04

Praticando classes e quantifier

Transcrição

No exemplo do CPF, usamos a classe de caracteres `\d` para descrever todos os dígitos. Se existe uma classe para dígitos, também deve ter uma para letras, certo?

Para ver isso em um exemplo real, vamos olhar novamente no nosso arquivo CSV. Repare que temos uma coluna que possui uma data extensa: `11 de Abril de 1995`. Que tal definir uma regex para esse padrão textual?

Vamos copiar a data e colocar no nosso formulário. É sempre útil analisar a regex por partes e testar passo a passo. O início é bem simples, temos dois dígitos e já sabemos descrever essa classe:

```
\d\d
```

No entanto, para ser uma data, isso é muito abrangente e por isso definimos nossa própria classe de dígitos, mais específica:

```
[0-3]?\d
```

Isso pega os dias com um ou dois dígitos, mas o primeiro dia pode ser apenas 0, 1, 2 ou 3. Dentro da classe até podemos deixar isso mais explicito:

```
[0123]?\d
```

Trabalhando com espaços

Depois do dia, vem um *espaço* e a sílaba `de`. Como poderíamos definir um espaço dentro de uma regex? E se não for o espaço e sim um *tab*? Felizmente já existe uma classe predefinida, a `\s`. `\s` significa *whitespace.* A definição do `\s` pode variar um pouco entre as implementações, mas normalmente é um atalho para `[ \t\r\n\f]` onde:

- O primeiro caractere é um espaço branco.
- `\t` é um *tab*.
- `\r` é *carriage return*.
- `\n` é *newline*.
- `\f` é *form feed*.

É muito coisa para se lembrar, então é melhor usar o `\s`. Vamos aplicar isso na nossa regex, já usando um *quantifier*, pois podemos ter mais do que um espaço:

```
[0123]?\d\s{1,}
```

Nos colchetes colocamos `1,` que significa um ou mais. Novamente existe um atalho, já que isso é muito comum:

```
[0123]?\d\s+
```

O símbolo `+` é um outro atalho para definir a quantidade e agora já conhecemos todos:

- `?` - zero ou uma vez.
- `*` - zero ou mais vezes.
- `+` - uma ou mais vezes.
- `{n}` - exatamente n vezes.
- `{n,}` - no mínimo n vezes.
- `{n,m}` - no mínimo n vezes, no máximo m vezes.

Agora sim podemos continuar com o valor literal `de` e o mês. O primeiro é muito simples pois basta colocar a palavra seguida por um ou mais *whitespaces*:

```
[0123]?\d\s+de\s+
```

Dá *match* em: `21 de`

Classes de letras

Para descrever o mês, devemos usar uma nova *classe de letras*, seguem alguns exemplos:

- `[A-Z]` significa de A até Z, sempre maiúscula.
- `[a-z]` significa de a até z, sempre minúscula,
- `[A-Za-z]` significa A-Z ou a-z.
- `[abc]` significa a, b ou c.

Sabendo disso, podemos combinar a classe com um *quantifier*. Vamos começar com uma letra maiúscula ,seguida por uma quantidade de letras minúsculas, adicionando ainda a letra `ç` (do mês *março*):

```
[A-Z][a-zç]+
```

Também podemos ser mais restritivos ainda, como um mês escrito tem no máximo 8 caracteres (depois da primeira letra), temos:

```
[0123]?\d\s+de\s+[A-Z][a-zç]{1,8}
```

Dá *match* em: `21 de Maio`

Continuando, tem mais uma vez a sílaba `de` , repetindo a expressão anterior:

```
[0123]?\d\s+de\s+[A-Z][a-zç]{1,8}\s+de\s+
```

Dá *match* em: `21 de Maio de`

Por fim podemos declarar o ano que é composto por quatro dígitos:

```
[0123]?\d\s+de\s+[A-Z][a-zç]{1,8}\s+de\s+\d\d\d\d
```

Dá *match* em: `21 de Maio de 1993`

Daria também para melhorar um pouco mais a expressão, deixando claro que o ano deve começar com 1 ou 2 seguido por 3 dígitos:

```
[0123]?\d\s+de\s+[A-Z][a-zç]{1,8}\s+de\s+[12]\d{3}
```

Dá *match* em: `21 de Maio de 1993`

Por fim, não podemos esquecer de mencionar a classe de *word char*. Um *word char* é apresentado com `\w` e é um atalho para `[A-Za-z0-9_]`.