RUBY STRINGS

A String object in Ruby holds and manipulates an arbitrary sequence of one or more bytes, typically representing characters that represent human language.

The simplest string literals are enclosed in single quotes *theapostrophecharacter*. The text within the quote marks is the value of the string:

```
'This is a simple Ruby string literal'
```

If you need to place an apostrophe within a single-quoted string literal, precede it with a backslash so that the Ruby interpreter does not think that it terminates the string:

```
'Won\'t you read O\'Reilly\'s book?'
```

The backslash also works to escape another backslash, so that the second backslash is not itself interpreted as an escape character.

Following are string-related features Ruby.

Expression Substitution:

Expression substitution is a means of embedding the value of any Ruby expression into a string using #{ and }:

```
#!/usr/bin/ruby

x, y, z = 12, 36, 72
puts "The value of x is #{ x }."
puts "The sum of x and y is #{ x + y }."
puts "The average was #{ (x + y + z)/3 }."
```

This will produce the following result:

```
The value of x is 12.
The sum of x and y is 48.
The average was 40.
```

General Delimited Strings:

With general delimited strings, you can create strings inside a pair of matching though arbitrary delimiter characters, e.g., !, , {, <, etc., preceded by a percent character (%. Q, q, and x have special meanings. General delimited strings can be

```
%{Ruby is fun.}  equivalent to "Ruby is fun."
%Q{ Ruby is fun. } equivalent to " Ruby is fun. "
%q[Ruby is fun.]  equivalent to a single-quoted string
%x!ls! equivalent to back tick command output `ls`
```

Escape Characters:

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

NOTE: In a double-quoted string, an escape character is interpreted; in a single-quoted string, an escape character is preserved.

Backslash notation	Hexadecimal character	Description

\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

Character Encoding:

The default character set for Ruby is ASCII, whose characters may be represented by single bytes. If you use UTF-8, or another modern character set, characters may be represented in one to four bytes.

You can change your character set using $KCODE at the beginning of your program, like this:

```
$KCODE = 'u'
```

Following are the possible values for $KCODE.

Code	Description
a	ASCII *sameasnone*. This is the default.
e	EUC.
n	None *sameasASCII*.
u	UTF-8.

String Built-in Methods:

We need to have an instance of String object to call a String method. Following is the way to create an instance of String object:

```
new [String.new(str="")]
```

This will return a new string object containing a copy of *str*. Now, using *str* object, we can all any available instance methods. For example:

```
#!/usr/bin/ruby

myStr = String.new("THIS IS TEST")
foo = myStr.downcase

puts "#{foo}"
```

This will produce the following result:

```
this is test
```

Following are the public String methods *AssumingstrisaStringobject*:

SN	Methods with Description
1	**str % arg** Formats a string using a format specification. arg must be an array if it contains more than one substitution. For information on the format specification. see sprintf under "Kernel Module."
2	**str * integer** Returns a new string containing integer times str. In other words, str is repeated integer imes.
3	**str + other_str** Concatenates other_str to str.
4	**str << obj** Concatenates an object to str. If the object is a Fixnum in the range 0.255, it is converted to a character. Compare it with concat.
5	**str <=> other_str** Compares str with other_str, returning -1 *lessthan*, 0 *equal*, or 1 *greaterthan*. The comparison is casesensitive.
6	**str == obj** Tests str and obj for equality. If obj is not a String, returns false; returns true if str <=> obj returns 0.
7	**str =~ obj** Matches str against a regular expression pattern obj. Returns the position where the match starts; otherwise, false.
8	**str.capitalize** Capitalizes a string.
9	**str.capitalize!** Same as capitalize, but changes are made in place.

10 **str.casecmp**

Makes a case-insensitive comparison of strings.

11 **str.center**

Centers a string.

12 **str.chomp**

Removes the record separator $/, usually \n, from the end of a string. If no record separator exists, does nothing.

13 **str.chomp!**

Same as chomp, but changes are made in place.

14 **str.chop**

Removes the last character in str.

15 **str.chop!**

Same as chop, but changes are made in place.

16 **str.concat**$other_str$

Concatenates other_str to str.

17 **str.count***str, . . .*

Counts one or more sets of characters. If there is more than one set of characters, counts the intersection of those sets

18 **str.crypt**$other_str$

Applies a one-way cryptographic hash to str. The argument is the salt string, which should be two characters long, each character in the range a.z, A.Z, 0.9, . or /.

19 **str.delete**$other_str$*, . . .*

Returns a copy of str with all characters in the intersection of its arguments deleted.

