Busca e Substituição de Padrões em Textos

Semana 13 | EaD

Introdução

Python oferece algumas opções para a busca de padrões em textos e possivelmente suas substituições por outros textos. Quando editamos textos frequentemente executamos ferramentas de pesquisar e substituir padrões em textos.

`String.replace()`

O método `String.replace()` substitui uma substring específica com alguma outra.

Sintaxe

`string.replace(str_ant,str_sub,cont)`

`str_ant` : string anterior

`str_sub` : string a ser substituída

`cont` : controla quantas aplicações (opcional)

Exemplos

Quando a contagem não está explícita, o Python substitui todas ocorrências do padrão na string.

In [11]:
```
"o brasil tá lascado!".replace("brasil", "Brasil")
```

Out[11]:
```
'o Brasil tá lascado!'
```

In [18]:
```
"o brasil brasil tá lascado!".replace("brasil", "Brasil")
```

Out[18]:
```
'o Brasil Brasil tá lascado!'
```

In [20]:
```
"o brasil brasil tá lascado! brasil".replace("brasil", "Brasil")
```

Out[20]:
```
'o Brasil Brasil tá lascado! Brasil'
```

Contagem explícita

Busca e substitui somente a 1a ocorrência de `"r"` .

In [12]:
```
"O rato roeu a roupa do rei de roma".replace("r", "R", 1)
```

Out[12]:
```
'O Rato roeu a roupa do rei de roma'
```

Busca e substitui as duas primeiras ocorrência de `"r"` .

In [13]:
```
"O rato roeu a roupa do rei de roma".replace("r", "R", 2)
```

Out[13]:
```
'O Rato Roeu a roupa do rei de roma'
```

Busca e substitui as 15 primeiras ocorrências de `"r"` .

In [17]:
```
"O rato roeu a roupa do rei de roma".replace("r", "R", 15)
```

Out[17]:
```
'O Rato Roeu a Roupa do Rei de Roma'
```

In [24]:
```
"O rato roeu a roupa do rei de roma 0".replace(r"[0-9]", "R", 15)
```

Out[24]:
```
'O rato roeu a roupa do rei de roma 0'
```

Exercícios de fixação

Implemente a função a seguir que receba uma string e remova todos os caracteres de espaço .

In [71]:
```
def rem_esp(str_num):
    # str_num = transforme o str_num com o string.replace() e o retorne trans
    return str_num

str_num = "12 344 666"
rem_esp(str_num)
```

Out[71]:
```
'12 344 666'
```

Implemente a função a seguir que remove todos os caracteres de nova linha `\n` do texto e os substituem por uma caractere de espaço .

In [39]:
```
def rem_nl(texto):
    # texto = transforme o texto com o string.replace() e o transformado
    return texto.replace("\n", " ")

texto = "Parágrafo 1. blá blá blá.\n\nParágrafo 2. ti ti ti.\n\n\n"
rem_nl(texto)
```

Out[39]:
```
'Parágrafo 1. blá blá blá.  Parágrafo 2. ti ti ti.   '
```

Implemente uma função que utilize o `String.replace()` para remover dígitos de 0-9?

In [77]:
```
def rem0_9(str_num):
    # Utilizando o replace, remova os dígitos de 0-9 da string
    return str_num

str_num = "nesta string não nos importamos com os dígitos0123456789, mas some
rem0_9(str_num)
```

Out[77]:
```
'nesta string não nos importamos com os dígitos0123456789, mas somente com ca
racteres do alfabeto'
```

Implemente uma função que substitua todas as duplas de aspas duplas `""` de uma string de texto por um caractere de aspas duplas `"` . O caractere de escape `\` indica ao Python que o símbolo faz parte do texto e não deve ser interpretado como um comando.

In [94]:
```
def rm_aspas(uma_str):
    return uma_str.replace("\"\"", "\"")

texto = "...e então o pastor disse-lhes: \"\"não temam, pois estarei convosco
rm_aspas(texto)
```

Out[94]:
```
'...e então o pastor disse-lhes: "não temam, pois estarei convosco".'
```

Exercícios de fixação

Explique com suas palavras porque o caractere de aspas duplas `"` necessita de escape.

Sua resposta

Que outro(s) caractere(s) necessita(m) de escape no uso de strings (comuns)?

