1, Regular Expression là gì?

Regular expression (dịch ra tiếng việt là biểu thức chính quy) là một đoạn các ký tự đặc biệt dùng để so khớp các chuỗi hoặc một tập các chuỗi. Bạn cũng có thể gọi Regular Expression là một ngôn ngữ. Và hầu như ngôn ngữ lập trình nào cũng hỗ trợ Regular expression, nhưng được sử dụng nhiều nhất và phổ biến nhất là trong UNIX.

2, Regular Expression trong Python.

Regular Expression trong Python được thể hiện qua module re, nên việc đầu tiên khi các bạn muốn sử dụng regular expression thì cần phải import module re vào chương trình, sử dụng với cú pháp như sau:

import re

Trong module này thì có rất nhiều các phương thức, và mình sẽ liệt kê một số phương thức hay được sử dụng kèm theo ví dụ với mỗi phương thức đó.

group(num)

Phương thức này trả về các giá trị so khớp giữa biểu thức chính quy và chuỗi cần so. Trong đó thì num này là vị trí index của list so khớp trả về mà bạn muốn lấy ra. Nếu không điền thì Python sẽ coi nó là 0.

**VD**:

import re

str = 'Học lập trình Toidicode.com'

match = re.search(r'Toidicode.com', str)

if match: #nếu tồn tại chuỗi khớp

print (match.group()) # in ra chuỗi đó

else:

print ('Khong tim thay!') # Không thì hiện thông báo

#Kết quả:

#Toidicode.com

groups()

Phương thức này sẽ trả về một tupel các chuỗi được so khớp khớp.

**VD**:

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.search(r'(.\*) Toidicode.com', str)

if match: #nếu tồn tại chuỗi khớp

print (match.groups()) # in ra chuỗi đó

else:

print ('Khong tim thay!') # Không thì hiện thông báo

#Kết quả:

#('Hoc lap trinh',)

search(pattern, string, flags)

Phương thức này thực hiện tìm kiếm chuỗi so khớp trên string và nó sẽ trả về các giá trị được so khớp.

**Trong đó**:

* pattern là chuỗi regular expression.
* string là chuỗi cần so khớp.
* flags là các kiểu so khớp mà bạn muốn thực hiện (xem ở dưới). Nếu để trống thì flags sẽ bằng 0.

**VD**:

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.search(r'Toidicode.com', str)

if match: #nếu tồn tại chuỗi khớp

print (match.group()) # in ra chuỗi đó

else:

print ('Khong tim thay!') # Không thì hiện thông báo

#Kết quả:

#Toidicode.com

match(pattern, string, flags)

Phương thức này cũng thực hiện việc so khớp chuỗi nhưng nó sẽ tính từ phạm vi đầu chuỗi cho đến kết thúc, còn các tham số truyền vào thì tương tự như đối với phương thức search().

**VD**:

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.match(r'Toidicode.com', str)

if match: #nếu tồn tại chuỗi khớp

print (match.group()) # in ra chuỗi đó

else:

print ('Khong tim thay!') # Không thì hiện thông báo

# Kết quả:

# Khong tim thay!

fullmatch(pattern, string, flags)

Phương thức này cũng thực hiện việc so khớp nhưng nó là so khớp hoàn toàn, còn các tham số truyền vào là tương tự như phương thức match.

**VD**: Đây là ví dụ phân biệt sự khác nhau giữa match và fullmatch.

+ fullmatch

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.fullmatch(r'Hoc lap trinh', str)

if match: #nếu tồn tại chuỗi khớp

print (match.group()) # in ra chuỗi đó

else:

print ('Khong tim thay!') # Không thì hiện thông báo

# Kết quả:

# Khong tim thay!

+ match

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.match(r'Hoc lap trinh', str)

if match: #nếu tồn tại chuỗi khớp

print (match.group()) # in ra chuỗi đó

else:

print ('Khong tim thay!') # Không thì hiện thông báo

# Kết quả:

# Hoc lap trinh

split(pattern, string, maxsplit)

Hàm này có tác dụng so khớp và cắt chuỗi so khớp thành công.

