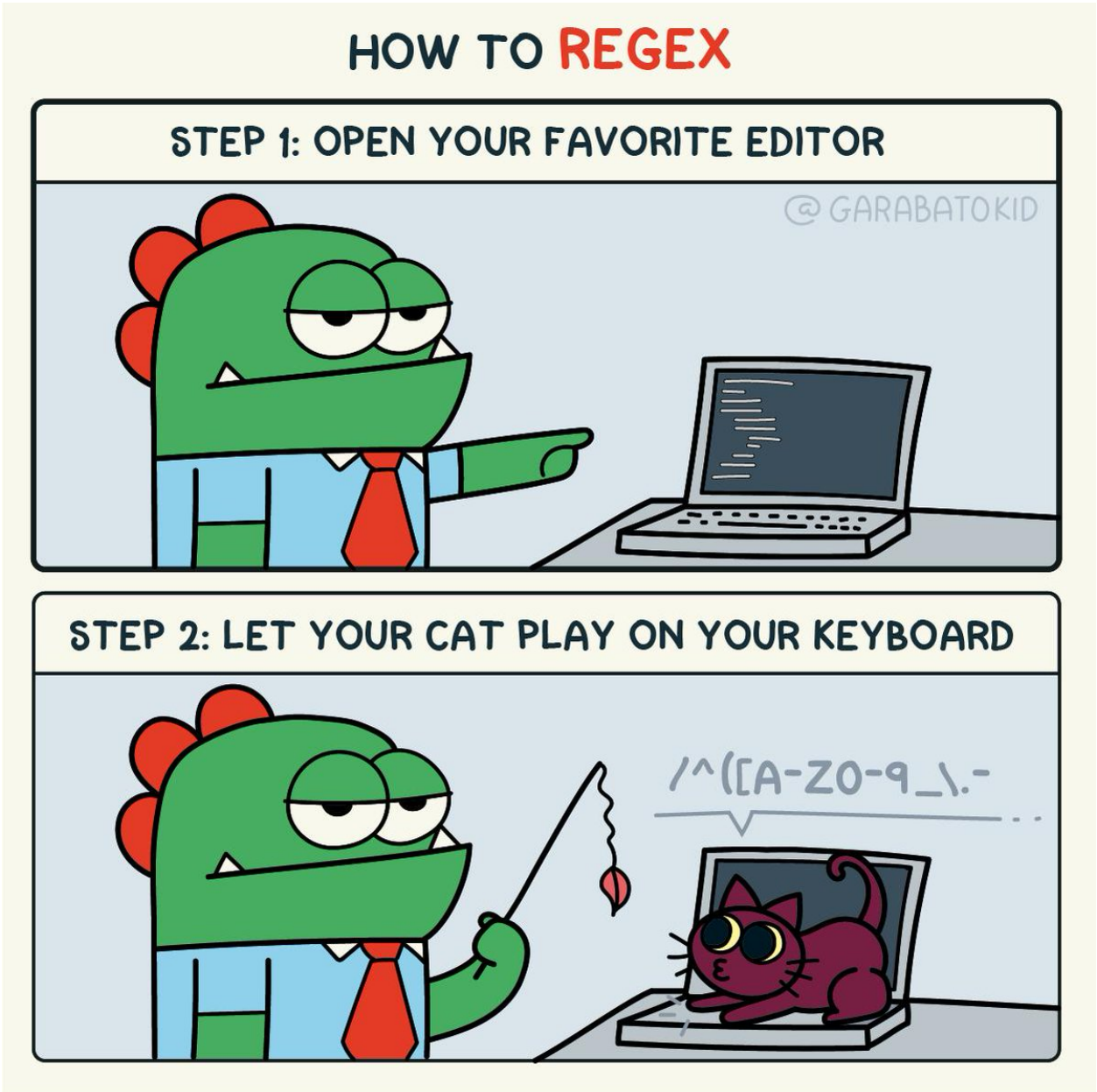
Workshop on regex in R - Part I

Edith Scheifele (B7, Z2)