20 **str.delete!**$other_str$*, . . .*

Same as delete, but changes are made in place.

21 **str.downcase**

Returns a copy of str with all uppercase letters replaced with lowercase.

22 **str.downcase!**

Same as downcase, but changes are made in place.

23 **str.dump**

Returns a version of str with all nonprinting characters replaced by \nnn notation and all special characters escaped.

24 **str.each***separator* = $/ **{ |substr| block }**

Splits str using argument as the record separator *$/bydefault*, passing each substring to the supplied block.

25 **str.each_byte { |fixnum| block }**

Passes each byte from str to the block, returning each byte as a decimal representation of the byte.

26 **str.each_line***separator* = $/ **{ |substr| block }**

Splits str using argument as the record separator *$/bydefault*, passing each substring to the supplied block.

27 **str.empty?**

Returns true if str is empty *hasazerolength*.

28 **str.eql?***other*

Two strings are equal if the have the same length and content.

29 **str.gsub***pattern, replacement* **[or]**
str.gsub*pattern* **{ |match| block }**

Returns a copy of str with all occurrences of pattern replaced with either replacement or the value of the block. The pattern will typically be a Regexp; if it is a String then no regular expression metacharacters will be interpreted
thatis, /\d/willmatchadigit, but'\d'willmatchabackslashfollowedbya'd'

30 **str[fixnum] [or] str[fixnum,fixnum] [or] str[range] [or] str[regexp] [or] str[regexp, fixnum] [or] str[other_str]**

References str, using the following arguments: one Fixnum, returns a character code at fixnum; two Fixnums, returns a substring starting at an offset *firstfixnum* to length *secondfixnum*; range, returns a substring in the range; regexp returns portion of matched string; regexp with fixnum, returns matched data at fixnum; other_str returns substring matching other_str. A negative Fixnum starts at end of string with -1.

31 **str[fixnum] = fixnum [or] str[fixnum] = new_str [or] str[fixnum, fixnum] = new_str [or] str[range] = aString [or] str[regexp] =new_str [or] str[regexp, fixnum] =new_str [or] str[other_str] = new_str]**

Replace *assign* all or part of a string. Synonym of slice!.

33 **str.gsub!***pattern, replacement* **[or] str.gsub!***pattern* **{ |match| block }**

Performs the substitutions of String#gsub in place, returning str, or nil if no substitutions were performed.

34 **str.hash**

Returns a hash based on the string's length and content.

35 **str.hex**

Treats leading characters from str as a string of hexadecimal digits *withanoptionalsignandanoptional*0*x* and returns the corresponding number. Zero is returned on error.

36 **str.include? other_str [or] str.include? fixnum**

Returns true if str contains the given string or character.

37 **str.index*substring*[, *offset*] [or]**
str.index*fixnum*[, *offset*] [or]
str.index*regexp*[, *offset*]

Returns the index of the first occurrence of the given substring, character *fixnum*, or pattern *regexp* in str. Returns nil if not found. If the second parameter is present, it specifies the position in the string to begin the search.

38 **str.insert*index, other*$_s$*tr***

Inserts other_str before the character at the given index, modifying str. Negative indices count from the end of the string, and insert after the given character. The intent is to insert a string so that it starts at the given index.

39 **str.inspect**

Returns a printable version of str, with special characters escaped.

40 **str.intern [or] str.to_sym**

Returns the Symbol corresponding to str, creating the symbol if it did not previously exist.

41 **str.length**

Returns the length of str. Compare size.

42 **str.ljust*integer, padstr =* ″**

If integer is greater than the length of str, returns a new String of length integer with str left-justified and padded with padstr; otherwise, returns str.

43 **str.lstrip**

Returns a copy of str with leading whitespace removed.

44 **str.lstrip!**

Removes leading whitespace from str, returning nil if no change was made.

45 **str.matchpattern**

Converts pattern to a Regexp if it isn't already one, then invokes its match method on str.

46 **str.oct**

Treats leading characters of str as a string of octal digits with an optional sign and returns the corresponding number. Returns 0 if the conversion fails.

47 **str.replaceother_str**

Replaces the contents and taintedness of str with the corresponding values in other_str.

48 **str.reverse**

Returns a new string with the characters from str in reverse order.

49 **str.reverse!**

Reverses str in place.

