Grep and Regular Expressions, Part 1

Grep and Regular Expressions, I

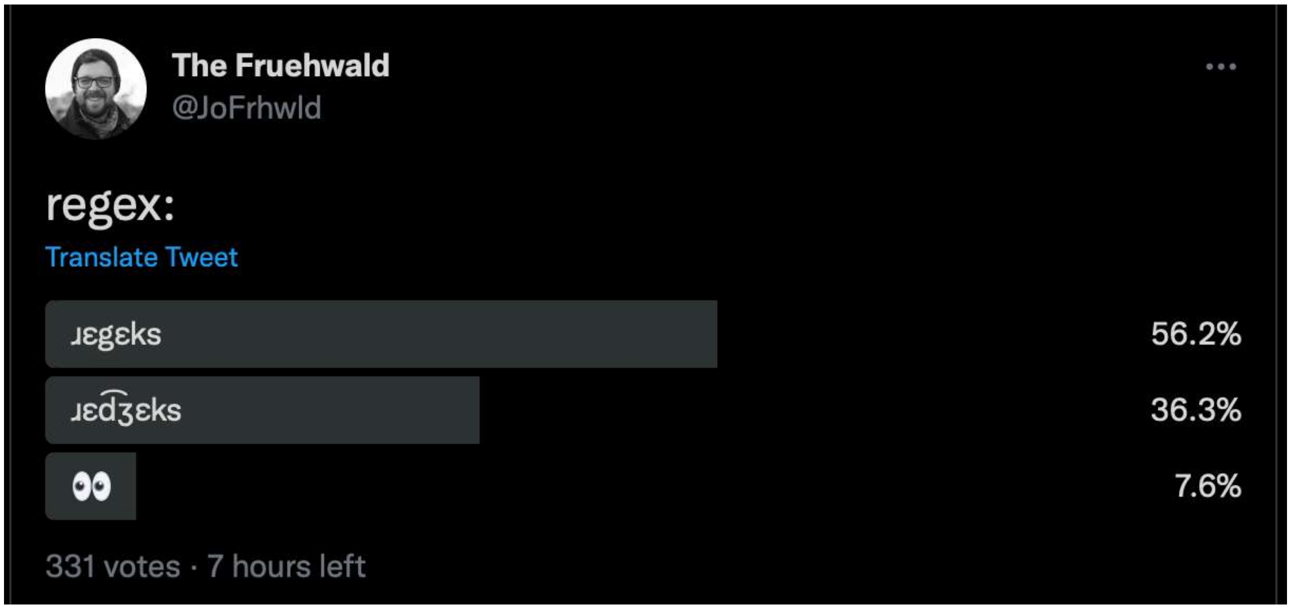
Regular Expressions

While `grep` is super useful for searching through text documents (https://uk.instructure.com/courses/2051722/pages/exploring-text-files) , its power is really unlocked when you start using "Regular Expressions", which is shortened to "RegEx" or "regex".

Regular Expressions are a kind of code for searching for strings in text, and while we'll be learning it initially to use with `grep`, Regular Expressions are used in almost every programming language. Understanding them just a little bit can be super useful.

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

[(https://xkcd.com/208/)](https://xkcd.com/208/)



The Fruehwald
@JoFrhwld

regex:

Translate Tweet

ɹɛgɛks	56.2%
ɹɛd͡ʒɛks	36.3%
👀	7.6%

331 votes · 7 hours left

The literal "mindedness" of computers

The reason we need something like regular expressions is because when we give `grep` a search string, it takes it extremely literally. For this example, I'm going to be using **Grimm's Fairly Tales from Project Gutenberg (https://www.gutenberg.org/ebooks/2591/)** . (Side note, Jacob Grimm was a well known historical linguist!)

You can download and rename this text file with the following code

```
wget https://www.gutenberg.org/files/2591/2591-0.txt
mv 2591-0.txt grimm.txt
```

Let's say I wanted to search the text to see how many of these fairy tales were about bears. I could try doing this by giving `grep` the search string `"bear"` like so:

```
grep --color=auto "bear" grimm.txt
```

Here's a few of the lines it returned, with the bolded text indicating what `grep` matched.