**Trong đó**:

* pattern là chuỗi regular expression.
* string là chuỗi cần so khớp.
* maxsplit là số chuỗi tối đa sẽ được split. Nếu để trống thì Python sẽ tìm và cắt tất cả các chuỗi

**VD**:

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.split(r'i', str)

if match: #nếu tồn tại chuỗi khớp

print (match) # in ra kết quả

else:

print ('Khong tim thay!') # Không thì hiện thông báo

# Kết quả:

# ['Hoc lap tr', 'nh To', 'd', 'code.com']

findall(partern, string, flags)

Phương thức này có tác dụng so khớp và trả về tất cả các chuỗi mà nó đã so khơp được, còn lại các tham số truyền và sử dụng như đối với phương thức search.

**VD**:

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.findall(r'i', str)

if match: #nếu tồn tại chuỗi khớp

print (match) # in ra kết quả

else:

print ('Khong tim thay!') # Không thì hiện thông báo

# Kết quả:

# ['i', 'i', 'i']

sub(pattern, replace, string, flags)

Phương thức này có tác dụng so khớp và thay thế chuỗi so khớp được.

**Trong đó**:

* pattern là chuỗi regular expression.
* replace là chuỗi thay thế cho các chuỗi được so khớp.
* string là chuỗi cần so khớp.
* flags là các kiểu so khớp mà bạn muốn thực hiện (xem ở dưới). Nếu để trống thì flags sẽ bằng 0.

**VD**:

import re

str = 'Hoc lap trinh Toidicode.com'

match = re.sub(r'Toidicode.com','https://toidicode.com', str)

if match: #nếu tồn tại chuỗi khớp

print (match) # in ra kết quả

else:

print ('Khong tim thay!') # Không thì hiện thông báo

# Kết quả:

# Hoc lap trinh https://toidicode.com

1, Các cú pháp pattern trong Python.

## Các Pattern trong Python

Ngoại trừ các ký tự điều khiển, **(+ ? . \* ^ $ ( ) [ ] { } | \)**, thì tất cả ký tự còn lại sẽ kết nối với chính chúng. Bạn có thể tránh một ký tự điều khiển bằng cách đặt trước nó một dấu \.

Dưới đây là danh sách các pattern có sẵn trong Python:

|  |  |
| --- | --- |
| **Pattern** | **Miêu tả** |
| ^ | Dưới đây là danh sách các pattern có sẵn trong Python: |
| $ | Kết nối với phần cuối của dòng |
| . | Kết nối bất kỳ ký tự đơn nào ngoại trừ newline (dòng mới). Sử dụng tùy chọn m cho phép nó kết nối với newline (dòng mới) |
| [...] | Kết nối với bất kỳ ký tự đơn nào trong [] |
| [^...] | Kết nối với bất kỳ ký tự đơn nào không ở trong [] |
| re\* | Kết nối với 0 hoặc nhiều sự xuất hiện của biểu thức đặt trước |
| re+ | Kết nối với 1 hoặc nhiều sự xuất hiện của biểu thức đặt trước |
| re? | Kết nối với 0 hoặc 1 sự xuất hiện của biểu thức đặt trước |
| re{ n} | Kết nối với n sự xuất hiện của biểu thức đặt trước |
| re{ n,} | Kết nối với n hoặc nhiều hơn sự xuất hiện của biểu thức đặt trước |
| re{ n, m} | Kết nối với ít nhất n và nhiều nhất m sự xuất hiện của biểu thức đặt trước |
| a| b | Kết nối với a hoặc b |
| (re) | Nhóm các Regular Expression và ghi nhớ text đã kết nối |
| (?imx) | Bật toggle tạm tời các tùy chọn i, m hoặc x bên trong một Regular Expression. Nếu trong cặp dấu ngoặc đơn thì chỉ khu vực đó bị ảnh hưởng |
| (?-imx) | Tắt toggle tạm tời các tùy chọn i, m hoặc x bên trong một Regular Expression. Nếu trong cặp dấu ngoặc đơn thì chỉ khu vực đó bị ảnh hưởng |
| (?: re) | Nhóm các Regular Expression mà không ghi nhớ text đã kết nối |
| (?imx: re) | Bật toggle tạm thời các tùy chọn i, m, hoặc x bên trong cặp dấu ngoặc đơn |
| (?-imx: re) | Tắt toggle tạm thời các tùy chọn i, m, hoặc x bên trong cặp dấu ngoặc đơn |
| (?#...) | Comment |
| **(?= re)** | Xác định vị trí bởi sử dụng một pattern. Không có một dãy giá trị |
| (?! re) | Xác định vị trí bởi sử dụng sự phủ định pattern. Không có một dãy giá trị |
| (?> re) | Kết nối pattern độc lập mà không backtrack |
| \w | Kết nối các ký tự từ |
| \W | Kết nối các ký tự không phải là từ |
| \s | Kết nối với whitespace. Tương đương với [\t\n\r\f] |
| \S | Kết nối với các ký tự không là whitespace |
| \d | Kết nối với các chữ số. Tương đương với [0-9] |
| \D | Kết nối với các ký tự không là chữ số |
| \A | Kết nối với phần đầu của chuỗi |
| \Z | Kết nối với phần cuối của chuỗi. Nếu một newline (dòng mới) tồn tại, nó kết nối với phần ở trước newline (dòng mới) |
| \z | Kết nối với phần cuối của chuỗi |
| \G | Kết nối với điểm mà tại đó kết nối cuối cùng được tìm thấy |
| \b | Kết nối với các giới hạn từ khi bên ngoài các dấu []. Kết nối với backspace (mã là 0x08) khi bên trong các dấu [] |
| \B | Kết nối với các giới hạn không phải là từ |
| \n, \t, etc. | Kết nối với newline (dòng mới), carriage return, tab, ... |
| \1...\9 | Kết nối với biểu thức con được nhóm thứ n |
| \10 | Kết nối biểu thức con được nhóm thứ n nếu nó đã kết nối. Nếu không, tham chiếu tới biểu diễn bát phân của một mã ký tự |