50 **str.rindexsubstring [, fixnum] [or]**

str.rindexfixnum [, fixnum] [or]

str.rindexregexp [, fixnum]

Returns the index of the last occurrence of the given substring, character fixnum, or pattern regexp in str. Returns nil if not found. If the second parameter is present, it specifies the position in the string to end the search.characters beyond this point won't be considered.

51 **str.rjustinteger, padstr=' '**

If integer is greater than the length of str, returns a new String of length integer with str right-justified and padded with padstr; otherwise, returns str.

52 **str.rstrip**

Returns a copy of str with trailing whitespace removed.

53 **str.rstrip!**

Removes trailing whitespace from str, returning nil if no change was made.

54 **str.scanpattern [or]**

str.scanpattern { |match, ...| block }

Both forms iterate through str, matching the pattern which may be a Regexp or a String. For each match, a result is generated and either added to the result array or passed to the block. If the pattern contains no groups, each individual result consists of the matched string, $&. If the pattern contains groups, each individual result is itself an array containing one entry per group.

55 **str.slicefixnum [or] str.slicefixnum, fixnum [or]**

str.slicerange [or] str.sliceregexp [or]

str.sliceregexp, fixnum [or] str.sliceother_str

See str[fixnum], etc.

str.slice!fixnum [or] str.slice!fixnum, fixnum [or]

str.slice!range [or] str.slice!regexp [or]

str.slice!other_str

Deletes the specified portion from str, and returns the portion deleted. The forms that take a Fixnum will raise an IndexError if the value is out of range; the Range form will raise a RangeError, and the Regexp and String forms will silently ignore the assignment.

56 **str.splitpattern=$;, [limit]**

Divides str into substrings based on a delimiter, returning an array of these substrings.

If *pattern* is a String, then its contents are used as the delimiter when splitting str. If pattern is a single space, str is split on whitespace, with leading whitespace and runs of contiguous whitespace characters ignored.

If *pattern* is a Regexp, str is divided where the pattern matches. Whenever the pattern matches a zero-length string, str is split into individual characters.

If *pattern* is omitted, the value of ; is used. If ; is nil which is the default, str is split on whitespace as if ` ` were specified.

If the *limit* parameter is omitted, trailing null fields are suppressed. If limit is a positive number, at most that number of fields will be returned if limit is 1, the entire string is returned as the only entry in an array. If negative, there is no limit to the number of fields returned, and trailing null fields are not suppressed.

57 **str.squeeze[other_str]***

Builds a set of characters from the other_str parameters using the procedure described for String#count. Returns a new string where runs of the same character that occur in this set are replaced by a single character. If no arguments are given, all runs of identical characters are replaced by a single character.

58 **str.squeeze![other_str]***

Squeezes str in place, returning either str, or nil if no changes were made.

59 **str.strip**

Returns a copy of str with leading and trailing whitespace removed.

60 **str.strip!**

Removes leading and trailing whitespace from str. Returns nil if str was not altered.

61

str.subpattern, replacement [or]

str.subpattern { |match| block }

Returns a copy of str with the first occurrence of pattern replaced with either replacement or the value of the block. The pattern will typically be a Regexp; if it is a String then no regular expression metacharacters will be interpreted.

62

str.sub!pattern, replacement [or]

str.sub!pattern { |match| block }

Performs the substitutions of String#sub in place, returning str, or nil if no substitutions were performed.

63 **str.succ [or] str.next**

Returns the successor to str.

64 **str.succ! [or] str.next!**

Equivalent to String#succ, but modifies the receiver in place.

65 **str.sumn=16**

Returns a basic n-bit checksum of the characters in str, where n is the optional Fixnum parameter, defaulting to 16. The result is simply the sum of the binary value of each character in str modulo 2n - 1. This is not a particularly good checksum.

66 **str.swapcase**

Returns a copy of str with uppercase alphabetic characters converted to lowercase and lowercase characters converted to uppercase.

67 **str.swapcase!**

Equivalent to String#swapcase, but modifies the receiver in place, returning str, or nil if no changes were made.

68 **str.to_f**

Returns the result of interpreting leading characters in str as a floating-point number. Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of str, 0.0 is returned. This method never raises an exception.

69 **str.to_ibase=10**

Returns the result of interpreting leading characters in str as an integer base base 2, 8, 10, or 16. Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of str, 0 is returned. This method never raises an exception.

70 **str.to_s [or] str.to_str**

Returns the receiver.

71 **str.trfrom_str, to_str**

Returns a copy of str with the characters in from_str replaced by the corresponding characters in to_str. If to_str is shorter than from_str, it is padded with its last character. Both strings may use the c1.c2 notation to denote ranges of characters, and from_str may start with a ^, which denotes all characters except those listed.

