Chapter 27

# Built-in Classes and Modules

This chapter documents the classes and modules built into the standard Ruby language. They are available to every Ruby program automatically; no require is required. This section does not contain the various predefined variables and constants; these are listed starting on page 339.

In the descriptions starting on page 447, we show sample invocations for each method:

**new** String.new( *some_string* ) → *new_string*

This description shows a class method that is called as String.new. The italic parameter indicates that a single string is passed in, and the arrow indicates that another string is returned from the method. Because this return value has a different name than that of the parameter, it represents a different object.

When we illustrate instance methods, we show a sample call with a dummy object name in italics as the receiver:

**each** *str*.each( *sep*=$/ ) {| *record* | *block* } → *str*

The parameter to String#each is shown to have a default value; call each with no parameter, and the value of $/ will be used. This method is an iterator, so the call is followed by a block. String#each returns its receiver, so the receiver's name (*str* in this case) appears again after the arrow.

Some methods have optional parameters. We show these parameters between angle brackets, ⟨ *xxx* ⟩. (Additionally, we use the notation ⟨ *xxx* ⟩$^*$ to indicate zero or more occurrences of *xxx* and use ⟨ *xxx* ⟩$^+$ to indicate one or more occurrences of *xxx*.)

**index** self.index( *str* ⟨ , *offset* ⟩ ) → *pos* or nil

Finally, for methods that can be called in several different forms, we list each form on a separate line.

# Alphabetical Listing

Standard classes are listed alphabetically, followed by the standard modules. Within each, we list the class (or module) methods, followed by its instance methods.

## Summary of Built-in Classes

**Array** (page 447): *Class:* [ ], new, try_convert. *Instance:* &, *, +, –, <<, <=>, ==, [ ], [ ]=, |, assoc, at, clear, combination, collect!, compact, compact!, concat, count, cycle, delete, delete_at, delete_if, each, each_index, empty?, eql?, fetch, fill, find_index, flatten, flatten!, frozen?, index, insert, join, last, length, map!, pack, permutation, pop, product, push, rassoc, reject!, replace, reverse, reverse!, reverse_each, rindex, sample, shift, shuffle, shuffle!, size, slice, slice!, sort!, to_a, to_ary, to_s, transpose, uniq, uniq!, unshift, values_at.

**BasicObject** (page 463): *Instance:* !, ==, !=, equal?, instance_eval, instance_exec, method_missing, __send__.

**Bignum** (page 466): *Instance:* Arithmetic operations, Bit operations, <=>, ==, [ ], abs, div, divmod, eql?, fdiv, magnitude, modulo, remainder, size, to_f, to_s.

**Binding** (page 469): *Instance:* eval.

**Class** (page 470): *Class:* inherited, new. *Instance:* allocate, new, superclass.

**Complex** (page 473): *Class:* polar, rect, rectangular. *Instance:* Arithmetic operations, ==, abs, abs2, angle, arg, conj, conjugate, denominator, eql?, fdiv, imag, imaginary, magnitude, numerator, phase, polar, quo, rect, rectangular, real, real?, to_f, to_i, to_r.

**Dir** (page 478): *Class:* [ ], chdir, chroot, delete, entries, exist?, exists?, foreach, getwd, glob, mkdir, new, open, pwd, rmdir, unlink. *Instance:* close, each, path, pos, pos=, read, rewind, seek, tell.

**Encoding** (page 483): *Class:* aliases, compatible?, default_external, default_external=, default_internal, default_internal=, find, list, locale_charmap, name_list. *Instance:* dummy?, name, names.

**Enumerator** (page 496): *Class:* new. *Instance:* each, each_with_index, each_with_object, next, rewind, with_index, with_object.

**Exception** (page 501): *Class:* exception, new. *Instance:* backtrace, exception, message, set_backtrace, status, success?, to_s.

**FalseClass** (page 504): *Instance:* &, ^, |.

**Fiber** (page 505): *Class:* new, yield. *Instance:* resume.

**File** (page 506): *Class:* absolute_path, atime, basename, blockdev?, chardev?, chmod, chown, ctime, delete, directory?, dirname, executable?, executable_real?, exist?, exists?, expand_path, extname, file?, fnmatch, fnmatch?, ftype, grpowned?, identical?, join, lchmod, lchown, link, lstat, mtime, new, owned?, path, pipe?, readable?, readable_real?, readlink, rename, setgid?, setuid?, size, size?, socket?, split, stat, sticky?, symlink, symlink?, truncate, umask, unlink, utime, world_readable?, world_writable?, writable?, writable_real?, zero?. *Instance:* atime, chmod, chown, ctime, flock, lchmod, lchown, lstat, mtime, path, to_path, truncate.

**File::Stat** (page 518): *Instance:* <=>, atime, blksize, blockdev?, blocks, chardev?, ctime, dev, dev_major, dev_minor, directory?, executable?, executable_real?, file?, ftype, gid, grpowned?, ino, mode, mtime, nlink, owned?, pipe?, rdev, rdev_major, rdev_minor, readable?, readable_real?, setgid?, setuid?, size, size?, socket?, sticky?, symlink?, uid, world_readable?, world_writable?, writable?, writable_real?, zero?.

**Fixnum** (page 525): *Class:* . *Instance:* Arithmetic operations, Bit operations, Comparisons, <=>, [ ], abs, div, even?, divmod, fdiv, magnitude, modulo, odd?, size, succ, to_f, to_s, zero?.

**Float** (page 528): *Instance:* Arithmetic operations, Comparisons, <=>, ==, abs, ceil, divmod, eql?, fdiv, finite?, floor, infinite?, magnitude, modulo, nan?, quo, round, to_f, to_i, to_int, to_r, to_s, truncate, zero?.

**Hash** (page 533): *Class:* [ ], new, try_convert. *Instance:* ==, [ ], [ ]=, assoc, clear, compare_by_identity, compare_by_identity?, default, default=, default_proc, default_proc=, delete, delete_if, each, each_key, each_pair, each_value, empty?, fetch, flatten, has_key?, has_value?, include?, index, invert, key, key?, keys, length, member?, merge, merge!, rassoc, rehash, reject, reject!, replace, select, shift, size, sort, store, to_a, to_hash, to_s, update, value?, values, values_at.

