

Programming Languages — Ruby

IPA Ruby Standardization WG Draft

December 1, 2009

©Information-technology Promotion Agency, Japan 2009

Contents		Page
1	Scope	1
2	Normative references	1
3	Conformance	1
4	Terms and definitions	2
5	Notational conventions	3
6	Objects	5
6.1	General description	5
6.2	Boolean values	7
7	Execution context	7
7.1	Contextual attributes	7
7.2	The initial state	8
8	Lexical structure	8
8.1	Source text	8
8.2	Line terminators	9
8.3	Whitespace	10
8.4	Comments	10
8.5	Tokens	11
8.5.1	Reserved words	11
8.5.2	Identifiers	12
8.5.3	Punctuators	13
8.5.4	Operators	13
8.5.5	Literals	14
8.5.5.1	Numeric literals	14
8.5.5.2	String literals	17
8.5.5.2.1	Single quoted strings	17
8.5.5.2.2	Double quoted strings	18
8.5.5.2.3	Quoted non-expanded literal strings	21
8.5.5.2.4	Quoted expanded literal strings	22
8.5.5.2.5	Here documents	23
8.5.5.2.6	External command execution	26
8.5.5.3	Array literals	26
8.5.5.4	Regular expression literals	29
8.5.5.5	Symbol literals	30
9	Scope of variables	31
9.1	Local variables	31
9.1.1	Scopes of local variables	31
9.1.2	References to local variables	32
9.2	Global variables	33
10	Program structure	33
10.1	Program	33
10.2	Compound statement	34
11	Expressions	35

11.1	Logical expressions	35
11.2	Method invocation expressions	37
11.2.1	Method arguments	42
11.2.2	Blocks	44
11.2.3	The super expression	47
11.2.4	The yield expression	50
11.3	Operator expressions	51
11.3.1	Assignments	51
11.3.1.1	Single assignments	52
11.3.1.1.1	Single variable assignments	52
11.3.1.1.2	Single indexing assignments	55
11.3.1.1.3	Single method assignments	55
11.3.1.2	Abbreviated assignments	56
11.3.1.2.1	Abbreviated variable assignments	56
11.3.1.2.2	Abbreviated indexing assignments	57
11.3.1.2.3	Abbreviated method assignments	58
11.3.1.3	Multiple assignments	59
11.3.1.4	Assignments with rescue modifiers	62
11.3.2	Unary operators	63
11.3.2.1	The defined? expression	64
11.3.3	Binary operators	65
11.4	Primary expressions	67
11.4.1	Control structures	68
11.4.1.1	Conditional expressions	68
11.4.1.1.1	The if expression	68
11.4.1.1.2	The unless expression	69
11.4.1.1.3	The case expression	70
11.4.1.1.4	Conditional operator	71
11.4.1.2	Iteration expressions	71
11.4.1.2.1	The while expression	72
11.4.1.2.2	The until expression	72
11.4.1.2.3	The for expression	73
11.4.1.3	Jump expressions	74
11.4.1.3.1	The return expression	74
11.4.1.3.2	The break expression	76
11.4.1.3.3	The next expression	76
11.4.1.3.4	The redo expression	77
11.4.1.3.5	The retry expression	78
11.4.1.4	Exceptions	78
11.4.1.4.1	The rescue expression	78
11.4.2	Grouping expression	80
11.4.3	Variable references	80
11.4.3.1	Constants	81
11.4.3.2	Scoped constants	82
11.4.3.3	Global variables	82
11.4.3.4	Class variables	82
11.4.3.5	Instance variables	83
11.4.3.6	Local variables	83
11.4.3.7	Pseudo variables	83
11.4.3.7.1	nil	84
11.4.3.7.2	true and false	84
11.4.3.7.3	self	84

11.4.4	Object constructors	85
11.4.4.1	Array constructor	85
11.4.4.2	Hash constructor	85
11.4.4.3	Range constructor	86
11.4.5	Literals	87
12	Statements	87
12.1	The expression statement	87
12.2	The if modifier statement	88
12.3	The unless modifier statement	88
12.4	The while modifier statement	89
12.5	The until modifier statement	89
12.6	The rescue modifier statement	89
13	Classes and modules	90
13.1	Modules	90
13.1.1	General description	90
13.1.2	Module definition	91
13.1.3	Module inclusion	92
13.2	Classes	92
13.2.1	General description	92
13.2.2	Class definition	93
13.2.3	Inheritance	94
13.2.4	Instance creation	95
13.3	Methods	95
13.3.1	Method definition	95
13.3.2	Method parameters	96
13.3.3	Method invocation	98
13.3.4	Method lookup	100
13.3.5	Method visibility	101
13.3.5.1	Public methods	101
13.3.5.2	Private methods	101
13.3.5.3	Protected methods	101
13.3.5.4	Visibility change	102
13.3.6	The alias statement	102
13.3.7	The undef statement	103
13.4	Eigenclass	104
13.4.1	General description	104
13.4.2	Eigenclass definition	104
13.4.3	Singleton method definition	105
14	Exceptions	106
14.1	Cause of exceptions	106
14.2	Exception handling	107
15	Built-in classes and modules	108
15.1	General description	108
15.2	Built-in classes	109
15.2.1	Object	109
15.2.1.1	Direct superclass	109
15.2.1.2	Included modules	109
15.2.1.3	Constants	109
15.2.1.4	Instance methods	110

15.2.1.4.1	Object#initialize	110
15.2.2	Module	110
15.2.2.1	Direct superclass	110
15.2.2.2	Singleton methods	110
15.2.2.2.1	Module.constants	110
15.2.2.2.2	Module.nesting	111
15.2.2.3	Instance methods	111
15.2.2.3.1	Module#<	111
15.2.2.3.2	Module#<=	111
15.2.2.3.3	Module#<=>	112
15.2.2.3.4	Module#==	112
15.2.2.3.5	Module#===	112
15.2.2.3.6	Module#>	112
15.2.2.3.7	Module#>=	113
15.2.2.3.8	Module#alias_method	113
15.2.2.3.9	Module#ancestors	113
15.2.2.3.10	Module#append_features	114
15.2.2.3.11	Module#attr	114
15.2.2.3.12	Module#attr_accessor	115
15.2.2.3.13	Module#attr_reader	115
15.2.2.3.14	Module#attr_writer	116
15.2.2.3.15	Module#class_eval	116
15.2.2.3.16	Module#class_variable_defined?	117
15.2.2.3.17	Module#class_variable_get	118
15.2.2.3.18	Module#class_variable_set	118
15.2.2.3.19	Module#class_variables	118
15.2.2.3.20	Module#const_defined?	119
15.2.2.3.21	Module#const_get	119
15.2.2.3.22	Module#const_missing	119
15.2.2.3.23	Module#const_set	120
15.2.2.3.24	Module#constants	120
15.2.2.3.25	Module#extend_object	120
15.2.2.3.26	Module#extended	121
15.2.2.3.27	Module#include	121
15.2.2.3.28	Module#include?	121
15.2.2.3.29	Module#included	122
15.2.2.3.30	Module#included_modules	122
15.2.2.3.31	Module#initialize	122
15.2.2.3.32	Module#initialize_copy	123
15.2.2.3.33	Module#instance_methods	123
15.2.2.3.34	Module#method_defined?	124
15.2.2.3.35	Module#module_eval	125
15.2.2.3.36	Module#private	125
15.2.2.3.37	Module#protected	125
15.2.2.3.38	Module#public	125
15.2.2.3.39	Module#remove_class_variable	126
15.2.2.3.40	Module#remove_const	126
15.2.2.3.41	Module#remove_method	127
15.2.2.3.42	Module#undef_method	127
15.2.3	Class	128
15.2.3.1	Direct superclass	128
15.2.3.2	Instance methods	128

15.2.3.2.1	Class#initialize	128
15.2.3.2.2	Class#initialize_copy	129
15.2.3.2.3	Class#new	129
15.2.3.2.4	Class#superclass	129
15.2.4	NilClass	130
15.2.4.1	Direct superclass	130
15.2.4.2	Instance methods	130
15.2.4.2.1	NilClass#&	130
15.2.4.2.2	NilClass#^	130
15.2.4.2.3	NilClass#nil?	130
15.2.4.2.4	NilClass# 	130
15.2.5	TrueClass	131
15.2.5.1	Direct superclass	131
15.2.5.2	Instance methods	131
15.2.5.2.1	TrueClass#&	131
15.2.5.2.2	TrueClass#^	131
15.2.5.2.3	TrueClass#to_s	132
15.2.5.2.4	TrueClass# 	132
15.2.6	FalseClass	132
15.2.6.1	Direct superclass	132
15.2.6.2	Instance methods	132
15.2.6.2.1	FalseClass#&	132
15.2.6.2.2	FalseClass#^	132
15.2.6.2.3	FalseClass#to_s	133
15.2.6.2.4	FalseClass# 	133
15.2.7	Numeric	133
15.2.7.1	Direct superclass	133
15.2.7.2	Included modules	133
15.2.7.3	Instance methods	133
15.2.7.3.1	Numeric#+@	133
15.2.7.3.2	Numeric#-@	134
15.2.7.3.3	Numeric#abs	134
15.2.7.3.4	Numeric#coerce	134
15.2.8	Integer	135
15.2.8.1	Direct superclass	136
15.2.8.2	Instance methods	136
15.2.8.2.1	Integer#+	136
15.2.8.2.2	Integer#-	136
15.2.8.2.3	Integer#*	137
15.2.8.2.4	Integer#/	138
15.2.8.2.5	Integer#%	138
15.2.8.2.6	Integer#<=>	139
15.2.8.2.7	Integer#==	139
15.2.8.2.8	Integer#~	140
15.2.8.2.9	Integer#&	140
15.2.8.2.10	Integer# 	140
15.2.8.2.11	Integer#^	141
15.2.8.2.12	Integer#<<	141
15.2.8.2.13	Integer#>>	141
15.2.8.2.14	Integer#ceil	142
15.2.8.2.15	Integer#downto	142
15.2.8.2.16	Integer#eql?	142

