Regular Expressions and Perl One-Liners

657.019 Scripting for Biotechnologists (WS 2018/19)

Goals

- Learn the syntax used to extract specific patterns from text, which is called Regular Expressions
- Learn to parse formatted text using regular expressions
- Be prepared to use them with Perl

What is a regular expression?

- A regular expression (regex) is a sequence of characters that define a search pattern
- Programmers use them to recognize specific information from data
- An email address
  - starts with some letters and/or numbers,
  - then there is the @ sign,
  - then some more letters,
  - then `.com`, `.at` or something of the sort,

  → which computers cannot understand
- Need to encode this kind of description as a regex.

Regexes and biology

- Many scripting languages and Linux commands are regex-aware
  - Perl, Python, Bash, awk, sed, grep
- With a regex, we can
  - extract, translate or substitute the recognized text to achieve data analysis goals
- Biological data are text-intensive
  - Sequences, database files, measurements, analysis results, …

Literal text

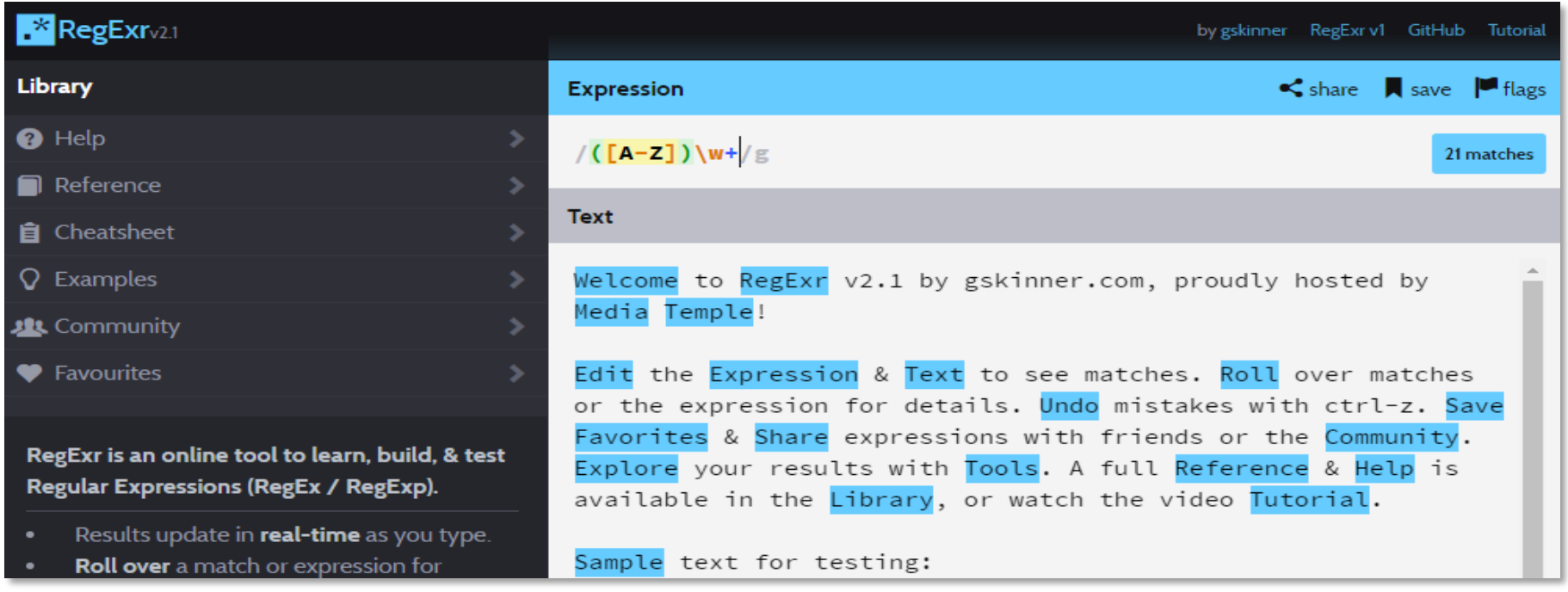
email_addrs.txt

```
Jung Soh: jung.soh@tugraz.at
Barack Obama: prez44@whitehouse.gov
Donald Trump: prez45@whitehouse.gov
Bill Gates: bill@microsoft.com
Canadian Prime Minister: pmo@parl.gc.ca (generic)
foo@bar.com
```

- Plain-old text is a valid regular expression
  - **`jung.soh@tugraz.at`**
  - Matches just one email address
  - One regex per email address required
  - Not very useful → need to be less literal

Online regular expression tool

http://regexr.com



Character class

- A character class indicates a group of possible characters
  - One character class indicates **exactly one** character slot
  - Starts with `\`

- `\d` : digit (0, 1, 2, …, 9)

- `prez\d\d@whitehouse.gov` matches
  - 100 email addresses (`\d\d` matches `00`, `01`, …, `99`)
  - `prez44@whitehouse.gov` and `prez45@whitehouse.gov` in our example file
  - not `prez100@whitehouse.gov` (only two digits match)

More character classes

- `\s`: whitespace (space, tab, line break)
- `\w`: word (letters, digits, and _)

- `\S`: non-whitespace
- `\W`: non-word

- `pre\w\w\w@whitehouse.gov`
  - matches `prez44@whitehouse.gov` and `prez45@whitehouse.gov`
  - but also `pretzl@whitehouse.gov`

- `\D`: What is this character class?