Dưới đây mình sẽ liệt kê ra một số các cú pháp pattern trong Python mà chúng ta hay sử dụng và kèm với theo nó là các ví dụ.

'.' (Chấm.): Trong chế độ mặc định, cái này khớp với bất kỳ ký tự nào ngoại trừ một dòng mới. Nếu [**DOTALL**](https://docs.python.org/2/library/re.html#re.DOTALL)cờ đã được chỉ định, điều này khớp với bất kỳ ký tự nào kể cả dòng mới.

'^'(Caret.): Khớp với phần bắt đầu của chuỗi và trong [**MULTILINE**](https://docs.python.org/2/library/re.html#re.MULTILINE)chế độ cũng khớp ngay sau mỗi dòng mới.

'$': Khớp với phần cuối của chuỗi hoặc ngay trước dòng mới ở cuối chuỗi và trong [**MULTILINE**](https://docs.python.org/2/library/re.html#re.MULTILINE)chế độ cũng khớp trước dòng mới. foo khớp với cả 'foo' và 'foobar', trong khi biểu thức thông thường foo$chỉ khớp với 'foo'. Thú vị hơn, tìm kiếm foo.$trong các 'foo1\nfoo2\n' trận đấu 'foo2' bình thường, nhưng 'foo1' trong [**MULTILINE**](https://docs.python.org/2/library/re.html#re.MULTILINE)chế độ; tìm kiếm một $trong một 'foo\n'sẽ tìm thấy hai kết quả (trống): một ngay trước dòng mới và một ở cuối chuỗi.

'\*': Làm cho RE kết quả khớp 0 hoặc nhiều lần lặp lại của RE trước đó, càng nhiều lần lặp lại càng tốt. ab\*sẽ khớp với 'a', 'ab' hoặc 'a' theo sau bởi bất kỳ số 'b' nào.

'+': Làm cho RE kết quả khớp với 1 hoặc nhiều lần lặp lại của RE trước đó. ab+sẽ khớp với 'a' theo sau bởi bất kỳ số nào khác của 'b'; nó sẽ không khớp với 'a'.

'?': Làm cho RE kết quả khớp với 0 hoặc 1 lần lặp lại của RE trước đó. ab?sẽ khớp với 'a' hoặc 'ab'.

