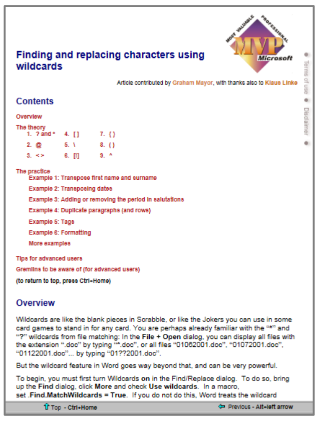
Regular Expressions: Further Investigations

Resources to support Tenderfoot Unit 5: Theoretical Computers

Suggestions for teachers

Regular expressions are closely related to finite state automata, and both are bound up with formal languages. Every *finite state automaton* can be converted to a *regular expression* that shows exactly what it does (and doesn't) match. Regular expressions are usually easier for humans to read. For machines, a computer program can convert any regular expression to an FSA, and then the computer can follow very simple rules to check the input.

Regular expressions are a simple way to search for things in an input, or to specify what kind of input will be accepted as legitimate. For example, many web scripting programs use them to check input for patterns like dates, email addresses and URLs. Applications like spreadsheets and word processors may have them and they're built into most programming languages. Whilst all are based on some foundation symbols, they differ in some particular symbol options they offer. For this reason, at this level it is probably best to select one context for exercises to minimise any potential confusion for students. By the same token, it is probably best to develop exercises that focus on the basic, widely used common symbol set.



Finding and replacing characters using wildcards

Article contributed by Graham Mayor, with thanks also to Klaus Linke

Contents

Overview

The theory

1. ? and * 4. [ ] 7. { }
2. @ 5. \ 8. ( )
3. < > 6. [!] 9. ^

The practice

Example 1: Transpose first name and surname
Example 2: Transposing dates
Example 3: Adding or removing the period in salutations
Example 4: Duplicate paragraphs (and rows)
Example 5: Tags
Example 6: Formatting
More examples

Tips for advanced users

Gremlins to be aware of (for advanced users)

(to return to top, press Ctrl+Home)

Overview

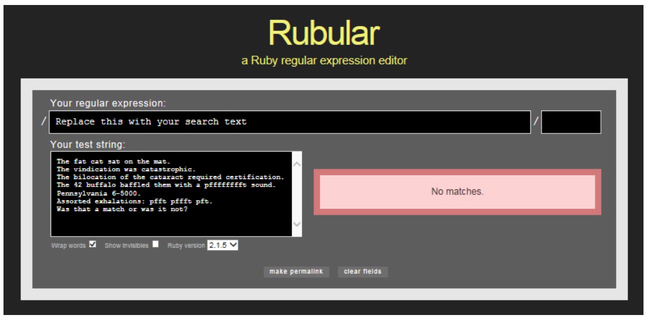
Wildcards are like the blank pieces in Scrabble, or like the Jokers you can use in some card games to stand in for any card. You are perhaps already familiar with the "*" and "?" wildcards from file matching: In the **File + Open** dialog, you can display all files with the extension ".doc" by typing "*.doc", or all files "01062001.doc", "01072001.doc", "01122001.doc"... by typing "01??2001.doc".

But the wildcard feature in Word goes way beyond that, and can be very powerful.

To begin, you must first turn Wildcards **on** in the Find/Replace dialog. To do so, bring up the **Find** dialog, click **More** and check **Use wildcards**. In a macro, set .**Find.MatchWildcards = True**. If you do not do this, Word treats the wildcard

The simplest kind of exercise is searching for matching text. This has particular practical value for students, who should be getting to grips with basic editing tools and techniques.

Microsoft Word has a Find command which can provide a good starting point and introduce children to the utility of the Find / Replace feature. When selected, if you enable the 'Use Wildcards' option, it can implement regular expressions. Graham Mayor provides instructions for using wildcards at goo.gl/I9XCoq. The basic symbols used are common to most regular expression editors. The article concludes with some excellent practical exercises, such as reversing forename and surnames in a list, transposing dates, and finding and formatting text such as quotations.



```
/ Replace this with your search text /
```

```
The fat cat sat on the mat.
The vindication was catastrophic.
The bilocation of the cataract required certification.
The 42 buffalo baffled them with a pffffffffft sound.
Pennsylvania 6-5000.
Assorted exhalations: pfft pffft pft.
Was that a match or was it not?
```

Rubular is a regular expression editor for the Ruby programming language. The symbol set it uses is common to many high level programming languages. You can access a simple text matching exercise using this link: goo.gl/TbAhYg A new window to the Rubular system will open as shown. If you enter "cat", it should find 6 matches in the test strings. Now try typing a dot (full stop) as the fourth character: "cat.". In a regular expression, "." can match any single character.

Try adding more dots before and after "cat". How about "cat.s" or "cat..n"?

Now try searching for "ic.". The "." matches any letter, but if you really wanted a full stop, you need to write it like this "ic\." The backslash 'escapes' the dot from being a regex symbol, and treats it as a character to match. You can use this search to find "ic" at the end of a sentence.

