Manual Simples de Expressões Regulares em Python

Autor: Bismarck Gomes Souza Júnior
E-mail: bismarckjunior@outlook.com
Data: Setembro, 2012

Conteúdo

Módulo re (Regular Expression)

Esse módulo fornece operações com expressões regulares (ER).
Para importar o módulo, basta fazer:
>>> import re

Versão do Python

Todos os exemplos foram testados com Python v2.7.3.

Raw strings

As expressões regulares utilizam em algumas situações a contra barra (“\”), porém para utilizá-la precisam-se de 2 contra barras. Por exemplo, para usar o retrovisor em uma ER deve-se escrever: “\\1”, o que representa “\1”.

A fim de eliminar essa necessidade, existe a notação de *raw string* no Python. Ela considera cada caractere isoladamente. Para utilizá-la, basta prefixar “r” à *string*. Assim, enquanto r”\\” contém dois caracteres (duas contra barras), a *string* “\\” contém apenas um (uma contra barra).

Exemplo:

```
>>> len(r'\\')
2
>>> len('\\')
1
>>> r'\\'
'\\\\'
>>> print r'\\'
\\
>>> print '\\'
\
```

Sintaxe de ER em Python

Tipo	Metacaractere	Apelido	Descrição	Exemplo	
				Padrão	Combinações
Representantes	.	Ponto	Casa qualquer caractere, exceto "\n". (Veja a seção "*Flags*", p.9).	.ato	gato, rato, pato, tato, mato, ...
				12.30	12:30, 12-30, 12.30, 12 30, 12u30, ...
				<.>	, , <u>, , <7>, <>>, ...
	[ ]	Lista	Lista os caracteres possíveis.	[prg]ato	gato, rato, pato
				<[Bubi]>	<B>, <u>, ,
			Dentro da lista os metacaracteres são caracteres normais.	12[:-.]30	12:30, 12-30, 12.30
				1[.+*/%-]3	1.3, 1+3, 1*3, 1/3, 1%3, 1-3
			Intervalo numérico.	[0-9]	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
				[0-36-8]	0, 1, 2, 3, 6, 7, 8
			Intervalo de letras.	[a-f]	a, b, c, d, e, f
				[A-D]	A, B, C, D
			Os intervalos seguem tabela ascii.	[Z-a]	Z, [, \,], ^, _, `, a
			Lista com "[" (deve ser inserida no início).	[]inicio]	], i, n, i, c, i, o
			Lista com "-" (deve ser inserida no final).	[final-]	f, i, n, a, l, -
	[^]	Lista Negada	Lista os caracteres que não são possíveis.	[^0-9]!	a!, o!,?!, #!, r!, y!, <!, ...
				[^kwy]	Todos os caracteres exceto: k, w, y
			Lista negada com "^" (deve ser inserida no final).	[^fim^]	Todos os caracteres exceto: f, i, m, ^
Quantificadores gulosos	?	Opcional	Casa uma ou nenhuma ocorrência do padrão anterior.	[prg]atos?	gato, pato, rato, gatos, patos, ratos
				</?[bip]>	, , , , ,
				[Hh]?um	Hum, hum, um
	*	Asterisco	Casa muitas ou nenhuma ocorrência do padrão anterior.	.*!	oi!,ah!, !, Ei!, humm!!, oh!, ...
				hu*m*	h, hm, huum, hummmm, humm, ...
				h[auh]*	hahuah, hauhauau, h, hhuahhuahua
	+	Mais	Casa uma ou mais ocorrências do padrão anterior.	k+	k, kk, kkk, kkkk, kkkkkkk, ...
				bo+m	bom, boom, boooom, ...
				hu+m+	hum, huum, humm, huumm, ...
	{n, m}	Chaves	Intervalo de repetição do padrão anterior.	hum{1,3}	hum, humm, hummm
				Ah!{0,2}	Ah, Ah!, Ah!!
			Pelo menos n	bo{3,}m	booom, boooom, booooom, ...
			Exatamente n	bo{3}m	booom