- door for him, and the **bear** was hurrying out, he caught against the bolt
- it. The **bear** ran away quickly, and was soon out of sight behind the
- snow-white **bear**d a yard long. The end of the **bear**d was caught in a
- **bear**d; so now it is tight and I cannot get away, and the silly, sleek,

We're getting a lot of results for the word "bear", which is good, but also a lot of results for "beard", which isn't what we wanted. `grep` is giving us "beard" results because the string of letters `bear` exist inside of `beard`. `grep` doesn't "know" that what we really wanted was to know about was the large brown animal, which is it's own word with a different pronunciation from the word for facial hair.

This little issue encapsulates the larger issue with working with computers. They will always do exactly what we tell them to, but what we *tell* them to do is not always the same thing as what we *want* them to do.

We can do a quick fix for our bear problem by putting a space before and after "bear" to get " bear ":

```
grep --color=auto " bear " grimm.txt
```

All of our results are bears now, because it's only showing us lines where the word "bear" is surrounded by spaces. But, we're also *missing* a few valid bears from our results now! Some bears have a comma or some other punctuation following them that aren't a space.

- ‘Poor **bear**,’ said the mother, ‘lie down by the fire, only take care that
- little’; so they brought the broom and swept the **bear**’s hide clean;
- **bear**: ‘You can lie there by the hearth, and then you will be safe from
- whole summer.’ ‘Where are you going, then, dear **bear**?’ asked Snow-white.
- the **bear**; ‘it is a wretched palace, and you are not King’s children, you

There are also a few instances of "bear" where the "B" is capitalized, and they're all missing too.

- honest people! **Bear**, you will have to pay for that!’
- announced to the **Bear**, and all four-footed animals were summoned to take
- he cried: ‘Dear Mr **Bear**, spare me, I will give you all my treasures;

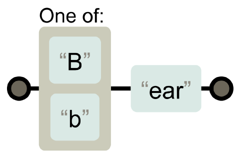
Options and Ranges

Options

The first place to start digging into regex is with options and ranges. Let's say we wanted to get all strings matching "bear", regardless of whether or not the "B" was capitalized. Here's a way to describe that in more precise words:

- Find a string that begins in "B" or "B" and then ends in "ear".

We can also visualize this description with a diagram like this:



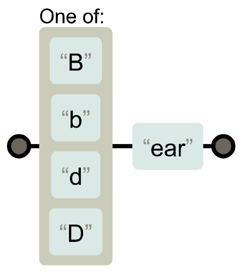
The way we communicate this request to `grep` is with `[Bb]ear`

```
grep --color=auto "[Bb]ear" grimm.txt
```

- honest people! **Bear**, you will have to pay for that!'
- **bear**d from the line, but all in vain, **bear**d and line were entangled fast
- it. The **bear** ran away quickly, and was soon out of sight behind the

By putting the `B` and the `b` inside of the brackets `[]` we told `grep` to find strings that began with either one. This is our first Regular Expression!

You could put as many characters inside the options `[]` as you want, in whatever order you want. If we wanted to search for bears and "dear" at the same time, we can just throw `d` into `[]`.

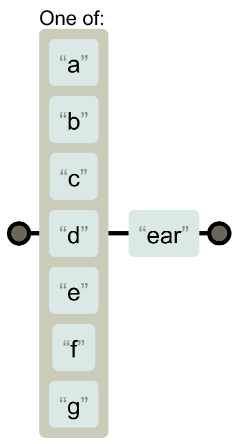


```
grep --color=auto "[BbdD]ear" grimm.txt
```

- whole summer.' 'Where are you going, then, **dear bear**?' asked Snow-white.

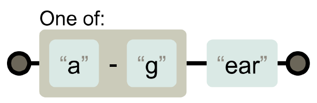
Ranges

What if we wanted to see all words that end in "ear", and begin with any of the first 7 letters of the alphabet. We could type out all 6 letters:



```
grep --color=auto "[abcdefg]ear" grimm.txt
```

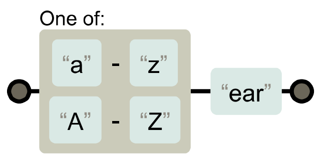
Or, we could enter in a range, and say "the first letter has to be within the range between `a` and `g`." with `[a-g]ear`.



```
grep --color=auto "[a-g]ear" grimm.txt
```

This works for numbers too. For example, if we search for any number between `0` and `9` with `[0-9]` in Grimm's Fairy Tales, we only get back the copyright and licensing agreements.