20th December 2019

regex?



hat tip to Martin Schäfer

Intro

- regular expressions (regex) are a very powerful pattern matching tool for strings
- several implementations with slightly different rules

Use Cases

- select all txt files, and only those, from a directory

Use Cases

- select all txt files, and only those, from a directory
- detect if there are any leading spaces in a string

Use Cases

- select all txt files, and only those, from a directory
- detect if there are any leading spaces in a string
- extract a subject id from a file name

Use Cases

- select all txt files, and only those, from a directory
- detect if there are any leading spaces in a string
- extract a subject id from a file name
- replace some strange looking characters (e.g., German umlauts)

Plan for today

1. 4 functions from *stringr* + exercises
2. basic regular expressions

Requirements

- a fresh R script in RStudio for doing the exercises
- packages (installed and loaded):
  - a relatively new version of *tidyverse* since it includes *stringr* by default
  - or *stringr* alone

stringr

- package for string manipulation

stringr

- package for string manipulation
- all functions begin with `str_` and take a character vector as their first argument
  - each piece in a character vector is a string, so everything that's enclosed with either double or single quotation marks is a string

stringr

- package for string manipulation
- all functions begin with `str_` and take a character vector as their first argument
  - each piece in a character vector is a string, so everything that's enclosed with either double or single quotation marks is a string
- all functions take a `pattern` argument where you can input your regex as a string

stringr

- package for string manipulation
- all functions begin with `str_` and take a character vector as their first argument
  - each piece in a character vector is a string, so everything that's enclosed with either double or single quotation marks is a string
- all functions take a `pattern` argument where you can input your regex as a string
- some functions require additional arguments (see `str_replace()`)

stringr

- package for string manipulation
- all functions begin with `str_` and take a character vector as their first argument
  - each piece in a character vector is a string, so everything that's enclosed with either double or single quotation marks is a string
- all functions take a `pattern` argument where you can input your regex as a string
- some functions require additional arguments (see `str_replace()`)
- there are 7 basic functions, but we will look at only 4 today:
  - `str_detect(string, pattern)`
  - `str_subset(string, pattern)`
  - `str_extract(string, pattern)`
  - `str_replace(string, pattern, replacement)`
  - plus their greedy sisters extended by `_all`, e.g. `str_replace_all()`

stringr: `str_detect()`

- `str_detect()` outputs a logical vector with `TRUE` if it detects the pattern and `FALSE` if not

stringr: `str_detect()`

- `str_detect()` outputs a logical vector with `TRUE` if it detects the pattern and `FALSE` if not

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

```
str_detect(fruits, "a")
```

```
[1] FALSE FALSE  TRUE  TRUE
```

stringr: `str_detect()`

- `str_detect()` outputs a logical vector with `TRUE` if it detects the pattern and `FALSE` if not

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

```
str_detect(fruits, "a")
```

```
[1] FALSE FALSE  TRUE  TRUE
```

- what does the output tell you?

stringr: `str_detect()`

- `str_detect()` outputs a logical vector with `TRUE` if it detects the pattern and `FALSE` if not

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

```
str_detect(fruits, "a")
```

```
[1] FALSE FALSE  TRUE  TRUE
```

- what does the output tell you?
  - as with all `stringr` functions, case matters: a is not A.
  - it does not matter how many matches there are: there is 1 in *Weintraube* and 2 in *Banane*

stringr: `str_detect()` – exercises

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

- check `fruits` for more sophisticated patterns, like *an* or *au*
- which cases can't you check (yet)?

stringr: `str_detect()` – exercises

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

- check `fruits` for more sophisticated patterns, like *an* or *au*
- which cases can't you check (yet)?
  - detect if a string ends with e.g. *e*, no matter if it (also) contains one in the beginning or middle
  - detect if a string contains an *a* followed by some other character and then followed by an *e*

stringr: `str_subset()`

- `str_subset()` does a similar thing like `str_detect()`, i.e. it detects a pattern in a string, but it outputs the matched string

stringr: `str_subset()`

- `str_subset()` does a similar thing like `str_detect()`, i.e. it detects a pattern in a string, but it outputs the matched string

