Strings and Regular Expressions

David Gerard

2019-02-21

Learning Objectives

- Manipulating strings with the stringr package.
- Regular expressions
- Chapter 14 of RDS.
- Work with Strings Cheatsheet.

Strings

- In R, strings (also called "characters") are created and displayed within quotes:

```
x <- "I am a string!"
x
```

```
## [1] "I am a string!"
```

- Anything within quotes is a string, even numbers!

```
y <- "3"
class(y)
```

```
## [1] "character"
```

- You can have a vector of strings.

```
x <- c("I", "am", "a", "string", "vector")
x[2:3]
```

```
## [1] "am" "a"
```

- The backslash `"\"` means that what is after the backslash is special in some way. For example, if you want to put a quotation mark in a string, you can "escape" the quotation mark with a backslash.

```
x <- "As Tolkein said, \"Not all those who wonder are lost\""
writeLines(x)
```

```
## As Tolkein said, "Not all those who wonder are lost"
```

- Above, `writeLines()` will print out the string itself. `print()` will print out the printed representation of the string (with backslashes and all).

```
print(x)
```

```
## [1] "As Tolkein said, \"Not all those who wonder are lost\""
```

- `"\n"` represents a new line.

```
x <- "Not all those\nwho wonder are lost."
writeLines(x)
```

```
## Not all those
## who wonder are lost.
```

- `"\t"` represents a tab.

```
x <- "Not all those\twho wonder are lost."
writeLines(x)
```

```
## Not all those    who wonder are lost.
```

- You can add any Unicode character with a `\u` followed by the hexadecimal unicode representation of that character.

```
mu <- "\u00b5"
writeLines(mu)
```

```
## µ
```

stringr Intro

- The stringr package contains a lot of convenience functions for manipulating strings (and they are a lot more user friendly than base R's string manipulation functions like `grep()` and `gsub()`).
- stingr is not part of the tidyverse so you have to load it separately.

```
library(tidyverse)
library(stringr)
```

- All of stringr's functions begin with "`str_`", so you can press tab after typing "`str_`" and a list of possible string manipulation functions will pop up (in RStudio).
- For example, to get the number of characters in a string, use `str_length()`.

```
str_length("Upon the hearth the fire is red,")
```

```
## [1] 32
```

Combining Strings

- Combine strings with `str_c()`.

```r
x <- "Faithless is he that says"
y <- "farewell when the road darkens."
str_c(x, y)
```

```
## [1] "Faithless is he that saysfarewell when the road darkens."
```

- The default is to separate strings by nothing, but you can use `sep` to change the separator.

```r
str_c(x, y, sep = " ")
```

```
## [1] "Faithless is he that says farewell when the road darkens."
```

- Just like `c()`, `str_c()` can take multiple arguments.

```r
str_c("Short", "cuts", "make", "long", "delays.", sep = " ")
```

```
## [1] "Short cuts make long delays."
```

- If you provide `str_c()` a vector of arguments, it will vectorize the combining unless you provide a `collapse` argument.

```r
x <- c("Short", "cuts", "make", "long", "delays.")
str_c(x, "LOTR", sep = " ")
```

```
## [1] "Short LOTR"   "cuts LOTR"    "make LOTR"    "long LOTR"
## [5] "delays. LOTR"
```

```r
str_c(x, collapse = " ")
```

```
## [1] "Short cuts make long delays."
```

- Combining with `NA` results in `NA`:

```r
str_c("Faithless is he that says", NA)
```

```
## [1] NA
```

Extracting substrings

- `str_sub()` extracts a substring between the location of two characters.

```r
x <- "The Road goes ever on and on"
str_sub(x, start = 2, end = 6)
```

```
## [1] "he Ro"
```

- Replace substrings with assignment

```
str_sub(x, start = 2, end = 6) <- " Tolkein "
x
```

```
## [1] "T Tolkein ad goes ever on and on"
```

- **Exercise**: Reproduce this quote

  > But under a tall tree I will lie, And let the clouds go sailing by.

  with these strings

```
w <- "But under a tall tree"
x <- "FRELL I will lie"
y <- "and let clouds go"
z <- "sailing by."
```

