Perl Compatible Regular Expressions in a Nutshell

All features on this cheat sheet are available in latest Perl, however, the implementation and UTF-8 support in other environments may differ slightly, e.g. Ruby supports named groups and UTF-8 only since version 1.9.1.

Syntax

Regular expressions are usually wrapped in forward slashes (e.g. /match me/), however, you can use almost any other character for instance to prevent heavy quoting when matching against file paths (e.g. \bin/grep\ is the same as /\bin/\grep/).

Perl:
```perl
$string =~ m/pattern/modifier
$string =~ s/pattern/replace/flags
```

Ruby:
```ruby
string.match(/pattern/flags) or string.match(/pattern/flags) |{|match|...}
string.sub(/pattern/flags, replace) or string.sub(/pattern/flags) |{|replace|...}
string.gsub(/pattern/flags, replace) or string.gsub(/pattern/flags) |{|replace|...}
```

Python:
```python
import re
re.search("flags", string)
re.sub("flags", replace, string, count)
```

PHP:
```php
preg_match('/pattern/flags', $string)
preg_replace('/pattern/flags', 'replace', $string)
```

Java:
```java
string.matches("flags\ pattern")
string.replaceFirst("flags\ pattern", "replace")
string.replaceAll("flags\ pattern", "replace")
```

JavaScript:
```javascript
string.match("flags\ pattern")
string.replace(/pattern/flags, "replace")
```

C++: (requires Boost.Regex library)
```cpp
boost::regex_search(string, boost::regex("pattern", "flags"))
boost::regex_replace(string, boost::regex("pattern", "flags"), "replace")
```

Objective-C: (requires RegexKit or RegexKit Lite)
```objc
NSString *flags = [NSString stringWithFormat:\"flags\ pattern\ replace:RKReplaceAll withString:@\"replace\"];
NSString *flags = [NSString stringWithFormat:\"flags\ pattern\ replace:1 withString:@\"replace\"];
NSString *flags = [NSString stringWithFormat:\"flags\ pattern\ replace:2 withString:@\"replace\"];
NSString *flags = [NSString stringWithFormat:\"flags\ pattern\ replace:RKReplaceAll withString:flags:8\"replace\"];
```

Shell:
```
% grep -P "flags\ pattern" file.txt
```

Characters

These are the usual suspects well known from any C-ish language:
```
a match the character a
3 match the number 3
$ or \$ match the contents of a variable \$a (e.g. Perl) or a (e.g. Ruby) respectively
\n newline (NL, LF)
\r return (CR)
\f form feed (FF)
\t tab (TAB)
\x3C character with the hex code 3C
\u561A character with the hex code 561A
\e escape character (alias \u001b)
\c control character
```

Wildcards

Wildcards match if a character belongs to the designated class of characters:
```
\w alphanumeric + underscore (shortcut for {0-9a-zA-Z_})
\W any character not covered by \w
\d numeric (shortcut for {0-9})
\D any character not covered by \d
\s whitespace (shortcut for {\n\t\f})
\S any character not covered by \s
\[ ] any character listed: \[a1*d-g\] means the characters a, 5, 1, * and \d, e, f, g
\[^-] any character not listed: \[^a5*\d-g\] means anything but the characters a, 5, 1, * and \d, e, f, g
```

Boundaries

Boundaries match the spots between characters and therefore have no width of their own (also called zero-width, \a extensions):
```
\b matches at a word boundary (spot between \w and \w)
\s matches anything but a word boundary
```
The following symbols have special meanings in the `replace` part:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>|</code>, <code>\2</code> etc</td>
<td>Include the contents of the corresponding group (⇒ grouping)</td>
</tr>
<tr>
<td><code>\1:000</code></td>
<td>Same as the above, use curly brackets if numbers follow the symbol</td>
</tr>
<tr>
<td><code>\1</code></td>
<td>Lowercase the following character</td>
</tr>
<tr>
<td><code>\1 \&amp;</code></td>
<td>Lowercase all characters in between</td>
</tr>
<tr>
<td><code>\1 \&amp;</code></td>
<td>Uppercase the following character</td>
</tr>
<tr>
<td><code>\1 \&amp;</code></td>
<td>Uppercase all characters in between</td>
</tr>
</tbody>
</table>

Flags

Optional flags determine the behaviour of the regex as a whole. May be used within the `{flags}` construct (⇒ extensions):

- `i` case-insensitive pattern matching
- `m` multiple lines: . does not match \n (Ruby uses this per default)
- `s` single line: . matches \n (Ruby uses `m` for this instead)
- `x` ignore whitespaces in pattern for better readability

The following `[cancel]` can be used within the `{flags}` construct:

- `g` apply the regex as many times as possible (i.e. for global replace)
- `e` evaluate the `replace` part as if it were source code (⇒ DANGEROUS!!)
- `o` compile the pattern only once and therefore perform variable substitutions only once