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

```
str_subset(fruits, "a")
```

```
[1] "Weintraube" "Banane"
```

- it is irrelevant how many matches there are in a string

stringr: `str_subset()` – exercises

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

- Which strings in `fruits` include a *b*?

stringr: `str_subset()` – exercises

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

- Which strings in `fruits` include a *b*?

```
str_subset(fruits, "b")
```

```
[1] "Weintraube"
```

stringr: `str_extract()` & `str_extract_all()`

- `str_extract()` extracts the first match and outputs it, when there is no match it outputs `NA`

stringr: `str_extract()` & `str_extract_all()`

- `str_extract()` extracts the first match and outputs it, when there is no match it outputs `NA`

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

```
str_extract(fruits, "a")
```

```
[1] NA  NA  "a" "a"
```

stringr: `str_extract()` & `str_extract_all()`

- `str_extract()` extracts the first match and outputs it, when there is no match it outputs `NA`

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

```
str_extract(fruits, "a")
```

```
[1] NA  NA  "a" "a"
```

- How many *a* did we get for *Banana*? What's the problem?

stringr: `str_extract()` & `str_extract_all()`

- `str_extract_all()` extracts all matches and outputs them in a list, when there is no match it outputs `character(0)` (i.e. an empty character vector)

```
str_extract_all(fruits, "a")
```

```
[[1]]
character(0)

[[2]]
character(0)

[[3]]
[1] "a"

[[4]]
[1] "a" "a"
```

stringr: `str_extract()` & `str_extract_all()` – exercises

```
fruits <- c("Birne", "Apfel", "Weintraube", "Banane")
```

- Extract all *n* from `fruits`
- How many are there all in all? Hint: `unlist()` and `length()`

stringr: `str_replace()` & `str_replace_all()`

- `str_replace(string, pattern, replacement)` replaces the first match of `pattern` with `replacement`

stringr: `str_replace()` & `str_replace_all()`

- `str_replace(string, pattern, replacement)` replaces the first match of `pattern` with `replacement`

```
fruits_with_upper_E <- c("BirnE", "ApfEl",
                         "WEintraubE", "BananE")
```

```
str_replace(fruits_with_upper_E, "E", "e")
```

```
[1] "Birne"      "Apfel"      "WeintraubE" "Banane"
```

stringr: `str_replace()` & `str_replace_all()`

- `str_replace(string, pattern, replacement)` replaces the first match of `pattern` with `replacement`

```
fruits_with_upper_E <- c("BirnE", "ApfEl",
                         "WEintraubE", "BananE")
```

```
str_replace(fruits_with_upper_E, "E", "e")
```

```
[1] "Birne"      "Apfel"      "WeintraubE" "Banane"
```

- Do you see any problem?

stringr: `str_replace()` & `str_replace_all()` – exercises

- `str_replace_all()` replaces all matches of `pattern` with `replacement`

stringr: `str_replace()` & `str_replace_all()` – exercises

- `str_replace_all()` replaces all matches of `pattern` with `replacement`

```
fruits_with_upper_E <- c("BirnE", "ApfEl",
                         "WEintraubE", "BananE")
```

- replace all *E* with *e* from `fruits_with_upper_E`

stringr: `str_replace()` & `str_replace_all()` – exercises

- `str_replace_all()` replaces all matches of `pattern` with `replacement`

```
fruits_with_upper_E <- c("BirnE", "ApfEl",
                         "WEintraubE", "BananE")
```

