Python - Expressões Regulares

Uma expressão regular é uma sequência especial de caracteres que ajuda a combinar ou encontrar outras strings ou conjuntos de strings, usando uma sintaxe especializada mantida em um padrão. Uma expressão regular também conhecida como regex é uma sequência de caracteres que define um padrão de pesquisa. Popularmente conhecido como regex ou regexp; é uma sequência de caracteres que especifica um padrão de correspondência no texto. Geralmente, esses padrões são usados por algoritmos de busca de strings para operações de "localização" ou "localização e substituição" em strings ou para validação de entrada.

O processamento de texto em grande escala em projetos de ciência de dados requer manipulação de dados textuais. O processamento de expressões regulares é suportado por muitas linguagens de programação, incluindo Python. A biblioteca padrão do Python possui o módulo 're' para essa finalidade.

Como a maioria das funções definidas no módulo re funcionam com strings brutas, vamos primeiro entender o que são strings brutas.

Cordas brutas

Expressões regulares usam o caractere de barra invertida ('\') para indicar formas especiais ou para permitir que caracteres especiais sejam usados sem invocar seu significado especial. Python, por outro lado, usa o mesmo caractere que o caractere de escape. Conseqüentemente, Python usa a notação de string bruta.

Uma string se torna uma string bruta se for prefixada com r ou R antes dos símbolos de cotação. Portanto, 'Hello' é uma string normal e r'Hello' é uma string bruta.

```
>>> normal="Hello"
>>> print (normal)
Hello
>>> raw=r"Hello"
>>> print (raw)
Hello
```

Em circunstâncias normais, não há diferença entre os dois. No entanto, quando o caractere de escape está incorporado na string, a string normal realmente interpreta a sequência de escape, enquanto a string bruta não processa o caractere de escape.

```
>>> normal="Hello\nWorld"
>>> print (normal)
Hello
World
```

```
>>> raw=r"Hello\nWorld"
>>> print (raw)
Hello\nWorld
```

No exemplo acima, quando uma string normal é impressa, o caractere de escape '\n' é processado para introduzir uma nova linha. No entanto, devido ao operador de string bruto 'r', o efeito do caractere de escape não é traduzido de acordo com seu significado.

Metacaracteres

A maioria das letras e caracteres simplesmente combinam entre si. No entanto, alguns caracteres são metacaracteres especiais e não correspondem entre si. Metacaracteres são caracteres com um significado especial, semelhante a * no curinga.

Aqui está uma lista completa dos metacaracteres -

```
. ^ $ * + ? { } [ ] \ | ( )
```

Os símbolos de colchetes [e] indicam um conjunto de caracteres que você deseja corresponder. Os caracteres podem ser listados individualmente ou como um intervalo de caracteres separados por um '-'.

Sr. Não.	Metacaracteres e descrição
1	[abc] corresponder a qualquer um dos caracteres a, b ou c
2	[ac] que usa um intervalo para expressar o mesmo conjunto de caracteres.
3	[az] corresponda apenas a letras minúsculas.
4	[0-9] corresponder apenas a dígitos.
5	'^' complementa o conjunto de caracteres em [].[^5] corresponderá a qualquer caractere, exceto '5'.

'\'é um metacaractere de escape. Quando seguido por vários caracteres, forma várias sequências especiais. Se precisar corresponder a [ou \, você pode precedê-los com uma barra invertida para remover seu significado especial: \[ou \\.

Conjuntos predefinidos de caracteres representados por sequências especiais começando com '\' estão listados abaixo -

Sr. Não.	Metacaracteres e descrição
1	\d Corresponde a qualquer dígito decimal; isso é equivalente à classe [0-9].
2	\D Corresponde a qualquer caractere que não seja um dígito; isso é equivalente à classe [^0-9].
3	**\s** Corresponde a qualquer caractere de espaço em branco; isso é equivalente à classe [\t\n\r\f\v].
4	\S Corresponde a qualquer caractere que não seja espaço em branco; isso é equivalente à classe [^\t\n\r\f\v].
5	\c Corresponde a qualquer caractere alfanumérico; isso é equivalente à classe [a-zAZ0-9_].
6	\C Corresponde a qualquer caractere não alfanumérico. equivalente à classe [^a-zAZ0-9_].
7	. Corresponde a qualquer caractere único, exceto nova linha '\n'.
8	? corresponder a 0 ou 1 ocorrência do padrão à sua esquerda
9	+ 1 ou mais ocorrências do padrão à sua esquerda
10	* 0 ou mais ocorrências do padrão à sua esquerda
11	\b limite entre palavra e não palavra e /B é oposto de /b
12	[..] Corresponde a qualquer caractere único entre colchetes e [^..] corresponde a qualquer caractere único que não esteja entre colchetes.
13	\ É usado para caracteres de significado especial como \. para corresponder a um ponto ou \+ para sinal de mais.

