# Stat405

## Regular expressions

Hadley Wickham

Project 2 now due
date Oct 30

1. Recap
2. Regular expressions
3. Other string processing functions

# Recap

```
library(stringr)
contents <- readRDS("email.rds")

breaks <- str_locate(contents, "\n\n")
headers <- str_sub(contents, end = breaks[, 1] - 1)
bodies <- str_sub(contents, start = breaks[, 2] + 1)
```

```
parse_headers <- function(x) {
  lines <- str_split(x, "\n")[[1]]

  continued <- str_sub(lines, 1, 1) %in% c(" ", "\t")

  # This is a useful trick!
  groups <- cumsum(!continued)

  fields <- rep(NA, max(groups))
  for (i in seq_along(fields)) {
    fields[i] <- str_c(lines[groups == i],
      collapse = "\n")
  }
  fields
}
```

```
# Now we want to apply that function to every
# element of headers

# What should our output data structure look like?

# It can't be a character vector.  Why not?
```

```r
n <- length(headers)

# Instead, we need to use a list.
# A list can contain any other data structure
# (including other lists).
output <- vector("list", n)

for (i in seq_len(n)) {
  output[[i]] <- parse_headers(headers[i])
}

str(output)
output[1]
output[[1]]
```

If list `x` is a train carrying objects, then `x[[5]]` is the object in car 5; `x[4:6]` is a train of cars 4-6.

http://twitter.com/#!/RLangTip/status/118339256388304896

```
str(strsplit(headers[1], "\n"))
str(strsplit(headers[1], "\n")[1])
str(strsplit(headers[1], "\n")[[1]])

str(strsplit(headers, "\n"))
```

# Your turn

Write a small function that given a single header field splits it into name and contents. Do you want to use `str_split()`, or `str_locate()` & `str_sub()`?

Remember to get the algorithm working before you write the function

```
test1 <- "Sender: <Lighthouse@independent.org>"

test2 <- "Subject: Alice: Where is my coffee?"
```

```
f1 <- function(input) {
  str_split(input, ": ")[[1]]
}

f2 <- function(input) {
  colon <- str_locate(input, ": ")
  c(
    str_sub(input, end = colon[, 1] - 1),
    str_sub(input, start = colon[, 2] + 1)
  )
}

f3 <- function(input) {
  str_split_fixed(input, ": ", 2)[1, ]
}
```

# Next steps

We split the content into header and body. And split up the header into fields. Both of these tasks used fixed strings.

What if the pattern we need to match is more complicated?

# Matching challenges

- How could we match a phone number?
- How could we match a date?
- How could we match a time?
- How could we match an amount of money?
- How could we match an email address?

# Regular expressions

# Pattern matching

Each of these types of data have a fairly regular pattern that we can easily pick out by eye

Today we are going to learn about regular expressions, which are an extremely concise language for describing patterns.

# First challenge

- Matching phone numbers
- How are phone numbers normally written?
- How can we describe this format?
- How can we extract the phone numbers?

Nothing.  It is a generic folder that has Enron Global Markets on the cover.  It is the one that I sent you to match your insert to when you were designing.  With the dots.  I am on my way over for a meeting, I'll bring one.

Juli Salvagio
Manager, Marketing Communications
Public Relations
Enron-3641
1400 Smith Street
Houston, TX 77002
713-345-2908-Phone
713-542-0103-Mobile
713-646-5800-Fax

Mark,
Good speaking with you. I'll follow up when I get your email.
Thanks,
Rosanna

Rosanna Migliaccio
Vice President
Robert Walters Associates
(212) 704-9900
Fax: (212) 704-4312
mailto:rosanna@robertwalters.com
http://www.robertwalters.com

# Your turn

Write down a description, in words, of the usual format of phone numbers.

# Phone numbers

- 10 digits, normally grouped 3-3-4
- Separated by space, - or ()

- How can we express that with a computer program? We'll use regular expressions
  - `[0-9]`: matches any number between 0 and 9
  - `[- ()]`: matches -, space, ( or )

```
phone <- "[ (][0-9][0-9][0-9][- )][0-9][0-9][0-9][- ]
[0-9][0-9][0-9][0-9]"

phone2 <- "[0-9]{3}[- .][0-9]{3}[- .][0-9]{4}"
phone3 <- "[(][0-9]{3}[)][- .][0-9]{3}[- ()][0-9]{4}"

test <- body[10]
cat(test)
```

```
str_detect(test, phone2)

str_locate(test, phone2)
str_locate_all(test, phone2)

str_extract(test, phone2)
str_extract_all(test, phone2)
```

| Qualifier | ≥ | ≤ |
|---|---|---|
| `?` | `0` | `1` |
| `+` | `1` | `Inf` |
| `*` | `0` | `Inf` |
| `{m,n}` | `m` | `n` |
| `{,n}` | `0` | `n` |
| `{m,}` | `m` | `Inf` |

```
# What do these regular expression match?

mystery1 <- "[0-9]{5}(-[0-9]{4})?"
mystery2 <- "[0-9]{3}-[0-9]{2}-[0-9]{4}"
mystery3 <- "[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,4}"
mystery4 <- "https?://[a-z]+([a-z0-9-]*[a-z0-9]+)?(\\.([a-z]+
([a-z0-9-]*[a-z0-9]+)?)+)*"

# Think about them first, then input to http://strfriend.com/
# or http://xenon.stanford.edu/~xusch/regexp/analyzer.html
# (select java for language - it's closest to R for regexps)
```