Quantificadores não gulosos	??	Opcional	Casa uma ou nenhuma ocorrência do padrão anterior. Casa o mínimo possível.	ba??	Casa "b" de: ba, baa, ...
				ba?	Casa "ba" de: ba, baa
				ba??c	Casa "bac" de: bac
	*?	Asterisco	Casa muitas ou nenhuma ocorrência do padrão anterior. Casa o mínimo possível.	<.*?>	Casa "" de: oi
				<.*>	Casa "oi" de: oi
				hu*?m*?	Casa "h" de: huumm, hum, ...
	+?	Mais	Casa uma ou mais ocorrências do padrão anterior. Casa o mínimo possível.	<.+?>	Casa "" de: oi
				<.+>	Casa "oi" de: oi
				hu+?m+?	Casa "huum" de: huumm, huum, ...
	{n, m}?	Chaves	Intervalo de repetição do padrão anterior. Casa o mínimo possível.	hum{1,3}?	Casa "hum" de: humm, hummm, ...
				hum{1,3}	Casa "humm" de: humm, hummm, ...
			Pelo menos n.	bom{2,}?	Casa "bomm" de: bomm, bommm, ...
Âncoras	^	Circunflexo	Casa o começo da cadeia de caracteres. (Veja a seção "*Flags*", p. 9)	^[0-9]	Textos que começam com números
				^[^0-9]	Textos que não começam com números
	$	Cifrão	Casa o final da cadeia de caracteres."$" só será metacaractere no final do padrão.	[aos]$	Textos que terminam com: a, o ou s
				^.{10,}$	Textos com pelo menos 10 caracteres
				^$	Textos em branco
Captura de Dados	( )	Grupo	Grupo de padrão. Podem ser consultados posteriormente.	(rs)+	rs, rsrs, rsrsrs, ...
				(h[au]+)+	haua,huahuaauu, hahuhau, ...
	\N	Retrovisor	Casa o n-ésimo grupo. "n" varia de 1 a 99.	(mal)-\1	mal-mal
				(.(.))\1\2	babaca, ...
Alternadores	|	Ou	Casa ou o padrão que precede ou o que sucede.	hi|ap	hi,ap
				h(ea|o)t	heat,hot
				(m|big)-?\1	mm, m-m, bigbig, big-big
				a|bc|def	a, bc, def
				a|(bc|de)f	a, bcf, def
Especiais	\	Escape	Transforma um metacaractere em caractere.	(w{3}\.)?z\.br	www.z.br, z.br
				[0-9]\.[0-9]+	2.718, 3.14, 1.4142, ...
				\\	\
				1 \+2	1+2

Padrões Individuais	\A	Início	Casa apenas o começo da cadeia de caracteres.	\AEra	Textos que comecem com: Era
				\A[AO]	Textos que comecem com: A ou O
	\b	Borda	Casa as bordas das palavras. Detecta as bordas por separação de letras.	\bera	Palavras que comecem com: era
				\bera\b	Casa a palavra "era" dentro do texto
				m\b	Palavras que terminem com: m
	\B	Não-borda	Negação de "\b". Não casa borda	py\B	Casa o "py" de: python, py3, py4, pyQt, ...
					Não casa o "py" de: py!, .py, py., ...
	\d	Dígito	Casa dígitos de 0 a 9. Equivalente a [0-9].	^\d	Linhas que começam com um dígito
				(\d)\1	00, 11, 22, 33, 44, 55, 66, 77, 88, 99
	\D	Não-dígito	Negação de "\d". Não casa dígitos. Equivalente a [^0-9].	^\D	Linhas que não começam com um dígito.
				(\D)\1	??, mm, !!, gg, pp, ...
	\s	Branco	Casa os espaços em branco. Equivalente a [\t\n\r\f\v].	.\s.	a c, e r, 1 d, ...
				!\s[A-D]	! A, ! B, ! C, ! D
	\S	Não-branco	Negação de "\s". Não casa espaços em branco. Equivalente a [^ \t\n\r\f\v]	a\Sa	aba, asa, a!a, aha, ata, a#a, ...
				!\s\S	! A, ! f, ! 3, ! =, ! y, ...
	\w	Palavra	Casa caracteres alfanuméricos e "_". Equivalente a [a-zA-Z0-9_].	(\w)\1	bb, cc, SS, 33, __, mm, ...
				\w\d	b1, b2, g5, a3, g7, v0, d6, ...
	\W	Não-palavra	Negação de "\w". Equivalente a [^a-zA-Z0-9_].	(\W)+	!, !@, !@#, $#, $@!?, $, ...
				\w\W	s!, D$, n?, o@, ...
	\Z	Fim	Casa apenas o fim da cadeia de caracteres.	\d\Z	Textos que terminem com um dígito
				\D\Z	Textos que não terminem com um dígito

