Expressões Regulares

Aula 11

Curso: BIG863 - Basic Python Programming for Ecologists

Professora: Dra. Cecilia F. Fiorini

Supervisor: Prof. Dr. Fernando A. O. Silveira

https://meet.google.com/zdi-ueoz-nsr, 24 de maio de 2023

Roteiro*

1. Expressões Regulares

*Conteúdo adaptado a partir de material desenvolvido pelo Prof. Zanoni Dias e disponível em https://ic.unicamp.br/ mc102.

Expressões Regulares

- Expressões regulares são formas concisas de descrever um conjunto de strings que satisfazem um determinado padrão.
- Por exemplo:
  - Podemos criar uma expressão regular para descrever todas as strings que representam datas no formato dd/dd/dddd, onde d é um dígito qualquer.
  - Podemos verificar se uma string contém um número de telefone, descrito por uma expressão regular.

ECMVS

Expressões Regulares

- Note que números de telefones e datas podem ser escritos em vários formatos diferentes.
- Números de telefones:
  - 31-91234-5678
  - (031) 91234 5678
  - (31)912345678
- Datas:
  - 09/10/2019
  - 09-10-19
  - 2019-10-09

ECMVS

Expressões Regulares

- Expressões regulares constituem uma mini-linguagem, que permite especificar as regras de construção de um conjunto de strings.
- Essa mini-linguagem de especificação é muito parecida entre as diferentes linguagens de programação que possuem o conceito de expressões regulares (também chamado de RE, REGEX ou RegExp).
- Assim, aprender a escrever expressões regulares em Python será útil para descrever expressões regulares em outras linguagens de programação.
- Expressões regulares são frequentemente utilizadas para encontrar ou extrair informações de textos (text parsing).

ECMVS

Expressões Regulares

- Exemplo de expressão regular:

```
1 '\d+\\\\'
```

- Essa expressão regular representa uma sequência de um ou mais dígitos seguidos por uma contrabarra (\).
- Podemos escrever expressões regulares iniciando com um caractere r para indicar uma raw string, ou seja, uma string onde o caractere \ é tratado como um caractere normal.
- Assim, a expressão regular resultante seria:

```
1 r'\d+\\'
```

Expressões Regulares - Regras

- Letras e números em uma expressão regular representam a si próprios.
- Assim a expressão regular r'Python' representa apenas a string 'Python'.
- Os caracteres especiais (chamados de meta-caracteres) são:

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


ECMVS

Expressões Regulares - Regras

- `.` um caractere qualquer.
- `^` o início da string.
- `$` o fim da string.
- `?` repetir zero ou uma vez.
- `+` repetir uma ou mais vezes.
- `*` repetir zero ou mais vezes.
- `\` usado para indicar caracteres especiais.

ECMVS

Expressões Regulares - Regras

indica um conjunto de caracteres.

- r'[0-9]': um dígito.
- r'[^0-9]': um caractere que não é um dígito.
- r'[a-z]': uma letra minúscula de a até z.
- r'[A-Z]': uma letra maiúscula de A até Z.
- r'[a-zA-Z]*': zero ou mais letras.
- r'[ACTG]+': uma sequência de DNA.

- indica a quantidade de vezes que o padrão será repetido.
  - r'[0-9]{2}': dois dígitos.
  - r'[a-z]{3}': três letras minúsculas.
  - r'[A-Z]{2,3}': duas ou três letras maiúsculas.
  - r'.{4,5}': quatro ou cinco caracteres quaisquer.
  - r'[01]{3,}': pelo menos três bits.
  - r'[0-9]{,6}': no máximo seis dígitos.

ECMVS

Expressões Regulares - Regras

- () indica um grupo em uma expressão regular.
  - r'([0-9]{3}\.){2}[0-9]{3}-[0-9]{2}': um CPF.
  - r'([a-z]+,)*[a-z]+': uma sequência de uma ou mais palavras separadas por vírgulas (e espaços).
- | similar ao operador lógico or para expressões regulares.
  - r'U(FMG|EMG)': uma de 2 universidades mineiras.
  - r'([0-9]{3}|[a-z]{4})': uma sequência de três dígitos ou uma sequência de quatro letras minúsculas.