# New features

- `()` group parts of a regular expression
- `.` matches any character
  (`\.` specifically matches `.`)
- `\d` matches a digit, `\s` matches a space
- Other characters that need to be escaped: `$`, `^`

Ceci n'est pas une pomme

# Escape

Tricky, because we are writing strings, but the regular expression is the contents of the string.  For example:

"`a\\.b`" represents the string `a\.b`, which only matches `a.b`

"`a\.b`" is an error

"`a.b`" matches a, then any letter then b

| String | Regexp | Matches |
|---|---|---|
| "." | . | Any character |
| "\\." | \. | . |
| "\\d" | \d | Any digit |
| "\\s" | \s | Any white space |
| "\"" | " | " |
| "\\(" | \( | ( |
| "\\\\" | \\ | \ |
| "\\b" | \b | Word border |

| String | Regexp | Matches |
|---|---|---|
| "." | . | Any character |
| "[.]" | [.] | . |
| "(" | ( | Error - no matching ) |
| "[(]" | [(] | ( |

# Your turn

Improve our initial regular expression for matching a phone number.

Hints: use the tools linked from the website to help. You can use `str_extract_all(body, pattern)` to extract matches from the emails

```
phone3 <- "[(]?[0-9]{3}[- )]+[0-9]{3}[- ]+[0-9]{4}"
str_extract_all(body, phone3)
```

# Your turn

Create a regular expression to match a date. Test it against the following cases:

```
c("10/14/1979", "20/20/1945",
"1/1/1905", "5/5/5")
```

Create a regular expression to match a time. Test it against the following cases:

```
c("12:30 pm", "2:15 AM", "312:23 pm",
"1:00 american", "08:20 am")
```

```
dates <- c("10/14/1979", "20/20/1945", "1/1/1905", "5/5/5")
str_detect(dates, "[1]?[0-9]/[1-3]?[0-9]/[0-9]{2,4}")

times <- c("12:30 pm", "2:15 AM", "312:23 pm",
  "1:00 american", "08:20 am")
str_detect(times, "[01]?[0-9]:[0-5][0-9] ([Aa]|[Pp])[Mm]")
str_detect(times, "\\b1?[0-9]:[0-5][0-9] ([Aa]|[Pp])[Mm]\\b")
```

# Your turn

Create a regular expression that matches dates like "12 April 2008". Make sure your expression works when the date is in a sentence: "On 12 April 2008, I went to town."

Then extract just the month (hint: use `str_split`)

```
test <- c("15 April 2005", "28 February 1982",
          "113 July 1984")

dates <- str_extract(test, "\\b[0-9]{2} [a-zA-Z]+ [0-9]{4}")
```

```
date <- dates[[1]]
str_split(date, " ")[[1]][2]
```

| String | Regexp | Matches |
|---|---|---|
| "[abc]" | [abc] | a, b, or c |
| "[a-c]" | [a-c] | a, b, or c |
| "[ac-"] | [ac-] | a, c, or - |
| "[ae-g.] | [ae-g.] | a, e, f, g, or . |
| "[^abc]" | [^abc] | Not a, b, or c |
| "[^a-c]" | [^a-c] | Not a, b, or c |
| "[ac^]" | [ac^] | a, c, or ^ |

| String | Regexp | Matches |
|---|---|---|
| "^a" | ^a | a at start of string |
| "a$" | a$ | a at end of string |
| "^a$" | ^a$ | complete string = a |
| "\\$a" | \$a | $a |

# Your turn

Write a regular expression to match dollar amounts.

```
money1 <- "\\$[0-9, ]+[0-9]"
str_extract_all(body, money1)

money2 <- "\\$[0-9, ]+[0-9](\\.[0-9]+)?"
str_extract_all(body, money2)
```

# Summary

| Function | Parameters | Result |
|---|---|---|
| `str_detect` | string, pattern | logical vector |
| `str_locate` | string, pattern | numeric matrix |
| `str_extract` | string, pattern | character vector |
| `str_replace` | string, pattern, **replacement** | character vector |
| `str_split_fixed` | string, pattern | character matrix |

| Single<br>(output usually vector or matrix) | Multiple<br>(output usually a list) |
|---|---|
| `str_detect` | |
| `str_locate` | `str_locate_all` |
| `str_extract` | `str_extract_all` |
| `str_replace` | `str_replace_all` |
| `str_split_fixed` | `str_split` |

**More info at:**

http://vita.had.co.nz/papers/stringr.html