Sua Resposta

String.strip()

Remove todos os caracteres passados como parâmetros do começo ao fim da string.

Sintaxe

```
string.strip(caracteres)
```

Exemplos

O `strip()` aplicado a seguir remove todos os caracters `\n` do começo e fim da string, mas não faz nada com os `\n` entre os parágrafos.

In [64]:
```
"\n\n\n.. \n\tParágrafo 1. Texto texto texto\n\n\n\nParágrafo 2. Mais textos.
```

Out[64]:
```
'.. \n\tParágrafo 1. Texto texto texto\n\n\n\nParágrafo 2. Mais textos. '
```

Neste outro exemplo, todos os caracteres listados como argumentos são extraídos do começo e fim da string.

In [67]:
```
"\n\n\n.. \n\tParágrafo 1. Texto texto texto\n\n\n\nParágrafo 2. Mais textos.
```

Out[67]:
```
'Parágrafo 1. Texto texto texto\n\n\n\nParágrafo 2. Mais textos'
```

Exercício de fixação

Explique o resultado do `strip()` apresentado a seguir.

In [68]:
```
",,,,,rrtggg.......banana.....rrrr".strip("r.,rtg")
```

Out[68]:
```
'banana'
```

`String.lstrip()`

Opcionalmente, pode-se extrair somente de um lado da string. O `String.lstrip()` extrai apenas do lado esquerdo da string.

In [69]:
```
"\n\nTexto texto texto\n\n\n\n\n".lstrip("\n")
```

Out[69]:
```
'Texto texto texto\n\n\n\n\n'
```

`String.rstrip()`

Para extrair apenas do um lado direito da string, utilize o `String.rstrip()` .

In [70]:
```
"\n\nTexto texto texto\n\n\n\n\n".rstrip("\n")
```

Out[70]:
```
'\n\nTexto texto texto'
```

Regex

re.sub()

Sintaxe

```
re.sub(regex, str_sub, string)
```

No entanto, embora o método `String.replace()` seja muito bom para encontrar strings específicas dentro de um texto. Por exemplo, podemos utilizar o `String.replace()` para encontrar o número `"1"`, mas o que fazer para substituir os caracteres de a-z e A-Z? O método manual é muito trabalho para este caso e inviável para vários outros.

As expressões regulares entram em cena. A biblioteca Python que implementa o uso de expressões regulares chama-se `re`, contração de *r egular e xpressions*.

In [1]:
```
# Digite o seguinte comando para utilizar expressões regulares em Python
import re
```

Substituição de duas ou mais ocorrências de um padrão por outro

Suponha o seguinte texto.

In [78]:
```
texto = "Coluna1\n\n\nColuna2\n\n\nColuna3"
print(texto)
```

```
Coluna1


Coluna2


Coluna3
```

Se minha intenção é transformar este texto em uma linha de um arquivo `.csv.`, preciso tratar os caracteres `\n`, uma vez que cada `\n` indica uma nova linha. Um `.csv.` tradicional é separado por `;`. Assim, vamos trocar cada trio de `\n` por um `;`.

In [81]:
```
import re
# O sinal de + após o \n indica que um ou mais \n serão substituídos pela str
texto = re.sub("\n+", ";", "Coluna1\n\n\nColuna2\n\n\nColuna3")
print(texto)
```

```
Coluna1;Coluna2;Coluna3
```

Exercício de fixação

Utilizando uma combinação de métodos vistos anteriormente, transforme o texto a seguir de tal forma que o resultado seja igual ao do exemplo anterior.

In [82]:
```
import re

def prep_texto(linha):
    # Transforme a string nesta parte do código
    return linha

linha = "\t\t\nColuna1\n\n\nColuna2\n\n\nColuna3\n\n\n"
prep_texto(linha)
```

Out[82]:
```
'\t\t\nColuna1\n\n\nColuna2\n\n\nColuna3\n\n\n'
```

Substituição de dois ou mais caracteres com escape

Como substituir uma ou mais aspas duplas por uma única ocorrência utilizando o método `re.sub()` em um texto qualquer?

In [105…
```
# Implemente o padrão de busca e a string de substituição
re.sub("", "", "O jogador saiu de campo dizendo: \"\"\"quando o jogo está a m
```

Out[105…
```
'O jogador saiu de campo dizendo: """quando o jogo está a mil, a naftalina so
be""".'
```

Explique porque o caractere `[` necessita de escape em uma expressão regular.

