Regular Expressions in **R**

**Regular Expression R Functions**

* **grep**(value = FALSE) returns a vector of the indices of the elements of ***x*** that yielded a match (or not, for invert = TRUE.
* **grep**(value = TRUE) returns a character vector containing the selected elements of ***x*** (after coercion, preserving names but no other attributes).
* **grepl** returns a logical vector (match or not for each element of ***x***).
* **sub** returns a character vector of the same length and with the same attributes as ***x*** with the first occurrence of the pattern replaced
* **gsub** returns a character vector of the same length and with the same attributes as ***x*** with all occurrences of the pattern replaced
* **regexpr** returns an integer vector of the same length as **text** giving the starting position of the first match or -1 if there is none, with attribute "match.length", an integer vector giving the length of the matched text (or -1 for no match).
* **gregexpr** returns a list of the same length as **text**, each element of which is an integer vector with the starting positions of every (disjoint) match in that element of **text** or -1 if there is not match in that element of **text**, with attribute "match.length", an integer vector giving the length of the matched text (or -1 for no match).
* **regexec** returns a list of the same length as ***text***, each element of which is either -1 if there is no match, or a sequence of integers with the starting positions of the match and all substrings corresponding to parenthesized subexpressions of pattern, with attribute "match.length" a vector giving the lengths of the matches (or -1 for no match).
* **regmatches** basically allows you to use regexpr, gregexpr, or regexec as though they had grep's *value = T* setting and also an *invert* parameter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Function** | **Return** | **Multiple**  **matches?** | **Subpatterns?** |
| Find or return the elements with matches | grep(value = T) | The elements that match | No | No |
| grep(value = F) | Indices of the elements that match | No | No |
| grepl | T/F - is a match in each element | No | No |
|  |  |  |  |  |
| Find the position of the match in the element | regexpr | Starting position and length | No | No |
| gregexpr | Starting position and length | Yes | No |
| regexec | Starting position and length | Yes | Yes |
|  |  |  |  |  |
| Replace the match in the element | sub | Modified elements | No | No |
| gsub | Modified elements | Yes | No |
|  |  |  |  |  |
| Return the match from the element | regmatches(regexpr) | The match | No | No |
| regmatches(gregexpr) | The match | Yes | No |
| regmatches(regexec) | The match | Yes | Yes |

**Regular Expression Details**

* **Most characters**, including all letters and digits, are regular expressions that match themselves.
* **Metacharacters** have special meanings. They include . \ | ( ) [ { ^ $ \* + ?
* Putting a **backslash** before a character changes the meaning of that character
  + Backslash Escape sequences for weird characters:
    - \a BEL – ring a warning or alert bell
    - \e ESC – the character created by hitting the escape key on a keyboard
    - \f FF – form feed (like a section break or page break)
    - \n LF – line feed (same as newline)
    - \r CR – carriage return
    - \t TAB – a tab space
* **Character classes**
  + Put a set of characters between square brackets. This matches any single character. If you start with ^, matching is inverted.
  + Example 1:
    - 0123 matches the exact string 0123
    - [0123456789] matches any single digit
  + Example 2:
    - abc matches the exact string abc
    - [abc] matches any single character that is an a, b, or c
    - [^abc] matches anything except the characters a, b or c.
  + Most special characters lose their special meaning inside classes. Only ^ - \ ] are special inside character classes.
* **Character ranges**
  + In a character class (that is, something surrounded by square brackets), putting a hyphen between two characters is the same as listing all characters from the first to the last. Note that the order of the character encoding will be different on different platforms and whatnot.
  + Note: I don't know whether a hyphen-created character range must be enclosed in square brackets. I know that, if it is, it doesn't have to be alone. For example, [a-z5] will match any lower case letter or the digit 5
* **Named character classes**
  + There are some special character classes. These always begin and end with square brackets and colons. In order to use them, you may have to put them inside another set of square brackets. eg, [[:lower:]] is equivalent to [a-z]
  + **Basic named character classes**
    - [:lower:] Lower-case letters
    - [:upper:] Upper-case letters
    - [:digit:] Digits
    - [:space:] Space characters: tab, newline, vertical tab, form feed, carriage return, space, etc.
    - [:blank:] Blank characters: space and tab, and possibly others such as non-breaking space
    - [:punct:] Punctuation characters: ! " # $ % & ' ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ ` { | } ~.
  + **Combo named character classes**
    - [:alpha:] Alphabetic characters: [:lower:] and [:upper:]
    - [:alnum:] Alphanumeric characters: [:alpha:] and [:digit:]
    - [:graph:] Graphical characters: [:alnum:] and [:punct:]
* **Special things that match stuff**
  + . any single character
  + \w word character - any letter, digit, or the underscore
  + \W nonword character - any character that is not a letter, digit, or underscore
  + \d digits
  + \D non-digits
  + \s space character
  + \S non-space character
* **Metacharacters that control achnoring**
  + ^ force match at the beginning of a string
  + $ force match at the end of a line
  + \< force match to the beginning of a word
  + \> force match to the end of a word
  + \b force match to beginning or end of a word
  + \B force match to the middle of a word (not beginning or end)
* **Repetition quantifiers**
  + Put these at the end of an expression
  + ? The preceding item is optional and will be matched at most once.
  + \* The preceding item will be matched zero or more times.
  + + The preceding item will be matched one or more times.
  + {n} The preceding item is matched exactly n times.
  + {n,} The preceding item is matched n or more times.
  + {n,m} The preceding item is matched at least n times, but not more than m times.
* **Backreferences**
  + \1 Match to the first parenthetical expression in this whole expression
  + \2 Match to the second parenthetical expression in this whole expression
  + ...and so on
* **Alternation** (Logical OR)
  + | the alternation (OR) operator - Match either the expression before or the one after
* **Parentheses**
  + Parentheses can be used to group a bunch of stuff together.
  + I’m still not clear on how this works.

**Replacement Expression Details**

* **Simple Strings**
  + The replacement can just be a simple string.
  + In that case, matches are replaced by the simple string
* **Replacement Expressions**
  + The replacement can also be composed of simple strings, backreferences, and case converters
  + **Backreferences**
    - \1 through \9 can be used to refer to the parenthesized subexpressions of the ***pattern***.
    - This is only available when fixed = FALSE
    - This inserts into the location of the match the part of the match found by the relevant parenthetical expression
  + **Case Converters**
    - These are only available when perl = TRUE
    - \U converts whatever comes after it (such as backreferences) to upper case
    - \L converts whatever comes after it to lower case
    - \E ends any case conversion (ie, closes out the toupper and tolower effects)