Another special symbol is "\d", which matches any digit. Try matching 2, 3 or 4 digits in a row (for example, two digits in a row is "\d\d").

To choose from a small set of characters, try "[ua]ff". Either of the characters in the square brackets will match. Try writing a regular expression that will match "fat", "sat" and "mat", but not "cat".

A suitable expression is [fsm]at

A shortcut for "[mnopqrs]" is "[m-s]"; try "[m-s]at" and "[4-6]".

BY NC SA

Source: CS Field Guide, New Zealand
csfieldguide.org.nz

Another useful shortcut is being able to match repeated letters. There are four common rules:

- a* matches 0 or more repetitions of a
- a+ matches 1 or more repetitions of a
- a? matches 0 or 1 occurrences of a (that is, a is optional)
- a{5} matches "aaaaa" (that is, a repeated 5 times)

```
f+
pf*t
af*
f*t
f{5}
.{5}n
```

Try experimenting with the examples in the box left.

If you want to choose between options, the vertical bar is useful. Try the expressions in the box rightt and work out what they match. You can type extra text into the test string area in Rubular if you want to experiment.

Notice the use of brackets to group parts of the regular expression. It's useful if you want the "+" or "*" to apply to more than one character.

```
was|that|hat
was|t?hat
th(at|e) cat
[Tt]h(at|e) [fc]at
(ff)+
f(ff)+
```

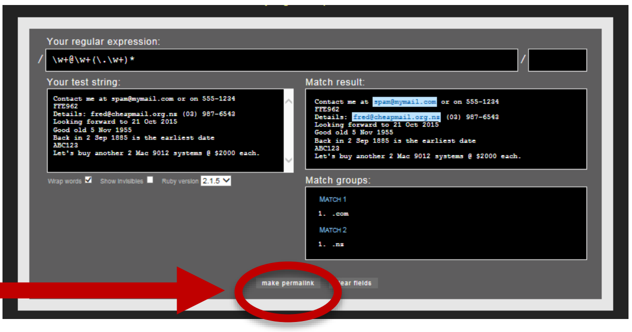
Once familiar with these try to write a regex that matches the first two words, but not the last three in the following link: goo.gl/9ZniUj.

Of course, regular expressions are mainly used for more serious purposes. Click on the following challenge to get some new text to search: goo.gl/MWIYCY. Can you write an expression to find the dates in the text? Here's one option, but it's not perfect: \d [A-Z][a-z][a-z] \d\d\d\d Can you improve on it? What about phone numbers? You'll need to think about what variations of phone numbers are common! How about finding email addresses?

Some of these are difficult challenges. A great feature of Rubular is that you can develop your own challenges with sets of test strings for students to use.

Note the 'make permalink' option in the screenshot.

This allows you to prepare your own test strings and share a link (like the examples above) with your students. If you are struggling with the e-mail challenge, the screenshot shows an example solution.



Your regular expression:

```
/\w+@\w+(\.\w+)*/
```

Your test string:

```
Contact me at spam@mymail.com or on 555-1234
FFE962
Details: fred@cheapmail.org.nz (03) 987-6543
Looking forward to 21 Oct 2015
Good old 5 Nov 1955
Back in 2 Sep 1885 is the earliest date
ABC123
Let's buy another 2 Mac 9012 systems @ $2000 each.
```

Wrap words ☑ Show invisibles ☐ Ruby version 2.1.5

Match result:

```
Contact me at spam@mymail.com or on 555-1234
FFE962
Details: fred@cheapmail.org.nz (03) 987-6543
Looking forward to 21 Oct 2015
Good old 5 Nov 1955
Back in 2 Sep 1885 is the earliest date
ABC123
Let's buy another 2 Mac 9012 systems @ $2000 each.
```

Match groups:

```
MATCH 1
1. .com
MATCH 2
1. .nz
```

Regular expressions do have their limits — for example, you won't be able to create one that can match palindromes (words and phrases that are the same backwards as forwards, such as "kayak", "rotator" and "hannah"), and you can't use one to detect strings that consist of n repeats of the letter "a" followed by n repeats of the letter "b". There are other systems for doing that and these are an important part of the theory of Formal Languages. For computer scientists, an important part of the value of Finite State Machines and other theoretical models is in exploring the limitations of what they can do. Nevertheless, regular expressions are very useful for a lot of common pattern matching requirements.

As noted previously, any regular expression has a corresponding Finite State Automata. At KS4 or KS5, you might want to consider challenging students to code a Finite State Automata. It's a fairly straightforward challenge. Each state can be defined as a function, with the transitions expressed in a series of IF statements. The input string needs to be sliced, the first character examined and passed to the start state function.

Subsequent characters can be retrieved by putting the string slicing within a counter controlled loop. The current state can be tracked with a variable.

Much of the material for this section comes from the CS Field Guide produced in New Zealand. The student guide can be found at csfieldguide.org.nz, with supplementary material in the teacher guide at goo.gl/nMpwPZ.