**Integer** (page 543): *Instance:* ceil, chr, denominator, downto, even?, floor, gcd, gcdlcm, integer?, lcm, next, numerator, odd?, ord, pred, round, succ, times, to_i, to_int, to_r, truncate, upto.

**IO** (page 546): *Class:* binread, copy_stream, for_fd, foreach, new, open, pipe, popen, read, readlines, select, sysopen, try_convert. *Instance:* <<, binmode, binmode?, bytes, chars, close, close_on_exec?, close_on_exec=, close_read, close_write, closed?, each, each_byte, each_char, each_line, eof, eof?, external_encoding, fcntl, fileno, flush, fsync, getbyte, getc, gets, internal_encoding, ioctl, isatty, lineno, lineno=, lines, pid, pos, pos=, print, printf, putc, puts, read, readbyte, readchar, readline, readlines, readpartial, read_nonblock, reopen, rewind, seek, set_encoding, stat, sync, sync=, sysread, sysseek, syswrite, tell, to_i, to_io, tty?, ungetbyte, ungetc, write, write_nonblock.

**MatchData** (page 585): *Instance:* [ ], begin, captures, end, length, names, offset, post_match, pre_match, regexp, size, string, to_a, to_s, values_at.

**Method** (page 591): *Instance:* [ ], ==, arity, call, eql?, name, owner, receiver, source_location, to_proc, unbind.

**Module** (page 594): *Class:* constants, nesting, new. *Instance:* <, <=, >, >=, <=>, ===, ancestors, autoload, autoload?, class_eval, class_exec, class_variable_defined?, class_variable_get, class_variable_set, class_variables, const_defined?, const_get, const_missing, const_set, constants, include?, included_modules, instance_method, instance_methods, method_defined?, module_eval, module_exec, name, private_class_method, private_instance_methods, private_method_defined?, protected_instance_methods, protected_method_defined?, public_class_method, public_instance_method, public_instance_methods, public_method_defined?, remove_class_variable. *Private:* alias_method, append_features, attr, attr_accessor, attr_reader, attr_writer, define_method, extend_object, extended, include, included, method_added, method_removed, method_undefined, module_function, private, protected, public, remove_const, remove_method, undef_method.

**Mutex** (page 612): *Instance:* lock, locked?, sleep, synchronize, try_lock, unlock.

**NilClass** (page 613): *Instance:* &, ^, |, nil?, to_a, to_c, to_f, to_i, to_r, to_s.

**Numeric** (page 615): *Instance:* +@, -@, <=>, abs, abs2, angle, arg, ceil, coerce, conj, conjugate, denominator, div, divmod, eql?, fdiv, floor, imag, imaginary, integer?, magnitude, modulo, nonzero?, numerator, phase, polar, quo, real, real?, rect, rectangular, remainder, round, step, to_c, to_int, truncate, zero?.

**Object** (page 622): *Instance:* ===, =~, !~, class, clone, define_singleton_method, display, dup, enum_for, eql?, extend, freeze, frozen?, hash, __id__, initialize_copy, inspect, instance_of?, instance_variable_defined?, instance_variable_get, instance_variable_set, instance_variables, is_a?, kind_of?, method, methods, nil?, object_id, private_methods, protected_methods, public_method, public_methods, public_send, respond_to?, send, singleton_methods, taint, tainted?, tap, to_enum, to_s, trust, untaint, untrust, untrusted?. *Private:* initialize, remove_instance_variable, singleton_method_added, singleton_method_removed, singleton_method_undefined.

**Proc** (page 637): *Class:* new. *Instance:* [ ], ==, ===, arity, call, curry, lambda?, source_location, to_proc, to_s, yield.

**Process::Status** (page 650): *Instance:* ==, &, >>, coredump?, exited?, exitstatus, pid, signaled?, stopped?, success?, stopsig, termsig, to_i, to_s.

**Range** (page 656): *Class:* new. *Instance:* ==, ===, begin, cover?, each, end, eql?, exclude_end?, first, include?, last, max, member?, min, step.

**Rational** (page 660): *Instance:* Arithmetic operations, Comparisons, <=>, ==, ceil, denominator, div, fdiv, floor, numerator, quo, round, to_f, to_i, to_r, truncate.

**Regexp** (page 663): *Class:* compile, escape, last_match, new, quote, try_convert, union. *Instance:* ==, ===, =~, ~, casefold?, encoding, fixed_encoding?, match, named_captures, names, options, source, to_s.

**String** (page 670): *Class:* new, try_convert. *Instance:* %, *, +, <<, <=>, ==, =~, [ ], [ ]=, ascii_only?, bytes, bytesize, capitalize, capitalize!, casecmp, center, chars, chr, clear, chomp, chomp!, chop, chop!, codepoints, concat, count, crypt, delete, delete!, downcase, downcase!, dump, each_byte, each_char, each_codepoint, each_line, empty?, encode, encode!, encoding, end_with?, eql?, force_encoding, getbyte, gsub, gsub!, hex, include?, index, insert, intern, length, lines, ljust, lstrip, lstrip!, match, next, next!, oct, ord, partition, replace, reverse, reverse!, rindex, rjust, rpartition, rstrip, rstrip!, scan, setbyte, size, slice, slice!, split, squeeze, squeeze!, start_with?, strip, strip!, sub, sub!, succ, succ!, sum, swapcase, swapcase!, to_c, to_f, to_i, to_r, to_s, to_str, to_sym, tr, tr!, tr_s, tr_s!, unpack, upcase, upcase!, upto, valid_encoding?.

**Struct** (page 696): *Class:* new, new, [ ], members. *Instance:* ==, [ ], [ ]=, each, each_pair, length, members, size, to_a, values, values_at.

**Struct::Tms** (page 700)

**Symbol** (page 701): *Class:* all_symbols. *Instance:* <=>, ==, =~, [ ], capitalize, casecmp, downcase, empty?, encoding, id2name, inspect, intern, length, match, next, size, slice, succ, swapcase, to_proc, to_s, to_sym, upcase.

**Thread** (page 705): *Class:* abort_on_exception, abort_on_exception=, current, exclusive, exit, fork, kill, list, main, new, pass, start, stop. *Instance:* [ ], [ ]=, abort_on_exception, abort_on_exception=, alive?, exit, group, join, keys, key?, kill, priority, priority=, raise, run, safe_level, status, stop?, terminate, value, wakeup.