- replace all *E* with *e* from `fruits_with_upper_E`

```
str_replace_all(fruits_with_upper_E, "E", "e")
```

```
[1] "Birne"      "Apfel"      "Weintraube" "Banane"
```

regex

- you actually already wrote some regex: the letter you specified in `pattern`, e.g. *a* was a regex
- `pattern` needs the regex as a string, so we put the *a* in quotes

regex: higher-level representations

- to use the full power of regex you need some higher-level representations

regex: higher-level representations

- to use the full power of regex you need some higher-level representations
- some higher-level representations:
  - any character, except `\n` (newline): `.`
  - word character (lower and upper case letters, digits, and the underscore): `\w`
  - digit: `\d`
  - range:
    - lower case letters a through c: [a-c]
    - all vowels: [aeiou]

regex: quantifiers

- if you want to match a regex variably often you need quantifiers
- they go behind the regex you want to quantifiy

regex: quantifiers

- if you want to match a regex variably often you need quantifiers
- they go behind the regex you want to quantifiy

- **quantifiers**:
  - ?: zero or one (e.g., `"a?"`)
  - *: zero or more (e.g., `"a*"`)
  - +: one or more (e.g., `"a+"`)
  - {n}: exactly n times (e.g., `"a{2}"`)
  - {n,}: n or more times (e.g., `"a{1,}"`)
  - {,n}: at most n times (e.g., `"a{,2}"`)
  - {n,m}: between n and m times (e.g., `"a{1,2}"`)

regex: escaping with \

- some characters have a special meaning in R: \, ", or .

regex: escaping with \

- some characters have a special meaning in R: \, ", or .
- in order to use them as a literal backslash, quote, or dot, you need to escape them with a backslash \

regex: escaping with \

- some characters have a special meaning in R: `\`, `"`, or `.`
- in order to use them as a literal backslash, quote, or dot, you need to escape them with a backslash `\`
- to extract a digit with the regex `\d`, you actually need to write the escaped string version `"\\d"` into `pattern`

regex: escaping with \

- some characters have a special meaning in R: \, ", or .
- in order to use them as a literal backslash, quote, or dot, you need to escape them with a backslash \
- to extract a digit with the regex \d, you actually need to write the escaped string version "\\d" into pattern

```
subject_id <- c("subject1xxx",
                "subject2xyy",
                "subject3yyx")
```

regex: escaping with \

- some characters have a special meaning in R: \, ", or .
- in order to use them as a literal backslash, quote, or dot, you need to escape them with a backslash \
- to extract a digit with the regex \d, you actually need to write the escaped string version "\\d" into pattern

```
subject_id <- c("subject1xxx",
                "subject2xyy",
                "subject3yyx")
```

```
str_extract(subject_id, pattern = "\\d")
```

```
[1] "1" "2" "3"
```

regex: escaping with \

- some characters have a special meaning in R: \, ", or .
- in order to use them as a literal backslash, quote, or dot, you need to escape them with a backslash \
- to extract a digit with the regex \d, you actually need to write the escaped string version "\\d" into pattern

```
subject_id <- c("subject1xxx",
                "subject2xyy",
                "subject3yyx")
```

```
str_extract(subject_id, pattern = "\\d")
```

```
[1] "1" "2" "3"
```

- sometimes you can avoid escaping by putting your special character inside the range backets [ ] where they are treated with their literal meaning (see exercise)

regex – exercises

```
test <- c("aba", "a.a", "xabax", "2a2a", "ya?ay")
```

- get all strings from `test` that follow the pattern: *a* followed by any character followed by *a*

regex – exercises

```
test <- c("aba", "a.a", "xabax", "2a2a", "ya?ay")
```

- get all strings from `test` that follow the pattern: *a* followed by any character followed by *a*

```
str_subset(test, "a.a")
```

```
[1] "aba"   "a.a"   "xabax" "2a2a"  "ya?ay"
```

regex – exercises

```
test <- c("aba", "a.a", "xabax", "2a2a", "ya?ay")
```

- get all strings from `test` that follow the pattern: *a* followed by any character followed by *a*

```
str_subset(test, "a.a")
```

```
[1] "aba"   "a.a"   "xabax" "2a2a"  "ya?ay"
```

