

Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



MÓDULO 2 – PROGRAMAÇÃO

Expressões regulares

EXPRESSÕES REGULARES

- ▶ São mecanismos que lhe permitem **de forma concisa e flexível identificar no texto caracteres ou cadeias de caracteres** de interesse ou ainda **padrões mais complexos**
- ▶ Esse termo deriva do trabalho de Stephen Kleene, um matemático, que as desenvolveu com o nome de **álgebra de conjuntos regulares** e foi base para os primeiros algoritmos de busca e ferramentas de processamento de texto em Unix
- ▶ Com expressões regulares podemos
 - ▶ **Casamento:** verificar se um texto apresenta casamento com um determinado padrão
 - ▶ **Substituição:** substituir no texto caso ele apresente casamento com um padrão
 - ▶ **Extração:** extrair partes de um texto caso ele apresente casamento com um determinado tipo de padrão

REGEX EM PYTHON

- ▶ Uma regex é uma expressão regular
- ▶ Em Python, há um módulo nativo chamado `re` que é usado para trabalhar com expressões regulares

```
import re
```

EXEMPLO

- ▶ Verificando se uma sequência começa com “>”, ou seja, se é uma linha de cabeçalho em um arquivo fasta (por exemplo)

```
import re

linha = '>1A6M:A|PDBID|CHAIN|SEQUENCE'
x = re.search('^>', linha)
if (x):
    print('Linha é cabeçalho')
else:
    print('Linha não é cabeçalho')
```

- ▶ Também poderia ser utilizada a função `match` que casa no início da *string*

FUNÇÕES DE CASAMENTO DO REGEX

Função	Descrição	Retorno
findall	Retorna uma lista de casamentos encontrados	list
search	Retorna o objeto “casado”, se houver um casamento	re.Match
split	Retorna uma lista na qual a string foi cortada em cada casamento	list
sub	Substitui um ou muitos casamentos em uma string	string
finditer	Retorna um iterador sobre objetos do tipo casamento	re.callable_iterator

METACARACTERES

- **Metacaracteres** são caracteres com significado especial

Caractere	Descrição
[]	Um conjunto de caracteres
\	Um caractere especial vem a seguir
.	Qualquer caractere (coringa), exceto “nova linha”
^	Começa com
\$	Termina com
*	ZERO ou mais ocorrências
+	UMA ou mais ocorrências
{ }	Exatamente o número especificado de ocorrências
	OU
()	Capturar e agrupar

SEQUÊNCIAS ESPECIAIS

- ▶ **Metacaracteres** são caracteres com significado especial

Caractere	Descrição
\A	Retorna uma correspondência se os caracteres estiverem no início da string
\b	Retorna uma correspondência se os caracteres estiverem início ou no fim da <i>string</i>
\B	Retorna uma correspondência se os caracteres estiverem presentes mas NÃO no início (ou no fim) da <i>string</i>
\d	Retorna uma correspondência onde a <i>string</i> contém dígitos (números de 0 a 9)
\D	Retorna uma correspondência onde a <i>string</i> NÃO contém dígitos
\s	Retorna uma correspondência onde a <i>string</i> contém um caractere de espaço em branco
\S	Retorna uma correspondência onde a <i>string</i> NÃO contém um caractere de espaço em branco

SEQUÊNCIAS ESPECIAIS

- ▶ **Metacaracteres** são caracteres com significado especial

Caractere	Descrição
\w	Retorna uma correspondência onde a <i>string</i> contém qualquer caractere formador de palavras ou alfanumérico (caracteres de a a Z, dígitos de 0 a 9 e o caractere de <i>underscore</i> _)
\W	Retorna uma correspondência onde a <i>string</i> contém qualquer caractere que não seja alfanumérico
\Z	Retorna uma correspondência se os caracteres especificados estiverem no final da <i>string</i>

CONJUNTOS

Caractere	Descrição
[ACTG]	Retorna uma corresponder onde um dos caracteres especificados (A, C, T ou G) está presente
[a-n]	Retorna uma correspondência para qualquer caractere minúsculo, alfabeticamente entre a e n
[^ACTG]	Retorna uma correspondência para qualquer caractere EXCETO A, C T e G
[0123]	Retorna uma correspondência onde qualquer um dos dígitos especificados (0, 1, 2 ou 3) estão presentes
[0-9]	Retorna uma correspondência para qualquer dígito entre 0 e 9
[0-5][0-9]	Retorna uma correspondência para quaisquer números de dois dígitos entre 00 e 59
[a-zA-Z]	Retorna uma correspondência para qualquer caractere em ordem alfabética entre a e z, minúscula OU maiúscula
[+]	Em conjuntos, +, *,., , (), \$, {} não tem significado especial, então [+] significa: retorna uma correspondência para qualquer caractere + na <i>string</i>