Regular Expressions

- Regular expressions (regex or regexp) is a syntax for pattern matching in strings.
- We'll use `str_replace()` and `str_replace_all()` to demonstrate using regex in stringr. These functions search for a pattern and then replace it with another string.
- But wherever there is a `pattern` argument in a stringr function, you can use regex (to extract strings, get a logical if there is a match, etc...).
- Basic usage: finds exact match of string.

```
x <- "Ho! Ho! Ho! to the bottle I go to heal my heart and drown my woe."
str_replace_all(x, "hea", "XX")
```

```
## [1] "Ho! Ho! Ho! to the bottle I go to XXl my XXrt and drown my woe."
```

- A period "`.`" matches any character.

```
str_replace_all(x, "hea.", "XX")
```

```
## [1] "Ho! Ho! Ho! to the bottle I go to XX my XXt and drown my woe."
```

- You can "escape" a period with two backslashes "`\\`" to match periods.

```
str_replace_all(x, ".", "X") ## Matches everything
```

```
## [1] "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

```
str_replace_all(x, "\\.", "X") ## Matches the only period
```

```
## [1] "Ho! Ho! Ho! to the bottle I go to heal my heart and drown my woeX"
```

- To match a backslash, you need four backslashes (to escape the escape).

```
y <- "Rain\\may\\fall\\and\\wind\\may\\blow"
writeLines(y)
```

```
## Rain\may\fall\and\wind\may\blow
```

```
str_replace_all(y, "\\\\", "XX")
```

```
## [1] "RainXXmayXXfallXXandXXwindXXmayXXblow"
```

- *Important note*: The actual regular expressions above are strings themselves, and so you view them with `writeLines()`. So using "`\\.`" as the pattern argument in R results in the regular expression "`\.`".
- **Exercise**: Construct a regular expression to match this string:

```
## \.\.\.
```

- **Exercise**: Use one function call to replace `"back"` and `"lack"` with `"foo"`.

```
x <- "but better is Beer if drink we lack, and Water Hot poured down the back."
```

Anchoring

- You can anchor the pattern to only match the start or end of a string.
  - `^` matches only the start of a string.
  - `$` matches only the end of a string.

```
x <- c("But", "under", "a", "tall", "tree", "I", "will", "lie")
str_replace(x, "^t", "XX")
```

```
## [1] "But"   "under" "a"     "XXall" "XXree" "I"     "will"  "lie"
```

```
str_replace(x, "t$", "XX")
```

```
## [1] "BuXX"  "under" "a"     "tall"  "tree"  "I"     "will"  "lie"
```

- Use both to match only a complete string.

```
x <- c("apple pie", "apple", "apple cake")
str_replace_all(x, "apple", "XX")
```

```
## [1] "XX pie"  "XX"      "XX cake"
```

```
str_replace_all(x, "^apple$", "XX")
```

```
## [1] "apple pie"  "XX"         "apple cake"
```

- **Exercise**: Use `str_replace()` to replace all four letter words beginning with an `"a"` with `"foo"` in the following list

```r
x <- c("apple", "barn", "ape", "cart", "alas", "pain", "ally")
```

Special Characters

- We'll use this character vector for practice:

```r
x <- c("Abba: 555-1234", "Anna: 555-0987", "Andy: 555-7654")
```

- `\\d`: matches any digit.

```r
str_replace(x, "\\d\\d\\d-\\d\\d\\d\\d", "XXX-XXXX")
```

```
## [1] "Abba: XXX-XXXX" "Anna: XXX-XXXX" "Andy: XXX-XXXX"
```

- `\\s`: matches any white space (e.g. space, tab, newline).

```r
str_replace(x, "\\s", "X")
```

```
## [1] "Abba:X555-1234" "Anna:X555-0987" "Andy:X555-7654"
```

- `[abc]`: matches `a`, `b`, or `c`.

```r
str_replace(x, "A[bn][bn]a", "XXXX")
```

```
## [1] "XXXX: 555-1234" "XXXX: 555-0987" "Andy: 555-7654"
```

- `[^abc]`: matches anything except `a`, `b`, or `c`.

```r
str_replace(x, "A[^b]", "XXXX")
```

```
## [1] "Abba: 555-1234"   "XXXXna: 555-0987" "XXXXdy: 555-7654"
```

- `abc|xyz`: matches either `abc` or `xyz`. This is called *alternation*
- You can use parentheses to control where the alternation occurs.
  - `a(bc|xy)z` matches either `abcz` or `axyz`.

