也就是儘可能多匹配，後面加?號使其變成惰性匹配即非貪婪匹配	‘abc’ >>>’abc*?’ >>>結果為:’ab’ ‘abc’ >>>’abc??’ >>>結果為:’ab’ ‘abc’ >>>’abc ?’ >>>結果為:’abc’
{m}	匹配前一個字元 m 次	‘abcccd’ >>>’abc{3}d’ >>>結果為:’abcccd’
{m,n}	匹配前一個字元 m 到 n 次	‘abcccd’ >>> ‘abc{2,3}d’ >>>結果為:’abcccd’
{m,n}?	匹配前一個字元 m 到 n 次，並且取儘可能少的情況	‘abccc’ >>> ‘abc{2,3}?’ >>>結果為:’abcc’
\	對特殊字元進行轉義，或者是指定特殊序列	‘a.c’ >>>’a\.c’ >>> 結果為: ‘a.c’
[]	表示一個字符集,所有特殊字元在其都失去特殊意義,只有： ^ –] \ 含有特殊含義	‘abcd’ >>>’a[bc]’ >>>結果為:’ab’
|	或者，只匹配其中一個表示式 ，如果|沒有被包括在()中，則它的範圍是整個正規表示式	‘abcd’ >>>’abc|acd’ >>>結果為:’abc’
(…)	被括起來的表示式作為一個分組. findall 在有組的情況下只顯示組的內容	‘a123d’ >>>’a(123)d’ >>>結果為:’123’
(?#…)	註釋，忽略括號內的內容 特殊構建不作為分組	‘abc123’ >>>’abc(?#fasd)123’ >>>結果為:’abc123’
(?= …)	表示式’…’之前的字串，特殊構建不作為分組	在字串’ pythonretest ’ 中 (?=test) 會匹配’ pythonre ’
(?!…)	後面不跟表示式’…’的字串，特殊構建不作為分組	如果’ pythonre ’ 後面不是字串’ test ’ ，那麼 (?!test) 會匹配’ pythonre ’
(?<= …)	跟在表示式’…’後面的字串符合括號之後的正規表示式，特殊構建不作為分組	正規表示式’ (?<=abc)def ’ 會在’ abcdef ’ 中匹配’ def ’
(?:)	取消優先列印分組的內容	‘abc’ >>>’(?:a)(b)’ >>>結果為’[b]’
?P<>	指定 Key	‘abc’ >>>’(?P<n1>a)>>>結果為:groupdict{n1:a}

```
import re

strTest = "abcccd"
mobj = re.match(r"abc{3}d", strTest, re.I)
print(mobj.group())
```

函數	說明
compile(pattern)	以配對形式字串 pattern 當參數，回傳 re.compile() 物件。
search(pattern, string, flags=0)	從 string 中找尋第一個配對形式字串 pattern ，找到回傳配對物件，沒有找到回傳 None 。
match(pattern, string, flags=0)	判斷配對形式字串 pattern 是否與 string 的開頭相符，如果相符就回傳配對物件，不相符就回傳 None 。
fullmatch(pattern, string, flags=0)	判斷 string 是否與配對形式字串 pattern 完全相符，如果完全相符就回傳配對物件，不完全相符就回傳 None 。
split(pattern, string, maxsplit=0, flags=0)	將 string 以配對形式字串 pattern 拆解，結果回傳拆解後的串列。
findall(pattern, string, flags=0)	從 string 中找到所有的 pattern ，結果回傳所有 pattern 的串列。
finditer(pattern, string, flags=0)	從 string 中找到所有的 pattern ，結果回傳所有 pattern 的迭代器。
sub(pattern, repl, string, count=0, flags=0)	依據 pattern 及 repl 對 string 進行處理，結果回傳處理過的新字串。
subn(pattern, repl, string, count=0, flags=0)	依據 pattern 及 repl 對 string 進行處理，結果回傳處理過的序對。
escape(pattern)	將 pattern 中的特殊字元加入反斜線，結果回傳新字串。
purge()	清除正規運算式的內部緩存