15.2.8.2.17	Integer#floor	143
15.2.8.2.18	Integer#hash	143
15.2.8.2.19	Integer#next	143
15.2.8.2.20	Integer#round	143
15.2.8.2.21	Integer#succ	143
15.2.8.2.22	Integer#times	144
15.2.8.2.23	Integer#to_f	144
15.2.8.2.24	Integer#to_i	144
15.2.8.2.25	Integer#truncate	144
15.2.8.2.26	Integer#upto	145
15.2.9	Float	145
15.2.9.1	Direct superclass	145
15.2.9.2	Instance methods	146
15.2.9.2.1	Float#+	146
15.2.9.2.2	Float#-	146
15.2.9.2.3	Float#*	147
15.2.9.2.4	Float#/	147
15.2.9.2.5	Float#%	148
15.2.9.2.6	Float#<=>	149
15.2.9.2.7	Float#==	149
15.2.9.2.8	Float#ceil	150
15.2.9.2.9	Float#finite?	150
15.2.9.2.10	Float#floor	150
15.2.9.2.11	Float#infinite?	151
15.2.9.2.12	Float#round	151
15.2.9.2.13	Float#to_f	151
15.2.9.2.14	Float#to_i	151
15.2.9.2.15	Float#truncate	152
15.2.10	String	152
15.2.10.1	Direct superclass	152
15.2.10.2	Included modules	152
15.2.10.3	Instance methods	152
15.2.10.3.1	String#*	152
15.2.10.3.2	String#+	153
15.2.10.3.3	String#<=>	153
15.2.10.3.4	String#==	154
15.2.10.3.5	String#=~	154
15.2.10.3.6	String#[]	155
15.2.10.3.7	String#capitalize	156
15.2.10.3.8	String#capitalize!	156
15.2.10.3.9	String#chomp	157
15.2.10.3.10	String#chomp!	157
15.2.10.3.11	String#chop	157
15.2.10.3.12	String#chop!	158
15.2.10.3.13	String#downcase	158
15.2.10.3.14	String#downcase!	158
15.2.10.3.15	String#each_line	159
15.2.10.3.16	String#empty?	159
15.2.10.3.17	String#eql?	159
15.2.10.3.18	String#gsub	160
15.2.10.3.19	String#gsub!	161
15.2.10.3.20	String#hash	161

15.2.10.3.21	String#include?	162
15.2.10.3.22	String#initialize	162
15.2.10.3.23	String#initialize_copy	162
15.2.10.3.24	String#intern	163
15.2.10.3.25	String#length	163
15.2.10.3.26	String#match	163
15.2.10.3.27	String#replace	164
15.2.10.3.28	String#reverse	164
15.2.10.3.29	String#reverse!	164
15.2.10.3.30	String#scan	164
15.2.10.3.31	String#size	165
15.2.10.3.32	String#slice	165
15.2.10.3.33	String#split	166
15.2.10.3.34	String#sub	167
15.2.10.3.35	String#sub!	168
15.2.10.3.36	String#upcase	168
15.2.10.3.37	String#upcase!	168
15.2.10.3.38	String#to_i	168
15.2.10.3.39	String#to_f	169
15.2.10.3.40	String#to_s	170
15.2.10.3.41	String#to_sym	170
15.2.11	Symbol	170
15.2.11.1	Direct superclass	170
15.2.11.2	Instance methods	170
15.2.11.2.1	Symbol#===	170
15.2.11.2.2	Symbol#id2name	171
15.2.11.2.3	Symbol#to_s	171
15.2.11.2.4	Symbol#to_sym	171
15.2.12	Array	171
15.2.12.1	Direct superclass	172
15.2.12.2	Included modules	172
15.2.12.3	Singleton methods	172
15.2.12.3.1	Array.[]	172
15.2.12.4	Instance methods	172
15.2.12.4.1	Array#*	172
15.2.12.4.2	Array#+	173
15.2.12.4.3	Array#<<	173
15.2.12.4.4	Array#[]	173
15.2.12.4.5	Array#[]=	174
15.2.12.4.6	Array#clear	175
15.2.12.4.7	Array#collect!	175
15.2.12.4.8	Array#concat	175
15.2.12.4.9	Array#each	176
15.2.12.4.10	Array#each_index	176
15.2.12.4.11	Array#empty?	176
15.2.12.4.12	Array#first	177
15.2.12.4.13	Array#initialize	177
15.2.12.4.14	Array#initialize_copy	178
15.2.12.4.15	Array#join	178
15.2.12.4.16	Array#last	179
15.2.12.4.17	Array#length	179
15.2.12.4.18	Array#map!	180

15.2.12.4.19	Array#pop	180
15.2.12.4.20	Array#push	180
15.2.12.4.21	Array#replace	180
15.2.12.4.22	Array#reverse	180
15.2.12.4.23	Array#reverse!	181
15.2.12.4.24	Array#shift	181
15.2.12.4.25	Array#size	181
15.2.12.4.26	Array#slice	181
15.2.12.4.27	Array#unshift	182
15.2.13	Hash	182
15.2.13.1	Direct superclass	183
15.2.13.2	Included modules	183
15.2.13.3	Instance methods	183
15.2.13.3.1	Hash#==	183
15.2.13.3.2	Hash#[]	183
15.2.13.3.3	Hash#[]=	184
15.2.13.3.4	Hash#clear	184
15.2.13.3.5	Hash#default	184
15.2.13.3.6	Hash#default=	185
15.2.13.3.7	Hash#default_proc	185
15.2.13.3.8	Hash#delete	186
15.2.13.3.9	Hash#each	186
15.2.13.3.10	Hash#each_key	186
15.2.13.3.11	Hash#each_value	187
15.2.13.3.12	Hash#empty?	187
15.2.13.3.13	Hash#has_key?	187
15.2.13.3.14	Hash#has_value?	187
15.2.13.3.15	Hash#include?	188
15.2.13.3.16	Hash#initialize	188
15.2.13.3.17	Hash#initialize_copy	188
15.2.13.3.18	Hash#key?	189
15.2.13.3.19	Hash#keys	189
15.2.13.3.20	Hash#length	189
15.2.13.3.21	Hash#member?	190
15.2.13.3.22	Hash#merge	190
15.2.13.3.23	Hash#replace	190
15.2.13.3.24	Hash#shift	190
15.2.13.3.25	Hash#size	191
15.2.13.3.26	Hash#store	191
15.2.13.3.27	Hash#value?	191
15.2.13.3.28	Hash#values	192
15.2.14	Range	192
15.2.14.1	Direct superclass	192
15.2.14.2	Included modules	192
15.2.14.3	Instance methods	192
15.2.14.3.1	Range#==	192
15.2.14.3.2	Range#===	193
15.2.14.3.3	Range#begin	193
15.2.14.3.4	Range#each	194
15.2.14.3.5	Range#end	194
15.2.14.3.6	Range#exclude_end?	194
15.2.14.3.7	Range#first	195

15.2.14.3.8	Range#include?	195
15.2.14.3.9	Range#initialize	195
15.2.14.3.10	Range#last	195
15.2.14.3.11	Range#member?	196
15.2.15	Regexp	196
15.2.15.1	Direct superclass	196
15.2.15.2	Constants	196
15.2.15.3	Patterns	197
15.2.15.4	Matching process	200
15.2.15.5	Singleton methods	201
15.2.15.5.1	Regexp.compile	201
15.2.15.5.2	Regexp.escape	202
15.2.15.5.3	Regexp.last_match	202
15.2.15.5.4	Regexp.quote	203
15.2.15.6	Instance methods	203
15.2.15.6.1	Regexp#initialize	203
15.2.15.6.2	Regexp#initialize_copy	204
15.2.15.6.3	Regexp#==	204
15.2.15.6.4	Regexp#===	205
15.2.15.6.5	Regexp#=~	205
15.2.15.6.6	Regexp#casefold?	206
15.2.15.6.7	Regexp#match	206
15.2.15.6.8	Regexp#source	206
15.2.16	MatchData	206
15.2.16.1	Direct superclass	207
15.2.16.2	Instance methods	207
15.2.16.2.1	MatchData#[]	207
15.2.16.2.2	MatchData#begin	207
15.2.16.2.3	MatchData#captures	207
15.2.16.2.4	MatchData#end	208
15.2.16.2.5	MatchData#initialize_copy	208
15.2.16.2.6	MatchData#length	209
15.2.16.2.7	MatchData#offset	209
15.2.16.2.8	MatchData#post_match	209
15.2.16.2.9	MatchData#pre_match	210
15.2.16.2.10	MatchData#size	210
15.2.16.2.11	MatchData#string	210
15.2.16.2.12	MatchData#to_a	210
15.2.16.2.13	MatchData#to_s	210
15.2.17	Proc	211
15.2.17.1	Direct superclass	211
15.2.17.2	Singleton methods	211
15.2.17.2.1	Proc.new	211
15.2.17.3	Instance methods	211
15.2.17.3.1	Proc#[]	211
15.2.17.3.2	Proc#arity	212
15.2.17.3.3	Proc#call	212
15.2.17.3.4	Proc#clone	213
15.2.17.3.5	Proc#dup	213
15.2.18	Struct	214
15.2.18.1	Direct superclass	214
15.2.18.2	Singleton methods	214

15.2.18.2.1	Struct.new	214
15.2.18.3	Instance methods	216
15.2.18.3.1	Struct#==	216
15.2.18.3.2	Struct#[]	216
15.2.18.3.3	Struct#[]=	217
15.2.18.3.4	Struct#each	218
15.2.18.3.5	Struct#each_pair	218
15.2.18.3.6	Struct#members	218
15.2.18.3.7	Struct#select	219
15.2.18.3.8	Struct#initialize	219
15.2.18.3.9	Struct#initialize_copy	219
15.2.19	Time	220
15.2.19.1	Direct superclass	220
15.2.19.2	Time computation	220
15.2.19.2.1	Day	220
15.2.19.2.2	Year	221
15.2.19.2.3	Month	221
15.2.19.2.4	Days of month	222
15.2.19.2.5	Hours, Minutes, and Seconds	222
15.2.19.3	Time zone and Local time	223
15.2.19.4	Daylight saving time	223
15.2.19.5	Singleton methods	223
15.2.19.5.1	Time.at	223
15.2.19.5.2	Time.gm	224
15.2.19.5.3	Time.local	226
15.2.19.5.4	Time.mktime	226
15.2.19.5.5	Time.now	226
15.2.19.5.6	Time.utc	226
15.2.19.6	Instance methods	227
15.2.19.6.1	Time#+	227
15.2.19.6.2	Time#-	227
15.2.19.6.3	Time#<=>	228
15.2.19.6.4	Time#asctime	228
15.2.19.6.5	Time#ctime	229
15.2.19.6.6	Time#day	229
15.2.19.6.7	Time#dst?	229
15.2.19.6.8	Time#getgm	230
15.2.19.6.9	Time#getlocal	230
15.2.19.6.10	Time#getutc	230
15.2.19.6.11	Time#gmt?	230
15.2.19.6.12	Time#gmt_offset	230
15.2.19.6.13	Time#gmtime	230
15.2.19.6.14	Time#gmtoff	231
15.2.19.6.15	Time#hour	231
15.2.19.6.16	Time#localtime	231
15.2.19.6.17	Time#mday	231
15.2.19.6.18	Time#min	232
15.2.19.6.19	Time#mon	232
15.2.19.6.20	Time#month	232
15.2.19.6.21	Time#sec	233
15.2.19.6.22	Time#to_f	233
15.2.19.6.23	Time#to_i	233