Your own character classes

- List possible characters in a character slot within [ ]
- **prez4[45]@whitehouse.gov**
  - **prez4**, then either **4** or **5**
  - matches only **prez44@whitehouse.gov** and **prez45@whitehouse.gov**
  - Does not match **prez445@whitehouse.gov**
- **[]** indicates **one** character
  - **[0123456789]=[9876543210]=\d**: literal list of digits
  - **[acgtACGT]**: literal list of letters
  - **[0-9]** = **\d**: range of digits
  - **[a-d]** = **[abcd]**: range of letters
  - **[a-dA-D]** = **[abcdABCD]**: list of two ranges of letters

A special character class

- `.` : any single character

- **`prez4[45]@whitehouse.gov`**
  - matches **`prez43@whitehouse!gov`**
- **`jung.soh@tugraz.at`**
  - matches **`jung-soh@tugrazzat`**

- Need to "escape" special interpretation with `\` to match a literal dot
  - **`prez4[45]@whitehouse\.gov`**

Quantifiers: special

- `.....@whitehouse\.gov`
  - matches any White House email addresses with a user name of exactly 6 characters
- Quantifiers for more flexible patterns
  - `*`: 0 or more times
  - `+`: 1 or more times (= at least one)
  - `?`: 0 or 1 time (= optional)

- `.+@whitehouse\.gov`
  - Any character (`.`), one or more times (`+`) → user name before @ must be at least 1-character long

Quantifiers: special

- **?** : makes the match of what comes before **?** optional

- **prez\d\d?@whitehouse\.gov**
  - **prez**, then a digit, then a digit optionally → both **prez1** and **prez01** match
  - matches **prez0** to **prez9** and **prez00** to **prez99**

- **prez\d\d?\d?@whitehouse\.gov**
  - To match **prez100** to **prez999** as well
  - matches **prez0** to **prez9**, **prez00** to **prez99**, and **prez000** to **prez999**

Quantifiers: general

- Most general form: **{min,max}**
  - What comes before can happen **min** to **max** times
  - **min** required, **max** optional

- **.{2,}@whitehouse\.gov**
  - Requires user name to have at least two characters
- **.{2,}@whitehouse\..{2,4}**
  - Restrict top-level domain to have 2 to 4 characters (**@whitehouse.org**, **@whitehouse.net**, …)

- Equivalent expressions:
  - **prez\d\d?\d?\d?\d?@whitehouse\.gov**
  - **prez\d{1,5}@whitehouse\.gov**

Quantifiers: making sense

- Just read one bit at a time, left to right

- `.{3,}@tugraz\.at = ...+@tugraz\.at`
  - Character (`.`), minimum 3 (`{3,`), no maximum (`}`) → At least 3 characters
  - Character (`.`), character (`.`), 1 or more characters (`.+`)

- `.{3,}@tugraz\..{2,4} = .{3,}@tugraz\....?.?`
  - Character (`.`), minimum 2 (`{2,`) maximum 4 (`4}`) → Between 2 and 4 characters
  - Character (`.`), character (`.`), character or nothing (`.?`), character or nothing (`.?`)

Alternatives

- Vertical bar (|)separates alternatives

- **`prez\d+\@whitehouse\.gov|jung\.soh@tugraz\.at`**
  - matches first 3 lines from this input file:

```
Jung Soh: jung.soh@tugraz.at
Barack Obama: prez44@whitehouse.gov
Donald Trump: prez45@whitehouse.gov
Bill Gates: bill@microsoft.com
Canadian Prime Minister: pmo@parl.gc.ca (generic)
foo@bar.com
```

Groups: indicating part

- Use a group to indicate part of regex with alternatives
  - Surround by parentheses: **()**

- **.{2,}@(whitehouse|tugraz)\..{2,4}**
  - matches email addresses, which has
  - user name of at least two characters (**.{2,}**), then
  - a **@**, then
  - either **whitehouse** or **tugraz** (**(whitehouse|tugraz)**) , then
  - a literal dot (**\.**) , then
  - 2-to-4 characters representing top-level domain (**.{2,4}**)