Expressões mais modernas

Padrão (?...)	Descrição		Exemplo	
			Padrão	Combinações
(?#comentario)ER	Insere um comentário, tal que "texto" é o comentário.		(?#corrigir)e.tenso	extenso, estenso, e5tenso,...
(?iLmsux)ER	Inclui *flags* a partes de uma ER.		(?imu)[a-z]{2}	Li, Na, Ka, Rb, Cs, Fr ...
	i	IGNORECASE	(?i)[a-z]{2}	Be, Mg, Ca, Sr, Ba, Ra, ...
	L	LOCALE	-	-
	m	MULTILINE	(?m)\d$	Linhas terminando com número
	s	DOTALL	(?s).*	Casa todo o texto
	u	UNICODE	(?u)\w	Casa "à", "é","ç", ...
	x	VERBOSE	(?x)\d #digito	7, 5, 3, 2, ...
(?P=id)	Casa o que foi casa pelo grupo nomeado de "id".		(?P<inicio>.)(?P=inicio)	uu,tt, mm, gg, pp, ...
			(?P<dia>\d\d)/(?P=dia)	12/12, 11/11, 04/04, ...
(?=ER)	Casar o padrão precedente somente se a ER casar.		Chico (?=Xavier)	Casa "Chico" de: Chico Xavier
			meia (?=pp|gg)	Casa "meia" de: meia pp, meia gg
(?!ER)	Casar o padrão precedente somente se a ER não casar.		Chico (?!Xavier)	Casa "Chico" de: Chico Anysio
			meia (?!m)	Casa "meia" de: meia pp, meia gg
(?<=ER)	Casar o padrão posterior somente se a ER casar.		(?<=meia) pp	Casa "pp" de: meia pp
			(?i)(?<=Ford) Ka	Casa "ka" de: ford ka
(?<!ER)	Casar o padrão posterior somente se a ER não casar.		(?<!meia)pp	Casa "pp" de: Apple
			(?i)(?<!Ford)Ka	Casa "ka" de: kaka
(?:ER)	Grupo ignorado pelo retrovisor.		(.+)-(?:a)-(\1)	passo-a-passo, dia-a-dia, ...
			(.*)-?(?:\1)	mm, big-big, mal-mal, ...
(?P<nome>ER)	Cria um nome para o grupo: "nome".		(?P<animal>[prg]ato)	pato, rato, gato
			(?P<nome>[Bb]runa)	Bruna, bruna
(?(id/nome)s|n)	Casa o padrão "s" se o grupo "id" ou "nome" tiver sido casado. Caso contrário, casar o padrão "n".		(<)?(\w)(?(1)>|!)	<b>, <p>, h!, j!, g!, ...
			(?P<n><)?(\w)(?(n)>|!)	<b>, <p>, h!, j!, g!, ...
			(<)?(\w)(?(1)>)	b, <b>, p, <p>, ...

Flags

Flags		**Descrição**	**Exemplo**	
			Padrão	**Combinações**
i	IGNORECASE	Casa tanto maiúsculas quanto minúsculas.	(?i)[a-c]	a, b, c, A, B, C
			(?i)[A-C]	a, b, c, A, B, C
L	LOCALE	Faz \w, \W, \b, \B, \s e \S depender do locale (localização.)	* Não consegui alterar a localização para testar. Dizem que [a-z] casará "ç", "á", "é", ...	
m	MULTILINE	Faz a âncora "^" casar o início de cada linha. Já o "$", o final.	(?m)\d$	Linhas terminando com número
			(?m)^[0-9]	Linhas iniciando com número
s	DOTALL	Faz o metacaractere "." casar tudo, incluindo "\n".	(?s).*	Casa todo o texto
			.*	Casa cada linha do texto
u	UNICODE	Faz \w, \W, \b, \B, \d, \D, \s and \S dependerem do padrão Unicode.	(?u)\w	Casa "à", "é","ç", ...
			\w	Não casa "à", "é", "ç", ...
x	VERBOSE	Comentários (#) e espaços em branco são ignorados. O espaço deve ser precedido por "\".	(?x)\d #digito	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
			(?x)\d\ \d	4 6, 5 3, 1 2, ...
			(?x)\d \d	07, 22, 24, 51, 69, 71

Funções do Módulo

- **re.compile (***pattern, flags=0***)**

  pattern: padrão, ER a ser casada.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: re.**RegexObject**

  Compila o *pattern*.