15.2.19.6.24	Time#usec	233
15.2.19.6.25	Time#utc	234
15.2.19.6.26	Time#utc?	234
15.2.19.6.27	Time#utc_offset	234
15.2.19.6.28	Time#wday	234
15.2.19.6.29	Time#yday	235
15.2.19.6.30	Time#year	235
15.2.19.6.31	Time#zone	235
15.2.19.6.32	Time#initialize	236
15.2.19.6.33	Time#initialize_copy	236
15.2.20	IO	236
15.2.20.1	Direct superclass	237
15.2.20.2	Included modules	237
15.2.20.3	Singleton methods	237
15.2.20.3.1	IO.open	237
15.2.20.4	Instance methods	238
15.2.20.4.1	IO#close	238
15.2.20.4.2	IO#closed?	238
15.2.20.4.3	IO#each	238
15.2.20.4.4	IO#each_byte	239
15.2.20.4.5	IO#each_line	239
15.2.20.4.6	IO#eof?	239
15.2.20.4.7	IO#flush	240
15.2.20.4.8	IO#getc	240
15.2.20.4.9	IO#gets	240
15.2.20.4.10	IO#initialize_copy	241
15.2.20.4.11	IO#print	241
15.2.20.4.12	IO#putc	241
15.2.20.4.13	IO#puts	242
15.2.20.4.14	IO#read	243
15.2.20.4.15	IO#readchar	243
15.2.20.4.16	IO#readline	243
15.2.20.4.17	IO#readlines	244
15.2.20.4.18	IO#sync	244
15.2.20.4.19	IO#sync=	245
15.2.20.4.20	IO#write	245
15.2.21	File	245
15.2.21.1	Direct superclass	246
15.2.21.2	Singleton methods	246
15.2.21.2.1	File.exist?	246
15.2.21.3	Instance methods	246
15.2.21.3.1	File#initialize	246
15.2.21.3.2	File#path	247
15.2.22	Exception	247
15.2.22.1	Direct superclass	247
15.2.22.2	Built-in exception classes	247
15.2.22.3	Singleton methods	247
15.2.22.3.1	Exception.exception	247
15.2.22.4	Instance methods	247
15.2.22.4.1	Exception#exception	247
15.2.22.4.2	Exception#message	249
15.2.22.4.3	Exception#to_s	249

15.2.22.4.4	Exception#initialize	249
15.2.23	StandardError	249
15.2.23.1	Direct superclass	249
15.2.24	ArgumentError	250
15.2.24.1	Direct superclass	250
15.2.25	LocalJumpError	250
15.2.25.1	Direct superclass	250
15.2.25.2	Instance methods	250
15.2.25.2.1	LocalJumpError#exit_value	250
15.2.25.2.2	LocalJumpError#reason	250
15.2.26	RangeError	250
15.2.26.1	Direct superclass	250
15.2.27	RegexpError	250
15.2.27.1	Direct superclass	250
15.2.28	RuntimeError	251
15.2.28.1	Direct superclass	251
15.2.29	TypeError	251
15.2.29.1	Direct superclass	251
15.2.30	ZeroDivisionError	251
15.2.30.1	Direct superclass	251
15.2.31	NameError	251
15.2.31.1	Direct superclass	251
15.2.31.2	Instance methods	251
15.2.31.2.1	NameError#name	251
15.2.31.2.2	NameError#initialize	252
15.2.32	NoMethodError	252
15.2.32.1	Direct superclass	252
15.2.32.2	Instance methods	252
15.2.32.2.1	NoMethodError#args	252
15.2.32.2.2	NoMethodError#initialize	252
15.2.33	IndexError	253
15.2.33.1	Direct superclass	253
15.2.34	StopIteration	253
15.2.34.1	Direct superclass	253
15.2.35	IOError	253
15.2.35.1	Direct superclass	253
15.2.36	EOFError	253
15.2.36.1	Direct superclass	253
15.2.37	SystemCallError	253
15.2.37.1	Direct superclass	253
15.2.38	ScriptError	253
15.2.38.1	Direct superclass	254
15.2.39	SyntaxError	254
15.2.39.1	Direct superclass	254
15.2.40	LoadError	254
15.2.40.1	Direct superclass	254
15.3	Built-in modules	254
15.3.1	Kernel	254
15.3.1.1	Singleton methods	254
15.3.1.1.1	Kernel.`	254
15.3.1.1.2	Kernel.block_given?	255
15.3.1.1.3	Kernel.eval	255

15.3.1.1.4	Kernel.global_variables	255
15.3.1.1.5	Kernel.iterator?	255
15.3.1.1.6	Kernel.lambda	256
15.3.1.1.7	Kernel.local_variables	257
15.3.1.1.8	Kernel.loop	257
15.3.1.1.9	Kernel.method_missing	257
15.3.1.1.10	Kernel.p	258
15.3.1.1.11	Kernel.print	258
15.3.1.1.12	Kernel.puts	258
15.3.1.1.13	Kernel.raise	259
15.3.1.1.14	Kernel.require	259
15.3.1.2	Instance methods	260
15.3.1.2.1	Kernel#==	260
15.3.1.2.2	Kernel#===	260
15.3.1.2.3	Kernel#_id_	261
15.3.1.2.4	Kernel#_send_	261
15.3.1.2.5	Kernel#‘	261
15.3.1.2.6	Kernel#block_given?	261
15.3.1.2.7	Kernel#class	262
15.3.1.2.8	Kernel#clone	262
15.3.1.2.9	Kernel#dup	262
15.3.1.2.10	Kernel#eql?	263
15.3.1.2.11	Kernel#equal?	263
15.3.1.2.12	Kernel#eval	263
15.3.1.2.13	Kernel#extend	263
15.3.1.2.14	Kernel#global_variables	264
15.3.1.2.15	Kernel#hash	264
15.3.1.2.16	Kernel#initialize_copy	264
15.3.1.2.17	Kernel#inspect	265
15.3.1.2.18	Kernel#instance_eval	265
15.3.1.2.19	Kernel#instance_of?	265
15.3.1.2.20	Kernel#instance_variable_defined?	266
15.3.1.2.21	Kernel#instance_variable_get	266
15.3.1.2.22	Kernel#instance_variable_set	267
15.3.1.2.23	Kernel#instance_variables	267
15.3.1.2.24	Kernel#is_a?	267
15.3.1.2.25	Kernel#iterator?	268
15.3.1.2.26	Kernel#kind_of?	268
15.3.1.2.27	Kernel#lambda	268
15.3.1.2.28	Kernel#local_variables	268
15.3.1.2.29	Kernel#loop	268
15.3.1.2.30	Kernel#method_missing	269
15.3.1.2.31	Kernel#methods	269
15.3.1.2.32	Kernel#nil?	269
15.3.1.2.33	Kernel#object_id	269
15.3.1.2.34	Kernel#p	270
15.3.1.2.35	Kernel#print	270
15.3.1.2.36	Kernel#private_methods	270
15.3.1.2.37	Kernel#protected_methods	271
15.3.1.2.38	Kernel#public_methods	271
15.3.1.2.39	Kernel#puts	271
15.3.1.2.40	Kernel#raise	271

15.3.1.2.41	Kernel#remove_instance_variable	272
15.3.1.2.42	Kernel#require	272
15.3.1.2.43	Kernel#respond_to?	272
15.3.1.2.44	Kernel#send	273
15.3.1.2.45	Kernel#singleton_methods	273
15.3.1.2.46	Kernel#to_s	274
15.3.2	Enumerable	274
15.3.2.1	Instance methods	274
15.3.2.1.1	Enumerable#all?	274
15.3.2.1.2	Enumerable#any?	275
15.3.2.1.3	Enumerable#collect	275
15.3.2.1.4	Enumerable#detect	275
15.3.2.1.5	Enumerable#each_with_index	276
15.3.2.1.6	Enumerable#entries	276
15.3.2.1.7	Enumerable#find	277
15.3.2.1.8	Enumerable#find_all	277
15.3.2.1.9	Enumerable#grep	277
15.3.2.1.10	Enumerable#include?	278
15.3.2.1.11	Enumerable#inject	278
15.3.2.1.12	Enumerable#map	279
15.3.2.1.13	Enumerable#max	279
15.3.2.1.14	Enumerable#min	280
15.3.2.1.15	Enumerable#member?	280
15.3.2.1.16	Enumerable#partition	280
15.3.2.1.17	Enumerable#reject	281
15.3.2.1.18	Enumerable#select	281
15.3.2.1.19	Enumerable#sort	282
15.3.2.1.20	Enumerable#to_a	282
15.3.3	Comparable	283
15.3.3.1	Instance methods	283
15.3.3.1.1	Comparable#<	283
15.3.3.1.2	Comparable#<=	283
15.3.3.1.3	Comparable#==	283
15.3.3.1.4	Comparable#>	284
15.3.3.1.5	Comparable#>=	284
15.3.3.1.6	Comparable#between?	284
Annex A (informative)	Grammar Summary	286

Information technology — Programming Languages — Ruby

1 Scope

This document specifies the syntax and semantics of the computer programming language Ruby by specifying requirements for a conforming processor and for a conforming program.

This document does not specify:

- the size or complexity of a program text that exceeds the capacity of any specific data processing system or the capacity of a particular processor;
- the minimal requirements of a data processing system that is capable of supporting a conforming processor;
- the method for activating the execution of programs on a data processing system;
- the method for reporting syntactic and runtime errors.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 646:1991 *Information technology – ISO 7-bit coded character set for information interchange*.

IEC 60559:1989 *Binary floating-point arithmetic for microprocessor systems*.

3 Conformance

A conforming Ruby program shall:

- use only those features of the language specified in this document;
- not rely on implementation dependent features;

A conforming Ruby processor shall:

- 1 • accept any conforming programs and behave as specified in this document;
- 2 • reject any program which does not conform to the syntax described in this document;
- 3 • report any unhandled exceptions raised during execution of the conforming program;

4 A conforming Ruby processor may use an internal model for the Ruby language other than the
5 one specified in this document, if it does not change the meaning of a conforming program.

6 4 Terms and definitions

7 For the purposes of this document, the following terms and definitions apply. Other terms are
8 defined where they appear in ***bold slant face*** or on the left side of a syntax rule.

9 4.1

10 **block**

11 sequence of statements which is passed to a method invocation

12 4.2

13 **class**

14 object which defines the behavior of a set of other objects called its instances

15 NOTE The behavior is a set of methods which can be invoked on an instance.

16 4.3

17 **class variable**

18 variable whose value is shared by all the instances of a class

19 4.4

20 **constant**

21 variable which is defined in a class or a module and is accessible outside the class or module

22 NOTE The value of a constant is regularly expected to remain constant during the execution of a
23 program, but Ruby does not force it. In some implementations, an assignment to a constant which
24 already exists causes a warning, but this document does not specify it.

25 4.5

26 **eigenclass**

27 special class which defines a behavior for only a single object

28 4.6

29 **exception**

30 object which represents an unexpected event

31 4.7

32 **global variable**

33 variable which is accessible everywhere in a Ruby program

34 4.8

35 **implementation defined**

36 possibly differing between implementations, but defined for every implementation

4.9

implementation dependent

possibly differing between implementations, and not necessarily defined for any particular implementation

4.10

instance method

method which can be invoked on all the instances of a class

4.11

instance variable

variable which belongs to a single object

4.12

local variable

variable which is accessible only in a certain scope introduced by a program construct such as a method definition, a block, a class definition, a module definition, an eigenclass definition, or the toplevel of a program

4.13

method

procedure which, when invoked on an object, performs a set of computations on the object

4.14

method visibility

attribute of a method which determines the conditions on which a method invocation is allowed

4.15

module

object which provides features to be included into a class or another module

4.16

object

computational entity which has a state and a behavior

4.17

singleton method

instance method of the eigenclass of an object

4.18

variable

computational entity which stores a reference to an object

5 Notational conventions

5.1 Syntax

The syntax of the language is presented as a series of productions. Each production consists of the name of the nonterminal symbol being defined followed by “::”, followed by one or more alternatives separated by “|”.