Sua resposta

Como remover as ocorrências do caractere `[` no padrão de busca do método `re.sub()` ?

In [109…
```
# Implemente o padrão de busca e a string de substituição
re.sub("", "", "e então ele disse: [se não comprar nada o desconto é maior] J
```

Out[109…
```
'e então ele disse: [se não comprar nada o desconto é maior] Julius Rock'
```

Substituição com mais de um padrão de busca

O uso de regex permite o uso de mais de um padrão de busca. Neste caso, usa-se o caractere pipe `|` , que costuma ser lido como `ou` .

In [114…
```
re.sub("\n|\t", "", "\n\n\t\nNão sabendo que era impossível foi lá e soube\n\
```

Out[114…
```
'Não sabendo que era impossível foi lá e soube'
```

Exercício de fixação

Como remover as ocorrências do caractere `[` e do `]` e do no padrão de busca do método `re.sub()` ?

In [116…
```
re.sub("", "", "Beber, [ter um livro, escrever uma árvore] e plantar um filho
```

Out[116…
```
'Beber, [ter um livro, escrever uma árvore] e plantar um filho'
```

Caractere Estrela

Explique este resultado. Numa expressão regular, o que significam o caractere `.` e a estrela `*` ?

In [125…
```
re.sub(".*", "", "O que o lápis escreveu a borracha apagou.")
```

Out[125…
```
''
```

Sua resposta

Explique porque a sring retorna vazia?

In [127…
```
re.sub("\[.*\]", "", "[1];[2];[3]")
```

Out[127…
```
''
```

Sua resposta

Explique porque o uso do caractere coringa `\d` limita o padrão de busca.

In [129…
```
re.sub("\[\d\]", "", "Fulano [1], Ciclano [2] e Beltrano [3]")
```

Out[129…
```
'Fulano , Ciclano  e Beltrano '
```

O que é necessário acrescentar ao padrão de busca para que todas as referências sejam removidas?

In [132…
```
re.sub("\[\d\]", "", "Fulano [1], Ciclano [2] e Beltrano [33]")
```

Out[132…
```
'Fulano , Ciclano  e Beltrano [33]'
```

Sua resposta

Como modificar o padrão de busca para que todas as referências sejam removidas, assumindo que dentro dos colchetes podem aparecer um único dígito `\d` ou uma única letra de `a-z` ? Dica: é necessário agrupar o padrão de busca.

In [136…
```
re.sub("\[\d\]", "", "Fulano [1], Ciclano [2] e Beltrano [a]")
```

Out[136…
```
'Fulano , Ciclano  e Beltrano [a]'
```

Modularizando código em funções

As funções são excelentes para agruparmos transformações comuns a várias casos. Por exemplo, suponha que temos duas strings de texto e duas transformações em cada uma delas.

In [118…
```
str_1 = "\t\t\nPaís\n\n\nCasos\n\n\nMortes\n\n\n"
str_2 = "\t\t\nBrasil\n\n\n1000\n\n\n100\n\n\n"

str_1 = str_1.strip("\n\t")
str_1 = re.sub("\n+", ";", str_1)

str_2 = str_2.strip("\n\t")
str_2 = re.sub("\n+", ";", str_2)

print(str_1)
print(str_2)
```

```
País;Casos;Mortes
Brasil;1000;100
```

As funções fazem com que essas transformações fiquem mais gerais, isto é, possam ser aplicadas em quaisquer strings que necessitem destas mesmas tranformações.

In [123…
```
str_1 = "\t\t\nPaís\n\n\nCasos\n\n\nMortes\n\n\n"
str_2 = "\t\t\nBrasil\n\n\n1000\n\n\n100\n\n\n"

def prep_linha(linha):
    linha = linha.strip("\n\t")
    linha = re.sub("\n+", ";", linha)
    return linha

print(prep_linha(str_1))
print(prep_linha(str_2))

# Novos casos podem utilizar a mesma função
str_3 = "\t\t\nEUA\n\n\n2000\n\n\n200\n\n\n"
print(prep_linha(str_3))
```

```
País;Casos;Mortes
Brasil;1000;100
EUA;2000;200
```