Strings

Byteflow Dynamics

10/15/2017

String basics

To define a string, use double quotes

```
string1 <- "I am a string"

string1
```

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

If you want to include quotes in your string, use escape character "\". Use writeLines( ) to see the raw content of the string.

```
string2 <- "She said \"I am a string\""
string2
```

```
## [1] "She said \"I am a string\""
```

```
writeLines(string2)
```

```
## She said "I am a string"
```

You can store multiple strings in a vector using c

```
c("apple", "banana", "carrot")
```

```
## [1] "apple"  "banana" "carrot"
```

This is the same as

```
a <- "apple"
b <- "banana"
c <- "carrot"

c(a, b, c)
```

```
## [1] "apple"  "banana" "carrot"
```

stringr

We use a package called **stringr**. It is not part of tidyverse, so first install and load stringr.

```
library(stringr)
```

Also, install a package **htmlwidgets**

stringr functions start with *str_*. If you type str_ you will see all the functions.

String length

To check the number of characters in a string, use **str_length( )**.

```
str_length(string1)
```

```
## [1] 13
```

```
str_length(c(a, b, c))
```

```
## [1] 5 6 6
```

Combine strings

str_c( ) combines multiple strings as a single string

```
str_c("ba", "nan", "a")
```

```
## [1] "banana"
```

If you want the original strings to be separated in the resulting string, use **sep =**

```
str_c("apple", "banana", "carrot", sep = ", ")
```

```
## [1] "apple, banana, carrot"
```

You can combine multiple strings of different length, the shorter string will be repeated.

```
str_c("I have two ", c("apple", "banana", "carrot"), "s")
```

```
## [1] "I have two apples"  "I have two bananas" "I have two carrots"
```

Subset strings

str_sub( ) will extract parts of a string by specifying start and end positions.

```
str_sub(string1, end = 5)
```

```
## [1] "I am "
```

```
x <- c("apple", "banana", "carrot")
str_sub(x, 1, 3)
```

```
## [1] "app" "ban" "car"
```

Negative numbers count backwards from end

```
str_sub(x, -3, -1)
```

```
## [1] "ple" "ana" "rot"
```

You can also use str_sub to modify strings

```
str_sub(x, 1, 1) <- str_to_upper(str_sub(x, 1, 1))
x
```

```
## [1] "Apple"  "Banana" "Carrot"
```

```
str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
x
```

```
## [1] "apple"  "banana" "carrot"
```

String search with regular expression

Regular expressions describe patterns in strings, which can be used to search through strings to find ones with matching patterns.

We will use **str_view( )** to learn how regular expressions work.

Basic pattern match

str_view( ) matches patterns to extract strings. Run the following code.

```
str_view(x, "an")
```

This finds a pattern *an* in the strings.

To match any character, use **.** For example,

```
str_view(x, "a.")
```

matches any pattern with **a** followed by another character. str_view( ) only shows the first match, while str_view_all( ) shows all matches

```
str_view_all(x, "a.")
```

Question: What do you do if you want to match the character "."?

Answer: Use the escape character "\"

However, this causes a problem. Both strings and regular expressions use "\" as an escape character. So, in order to create a regular expression \., we need **"\\."**

Consider the following code

```
str_view(c("abc", "a.c", "bef"), "a\\.c")
```

If you want to match a literal \, you need a regular expression \\. To get this, we need a string that escapes both backslashes, **"\\\\"**. To match a single backslash, we need four.

```
writeLines("\\\\")
```

```
## \\
```

```
x <- "a\\b"
writeLines(x)
```

```
## a\b
```

```
str_view(x, "\\\\")
```

Exercises:

1. How would you match the sequence "'\ ?
2. What patterns will the regular expression \..\..\.. match? How would you represent it as a string?