1 Terminal symbols are shown in **typewriter face**, and represent sequences of characters as they
2 appear in a program text. Non-terminal symbols are shown in *italic face*.

3 Each alternative in a production consists of a sequence of terminal and/or nonterminal symbols
4 separated by whitespace.

5 If the same nonterminal symbol occurs on the right side of a production more than once, each
6 occurrence is subscripted with a number to distinguish it from the other occurrences of the same
7 name.

8 An optional symbol is denoted by postfixing the symbol with “?”.

9 A sequence of zero or more repetitions of a symbol is denoted by postfixing the symbol with
10 “*”.

11 A sequence of one or more repetitions of a symbol is denoted by postfixing the symbol with “+”.

12 Parentheses are used to treat a sequence of symbols as a single symbol.

13 A symbol followed by the phrase **but not** and another symbol represents all sequences of charac-
14 ters represented by the first symbol except for sequences of characters represented by the second
15 symbol.

16 EXAMPLE 1 The following example means that *non-escaped-character* is any member of *source-*
17 *character* except *escape-character*:

18 *non-escaped-character* ::
19 *source-character* **but not** *escape-character*

20 Text enclosed by “[” and “]” is used to describe a sequence of characters or a location in a
21 program text.

22 EXAMPLE 2 The following example means that *source-character* is any character specified in ISO/IEC
23 646:

24 *source-character* ::
25 [any character in ISO/IEC 646]

26 In particular, the notation “[lookahead \notin *set*]” indicates that the token immediately following
27 the notation shall not begin with a sequence of characters represented by one of the members
28 of *set*. The *set* is represented as a list of one or more terminal symbols separated by commas,
29 and the list is enclosed by “{” and “}”.

30 EXAMPLE 3 The following example means that the *argument* following the *method-modifier* shall not
31 begin with “{”:

32 *command* ::

1 *method-identifier* [lookahead $\notin \{ \{ \} \}$] *argument*

2

3 In this document, use of the words **of** and **in**, when expressing a relationship between nonter-
4 minal symbols, has the following meanings:

5 ***X of Y***: refers to the *X* occurring directly in a production defining *Y*.

6 ***X in Y***: refers to any *X* occurring in a sequence which is derived directly or indirectly
7 from *Y*.

8 5.2 Conceptual name

9 A **conceptual name** is a common name given to a set of semantically related nonterminal
10 symbols in the grammar in order to refer to this set in a semantic description. A conceptual
11 name is defined by a **conceptual name definition**. A conceptual name definition consists
12 of the conceptual name to be defined followed by “::=”, followed by one or more nonterminal
13 symbols or conceptual names, separated by “|”.

14 EXAMPLE The following example defines the conceptual name *assignment*, which can be used to refer
15 either *assignment-expression* or *assignment-statement*.

16 *assignment* ::=
17 *assignment-expression*
18 | *assignment-statement*

19 6 Objects

20 6.1 General description

21 Ruby is a pure object-oriented language. It is pure in the sense that every value manipulated
22 in a Ruby program is an object including primitive values such as integers.

23 An object is a computational entity which has a state and a behavior.

24 A variable is a computational entity which stores a reference to an object. A variable has a
25 name. A variable is said to be **bound** to an object if the variable stores a reference to the
26 object. This association of a variable with an object is called a **variable binding**. When a
27 variable with name *N* is bound to an object *O*, *N* is called the name of the binding, and *O* is
28 called the value of the binding.

29 An object has a set of variable bindings. A variable whose binding is in this set is an instance
30 variable of that object. This set of bindings of instance variables represents the state of that
31 object and is encapsulated in that object.

32 A method is a procedure which, when invoked, performs a set of computations. The behavior
33 of an object is defined by a set of methods which can be invoked on that object. A method
34 has one or more (when aliased) names associated with it. An association between a name and

1 a method is called a **method binding**. When a name *N* is bound to a method *M*, *N* is called
2 the name of the binding, and *M* is called the value of the binding. A name to which a method
3 is bound is called the **method name**. A method can be invoked on an object by specifying one
4 of its names. The object on which the method is invoked is called the **receiver** of the method
5 invocation.

6 There are three constructs which define the behavior of objects: classes, eigenclasses, and mod-
7 ules. A class defines methods shared by objects of the same class. An eigenclass is a special class
8 which defines methods for only a single object. A module defines, and provides methods to be
9 included into a class or another module. All these three constructs are represented as objects,
10 which are dynamically created and modified at run-time.

11 A class creates objects, and the created objects are called **direct instances** of the class. A
12 class defines a set of methods which can be invoked on all the instances of the class. These
13 methods are instance methods of the class. A class is itself an object, and created by a class
14 definition (see §13.2.2). A class has two sets of variable bindings besides a set of bindings of
15 instance variables. The one is a set of bindings of constants. The other is a set of bindings of
16 class variables, which represents the state shared by all the instances of the class.

17 Every object, including classes, can be associated with at most one special class to the object.
18 This special class is called the eigenclass of the object. The eigenclass defines methods which can
19 be invoked on that object. Those methods are singleton methods of the object. If the object is
20 not a class, the singleton methods of the object can be invoked on only that object. If the object
21 is a class, the singleton methods of the class are similar to so-called class methods because they
22 can be invoked on only that class and its subclasses. An eigenclass is created, and associated
23 with an object by an eigenclass definition (see §13.4.2) or a singleton method definition (see
24 §13.4.3).

25 A class has a single class or `nil` as its **direct superclass**. If a class *A* has a class *B* as its direct
26 superclass, *A* is called a **direct subclass** of *B*. Classes form a tree-like hierarchy defined by the
27 direct superclass-subclass relation. There is only one class which has `nil` as its direct superclass.
28 It is the root of the tree. All the ancestors of a class in the tree are called **superclasses** of the
29 class. All the descendants of a class in the tree are called **subclasses** of the class. A class inherits
30 constants, class variables, singleton methods, and instance methods from its superclasses, if any
31 (see §13.2.3). If an object *C* is a direct instance of a class *D*, *C* is called an instance of *D* and
32 all its superclasses.

33 Ruby does not support multiple inheritance; that is, a class can have only one direct superclass.
34 However, Ruby supports module inclusion, which is a mechanism to append features into a class
35 from multiple sources.

36 A module is an object which has the same structure as a class except that it cannot create an
37 instance of itself and cannot be inherited. As with classes, a module has a set of class variables
38 and instance methods. Instance methods and class variables defined in a module can be used
39 by other classes, modules and eigenclasses by including the module into them. While a class
40 can have only one direct superclass, a class or a module can include multiple modules. Instance
41 methods defined in a module can be invoked on an instance of a class which includes the module.
42 A module is created by a module definition (see §13.1.2).

43 Objects are created at some time during program execution. The lifetime of an object begins
44 when the object is created and ends when all references to it are no longer possible.

6.2 Boolean values

An object is classified into either a **true value** or a **false value**.

Only **false** and **nil** are false values. The pseudo variable **false** is the only instance of the class **FalseClass**, and is represented by the keyword **false**. The pseudo variable **nil** is the only instance of the class **NilClass**, and represented by the keyword **nil**.

Objects other than **false** and **nil** are classified into true values. The pseudo variable **true** is the only instance of the class **TrueClass**, and represented by the keyword **true**.

7 Execution context

7.1 Contextual attributes

An **execution context** is a set of attributes which affects an evaluation of a program.

An execution context is not a part of the language. It is defined in this document only for the description of the semantics of a program. A conforming processor shall evaluate a program as if it acted upon an execution context in the manner described in this document.

An execution context consists of a set of attributes as described below. Each attribute of an execution context except `[[global-variable-bindings]]` forms a logical stack. The names of attributes are enclosed in double square brackets “`[[`” and “`]]`”. Attributes of an execution context are changed when a program construct is evaluated.

The following are the attributes of an execution context:

`[[self]]`: A logical stack of objects, the top of which is the object to which the pseudo variable **self** is bound (see §11.4.3.7.3). The object at the top of the stack is called the **current self**.

`[[class-module-list]]`: A logical stack of lists of classes or modules. The class or module at the head of the list which is on the top of the stack is called the **current class or module**.

`[[default-method-visibility]]`: A logical stack of visibilities of methods, each of which is one of the **public**, **private**, and **protected** visibility. The top of the stack is called the **current visibility**.

`[[local-variable-bindings]]`: A logical stack of sets of bindings of local variables. The element at the top of the stack is called the **current set of local variable bindings**. A set of bindings is pushed onto the stack on every entry into a local variable scope (see §9.1.1), and the top element is removed from the stack on every exit from the scope. The scope with which an element in the stack is associated is called the **scope of the set of local variable bindings**.

`[[invoked-method-name]]`: A logical stack of names by which methods are invoked.

`[[defined-method-name]]`: A logical stack of names with which the invoked methods are defined.

1 [[block]] : A logical stack of blocks passed to method invocations. An element of the stack
2 may be **block-not-given**, which indicates that no block is passed to a method invocation.

3 [[global-variable-bindings]] : A set of bindings of global variables.

4 The term **unset** is used to describe the state of an attribute which is set to nothing.

5 7.2 The initial state

6 Immediately prior to an execution of a program, the attributes of the execution context is
7 initialized as follows:

8 a) Create an empty set of variable bindings, and set [[global-variable-bindings]] to the set of
9 variable bindings.

10 b) Create built-in classes and modules as described in §15.

11 c) Create an empty stack for each attribute of the execution context except [[global-variable-
12 bindings]].

13 d) Create a direct instance of the class **Object** and push it onto [[self]].

14 e) Create a list containing only the class **Object** and push the list onto [[class-module-list]].

15 f) Push the private visibility onto [[default-visibility]].

16 g) Push block-not-given onto [[block]].

17 8 Lexical structure

18 When several prefixes of the input under parsing process have matching productions, the pro-
19 duction that matches the longest prefix is selected.

20 8.1 Source text

21 Syntax

22 *source-character* ::

23 [any character in ISO/IEC 646]

24 A program is represented as a sequence of characters. A conforming processor shall accept
25 any conforming program which consists of characters in ISO/IEC 646, encoded with the octet
26 values as specified in ISO/IEC 646. The support for any other character sets and encodings is
27 implementation dependent.

28 Terminal symbols are sequences of those characters in ISO/IEC 646. Control characters in
29 ISO/IEC 646 are represented by hexadecimal notation.