  Exemplo:
  ```
  >>> import re
  >>> regexobj1 = re.compile(r“\w”, re.I)    # ou
  >>> re.compile(“\\w”, re.IGNORECASE)
  <_sre.SRE_Pattern object at … >
  ```

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

  pattern: padrão, ER a ser casada.
  string: texto a ser scaneado.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: re.**MatchObject** ou **None**

  Procura a ocorrência do *pattern* dentro da *string*.

  Exemplo:
  ```
  >>> import re
  >>> re.search(r“\w”, ‘7s’)
  <_sre.SRE_Match object at … >
  ```

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

  pattern: padrão, ER a ser casada.
  string: texto a ser scaneado.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: re.**MatchObject** ou **None**

  Procura a ocorrência do *pattern* no início da *string*. Se o *pattern* casar o início da *string*, a função retorna o re.**MatchObject** correspondente. Senão, retorna **None**.

  Mesmo que a *flag* re.**MULTILINE** seja usada, esta função não irá casar o começo de cada linha e sim o começo da *string*.

Exemplo:
```
>>> import re
>>> re.match(r'\d', 'a1b2c3')    # Retorna None
>>> re.match(r'.\d', 'a1b2c3')
<_sre.SRE_Match object at … >
```

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

  pattern: padrão, ER a ser casada.
  string: texto a ser scaneado.
  maxsplit: número máximo de "pedaços".
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: lista

  Fatia a *string* nos pontos onde o *pattern* casa com a *string*.

  Quando existir grupos dentro do *pattern*, estes também serão adicionados à lista.

  Quando o *maxsplit* é alcançado, é adicionado o restante da *string* no final da lista.

  Exemplo:
```
>>> import re
>>> re.split(r'q\d', 'a1b2c3')
['a', 'b', 'c', '']
>>> re.split(r'\d', 'a1b2c3'[:-1])
['a', 'b', 'c']
>>> re.split(r'(\d)', 'a1b2c3')
['a', '1', 'b', '2', 'c', '3']
>>> re.split(r'(\d)', 'a1b2c3', 2)
['a', '1', 'b2c3']
>>> re.split(r'\w', 'a1b2c3', 2)
['', '', '', '', '', '', '']
>>> re.split(r'\W', 'a1b2c3', 2)
['a1b2c3']
```

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

  pattern: padrão, ER a ser casada.
  string: texto a ser scaneado.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: lista

  Retorna os valores casados em forma de lista. Se não encontrar nada, retorna uma lista vazia.
  Caso exista um grupo, retornar-se-á uma lista deste. Caso exista mais de um grupo, retornar-se-á uma lista de tuplas.

Exemplo:
```
>>> import re
>>> re.findall(r'\d\D', 'a1b2c3')
['1b', '2c']
>>> re.findall(r'\d(\D)', 'a1b2c3')
['b', 'c']
>>> re.findall(r'\d(?:\D)', 'a1b2c3')
['1b', '2c']
>>> re.findall(r'(\d)(\D)', 'a1b2c3')
[('1', 'b'), ('2', 'c')]
>>> re.findall(r'\d\d', 'a1b2c3')
[]
```

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

  pattern: padrão, ER a ser casada.
  string: texto a ser scaneado.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: iterador para re.**MatchObject**

  Exemplo:
```
>>> import re
>>> for m in re.finditer('\d', 'a1b2c3'):
...   print m.start(),
1 3 5
```

- **re.sub (***pattern, repl, string, count=0, flags=0***)**

  pattern: padrão, ER a ser casada.
  repl: *string* ou função que substituirá o *pattern*.
  string: texto a ser scaneado.
  count: número máximo de substiuições.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: *string* substituida

  Se *repl* for uma *string*, os escapes (“\n”, “\r”, “\t”, ...) serão aceitos. Além disso, as referências aos grupos casados no padrão também são aceitas. Para isso, pode-se utilizar as expressões “\N” e “\g<nome/id>”, por exemplo: “\1” ou “\g<1>”, “\11” ou “\g<11>”, “\g<nome>”.

  Se *repl* for uma função, ela será chamada para cada ocorrência do padrão. Essa função deve receber um objeto do tipo re.**MachObject** e retornar a *string* a ser substituída.

  Se o *pattern* não for encontrado, a função retornará a *string* inalterada.