```r
str_replace(x, "An(na|dy)", "XXXX")
```

```
## [1] "Abba: 555-1234" "XXXX: 555-0987" "XXXX: 555-7654"
```

- To ignore case, place a `(?i)` before the regex.

```r
str_replace("AB", "ab", "X")
```

```
## [1] "AB"
```

```
str_replace("AB", "(?i)ab", "X")
```

```
## [1] "X"
```

- **Exercise**: Create separate regular expressions to find all words that:

  1. Start with a vowel. Test on

```
x1 <- c("abba", "cat", "eal", "ion", "oops", "Uganda", "Anna", "dog")
```

  2. That end in consonants. (Hint: thinking about matching "not"-vowels.) test on

```
x2 <- c("bob", "Anna", "dog")
```

  3. End with `ed`, but not with `eed`. Test on

```
x3 <- c("tired", "need", "bad", "rod")
```

  4. End with `ing` or `ise`. Test on

```
x4 <- c("paradise", "firing", "jaded", "kin")
```

Repetition

- Can match a pattern multiple times in a row:
  - `?`: 0 or 1
  - `+`: 1 or more
  - `*`: 0 or more

```
x <- c("A", "AA", "AAA", "AAAA", "B", "BB")
str_replace_all(x, "^A?", "X")
```

```
## [1] "X"    "XA"   "XAA"  "XAAA" "XB"   "XBB"
```

```
str_replace_all(x, "^A+", "X")
```

```
## [1] "X"  "X"  "X"  "X"  "B"  "BB"
```

```
str_replace_all(x, "^A*", "X")
```

```
## [1] "X"   "X"   "X"   "X"   "XB"  "XBB"
```

- A more realistic example:

```
str_replace_all("color and colour", "colou?r", "X")
```

```
## [1] "X and X"
```

- Control exactly how many repetitions allowed in a match:
  - `{n}`: exactly `n`.
  - `{n,}`: `n` or more.
  - `{0,m}`: at most `m`.
  - `{n,m}`: between `n` and `m`.

```
str_replace_all(x, "A{2}", "X")
```

```
## [1] "A"  "X"  "XA" "XX" "B"  "BB"
```

```
str_replace_all(x, "A{2,}", "X")
```

```
## [1] "A"  "X"  "X"  "X"  "B"  "BB"
```

```
str_replace_all(x, "A{0,2}", "X")
```

```
## [1] "XX"   "XX"   "XXX"  "XXX"  "XBX"  "XBXBX"
```

```
str_replace_all(x, "A{3,4}", "X")
```

```
## [1] "A"  "AA" "X"  "X"  "B"  "BB"
```

- Regex will automatically match the longest string possible.

```
str_replace("AAAA", "A*", "X")
```

```
## [1] "X"
```

- **Exercise**: Create regular expressions to find all words that:
  1. Start with three consonants. Test on

```
x1 <- c("string", "priority", "value", "distinction")
```

  2. Have three or more vowels in a row. Test on

```
x2 <- c("honorific", "delicious", "priority", "queueing")
```

  3. Have two or more vowel-consonant pairs in a row. Test on

```
x3 <- c("honorific", "sam", "prior")
```

Grouping and Backreferences

- Parentheses create a numbered group that you can then back reference with `\\1` for the match in the first parentheses, `\\2` in the second parentheses, etc...

```r
str_replace("cococola", "(..)\\1", "pepsi")
```

```
## [1] "pepsicola"
```

```r
str_replace("banana", "([aeiou][^aeiou])\\1", "XX")
```

```
## [1] "bXXa"
```

stringr tools

- There are a lot of tools, so we'll go over them briefly and do an exercise where you can use them in more detail.
- `str_to_lower()` and `str_to_upper()` convert all letters to lower or capital case.

```r
x <- "Deeds will not be less valiant because they are unpraised."
str_to_lower(x)
```

```
## [1] "deeds will not be less valiant because they are unpraised."
```

```r
str_to_upper(x)
```

```
## [1] "DEEDS WILL NOT BE LESS VALIANT BECAUSE THEY ARE UNPRAISED."
```

- `str_detect()`: Returns `TRUE` if a regex pattern matches a string and `FALSE` if it does not. Very useful for filters.