14	{n,m} Corresponde a pelo menos n e no máximo m ocorrências de anteriores
15	uma| b Corresponde a a ou b

O módulo re do Python fornece funções úteis para encontrar uma correspondência, procurar um padrão e substituir uma string correspondente por outra string, etc.

Função re.match()

Esta função tenta combinar o padrão RE no início da string com sinalizadores opcionais .

Aqui está a **sintaxe** desta função -

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

Aqui está a descrição dos parâmetros -

Sr. Não.	Parâmetro e Descrição
1	**padrão** Esta é a expressão regular a ser correspondida.
2	**Corda** Esta é a string que seria pesquisada para corresponder ao padrão no início da string.
3	**Bandeiras** Você pode especificar sinalizadores diferentes usando OR bit a bit (|). Esses são modificadores, listados na tabela abaixo.

A função re.match retorna um objeto **de correspondência em caso de sucesso e Nenhum** em caso de falha. Uma instância do objeto match contém informações sobre a correspondência: onde ela começa e termina, a substring com a qual correspondeu, etc.

O método start() do objeto match retorna a posição inicial do padrão na string e end() retorna o ponto final.

Se o padrão não for encontrado, o objeto de correspondência será Nenhum.

Usamos a função group(num) ou groups() do objeto **match** para obter a expressão correspondente.

Sr. Não.	Métodos e descrição do objeto de correspondência

1	**group(num=0)** Este método retorna a correspondência inteira (ou num subgrupo específico)
2	**groups()** Este método retorna todos os subgrupos correspondentes em uma tupla (vazia se não houver nenhum)

Exemplo

```
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'Cats', line)
print (matchObj.start(), matchObj.end())
print ("matchObj.group() : ", matchObj.group())
```

Ele produzirá a seguinte **saída** -

```
0 4
matchObj.group() : Cats
```

Função re.search()

Esta função procura a primeira ocorrência do padrão RE dentro da string , com flags opcionais .

Aqui está a **sintaxe** desta função -

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

Aqui está a descrição dos parâmetros -

Sr. Não.	Parâmetro e Descrição
1	**Padrão** Esta é a expressão regular a ser correspondida.
2	**Corda** Esta é a string que seria pesquisada para corresponder ao padrão em qualquer lugar da string.
3	**Bandeiras** Você pode especificar sinalizadores diferentes usando OR bit a bit (|). Esses são modificadores, listados na tabela abaixo.

A função re.search retorna um objeto **de correspondência** em caso de sucesso, **nenhum** em caso de falha. Usamos a função group(num) ou groups() do objeto **match** para obter a expressão correspondente.

Sr. Não.	Métodos e descrição do objeto de correspondência
1	**group(num=0)** Este método retorna a correspondência inteira (ou num subgrupo específico)
2	**groups()** Este método retorna todos os subgrupos correspondentes em uma tupla (vazia se não houver nenhum)

Exemplo

```
import re
line = "Cats are smarter than dogs"
matchObj = re.search( r'than', line)
print (matchObj.start(), matchObj.end())
print ("matchObj.group() : ", matchObj.group())
```

Ele produzirá a seguinte **saída** -

```
17 21
matchObj.group() : than
```


Correspondência versus pesquisa

Python oferece duas operações primitivas diferentes baseadas em expressões regulares: match verifica uma correspondência apenas no início da string, enquanto search verifica uma correspondência em qualquer lugar da string (isso é o que Perl faz por padrão).

Exemplo

```
import re
line = "Cats are smarter than dogs";
matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
   print ("match --> matchObj.group() : ", matchObj.group())
else:
```

```
   print ("No match!!")
searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
   print ("search --> searchObj.group() : ", searchObj.group())
else:
   print ("Nothing found!!")
```

Quando o código acima é executado, ele produz a seguinte **saída** -

```
No match!!
search --> matchObj.group() : dogs
```


Função re.findall()

A função findall() retorna todas as correspondências não sobrepostas do padrão na string, como uma lista de strings ou tuplas. A string é digitalizada da esquerda para a direita e as correspondências são retornadas na ordem encontrada. As correspondências vazias são incluídas no resultado.

Sintaxe

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

Parâmetros

Sr. Não.	Parâmetro e Descrição
1	**Padrão** Esta é a expressão regular a ser correspondida.
2	**Corda** Esta é a string que seria pesquisada para corresponder ao padrão em qualquer lugar da string.
3	**Bandeiras** Você pode especificar sinalizadores diferentes usando OR bit a bit (|). Esses são modificadores, listados na tabela abaixo.