- extract only the match, not the whole string

regex – exercises

```
test <- c("aba", "a.a", "xabax", "2a2a", "ya?ay")
```

- get all strings from `test` that follow the pattern: *a* followed by any character followed by *a*

```
str_subset(test, "a.a")
```

```
[1] "aba"   "a.a"   "xabax" "2a2a"  "ya?ay"
```

- extract only the match, not the whole string

```
str_extract(test, "a.a")
```

```
[1] "aba" "a.a" "aba" "a2a" "a?a"
```

regex – exercises

```
test <- c("aba", "a.a", "xabax", "2a2a", "ya?ay")
```

- get all strings from `test` that follow the pattern: *a* followed by any character followed by *a*

```
str_subset(test, "a.a")
```

```
[1] "aba"   "a.a"   "xabax" "2a2a"  "ya?ay"
```

- extract only the match, not the whole string

```
str_extract(test, "a.a")
```

```
[1] "aba" "a.a" "aba" "a2a" "a?a"
```

- design a pattern that only matches *a.a* and get the string

regex – exercises

```
test <- c("aba", "a.a", "xabax", "2a2a", "ya?ay")
```

- get all strings from `test` that follow the pattern: *a* followed by any character followed by *a*

```
str_subset(test, "a.a")
```

```
[1] "aba"   "a.a"   "xabax" "2a2a"  "ya?ay"
```

- extract only the match, not the whole string

```
str_extract(test, "a.a")
```

```
[1] "aba" "a.a" "aba" "a2a" "a?a"
```

- design a pattern that only matches *a.a* and get the string

```
str_subset(test, "a\\.a")
```

```
[1] "a.a"
```

regex – exercises

```
filenames <- c("eprime_subject_1_2019-12-08",
               "eprime_subject_2_2019-12-18",
               "eprime_subject_11_2019-12-28")
```

- extract the subject id from `filenames`

regex – exercises

```
filenames <- c("eprime_subject_1_2019-12-08",
               "eprime_subject_2_2019-12-18",
               "eprime_subject_11_2019-12-28")
```

- extract the subject id from `filenames`

```
str_extract(filenames, "\\d{1,}")
```

```
[1] "1"  "2"  "11"
```

```
str_extract(filenames, "\\d+")
```

```
[1] "1"  "2"  "11"
```

regex – exercises

```
filenames <- c("eprime_subject_1_2019-12-08",
               "eprime_subject_2_2019-12-18",
               "eprime_subject_11_2019-12-28")
```

- extract the subject id from `filenames`

```
str_extract(filenames, "\\d{1,}")
```

```
[1] "1"  "2"  "11"
```

```
str_extract(filenames, "\\d+")
```

```
[1] "1"  "2"  "11"
```

- extract the date from `filenames`

regex – exercises

```
filenames <- c("eprime_subject_1_2019-12-08",
               "eprime_subject_2_2019-12-18",
               "eprime_subject_11_2019-12-28")
```

- extract the subject id from `filenames`

```
str_extract(filenames, "\\d{1,}")
```

```
[1] "1"  "2"  "11"
```

```
str_extract(filenames, "\\d+")
```

```
[1] "1"  "2"  "11"
```

- extract the date from `filenames`

```
str_extract(filenames, "\\d{4}-\\d{2}-\\d{2}")
```

```
[1] "2019-12-08" "2019-12-18" "2019-12-28"
```

regex – exercises

```
christmas <- c("Merry", "christmas.")
```

- extract all vowels from `christmas`

regex – exercises

```
christmas <- c("Merry", "christmas.")
```

- extract all vowels from `christmas`

```
str_extract_all(christmas, "[aeiou]")
```

```
[[1]]
[1] "e"

[[2]]
[1] "i" "a"
```

regex – exercises

```
christmas <- c("Merry", "christmas.")
```

- extract all vowels from `christmas`

```
str_extract_all(christmas, "[aeiou]")
```

```
[[1]]
[1] "e"

[[2]]
[1] "i" "a"
```