Groups: applying quantifiers

- Can be used to apply quantifiers on any part of regex
- Generalize previous email regex
  - Other lower-level domains than **whitehouse** or **tugraz** exist, and there can be multiple of them (**pmo@parl.gc.ca**)
- **.{2,}\@(\w+\.)+.{2,4}**
  - **(** → group begins
  - **\w+** → letters and numbers
  - **\.** → a literal dot
  - **)** → group ends
  - matches **tugraz.**, **whitehouse.**, **microsoft.**, **parc.**, **bar.**
  - **+** → Group can be repeated any number of times (matches **gc.**)
  - No literal characters except **@** and **.** → Similar to how humans recognize email addresses

Anchors

- Match a pattern only when it is at a particular location on a line
- `^`: start (of a line)
- `$`: end (of a line)
- `.{2,}\@(\w+\.)+.{2,4}$` (anchor to end of line)

```
Jung Soh: jung.soh@tugraz.at
Barack Obama: prez44@whitehouse.gov
Donald Trump: prez45@whitehouse.gov
Bill Gates: bill@microsoft.com
Canadian Prime Minister: pmo@parl.gc.ca (generic)
foo@bar.com
```

Anchors

- `^.{2,}\@(\w+\.)+.{2,4}` (anchor to start of line)
  - matches all lines, because we allow any two or more characters before `@`

```
Jung Soh: jung.soh@tugraz.at
Barack Obama: prez44@whitehouse.gov
Donald Trump: prez45@whitehouse.gov
Bill Gates: bill@microsoft.com
Canadian Prime Minister: pmo@parl.gc.ca (generic)
foo@bar.com
```

- `^\S{2,}\@(\w+\.)+.{2,4}`
  - If we wanted user name to be first thing on the line
  - Matches only the last line

Email address regex

- `^\S{2,}\@(\w+\.)+\w{2,4}`
  - Top-level domain contains only word characters

- `\S{2,}\@(\w+\.)+\w{2,4}`
  - Email address can be anywhere on the line

```
Jung Soh: jung.soh@tugraz.at
Barack Obama: prez44@whitehouse.gov
Donald Trump: prez45@whitehouse.gov
Bill Gates: bill@microsoft.com
Canadian Prime Minister: pmo@parl.gc.ca (generic)
foo@bar.com
```

Greediness

- Regexes by default are "greedy"
  - Each slot will eat up as many characters as possible
- Open reading frame finder for **ATGGCGTGCTAACTTTGA**
  - A start codon, then one or more triplets of nucleotide characters, then a stop codon
- `(ATG|GTG|TTG)(...)+(TAA|TAG|TGA)`
  - Matches **ATGGCGTGCTAACTTTGA** because `(...)+` is greedy
- `(ATG|GTG|TTG)(...)+?(TAA|TAG|TGA)`
  - Matches **ATGGCGTGCTAA**, because `(...)+` is not greedy
  - `?` after a quantifier makes the match not greedy (lazy: as few characters as possible)

Escaping metacharacters

- Some characters have special meaning
  - add a \ to escape (get literal value)

Expression	What it means	If not escaped with \
\.	A literal dot	Any single character
\+	A literal plus sign	One or more of the preceding character
*	A literal asterisk	Zero or more of the preceding character
\[	A literal left square bracket	Start of a character set
\(	A literal left parenthesis	Start of a group
\{	A literal left curly brace	Start of a quantifier such as {2,4}
\\	A literal slash	Start of a metacharacter

Negation

- ^ at the beginning of a character class negates it
- **[acgtnACGTN]{20,}**
  - DNA string that is at least 20 characters long
- **[^acgtnACGTN]{20,}**
  - Non-DNA string that is at least 20 characters long
  - Better than **[bdefh-mo-su-zBDEF** …

- What was another use of the ^ character?

Lookahead

- Restrict matches based on what comes after match
  - But the look-ahead portion is not included in the match

- `[acgtnACGTN]{20,}(?=\s|$)`
  - At least 20-character long DNA string followed by whitespace (`\s`) or end of line (`$`)
  - Positive lookahead
- `[acgtnACGTN]{20,}(?!\S)`
  - At least 20-character long DNA string **not** followed by a non-whitespace character (`\S`)
  - Negative lookahead

Lookbehind

- `(?<=\t)\w+`
  - Matches a word that follows a tab, without including the tab in the match
  - Positive lookbehind
- `(?<!bar)foo`
  - Matches any occurrence of `foo` that does **not** follow `bar`
  - Negative lookbehind

- Compare with lookaheads
  - `\w+(?=\t)` : matches such words that …
  - `foo(?!bar)` : matches such `foo`s that …