EXEMPLO: FINDALL

- ▶ Encontra todas as ocorrências do padrão “AA” na linha

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAAILIRLFKSHAATLEKFDRFKHLK'
x = re.findall('AA', linha)
print(x)
# Imprimirá
# ['AA', 'AA', 'AA', 'AA']
# OU
# [] se nenhum casamento for encontrado
```

EXEMPLO: SPLIT

- ▶ Retorna a primeira ocorrência do padrão “AA” na linha

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAILIRLFKSHAATLEKFDRFKHLK'
x = re.split('AA', linha)
print(x)

# Imprimirá ['VLS', 'EWQLVLHVW', 'VEADVAGHG', 'ILIRLFKSH', 'TLEKFDRFKHLK']
```

EXEMPLO: SPLIT

- ▶ Você pode controlar o número de *splits* executados se usar um parâmetro numérico a mais indicando o número máximo de casamentos que deseja realizar

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAAILIRLFKSHAATLEKFDRFKHLK'
x = re.split('AA', linha, 2)
print(x)

# Imprimirá ['VLS', 'EWQLVLHVW', 'VEADVAGHGAAILIRLFKSHAATLEKFDRFKHLK']
```

EXEMPLO: SUB

- ▶ Substitui as ocorrências de “AA” por “--”

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAAILIRLFKSHAATLEKFDRFKHLK'
x = re.sub('AA', '--', linha)
print(x)
print(linha)
# Imprimirá
# VLS--EWQLVLHVW--VEADVAGHG--ILIRLFKSH--TLEKFDRFKHLK
# VLSAAEWQLVLHVWAAVEADVAGHGAAILIRLFKSHAATLEKFDRFKHLK
```

EXEMPLO: SUB

- ▶ De forma análoga ao que se faz usando a função `split`, você também pode controlar quantas substituições fazer no máximo passando um parâmetro inteiro a mais

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAILIRLFKSHAATLEKFDRFKHLK'
x = re.sub('AA', '--', linha, 2)
print(x)
print(linha)
# Imprimirá
# VLS--EWQLVLHVW--VEADVAGHGAILIRLFKSHAATLEKFDRFKHLK
# VLSAAEWQLVLHVWAAVEADVAGHGAILIRLFKSHAATLEKFDRFKHLK
```


O OBJETO MATCH

- ▶ Um objeto do tipo `Match` contém informações sobre a pesquisa e o resultado
 - ▶ Se não houver correspondência, o valor **NONE** será retornado, em vez do objeto
 - ▶ O objeto tem propriedades e métodos que são usados para recuperar informações sobre a busca e sobre o resultado:
 - ▶ `span()`: retorna uma tupla contendo as posições inicial (`start()`) e final (`end()`) do casamento
 - ▶ Cuidado: a posição final vem sempre somada de uma unidade!
 - ▶ `string`: a *string* passada para a função
 - ▶ A *string* em que se faz a busca
 - ▶ `group()`: retorna a parte da *string* onde houve um casamento
 - ▶ A *string* casada

EXEMPLO: SEARCH

- ▶ Retorna a primeira ocorrência do padrão “AA” na linha

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAILIRLFKSHAATLEKFDRFKHLK'
x = re.search('AA', linha)
print('O primeiro AA encontrado foi na posição', x.start())
# Imprimirá "O primeiro AA encontrado foi na posição 3"
# ou "O primeiro AA encontrado foi na posição None" se não for encontrado
```

EXEMPLO: FINDITER

- ▶ Encontra todas as ocorrências do padrão “AA” na linha mas retorna um objeto iterador através do qual se pode obter diversas informações sobre o padrão encontrado

```
import re

linha = 'VLSAAEWQLVLHVWAAVEADVAGHGAAILIRLFKSHAATLEKFDRFKHLK'
for i in re.finditer('AA', linha):
    print(linha[i.start():i.end()], i.start(), i.end())

# Imprimirá
# AA 3 5
# AA 14 16
# AA 27 29
# AA 39 41
```

BUSCA GULOSA (GREEDY) E PREGUIÇOSA (LAZY)

- ▶ Os qualificadores ' * ', ' + ' e ' { } ' são **gulosos**
 - ▶ Eles buscam a mais distante correspondência possível
 - ▶ Às vezes esse comportamento não é o desejado
- ▶ Adicionar ' ? ' após o qualificador faz com que a busca seja **preguiçosa**, ou seja, a correspondência mais próxima será retornada

```
import re

linha = '<h1>Título</h1>'
x = re.search('<.*>', linha)
print(x.group())
# Imprimirá "<h1>Título</h1>"

x = re.search('<.*?>', linha)
print(x.group())
# Imprimirá "<h1>"
```