- extract the punctuation mark from `christmas`

regex – exercises

```
christmas <- c("Merry", "christmas.")
```

- extract all vowels from `christmas`

```
str_extract_all(christmas, "[aeiou]")
```

```
[[1]]
[1] "e"

[[2]]
[1] "i" "a"
```

- extract the punctuation mark from `christmas`

```
str_extract(christmas, "\\.")
```

```
[1] NA  "."
```

```
str_extract(christmas, "[.]")
```

regex: anchors

- match from the beginning or end of a string:
  - beginning: ^
  - end: $

regex: anchors

- match from the beginning or end of a string:
  - beginning: ^
  - end: $

```
flowers <- c("Veilchen", "Rose", "Tulpe",
             "Engelsflügel")
```

- match flowers that end with an *e*

regex: anchors

- match from the beginning or end of a string:
  - beginning: ^
  - end: $

```
flowers <- c("Veilchen", "Rose", "Tulpe",
             "Engelsflügel")
```

- match flowers that end with an *e*

```
str_subset(flowers, "e$")
```

```
[1] "Rose"  "Tulpe"
```

regex: anchors

- match from the beginning or end of a string:
  - beginning: ^
  - end: $

```
flowers <- c("Veilchen", "Rose", "Tulpe",
             "Engelsflügel")
```

- match flowers that end with an *e*

```
str_subset(flowers, "e$")
```

```
[1] "Rose"  "Tulpe"
```

- match flowers that begin with an *e*

regex: anchors

- match from the beginning or end of a string:
  - beginning: ^
  - end: $

```
flowers <- c("Veilchen", "Rose", "Tulpe",
             "Engelsflügel")
```

- match flowers that end with an *e*

```
str_subset(flowers, "e$")
```

```
[1] "Rose"  "Tulpe"
```

- match flowers that begin with an *e*

```
str_subset(flowers, "^E")
```

```
[1] "Engelsflügel"
```

regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:

regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:
  - select all txt files, and only those, from a directory by specifying the `pattern` argument in `list.files()` with `"txt"`

regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:
  - select all txt files, and only those, from a directory by specifying the `pattern` argument in `list.files()` with `"txt"`
  - detect if there are any leading spaces in a string by combining ^ and spaces

regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:
  - select all txt files, and only those, from a directory by specifying the `pattern` argument in `list.files()` with `"txt"`
  - detect if there are any leading spaces in a string by combining `^` and spaces
  - extract a subject id from a file name with a version of `\d`

regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:
  - select all txt files, and only those, from a directory by specifying the `pattern` argument in `list.files()` with `"txt"`
  - detect if there are any leading spaces in a string by combining `^` and spaces
  - extract a subject id from a file name with a version of `\d`
  - replace some strange looking characters (e.g., German umlauts)