**ThreadGroup** (page 712): *Class:* new. *Instance:* add, enclose, enclosed?, list.

**Time** (page 714): *Class:* at, gm, local, mktime, new, now, utc. *Instance:* +, –, <=>, asctime, ctime, day, dst?, getgm, getlocal, getutc, gmt?, gmtime, gmt_offset, gmtoff, hour, isdst, localtime, mday, min, mon, month, nsec, sec, strftime, succ, to_a, to_f, to_i, to_s, tv_nsec, tv_sec, tv_usec, usec, utc, utc?, utc_offset, wday, yday, year, zone.

**TrueClass** (page 723): *Instance:* &, ^, |.

**UnboundMethod** (page 724): *Instance:* arity, bind, name, owner, source_location.

## Summary of Built-in Modules

**Comparable** (page 472): *Instance:* Comparisons, between?.

**Enumerable** (page 487): *Instance:* all?, any?, collect, count, cycle, detect, drop, drop_while, each_cons, each_slice, each_with_index, each_with_object, entries, find, find_all, find_index, first, grep, group_by, include?, inject, map, max, max_by, member?, min, min_by, minmax, minmax_by, none?, one?, partition, reduce, reject, reverse_each, select, sort, sort_by, take, take_while, to_a, zip.

**Errno** (page 500)

**FileTest** (page 524)

**GC** (page 532): *Class:* count, disable, enable, start, stress, stress=. *Instance:* garbage_collect.

**Kernel** (page 564): *Class:* __callee__, __method__, Array, Complex, Float, Integer, Rational, String, ` (backquote), abort, at_exit, autoload, autoload?, binding, block_given?, caller, catch, chomp, chop, eval, exec, exit, exit!, fail, fork, format, gem, gets, global_variables, gsub, iterator?, lambda, load, local_variables, loop, open, p, print, printf, proc, putc, puts, raise, rand, readline, readlines, require, require_relative, select, set_trace_func, sleep, spawn, sprintf, srand, sub, syscall, system, test, throw, trace_var, trap, untrace_var, warn.

**Marshal** (page 583): *Class:* dump, load, restore.

**Math** (page 588): *Class:* acos, acosh, asin, asinh, atan, atanh, atan2, cbrt, cos, cosh, erf, erfc, exp, frexp, gamma, hypot, ldexp, lgamma, log, log10, log2, sin, sinh, sqrt, tan, tanh.

**ObjectSpace** (page 635): *Class:* _id2ref, count_objects, define_finalizer, each_object, garbage_collect, undefine_finalizer.

**Process** (page 641): *Class:* abort, daemon, detach, egid, egid=, euid, euid=, exec, exit, exit!, fork, getpgid, getpgrp, getpriority, getrlimit, gid, gid=, groups, groups=, initgroups, kill, maxgroups, maxgroups=, pid, ppid, setpgid, setpgrp, setpriority, setrlimit, setsid, spawn, times, uid, uid=, wait, waitall, wait2, waitpid, waitpid2.

**Process::GID** (page 648): *Class:* change_privilege, eid, eid=, grant_privilege, re_exchange, re_exchangeable?, rid, sid_available?, switch.

**Process::Sys** (page 653): *Class:* getegid, geteuid, getgid, getuid, issetugid, setegid, seteuid, setgid, setregid, setresgid, setresuid, setreuid, setrgid, setruid, setuid.

**Process::UID** (page 655): *Class:* change_privilege, eid, eid=, grant_privilege, re_exchange, re_exchangeable?, rid, sid_available?, switch.

**Signal** (page 668): *Class:* list, trap.

## Class **Array** < Object

Relies on: each, <=>

Arrays are ordered, integer-indexed collections of any object. Array indexing starts at 0, as in C or Java. A negative index is assumed to be relative to the end of the array; that is, an index of −1 indicates the last element of the array, −2 is the next to last element in the array, and so on.

### Mixes in

**Enumerable:**

`all?, any?, collect, count, cycle, detect, drop, drop_while, each_cons, each_slice, each_with_index, entries, find, find_all, find_index, first, grep, group_by, include?, inject, map, max, max_by, member?, min, min_by, minmax, minmax_by, none?, one?, partition, reduce, reject, select, sort, sort_by, take, take_while, to_a, zip`

### Class methods

**[ ]** Array[ ⟨ obj ⟩$^*$ ] → *an_array*

Returns a new array populated with the given objects. Equivalent to the operator form Array.[](. . . ).

```
Array.[]( 1, 'a', /^A/ )   # =>   [1, "a", /^A/]
Array[ 1, 'a', /^A/ ]      # =>   [1, "a", /^A/]
[ 1, 'a', /^A/ ]           # =>   [1, "a", /^A/]
```

**new**

Array.new → *an_array*
Array.new ( *size*=*0*, *obj*=*nil* ) → *an_array*
Array.new( *array* ) → *an_array*
Array.new( *size* ) {| *i* | *block* } → *an_array*

Returns a new array. In the first form, the new array is empty. In the second it is created with *size* copies of *obj* (that is, *size* references to the same *obj*). The third form creates a copy of the array passed as a parameter (the array is generated by calling to_ary on the parameter). In the last form, an array of the given size is created. Each element in this array is calculated by passing the element's index to the given block and storing the return value.

```
Array.new           # =>   []
Array.new(2)        # =>   [nil, nil]
Array.new(5, "A")   # =>   ["A", "A", "A", "A", "A"]

# only one instance of the default object is created
a = Array.new(2, Hash.new)
a[0]['cat'] = 'feline'
a   # =>   [{"cat"=>"feline"}, {"cat"=>"feline"}]
a[1]['cat'] = 'Felix'
a   # =>   [{"cat"=>"Felix"}, {"cat"=>"Felix"}]
```

```
a = Array.new(2) { Hash.new }  # Multiple instances
a[0]['cat'] = 'feline'
a   # =>   [{"cat"=>"feline"}, {}]

squares = Array.new(5) {|i| i*i}
squares   # =>   [0, 1, 4, 9, 16]

copy = Array.new(squares)      # initialized by copying
squares[5] = 25
squares   # =>   [0, 1, 4, 9, 16, 25]
copy      # =>   [0, 1, 4, 9, 16]
```