30 EXAMPLE “0x0a” represents a line feed character.

1 8.2 Line terminators

2 Syntax

3 *line-terminator* ::
4 0x0d? 0x0a

5 A *line-terminator* is ignored when it is used to separate *tokens*. For this reason, except in §8.4
6 and §8.5, *line-terminators* are omitted from productions. However, in some cases, the presence
7 or absence of a *line-terminator* changes the meaning of a program.

8 A location of program text where a *line-terminator* shall occur is indicated by the notation “[
9 *line-terminator* here]”. A location of program text where a *line-terminator* shall not occur is
10 indicated by the notation “[no *line-terminator* here]”; however, a conforming processor may
11 ignore the notation where the ignorance does not introduce ambiguity.

12 EXAMPLE *statements* are separated by *separators* (see §10.2). The syntax of the *separators* is as
13 follows:

14 *separator* ::
15 ;
16 | [*line-terminator* here]

17 The source

18 x = 1 + 2
19 puts x

20 is therefore separated to two statements `x = 1 + 2` and `puts x` by a line-terminator.

21 The source

22 x =
23 1 + 2

24 is parsed as a single statement `x = 1 + 2` because `x =` is not a valid *statement*. However, the source

25 x
26 = 1 + 2

27 is not a valid Ruby program because a *line-terminator* shall not occur before `=` in a *single-variable-*
28 *assignment-expression*, and `= 1 + 2` is not a valid *statement*. The fact that a *line-terminator* shall not
29 occur before `=` is indicated in the syntax of the *single-variable-assignment-expression* as follows (see
30 §11.3.1.1.1):

1 *single-variable-assignment-expression* ::
2 *variable* [no *line-terminator* here] = *operator-expression*

3 8.3 Whitespace

4 Syntax

5 *whitespace* ::
6 0x09 | 0x0b | 0x0c | 0x0d | 0x20 | \ 0x0d? 0x0a
7

8 *whitespace* is ignored when it is used to separate *tokens*. For this reason, except in §8.4 and
9 §8.5, *whitespace* is omitted from productions. However, in some cases, the presence or absence
10 of *whitespace* changes the meaning of a program.

11 A location of program text where *whitespace* shall occur is indicated by the notation “[*whitespace*
12 here]”. A location of program text where *whitespace* shall not occur is indicated by the notation
13 “[no *whitespace* here]”. A *line-terminator* shall not occur in the location where *whitespace* shall
14 not occur. Therefore, this notation also indicates that a *line-terminator* shall not occur.

15 8.4 Comments

16 Syntax

17 *comment* ::
18 *single-line-comment*
19 | *multi-line-comment*

20 *single-line-comment* ::
21 # *comment-content*?

22 *comment-content* ::
23 *line-content*

24 *line-content* ::
25 *source-character* +

26 *multi-line-comment* ::
27 *multi-line-comment-begin-line* *multi-line-comment-line*?
28 *multi-line-comment-end-line*

29 *multi-line-comment-begin-line* ::
30 [beginning of a line] = **begin** *rest-of-begin-end-line*? *line-terminator*

```

1  multi-line-comment-end-line ::
2      [ beginning of a line ] =end rest-of-begin-end-line?
3      ( line-terminator | [ end of a program ] )

```

```

4  rest-of-begin-end-line ::
5      whitespace + comment-content

```

```

6  line ::
7      comment-content line-terminator

```

```

8  multi-line-comment-line ::
9      line but not multi-line-comment-end-line

```

10 The notation “[beginning of a line]” indicates the beginning of a program or the position
11 immediately after a *line-terminator*.

12 Any characters that are considered as *line-terminators* are not allowed within a *line-content*.

13 A *comment* is either a *single-line-comment* or a *multi-line-comment*. A *comment* is considered
14 to be *whitespace*.

15 A *single-line-comment* begins with “#” and continues to the end of the line. A *line-terminator*
16 at the end of the line is not considered to be a part of the comment. A *single-line-comment* can
17 contain any characters except *line-terminators*.

18 A *multi-line-comment* begins with a line beginning with =begin, and continues until and in-
19 cluding a line that begins with =end. Unlike *single-line-comments*, a *line-terminator* on a
20 *multi-line-comment-end-line*, if any, is considered to be part of the comment.

21 8.5 Tokens

22 Syntax

```

23  token ::
24      reserved-word
25      | identifier
26      | punctuator
27      | operator
28      | literal

```

29 8.5.1 Reserved words

30 Syntax

```

31  reserved-word ::
32      __LINE__ | __ENCODING__ | __FILE__ | BEGIN | END | alias | and | begin

```

1		break		case		class		def		defined?		do		else		elsif		end				
2		ensure		for		false		if		in		module		next		nil		not		or		redo
3		rescue		retry		return		self		super		then		true		undef		unless				
4		until		when		while		yield														

5 Reserved words are case-sensitive.

6 8.5.2 Identifiers

7 Syntax

```

8  identifier ::
9      local-variable-identifier
10     | global-variable-identifier
11     | class-variable-identifier
12     | instance-variable-identifier
13     | constant-identifier
14     | method-identifier

15  local-variable-identifier ::
16      ( lowercase-character | _ ) identifier-character*

17  global-variable-identifier ::
18      $ identifier-start-character identifier-character*

19  class-variable-identifier ::
20      @@ identifier-start-character identifier-character*

21  instance-variable-identifier ::
22      @ identifier-start-character identifier-character*

23  constant-identifier ::
24      uppercase-character identifier-character*

25  method-identifier ::
26      method-only-identifier
27     | assignment-like-method-identifier
28     | constant-identifier
29     | local-variable-identifier

30  method-only-identifier ::
31      ( constant-identifier | local-variable-identifier ) ( ! | ? )

32  assignment-like-method-identifier ::
33      ( constant-identifier | local-variable-identifier ) =

```

```

1  identifier-character ::
2      lowercase-character
3      | uppercase-character
4      | decimal-digit
5      | -

6  identifier-start-character ::
7      lowercase-character
8      | uppercase-character
9      | -

10 uppercase-character ::
11     A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R
12     | S | T | U | V | W | X | Y | Z

13 lowercase-character ::
14     a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r
15     | s | t | u | v | w | x | y | z

16 decimal-digit ::
17     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

18 An *identifier* is a sequence of *identifier-characters* optionally prefixed by one of “\$”, “@@”, or
19 “@”, and optionally postfixed by one of “?”, “!”, or “=”.

20 A *global-variable-identifier* begins with “\$”. A *class-variable-identifier* starts with “@@”. An
21 *instance-variable-identifier* begins with “@”. A *constant-identifier* begins with an *uppercase-*
22 *character*.

23 A *local-variable-identifier* begins with a *lowercase-character* or “_”. A *method-identifier* is a
24 *constant-identifier* or a *local-variable-identifier* optionally followed by one of “?”, “!”, or “=”.

25 8.5.3 Punctuators

26 Syntax

```

27 punctuator ::
28     [ | ] | ( | ) | { | } | :: | , | ; | .. | ... | ? | : | =>

```

29 8.5.4 Operators

30 Syntax

```

31 operator ::
32     operator-method-name

```

```

1      | assignment-operator

2  operator-method-name ::
3      ^ | & | | | <=> | == | === | !~ | =~ | > | >= | < | <= | << | >> | +
4      | - | * | / | % | ** | ~ | +@ | -@ | [] | []= | '

5  assignment-operator ::
6      assignment-operator-name =

7  assignment-operator-name ::
8      + | - | * | ** | / | ^ | % | << | >> | & | && | || | |

```

9 8.5.5 Literals

```

10  literal ::
11      numeric-literal
12      | string-literal
13      | array-literal
14      | regular-expression-literal
15      | symbol

```

16 8.5.5.1 Numeric literals

17 Syntax

```

18  numeric-literal ::
19      signed-number
20      | unsigned-number

21  unsigned-number ::
22      integer-literal
23      | float-literal

24  integer-literal ::
25      decimal-integer-literal
26      | binary-integer-literal
27      | octal-integer-literal
28      | hexadecimal-integer-literal

29  decimal-integer-literal ::
30      digit-decimal-integer-literal
31      | prefixed-decimal-integer-literal

```

```

1  digit-decimal-integer-literal ::
2      0
3      | decimal-digit-without-zero ( _? decimal-digit )*

4  prefixed-decimal-integer-literal ::
5      0 ( d | D ) digit-decimal-part

6  digit-decimal-part ::
7      decimal-digit ( _? decimal-digit )*

8  binary-integer-literal ::
9      0 ( b | B ) binary-digit ( _? binary-digit )*

10 octal-integer-literal ::
11  0 ( _ | o | O )? octal-digit ( _? octal-digit )*

12 hexadecimal-integer-literal ::
13  0 ( x | X ) hexadecimal-digit ( _? hexadecimal-digit )*

14 float-literal ::
15     decimal-float-literal
16     | exponent-float-literal

17 decimal-float-literal ::
18     digit-decimal-integer-literal . digit-decimal-part

19 exponent-float-literal ::
20     base-part exponent-part

21 base-part ::
22     decimal-float-literal
23     | digit-decimal-integer-literal

24 exponent-part ::
25     ( e | E ) ( + | - )? digit-decimal-part

26 signed-number ::
27     ( + | - ) unsigned-number

28 decimal-digit-without-zero ::
29     1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