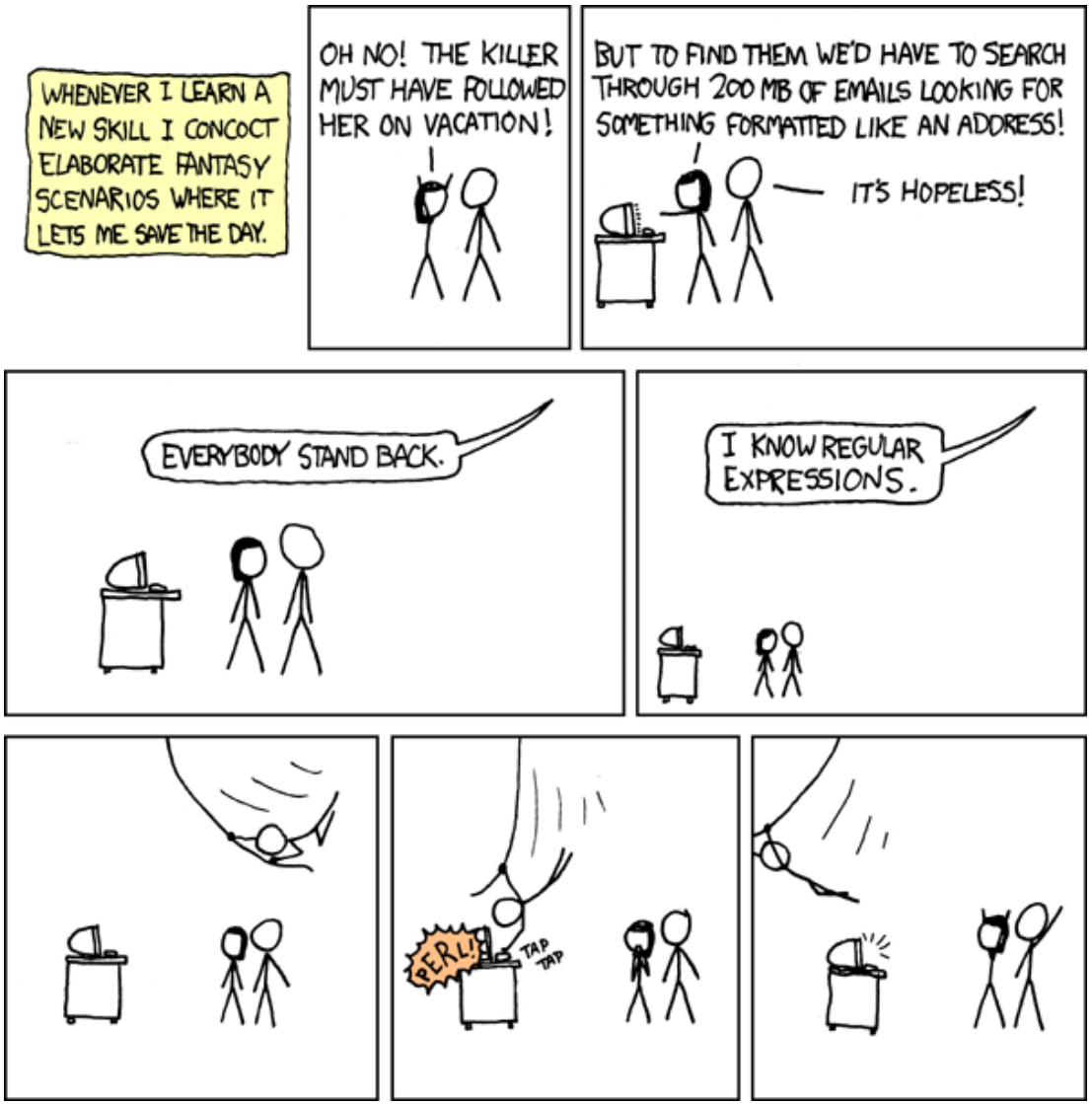
regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:
  - select all txt files, and only those, from a directory by specifying the `pattern` argument in `list.files()` with `"txt"`
  - detect if there are any leading spaces in a string by combining `^` and spaces
  - extract a subject id from a file name with a version of `\d`
  - replace some strange looking characters (e.g., German umlauts)
  - detect if a string ends with e.g. *e*, no matter if it (also) contains one in the beginning or middle, by using `e$`

regex: Recap

- by cleverly combining the concepts from above, we can solve all our use cases and the cases we weren't able to check before:
  - select all txt files, and only those, from a directory by specifying the `pattern` argument in `list.files()` with `"txt"`
  - detect if there are any leading spaces in a string by combining `^` and spaces
  - extract a subject id from a file name with a version of `\d`
  - replace some strange looking characters (e.g., German umlauts)
  - detect if a string ends with e.g. *e*, no matter if it (also) contains one in the beginning or middle, by using `e$`
  - detect if a string contains an *a* followed by some other character and then followed by an *e* by using the `.`

regex!



https://xkcd.com/208/

resources

stringr

https://stringr.tidyverse.org/articles/regular-expressions.html
https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html

regex & stringr

https://r4ds.had.co.nz/strings.html

cheatsheets

https://rstudio.com/wp-content/uploads/2016/09/regexCheatsheet.pdf
https://github.com/rstudio/cheatsheets/blob/master/strings.pdf