**try_convert** Array.try_convert( *obj* ) → *an_array* or nil

1.9

If *obj* is not already an array, attempts to convert it to one by calling its to_ary method. Returns nil if no conversion could be made.

```
class Stooges
  def to_ary
    [ "Larry", "Curly", "Moe" ]
  end
end
Array.try_convert(Stooges.new)   # =>   ["Larry", "Curly", "Moe"]
Array.try_convert("Shemp")       # =>   nil
```

## Instance methods

**&** *enum* & *other_array* → *an_array*

Set Intersection—Returns a new array containing elements common to the two arrays, with no duplicates. The rules for comparing elements are the same as for hash keys. If you need setlike behavior, see the library class Set on page 808.

```
[ 1, 1, 3, 5 ] & [ 1, 2, 3 ]   # =>   [1, 3]
```

***** *enum* * *int* → *an_array*
*enum* * *str* → *a_string*

Repetition—With an argument that responds to to_str, equivalent to *enum*.join(*str*). Otherwise, returns a new array built by concatenating *int* copies of *enum*.

```
[ 1, 2, 3 ] * 3      # =>   [1, 2, 3, 1, 2, 3, 1, 2, 3]
[ 1, 2, 3 ] * "--"   # =>   "1--2--3"
```

**+** *enum* + *other_array* → *an_array*

Concatenation—Returns a new array built by concatenating the two arrays together to produce a third array.

```
[ 1, 2, 3 ] + [ 4, 5 ]   # =>   [1, 2, 3, 4, 5]
```

**–** *enum* - *other_array* → *an_array*

Array Difference—Returns a new array that is a copy of the original array, removing any items that also appear in *other_array*. If you need setlike behavior, see the library class Set on page 808.

```
[ 1, 1, 2, 2, 3, 3, 4, 5 ] - [ 1, 2, 4 ]   # =>   [3, 3, 5]
```

**<<** *enum* << *obj* → *enum*

Append—Pushes the given object on to the end of this array. This expression returns the array itself, so several appends may be chained together. See also Array#push.

```
[ 1, 2 ] << "c" << "d" << [ 3, 4 ]   # =>   [1, 2, "c", "d", [3, 4]]
```

**<=>** *enum* <=> *other_array* → $-1, 0, +1$

Comparison—Returns an integer $-1$, $0$, or $+1$ if this array is less than, equal to, or greater than *other_array*. Each object in each array is compared (using <=>). If any value isn't equal, then that inequality is the return value. If all the values found are equal, then the return is based on a comparison of the array lengths. Thus, two arrays are "equal" according to Array#<=> if and only if they have the same length and the value of each element is equal to the value of the corresponding element in the other array.

```
[ "a", "a", "c" ]    <=> [ "a", "b", "c" ]   # =>   -1
[ 1, 2, 3, 4, 5, 6 ] <=> [ 1, 2 ]            # =>   1
```

**==** *enum* == *obj* → true or false

Equality—Two arrays are equal if they contain the same number of elements and if each element is equal to (according to Object#==) the corresponding element in the other array. If *obj* is not an array, attempt to convert it using to_ary and return *obj*==*enum*.

```
[ "a", "c" ]    == [ "a", "c", 7 ]     # =>   false
[ "a", "c", 7 ] == [ "a", "c", 7 ]     # =>   true
[ "a", "c", 7 ] == [ "a", "d", "f" ]   # =>   false
```

**[ ]** *enum*[*int*] → *obj* or nil
*enum*[*start*, *length*] → *an_array* or nil
*enum*[*range*] → *an_array* or nil

Element Reference—Returns the element at index *int*, returns a subarray starting at index *start* and continuing for *length* elements, or returns a subarray specified by *range*. Negative indices count backward from the end of the array ($-1$ is the last element). Returns nil if the index of the first element selected is greater than the array size. If the start index equals the array size and a *length* or *range* parameter is given, an empty array is returned. Equivalent to Array#slice.

```
a = [ "a", "b", "c", "d", "e" ]
a[2] +  a[0] + a[1]    # =>   "cab"
a[6]                   # =>   nil
a[1, 2]                # =>   ["b", "c"]
a[1..3]                # =>   ["b", "c", "d"]
a[4..7]                # =>   ["e"]
a[6..10]               # =>   nil
a[-3, 3]               # =>   ["c", "d", "e"]
```

```
# special cases
a[5]       # =>   nil
a[5, 1]    # =>   []
a[5..10]   # =>   []
```

**[ ]=**

*enum*[*int*] = *obj* → *obj*
*enum*[*start*, *length*] = *obj* → *obj*
*enum*[*range*] = *obj* → *obj*

Element Assignment—Sets the element at index *int*, replaces a subarray starting at index *start* and continuing for *length* elements, or replaces a subarray specified by *range*. If *int* is greater than the current capacity of the array, the array grows automatically. A negative *int* will count backward from the end of the array. Inserts elements if *length* is zero. If *obj* is an array, the form with the single index will insert that array into *enum*, and the forms with a length or with a range will replace the given elements in *enum* with the array contents. An IndexError is raised if a negative index points past the beginning of the array. (Prior to 1.9 Ruby 1.9, assigning nil with the second and third forms of element assignment could delete the corresponding array elements; now it simply assigns nil to them.) See also Array#push and Array#unshift.

```
a = Array.new                     # =>   []
a[4] = "4";                   a   # =>   [nil, nil, nil, nil, "4"]
a[0] = [ 1, 2, 3 ];           a   # =>   [[1, 2, 3], nil, nil, nil, "4"]
a[0, 3] = [ 'a', 'b', 'c' ]; a   # =>   ["a", "b", "c", nil, "4"]
a[1..2] = [ 1, 2 ];           a   # =>   ["a", 1, 2, nil, "4"]
a[0, 2] = "?";                a   # =>   ["?", 2, nil, "4"]
a[0..2] = "A", "B", "C";      a   # =>   ["A", "B", "C", "4"]
a[-1]   = "Z";                a   # =>   ["A", "B", "C", "Z"]
a[1..-1] = nil;               a   # =>   ["A", nil]
```

**|**

*enum* | *other_array* → *an_array*

Set Union—Returns a new array by joining this array with *other_array*, removing duplicates. The rules for comparing elements are the same as for hash keys. If you need setlike behavior, see the library class Set on page 808.

```
[ "a", "b", "c" ] | [ "c", "d", "a" ]   # =>   ["a", "b", "c", "d"]
```

