

Representation

'string literal in single quote'	
"escaped string \t \n"	
%q[a string]	single quoted string
%Q[escaped \t]	double quoted string
<<EOF {{multiline string}} EOF	double quoted string
<<'EOF' {{multiline string}} EOF	single quoted string
<<-EOF {{multiline string}} EOF	preserve line indentation
<<EOF.#{method} {{multiline string}} EOF	method is defined as a String instance method
<<eof str1 #{eof2} str2 eof	here doc can be nested

Length

s.length, s.size

Iterate Line, Byte, Char

each_char	{ char puts char}
each_line	{ line puts line}
each_byte	{ byte puts byte}
each_line.with_index	{ line, index puts line, index}
s.scan(/./) { char puts char}	. matches a char

Specialized string comparison

alias old_compare => * use the old_compare in the new <=> method
 redefine == method
 Redefine <=> method under String class

Tokenize

s1 = "It was a dark"; s1.split # ["It", "was", "dark"]
 s2.split(", ") use a string
 s3.split(/ and /) use regex
 s = "alpha,beta,gamma,"
 s.split(", ", 4) #["alpha", "beta", "gamma", ", "] - rest of the string
 s.split(", ", -1) ["alpha", "beta", "gamma", "", ""] - empty matches

Format a string

Scan a string

s = "a,a,a"; => ["a", "a", "a"]
 s.scan("a")
 s.scan(/w+/) regex or string
 str.scan(/w+/) {|x| puts x } each scan result is passed to the block
 StringScanner require 'strscan'
 ss = StringScanner.new(str) str = "Watch how I soar!"
 ss.bol? beginning of line
 ss.eos? end of string
 ss.pos, ss.pos= query or set current position
 ss.match, ss.matched?, ss.matched check if a match exists at pos
 ss.pre_match, ss.post_match check a pre or post match - only if matched?
 ss.rest, ss.rest? get the rest of the string

Scan a string (cont)

ss.reset, move to beginning or end
 ss.terminate
 ss.scan, scan and unscan
 ss.unscan
 ss.exist? check if a substring exists in the rest
 StringScanner acts like a file pointer - it forwards the pointer when there is a match, thus allowing for multiple scan capabilities on the same string.
 * usually methods ending with ? returns boolean

Sub strings

s[7,4] start index, length
 s[-3,3] index from end (end is -1), length
 s[8..13] index range
 s[/foo/]; regex or string
 s["foo"]
 s[4] single index
 s[8..13]= All above sub string look ups can also be used for assignment

Substitute / replace a string

s.sub(pattern, replacement) pattern - regex or string
 s.sub(pattern) {block} each match of the pattern is passed to the block
 special symbols \1, \2 can be used in replacement string; gsub is a method which replaces all matches while sub does it only for first match
 sub!, gsub! - inplace replacements



Search a string

<code>s.index(string regex)</code> <code>=> int</code>	<code>s.rindex(string regex)</code> <code>=> int</code>
<code>s.include?(string regex)</code> <code>=> bool</code>	<code>s.scan(str reg)</code> <code>=> array</code>

ASCII and Char conversion

<code>"A".ord => 65</code>	<code>65.chr => "A"</code>
<code>233.chr("UTF-8") => "é"</code>	



By **surendraa**

cheatography.com/surendraa/

Not published yet.

Last updated 3rd April, 2018.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>