Regular expressions

In Word, when you search for `hello`, it will find all sequences of five characters that match these letters. Word gives you the option of the search being case sensitive or not. If the search is case sensitive, then hello will only find the string of characters hello. If the case is not case sensitive, `hello` will find `hello`, `Hello`, `hEllo`, `HEllo`, `heLlo`, `HeLlo`, etc. There is no way in Word, however, to find only `Hello` or `hello` but nothing else.

Matching a single character

For a regular expression, any letter or digit matches exactly that character. Thus, the regular express

```
Q
```

matches a capital 'Q' and nothing else. If you want to match one of a number of letters or digits, you put all the characters you want inside brackets. For example,

```
[aeiou]
```

matches all lower case vowels, and

```
[aeiouAEIOU]
```

matches either a lower- or upper-case vowel. To match a digit, you could use

```
[0123456789]
```

and to represent any lower-case letter or lower-case consonants, you could use

```
[abcdefghijklmnopqrstuvwxyz]
  [bcdfghjklmnpqrstvwxyz]
```

respectively, but regular expressions allow a short-cut, where a dash can be used to represent a sequence of digits, so

```
[0-9]
```

matches a digit and

```
[1-9]
```

matches a non-zero digit. You can use the same to represent ranges of letters:

```
[a-z]
[A-Z]
[a-zA-Z]
[bcdfghj-np-tv-z]
```

which match any lower-case letter, any upper-case letter, any letter, or any lower-case consonant.

Inside brackets, you may assume that most characters other than the dash (`-`) represent that actual character. There are a few exceptions, such as the dash which must be escaped, so to search for either an '`a`' or a '`-`', your regular expression is `[a\-]` or `[\-a]`. As you may guess, if you want to search for a backslash, you must also escape that character, so `[\\\-a-z]` matches either a backslash, a dash or a lower-case letter.

If you want to match any character what-so-ever, you use the period '`.`' Thus, if you want to match an actual period, you must escape it, so the pattern would be

```
\.
```

Similarly, if you want to match a backslash, you must escape that, as well.

Matching repetitions of a character

You can now take any pattern in the previous section and describe multiple instances of that pattern as follows. Let *r* represent any pattern described previously, so:

r`?`	Match zero or one instances of the pattern *r*.
r`*`	Match zero or more consecutive instances of the pattern *r*.
r`+`	Match one or more consecutive instances of the pattern *r*.

Thus, for example,

- `a?` matches the letter '`a`' or nothing.
- `6*` matches any number of consecutive sequence of the digit '`6`', including none.
- `.+` matches any consecutive sequence of one characters.
- `[aeiou]?` matches either a vowel or nothing.
- `[0-9]*` matches any number of consecutive digits, including none.
- `[a-z]+` matches any consecutive sequence of one or more lower-case letters.

For example, `[xo]+` will match all of `x`, `xox`, `xooxoxoxoxxxo` and `ooooooooooo`.

You may wonder why we may want to match zero instances of a pattern, but that will become clear in the next section.

Matching consecutive characters

If *r* and *s* are two different patterns described above, then *rs* matches anything that matches the first pattern *r* followed by anything that matches the second pattern *s*. Therefore,

```
[Hh]i
```

matches either a lower-case or upper-case '`h`' followed by an '`i`', so either `Hi` or `hi`. On the other hand,

```
[+-]?[0-9]
```

will match any of

```
 0  1  2  3  4  5  6  7  8  9
-0 -1 -2 -3 -4 -5 -6 -7 -8 -9
+0 +1 +2 +3 +4 +5 +6 +7 +8 +9
```

Similarly, `[a-z][0-9]*` will match any lower case letter followed by zero or more digits.

It is, of course possible to string an arbitrary number of patterns together. For example,

```
[BCDFGHJ-NP-TV-Z][aeiou][bcdfghj-np-tv-z]e
```

will match an upper-case consonant followed by a lower-case vowel, followed by another lower-case consonant, and ending in 'e'. If you wanted to make the trailing 'e' optional, you could use

```
[BCDFGHJ-NP-TV-Z][aeiou][bcdfghj-np-tv-z]e?
```

Describe in English what the following pattern describes:

```
[_a-zA-Z][_a-zA-Z0-9]*
```

Thus, it is now possible to describe all integers as being one of the following two:

```
[+-]?0  or  [+-]?[1-9][0-9]*
```

The first matches an optional '+' or '-' followed by a '0', while the second matches an optional sign followed by a non-zero digit, followed by zero or more other digits.

Matching one of two or more characters

Suppose you want to match one of two or more possible patterns. In this case, you need to join the patterns separated by a pipe '|'. For example,

```
[+-]?0|[+-]?[1-9][0-9]*
```

matches either a `0`, `+0` or `-0` or any sequence of digits starting with a non-zero digit possibly proceeded by either a '+' or a '-'. If you wanted to match "hello" in any one of a number of languages, you could use:

```
[Hh]ello|[Aa]llo|[Bb]onjour|[Hh]ola|[Cc]iao
```

Matching repetitions of patterns

Suppose you want to match either any of the following:

```
Douglas Harder
Douglas Wilhelm Harder
Doug Harder
Doug Wilhelm Harder
```

If we were to use `Douglas?`, this would only make the 's' optional. To indicate that an entire pattern is optional, we wrap that pattern in parentheses, so `Doug(las)?` matches either `Doug` or `Douglas`. Extrapolating this idea,

```
Doug(las)? (Wilhelm )?Harder
```

will match all four strings above, and nothing else. Question: what does

```
Doug(las)? (Wilhelm)? Harder
```

match?

Similarly, `(xo)+` will match `xo`, `xoxo`, `xoxoxo`, and so on but not, for example, `x`, `xox` or `xxoo`.

Previously, we saw that

```
[+-]?0|[+-]?[1-9][0-9]*
```

matches the representation of literal integers in C++; however both start with the optional sign. We could therefore write this more sussinctly as

```
[+-]?(0|[1-9][0-9]*)
```

This matches an optional sign followed by either a zero or any sequence of one or more digits that starts with a non-zero digit.

Note for advanced students

The above describes a subset of one version of regular expression. When you use a tool such as the vi editor, you will have to learn the patterns for that tool separately, so for example, some characters that are not escaped above will need to be, and vice versa.