30 octal-digit ::
31     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

```

1 *binary-digit* ::
2 0 | 1

3 *hexadecimal-digit* ::
4 *decimal-digit* | a | b | c | d | e | f | A | B | C | D | E | F

5 Semantics

6 A *numeric-literal* evaluates to either an instance of the class **Integer** or a direct instance of the
7 class **Float**.

8 An *unsigned-number* of the form *integer-literal* evaluates to an instance of the class **Integer**
9 whose value is the value of one of the alternatives on the right-hand side.

10 An *unsigned-number* of the form *float-literal* evaluates to a direct instance of the class **Float**
11 whose value is the value of one of the alternatives on the right-hand side.

12 A *signed-number* which begins with “+” evaluates to an instance represented by the *unsigned-*
13 *number*. A *signed-number* which begins with “-” evaluates to an instance of the class **Integer** or
14 a direct instance of the class **Float** whose value is the negated value of the instance represented
15 by the *unsigned-number*.

16 The value of an *integer-literal*, a *decimal-integer-literal*, a *float-literal*, or a *base-part* is the value
17 of one of the alternatives on the right-hand side.

18 The value of a *digit-decimal-integer-literal* is either 0 or the value of a sequence of characters,
19 which consist of a *decimal-digit-without-zero* followed by sequence of *decimal-digits*, ignoring
20 interleaving “_”s, computed using base 10.

21 The value of a *prefixed-decimal-integer-literal* is the value of the *digit-decimal-part*.

22 The value of a *digit-decimal-part* is the value of the sequence of *decimal-digits*, ignoring inter-
23 leaving “_”s, computed using base 10.

24 The value of a *binary-integer-literal* is the value of the sequence of *binary-digits*, ignoring inter-
25 leaving “_”s, computed using base 2.

26 The value of an *octal-integer-literal* is the value of the sequence of *octal-digits*, ignoring inter-
27 leaving “_”s, computed using base 8.

28 The value of a *hexadecimal-integer-literal* is the value of the sequence of *hexadecimal-digits*,
29 ignoring interleaving “_”s, computed using base 16.

30 The value of a *decimal-float-literal* is the value of the *digit-decimal-integer-literal* plus the value
31 of the *digit-decimal-part* times 10^{-n} where n is the number of *decimal-digits* of the *digit-decimal-*
32 *part*.

33 The value of an *exponent-float-literal* is the value of the *base-part* times 10^n where n is the value
34 of the *exponent-part*.

35 The value of an *exponent-part* is the negative value of the *digit-decimal-part* if “-” occurs,

1 otherwise, it is the value of the *digit-decimal-part*.

2 There is no limitation on the maximum magnitude for the value of an *integer-literal*. The
3 precision of the value of a *float-literal* is implementation defined; however, if the underlying
4 platform of a conforming processor supports IEC 60559:1989, the representation of an instance
5 of the class `Float` should be the 64-bit double format as specified in §3.2.2 of IEC 60559:1989.
6 The value of a float-literal is rounded to fit in the representation of an instance of the class `Float`
7 in an implementation defined way.

8 8.5.5.2 String literals

9 Syntax

10 *string-literal* ::
11 *single-quoted-string*
12 | *double-quoted-string*
13 | *quoted-non-expanded-literal-string*
14 | *quoted-expanded-literal-string*
15 | *here-document*
16 | *external-command-execution*

17 Semantics

18 A *string-literal* evaluates to a direct instance of the class `String`.

19 8.5.5.2.1 Single quoted strings

20 Syntax

21 *single-quoted-string* ::
22 ' *single-quoted-string-character** '

23 *single-quoted-string-character* ::
24 *non-escaped-single-quoted-string-character*
25 | *single-quoted-escape-sequence*

26 *single-quoted-escape-sequence* ::
27 *single-escape-character-sequence*
28 | *non-escaped-single-quoted-string-character-sequence*

29 *single-escape-character-sequence* ::
30 \ *single-escaped-character*

31 *non-escaped-single-quoted-string-character-sequence* ::
32 \ *non-escaped-single-quoted-string-character*

1 *single-escaped-character* ::
2 ' | \

3 *non-escaped-single-quoted-string-character* ::
4 *source-character* **but not** *single-escaped-character*

5 Semantics

6 A *single-quoted-string* consists of zero or more characters enclosed by single quotes. The sequence
7 of *single-quoted-string-characters* within the pair of single quotes represents the content of a
8 string as it occurs in program text literally, except for *single-escape-character-sequences*. The
9 sequence “\\” represents “\”. The sequence “\’” represents “’”.

10 8.5.5.2.2 Double quoted strings

11 Syntax

12 *double-quoted-string* ::
13 " *double-quoted-string-character** "

14 *double-quoted-string-character* ::
15 *source-character* **but not** (" | \)
16 | *double-escape-sequence*
17 | *interpolated-character-sequence*

18 *double-escape-sequence* ::
19 *simple-escape-sequence*
20 | *non-escaped-sequence*
21 | *line-terminator-escape-sequence*
22 | *octal-escape-sequence*
23 | *hex-escape-sequence*
24 | *control-escape-sequence*

25 *simple-escape-sequence* ::
26 \ *double-escaped-character*

27 *non-escaped-sequence* ::
28 \ *non-escaped-double-quoted-string-character*

29 *line-terminator-escape-sequence* ::
30 \ *line-terminator*

31 *non-escaped-double-quoted-string-character* ::
32 *source-character* **but not** (*double-escaped-character* | *line-terminator*)

```

1  double-escaped-character ::
2      \ | n | t | r | f | v | a | e | b | s

```

```

3  octal-escape-sequence ::
4      \ octal-digit ( octal-digit octal-digit? )?

```

```

5  hex-escape-sequence ::
6      \ x hexadecimal-digit hexadecimal-digit?

```

```

7  control-escape-sequence ::
8      \ ( C - | c ) control-escaped-character

```

```

9  control-escaped-character ::
10     double-escape-sequence
11     | ?
12     | source-character but not ( \ | ? )

```

```

13 interpolated-character-sequence ::
14     # global-variable-identifier
15     | # class-variable-identifier
16     | # instance-variable-identifier
17     | # { compound-statement }

```

18 Semantics

19 A *double-quoted-string* consists of zero or more characters enclosed by double quotes. The sequence of *double-quoted-string-characters* within the pair of double quotes represents the content of a string.

22 Except for a *double-escape-sequence* and an *interpolated-character-sequence*, a *double-quoted-string-character* represents a character as it occurs in program text.

24 A *simple-escape-sequence* represents a character as shown in Table 1.

25 An *octal-escape-sequence* represents a character the code of which is the value of the sequence of *octal-digits* computed using base 8.

27 A *hex-escape-sequence* represents a character the code of which is the value of the sequence of *hexadecimal-digits* computed using base 16.

29 A *non-escaped-sequence* represents a *non-escaped-double-quoted-string-character*.

30 A *line-terminator-escape-sequence* is used to break the content of a string into separate lines in program text without inserting a *line-terminator* into the string. A *line-terminator-escape-sequence* does not count as a character of the string.

33 A *control-escape-sequence* represents a character the code of which is computed by performing a bitwise AND operation between 0x9f and the code of the character represented by the *control-*

Table 1 – Simple escape sequences

Escape sequence	Character code
\\	0x5c
\n	0x0a
\t	0x09
\r	0x0d
\f	0x0c
\v	0x0b
\a	0x07
\e	0x1b
\b	0x08
\s	0x20

1 *escaped-character*, except when the *control-escaped-character* is `?`, in which case, the *control-*
2 *escape-sequence* represents a character the code of which is 127.

3 An *interpolated-character-sequence* is a part of a *string-literal* which is dynamically evalu-
4 ated when the *string-literal* in which it is embedded is evaluated. The *interpolated-character-*
5 *sequences* within a *string-literal* are evaluated in the order in which they occur in program
6 text.

7 The value of a *string-literal* which contains *interpolated-character-sequences* is a direct instance
8 of the class **String** the content of which is made from the *string-literal* where each occurrence
9 of *interpolated-character-sequence* is replaced by the content of an instance of the class **String**
10 which is the dynamically evaluated value of the *interpolated-character-sequence*.

11 An *interpolated-character-sequence* is evaluated as follows:

- 12 a) If it is of the form `# global-variable-identifier`, evaluate the *global-variable-identifier* (see
13 §11.4.3.3). Let *V* be the resulting value.
- 14 b) If it is of the form `# class-variable-identifier`, evaluate the *class-variable-identifier* (see
15 §11.4.3.4). Let *V* be the resulting value.
- 16 c) If it is of the form `# instance-variable-identifier`, evaluate the *instance-variable-identifier*
17 (see §11.4.3.5). Let *V* be the resulting value.
- 18 d) If it is of the form `# { compound-statement }`, evaluate the *compound-statement* (see §10.2).
19 Let *V* be the resulting value.
- 20 e) If *V* is an instance of the class **String**, *V* is the value of *interpolated-character-sequence*.
- 21 f) Otherwise, invoke the method `to_s` on *V* with an empty list of arguments. Let *S* be the
22 resulting value.
- 23 g) If *S* is an instance of the class **String**, *S* is the value of *interpolated-character-sequence*.

- 1 h) Otherwise, the value of *interpolated-character-sequence* is an instance of the class **String**,
2 the content of which is implementation defined.

3 8.5.5.2.3 Quoted non-expanded literal strings

4 Syntax

5 *quoted-non-expanded-literal-string* ::
6 %q *literal-beginning-delimiter* *non-expanded-literal-string** *literal-ending-delimiter*

7 *non-expanded-literal-string* ::
8 *non-expanded-literal-character*
9 | *non-expanded-delimited-string*

10 *non-expanded-delimited-string* ::
11 *literal-beginning-delimiter* *non-expanded-literal-string** *literal-ending-delimiter*

12 *non-expanded-literal-character* ::
13 *non-escaped-literal-character*
14 | *non-expanded-literal-escape-sequence*

15 *non-escaped-literal-character* ::
16 *source-character* **but not** *quoted-literal-escape-character*

17 *non-expanded-literal-escape-sequence* ::
18 *non-expanded-literal-escape-character-sequence*
19 | *non-escaped-non-expanded-literal-character-sequence*

20 *non-expanded-literal-escape-character-sequence* ::
21 \ *non-expanded-literal-escaped-character*

22 *non-expanded-literal-escaped-character* ::
23 *literal-beginning-delimiter*
24 | *literal-ending-delimiter*
25 | \

26 *quoted-literal-escape-character* ::
27 *non-expanded-literal-escaped-character*

28 *non-escaped-non-expanded-literal-character-sequence* ::
29 \ *non-escaped-non-expanded-literal-character*

30 *non-escaped-non-expanded-literal-character* ::
31 *source-character* **but not** *non-expanded-literal-escaped-character*

1 The *literal-beginning-delimiter* of a *non-expanded-delimited-string* shall be the same character
2 as the *literal-beginning-delimiter* of the *quoted-non-expanded-literal-string*.

3 A *literal-ending-delimiter* shall be the same character as the corresponding *literal-beginning-*
4 *delimiter*, except when the *literal-beginning-delimiter* is one of the characters on the left in
5 Table 2. In that case, the *literal-ending-delimiter* is the corresponding character on the right in
6 Table 2.

Table 2 – Matching *literal-beginning-delimiter* *literal-ending-delimiter*

<i>literal-beginning-delimiter</i>	<i>literal-ending-delimiter</i>
{	}
()
[]
<	>

7 The production *non-expanded-delimited-string* applies only when the *literal-beginning-delimiter*
8 is one of the characters of *matching-literal-beginning-delimiter*.

9 Semantics

10 A *non-expanded-literal-string* represents the content of a string as it occurs in program text
11 literally, except for *non-expanded-literal-escape-character-sequences*.

12 A *non-expanded-literal-escape-character-sequence* represents a character as follows. The se-
13 quence “\” represents “\”; the sequence \ *literal-beginning-delimiter*, a *literal-beginning-delimiter*;
14 the sequence \ *literal-ending-delimiter*, a *literal-ending-delimiter*.

15 8.5.5.2.4 Quoted expanded literal strings

16 Syntax

17 *quoted-expanded-literal-string* ::
18 % Q? *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

19 *expanded-literal-string* ::
20 *expanded-literal-character*
21 | *expanded-delimited-string*

22 *expanded-literal-character* ::
23 *non-escaped-literal-character*
24 | *double-escape-sequence*
25 | *interpolated-character-sequence*

26 *expanded-delimited-string* ::
27 *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

28 *literal-beginning-delimiter* ::
29 *source-character* **but not** *alpha-numeric-character-or-separator*

```

1  alpha-numeric-character-or-separator ::
2      whitespace
3      | line-terminator
4      | uppercase-character
5      | lowercase-character
6      | decimal-digit

7  literal-ending-delimiter ::
8      [ depending on the literal-beginning-delimiter ]