72 **str.tr!from_str, to_str**

Translates str in place, using the same rules as String#tr. Returns str, or nil if no changes were made.

73 **str.tr_sfrom_str, to_str**

Processes a copy of str as described under String#tr, then removes duplicate characters in regions that were affected by the translation.

74 **str.tr_s!from_str, to_str**

Performs String#tr_s processing on str in place, returning str, or nil if no changes were made.

75 **str.unpackformat**

Decodes str which may contain binary data according to the format string, returning an array of each value extracted. The format string consists of a sequence of single-character directives, summarized in Table 18. Each directive may be followed by a number, indicating the number of times to repeat with this directive. An asterisk * will use up all remaining elements. The directives sSiIlL may each be followed by an underscore _ to use the underlying platform's native size for the specified type; otherwise, it uses a platform-independent consistent size. Spaces are ignored in the format string.

76 **str.upcase**

Returns a copy of str with all lowercase letters replaced with their uppercase counterparts. The operation is locale insensitive.only characters a to z are affected.

77 **str.upcase!**

Changes the contents of str to uppercase, returning nil if no changes are made.

78 **str.uptoother_str { |s| block }**

Iterates through successive values, starting at str and ending at other_str inclusive, passing each value in turn to the block. The String#succ method is used to generate each value.

String unpack directives:

Following table lists unpack directives for method String#unpack.

Directive	Returns	Description
A	String	With trailing nulls and spaces removed.
a	String	String.
B	String	Extract bits from each character most significant bit first.
b	String	Extract bits from each character least significant bit first.
C	Fixnum	Extract a character as an unsigned integer.
c	Fixnum	Extract a character as an integer.
D, d	Float	Treat sizeofdouble characters as a native double.
E	Float	Treat sizeofdouble characters as a double in littleendian byte order.
e	Float	Treat sizeoffloat characters as a float in littleendian byte order.
F, f	Float	Treat sizeoffloat characters as a native float.
G	Float	Treat sizeofdouble characters as a double in network byte order.
g	String	Treat sizeoffloat characters as a float in network byte order.

H	String	Extract hex nibbles from each character most significant bit first
h	String	Extract hex nibbles from each character least significant bit first.
I	Integer	Treat sizeofint modified by _ successive characters as an unsigned native integer.
i	Integer	Treat sizeofint modified by _ successive characters as a signed native integer.
L	Integer	Treat four modified by _ successive characters as an unsigned native long integer.
l	Integer	Treat four modified by _ successive characters as a signed native long integer.
M	String	Quoted-printable.
m	String	Base64-encoded.
N	Integer	Treat four characters as an unsigned long in network byte order.
n	Fixnum	Treat two characters as an unsigned short in network byte order.
P	String	Treat sizeofchar * characters as a pointer, and return \emph{len} characters from the referenced location.
p	String	Treat sizeofchar * characters as a pointer to a null-terminated string.
Q	Integer	Treat eight characters as an unsigned quad word 64 bits.
q	Integer	Treat eight characters as a signed quad word 64 bits.
S	Fixnum	Treat two different if _ used successive characters as an unsigned short in native byte order.
s	Fixnum	Treat two different if _ used successive characters as a signed short in native byte order.
U	Integer	UTF-8 characters as unsigned integers.
u	String	UU-encoded.
V	Fixnum	Treat four characters as an unsigned long in little-endian byte order.
v	Fixnum	Treat two characters as an unsigned short in little-endian byte order.
w	Integer	BER-compressed integer.
X		Skip backward one character.
x		Skip forward one character.
Z	String	With trailing nulls removed up to first null with *.
@		Skip to the offset given by the length argument.

Example:

Try the following example to unpack various data.

```
"abc \0\0abc \0\0".unpack('A6Z6')   #=> ["abc", "abc "]
"abc \0\0".unpack('a3a3')           #=> ["abc", " \000\000"]
"abc \0abc \0".unpack('Z*Z*')       #=> ["abc ", "abc "]
"aa".unpack('b8B8')                 #=> ["10000110", "01100001"]
"aaa".unpack('h2H2c')               #=> ["16", "61", 97]
"\xfe\xff\xfe\xff".unpack('sS')     #=> [-2, 65534]
"now=20is".unpack('M*')             #=> ["now is"]
"whole".unpack('xax2aX2aX1aX2a')    #=> ["h", "e", "l", "l", "o"]
```