**assoc**

*enum*.assoc( *obj* ) → *an_array* or nil

Searches through an array whose elements are also arrays comparing *obj* with the first element of each contained array using *obj*.== . Returns the first contained array that matches (that is, the first *asso*ciated array) or nil if no match is found. See also Array#rassoc.

```
s1 = [ "colors", "red", "blue", "green" ]
s2 = [ "letters", "a", "b", "c" ]
s3 = "foo"
a  = [ s1, s2, s3 ]
a.assoc("letters")   # =>   ["letters", "a", "b", "c"]
a.assoc("foo")       # =>   nil
```

**at** *enum*.at( *int* ) → *obj* or nil

Returns the element at index *int*. A negative index counts from the end of *enum*. Returns nil if the index is out of range. See also Array#[].

```
a = [ "a", "b", "c", "d", "e" ]
a.at(0)    # =>   "a"
a.at(-1)   # =>   "e"
```

**clear** *enum*.clear → *enum*

Removes all elements from *enum*.

```
a = [ "a", "b", "c", "d", "e" ]
a.clear   # =>   []
```

**combination** *enum*.combination( *size* ) → *enumerator*
*enum*.combination( *size* ) {| *array* | *block* } → *enum*

1.9 Constructs all combinations of the elements of *enum* of length *size*. If called with a block, passes each combination to that block; otherwise, returns an enumerator object. An empty result is generated if no combinations of the given length exist. See also Array#permutation.

```
a = [ "a", "b", "c" ]
a.combination(1).to_a   # =>   [["a"], ["b"], ["c"]]
a.combination(2).to_a   # =>   [["a", "b"], ["a", "c"], ["b", "c"]]
a.combination(3).to_a   # =>   [["a", "b", "c"]]
a.combination(4).to_a   # =>   []
```

**collect!** *enum*.collect! {| *obj* | *block* } → *enum*

Invokes *block* once for each element of *enum*, replacing the element with the value returned by *block*. See also Enumerable#collect.

```
a = [ "a", "b", "c", "d" ]
a.collect! {|x| x + "!" }   # =>   ["a!", "b!", "c!", "d!"]
a                           # =>   ["a!", "b!", "c!", "d!"]
```

**compact** *enum*.compact → *an_array*

Returns a copy of *enum* with all nil elements removed.

```
[ "a", nil, "b", nil, "c", nil ].compact   # =>   ["a", "b", "c"]
```

**compact!** *enum*.compact! → *enum* or nil

Removes nil elements from *enum*. Returns nil if no changes were made.

```
[ "a", nil, "b", nil, "c" ].compact!   # =>   ["a", "b", "c"]
[ "a", "b", "c" ].compact!             # =>   nil
```

**concat** *enum*.concat( *other_array* ) → *enum*

Appends the elements in *other_array* to *enum*.

```
[ "a", "b" ].concat( ["c", "d"] )   # =>   ["a", "b", "c", "d"]
```

**count** *enum*.count( *obj* ) → *int*
*enum*.count {| *obj* | *block* } → *int*

1.9 Returns the count of objects in *enum* that equal *obj* or for which the block returns a true value. Returns an Enumerator if neither an argument nor a block is given (which seems strange...). Shadows the corresponding method in Enumerable.

```
[1, 2, 3, 4].count(3)              # =>   1
[1, 2, 3, 4].count {|obj| obj > 2 }  # =>   2
```

**cycle** *enum*.cycle {| *obj* | *block* } → nil or *enumerator*
*enum*.cycle( *times* ) {| *obj* | *block* } → nil or *enumerator*

1.9 Returns nil if *enum* has no elements; otherwise, passes the elements, one at a time to the block. When it reaches the end, it repeats. The number of times it repeats is set by the parameter. If the parameter is missing, cycles forever. Equivalent to *enum*.to_a.cycle. See Array#cycle. Returns an Enumerator object if no block is given.

```
[1,2,3].cycle(3)       # =>   #<Enumerator:0x0a4fec>
[1,2,3].cycle(3).to_a  # =>   [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
columns = [ 1, 2, 3 ]
data = %w{ a b c d e f g h }
columns.cycle do |column_number|
  print data.shift, "\t"
  break if data.empty?
  puts if column_number == columns.last
end
puts
```

*produces:*

```
a b c
d e f
g h
```

**delete** *enum*.delete( *obj* ) → *obj* or nil
*enum*.delete( *obj* ) { *block* } → *obj* or nil

Deletes items from *enum* that are equal to *obj*. If the item is not found, returns nil. If the optional code block is given, returns the result of *block* if the item is not found.

```
a = [ "a", "b", "b", "b", "c" ]
a.delete("b")                   # =>   "b"
a                               # =>   ["a", "c"]
a.delete("z")                   # =>   nil
a.delete("z") { "not found" }   # =>   "not found"
```

**delete_at** *enum*.delete_at( *index* ) → *obj* or nil

Deletes the element at the specified index, returning that element or nil if the index is out of range. See also Array#slice!.

```
a = %w( ant bat cat dog )
a.delete_at(2)    # =>   "cat"
a                 # =>   ["ant", "bat", "dog"]
a.delete_at(99)   # =>   nil
```

### delete_if

*enum*.delete_if {| *item* | *block* } → *enum*

Deletes every element of *enum* for which *block* evaluates to true.

```
a = [ "a", "b", "c" ]
a.delete_if {|x| x >= "b" }   # =>   ["a"]
```

### each

*enum*.each {| *item* | *block* } → *enum*

Calls *block* once for each element in *enum*, passing that element as a parameter.

```
a = [ "a", "b", "c" ]
a.each {|x| print x, " -- " }
```

*produces:*

```
a -- b -- c --
```

### each_index

*enum*.each_index {| *index* | *block* } → *enum*

Same as Array#each but passes the index of the element instead of the element itself.

```
a = [ "a", "b", "c" ]
a.each_index {|x| print x, " -- " }
```

*produces:*

```
0 -- 1 -- 2 --
```

### empty?

*enum*.empty? → true or false

Returns true if *enum* array contains no elements.

```
[].empty?            # =>   true
[ 1, 2, 3 ].empty?   # =>   false
```

