Home

Pages Classes Methods

Search

Table of Contents

What's Here
Methods for Querying
Methods for Comparing
Methods for Converting

Show/hide navigation

Parent

Object

Included Modules

Comparable

Methods

::all_symbols
#<=>
#==
#===
#=~
#[]
#capitalize
#casecmp
#casecmp?
#downcase
#empty?
#encoding
#end_with?
#id2name
#inspect
#intern
#length
#match
#match?
#name
#next
#size
#slice
#start_with?
#succ
#swapcase
#to_proc
#to_s
#to_sym
#upcase

class Symbol

Symbol objects represent named identifiers inside the Ruby interpreter.

You can create a Symbol object explicitly with:

- A symbol literal.

The same Symbol object will be created for a given name or string for the duration of a program's execution, regardless of the context or meaning of that name. Thus if `Fred` is a constant in one context, a method in another, and a class in a third, the Symbol `:Fred` will be the same object in all three contexts.

```
module One
  class Fred
  end
  $f1 = :Fred
end
module Two
  Fred = 1
  $f2 = :Fred
end
def Fred()
end
$f3 = :Fred
$f1.object_id   #=> 2514190
$f2.object_id   #=> 2514190
$f3.object_id   #=> 2514190
```

Constant, method, and variable names are returned as symbols:

```
module One
  Two = 2
  def three; 3 end
  @four = 4
  @@five = 5
  $six = 6
end
```

```
seven = 7

One.constants
# => [:Two]
One.instance_methods(true)
# => [:three]
One.instance_variables
# => [:@four]
One.class_variables
# => [:@@five]
global_variables.grep(/six/)
# => [:$six]
local_variables
# => [:seven]
```

Symbol objects are different from `String` objects in that Symbol objects represent identifiers, while `String` objects represent text or data.

What's Here

First, what's elsewhere. Class Symbol:

- Inherits from class Object.
- Includes module Comparable.

Here, class Symbol provides methods that are useful for:

- Querying
- Comparing
- Converting

Methods for Querying

- `::all_symbols` : Returns an array of the symbols currently in Ruby's symbol table.
- #=~: Returns the index of the first substring in symbol that matches a given `Regexp` or other object; returns `nil` if no match is found.
- `[]`, `slice` : Returns a substring of symbol determined by a given index, start/length, or range, or string.
- `empty?` : Returns `true` if `self.length` is zero; `false` otherwise.
- `encoding` : Returns the `Encoding` object that represents the encoding of symbol.
- `end_with?` : Returns `true` if symbol ends with any of the given strings.
- `match` : Returns a `MatchData` object if symbol matches a given `Regexp` ; `nil` otherwise.

- match? : Returns `true` if symbol matches a given Regexp; `false` otherwise.
- length, size : Returns the number of characters in symbol.
- start_with? : Returns `true` if symbol starts with any of the given strings.

Methods for Comparing

- #<=>: Returns -1, 0, or 1 as a given symbol is smaller than, equal to, or larger than symbol.
- ==, ===: Returns `true` if a given symbol has the same content and encoding.
- casecmp : Ignoring case, returns -1, 0, or 1 as a given symbol is smaller than, equal to, or larger than symbol.
- casecmp? : Returns `true` if symbol is equal to a given symbol after Unicode case folding; `false` otherwise.

Methods for Converting

- capitalize : Returns symbol with the first character upcased and all other characters downcased.
- downcase : Returns symbol with all characters downcased.
- inspect : Returns the string representation of `self` as a symbol literal.
- name : Returns the frozen string corresponding to symbol.
- succ, next : Returns the symbol that is the successor to symbol.
- swapcase : Returns symbol with all upcase characters downcased and all downcase characters upcased.
- to_proc : Returns a Proc object which responds to the method named by symbol.
- to_s, id2name : Returns the string corresponding to `self`.
- to_sym, intern : Returns `self`.
- upcase : Returns symbol with all characters upcased.

Public Class Methods

`all_symbols → array_of_symbols`

Returns an array of all symbols currently in Ruby's symbol table:

```
Symbol.all_symbols.size    # => 9334
Symbol.all_symbols.take(3) # => [:!, :"\"", :"#"]
```

Public Instance Methods

symbol <=> object → -1, 0, +1, or nil

If `object` is a symbol, returns the equivalent of `symbol.to_s <=> object.to_s`:

```
:bar <=> :foo # => -1
:foo <=> :foo # => 0
:foo <=> :bar # => 1
```

Otherwise, returns `nil`:

```
:foo <=> 'bar' # => nil
```

Related: String#<=>.

symbol == object → true or false

Returns `true` if `object` is the same object as `self`, `false` otherwise.

Also aliased as: ===

===(p1)

Returns `true` if `object` is the same object as `self`, `false` otherwise.

Alias for: ==

symbol =~ object → integer or nil

Equivalent to `symbol.to_s =~ object`, including possible updates to global variables; see String#=~.

symbol[index] → string or nil
symbol[start, length] → string or nil
symbol[range] → string or nil
symbol[regexp, capture = 0] → string or nil
symbol[substring] → string or nil

Equivalent to `symbol.to_s[]`; see `String#[]`.

