RegEx in R

Stefano Allesina *University of Chicago*

Regular expressions in `R`

Sometimes data is hidden in free text. Think of citations in a manuscript, mentions of DNA motifs in tables, etc. You could copy and paste data from these unstructured texts yourself, but if you have much text, the task is very boring and error-prone. What you need is a way to describe a text pattern to a computer, and then have it extract the data automatically. Regular Expressions do exactly that.

Because you want to describe a text pattern using text, a level of abstraction is inevitable. What you want to do is to construct a pattern using **literal** characters and **metacharacters**.

For all our examples, we will use the package `stringr`, which makes the regular expression syntax consistent (there are many *dialects*), and provides a set of easy-to-use functions:

```
library(stringr)
```

All the functions have a common structure. For example, `str_extract` extracts text matching a pattern: `str_extract(text, pattern)`. The simplest possible expression is one in which the pattern is described literally (i.e., we want to find exactly the text we're typing):

```
str_extract("a string of text", "t")
```

```
## [1] "t"
```

```
str_extract_all("a string of text", "t")
```

```
## [[1]]
## [1] "t" "t" "t"
```

Of course, you need to be able to describe much more general patterns. Use the following metacharacters:

- `\d` Match a digit character (`0-9`)
- `\D` Match any character that is not a digit
- `\n` Match a newline
- `\s` Match a space
- `\t` Match a `Tab`
- `\b` Match a "word boundary"
- `\w` Match a "word" character (alphanumeric)
- `.` Match any character

Some examples (note that to escape characters, you want to use two backslashes — you need to escape the backslash itself!):

```
# find the first digit
str_extract("123.25 grams", "\\d")
```

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

```
# find word separator + word character + word separator
str_extract("Albert Einstein was a genius", "\\b\\w\\b")
```

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

```
# find all digits
str_extract_all("my cell is 773 345 6789", "\\d")
```

```
## [[1]]
##  [1] "7" "7" "3" "3" "4" "5" "6" "7" "8" "9"
```

```
# extract all characters
str_extract_all("for example, this and that", ".")
```

```
## [[1]]
##  [1] "f" "o" "r" " " "e" "x" "a" "m" "p" "l" "e" "," " " "t" "h" "i" "s"
## [18] " " "a" "n" "d" " " "t" "h" "a" "t"
```

Of course, you don't want to type `\\w` fifteen times, in case you are looking for a string that is 15 characters long! Rather, you can use quantifiers:

- `*` Match zero or more times. Match as many times as possible.
- `*?` Match zero or more times. Match as few times as possible.
- `+` Match one or more times. Match as many times as possible.
- `+?` Match one or more times. Match as few times as possible.
- `?` Match zero or one times. In case both zero and one time match, prefer one.
- `??` Match zero or one times, prefer zero.
- `{n}` Match exactly `n` times.
- `{n,}` Match at least `n` times. Match as many times as possible.
- `{n,m}` Match between `n` and `m` times.

Exercise

What does this do? Try to guess, and then type the command into `R`

```
str_extract_all("12.06+3.21i", "\\d+\\.?\\d+")
my_str <- "most beautiful and most wonderful have been, and are being, evolved."
str_extract(my_str, "\\b\\w{6,10}\\b")
str_extract(my_str, "\\b\\w+\\b")
str_extract(my_str, "w\\w*")
str_extract(my_str, "b\\w*")
str_extract(my_str, "b\\w+?")
str_extract(my_str, "\\s\\wn\\w+")
```

What if you want to match the characters `?`, `+`, `*`, `.`? You will need to escape them: for example, `\\.` matches the "dot" character.