1.

```
x <- "a\"\'\\b"
writeLines(x)
```

```
## a"'\b
```

```
str_view(x, "\"\'\\\\")
```

2.

```
x <- "a.b.c.d"
writeLines(x)
```

```
## a.b.c.d
```

```
str_view(x, "\\..\\..\\..")
```

Match start and end of strings

Sometimes it's useful to match start or end of a string. To do this, we use

- ^ to match start of string
- $ to match end of string

```
x <- c("apple pie", "green apple", "apple")

str_view(x, "^apple")
```

```
str_view(x, "apple$")
```

To only match a complete string, you can use both ^ and $.

```
x <- c("apple pie", "green apple", "apple")

str_view(x, "^apple$")
```

Exercise

words contains common words.

```
words
```

Find all words that:

1. Start with "y"
2. End with "x"
3. Are exactly three letters long
4. Have seven letters or more

Use match=TRUE argument in str_view( ) so it only displays matching words.

```
str_view(words, "^y", match=TRUE)
```

```
str_view(words, "x$", match=TRUE)
```

```
str_view(words, "^...$", match=TRUE)
```

```r
str_view(words, ".......", match=TRUE)
```

Character classes

There are a few useful tools to match patterns that are more than one character. `.` is used to match any character. Here are the others:

- \d: matches digit
- \s: matches whitespace (space, tab, newline)
- [abc]: matches a, b, or c
- [^abc]: matches anything except a, b, or c

```r
str_view(c("2 apples", "5 bananas", "4 carrots"), "\\d")
```

You can give alternatives by using |. For example, (abc|xyz) matches abc or xyz. This is useful when there are multiple ways to spell a word.

```r
str_view(c("grey", "gray"), "gr(e|a)y")
```

```r
str_view(c("color", "colour"), "col(ou|o)r")
```

Exercises

Use the same words list and create regular expressions to find all words that:

1. Start with a vowel
2. Only contain consonants. (Hint: thinking about matching "not"-vowels.)
3. End with ing or ise.
4. Is "q" always followed by a "u"?

```r
str_view(words, "^[aeiou]", match = TRUE)
```

```r
str_view(words, "[aeiou]", match = FALSE)
```

```r
str_view(words, "(ing|ise)$", match = TRUE)
```

```r
str_view(words, "q.", match = TRUE)
```

Repetition

We can control repetition of patterns using these characters:

- ?: 0 or 1
- +: 1 or more
- *: 0 or more

For example,

```r
str_view(c("color", "colour"), "colo(u)?r")
```

```r
str_view(c("apple", "banana"), "(na)+")
```

```r
str_view(c("10 apples", "15 bananas", "40 carrots"), "\\d+")
```

You can also specify the number of matches precisely:

- {n}: exactly n
- {n,}: n or more
- {n,m}: between n and m

```r
str_view(c("apple", "banana"), "(na){1}")
```

```r
str_view(c("apple", "banana"), "(na){1,}")
```

```r
str_view(c("apple", "banana"), "p{1,2}")
```

By default, these expressions will match the longest string possible. If you want to match the shortest string possible, you can add ? at the end.

```r
str_view(c("apple", "banana"), "p{1,2}?")
```

```r
str_view(c("apple", "banana"), "(na)+?")
```

Exercises

Create regular expressions to find all words that:

1. Start with three consonants.
2. Have three or more vowels in a row.
3. Have two or more vowel-consonant pairs in a row.

```r
str_view(words, "^[^aeiou]{3}", match = TRUE)
```

```r
str_view(words, "[aeiou]{3,}", match = TRUE)
```

```r
str_view(words, "([aeiou][^aeiou]){2,}", match = TRUE)
```

Other useful functions

Count number of matches

Use **str_count( )** to count the number of matches in a string

```r
str_count("mississippi", "s")
```

```
## [1] 4
```

To count the mean number of vowels per word,

```r
mean(str_count(words, "[aeiou]"))
```

```
## [1] 1.991837
```

Replace matches

str_replace() and **str_replace_all()** to replace matches with other strings.

```r
x <- c("four apples", "two bananas", "five carrots")

str_replace(x, "a", "A")
```

```
## [1] "four Apples"  "two bAnanas"  "five cArrots"
```

```r
str_replace_all(x, "a", "A")
```

```
## [1] "four Apples"  "two bAnAnAs"  "five cArrots"
```

```r
str_replace_all(x, c("four" = "4", "two" = "2", "five" = "5"))
```

```
## [1] "4 apples"  "2 bananas" "5 carrots"
```

Split strings

Use **str_split()** to split a string up into pieces. Let's split string1 from earlier into words.

```r
str_split(string1, " ")
```

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

Exercise: Split up a string "apples, carrots, and bananas" into individual components

---

```r
str_split("apples, carrots, and bananas", "(, and |, )")
```

```
## [[1]]
## [1] "apples"  "carrots" "bananas"
```