Títulos Texto

python brasil

ManipulandoStringsComPython

por: Peyton McCullough
original: DevShed

Esse artigo vai dar uma geral nos vários métodos de manipular uma string, cobrindo coisas de métodos básicos até expressões regulares em Python. Manipular uma string é uma técnica que todo programador Python devia se familiarizar.

Métodos de Strings

O jeito mais básico de manipular strings é através de métodos que estão dentro delas (strings). Podemos fazer um limitado número de tarefas em strings através desses métodos. Abra sua Python Shell e vamos criar uma string e brincar um pouco com elas.

```
>>> test = 'This is just a simple string.'
```

Vamos dar uma volta rápida e usar a função *len*. Ela pode ser usada para encontrar o tamanho de uma string contando espacos e caracteres especiais, ou seja, tudo que pertencer a essa string.

```
>>> len(test)
29
```

Beleza, agora vamos voltar aos métodos que eu estava falando.Vamos pegar nossa string e substituir uma palavra usando o método *replace*.

```
>>> test = test.replace('simple', 'short')
>>> test
'This is just a short string.'
```

Agora vamos contar o numero de vezes que a palavra especificada aparece na string, nesse caso estou apenas procurando por um caracter 'r'.

```
>>> test.count('r')
2
```

Podemos também achar em que posição está certa levra ou palavra.

```
>>> test.find('r')
18
>>> test[18]
'r'
```

Separar uma string é uma coisa que eu frequentemente faço.O método *split* é usado para isso.

```
>>> test.split()
['This', 'is', 'just', 'a', 'short', 'string.']
```

Podemos escolher o ponto a ser separado.

```
>>> test.split('a')
['This is just ', ' short string.']
```

Para juntar nossa string separada, podemos usar o método *join*.

```
>>> ' some '.join(test.split('a'))
'This is just  some  short string.'
```

Podemos brincar com a caixa das letras (maiúsculo ou minúsculo). Vamos deixar tudo maiúsculo.

```
>>> test.upper()
'THIS IS JUST A SHORT STRING.'
```

Agora vamos deixar tudo minúsculo.

```
>>> test.lower()
'this is just a short string.'
```

Vamos deixar apenas a primeira letra maiúscula de uma string minúscula.

```
>>> test.lower().capitalize()
'This is just a short string.'
```

Podemos usar o método *title*, que deixa as letras de cada palavra da string maiúscula.

```
>>> test.title()
'This Is Just A Short String.'
```

Uma troca também é possível.O que for maiúsculo vira minúsculo e vice-versa.

```
>>> test.swapcase()
'tHIS IS JUST A SHORT STRING.'
```

Podemos rodar alguns testes numa string usando poucos métodos. Vamos ver se a string dada é totalmente maiúscula.

```
>>> 'UPPER'.isupper()
True
>>> 'UpPEr'.isupper()
False
```

Do mesmo modo, podemos checar se a string dada é minúscula.

```
 >>> 'lower'.islower()
True
>>> 'Lower'.islower()
False
```

Checando se ela é um *title*, no caso, todas as palavras com a primeira letra maiúscula.

```
>>> 'This Is A Title'.istitle()
True
>>> 'This is A title'.istitle()
False
```

Podemos checar se a string é alfa-numérica, ou seja, contém apenas letras e números, sem caracteres especiais.

```
>>> 'aa44'.isalnum()
True
>>> 'a$44'.isalnum()
False
```

É possível checar se uma string contém apenas letras.

```
>>> 'letters'.isalpha()
True
>>> 'letters4'.isalpha()
False
```

Agora checando se ela contém apenas números.

```
>>> '306090'.isdigit()
True
>>> '30-60-90 Triangle'.isdigit()
False
```

Podemos checar se uma string contém apenas espacos.

```
>>> '   '.isspace()
True
>>> ''.isspace()
False
```

Falando em espacos, podemos adicionar espacos em ambos os lados de uma string.Vamos adicionar espacos no lado direito de uma string.

```
>>> 'A string.'.ljust(15)
'A string.      '
```

Para adicionar espacos do lado esquerdo, o método *rjust* é usado.

```
>>> 'A string.'.rjust(15)
'      A string.'
```

O método *center* é usado para centralizar uma string dentro de espacos.

```
>>> 'A string.'.center(15)
'   A string.   '
```

Podemos separar os espacos de ambos os lados de uma string.

```
>>> 'String.'.rjust(15).strip()
'String.'
>>> 'String.'.ljust(15).rstrip()
'String.'
```

Expressões regulares

Expressões regulares são uma ferramenta muito poderosa em qualquer linguagem.Elas permitem que padrões sejam **"achados"** ou **"casados"** dentro de strings.Ações como substituição podem ser feitas na string se a expressão regular **"casar"** com alguma parte da string.O módulo que cuida de expressões regulares no Python é o *re*.De volta à nossa shell...

```
>>> import re
```