To search for all strings that start with any letter at all, followed by `ear` we can use `[a-zA-Z]ear`



```
grep --color=auto "[a-zA-Z]ear" grimm.txt
```

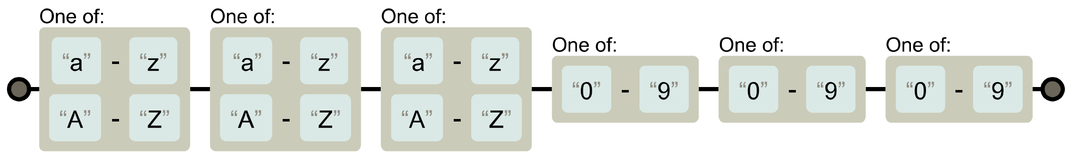
A Possible Application

Everybody's LinkBlue login name seems to follow a common pattern

- First initial
- If there's a middle initial, middle initial

- First two letters of second name
- Three numbers

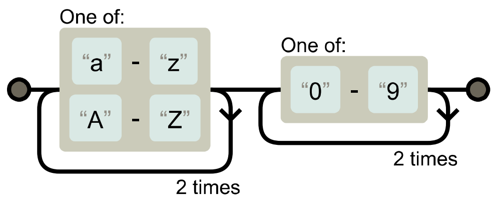
With what we've covered so far, we could write a regex that would match people without middle initial's logins. We need three letters, then three numbers.



That would look like `[a-zA-Z][a-zA-Z][a-zA-Z][0-9][0-9][0-9]`

Simplify with loops

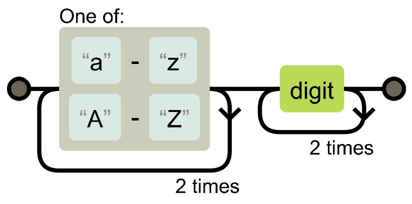
One way we could make this a little easier to read and write is by first telling the regex to repeat some of the expressions. Instead of typing out `[a-zA-Z][a-zA-Z][a-zA-Z]` for three repeats of all the letters, we could say `[a-zA-Z]{3}`. The `{3}` at the end of the brackets means "do this three times." We can do the same thing and simplify `[0-9][0-9][0-9]` to `[0-9]{3}`.



Which in all looks like `[a-zA-Z]{3}[0-9]{3}`.

Simplify more with a shortcut

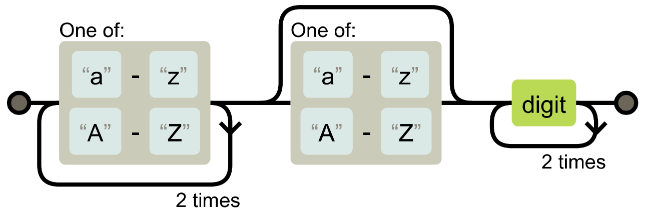
People need to search for any number a lot when writing regex, so there's a shortcut included for `[0-9]`, which is `\d` (short for digit).



Which looks like `[a-zA-Z]{3}\d{3}`.

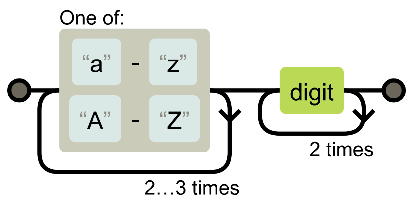
Getting in the middle initial

LinkBlue logins can start with either three or four letters. There are at least two ways we could make this work in regex. The first is to include an optional character. This is a character, set of options, or a range that you follow with a `?`.