ECMVS

Expressões Regulares - Classes de Caracteres

- Python possui algumas classes pré-definidas de caracteres:
  - \d um dígito, ou seja, [0-9].
  - \D o complemento de \d, ou seja, [^0-9].
  - \s um espaço em branco, ou seja, a [\t\n\r\f\v].
  - \S o complemento de \s, ou seja, [^ \t\n\r\f\v].
  - \w um caractere alfanumérico, ou seja, [a-zA-Z0-9].
  - \W o complemento de \w, ou seja, [^a-zA-Z0-9].

ECMVS

Expressões Regulares - Biblioteca re

- Em Python, expressões regulares são implementadas pela biblioteca re.
- Sendo assim, para usar expressões regulares precisamos importar a biblioteca re:

```
1 import re
```

- Documentação da biblioteca re: Link

ECMVS

Expressões Regulares

- A principal função da biblioteca re é a search.
- Dada uma expressão regular e uma string, a função search busca na string a primeira ocorrência de uma substring com o padrão especificado pela expressão regular.
- Se o padrão especificado pela expressão regular for encontrado, a função search retornará um objeto do tipo Match, caso contrário retornará None.
- Objetos do tipo Match possuem dois métodos:
  - span: retorna uma tupla com o local na string (posição inicial, posição final) onde a expressão regular foi encontrada.
  - group: retorna a substring encontrada.

ECMVS

Expressões Regulares

- Exemplo de uso da função search:

```
1  import re
2  texto = "Algoritmos e Programação de Computadores"
3  result = re.search(r'(\w*)ama(\w*)', texto)
4  print(type(result))
5  # <class 're.Match'>
6  print(result.group())
7  # Programação
8  print(result.span())
9  # (13, 24)
10 print(re.search(r'^\w*', texto))
11 # <re.Match object; span=(0, 10), match='Algoritmos'>
12 print(re.search(r'\w*$', texto))
13 # <re.Match object; span=(28, 40), match='Computadores'>
14 print(re.search(r'(^|\s)\w{3,9}(\s|$)', texto))
15 # None
```


ECMVS

Expressões Regulares

- Outro exemplo utilizando a função search:

```
1 import re
2 texto = "Algoritmos e Programação de Computadores"
3 result = re.search(r'\w+', texto)
4 print(result.group())
5 # Algoritmos
6 print(result.span())
7 # (0, 10)
```

- Note que a função search retorna apenas a primeira ocorrência do padrão especificado.

ECMVS

Expressões Regulares

- Dada uma expressão regular e uma string, a função findall retorna uma lista com todas as ocorrências do padrão especificado pela expressão regular.
- Exemplo:

```
1 import re
2 texto = "Algoritmos e Programação de Computadores"
3 result = re.findall(r'\w+', texto)
4 print(result)
5 # ['Algoritmos', 'e', 'Programação', 'de', 'Computadores']
6 telefone = "(019) 91234-5678"
7 result = re.findall(r'[0-9]+', telefone)
8 print(result)
9 # ['019', '91234', '5678']
```


ECMVS

Expressões Regulares

- Podemos construir uma expressão regular concatenando duas ou mais strings.
- Podemos usar o resultado das funções search e findall em expressões condicionais: None e [] são considerados False.

```
1  import re
2  texto = "Algoritmos e Programação de Computadores"
3  ini = "Algo"
4  meio = "ação"
5  fim = "dores"
6  regexp = r'^' + ini + r'.*' + meio + r'.*' + fim + r'$'
7  if re.search(regexp, texto):
8  print("OK")
9  else:
10 print("ERRO")
11 # OK
```


ECMVS

Expressões Regulares

- Podemos construir uma expressão regular concatenando duas ou mais strings.
- Podemos usar o resultado das funções search e findall em expressões condicionais: None e [] são considerados False.

```
1  import re
2  texto = "Algoritmos e Programação de Computadores"
3  ini = "Algo"
4  meio = "ação"
5  fim = "dores"
6  regexp = r'^' + ini + r'.*' + meio + r'.*' + fim + r'$'
7  if re.findall(regexp, texto):
8  print("OK")
9  else:
10 print("ERRO")
11 # OK
```


ECMVS

Expressões Regulares

- Expressões regulares podem ser utilizadas para dividir strings, similar ao método split visto na aula de strings.
- Dada uma expressão regular e uma string, a função split retorna uma lista com a divisão da string conforme especificado pela expressão regular.
- Exemplo:

```
1 import re
2 texto = "f1i1b2o3n5a8c13c21i"
3 letras = re.split(r'\d+', texto)
4 print(letras)
5 # ['f', 'i', 'b', 'o', 'n', 'a', 'c', 'c', 'i']
6 números = re.split(r'\D+', texto)
7 print(números)
8 # ['', '1', '1', '2', '3', '5', '8', '13', '21', '']
```