9  matching-literal-beginning-delimiter ::
10     ( | { | < | [

```

11 The *literal-beginning-delimiter* of an *expanded-delimited-string* shall be the same character as
12 the *literal-beginning-delimiter* of the *quoted-expanded-literal-string*.

13 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.5.5.2.3.

14 The production *expanded-delimited-string* applies only when the *literal-beginning-delimiter* is
15 one of the characters of *matching-literal-beginning-delimiter*.

16 Semantics

17 A *expanded-literal-string* represents the content of a string.

18 A character in an *expanded-literal-string* other than a *double-escape-sequence* or an *interpolated-*
19 *character-sequence* represents a character as it occurs in program text. A *double-escape-sequence*
20 and an *interpolated-character-sequence* represent characters as described in §8.5.5.2.2.

21 8.5.5.2.5 Here documents

22 Syntax

```

23 here-document ::
24     heredoc-start-line heredoc-body heredoc-end-line

25 heredoc-start-line ::
26     heredoc-signifier rest-of-line

27 heredoc-signifier ::
28     << heredoc-delimiter-specifier

29 rest-of-line ::
30     line-content? line-terminator

31 heredoc-body ::
32     heredoc-body-line*

```

1 *heredoc-body-line* ::
2 *line* **but not** *heredoc-end-line*

3 **Semantics**

4 A *here-document* is represented by several lines of program text, and evaluates to a direct
5 instance of the class **String** or the value of the invocation of the method ‘.

6 The *heredoc-signifier*, the *heredoc-body*, and the *heredoc-end-line* in a *here-document* are treated
7 as a unit and considered to be a single token occurring at the place where the *heredoc-signifier*
8 occurs. The first character of the *rest-of-line* becomes the head of the input after the *here-*
9 *document* has been processed.

10 The object to which *here-document* evaluates is either a direct instance *S* of the class **String**
11 whose content is represented by the *heredoc-body* or the value of the invocation of the method
12 ‘ with *S* as the only argument.

13 The form of the *heredoc-delimiter-specifier* determines both the form of the *heredoc-end-line* and
14 the way in which the *here-document* is processed, as described below.

15 **Syntax**

16 *heredoc-delimiter-specifier* ::
17 -? *heredoc-delimiter*

18 *heredoc-delimiter* ::
19 *non-quoted-delimiter*
20 | *single-quoted-delimiter*
21 | *double-quoted-delimiter*
22 | *command-quoted-delimiter*

23 *non-quoted-delimiter* ::
24 *non-quoted-delimiter-identifier*

25 *non-quoted-delimiter-identifier* ::
26 *identifier-character**

27 *single-quoted-delimiter* ::
28 ‘ *single-quoted-delimiter-identifier** ’

29 *single-quoted-delimiter-identifier* ::
30 *source-character* **but not** ’

31 *double-quoted-delimiter* ::
32 " *double-quoted-delimiter-identifier** "


```

1  double-quoted-delimiter-identifier ::
2      source-character but not "

3  command-quoted-delimiter ::
4      ' command-quoted-delimiter-identifier* '

5  command-quoted-delimiter-identifier ::
6      source-character but not '

7  heredoc-end-line ::
8      indented-heredoc-end-line
9      | non-indented-heredoc-end-line

10 indented-heredoc-end-line ::
11     [ beginning of a line ] whitespace* heredoc-delimiter-identifier line-terminator

12 non-indented-heredoc-end-line ::
13     [ beginning of a line ] heredoc-delimiter-identifier line-terminator

14 heredoc-delimiter-identifier ::
15     non-quoted-delimiter-identifier
16     | single-quoted-delimiter-identifier
17     | double-quoted-delimiter-identifier
18     | command-quoted-delimiter-identifier

```

19 Semantics

20 The form of a *heredoc-end-line* depends on the presence or absence of the beginning “-” of the
21 *heredoc-delimiter-specifier*.

22 If the *heredoc-delimiter-specifier* begins with “-”, a line of the form *indented-heredoc-end-line*
23 is treated as the *heredoc-end-line*, otherwise, a line of the form *non-indented-heredoc-end-line*
24 is treated as the *heredoc-end-line*. In both forms, the *heredoc-delimiter-identifier* shall be the
25 same sequence of characters as it occurs in the corresponding part of *heredoc-delimiter*.

26 If the *heredoc-delimiter* is of the form *non-quoted-delimiter*, the *heredoc-delimiter-identifier* shall
27 be the same sequence of characters as the *non-quoted-delimiter-identifier*; if it is of the form
28 *single-quoted-delimiter*, the *single-quoted-delimiter-identifier*; if it is of the form of *double-quoted-*
29 *delimiter*, the *double-quoted-delimiter-identifier*; if it is of the form of *command-quoted-delimiter*,
30 the *command-quoted-delimiter-identifier*.

31 The object to which a *here-document* evaluates is created as follows:

32 a) Create a direct instance of the class **String** from the *heredoc-body*, the treatment of which
33 depends on the form of the *heredoc-delimiter* as follows:

- 34 • If *heredoc-delimiter* is of the form *single-quoted-delimiter*, the *heredoc-body* is treated
35 as a sequence of *source-characters* as it occurs in program text literally.

- If *heredoc-delimiter* is in any of the forms *non-quoted-delimiter*, *double-quoted-delimiter*, or *command-quoted-delimiter*, the *heredoc-body* is treated as a sequence of *double-quoted-string-characters* as described in §8.5.5.2.2.

Let S be that instance of the class **String**.

- b) If the *heredoc-delimiter* is not of the form *command-quoted-delimiter*, let V be S .
- c) Otherwise, invoke the method ‘ on the current self with the list of arguments whose only element is S . Let V be the resulting value of the method invocation.
- d) V is the object to which the *here-document* evaluates.

8.5.5.2.6 External command execution

Syntax

```
external-command-execution ::
    backquoted-external-command-execution
    | quoted-external-command-execution
```

```
backquoted-external-command-execution ::
    ‘ double-quoted-string-character* ‘
```

```
quoted-external-command-execution ::
    %x literal-beginning-delimiter expanded-literal-string* literal-ending-delimiter
```

The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.5.5.2.

Semantics

An *external-command-execution* is a form to invoke the method “‘”.

An *external-command-execution* is evaluated as follows:

- a) If the *external-command-execution* is of the form *backquoted-external-command-execution*, construct a direct instance of the class **String** S by replacing the two “‘” with “”” and evaluating the resulting *double-quoted-string* as described in §8.5.5.2.2.
- b) If the *external-command-execution* is of the form *quoted-external-command-execution*, construct a direct instance of the class **String** S by replacing “%x” with “%Q” and evaluating the resulting *quoted-expanded-literal-string* as described in §8.5.5.2.4.
- c) Invoke the method “‘” on the current self with a list of arguments whose only element is S .
- d) The resulting value is the value of the *external-command-execution*.

8.5.5.3 Array literals

Syntax

```

1  array-literal ::
2      quoted-non-expanded-array-constructor
3      | quoted-expanded-array-constructor

4  quoted-non-expanded-array-constructor ::
5      %w literal-beginning-delimiter non-expanded-array-content literal-ending-delimiter

6  non-expanded-array-content ::
7      quoted-array-item-separator-list? non-expanded-array-item-list?
8      quoted-array-item-separator-list?

9  non-expanded-array-item-list ::
10     non-expanded-array-item ( quoted-array-item-separator-list non-expanded-array-item )*

11 quoted-array-item-separator-list ::
12     quoted-array-item-separator +

13 quoted-array-item-separator ::
14     whitespace
15     | line-terminator

16 non-expanded-array-item ::
17     non-expanded-array-item-character +

18 non-expanded-array-item-character ::
19     non-escaped-array-item-character
20     | non-expanded-array-escape-sequence

21 non-escaped-array-item-character ::
22     non-escaped-array-character
23     | matching-literal-delimiter

24 non-escaped-array-character ::
25     non-escaped-literal-character but not quoted-array-item-separator

26 matching-literal-delimiter ::
27     ( | { | < | [ | ) | } | > | ]

28 non-expanded-array-escape-sequence ::
29     non-expanded-literal-escape-sequence but not escaped-quoted-array-item-separator
30     | escaped-quoted-array-item-separator

31 escaped-quoted-array-item-separator ::
32     \ quoted-array-item-separator

```

```

1  quoted-expanded-array-constructor ::
2      %W literal-beginning-delimiter expanded-array-content literal-ending-delimiter

3  expanded-array-content ::
4      quoted-array-item-separator-list? expanded-array-item-list?
5      quoted-array-item-separator-list?

6  expanded-array-item-list ::
7      expanded-array-item ( quoted-array-item-separator-list expanded-array-item )*

8  expanded-array-item ::
9      expanded-array-item-character +

10 expanded-array-item-character ::
11     non-escaped-array-item-character
12     | expanded-array-escape-sequence
13     | interpolated-character-sequence

14 expanded-array-escape-sequence ::
15     double-escape-sequence but not escaped-quoted-array-item-separator
16     | escaped-quoted-array-item-separator

```

17 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.5.5.2.

18 When the *literal-beginning-delimiter* is one of the *matching-literal-beginning-delimiter*, the *quoted-*
19 *non-expanded-array-constructor* and the *quoted-expanded-array-constructor* is determined as fol-
20 lows.

21 Let N be 0. For each character C which appears after “%w” or “%W”, take the following steps.

- 22 a) If C is a *literal-beginning-delimiter* which is not prefixed by a “\”, increment N by 1.
- 23 b) If C is a *literal-ending-delimiter* which is not prefixed by a “\”, decrement N by 1.
- 24 c) If N is 0 and C is the *literal-ending-delimiter*, terminate these steps.

25 The *literal-ending-delimiter* in Step c is the *literal-ending-delimiter* of the *quoted-non-expanded-*
26 *array-constructor* or the *quoted-expanded-array-constructor*.

27 Semantics

28 An *array-literal* evaluates to a direct instance of the class **Array**.

29 A *quoted-non-expanded-array-constructor* is evaluated as follows:

- 30 a) Create an empty direct instance of the class **Array**. Let A be the instance.
- 31 b) If *non-expanded-array-item-list* occurs, for each *non-expanded-array-item* of the *non-expanded-*

array-item-list, take the following steps:

- 1) Create a direct instance of the class **String** *S*, the content of which is represented by the sequence of *non-expanded-array-item-characters*.

A *non-expanded-array-item-character* represents itself, except in the case of a *non-expanded-array-escape-sequence*. A *non-expanded-array-escape-sequence* represents a character as described in §8.5.5.2.3, except in the case of an *escaped-quoted-array-item-separator*. An *escaped-quoted-array-item-separator* represents a *quoted-array-item-separator*.

- 2) Append *S* to *A*.

- c) The value of the *quoted-non-expanded-array-constructor* is *A*.

A *quoted-expanded-array-constructor* is evaluated as follows:

- a) Create an empty direct instance of the class **Array**. Let *A* be the instance.

- b) If *expanded-array-item-list* occurs, process each *expanded-array-item* of the *expanded-array-item-list* as follows:

- 1) Create a direct instance of the class **String** *S*, the content of which is represented by the sequence of *expanded-array-item-characters*.

An *expanded-array-item-character* represents itself, except in the case of an *expanded-array-escape-sequence* and an *interpolated-character-sequence*. An *expanded-array-escape-sequence* represents a character as described in §8.5.5.2.2, except in the case of an *escaped-quoted-array-item-separator*. An *escaped-quoted-array-item-separator* represents a *quoted-array-item-separator*. An *interpolated-character-sequence* represents a sequence of characters as described in §8.5.5.2.2.

- 2) Append *S* to *A*.

- c) The value of the *quoted-expanded-array-constructor* is *A*.

8.5.5.4 Regular expression literals

Syntax