### eql?

*enum*.eql?( *other* ) → true or false

Returns true if *enum* and *other* are the same object or if *other* is an object of class Array with the same length and content as *enum*. Elements in the arrays are compared using Object#eql?. See also Array#<=>.

```
[ "a", "b", "c" ].eql?(["a", "b", "c"])   # =>   true
[ "a", "b", "c" ].eql?(["a", "b"])        # =>   false
[ "a", "b", "c" ].eql?(["b", "c", "d"])   # =>   false
```

### fetch

*enum*.fetch( *index* ) → *obj*
*enum*.fetch( *index*, *default* ) → *obj*
*enum*.fetch( *index* ) {| *i* | *block* } → *obj*

Tries to return the element at position *index*. If the index lies outside the array, the first form throws an IndexError exception, the second form returns *default*, and the third form returns the value of invoking the block, passing in the index. Negative values of *index* count from the end of the array.

```
a = [ 11, 22, 33, 44 ]
a.fetch(1)              # =>   22
a.fetch(-1)             # =>   44
a.fetch(-1, 'cat')      # =>   44
a.fetch(4, 'cat')       # =>   "cat"
a.fetch(4) {|i| i*i }   # =>   16
```

### fill

*enum*.fill( *obj* ) → *enum*
*enum*.fill( *obj*, *start* ⟨ , *length* ⟩ ) → *enum*
*enum*.fill( *obj*, *range* ) → *enum*
*enum*.fill {| *i* | *block* } → *enum*
*enum*.fill( *start* ⟨ , *length* ⟩ ) {| *i* | *block* } → *enum*
*enum*.fill( *range* ) {| *i* | *block* } → *enum*

The first three forms set the selected elements of *enum* (which may be the entire array) to *obj*. A *start* of nil is equivalent to zero. A *length* of nil is equivalent to *enum*.length. The last three forms fill the array with the value of the block. The block is passed the absolute index of each element to be filled.

```
a = [ "a", "b", "c", "d" ]
a.fill("x")             # =>   ["x", "x", "x", "x"]
a.fill("z", 2, 2)       # =>   ["x", "x", "z", "z"]
a.fill("y", 0..1)       # =>   ["y", "y", "z", "z"]
a.fill {|i| i*i}        # =>   [0, 1, 4, 9]
a.fill(-3) {|i| i+100}  # =>   [0, 101, 102, 103]
```

### find_index

*enum*.find_index( *obj* ) → *int* or nil
*enum*.find_index {| *item* | *block* } → *int* or nil

1.9 Returns the index of the first object in *enum* that is == to *obj* or for which the block returns a true value. Returns nil if no match is found. See also Enumerable#select and Array#rindex.

```
a = [ "a", "b", "c", "b" ]
a.find_index("b")                 # =>   1
a.find_index("z")                 # =>   nil
a.find_index {|item| item > "a"}  # =>   1
```

### flatten

*enum*.flatten( *level* = -1 ) → *an_array*

1.9 Returns a new array that is a one-dimensional flattening of this array (recursively). That is, for every element that is an array, extracts its elements into the new array. The level parameter controls how deeply the flattening occurs. If less than zero, all subarrays are expanded. If zero, no flattening takes place. If greater than zero, only that depth of subarray is expanded.

```
s = [ 1, 2, 3 ]           # =>  [1, 2, 3]
t = [ 4, 5, 6, [7, 8] ]   # =>  [4, 5, 6, [7, 8]]
a = [ s, t, 9, 10 ]       # =>  [[1, 2, 3], [4, 5, 6, [7, 8]], 9, 10]
a.flatten(0)              # =>  [[1, 2, 3], [4, 5, 6, [7, 8]], 9, 10]
a.flatten                 # =>  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
a.flatten(1)              # =>  [1, 2, 3, 4, 5, 6, [7, 8], 9, 10]
a.flatten(2)              # =>  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

**flatten!** *enum*.flatten!( *level* = -1 ) → *enum* or nil

1.9 Same as Array#flatten but modifies the receiver in place. Returns nil if no modifications were made (i.e., *enum* contains no subarrays).

```
a = [ 1, 2, [3, [4, 5] ] ]
a.flatten!   # =>  [1, 2, 3, 4, 5]
a.flatten!   # =>  nil
a            # =>  [1, 2, 3, 4, 5]
```

**frozen?** *enum*.frozen?! → true or false

1.9 Returns true if *enum* is frozen or if it is in the middle of being sorted.

**index** *enum*.index( *obj* ) → *int* or nil
*enum*.index {| *item* | *block* } → *int* or nil

1.9 Synonym for Array#find_index.

**insert** *enum*.insert( *index*, ⟨ *obj* ⟩$^+$ ) → *enum*

If *index* is not negative, inserts the given values before the element with the given index. If *index* is negative, adds the values after the element with the given index (counting from the end).

```
a = %w{ a b c d }
a.insert(2, 99)         # =>  ["a", "b", 99, "c", "d"]
a.insert(-2, 1, 2, 3)   # =>  ["a", "b", 99, "c", 1, 2, 3, "d"]
a.insert(-1, "e")       # =>  ["a", "b", 99, "c", 1, 2, 3, "d", "e"]
```

**join** *enum*.join( *separator*=$, ) → *str*

Returns a string created by converting each each element of the array to a string and concatenating them, separated each by *separator*.

```
[ "a", "b", "c" ].join        # =>  "abc"
[ "a", "b", "c" ].join("-")   # =>  "a-b-c"
```

**last** *enum*.last → *obj* or nil
*enum*.last( *count* ) → *an_array*

Returns the last element, or last *count* elements, of *enum*. If the array is empty, the first form returns nil, and the second returns an empty array. (first is defined by Enumerable.)

