Searching for Answers in a Text-Based Universe

A Brief Introduction to Regex

Jacob Wilkins

Regular Expressions - Not so Regular

- Regular Expressions (RegEx) are a language which describes language
- Provides concise language for very general string matching
- Useful at many levels for general purpose pattern matching
- Used throughout applications which you already know e.g. Syntax Highlighting

Why should I care

- Useful in science for large range of applications:
  - Filtering data
  - Searching files
  - Parsing input
  - Automating workflow
  - Mass updating or changing files

Where can I find RegEx

Anywhere:

- Python – `import re;tag_re = re.compile('<-- ([a-zA-Z])$')`
- Javascript – `var tag_re = new RegExp('<-- ([a-zA-Z])$')`
- C – `#include <regex.h>; regex_t tag_re;`
  `int err = regcomp(& tag_re, '<-- ([a-zA-Z])$');`
- C++ – `#include <regex>; std::regex tag_re('<-- ([a-zA-Z])$');`
- Perl – `$tag_re = qr'<-- ([a-zA-Z])$';`
- Bash
  - sed – `sed -r "/<-- ([a-zA-Z])$/"`
  - grep – `grep -E "<-- ([a-zA-Z])$"`
  - awk – `tag_re = "<-- ([a-zA-Z])$"`
  - less – `/<-- ([a-zA-Z])$ RET`
- emacs – `(setq tag_re "<--\s-\([a-zA-Z]\)$")`
- vim – `/<-- ([a-zA-Z])$ RET`
- And many more...

Worked example

- Let's try a RegEx for ourselves. Download `warning.txt` from the PCC repo.
- This data file is a fictional test of an RNG trying to generate $N(0,1]$, but struggling.
- All data over 1 are marked `warning`, all over 1.09 are marked `Error`.
- We can use tools like grep to see if we have any warnings or errors, just by looking for the words separately.

  ```
  grep -c "warning"; grep -c "Error";
  ```

- How about if we wanted to see how many warnings OR errors there were?

  ```
  grep -Ec "(warning|Error)"
  ```

- This is called "Alternation" and is a RegEx feature.

More features

- How about if we don't know the word exactly?
  - We don't know if it is capitalised or not!
  - We don't know if it is a plural or not!
  - Why have they used an American spelling?!
  - I know it had a "ment" in there somewhere... and maybe a q, b, d, or p at the start...

Spelling is optional

- How about if we don't know the word exactly?
  - We don't know if it is capitalised or not!
    - We can use alternation here!
    - `(M|m)aybe it's capitalised`
  - We don't know if it is a plural or not!
    - We can use optional characters!
    - `Perhaps it contains (a )?plurals?`
  - Why have they used an American spelling?!
    - We can use a different kind of alternation if we want
    - `Capitali[sz]ed\? Don't you mean capitali[sz]ed\?`
  - I know it had a "ment" in there somewhere... and maybe a q, b, d, or p at the start...
    - We can use a match to anything or blocks of letters
    - `[qbdp].+ment.*` Ah! Parliament or demented... They're synonyms aren't they?

More Features

- How about if we care about where it is, not what it is
  - I only care if it's at the beginning or end of a line!
  - I only care if it's the whole word.

Anchors aweigh!

- How about if we care about where it is, not what it is
  - I only care if it's at the beginning or end of a line!
    - We can use anchors to say this
    - `^This is at the beginning`
    - `This is at the end$`
  - I only care if it's the whole word.
    - We can use barriers to do this
    - `\bThis\b` word is the whole word, I don't care about thistles!

Worked Example 2

- Let's try another RegEx. Download `gibberish.txt` from the PCC repo.
- Try to find all the words ending with "e".
- I have provided the following script in the PCC repo to serve as a basis `find_boo.py`.

```
import re

with open('gibberish.txt', 'r') as example:
    for line in example:
        if (re.match('boo',line)):
            print(line)
```

Substitution

- RegEx can do more than just find things
- We can use RegEx to change things, too
- This is very useful in automation

Once upon a time..

https://sites.psu.edu/siowfa16/files/2016/10/coffee-1byged6.jpg

```
jw952@bagpu55:~/Teaching/PCC/example$ ls -l
total 13
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Al.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Ar.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Be.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_B.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Ca.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_C.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Cl.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Cr.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_F.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_He.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_H.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_K.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Li.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Mg.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Mn.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Na.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Ne.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_N.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_O.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_P.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Sc.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Si.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_S.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_Ti.input
-rw-rw-r-- 1 jw952 jw952 0 Jul 11 12:37 test_V.input
```

Aha!

```
#!/bin/bash

for file in *.input; do
    echo "target_pressure 10 GPa" >> $file;
    ./run_program $file;
done
```

In no time...

https://sites.psu.edu/siowfa16/files/2016/10/coffee-1byged6.jpg

What can we do?

```
#!/bin/bash

for press in {5..100..5}; do
    for file in *.input; do
        sed -i -r 's/(target_pressure).*/\1 '$press \
        ' GPa' $file;
        ./run_program $file;
    done
done
```

Substitute your life for mine

- Python – `import re;re.sub('/water/', 'wine',"Let's drink water")`
- Javascript – `"Let's drink water".replace(/water/g,'wine')`
- C – Does not natively support replacement, though there are modules such as PCRE.
- C++ – `#include <regex>;`
  `std::regex_replace("Let's drink water",std::regex ('water'), 'wine');`
- Perl – `"Let's drink water"=~s/water/wine/;`
- Bash
  - sed – `sed -r "s/water/wine/" file`
  - awk – `gensub(/water/, 'wine', "", "Let's drink water")`
- emacs – `(replace-regexp 'water 'wine)`
- vim – `:s/water/wine/ RET`
- And many more...

Worked Example 3

- Have a go at replacing some strings in `gibberish.txt`
- Try replacing all instances of “boo” with “foo”
- How about all final letters with “s”
- How about all “o”s with “e”
  Note: For this one, if you are not using Python you might need to use the “g” flag, Python has optional `count` arg.
- I have provided the following script in the PCC repo to serve as a basis `sub_boo.py`.

```
import re

with open('gibberish.txt', 'r') as example:
    for line in example:
        print(re.sub('mach','inl',line))
```

More features

- It's not a word, it's a number!
  `/[0-9]+/` Finds only lines containing a number
- We want to repeat the word!
  `s/^\W*(\w+)\s/\1 \1/` Prints the first word of a sentence twice
- We only want part of the word!
  `s/\b(\w{1,4})\w*\b/\1/g` prints only the first 4 letters of every word
- We only know how it should look, not what's in there!
  `/[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?/` Finds any valid float in most programming languages

Extra Exercises

- I have added some extra exercises, if anyone wants to try using some of this more advanced RegEx.
- `Quartz.geom` contains an excerpt from a geometry optimisation in CASTEP.
- Say we wanted this in `.xyz` format, could we do this?
- `https://en.wikipedia.org/wiki/XYZ_file_format`
- `Alice.txt` contains over 600 lines of Markov chain generated Alice in Wonderland.
- It also contains something formatted like a post-code, can you find it?

More places to learn

- `http://www.regular-expressions.info/tutorial.html` – Full guide for RegExes from start to finish
- `http://regexr.com/` – Site where you can build, test and get explanations for RegExes
- `https://alf.nu/RegexGolf` – Game where you try to match words in as short a RegEx as possible
- `https://regexcrossword.com/` – Puzzle where you fill in a RegEx like game of battleships