Ví dụ: ab\* thì ở đây ta có 2 RE đó là a và b. Phép \* là khớp 0 hoặc nhiều lần lặp lại của RE trước đó tức là 0 hoặc nhiều lần sự có mặt của b. Tức là sẽ khớp a hoặc ab hoặc a theo sau bởi nhiều b bất kỳ.

ab+ thì ở đây ta có 2 RE đó là a và b. Phép + là khớp 1 hoặc nhiều lần lặp lại của RE trước đó tức là 1 hoặc nhiều lần sự có mặt của b. Ở đây sẽ khớp ít nhất từ 1 lần nên sẽ khớp ab hoặc a theo sau bởi nhiều b bất kỳ không khớp riêng a vì b phải có ít nhất 1 lần xuất hiện trong biểu thức khớp

ab? thì ở đây ta có 2 RE đó là a và b. Phép ? là khớp 0 hoặc 1 lần lặp lại của RE trước đó. Tức là sẽ khớp a hoặc ab

[]

Trong cặp dấu này chứa các ký tự có thể so khớp với văn bẳn. Nó gần giống như toán tử hoặc, nghĩa là chỉ cần nó so khớp 1 trong các ký tự được liệt kê trong cặp dấu này thì chuỗi đó được so khớp.

**VD**: pattern chuỗi bắt đầu bằng T hoặc t.

import re

str = 'thanhtaivtt'

match = re.match(r'[Tt]', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!')

else:

print ('Khong tim thay!')

# Kết quả:

# t

Và bạn cũng có thể liệt kê ra các ký tự dưới dạng từ ... đến ... bằng cách thể hiện dấu - giữa hai khoảng đó.

**VD**: Pattern chuỗi bắt đầu bằng ký tự in thường hoặc bằng số.

import re

str = 'thanhtaivtt'

match = re.match(r'[a-z0-9]', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!')

else:

print ('Khong tim thay!')

# Kết quả:

# t

.

Ký tự dấu . này tương đương với việc so khớp một chuỗi phải chứa ít nhất một ký tự.

**VD**:

import re

str = 'thanhtaivtt'

match = re.match(r'.', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!')

else:

print ('Khong tim thay!')

# Kết quả:

# t

^

Ký tự này đại diện cho việc so khớp từ đầu của chuỗi.

**VD**: so khớp xem có phải đầu chuỗi là chữ h

import re

str = 'thanhtaivtt'

match = re.search(r'^h', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!')

else:

print ('Khong tim thay!')

# Kết quả:

# Khong tim thay!

$

Ký tự này đại diện cho việc so khớp đến cuối chuỗi.

**VD**: so khớp xem cuỗi chuỗi có phải chữ t

import re

str = 'thanhtaivtt'

match = re.search(r't$', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong tim thay!')

# Kết quả:

# Khop!

\*

Ký tự này đại diện cho có thể có hoặc không có ký tự trước nó.

**VD**: So khớp một chuỗi xem có thể bắt đầu bằng t hoặc không bắt đầu bằng t hoặc là chuỗi rỗng.

import re

str = ''

match = re.match(r't\*', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong tim thay!')

# Kết quả:

# Khop!

+

Ký tự này đại diện cho có thể xuất hiện ít nhất hoặc nhiều ký tự trước nó.

**VD**: So khớp xem một chuỗi chỉ chứa chữ cái in thường.

import re

str = 'vuthanhtai'

match = re.match(r'[a-z]+', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong tim thay!')

# Kết quả:

# Khop!

?

Ký tự này đại diện cho chuỗi sẽ khớp với một trong các ký tự đằng trước nó.

**VD**: Kiểm tra xem một chuỗi có được bắt đầu bằng a hoặc b hay không.

import re

str = 'vuthanhtai'

match = re.match(r'ab?', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong tim thay!')

# Kết quả:

# Khong tim thay!

{m, n}

Ký tự này đại diện cho việc so khớp xem chuỗi đằng trước nó xuất hiện bao nhiêu tối thiểu m lần và tối đa n lần. Nếu bỏ trống n thì là so khớp sự xuất hiện m lần của chuỗi đằng trước nó.

**VD**: Kiểm tra xem 3 chữ cái đầu của chuỗi có phải là v không.

import re

str = 'vuthanhtai'

match = re.match(r'v{3}', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong tim thay!')

