Text is Best , Text Markup Languages - Regular Expressions (regexps)

Regular Expressions (Regex)

Regular Expression, or *regex* or *regexp* in short, is extremely and amazingly powerful in searching and manipulating text strings, particularly in processing text files. One line of regex can easily replace several dozen lines of programming codes.

Regex is supported in all the scripting languages (such as Perl, Python, PHP, and JavaScript); as well as general purpose programming languages such as Java; and even word processors such as Word for searching texts. Getting started with regex may not be easy due to its geeky syntax, but it is certainly worth the investment of your time.

1. Regex By Examples

This section is meant for those who need to refresh their memory. For novices, go to the next section to learn the syntax, before looking at these examples.

1.1 Regex Syntax Summary

- **Character**: All characters, except those having special meaning in regex, matches themselves. E.g., the regex x matches substring "x"; regex 9 matches "9"; regex = matches "="; and regex @ matches "@".
- **Special Regex Characters**: These characters have special meaning in regex (to be discussed below): ., +, *, ?, ^, $, (,), [,], {, }, |, \.
- **Escape Sequences (\char)**:
  - To match a character having special meaning in regex, you need to use a escape sequence prefix with a backslash (\). E.g., \. matches "."; regex \+ matches "+"; and regex \(matches "(".
  - You also need to use regex \\ to match "\" (back-slash).
  - Regex recognizes common escape sequences such as \n for newline, \t for tab, \r for carriage-return, \nnn for a up to 3-digit octal number, \xhh for a two-digit hex code, \uhhhh for a 4-digit Unicode, \uhhhhhhhh for a 8-digit Unicode.

- **A Sequence of Characters (or String)**: Strings can be matched via combining a sequence of characters (called sub-expressions). E.g., the regex Saturday matches "Saturday". The matching, by default, is case-sensitive, but can be set to case-insensitive via *modifier*.
- **OR Operator (|)**: E.g., the regex four|4 accepts strings "four" or "4".
- **Character class (or Bracket List)**:
  - [...]: Accept ANY ONE of the character within the square bracket, e.g., [aeiou] matches "a", "e", "i", "o" or "u".
  - [.-.] (Range Expression): Accept ANY ONE of the character in the *range*, e.g., [0-9] matches any digit; [A-Za-z] matches any uppercase or lowercase letters.
  - [^...]: NOT ONE of the character, e.g., [^0-9] matches any non-digit.
  - Only these four characters require escape sequence inside the bracket list: ^, -,], \.
- **Occurrence Indicators (or Repetition Operators)**:
  - +: one or more (1+), e.g., [0-9]+ matches one or more digits such as '123', '000'.
  - *: zero or more (0+), e.g., [0-9]* matches zero or more digits. It accepts all those in [0-9]+ plus the empty string.
  - ?: zero or one (optional), e.g., [+-]? matches an optional "+", "-", or an empty string.
  - {m,n}: m to n (both inclusive)
  - {m}: exactly m times
  - {m,}: m or more (m+)
- **Metacharacters**: matches a character
  - . (dot): ANY ONE character except newline. Same as [^\n]
  - \d, \D: ANY ONE digit/non-digit character. Digits are [0-9]
  - \w, \W: ANY ONE word/non-word character. For ASCII, word characters are [a-zA-Z0-9_]
  - \s, \S: ANY ONE space/non-space character. For ASCII, whitespace characters are [\n\r\t\f]
- **Position Anchors**: does not match character, but position such as start-of-line, end-of-line, start-of-word and end-of-word.
  - ^, $: start-of-line and end-of-line respectively. E.g., ^[0-9]$ matches a numeric string.
  - \b: boundary of word, i.e., start-of-word or end-of-word. E.g., \bcat\b matches the word "cat" in the input string.

- \B: Inverse of \b, i.e., non-start-of-word or non-end-of-word.
- \<, \>: start-of-word and end-of-word respectively, similar to \b. E.g., \<cat\> matches the word "cat" in the input string.
- \A, \Z: start-of-input and end-of-input respectively.

- **Parenthesized Back References**:
  - Use parentheses ( ) to create a back reference.
  - Use $1, $2, ... (Java, Perl, JavaScript) or \1, \2, ... (Python) to retreive the back references in sequential order.
- **Laziness (Curb Greediness for Repetition Operators)**: *?, +?, ??, {m,n}?, {m,}?

1.2 Example: Numbers [0-9]+ or \d+

1. A regex (*regular expression*) consists of a sequence of *sub-expressions*. In this example, [0-9] and +.
2. The [...], known as *character class* (or *bracket list*), encloses a list of characters. It matches any SINGLE character in the list. In this example, [0-9] matches any SINGLE character between 0 and 9 (i.e., a digit), where dash (-) denotes the *range*.
3. The +, known as *occurrence indicator* (or *repetition operator*), indicates one or more occurrences (1+) of the previous sub-expression. In this case, [0-9]+ matches one or more digits.
4. A regex may match a portion of the input (i.e., substring) or the entire input. In fact, it could match zero or more substrings of the input (with global modifier).
5. This regex matches any numeric substring (of digits 0 to 9) of the input. For examples,
   a. If the input is "abc123xyz", it matches substring "123".
   b. If the input is "abcxyz", it matches nothing.
   c. If the input is "abc00123xyz456_0", it matches substrings "00123", "456" and "0" (three matches).

   Take note that this regex matches number with leading zeros, such as "000", "0123" and "0001", which may not be desirable.
6. You can also write \d+, where \d is known as a *metacharacter* that matches any digit (same as [0-9]). There are more than one ways to write a regex! Take note that many programming languages (C, Java, JavaScript, Python) use backslash \ as the prefix for escape sequences (e.g., \n for newline), and you need to write "\\d+" instead.