Also aliased as: slice

capitalize(*options) → symbol

Equivalent to `sym.to_s.capitalize.to_sym`.

See `String#capitalize`.

casecmp(object) → -1, 0, 1, or nil

Like Symbol#<=>, but case-insensitive; equivalent to `self.to_s.casecmp(object.to_s)`:

```
lower = :abc
upper = :ABC
upper.casecmp(lower) # => 0
lower.casecmp(lower) # => 0
lower.casecmp(upper) # => 0
```

Returns nil if `self` and `object` have incompatible encodings, or if `object` is not a symbol:

```
sym = 'äöü'.encode("ISO-8859-1").to_sym
other_sym = 'ÄÖÜ'
sym.casecmp(other_sym) # => nil
:foo.casecmp(2)        # => nil
```

Unlike `Symbol#casecmp?`, case-insensitivity does not work for characters outside of 'A'..'Z' and 'a'..'z':

```
lower = :äöü
upper = :ÄÖÜ
upper.casecmp(lower) # => -1
lower.casecmp(lower) # => 0
lower.casecmp(upper) # => 1
```

Related: `Symbol#casecmp?`, `String#casecmp`.

casecmp?(object) → true, false, or nil

Returns `true` if `self` and `object` are equal after Unicode case folding, otherwise `false`:

```
lower = :abc
upper = :ABC
upper.casecmp?(lower) # => true
```

```
lower.casecmp?(lower) # => true
lower.casecmp?(upper) # => true
```

Returns nil if `self` and `object` have incompatible encodings, or if `object` is not a symbol:

```
sym = 'äöü'.encode("ISO-8859-1").to_sym
other_sym = 'ÄÖÜ'
sym.casecmp?(other_sym) # => nil
:foo.casecmp?(2)        # => nil
```

Unlike Symbol#casecmp, works for characters outside of 'A'..'Z' and 'a'..'z':

```
lower = :äöü
upper = :ÄÖÜ
upper.casecmp?(lower) # => true
lower.casecmp?(lower) # => true
lower.casecmp?(upper) # => true
```

Related: Symbol#casecmp, String#casecmp?.

downcase(*options) → symbol

Equivalent to `sym.to_s.downcase.to_sym`.

See String#downcase.

Related: Symbol#upcase.

empty? → true or false

Returns `true` if `self` is `:''`, `false` otherwise.

encoding → encoding

Equivalent to `self.to_s.encoding`; see String#encoding.

end_with?(*strings) → true or false

Equivalent to `self.to_s.end_with?`; see String#end_with?.

id2name()

Returns a string representation of `self` (not including the leading colon):

```
:foo.to_s # => "foo"
```

Related: Symbol#inspect, Symbol#name.

Alias for: to_s

inspect → string

Returns a string representation of `self` (including the leading colon):

```
:foo.inspect # => ":foo"
```

Related: Symbol#to_s, Symbol#name.

intern()

Alias for: to_sym

length → integer

Equivalent to `self.to_s.length`; see String#length.

Also aliased as: size

match(pattern, offset = 0) → matchdata or nil
match(pattern, offset = 0) {|matchdata| } → object

Equivalent to `self.to_s.match`, including possible updates to global variables; see String#match.

match?(pattern, offset) → true or false

Equivalent to `sym.to_s.match?`; see String#match.

name → string

Returns a frozen string representation of `self` (not including the leading colon):

```
:foo.name         # => "foo"
:foo.name.frozen? # => true
```

Related: Symbol#to_s, Symbol#inspect.

next()

Equivalent to `self.to_s.succ.to_sym`:

```
:foo.succ # => :fop
```

Related: String#succ.

Alias for: succ

size()

Equivalent to `self.to_s.length`; see String#length.

Alias for: length

slice(*args)

Equivalent to `symbol.to_s[]`; see String#[].

Alias for: []

start_with?(*string_or_regexp) → true or false

Equivalent to `self.to_s.start_with?`; see String#start_with?.

succ

Equivalent to `self.to_s.succ.to_sym`:

```
:foo.succ # => :fop
```

Related: String#succ.

Also aliased as: next

swapcase(*options) → symbol

Equivalent to `sym.to_s.swapcase.to_sym`.

See String#swapcase.

to_proc

Returns a `Proc` object which calls the method with name of `self` on the first parameter and passes the remaining parameters to the method.

```
proc = :to_s.to_proc   # => #<Proc:0x000001afe0e48680(&:to_s) (lambda)>
proc.call(1000)        # => "1000"
proc.call(1000, 16)    # => "3e8"
(1..3).collect(&:to_s) # => ["1", "2", "3"]
```

to_s → string

Returns a string representation of `self` (not including the leading colon):

```
:foo.to_s # => "foo"
```

Related: `Symbol#inspect`, `Symbol#name`.

Also aliased as: id2name

to_sym → self

Returns `self`.

Related: `String#to_sym`.

Also aliased as: intern

upcase(*options) → symbol

Equivalent to `sym.to_s.upcase.to_sym`.

See `String#upcase`.