RUBY REGULAR EXPRESSIONS

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings using a specialized syntax held in a pattern.

A *regular expression literal* is a pattern between slashes or between arbitrary delimiters followed by %r as follows:

Syntax:

```
/pattern/
/pattern/im    # option can be specified
%r!/usr/local! # general delimited regular expression
```

Example:

```
#!/usr/bin/ruby

line1 = "Cats are smarter than dogs";
line2 = "Dogs also like meat";

if ( line1 =~ /Cats(.*)/ )
  puts "Line1 contains Cats"
end
if ( line2 =~ /Cats(.*)/ )
  puts "Line2 contains  Dogs"
end
```

This will produce the following result:

```
Line1 contains Cats
```

Regular-expression modifiers:

Regular expression literals may include an optional modifier to control various aspects of matching. The modifier is specified after the second slash character, as shown previously and may be represented by one of these characters:

Modifier	Description
i	Ignore case when matching text.
o	Perform #{} interpolations only once, the first time the regexp literal is evaluated.
x	Ignores whitespace and allows comments in regular expressions
m	Matches multiple lines, recognizing newlines as normal characters
u,e,s,n	Interpret the regexp as Unicode $UTF-8$, EUC, SJIS, or ASCII. If none of these modifiers is specified, the regular expression is assumed to use the source encoding.

Like string literals delimited with %Q, Ruby allows you to begin your regular expressions with %r followed by a delimiter of your choice. This is useful when the pattern you are describing contains a lot of forward slash characters that you don't want to escape:

```
# Following matches a single slash character, no escape required
%r|/|
# Flag characters are allowed with this syntax, too
%r[</(.*)>]i
```

Regular-expression patterns:

Except for control characters, **+?. *$([ ] { } | \)**, all characters match themselves. You can escape a control character by preceding it with a backslash.

Following table lists the regular expression syntax that is available in Ruby.

Pattern	Description
^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character except newline. Using m option allows it to match newline as well.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
re	Groups regular expressions and remembers matched text.
?*imx*	Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.
? − *imx*	Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
?: *re*	Groups regular expressions without remembering matched text.
?*imx*: *re*	Temporarily toggles on i, m, or x options within parentheses.
? − *imx*: *re*	Temporarily toggles off i, m, or x options within parentheses.
?#...	Comment.
? = *re*	Specifies position using a pattern. Doesn't have a range.
?!*re*	Specifies position using pattern negation. Doesn't have a range.
? > *re*	Matches independent pattern without backtracking.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches nonwhitespace.

\d	Matches digits. Equivalent to [0-9].
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\b	Matches word boundaries when outside brackets. Matches backspace $0x08$ when inside brackets.
\B	Matches nonword boundaries.
\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

Regular-expression Examples:

Literal characters:

Example	Description
/ruby/	Match "ruby".
¥	Matches Yen sign. Multibyte characters are supported in Ruby 1.9 and Ruby 1.8.

Character classes:

Example	Description
/[Rr]uby/	Match "Ruby" or "ruby"
/rub[ye]/	Match "ruby" or "rube"
/[aeiou]/	Match any one lowercase vowel
/[0-9]/	Match any digit; same as /[0123456789]/
/[a-z]/	Match any lowercase ASCII letter
/[A-Z]/	Match any uppercase ASCII letter
/[a-zA-Z0-9]/	Match any of the above
/[^aeiou]/	Match anything other than a lowercase vowel
/[^0-9]/	Match anything other than a digit

Special Character Classes:

Example	Description
/./	Match any character except newline
/./m	In multiline mode . matches newline, too
/\d/	Match a digit: /[0-9]/
/\D/	Match a nondigit: /[^0-9]/
/\s/	Match a whitespace character: /[\t\r\n\f]/
/\S/	Match nonwhitespace: /[^ \t\r\n\f]/
/\w/	Match a single word character: /[A-Za-z0-9_]/
/\W/	Match a nonword character: /[^A-Za-z0-9_]/

Repetition Cases:

Example	Description
/ruby?/	Match "rub" or "ruby": the y is optional
/ruby*/	Match "rub" plus 0 or more ys
/ruby+/	Match "rub" plus 1 or more ys
/\d{3}/	Match exactly 3 digits
/\d{3,}/	Match 3 or more digits
/\d{3,5}/	Match 3, 4, or 5 digits

Nongreedy repetition:

This matches the smallest number of repetitions:

Example	Description
/<.*>/	Greedy repetition: matches "<ruby>perl>"
/<.*?>/	Nongreedy: matches "<ruby>" in "<ruby>perl>"

Grouping with parentheses:

Example	Description
/\D\d+/	No group: + repeats \d
/\D\d+/	Grouped: + repeats \D\d pair
/[Rr]uby(, ?)+/	Match "Ruby", "Ruby, ruby, ruby", etc.

Backreferences:

This matches a previously matched group again:

Example	Description
/[*Rr*]uby&\1ails/	Match ruby&rails or Ruby&Rails
/['"]?:(?!\1.)*\1/	Single or double-quoted string. \1 matches whatever the 1st group matched . \2 matches whatever the 2nd group matched, etc.

Alternatives:

Example	Description
/ruby|rube/	Match "ruby" or "rube"
/ruby|le)/	Match "ruby" or "ruble"
/ruby!+|\?/	"ruby" followed by one or more ! or one ?

Anchors:

This need to specify match position

Example	Description
/^Ruby/	Match "Ruby" at the start of a string or internal line
/Ruby$/	Match "Ruby" at the end of a string or line
/\ARuby/	Match "Ruby" at the start of a string
/Ruby\Z/	Match "Ruby" at the end of a string
/\bRuby\b/	Match "Ruby" at a word boundary
/\brub\B/	\B is nonword boundary: match "rub" in "rube" and "ruby" but not alone
/Ruby?=!/	Match "Ruby", if followed by an exclamation point
/Ruby?!!/	Match "Ruby", if not followed by an exclamation point

Special syntax with parentheses:

Example	Description
/R?#comment/	Matches "R". All the rest is a comment
/R?iuby/	Case-insensitive while matching "uby"
/R?i:uby/	Same as above
/rub?:y|le)/	Group only without creating \1 backreference

Search and Replace:

Some of the most important String methods that use regular expressions are **sub** and **gsub** , and their in-place variants **sub!** and **gsub!**.

All of these methods perform a search-and-replace operation using a Regexp pattern. The **sub** &

sub! replace the first occurrence of the pattern and **gsub** & **gsub!** replace all occurrences.

The **sub** and **gsub** return a new string, leaving the original unmodified where as **sub!** and **gsub!** modify the string on which they are called.

Following is the example:

```
#!/usr/bin/ruby

phone = "2004-959-559 #This is Phone Number"

# Delete Ruby-style comments
phone = phone.sub!(/#.*$/, "")
puts "Phone Num : #{phone}"

# Remove anything other than digits
phone = phone.gsub!(/\D/, "")
puts "Phone Num : #{phone}"
```

This will produce the following result:

```
Phone Num : 2004-959-559
Phone Num : 2004959559
```

Following is another example:

```
#!/usr/bin/ruby

text = "rails are rails, really good Ruby on Rails"

# Change "rails" to "Rails" throughout
text.gsub!("rails", "Rails")

# Capitalize the word "Rails" throughout
text.gsub!(/\brails\b/, "Rails")

puts "#{text}"
```

This will produce the following result:

```
Rails are Rails, really good Ruby on Rails
```