Class

Symbol < Object

Symbol objects represent names inside the Ruby interpreter. They are generated using the :name or :"arbitrary text" literal syntax and by using the various to_sym methods. The same Symbol object will be created for a given name string for the duration of a program's execution, regardless of the context or meaning of that name. Symbols can be arbitrary sequences
1.9 of characters. Like strings, a symbol literal containing any characters with the top-bit set will have an encoding determined by the encoding of the source file containing the definition.

1.9 Ruby 1.9 adds a lot of new string-like functionality to symbols.

Mixes in

Comparable:

1.9 `<, <=, ==, >=, >, between?`

Class methods

all_symbols

Symbol.all_symbols → *array*

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

```
Symbol.all_symbols.size    # =>  1485
Symbol.all_symbols[1,20]   # =>  [:"<IFUNC>", :"<CFUNC>", :respond_to?,
                                 :"#__ThrowState__",
                                 :"core#set_method_alias",
                                 :"core#set_variable_alias",
                                 :"core#undef_method",
                                 :"core#define_method",
                                 :"core#define_singleton_method",
                                 :"core#set_postexe", :each, :length,
                                 :lambda, :intern, :gets, :succ,
                                 :method_missing, :send, :__send__,
                                 :initialize]
```

Instance methods

<=>

sym <=> *other_sym* → −1, 0, +1

1.9 Compares *sym* to *other_sym* after converting each to strings. <=> is the basis for the methods <, <=, >, >=, and between?, included from module Comparable. The method Symbol#== does not use Comparable#==.

```
:abcdef <=> :abcde     # =>  1
:abcdef <=> :abcdef    # =>  0
:abcdef <=> :abcdefg   # =>  -1
:abcdef <=> :ABCDEF    # =>  1
```

==

sym == *obj* → true or false

1.9 Returns true only if *sym* and *obj* are symbols with the same object_id.

```
:abcdef == :abcde    # =>  false
:abcdef == :abcdef   # =>  true
```

=~

sym =~ *obj* → *int* or nil

1.9

Converts *sym* to a string and matches it against *obj*. with the same object_id.

```
:abcdef =~ /.[aeiou]/   # =>  3
:abcdef =~ /xx/         # =>  nil
```

[ ]

sym[*int*] → *string* or nil
sym[*int*, *int*] → *string* or nil
sym[*range*] → *string* or nil
sym[*regexp*] → *string* or nil
sym[*regexp*, *int*] → *string* or nil
sym[*string*] → *string* or nil

1.9

Converts *sym* to a string and then indexes it using the same parameters as String#[].

```
:"hello there"[1]                  # =>  "e"
:"hello there"[1,3]                # =>  "ell"
:"hello there"[1..3]               # =>  "ell"
:"hello there"[1...3]              # =>  "el"
:"hello there"[-3,2]               # =>  "er"
:"hello there"[-4..-2]             # =>  "her"
:"hello there"[-2..-4]             # =>  ""
:"hello there"[/[aeiou](.)\1/]     # =>  "ell"
:"hello there"[/[aeiou](.)\1/, 0]  # =>  "ell"
:"hello there"[/[aeiou](.)\1/, 1]  # =>  "l"
:"hello there"[/[aeiou](.)\1/, 2]  # =>  nil
:"hello there"[/(..)e/]            # =>  "the"
:"hello there"[/(..)e/, 1]         # =>  "th"
:"hello there"["lo"]               # =>  "lo"
:"hello there"["bye"]              # =>  nil
```

capitalize

sym.capitalize → *symbol*

1.9

Returns a symbol with the first character of *sym* converted to uppercase and the remainder to lowercase.

```
:hello.capitalize           # =>  :Hello
:"HELLO WORLD".capitalize   # =>  :"Hello world"
:"123ABC".capitalize        # =>  :"123abc"
```