Exemplo:

```
>>> import re
>>> re.sub(r'\d\B', '', 'a1b2c3', 1)
'ab2c3'
>>> re.sub(r'(\d)\B', r'\1\1', 'a1b2c3')
'a11b22c3'
>>> def funcao(matchobj):
...   if matchobj.group(0) == '1':    return ' '
...   elif matchobj.group(0) == '2': return 'o'
...   else: return 'a'
...
>>> re.sub(r'\d', funcao, 'a1b2c3')
'a boca'
```

- **re.subn (***pattern, repl, string, count=0, flags=0***)**

  pattern: padrão, ER a ser casada.
  repl: *string* ou função para substituir o *pattern*.
  string: texto a ser scaneado.
  count: número máximo de substiuições.
  flags: re.**IGNORECASE** (re.**I**), re.**LOCALE** (re.**L**), re.**MULTILINE** (re.**M**), re.**DOTALL** (re.**S**), re.**UNICODE** (re.**U**), re.**VERBOSE** (re.**X**)
  retorna: tupla com a nova *string* e o número de substituições feitas.

  Exemplo:

```
>>> import re
>>> re.subn(r'\d\B', '', 'a1b2c3', 4)
('abc3', 2)
```

- **re.escape (***string***)**

  string: texto a ser transformado.
  retorna: texto com "\".

  Exemplo:

```
>>> import re
>>> re.escape('\\')
'\\\\'
>>> re.escape('\n')
'\\\n'
```

- **re.purge ( )**

  Limpa o cache do módulo.

Classes do Módulo

re.RegexObject

Exemplo:

```
>>> import re
>>> regexobj = re.compile (‘(\d)(?P<nome>\D)’, re.I)
```

- **search (***string[, pos[, endpos]]***)**

  string: texto a ser scaneado.
  pos: posição inicial para procura
  endpos: posição final para procura
  retorna: re.**MatchObject** ou **None**

- **match (***string[, pos[, endpos]]***)**

  string: texto a ser scaneado.
  pos: posição inicial para procura
  endpos: posição final para procura
  retorna: re.**MatchObject** ou **None**

- **split (***string, maxsplit=0***)**

  string: texto a ser scaneado.
  maxsplit: número máximo de “pedaços”.
  retorna: lista

- **findall (***string[, pos[, endpos]]***)**

  string: texto a ser scaneado.
  pos: posição inicial para procura.
  endpos: posição final para procura.
  retorna: lista

- **finditer (***string[, pos[, endpos]]***)**

  string: texto a ser scaneado.
  pos: posição inicial para procura.
  endpos: posição final para procura.
  retorna: iterador para re.**MatchObject**

- **sub (***repl, string, count=0***)**

  repl: *string* ou função para substituir o *pattern*.
  string: texto a ser scaneado.
  count: número máximo de substiuições.
  retorna: *string* substituida

- **subn (***repl, string, count=0***)**

  repl: *string* ou função para substituir o *pattern*.
  string: texto a ser scaneado.
  count: número máximo de substiuições.
  retorna: tupla com a nova *string* e o número de substituições feitas.

o **flags**

Retorna um inteiro que representa uma combinação de flags utilizadas na compilação ou no *pattern*.

Exemplo:
```
>>> regexobj.flags
0
```

o **groups**

Número de grupos no *pattern.*

Exemplo:
```
>>> regexobj.groups
2
```

o **groupindex**

Dicionário com os nomes dos grupos definidos por “(?P<nome>)”, onde a chave é o “nome” e o valor é o número do grupo.

Exemplo:
```
>>> regexobj.groupindex
{‘nome’: 2}
```

o **pattern**

Padrão (ER) a partir do qual foi compilado.

Exemplo:
```
>>> regexobj.pattern
'(\\d)(?P<nome>\\D)'
```

re.MatchObject

Exemplo:

```
>>> import re
>>> matchobj1 = re.match(r'(..)', 'a1b2c3')   #casa 1 vez
>>> matchobj2 = re.match(r'(..)+', 'a1b2c3')  #casa 3 vezes
>>> matchobj3 = re.search(r'(..)(?P<dp>\d\D)', 'a1b2c3')
>>> matchobj4 = re.match(r'(?P<g1>\d\D)(?P<g2>\D\d)?', 'a1b2c3')
```

- **expand (***template***)**

  template: modelo
  retorna: *string*

  Retorna a *string* obtida pela substituição das contra barras. O *template* pode utilizar as expressões como "\N" e "\g<nome/id>", além dos escapes.