Backreferences

- Use part of regex match to define the regex itself
- `(["'])`.*?`\1`
  - Either start with " and end with ", or start with ' and end with '
  - `\1` : place holder for whatever was matched inside first `()`
  - `"It's a wonderful day."`
  - `'An English pangram is, "The quick brown fox jumps over the lazy dog"'`
  - `She said: "It's a wonderful day." He said, "The quick brown fox jumps over the lazy dog!"`
  - `She said: "It's a wonderful day." He said, "The quick brown fox jumps over the lazy dog!"` ← Without the `?`
- There can be several backreferences: `\1`, `\2`, …

Perl: first one-liner

- Perl script: `print_regex_match_lines.pl`

```
#!/usr/bin/perl
open DATA, $ARGV[0]
  or die "Cannot open $ARGV[0]: $!";
while (<DATA>) {
  # print line if regex matches something in line
  print if /regex/;
}
close DATA;
```

- Equivalent Perl one-liner:
  - `perl -ne 'print if /regex/' data.txt`

perl -ne

- Repeat the Perl command that comes after **-ne** for each line of input file
- No automatic printing
- Result of regex match is in variable **$&**

- **perl -ne 'print if /regex/'**
  - Print whole line (default) if **regex** matches something in line
- **perl -ne 'print "$&\n" if /regex/'**
  - Print only the match result (and a newline) if **regex** matches something in line

Perl regex flags

- **`/regex/i`**
  - Case insensitive match
- **`/regex/s`**
  - Treat input as a single line → `.` matches a newline character
- **`/regex/g`**
  - Match globally → match all occurrences in a line, instead of matching only the first occurrence

perl -pe

- Repeat the Perl command that comes after **-pe** for each line of input file, and print each line after running the Perl command
- Used for mass editing (search & replace) of text files
- Two methods of replacing
  - By substitution
  - By translation

perl -pe 's/regex/replacement/'

- Substitute regex with replacement, then print line

- **perl -pe 's/aaa|ccc|ggg|ttt/n/g' sequence.txt**
  - Replace all **aaa, ccc, ggg, ttt** with **n**
- **perl -pe 's/aaa|ccc|ggg|ttt/n/ig' sequence.txt**
  - Sequence can be any mixture of upper- and lower-case characters (safer)
- **perl -pe 's/gene_id/gid/g' annot.txt**
  - Replace all occurrences of **gene_id** with **gid** in an annotation file
- Without **g** flag, only the first match in a line is replaced

`perl -pe 'tr/character list/replacement list/'`

- Replace each character in **`character list`** with corresponding character in **`replacement list`**
  - Literal lists, no regular expression syntax!
- **`perl -pe 'tr/acgt/tgca/' sequence.txt`**
  - Complement a sequence (**`a`**->**`t`**, **`c`**->**`g`**, **`g`**->**`c`**, **`t`**->**`a`**)

- Flags can be **`c`** for complement, and **`d`** for delete
- **`perl -pe 'tr/acgt/n/c' sequence.txt`**
- **`perl -pe 'tr/bd-fh-su-z/n/' sequence.txt`**
  - Translate all ambiguous (non-ACGT) bases to **`n`**
- **`perl -pe 'tr/bd-fh-su-z//d' sequence.txt`**
  - Delete ambiguous bases

Extracting regex match

- **`$&`** contains whole regex match result
- **`$1`**, **`$2`**, … captures user-defined match groups

- **`perl -ne 'print "$1\t$2\n" if /^>\s*seqid=(\S+)\s+len=(\d+)\s*/' sequence.fasta`**
  - Extract sequence ID and length, separated by tab, given a FASTA file like:

```
>seqid=contig.03876 len=1014
cgttnnnnnnccccgg
>seqid=contig.03889 len=1001
nnnnnnnnnttttttcccgggacacaaa
```

perl -a

- Access certain fields (columns) of table data
  - Auto-splitting (of a line into fields) using whitespace
  - Field values in **$F[i]** starting from **i=0**
    - First field = **$F[0]**, second field=**$F[1]**, …

- **perl -ane 'print "$F[2] $F[3]\n"' table.txt**
  - Print third and fourth fields of each line in the table

Perl one-liner summary

- **`perl -ne 'print if /regex/'`**
  - Print line if line contains a regex match
- **`perl -ane 'print "$F[1]\n" if /regex/'`**
  - Print second field if line contains a regex match
- **`perl -pe 's/regex/replacement/flag'`**
  - Replace regex with replacement, then print
- **`perl -pe 'tr/character list/replacement list/flag'`**
  - Translate from character list to replacement list, then print

- Use whole regex (**`$&`**) or match groups (**`$1`**, **`$2`**, …) to extract specific match results
- Use backreferences (**`\1`**, **`\2`**, …) to use match results inside regex itself