< VOLTAR

Expressões regulares em Python

Apresentar as funções que trabalham com expressões regulares em Python.

NESTE TÓPICO

> EXPRESSÕES
REGULARES

> Dê uma olhada nos links abaixo para saber mais sobre a linguagem Python:



Marcar tópico

AUTOR(A)
PROF.
DENILSON
JOSE
SCHAFFER



Olá alunos.

Como já foi dito, uma string é um vetor de caracteres e assim é internamente tratado pelo interpretador ou um compilador e as expressões regulares estão relacionadas a este conjunto de caracteres, expressões regulares, do inglês: **regular expression** ou pela sigla **regex**, veio da área da matemática e este termo foi introduzido na Ciência da Computação para designar um conjunto de caracteres a serem processados.

EXPRESSÕES REGULARES

Expressões regulares são utilizadas nas seguintes situações:

- Para validação de dados: se os dados de entrada obedecem ao padrão estabelecido.
- Para busca de dados: encontra dados que podem conter uma parte (sub-string) de uma string.
- Para Busca e substituição: recurso utilizado pela maioria dos aplicativos de editores de texto: buscam uma palavra correspondente e substituem por outro valor.
- Para fragmentar(Split) de dados: Toda vez que encontrar um determinado caractere em uma string, esta string pode ser dividida (fragmentada).

Python fornece uma biblioteca para trabalhar com expressões regulares semelhante a linguagem Perl.

Esta biblioteca está no módulo **re**, e a principal função que utilizamos é a função de correspondência **match**, já que vamos precisar buscar, validar, fragmentar dados de um conjunto de caracteres.

Por exemplo, para a string de valor Uninove, o interpretador considera uma cadeia (vetor) simples de caracteres literais. Já o conjunto de caracteres, conhecidos como especiais: \'? \\$ + -*[] { }() | são considerados símbolos.

Vimos que \n utilizados em uma mensagem de texto, tanto no comando input quanto no print pula uma linha, porém, se tirarmos a barra \n passa a ser um simples literal.

Algo importante também que utilizamos para fazer buscas é o quantificador, que é representado entre { } que podemos adicionar um valor interno.

Por exemplo, para aceitar valores iniciais de um número de celular (por exemplo, **99876**-1234) que tem 5 números de 0 a 9, podemos definir: \d{5}, onde \d representa qualquer digito entre 0 a 9 e {5} é a quantidade de números iniciais do celular (99876).

Agora se quisermos aceitar todo o número de qualquer celular, com o padrão, por exemplo: 99876 - 1234, poderemos definir como: \d{5} - \d{4}, onde \d representa qualquer dígito de o a 9, {5} quantidade de números iniciais, - o hífen, \d qualquer dígito de 0 a 9 e {4} quantidade de 4 dígitos finais.

Para busca: podemos utilizar os métodos match e search do módulo re.

Para substituição: podemos utilizar os métodos sub e subn do módulo re.

Para fragmentação: podemos utilizar os métodos split do módulo re.

Exemplo:

Buscar um número válido para o CPF:

```
    # valida o número do cpf com pontos e hífens
    import re
    cpf = str(input('Entre com o número do CPF, com pontos e hífen: \n'))
    if re.search('\d{3}.\d{3}.\d{3}-\d{2}', cpf):
    print('Número CPF validado')
    else:
    print('Número do CPF fora do padrão')
    input('Pressione ENTER para sair...')
```

Na linha 1, incluímos o módulo re.

Na linha **2,** criamos a variável **cpf**, tipo string (**str**), para receber o número do cpf no padrão **000.000.000-00**.

Na linha **4**, utilizamos o método **search** do módulo **re**. O formato do método é: **search(padrão, string a ser comparada)**, no caso, o valor da variável cpf é a string a ser comaparada.

O padrão é a formatação '\d{3}.\d{3}.\d{3}-\d{2}' que significa: três números de 0 a 9, o ponto, mais três números, mais um ponto, mais três números, um hífen e mais dois números.

Vamos ver um exemplo, entrando com um número de cpf:

- Entre com o número do CPF, com pontos e hífen:
 038.745.211-46
- 3. Número CPF validado
- 4. Pressione ENTER para sair...

Agora, digitando números de CPF fora do padrão.

- 1. Entre com o número do CPF, com pontos e hífen:
- 2. 038745.211-46
- 3. Número do CPF fora do padrão
- 4. Pressione ENTER para sair...
- 1. Entre com o número do CPF, com pontos e hífen:
- 2. 038.74.211-46
- 3. Número do CPF fora do padrão
- 4. Pressione ENTER para sair...

Modificando o código para aceitar o padrão 00000000-00 para o CPF:

- 1. # valida o número do cpf com pontos e hífens
- 2. import re
- 3. cpf = str(input('Entre com o número do CPF, com pontos e hífen: \n'))
- 4. if re.search('\d{3}\d{3}\d{3}-\d{2}', cpf):
- 5. print('Número CPF validado')
- 5. else:
- 7. print('Número do CPF fora do padrão')
- 8. input('Pressione ENTER para sair...')

Veja a resposta do exemplo.

- 1. Entre com o número do CPF, com pontos e hífen:
- 2. 038745211-46
- 3. Número CPF validado
- 4. Pressione ENTER para sair...

Este é só um exemplo, mas também podemos fazer o contrário, considerar os números do cpf sem ponto e sem hífen, no padrão 12345678911, daí é só modificar a linha 4, passando para o padrão search('\d{11}', cpf).

```
    # valida o número do cpf com pontos e hífens
    import re
    cpf = str(input('Entre com o número do CPF, com pontos e hífen: \n'))
    if re.search('\d{11}', cpf):
    print('Número CPF validado')
    else:
    print('Número do CPF fora do padrão')
    input('Pressione ENTER para sair...')
```

Agora, executando o código com um exemplo:

- 1. Entre com o número do CPF, com pontos e hífen:
- 2. 03874521146
- 3. Número CPF validado
- 4. Pressione ENTER para sair...

E ai, tente modificar o código para validar um CEP no padrão: 00000-000.

SAIBA MAIS...

Dê uma olhada nos links abaixo para saber mais sobre a linguagem Python:

https://www.python.org/doc/ (https://www.python.org/doc/)

https://wiki.python.org/moin/PythonBooks (https://wiki.python.org/moin/PythonBooks)

Neste tópico vimos alguns exemplos de aplicações de funções com expressões regulares para validação de dados de acordo com um padrão proposto.

Quiz

Exercício Final

Expressões regulares em Python

INICIAR >

Referências

SUMMERFIELD, M. *Programação em Python 3*: Uma introdução completa à linguagem Python. Rio de Janeiro Alta Books, 2012. 495 p.

MENEZES, N. N. C. *Introdução à programação com Python:* algoritmos e lógica de programação para iniciantes. 2. ed. São Paulo: Novatec, 2014. 328 p.

SWEIGART, AL. *Automatize tarefas maçantes com Python:* programação prática para verdadeiros iniciantes. São Paulo: Novatec, 2015. 568 p.

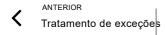
PYTHON, doc. Disponível em: https://www.python.org/doc/>. Acesso em: Junho/2018.

PYTHON, books. Disponível em: https://wiki.python.org/moin/PythonBooks. Acesso em: Junho/2018.



Avalie este tópico







Biblioteca

(https://www.uninove.br/conheca-

a-

uninove/biblioteca/sobre-

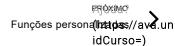
а-

biblioteca/apresentacao/)

Portal Uninove

(http://www.uninove.br)

Mapa do Site



® Todos os direitos reservados