標誌位	說明
re.I	在匹配時忽略字串和模式的大小寫
re.L	匹配 {\w \W \b \B} 跟本地語言相關。不推薦使用
re.M	$ 匹配行末尾，而不是字串末尾，同理^匹配行開頭而不是字串開頭
re.S	. 匹配任何字元，也包括新的一行
re.U	使用 Unicode 字符集
re.X	忽略各種空格以及以 # 開頭的註釋，這使得長匹配模式可以分行來寫，提高了可讀性

re.match(pattern, string, flags)
pattern 是要匹配的正規表示式模式
string 是與正規表示式匹配的給定字串
flags 用於更改正規表示式的行為，這是個可選項
如果匹配成功就返回 Match 物件，否則返回 NONE。

```
import re
strTest = "Hello Python Programming"
mobj = re.match(r"hello", strTest, re.I)
print(mobj.group())
#Hello
```

```
import re
str = "\\tHello Python Programming"
mobj = re.match("\\thello", str, re.I) #no match
```

```
str = "\tHello Python Programming"
mobj = re.match("\\thello", str, re.I) #\thello is matching
```

```
import re
str = "Hello Python Programming"
sobj = re.search(r"programming", str, re.I)
print(sobj.group())
#Programming
```

```
import re
str = "Hello Python Programming"
sobj = re.search(r"^programming", str, re.I)
print(sobj.group()) #no match is found

sobj = re.search(r"^hello", str, re.I)
print(sobj.group()) #matching: Hello
```

```
import re
str = "Hello Python Programming"
sobj = re.search(r"programming$", str, re.I)
print(sobj.group()) #matching: Programming

sobj = re.search(r"hello$", str, re.I)
print(sobj.group()) #no match found
```

```
import re
compPat = re.compile(r"(\d)")
sobj = compPat.search("Lalalala 123")
print(mobj.group())

mobj = compPat.match("234Lalalala 123456789")
print(mobj.group())
```

```
import re
s = re.search("L", "Hello")
print(s)            #Output: None

s = re.search("L", "Hello", re.I)
print(s)            #Output: 1

s = re.search("L", "^Hello", re.I | re.M)
print(s)            #Output: 1
```

```
import re
def check(str):
    s = re.compile(r'[^A-Z]')
    str = s.search(str)
    return not bool(str)
```

```
print(check("HELLOPYTHON"))          # 輸出: True
print(check("hellopython"))          # 輸出: False
```

```
import re
s = "Playing 4 hours a day"
obj = re.sub(r'^.*$',"Working",s)
print(obj)
```

```
import re
s = "768 Working 2343 789 five 234 656 hours 324 4646 a 345 day"
obj = re.sub(r'\d',"",s)
print(obj)
```

```
import re
s = "768 Working 2343 789 five 234 656 hours 324 4646 a 345 day"
obj = re.sub(r'\D',"",s)
print(obj)
```

```
import re
str = "Working 6 hours a day. Studying 4 hours a day."
mobj = re.findall(r'[0-9]', str)
print(mobj)
```

```
import re
file = open('asd.txt', 'r')
mobj = re.findall(r'arg.', file.read())
print(mobj)
file.close()
```

```
import re
str = "Working 6 hours a day. Studying 4 hours a day."
pat = r'[0-9]'
for mobj in re.finditer(pat, str):
    s = mobj.start()
    e = mobj.end()
    g = mobj.group()
    print('{} found at location [{},{}]'.format(g, s, e))
```

```
import re
str = "Birds fly high in the sky for ever"
mobj = re.split('\s+', str, 5)
print(mobj)
```

相關資料參考

https://www.w3schools.com/python/python_regex.asp

https://www.delftstack.com/zh-tw/tutorial/python-modules-tutorial/python-regular-expression/