Exemplo

```
import re
string="Simple is better than complex."
obj=re.findall(r"ple", string)
print (obj)
```

Ele produzirá a seguinte **saída** -

```
['ple', 'ple']
```

O código a seguir obtém a lista de palavras em uma frase com a ajuda da função findall().

```
import re
string="Simple is better than complex."
obj=re.findall(r"\w*", string)
print (obj)
```

Ele produzirá a seguinte **saída** -

```
['Simple', '', 'is', '', 'better', '', 'than', '', 'complex', '', '']
```


Função re.sub()

Um dos métodos mais importantes **que** usam expressões regulares é **sub** .

Sintaxe

```
re.sub(pattern, repl, string, max=0)
```

Este método substitui todas as ocorrências do padrão RE em string por repl , substituindo todas as ocorrências, a menos que max seja fornecido. Este método retorna uma string modificada.

Exemplo

```
import re
phone = "2004-959-559 # This is Phone Number"

# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
```

```
print ("Phone Num : ", num)

# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print ("Phone Num : ", num)
```

Ele produzirá a seguinte **saída** -

```
Phone Num : 2004-959-559
Phone Num : 2004959559
```

Exemplo

O exemplo a seguir usa a função sub() para substituir todas as ocorrências de is por was word -

```
import re
string="Simple is better than complex. Complex is better than complicated."
obj=re.sub(r'is', r'was',string)
print (obj)
```

Ele produzirá a seguinte **saída** -

```
Simple was better than complex. Complex was better than complicated.
```

Função re.compile()

A função compile() compila um padrão de expressão regular em um objeto de expressão regular, que pode ser usado para correspondência usando match(), search() e outros métodos.

Sintaxe

```
re.compile(pattern, flags=0)
```

Bandeiras

Sr. Não.	Modificador e descrição
1	**re.eu** Executa correspondência sem distinção entre maiúsculas e minúsculas.

2	**re.L** Interpreta palavras de acordo com a localidade atual. Esta interpretação afeta o grupo alfabético (\w e \W), bem como o comportamento dos limites das palavras (\b e \B).
3	**ré.** M Faz com que $ corresponda ao final de uma linha (não apenas ao final da string) e faz com que ^ corresponda ao início de qualquer linha (não apenas ao início da string).
4	**re.S** Faz com que um ponto final corresponda a qualquer caractere, incluindo uma nova linha.
5	**re.U** Interpreta letras de acordo com o conjunto de caracteres Unicode. Este sinalizador afeta o comportamento de \w, \W, \b, \B.
6	**re.X** Permite uma sintaxe de expressão regular "mais bonita". Ele ignora espaços em branco (exceto dentro de um conjunto [] ou quando escapado por uma barra invertida) e trata # sem escape como um marcador de comentário.

A sequência -

```
prog = re.compile(pattern)
result = prog.match(string)
```

é equivalente a -

```
result = re.match(pattern, string)
```

Mas usar re.compile() e salvar o objeto de expressão regular resultante para reutilização é mais eficiente quando a expressão será usada várias vezes em um único programa.

Exemplo

```
import re
string="Simple is better than complex. Complex is better than complicated."
pattern=re.compile(r'is')
obj=pattern.match(string)
obj=pattern.search(string)
print (obj.start(), obj.end())

obj=pattern.findall(string)
print (obj)
```

```
obj=pattern.sub(r'was', string)
print (obj)
```

Ele produzirá a seguinte saída -

```
7 9
['is', 'is']
Simple was better than complex. Complex was better than complicated.
```

Função re.finditer()

Esta função retorna um iterador que produz objetos de correspondência sobre todas as correspondências não sobrepostas para o padrão RE em string.

Sintaxe

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

Exemplo

```
import re
string="Simple is better than complex. Complex is better than
complicated."
pattern=re.compile(r'is')
iterator = pattern.finditer(string)
print (iterator )

for match in iterator:
   print(match.span())
```

Ele produzirá a seguinte **saída** -

```
(7, 9)
(39, 41)
```

Casos de uso de Python Regex

Encontrando todos os advérbios

findall() corresponde a todas as ocorrências de um padrão, não apenas à primeira, como faz search(). Por exemplo, se um escritor quiser encontrar todos os advérbios em algum texto, ele poderá usar findall() da seguinte maneira -

```
import re
text = "He was carefully disguised but captured quickly by police."
obj = re.findall(r"\w+ly\b", text)
print (obj)
```

Ele produzirá a seguinte **saída** -

```
['carefully', 'quickly']
```

Encontrar palavras que começam com vogais

```
import re
text = 'Errors should never pass silently. Unless explicitly silenced.'
obj=re.findall(r'\b[aeiouAEIOU]\w+', text)
print (obj)
```

Ele produzirá a seguinte **saída** -

```
['Errors', 'Unless', 'explicitly']
```