# Basic Regular Expressions in R

Cheat Sheet

## Character Classes

| | |
|---|---|
| [[:digit:]] or \\d | Digits; `[0-9]` |
| \\D | Non-digits; `[^0-9]` |
| [[:lower:]] | Lower-case letters; `[a-z]` |
| [[:upper:]] | Upper-case letters; `[A-Z]` |
| [[:alpha:]] | Alphabetic characters; `[A-z]` |
| [[:alnum:]] | Alphanumeric characters `[A-z0-9]` |
| \\w | Word characters; `[A-z0-9_]` |
| \\W | Non-word characters |
| [[:xdigit:]] or \\x | Hexadec. digits; `[0-9A-Fa-f]` |
| [[:blank:]] | Space and tab |
| [[:space:]] or \\s | Space, tab, vertical tab, newline, form feed, carriage return |
| \\S | Not space; `[^[:space:]]` |
| [[:punct:]] | Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~` |
| [[:graph:]] | Graphical char.; `[[:alnum:][:punct:]]` |
| [[:print:]] | Printable characters; `[[:alnum:][:punct:]\\s]` |
| [[:cntrl:]] or \\c | Control characters; `\n, \r` etc. |

## Special Metacharacters

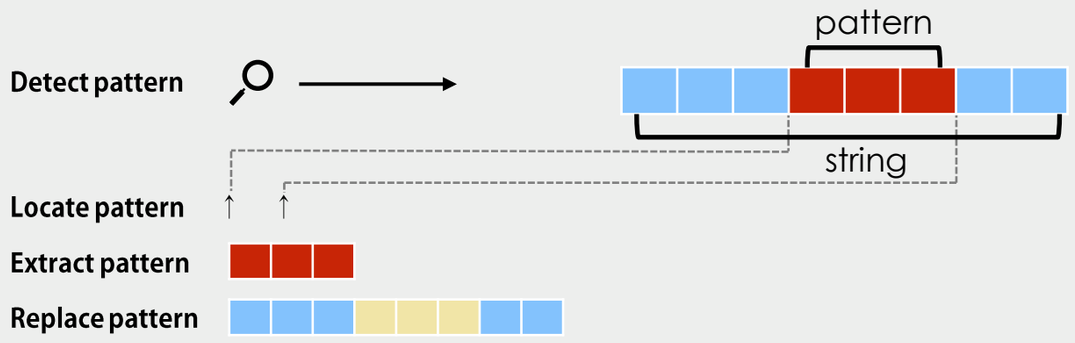
| | |
|---|---|
| \n | New line |
| \r | Carriage return |
| \t | Tab |
| \v | Vertical tab |
| \f | Form feed |

## Lookaraounds and Conditionals*

| | |
|---|---|
| (?=) | Lookahead (requires PERL = TRUE), e.g. `(?=yx)`: position followed by `'xy'` |
| (?!) | Negative lookahead (PERL = TRUE); position NOT followed by pattern |
| (?<=) | Lookbehind (PERL = TRUE), e.g. `(?<=yx)`: position following `'xy'` |
| (?<!) | Negative lookbehind (PERL = TRUE); position NOT following pattern |
| ?(if)then | If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause |
| ?(if)then\|else | If-then-else-condition (PERL = TRUE) |

*see, e.g. http://www.regular-expressions.info/lookaround.html
http://www.regular-expressions.info/conditional.html

# Functions for Pattern Matching




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

## Detect Patterns

**grep(pattern, string)**
```
[1] 1 3
```

**grep(pattern, string, value = TRUE)**
```
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics“
```

**grepl(pattern, string)**
```
[1]  TRUE FALSE  TRUE
```

**stringr::str_detect(string, pattern)**
```
[1]  TRUE FALSE  TRUE
```

## Locate Patterns

**regexpr(pattern, string)**
find starting position and length of first match

**gregexpr(pattern, string)**
find starting position and length of all matches

stringr::**str_locate(string, pattern)**
find starting and end position of first match

stringr::**str_locate_all(string, pattern)**
find starting and end position of all matches

## Split a String using a Pattern

**strsplit(string, pattern)** or stringr::**str_split(string, pattern)**

## Extract Patterns

**regmatches(string, regexpr(pattern, string))**
extract first match `[1] "tam" "tim"`

**regmatches(string, gregexpr(pattern, string))**
extracts all matches, outputs a list
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

stringr::**str_extract(string, pattern)**
extract first match `[1] "tam" NA "tim"`

stringr::**str_extract_all(string, pattern)**
extract all matches, outputs a list

stringr::**str_extract_all(string, pattern, simplify = TRUE)**
extract all matches, outputs a matrix

stringr::**str_match(string, pattern)**
extract first match + individual character groups

stringr::**str_match_all(string, pattern)**
extract all matches + individual character groups

## Replace Patterns

**sub(pattern, replacement, string)**
replace first match

**gsub(pattern, replacement, string)**
replace all matches

stringr::**str_replace(string, pattern, replacement)**
replace first match

stringr::**str_replace_all(string, pattern, replacement)**
replace all matches

## Character Classes and Groups

| | |
|---|---|
| . | Any character except `\n` |
| \| | Or, e.g. `(a\|b)` |
| […] | List permitted characters, e.g. `[abc]` |
| [a-z] | Specify character ranges |
| [^…] | List excluded characters |
| (…) | Grouping, enables back referencing using `\\N` where `N` is an integer |

## Anchors

| | |
|---|---|
| ^ | Start of the string |
| $ | End of the string |
| \\b | Empty string at either edge of a word |
| \\B | NOT the edge of a word |
| \\< | Beginning of a word |
| \\> | End of a word |

## Quantifiers

| | |
|---|---|
| * | Matches at least 0 times |
| + | Matches at least 1 time |
| ? | Matches at most 1 time; optional string |
| {n} | Matches exactly n times |
| {n,} | Matches at least n times |
| {,n} | Matches at most n times |
| {n,m} | Matches between n and m times |

## General Modes

By default R uses *POSIX extended regular expressions*. You can switch to *PCRE regular expressions* using `PERL = TRUE` for base or by wrapping patterns with `perl()` for stringr.

All functions can be used with literal searches using `fixed = TRUE` for base or by wrapping patterns with `fixed()` for stringr.

All base functions can be made case insensitive by specifying `ignore.cases = TRUE`.

## Escaping Characters

Metacharacters (`. * +` etc.) can be used as literal characters by escaping them. Characters can be escaped using `\\` or by enclosing them in `\\Q...\\E`.

## Case Conversions

Regular expressions can be made case insensitive using `(?i)`. In backreferences, the strings can be converted to lower or upper case using `\\L` or `\\U` (e.g. `\\L\\1`). This requires `PERL = TRUE`.

## Greedy Matching

By default the asterisk `*` is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding `?`, i.e. `*?`.

Greedy mode can be turned off using `(?U)`. This switches the syntax, so that `(?U)a*` is lazy and `(?U)a*?` is greedy.

## Note

Regular expressions can conveniently be created using rex::**rex()**.

CC BY Ian Kopacka • ian.kopacka@ages.at

Updated: 09/16