```
[ "w", "x", "y", "z" ].last      # =>  "z"
[ "w", "x", "y", "z" ].last(1)   # =>  ["z"]
[ "w", "x", "y", "z" ].last(3)   # =>  ["x", "y", "z"]
```

Table 27.1. Template Characters for Array#pack

| | Directive | Meaning |
|---|---|---|
| | @ | Move to absolute position |
| 1.9 | A | Sequence of bytes (space padded, count is width) |
| 1.9 | a | Sequence of bytes (null padded, count is width) |
| | B | Bit string (descending bit order) |
| | b | Bit string (ascending bit order) |
| 1.9 | C | Unsigned byte |
| 1.9 | c | Byte |
| | D, d | Double-precision float, native format |
| | E | Double-precision float, little-endian byte order |
| | e | Single-precision float, little-endian byte order |
| | F, f | Single-precision float, native format |
| | G | Double-precision float, network (big-endian) byte order |
| | g | Single-precision float, network (big-endian) byte order |
| | H | Hex string (high nibble first) |
| | h | Hex string (low nibble first) |
| | I | Unsigned integer |
| | i | Integer |
| | L | Unsigned long |
| | l | Long |
| | M | Quoted printable, MIME encoding (see RFC2045) |
| | m | Base64-encoded string; by default adds linefeeds every 60 characters; "m0" suppresses linefeeds |
| | N | Long, network (big-endian) byte order |
| | n | Short, network (big-endian) byte order |
| | P | Pointer to a structure (fixed-length string) |
| | p | Pointer to a null-terminated string |
| | Q, q | 64-bit number |
| | S | Unsigned short |
| | s | Short |
| | U | UTF-8 |
| | u | UU-encoded string |
| | V | Long, little-endian byte order |
| | v | Short, little-endian byte order |
| | w | BER-compressed integer[1] |
| | X | Back up a byte |
| | x | Null byte |
| | Z | Same as “a,” except a null byte is appended if the * modifier is given |

[1] The octets of a BER-compressed integer represent an unsigned integer in base 128, most significant digit first, with as few digits as possible. Bit eight (the high bit) is set on each byte except the last (*Self-Describing Binary Data Representation*, MacLeod).

**length** *enum*.length → *int*

Returns the number of elements in *enum*.

```
[ 1, nil, 3, nil, 5 ].length   # =>   5
```

**map!** *enum*.map! {| *obj* | *block* } → *enum*

Synonym for Array#collect!.

**pack** *enum*.pack ( *template* ) → *binary_string*

Packs the contents of *enum* into a binary sequence according to the directives in *template* (see Table 27.1 on the previous page). Directives A, a, and Z may be followed by a count, which gives the width of the resulting field. The remaining directives also may take a count, indicating the number of array elements to convert. If the count is an asterisk (*), all remaining array elements will be converted. Any of the directives sSiIlL may be followed by an 1.9 underscore (_) or bang (!) to use the underlying platform's native size for the specified type; otherwise, they use a platform-independent size. Spaces are ignored in the template string. Comments starting with # to the next newline or end of string are also ignored. See also String#unpack on page 693.

```
a = [ "a", "b", "c" ]
n = [ 65, 66, 67 ]
a.pack("A3A3A3")   # =>   "a␣␣b␣␣c␣␣"
a.pack("a3a3a3")   # =>   "a\x00\x00b\x00\x00c\x00\x00"
n.pack("ccc")      # =>   "ABC"
```

**permutation** *enum*.permutation( *size* ) → *enumerator*
*enum*.permutation( *size* ) {| *array* | *block* } → *enum*

1.9 Constructs all permutations of the elements of *enum* of length *size*. If called with a block, passes each permutation to that block; otherwise, returns an enumerator object. An empty result is generated if no permutations of the given length exist. See also Array#combination.

```
words = {}
File.readlines("/usr/share/dict/words").map(&:chomp).each do |word|
  words[word.downcase] = 1
end
%w{ c a m e l }.permutation(5) do |letters|
  anagram = letters.join
  puts anagram if words[anagram]
end
```

*produces:*

```
camel
clame
cleam
macle
```

**pop** *enum*.pop( ⟨ n ⟩* ) → *obj* or nil

1.9

Removes the last element (or the last *n* elements) from *enum*. Returns whatever is removed or nil if the array is empty.

```
a = %w{ f r a b j o u s }
a.pop      # =>   "s"
a          # =>   ["f", "r", "a", "b", "j", "o", "u"]
a.pop(3)   # =>   ["j", "o", "u"]
a          # =>   ["f", "r", "a", "b"]
```

**product** *enum*.product( ⟨ *arrays* ⟩* ) → result_array

1.9

Generates all combinations of selecting an element each from *enum* and from any arrays passed as arguments. The number of elements in the result is the product of the lengths of *enum* and the lengths of the arguments (so if any of these arrays is empty, the result will be an empty array). Each element in the result is an array containing $n + 1$ elements, where $n$ is the number of arguments.

```
[1, 2].product([3, 4])        # =>   [[1, 3], [1, 4], [2, 3], [2, 4]]
[1, 2].product([3, 4], [5])   # =>   [[1, 3, 5], [1, 4, 5], [2, 3, 5], [2,
                                     4, 5]]
[1, 2].product                # =>   [[1], [2]]
```

**push** *enum*.push( ⟨ *obj* ⟩* ) → *enum*

Appends the given argument(s) to *enum*.

```
a = [ "a", "b", "c" ]
a.push("d", "e", "f")   # =>   ["a", "b", "c", "d", "e", "f"]
```

**rassoc** *enum*.rassoc( *key* ) → *an_array* or nil

Searches through the array whose elements are also arrays. Compares *key* with the second element of each contained array using ==. Returns the first contained array that matches. See also Array#assoc.

```
a = [ [ 1, "one"], [2, "two"], [3, "three"], ["ii", "two"] ]
a.rassoc("two")    # =>   [2, "two"]
a.rassoc("four")   # =>   nil
```

**reject!** *enum*.reject! { *block* } item → *enum* or nil

Equivalent to Array#delete_if but returns nil if no changes were made. Also see Enumerable#reject.

**replace** *enum*.replace( *other_array* ) → *enum*

Replaces the contents of *enum* with the contents of *other_array*, truncating or expanding if necessary.

```
a = [ "a", "b", "c", "d", "e" ]
a.replace([ "x", "y", "z" ])   # =>   ["x", "y", "z"]
a                              # =>   ["x", "y", "z"]
```

### reverse
*enum*.reverse → *an_array*

Returns a new array using *enum*'s elements in reverse order.

```
[ "a", "b", "c" ].reverse   # =>   ["c", "b", "a"]
[ 1 ].reverse               # =>   [1]
```

### reverse!
*enum*.reverse! → *enum*

Reverses *enum* in place.

```
a = [ "a", "b", "c" ]
a.reverse!       # =>   ["c", "b", "a"]
a                # =>   ["c", "b", "a"]
[ 1 ].reverse!   # =>   [1]
```

### reverse_each
*enum*.reverse_each ⟨ {| *item* | *block* } ⟩ → *enum*

Same as Array#each but traverses *enum* in reverse order.