casecmp

sym.casecmp(*other*) → −1, 0, +1, or nil

1.9

Case-insensitive version of Symbol#<=>. Returns nil if *other* is not a symbol.

```
:abcdef.casecmp(:abcde)     # =>  1
:abcdef.casecmp(:abcdef)    # =>  0
:abcdef.casecmp(:ABCDEF)    # =>  0
:aBcDeF.casecmp(:abcdef)    # =>  0
:abcdef.casecmp(:abcdefg)   # =>  -1
:abcdef.casecmp("abcdef")   # =>  nil
```

downcase

sym.downcase → *symbol*

1.9

Returns a symbol with all the characters of *sym* converted to lowercase.

```
:Hello.downcase           # =>   :hello
:"HELLO WORLD".downcase   # =>   :"hello world"
:"123ABC".downcase        # =>   :"123abc"
```

empty?

sym.empty → true or false

1.9

Returns true if the string representation of *sym* is empty.

```
:hello.empty?   # =>   false
:"".empty?      # =>   true
```

encoding

sym.encoding → *enc*

1.9

Returns the encoding of *sym*.

```
# encoding: utf-8
:hello.encoding   # =>   #<Encoding:US-ASCII>
:"δog".encoding   # =>   #<Encoding:UTF-8>
```

id2name

sym.id2name → *string*

Returns the string representation of *sym*.

```
:fred.id2name                 # =>   "fred"
:"99 red balloons!".id2name   # =>   "99 red balloons!"
```

inspect

sym.inspect → *string*

Returns the representation of *sym* as a symbol literal.

```
:fred.inspect                 # =>   :fred
:"99 red balloons!".inspect   # =>   :"99 red balloons!"
```

intern

sym.intern → *sym*

Synonym for Symbol#to_sym.

length

sym.length → *int*

1.9

Returns the number of characters in the string representation *sym*.

```
# encoding: utf-8
:dog.length   # =>   3
:δog.length   # =>   3
```

match

sym.match(*regexp*) → *int* or nil

1.9

Converts *self* to a string and then matches it against *regexp*. Unlike String#match, does not support blocks or non-regexp parameters.

```
:hello.match(/(.)\1/)   # =>   2
:hello.match(/ll/)      # =>   2
```

next *sym*.next → *symbol*

1.9 Synonym for Symbol#succ.

size *sym*.size → *int*

1.9 Synonym for Symbol#length.

slice
sym.slice(*int*) → *string* or nil
sym.slice(*int*, *int*) → *string* or nil
sym.slice(*range*) → *string* or nil
sym.slice(*regexp*) → *string* or nil
sym.slice(*match_string*) → *string* or nil

1.9 Synonym for Symbol#[ ].

succ *sym*.succ → *symbol*

1.9 Returns the successor to *sym* using the same rules as String#succ.

```
:abcd.succ          # =>   :abce
:THX1138.succ       # =>   :THX1139
:"<<koala>>".succ   # =>   :"<<koalb>>"
:"1999zzz".succ     # =>   :"2000aaa"
:ZZZ9999.succ       # =>   :AAAA0000
:"***".succ         # =>   :"**+"
```

swapcase *sym*.swapcase → *symbol*

1.9 Returns a symbol with the case of all the characters of *sym* swapped.

```
:Hello.swapcase      # =>   :hELLO
:"123ABC".swapcase   # =>   :"123abc"
```

to_proc *sym*.to_proc → *proc*

1.9 Allows a symbol to be used when a block is expected. The symbol acts as a method invoked on each parameter to the block. See page 379 for more information.

```
%w{ant bee cat}.map(&:reverse)   # =>   ["tna", "eeb", "tac"]
```

to_s *sym*.to_s → *string*

Synonym for Symbol#id2name.

to_sym *sym*.to_sym → *sym*

Symbols are symbol-like!

upcase *sym*.upcase → *symbol*

1.9 Returns a symbol with of all the characters of *sym* in uppercase.

```
:Hello.upcase      # =>   :HELLO
:"123Abc".upcase   # =>   :"123ABC"
```