You can specify anchors to signal that the match has to be in certain special positions in the text:

- `^` Match at the beginning of a line.
- `$` Match at the end of a line.

```
str_extract("Ah, ba ba ba ba Barbara Ann", "\\w{2,}$")
```

```
## [1] "Ann"
```

```
str_extract("Ah, ba ba ba ba Barbara Ann", "^\\w{2,}")
```

```
## [1] "Ah"
```

To match one of several characters, list them between brackets:

```
str_extract("01234567890", "[3120]+")
```

```
## [1] "0123"
```

```
str_extract("01234567890", "[3-5]+")
```

```
## [1] "345"
```

```
str_extract("supercalifragilisticexpialidocious", "[a-i]{3,}")
```

```
## [1] "agi"
```

To match either of two patterns, use alternations:

```
str_extract_all("The quick brown fox jumps over the lazy dog", "fox|dog")
```

```
## [[1]]
## [1] "fox" "dog"
```

If you need more complex alternations, use parentheses to separate the patterns.

Parentheses can also be used to define **groups**, which are used when you want to capture unknown text that is however flanked by known patterns. For example, suppose you want to save the user name of a UofC email:

```
str_match_all("sallesina@uchicago.edu mjsmith@uchicago.edu",
              "\\b([a-zA-Z0-9]*?)@uchicago.edu")
```

```
## [[1]]
##      [,1]                     [,2]
## [1,] "sallesina@uchicago.edu" "sallesina"
## [2,] "mjsmith@uchicago.edu"   "mjsmith"
```

Note that you have to use `str_match` or `str_match_all` to obtain information on the groups.

Useful functions

Many functions have a similar `str_***_all` version, returning all matches.

- `str_detect(strings, pattern)` do the `strings` contain the `pattern`? Returns a logical vector.
- `str_locate(strings, pattern)` find the character position of the pattern
- `str_extract(strings, pattern)` extracts the first match
- `str_match(strings, pattern)` like `extract` but capture groups defined by parentheses
- `str_replace(strings, pattern, newstring)` replaces the first matched `pattern` with `newstring`

Extract primers and polymorphic sites

In the file `data/Ptak_etal_2004.txt` you find a text version of the supplementary materials of Ptak *et al.* (PLoS Biology 2004, doi:10.1371/journal.pbio.0020155).

- Take a look at the file. You see that it first lists the primers used for the study (e.g., `TAP2-17-3' -> CTTGGATATAACACCAAACGCA`), and then the polymorphic sites for 24 chimpanzees (e.g., .
- Read the text as a single string, intervaled by new lines

```
my_txt <- paste(readLines("../data/Ptak_etal_2004.txt"), collapse="\n")
```

- Use regular expressions to produce a data frame containing the primers used in the study:

```
head(primers)
```

```
         ID                Sequence
1 TAP2-1-5' GAGAATCACTTGAACCTGGGAG
2 TAP2-2-5' TTGTCCACAGTGTACCACATGA
3 TAP2-3-5' TATTTCTTCCTGGGGTTTCCTT
4 TAP2-4-5' CATGATGTGTCATGCTGAATTG
5 TAP2-5-5' ATAGAACAAGAACCAAAGCCCA
6 TAP2-6-5' GGACAACAGATAAAGTTGCCCT
```

- Write another regular expression to extract the polymorphic region for each chimp. Use `str_replace_all` to remove extra spaces and newlines. The results should look like:

```
  Chimp                                                Sequence
1   311 ATACCCTGGAGGCAAGAATCTTCCGATAGACGCCAGTCCCTAGTTGT...
2   312 GCACCCTGGAGGCAAGAGTCTTCCGATAGACGCCAGTCCCTAGTTGC...
3   313 GCACCCTGGAGGCAAGAGTCTTCCGATAGACGCCAGTCCCCAGTTGT...
4   314 GCACCCTGGAGGCAAGAGTCTTCCGATAGACCCCAGTCCCCAGTTGC...
5   317 GCACCCTGGAGGTGGGGGCCCCCCAGGAGGCCCCAGCCCCCGGTCGT...
6   320 GCACCCTGGAGATAAGGGCCCCCCAGGAGGCCCCAGCCCCCGGTCGT...
```