CSM 61B Spring 2018

Regex Guide

Mentoring Extra: 2018

1 Introduction

A regular expression (regex) is an expression that defines a search pattern which is then used to match text. Common uses of regular expressions include: data scraping, parsing, find and replace, and syntax highlighting. A regular expression consists of a series of simple literal characters as well as special metacharacters. Literal characters match exactly the text that they represent. For example, the regex "**a**" will only match the text "**a**". Metacharacters, on the other hand, represent more powerful/generalized patterns. For example, the regex "**.**" (explained below) will match any single character: "**a**", "**b**", "**3**", " ". Below is a table that will show and explain some of the mechanics and metacharacters of basic regex.

Consider 2 regular expressions R_1 and R_2

Operation	Definition	Explanation	Order	Example	Matches
concatenation	R_1R_2	Matches what R_1 matches followed by what R_2 matches	3	AB $ABCD$	AB $ABCD$
union (or)	$R_1\|R_2$	Matches what R_1 matches OR matches what R_2 matches	4	$A\|B$ $ABC\|DEF$	A, B ABC, DEF
closure (zero or more)	R_1*	Matches 0 or more occurrences of R_1	2	$A*$ $AB*C$	A, AA, AAA, etc. $AC, ABC, ABBC$, etc.
parenthesis	(R_1)	Groups the regex defined by R_1 into a single entity	1	$(AB)*C$ $AB(C\|D)EF$	$C, ABC, ABABC$, etc. $ABCEF, ABDEF$

2 Useful Shorthands and Tricks

Operation	Definition	Explanation	Example	Matches
digit	$\backslash d$	Represents the character class of digits: [0-9]	$A\backslash dB$	A0B, A1B, ..., A9B
word	$\backslash w$	Represents the character class of word characters: [a-zA-Z0-9_]	$\backslash w+$	camelCase, under_score, nUmb3r5_2
optional	?	0 or 1 instance	AB?C (AB)?C	ABC, AC ABC, C
quantifier	{x} {x,} {x,y}	x times x or more times x to y times (inclusive)	(AB){3} $\backslash d${2,} (A$\backslash w$B){2,4}	ABABAB 12, 909, 4444, etc. A_BA8B, AABABBA4B, AxBAyBAzBA0B

3 Warm Up

3.1 Match `a` followed by at least one instance of `b` followed by a `c`. As an example, we would match `abbc` and `abc` but not `ac`.

```
ab+c
```

3.2 Match all words that end in `cake`. `cake`, `chocolate cake`, and `#cake` would all be examples of matches.

```
.*cake
```

3.3 Match digits repeated between 2 and 5 times.

```
(\d){1,5}
```

3.4 Match alphanumeric sequences.

```
[a-z0-9]+
```

4 Pattern Matching

4.1 Match emails of the form `[username]@[domain].edu`, where usernames are alphanumeric sequences. Periods and underscores are also allowed.

```
[\w\.]*@[\w]+.com
```

4.2 Match IICCAAAAACAN, ICACACACACA, IANANCA, AICICICAICAN

```
[ICAN]*
```

4.3 Which of the following matches the regex pattern `[BORF].`?

```
Bo    BORF    B.    OF!    R2    BORD
```

```
Bo R2
```

4.4 Which of the following matches the regex pattern `(RE|QR)[QUAT]*`?

```
QUART    REQR    REAUT    RER    RQT    QRUTA
```

```
REAUT QRUTA
```

5 Tips and Tricks

5.1 Match integers and floats, which may be positive or negative. No commas may be present in the number. For instance, we would match 3.1415, -255.34, 128, but not 123,340.00 or -zzz.

```
-?\d+(\.)?\d+
```

Negative numbers are allowed here, thus we include a - to account for this. We follow it by a ?, which allows for 0 or 1 occurrences of the preceding character. Then we allow any number of digits before and after the decimal point. Similar to before, there can only be at most 1 decimal point, so we use ?.

5.2 Match HTML tags, such as `<p>Hello</p>`.

```
<([\w]+)>(.*)<\/\1>
```

The `<([\w]+)>` part takes care of the opening HTML tag; we capture the start tag by wrapping parentheses around regex that matches one or more word characters. The second set of parentheses matches the content between the tags, which may be empty. Note that we must escape the /. A neat trick is that we can reference captures as ıwhere i is the ith capturing group. This is necessary because the opening and closing tags must be the same.

5.3 Match phone numbers of the forms xxx-xxx-xxxx, (xxx)xxx-xxxx, xxx xxx xxxx, xxxxxxxxxx, and x xxx xxx xxxx

```
?[\s-]?\(?(\d{3})\)?[\s-]?\d{3}[\s-]?\d{4}
```

The 1 is optional. We then match an optional space or dash. Parenthesis around the area code are optional. Following this, we continue a pattern of 3 or 4 digits followed by an optional space or dash.