```r
## Get all John's and Joe's from the Lahman dataset
library(Lahman)
data("Master")
Master %>%
  filter(str_detect(nameFirst, "^Jo(e|hn)$")) %>%
  select(nameFirst) %>%
  head()
```

```
##   nameFirst
## 1      John
## 2       Joe
## 3       Joe
## 4       Joe
## 5       Joe
## 6      John
```

- `str_subset()`: Returns the words where there is a match. Not often as useful as `str_detect()` because you don't use it in data frames that often.

```r
str_subset(Master$nameFirst, "^Jo(e|hn)$") %>%
  head()
```

```
## [1] "John" "Joe"  "Joe"  "Joe"  "Joe"  "John"
```

- `str_count()`: Counts the occurrence of a match within a string.

```r
str_count(c("banana", "coco"), "[^aeiou][aeiou]")
```

```
## [1] 3 2
```

They count *non-overlapping* matches

```r
str_count("abababa", "aba")
```

```
## [1] 2
```

- `str_extract()`: Returns the pattern that it finds. `str_extract()` will only return the first match but `str_extract_all()` will return all matches.

```r
colorstr <- str_c("red", "blue", "yellow", "orange", "brown", sep = "|")
colorstr
```

```
## [1] "red|blue|yellow|orange|brown"
```

```r
str_extract("I like blue and brown and that's it", colorstr)
```

```
## [1] "blue"
```

```r
str_extract_all("I like blue and brown and that's it", colorstr)
```

```
## [[1]]
## [1] "blue"  "brown"
```

- `str_match()`: returns a matrix where each column is a grouped component.

```r
x <- "I like blue and brown and that's it, or black"
str_extract_all(x, "(and|or)\\s([^\\s]+)")
```

```
## [[1]]
## [1] "and brown"  "and that's" "or black"
```

```r
str_match_all(x, "(and|or)\\s([^\\s]+)")
```

```
## [[1]]
##      [,1]         [,2]  [,3]
## [1,] "and brown"  "and" "brown"
## [2,] "and that's" "and" "that's"
## [3,] "or black"   "or"  "black"
```

- Let's look at the poem "Farewell We Call to Hearth and Hall!"

```r
farewell <- c("Farewell we call to hearth and hall!
              Though wind may blow and rain may fall,
              We must away ere break of day
              Far over wood and mountain tall.")
writeLines(farewell)
```

```
## Farewell we call to hearth and hall!
##               Though wind may blow and rain may fall,
##               We must away ere break of day
##               Far over wood and mountain tall.
```

- `str_split()` will split up a string based on a character we choose.

```
## Split based on spaces
str_split(farewell, pattern = "\\s+", simplify = TRUE) ## use one or more space to split
```

```
##      [,1]       [,2] [,3]   [,4] [,5]     [,6]   [,7]    [,8]     [,9]
## [1,] "Farewell" "we" "call" "to" "hearth" "and" "hall!" "Though" "wind"
##      [,10] [,11]  [,12] [,13]  [,14] [,15]   [,16] [,17]  [,18]  [,19]
## [1,] "may" "blow" "and" "rain" "may" "fall," "We"  "must" "away" "ere"
##      [,20]   [,21] [,22] [,23] [,24]  [,25]  [,26] [,27]      [,28]
## [1,] "break" "of"  "day" "Far" "over" "wood" "and" "mountain" "tall."
```

- `str_replace()` and `str_replace_all()` will replace patterns with provided strings. So say we want to get rid of all punctuation.

```
str_split(farewell, pattern = "\\s+", simplify = TRUE) %>%
  str_replace_all("\\.|\\!|,", "")
```

```
##  [1] "Farewell" "we"       "call"     "to"       "hearth"   "and"
##  [7] "hall"     "Though"   "wind"     "may"      "blow"     "and"
## [13] "rain"     "may"      "fall"     "We"       "must"     "away"
## [19] "ere"      "break"    "of"       "day"      "Far"      "over"
## [25] "wood"     "and"      "mountain" "tall"
```

- You can use back references to populate the replacement.

```
str_replace_all("It is 1am", "(\\d+)(am|pm)", "\\2")
```

```
## [1] "It is am"
```

- More stringr options can be found in RDS.