Which looks like `[a-zA-Z]{3}[a-zA-Z]?\d{3}`

Or, we could tell the loop to run 3 *or* 4 times

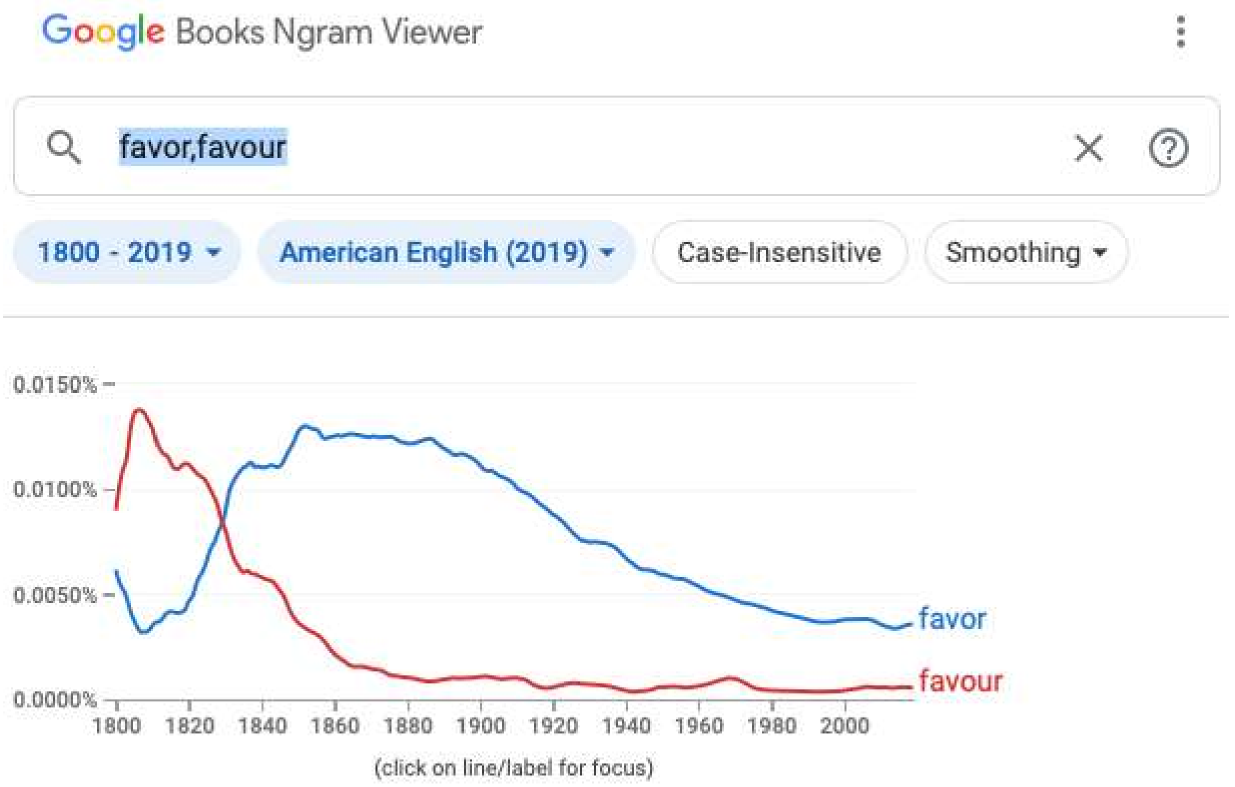


Which would look like `[a-zA-Z]{3,4}\d{3}`

This is probably the most compact regular expression that would successfully match everyone's LinkBlue id, and would reject anything that's not formatted like a LinkBlue ID.

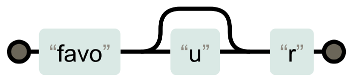
Another Application

Those of you playing Wordle might have been aware that there was a bit of controversy on the day the word was "FAVOR" for British players. That's because in British spelling, it's "FAVOUR". The "our" spelling isn't exclusive to British texts though. In older American books, the "our" spelling was predominant, and eventually got replaced by the "or" spelling.



If I wanted to search a mixture of texts written in both British and American texts, and wanted to do just one search for the word "favo(u)r", I could do it with the `?` operator in regex.

In my plaintext book folder, I've saved both the Grimm's Fairy Tales, which have been translated into British spelling, and A Connecticut Yankee in King Arthur's Court by Mark Twain, which is written in American spelling. To search for the word favor in both of these texts, I'll tell regex to optionally include a `u`.



Which looks like `favou?r`

To use the option operator, we need to use a slightly different `grep` called `egrep`.

To search both texts, I'll tell `egrep` to search `*`, which means "everything you find".

```
egrep --color=auto "favou?r" *
```

Which gets me back results from both books!

- ct_yank.txt:their way into favor. We had a steamboat or two on the Thames,
- ct_yank.txt:Everything would be favorable; it was balmy and beautiful spring
- grimm.txt:People are so kind; they seem really to think I do them a favour in
- grimm.txt:favourite plaything; and she was always tossing it up into the air, and