```
a = [ "a", "b", "c" ]
a.reverse_each {|x| print x, " " }
```

*produces:*

```
c b a
```

### rindex
*enum*.rindex( *obj* ) → *int* or nil
*enum*.rindex {| *item* | *block* } → *int* or nil

1.9 Returns the index of the last object in *enum* that is == to *obj* or for which the block returns a true value. Returns nil if no match is found. See also Enumerable#select and Array#index.

```
a = [ "a", "b", "e", "b", "d" ]
a.rindex("b")                      # =>   3
a.rindex("z")                      # =>   nil
a.rindex {|item| item =~ /[aeiou]/}   # =>   2
```

### sample
*enum*.sample( *n*=1 ) → *an_array* or nil

1.9 Returns min(*n*, *enum*.size) random elements from *enum*or nil if *enum* is empty and no argument is given.

```
a = [ "a", "b", "c", "d" ]
a.sample      # =>   "c"
a.sample(3)   # =>   ["c", "d", "a"]
a.sample(6)   # =>   ["c", "a", "d", "b"]
b = []
b.sample      # =>   nil
```

### shift
*enum*.shift( n = 1 ) → *obj* or nil

1.9 Returns the first *n* elements (or the first element with no argument) of *enum* and removes it (shifting all other elements down by one). Returns nil if the array is empty.

```
args = [ "-m", "-q", "-v", "filename" ]
args.shift      # =>   "-m"
args.shift(2)   # =>   ["-q", "-v"]
args            # =>   ["filename"]
```

**shuffle** *enum*.shuffle → *an_array*

1.9

Returns an array containing the elements of *enum* in random order.

```
[ 1, 2, 3, 4, 5 ].shuffle   # =>   [5, 3, 4, 1, 2]
```

**shuffle!** *enum*.shuffle! → *enum*

1.9

Randomizes the order of the elements of *enum*.

**size** *enum*.size → *int*

Synonym for Array#length.

**slice**

*enum*.slice( *int* ) → *obj*
*enum*.slice( *start*, *length* ) → *an_array*
*enum*.slice( *range* ) → *an_array*

Synonym for Array#[ ].

```
a = [ "a", "b", "c", "d", "e" ]
a.slice(2) + a.slice(0) + a.slice(1)   # =>   "cab"
a.slice(6)                             # =>   nil
a.slice(1, 2)                          # =>   ["b", "c"]
a.slice(1..3)                          # =>   ["b", "c", "d"]
a.slice(4..7)                          # =>   ["e"]
a.slice(6..10)                         # =>   nil
a.slice(-3, 3)                         # =>   ["c", "d", "e"]
# special cases
a.slice(5)                             # =>   nil
a.slice(5, 1)                          # =>   []
a.slice(5..10)                         # =>   []
```

**slice!**

*enum*.slice!( *int* ) → *obj* or nil
*enum*.slice!( *start*, *length* ) → *an_array* or nil
*enum*.slice!( *range* ) → *an_array* or nil

Deletes the element(s) given by an index (optionally with a length) or by a range. Returns the deleted object, subarray, or nil if the index is out of range.

```
a = [ "a", "b", "c" ]
a.slice!(1)     # =>   "b"
a               # =>   ["a", "c"]
a.slice!(-1)    # =>   "c"
a               # =>   ["a"]
a.slice!(100)   # =>   nil
a               # =>   ["a"]
```

**sort!**

*enum*.sort! → *enum*
*enum*.sort! {| *a,b* | *block* } → *enum*

Sorts *enum* in place (see Enumerable#sort). *enum* is effectively frozen while a sort is in progress.

```
a = [ "d", "a", "e", "c", "b" ]
a.sort!   # =>  ["a", "b", "c", "d", "e"]
a         # =>  ["a", "b", "c", "d", "e"]
```

**to_a**

*enum*.to_a → *enum*
*array_subclass*.to_a → *array*

If *enum* is an array, returns *enum*. If *enum* is a subclass of Array, invokes to_ary and uses the result to create a new array object.

**to_ary**

*enum*.to_ary → *enum*

Returns *enum*.

**to_s**
1.9

*enum*.to_s → *str*

Returns a string representation of *enum*. (Prior to Ruby 1.9, this representation was the same as *enum*.join. Now it is the array as a literal.)

```
[ 1, 3, 5, 7, 9 ].to_s   # =>  "[1, 3, 5, 7, 9]"
```

**transpose**

*enum*.transpose → *an_array*

Assumes that *enum* is an array of arrays and transposes the rows and columns.

```
a = [[1,2], [3,4], [5,6]]
a.transpose   # =>  [[1, 3, 5], [2, 4, 6]]
```

**uniq**

*enum*.uniq → *an_array*

Returns a new array by removing duplicate values in *enum*, where duplicates are detected by comparing using eql? and hash.

```
a = [ "a", "a", "b", "b", "c" ]
a.uniq   # =>  ["a", "b", "c"]
```

**uniq!**

*enum*.uniq! → *enum* or nil

Same as Array#uniq but modifies the receiver in place. Returns nil if no changes are made (that is, no duplicates are found).

```
a = [ "a", "a", "b", "b", "c" ]
a.uniq!   # =>  ["a", "b", "c"]
b = [ "a", "b", "c" ]
b.uniq!   # =>  nil
```

**unshift**

*enum*.unshift( ⟨ *obj* ⟩$^+$ ) → *enum*

Prepends object(s) to *enum*.

```
a = [ "b", "c", "d" ]
a.unshift("a")    # =>   ["a", "b", "c", "d"]
a.unshift(1, 2)   # =>   [1, 2, "a", "b", "c", "d"]
```

**values_at** *enum*.values_at( ⟨ *selector* ⟩$^*$ ) → *an_array*

Returns an array containing the elements in *enum* corresponding to the given selector(s). The selectors may be either integer indices or ranges.

```
a = %w{ a b c d e f }
a.values_at(1, 3, 5)          # =>   ["b", "d", "f"]
a.values_at(1, 3, 5, 7)       # =>   ["b", "d", "f", nil]
a.values_at(-1, -3, -5, -7)   # =>   ["f", "d", "b", nil]
a.values_at(1..3, 2...5)      # =>   ["b", "c", "d", "c", "d", "e"]
```