Regular Expressions

Powerful pattern matching

What is a "regular expression" or "regex"?

Regular expressions provide a way to look for patterns of characters within text when you know the pattern you need to find but not exactly where or how within the text it may appear.

For example

Consider the text strings:

1. "I love data journalism"
2. "Data j is the best!"
3. "Analyzing data is so much fun."

The word "data" appears in all of them, but in different places and (slightly) different forms. A "find" search could locate them, but this wouldn't help you flag spreadsheet entries that had that term.

How does a regex work?

Regular expressions use a kind of shorthand to represent the organization and repetition of individual characters or groups of characters (sometimes called *classes* of characters). It then looks at each text string one character at a time to see if it fits within the rules described by the regular expression.

Essentially, a regular expression describes a set of rules for sorting text strings.

Another example

Recall our three strings:

1. "I love data journalism"
2. "Data j is the best!"
3. "Analyzing data is so much fun."

Sometimes "data" is at the beginning, sometimes it's not. In one case, it's capitalized. A regular expression that would find all these instances of "data" would be:

.*[Dd]ata.*

But why?

A closer look

.*[Dd]ata.*

Let's break this down.

-> .* The period (.) stands for "any character". The asterisk (*) means "zero or more times."

-> [Dd] Putting brackets around a set of characters is an "or," i.e. "A capital or lowercase d."

-> ata After the D or d, you must find *exactly* the letters ata

-> .* But that can be followed by any character, zero or more times.

A little grammar

*	Zero or more times (applies to *preceding* character)
+	One or more time (applies to *preceding* character)
^	"Not" (applies to *subsequent* character)
\	"Escape" (applies to *subsequent* character)
[]	"Or" (applies to bracketed characters or classes)
&&	"And" (applies to adjacent characters or classes)
{n}	"Exactly 'n' times" (applies to *preceding* character)

A little vocabulary

.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

A little additional reading

Regular expressions are a features of almost all programming languages because they are fast and powerful. Please read this tutorial about regular expressions in OpenRefine.