ECMVS

Expressões Regulares

- Expressões regulares podem ser utilizadas para substituir substrings, similar ao método replace visto na aula de strings.
- Dados dois padrões (strings ou expressões regulares) e uma string, a função sub retorna uma string com a subtituição na string de toda ocorrência do primeiro padrão pelo segundo padrão.

```
1 import re
2 texto = "f1i1b2o3n5a8c13c21i"
3 letras = re.sub(r'\d+', "", texto)
4 print(letras)
5 # fibonacci
6 números = re.sub(r'(\D+)', ":", texto)
7 print(números)
8 # :1:1:2:3:5:8:13:21:
```


ECMVS

Expressões Regulares

- Usando a função sub, podemos utilizar expressões regulares para indicar como a string será modificada, com base nos grupos da expressão regular (\1, \2, etc).
- Exemplo:

```
1  import re
2  data = "19/09/1975"
3  antigo = r'(\d{2})/(\d{2})/(\d{4})'
4  novo1 = r'\1-\2-\3'
5  data1 = re.sub(antigo, novo1, data)
6  print(data1)
7  # 19-09-1975
8  novo2 = r'\3/\2/\1'
9  data2 = re.sub(antigo, novo2, data)
10 print(data2)
11 # 1975/09/19
```

Expressões Regulares

- Podemos referenciar os grupos dentro da própria expressão regular para construir padrões mais complexos.
- Exemplo:

```
1  import re
2  dna = "AGTTAGTGCACACACTGAGGTTC"
3  print(re.search(r'(G[ACTG]{2})(.*)\1', dna).group())
4  # GTTAGTGCACACACTGAGGTT
5  # 111222222222222222111
6  print(re.search(r'([ACTG]{2})(.*)\1(.*)\1', dna).group())
7  # AGTTAGTGCACACACTGAG
8  # 1122113333333333311
9  print(re.sub(r'([ACTG]{2})(.*)\1(.*)\1', r'\1\3\1\2\1', dna))
10 # AGTGCACACACTGAGTTAGGTTC
11 # 1133333333333112211----
```


ECMVS

Expressões Regulares

- Podemos recuperar cada um dos grupos de uma expressão regular com a função group. Exemplo:

```
1  import re
2  texto = "Data de Nascimento: 19/09/1975"
3  result = re.search(r'(\d{2})/(\d{2})/(\d{4})', texto)
4  print(result.group())
5  # 19/09/1975
6  print("Dia:", result.group(1))
7  # Dia: 19
8  print("Mês:", result.group(2))
9  # Mês: 09
10 print("Ano:", result.group(3))
11 # Ano: 1975
12 print(result.group(1, 2, 3))
13 # ('19', '09', '1975')
```

Expressões Regulares

- Por padrão, os operadores +, *, ? e {,} são executados de forma gulosa, ou seja, eles tentam casar com o maior número possível de caracteres.
- Usando o caractere ? na frente daqueles operadores, eles são executados de forma não gulosa.
- Exemplo:

```
1 import re
2 texto = "Algoritmos e Programação de Computadores"
3 print(re.search(r'o(.*)e(.*)o', texto).group())
4 # oritmos e Programação de Computado
5 print(re.search(r'o(.*)e(.*?)o', texto).group())
6 # oritmos e Programação de Co
7 print(re.search(r'o(.*?)e(.*?)o', texto).group())
8 # oritmos e Pro
```


ECMVS

Expressões Regulares

- Por padrão, os operadores +, *, ? e {,} são executados de forma gulosa, ou seja, eles tentam casar com o maior número possível de caracteres.
- Usando o caractere ? na frente daqueles operadores, eles são executados de forma não gulosa.
- Exemplo:

```
1 import re
2 texto = "Removendo as <em>marcas</em> do <pre>texto</pre>."
3 print(re.sub(r'<.*>', "", texto))
4 # Removendo as .
5 print(re.sub(r'</.*>', "", texto))
6 # Removendo as <em>marcas.
7 print(re.sub(r'<.*?>', "", texto))
8 # Removendo as marcas do texto.
```


ECMVS

The regular expression game

Link