# Kết quả:

# Khong tim thay!

|

Ký tự này đại diện cho sự tồn tại của một trong 2 ký tự trước và sau nó.

**VD**: kiểm tra xem đầu chuỗi cõ phải là a hoặc b hay không.

import re

str = 'vuthanhtai'

match = re.match(r'a|b', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong khop!')

# Kết quả:

# Khong Khop!

()

Ký tự này dùng để gom nhóm các pattern lại với nhau.

|  |  |
| --- | --- |
| (re) | Nhóm các Regular Expression và ghi nhớ text đã kết nối. Kết quả in ra sẽ là các chuỗi thỏa mãn điều kiện trong () |

**VD**: Kiểm tra xem chuỗi chó chứa chũ '*thanh*' không.

import re

str = 'vuthanhtai'

match = re.match(r'(.\*)(thanh)(.\*)', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong khop!')

# Kết quả:

# Khop!

\

Ký tự này giúp phân biệt chuỗi sau nó không phải là ký tự đặc biệt.

**VD**: Kiểm tra xem đầu chuỗi có phải là dấu . không.

import re

str = '.vuthanhtai'

match = re.match(r'\.', str)

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong khop!')

# Kết quả:

# Khop!

Các pattern khác

|  |  |
| --- | --- |
| **Pattern** | **Mô Tả** |
| [^...] | Kết nối với bất kỳ ký tự đơn nào không ở trong [] |
| \A | So khớp chuỗi là chuỗi. |
| \B | So khớp nếu vị trí đặt \B không phải là đầu hoặc cuối chuỗi. |
| \d | So khớp với số nguyên. |
| \D | So khớp với các ký tự không phải là số. |
| \s | So khớp với khoảng trắng |
| \S | So khớp với các kỹ tự không phải là khoảng trắng. |
| \w | So khớp với chữ hoặc số. |
| \W | So khớp với các ký tự không phải là chữ hoặc số. |

2, Các flags trong Python.

Như ở phần 1 mình có trình bày các thông số flag trong Regular Expression, nhưng mình chưa liệt kê ra các flag đó. Và dưới đây là một số các flags hay dùng trong Python.

* I hoặc IGNORECASE - Không phân biệt hoa thường khi so khớp.
* L hoặc LOCALE - So Khớp với local hiện tại.
* M hoặc MULTILINE - Thay đổi $ và ^ thành kết thúc của một dòng và bắt đầu của một dòng thay vì mặc định là kết thúc chuỗi và bắt đầu chuỗi.
* A hoặc ACSII - Thay đổi \w, \W, \b, \B, \d, \D, \S và \s thành so khơp full unicode.
* S hoặc DOTALL -Thay đổi pattern . thành khớp với bất kỳ ký tự nào và dòng mới.
* .....

|  |  |
| --- | --- |
| **Modifier** | **Miêu tả** |
| re.I | Thực hiện việc kết nối hoặc so khớp không phân biệt kiểu chữ |
| re.L | Thông dịch các từ theo Locale hiện tại |
| re.M | Làm cho $ kết nối với phần cuối của một dòng (mà không chỉ kết nối phần cuối của chuỗi) và làm cho ^ kết nối với phần đầu của bất cứ dòng nào (mà không chỉ kết nối phần đầu của chuỗi) |
| re.S | Làm cho dot (dấu chấm) kết nối với bất kỳ ký tự nào, bao gồm một newline (dòng mới) |
| re.U | Thông dịch các chữ cái theo bộ ký tự Unicode. Flag này ảnh hưởng tới hành vi của \w, \W, \b, \B |
| re.X | Cho phép cú pháp "cuter" regular expression. Nó bỏ qua các khoảng trống trắng whitespace (ngoại trừ bên trong một [] hoặc khi được tránh bởi \) và coi # dưới dạng một comment |

**VD**: So khớp một chuỗi bắt đầu bằng v hoặc V.

import re

str = 'vuthanhtai'

match = re.match(r'V', str, re.IGNORECASE) # hoặc re.I

if match: #nếu tồn tại chuỗi khớp

print('Khop!');

else:

print ('Khong khop!')

# Kết quả:

# Khop!