Vamos criar uma string simples para brincarmos um pouco.

```
>>> test = 'This is for testing regular expressions in Python.'
```

Vamos começar com padrões simples para serem **"achados"** dentro da string, depois passamos para alguns mais complexos.Existem dois métodos para achar padrões em strings com o módulo *re*: *search* e *match*.Vamos dar uma olhada no *search* primeiro.

```
>>> result = re.search('This', test)
```

Podemos extrair o resultado usando o método *group*.

```
>>> result.group(0)
'This'
```

Você provavelmente está se perguntando sobre o método *group* e por que passamos zero para ele.É simples, e eu vou explicar.Veja só, **padrões são organizados em grupos**, desse jeito:

```
>>> result = re.search ('(Th)(is)',test)
```

Aqui há dois grupos dentro dos parenteses.Podemos extraí-los usando o método *group*.

```
>>> result.group(1)
'Th'
>>> result.group(2)
'is'
```

Passando zero para o método retorna ambos os grupos.

```
>>> result.group(0)
'This'
```

O benefício dos grupos se tornará claro depois que trabalharmos desse jeito em padrões normais. Primeiro vamos dar uma olhada na função *match*.Ela funciona similarmente à função *search*, mas existe uma diferença crucial.

```
>>> result =  re.match('This', test)
>>> print result
<_sre.SRE_Match object at 0x00994250>
>>> print result.group(0)
'This'
>>> result = re.match ('regular', test)
>>> print result
None
```

Note que *None* (o valor nulo do Python) foi retornado, mesmo com a palavra "*regular*" dentro da string.Se você não entendeu, o método *match* acha os padrões no início da string, e o *search* examina a função inteira.Você deve estar se perguntando se é possível o *match* encontar a palavra "regular" na string,mesmo a palavra não estando no início da string.A resposta é **sim**, é possível e isso nos leva a aprender um pouco sobre **padrões ou patterns**.

O caracter '.' casa com qualquer caracter.Podemos usar o método *match* para achar a palavra "*regular*" colocando um ponto para cada letra antes dele.Vamos separar isso em dois grupos.Um vai conter os pontos, e o outro vai conter o "*regular*".

```
>>> result = re.match ('(..................)(regular)', test)
>>> result.group(0)
'This is for testing regular'
>>> result.group(1)
```

```
'This is for testing '
>>> result.group(2)
'regular'
```

Viu só, conseguimos.No entanto é **ridículo** ter que colocar todos esses pontos.A boa notícia é que realmente não precisamos colocá-los.Dê uma olhada nisso e lembre que existem 20 caracteres antes da palavra "*regular*", é só contar.

```
>>> result = re.match('(.{20})(regular)', test)
>>> result.group(0)
'This is for testing regular'
>>> result.group(1)
'This is for testing '
>>> result.group(2)
'regular'
```

Assim é bem mais fácil.Agora vamos ver um pouco mais sobre padrões.Aqui é como você pode usar chaves de um jeito mais avançado.

```
>>> result = re.match ('(.{10,20})(regular)', test)
>>> result.group(0)
'This is for testing regular'
>>> result = re.match('(.{10,20})(testing)', test)
'This is for testing'
```

Colocando dois argumentos *{10,20}* , quer dizer que você pode achar qualquer número de caracteres em uma *escala (range)*, nesse caso de 10 a 20.Algumas vezes, no entanto, isso pode mostrar-nos um comportamento indesejado.Olhe só:

```
>>> anotherTest = 'a cat, a dog, a goat, a person'
```

Vamos fazer uma escala (range) de caracteres.

```
>>> result = re.match('(.{5,20})(,)', anotherTest)
>>> result.group(1)
'a cat, a dog, a goat'
```

Agora vamos pegar somente "*a cat*".E isso pode ser feito adicionando "*?*" ao final das chaves:

```
>>> result = re.match('(.{5,20}?)(,)', anotherTest)
>>> result.group(1)
'a cat'
```

Adicionando um ponto de interrogação faz a função "*achar*" a menor quantidade possível de caracteres.Um ponto de interrogação faz isso e não deve ser confundido com esse padrão abaixo:

```
>>> anotherTest = '012345'
>>> result = re.match('01?', anotherTest)
>>> result.group(0)
'01'
>>> result = re.match('0123456?', anotherTest)
>>> result.group(0)
'012345'
```

Como você pode ver, o caracter antes do ponto de interrogação é opcional na procura.

Bom, agora que você já sabe brincar com expressões regulares dentro do Python, aprenda um pouco mais sobre elas em Site do Aurélio e compre o seu guia de consulta rápida, muito útil. É isso ai pessoal, por hoje é só. Abraço a todos

EduardoOliva

Sobre esta página

ManipulandoStringsComPython (editada pela última vez em 2008-09-26 14:06:33 por localhost)

Visualizar Texto | Visualizar Impressão | Information | Fazer Usuário Acompanhar | Anexos