  Exemplo:

```
>>> matchobj1.expand(r'casa: \1')
'casa: a1'
>>> matchobj2.expand(r'casa: \g<1>')
'casa: c3'
```

- **group (***[group1, …]***)**

  [group1, …]: id ou nome do grupo.
  retorna: retorna um ou mais subgrupos ou **None**

  O número default é 0, o qual retorna toda *string* casada. Retorna **None** quando o grupo não casa nada.

  Exemplo:

```
>>> matchobj3.group( )
'1b2c'
>>> matchobj3.group(0)
'1b2c'
>>> matchobj3.group(1)
'1b'
>>> matchobj3.group('dp')
'2c'
>>> matchobj3.group(1, 2)
('1b', '2c')
```

- **groups (***[default]***)**

  [default]: valor a ser retornado quando um grupo não for casado.
  retorna: tupla com os grupos casados.

  Quando o grupo não for casado, ele retorna **None**, a menos que o *default* seja indicado.

Exemplo:
```
>>> matchobj3.groups()
('1b', '2c')
>>> matchobj4.groups()
('a1', None)
>>> matchobj4.groups('vazio')
('a1', 'vazio')
```

- **groupdict (***[default]***)**

  [default]: valor a ser retornado quando um grupo não for casado.
  retorna: dicionário tal que as chaves são os nomes dos grupos.

  Exemplo:
```
>>> match4.groupdict( )
{'g2': None, 'g1': 'a1'}
>>> match4.groupdict('vazio')
{'g2': 'vazio', 'g1': 'a1'}
```

- **start (***[group]***)**

  [group]: id ou nome do grupo.
  retorna: posição inicial do grupo na *string* recebida.

  Exemplo:
```
>>> matchobj3.start()
1
>>> matchobj4.start('g1')
0
>>> matchobj4.start('g2')
-1
```

- **end (***[group]***)**

  [group]: id ou nome do grupo casado.
  retorna: posição final do grupo na *string* recebida.

  Exemplo:
```
>>> matchobj3.end()
5
>>> matchobj4.end('g1')
2
>>> matchobj4.end('g2')
-1
```

- **span (***[group]***)**

  [group]: id ou nome do grupo.
  retorna: tupla com a posição inicial e final do grupo na *string* recebida.

Exemplo:

```
>>> matchobj3.span()
(1, 5)
>>> matchobj4.span('g1')
(0, 2)
>>> matchobj4.span('g2')
(-1, -1)
```

- **pos**

  Posição inicial para procura na *string* passada. Veja os métodos **search( )** e **match( )** da classe re.**RegexObject**.

  Exemplo:

```
>>> m1.pos
0
```

- **endpos**

  Posição final para procura na *string* passada. Veja os métodos **search( )** e **match( )** da classe re.**RegexObject**.

  Exemplo:

```
>>> m1.endpos
6
```

- **lastindex**

  Índice do último grupo capturado. Retorna **None** se nenhum grupo for capturado.

  Exemplo:

```
>>> matchobj1.lastindex
1
>>> matchobj3.lastindex
2
>>> matchobj4.lastindex
1
```

- **lastgroup**

  Nome do último grupo capturado. Retorna **None** se nenhum grupo for capturado.

  Exemplo:

```
>>> matchobj1.lastgroup
>>> matchobj3.lastgroup
'dp'
>>> matchobj4.lastgroup
'g1'
```

- **re**

  O objeto da classe re.**RegexObject** que representa a expressão regular que gerou a classe.

  Exemplo:

```
>>> matchobj1.re.pattern
‘(..)’
>>> matchobj4.re.pattern
'(?P<g1>\\D\\d)(?P<g2>\\d\\D)?'
```

- **string**

  Retorna a *string* da classe.

  Exemplo:

```
>>> matchobj1.string
‘a1b2c3’
>>> matchobj4.string
‘a1b2c3’
```

Referências

[1] http://www.diveintopython.net/regular_expressions/index.html
[2] http://docs.python.org/library/re.html
[3] http://pt.wikipedia.org/wiki/Express%C3%A3o_regular
[4] http://en.wikipedia.org/wiki/Regular_expression
[5] http://aurelio.net/regex/guia/
[6] http://linux.studenti.polito.it/elda/elda/GNUtemberg/python/regex.pdf
[*] Todos os acessos em agosto de 2012

Site sugerido

http://www.pythonregex.com/

Críticas e sugestões

bismarckjunior@outlook.com