Regex cheatsheet

Many programs use regular expression to find & replace text. However, they tend to come with their own different flavor.

You can probably expect most modern software and programming languages to be using some variation of the Perl flavor, "PCRE"; however command-line tools (grep, less, ...) will often use the POSIX flavor (sometimes with an extended variant, e.g. `egrep` or `sed -r`). Vim also comes with its own syntax (a superset of what Vi accepts).

This cheatsheet lists the respective syntax of each flavor, and the software that uses it.

If you spot errors or missing data, or just want to make this prettier/more accurate, don't hesitate to open an issue or a pull request.

Syntax

What	Perl/PCRE	Python's `re`	POSIX (BRE)	POSIX extended (ERE)	Vim
Basics					
Custom character class	`[...]`	`[...]`	`[...]`	`[...]`	`[...]`
Negated custom character class	`[^...]`	`[^...]`	`[^...]`	`[^...]`	`[^...]`
\ special in class?	yes	yes	no, `]` escaped if comes first	no, `]` escaped if comes first	yes
Ranges	`[a-z]`, `-` escaped if comes last	`[a-z]`, `-` escaped if first or last	`[a-z]`, `-` escaped if comes last		`[a-z]`, `-` escaped if comes last
Alternation	`|`	`|`	`|`	`|`	`|` `\&` (low precedence)
Escaped character	`\033` `\x1B` `\x{1234}` `\N{name}` `\N{U+263D}`	`\x12`			`\%d123` `\%x2A` `\%u1234` `\%U1234ABCD`
Character classes					
Any character (except newline)	`.`	`.`	`.`	`.`	`.`
Any character (including newline)					`\_.`
Match a "word" character (alphanumeric plus `_`)	`\w` `[[:word:]]`	`\w`	`\w`	`\w`	`\w`
Case	`[[:upper:]]` / `[[:lower:]]`		`[[:upper:]]` / `[[:lower:]]`	`[[:upper:]]` / `[[:lower:]]`	`\u` `[[:upper:]]` / `\l` `[[:lower:]]`

What	Perl/PCRE	Python's `re`	POSIX (BRE)	POSIX extended (ERE)	Vim
Match a non-"word" character	`\W`	`\W`			`\W`
Match a whitespace character (except newline)			`\s` `[[:space:]]`	`\s` `[[:space:]]`	`\s` `[[:space:]]`
Whitespace including newline	`\s` `[[:space:]]`	`\s`			`\_s`
Match a non-whitespace character	`\S`	`\S`	`[^[:space:]]`	`[^[:space:]]`	`\S` `[^[:space:]]`
Match a digit character	`\d` `[[:digit:]]`	`\d`	`[[:digit:]]`	`[[:digit:]]`	`\d` `[[:digit:]]`
Match a non-digit character	`\D`	`\D`	`[^[:digit:]]`	`[^[:digit:]]`	`\D` `[^[:digit:]]`
Any hexadecimal digit	`[[:xdigit:]]`		`[[:xdigit:]]`	`[[:xdigit:]]`	`\x` `[[:xdigit:]]`
Any octal digit					`\o`
Any graphical character excluding "word" characters	`[[:punct:]]`		`[[:punct:]]`	`[[:punct:]]`	`[[:punct:]]`
Any alphabetical character	`[[:alpha:]]`		`[[:alpha:]]`	`[[:alpha:]]`	`\a` `[[:alpha:]]`
Non-alphabetical character			`[^[:alpha:]]`	`[^[:alpha:]]`	`\A` `[^[:alpha:]]`
Any alphanumerical character	`[[:alnum:]]`		`[[:alnum:]]`	`[[:alnum:]]`	`[[:alnum:]]`
ASCII	`[[:ascii:]]`				
Character equivalents (e = é = è) (as per locale)			`[[=e=]]`	`[[=e=]]`	`[[=e=]]`
Zero-width assertions					
Word boundary	`\b`	`\b`	`\b` / `\<` (start) / `\>` (end)	`\b` / `\<` (start) / `\>` (end)	`\<` (start) / `\>` (end)

What	Perl/PCRE	Python's `re`	POSIX (BRE)	POSIX extended (ERE)	Vim
Anywhere but word boundary	`\B`	`\B`	`\B`	`\B`	
Beginning of line/string	`^` / `\A`	`^` / `\A`	`^`	`^`	`^` (beginning of pattern) `\_^`
End of line/string	`$` / `\Z`	`$` / `\Z`	`$`	`$`	`$` (end of pattern) `\_$`
Captures and groups					
Capturing group	`(...)` `(?<name>...)`	`(...)` `(?P<name>...)`	`\(...\)`	`(...)`	`\(...\)`
Non-capturing group	`(?:...)`	`(?:...)`			`\%(...\)`
Backreference to a specific group.	`\1` `\g1` `\g{-1}`	`\1`	`\1`	`\1` non-official	`\1`
Named backreference	`\g{name}` `\k<name>`	`(?P=name)`			
Look-around					
Positive look-ahead	`(?=...)`	`(?=...)`			`\(...\)\@=`
Negative look-ahead	`(?!...)`	`(?!...)`			`\(...\)\@!`
Positive look-behind	`(?<=...)`	`(?<=...)`			`\(...\)\@<=`
Negative look-behind	`(?<!...)`	`(?<!...)`			`\(...\)\@<!`
Multiplicity					
0 or 1	`?`	`?`	`\?`	`?`	`\?`
0 or more	`*`	`*`	`*`	`*`	`*`
1 or more	`+`	`+`	`\+`	`+`	`\+`
Specific number	`{n}` `{n,m}` `{n,}`	`{n}` `{n,m}` `{n,}`	`\{n\}` `\{n,m\}` `\{n,\}`	`{n}` `{n,m}` `{n,}`	`\{n}` `\{n,m}` `\{n,}`
0 or 1, non-greedy	`??`	`??`			

What	Perl/PCRE	Python's `re`	POSIX (BRE)	POSIX extended (ERE)	Vim
0 or more, non-greedy	`*?`	`*?`			`\{-}`
1 or more, non-greedy	`+?`	`+?`			
Specific number, non-greedy	`{n,m}?` `{n,}?`	`{n,m}?` `{n,}?`			`\{-n,m}` `\{-n,}`
0 or 1, don't give back on backtrack	`?+`				
0 or more, don't give back on backtrack	`*+`				
1 or more, don't give back on backtrack	`++`				
Specific number, don't give back on backtrack	`{n,m}+` `{n,}+`				
Other					
Independent non-backtracking pattern	`(?>...)`				`\(...\)\@>`
Make case-sensitive/insensitive	`(?i)` / `(?-i)`	`(?i)` / `(?-i)`			`\c` / `\C`

Programs

What	Syntax	Comments/gotchas
Programming languages		
Perl	PCRE	PCRE is actually a separate implementation from Perl's, with slight differences
Python's `re` standard lib	Python's own syntax (Perl-inspired)	
Ruby	Ruby's own syntax (Perl-inspired)	

What	Syntax	Comments/gotchas
Java's java.util.regex	Almost PCRE	
Boost.Regex	PCRE	
Text editors		
Eclipse	PCRE	
Emacs	?	
Netbeans	PCRE	
Notepad++	PCRE (Boost.Regex)	
PyCharm	PCRE	Perl-inspired
Sublime Text	?	
UltraEdit	PCRE	
Vim	Vim	
Command-line tools		
awk	ERE	might depend on the implementation
grep	BRE, `egrep` for ERE, `grep -P` for PCRE (optional)	
less	ERE	usually; man page says "regular expression library supplied by your system"
screen	plain text	
sed	BRE, `-E` switches to ERE	