```
regular-expression-literal ::  
    / regular-expression-body / regular-expression-option*  
    | %r literal-beginning-delimiter expanded-literal-string*  
    literal-ending-delimiter regular-expression-option*
```

```
regular-expression-body ::  
    regular-expression-character*
```

```
regular-expression-character ::  
    source-character but not ( / | \ )  
    | \ \
```

1 | *line-terminator-escape-sequence*
2 | *interpolated-character-sequence*

3 *regular-expression-option* ::
4 i | m

5 Within an *expanded-literal-string*, a *literal-beginning-delimiter* shall be the same character as
6 the *literal-beginning-delimiter* of a *regular-expression-literal*.

7 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.5.5.2.3.

8 If a *regular-expression-literal* of the form / *regular-expression-body* / *regular-expression-option**
9 is the first argument (see §11.2.1), the first character of the *regular-expression-body* shall not be
10 *whitespace*.

11 Semantics

12 A *regular-expression-literal* evaluates to a direct instance of the class **Regexp**.

13 The pattern of an instance of the class **Regexp** resulting from a *regular-expression-literal* is the
14 string which *regular-expression-characters* or *expanded-literal-strings* represent. If the string
15 cannot be derived from the *pattern* (see §15.2.15.3), the evaluation of the program shall be
16 terminated and a syntax error shall be reported.

17 A *regular-expression-character* other than the sequence \ \, a *line-terminator-escape-sequence*,
18 or *interpolated-character-sequence* represents themselves. A *expanded-literal-string* other than a
19 *line-terminator-escape-sequence* or *interpolated-character-sequence* represents themselves.

20 The sequence \ \ of *regular-expression-character* represents a single character \.

21 A *line-terminator-escape-sequence* in a *regular-expression-character* and an *expanded-literal-*
22 *string* is ignored in the resulting pattern of an instance of the class **Regexp**.

23 An *interpolated-character-sequence* in a *regular-expression-literal* and an *expanded-literal-string*
24 is evaluated as described in §8.5.5.2.2, and represents a string which is the content of the resulting
25 an instance of the class **String**.

26 A *regular-expression-option* specifies the ignorecase and the multiline properties of an instance
27 of the class **Regexp** resulting from a *regular-expression-literal*. If i occurs in a *regular-expression-*
28 *option*, the ignorecase property of the resulting instance of the class **Regexp** is set to true. If
29 m occurs in a *regular-expression-option*, the multiline property of the resulting instance of the
30 class **Regexp** is set to true.

31 The grammar for a pattern of an instance of the class **Regexp** created from a *regular-expression-*
32 *literal* is described in §15.2.15.

33 8.5.5.5 Symbol literals

34 Syntax

```

1  symbol ::
2      symbol-literal
3      | dynamic-symbol

4  symbol-literal ::
5      : symbol-name

6  dynamic-symbol ::
7      : single-quoted-string
8      | : double-quoted-string
9      | %s literal-beginning-delimiter non-expanded-literal-string* literal-ending-delimiter

10 symbol-name ::
11     method-identifier
12     | operator-method-name
13     | reserved-word
14     | instance-variable-identifier
15     | global-variable-identifier
16     | class-variable-identifier

```

17 *single-quoted-strings*, *double-quoted-strings*, and *non-expanded-literal-strings* shall not contain
18 any sequences which represent the character 0x00.

19 Within a *non-expanded-literal-string*, *literal-beginning-delimiter* shall be the same character as
20 the *literal-beginning-delimiter* of the *dynamic-symbol*.

21 The *literal-ending-delimiter* shall match the *literal-beginning-delimiter* as described in §8.5.5.2.3.

22 Semantics

23 A *symbol* evaluates to a direct instance of the class **Symbol**. A *symbol-literal* evaluates to a direct
24 instance of the class **Symbol** whose name is the *symbol-name*. A *dynamic-symbol* evaluates to a
25 direct instance of the class **Symbol** whose name is the content of an instance of the class **String**
26 which is the value of the *single-quoted-string*(see §8.5.5.2.1), *double-quoted-string*(see §8.5.5.2.2),
27 or *non-expanded-literal-string*(see §8.5.5.2.3).

28 9 Scope of variables

29 A **scope** is a region of a program text with which a set of bindings of variables is associated.

30 9.1 Local variables

31 A local variable is referred to by a *local-variable-identifier*.

32 9.1.1 Scopes of local variables

33 Scopes for local variables are introduced by the following program constructs:

- 1 • *program* (see §10.1)
- 2 • *class-body* (see §13.2.2)
- 3 • *module-body* (see §13.1.2)
- 4 • *eigenclass-body* (see §13.4.2)
- 5 • *method-definition* (see §13.3.1) and *singleton-method-definition* (see §13.4.3), for both of
- 6 which the scope starts with the *method-parameter-part* and continues up to and including
- 7 the *method-body*.
- 8 • *block* (see §11.2.2)

9 Let P be any of the above program constructs. Let S be the region of P excluding all the regions
 10 of any of the above program constructs (except *block*) nested within P . Then, S is the **local**
 11 **variable scope** which corresponds to the program construct P .

12 The scope of a local variable is the local variable scope whose set of local variable bindings
 13 contains the binding of the local variable, which is resolved as described below.

14 When a *local-variable-identifier* which is a reference to a local variable occurs (see §9.1.2), the
 15 binding of the local variable is resolved as follows:

- 16 a) Let N be the *local-variable-identifier*. Let B be the current set of local variable bindings.
- 17 b) Let S be the scope of B .
- 18 c) If a binding with name N exists in B , that binding is the resolved binding.
- 19 d) If a binding with name N does not exist in B :
 - 20 • If S is a local variable scope which corresponds to a *block*:
 - 21 1) If the *local-variable-identifier* occurs as a *left-hand-side* of a *block-formal-argument-*
 - 22 *list*, whether to proceed to the next step or not is implementation defined.
 - 23 2) Replace B with the element immediately below the current B on \llbracket local-variable-
 - 24 *bindings* \rrbracket , and continue searching for a binding with name N from Step b.
 - 25 • Otherwise, a binding is considered not resolved.

26 9.1.2 References to local variables

27 An occurrence of a *local-variable-identifier* can be a reference to a local variable or a method
 28 invocation. In order to determine whether the occurrence of a *local-variable-identifier* is a
 29 reference to a local variable or a method invocation, before the evaluation of a local variable
 30 scope, the scope is scanned sequentially for *local-variable-identifiers*.

31 For each occurrence of a *local-variable-identifier* I , take the following steps:

- 32 a) If I occurs in one of the forms below, I is a reference to a local variable.

- 1 • *mandatory-parameter*
- 2 • *optional-parameter-name*
- 3 • *array-parameter-name*
- 4 • *block-parameter-name*
- 5 • *variable of left-hand-side*
- 6 • *variable of single-variable-assignment-expression*
- 7 • *variable of single-variable-assignment-statement*
- 8 • *variable of abbreviated-variable-assignment-expression*
- 9 • *variable of abbreviated-variable-assignment-statement*

10 b) If I occurs in one of the forms below:

- 11 • *variable of singleton*
- 12 • *variable of primary-expression*

13 and the following condition holds, I is a reference to a local variable.

- 14 • Let P be the point where I occurs and let S be the innermost local variable scope
15 which encloses P and which does not correspond to a *block*. Let R be the region of a
16 program between the beginning of S and P .

17 The same identifier as I occurs as a reference to a local variable in R .

18 c) Otherwise, I is a method invocation.

19 **9.2 Global variables**

20 The scope of global variables is global in the sense that they are accesible everywhere in a Ruby
21 program. Global variable bindings are created in \llbracket global-variable-bindings \rrbracket .

22 **10 Program structure**

23 **10.1 Program**

24 **Syntax**

25 *program* ::
26 *compound-statement*

1 Semantics

2 A *program* is evaluated as follows:

- 3 a) Push an empty set of bindings onto $\llbracket \text{local-variable-bindings} \rrbracket$.
- 4 b) Evaluate the *compound-statement*.
- 5 c) The resulting value is the value of the *program*.
- 6 d) Restore the execution context by removing the element from the top of $\llbracket \text{local-variable-bindings} \rrbracket$, even when an exception is raised and not handled during Step b.

8 10.2 Compound statement

9 Syntax

```
10  compound-statement ::  
11      statement-list? separator-list?  
  
12  statement-list ::  
13      statement ( separator-list statement )*  
  
14  separator-list ::  
15      separator ;*  
  
16  separator ::  
17      ;  
18      | [ line-terminator here ]
```

19 Semantics

20 A *compound-statement* is evaluated as follows:

- 21 a) If the *statement-list* does not occur, the value of the *compound-statement* is **nil**.
- 22 b) If the *statement-list* occurs, evaluate each *statement* of the *statement-list* in the order it
23 appears in the program text.
- 24 c) If one of the *statements* of the *statement-list* is terminated by a *jump-expression*, terminate
25 the evaluation of the *statement-list* immediately. None of the following *statements* of the
26 *statement-list* is evaluated. In this case, the value of the *compound-statement* is undefined.
- 27 d) If none of the *statements* of the *statement-list* is terminated by a *jump-expression*, the value
28 of the *compound-statement* is the value of the last *statement* of the *statement-list*.

1 11 Expressions

2 Syntax

3 *expression* ::
4 *keyword-logical-expression*

5 Semantics

6 See §11.1 for *keyword-logical-expression*.

7 11.1 Logical expressions

8 Syntax

9 *keyword-logical-expression* ::
10 *keyword-NOT-expression*
11 | *keyword-AND-expression*
12 | *keyword-OR-expression*

13 *keyword-NOT-expression* ::
14 *method-invocation-without-parentheses*
15 | *operator-expression*
16 | *logical-NOT-with-method-invocation-without-parentheses*
17 | **not** *keyword-NOT-expression*

18 *logical-NOT-expression* ::=
19 *logical-NOT-with-method-invocation-without-parentheses*
20 | *logical-NOT-with-unary-expression*

21 *logical-NOT-with-method-invocation-without-parentheses* ::
22 **!** *method-invocation-without-parentheses*

23 *logical-NOT-with-unary-expression* ::
24 **!** *unary-expression*

25 *keyword-AND-expression* ::
26 *expression* **and** *keyword-NOT-expression*

27 *keyword-OR-expression* ::
28 *expression* **or** *keyword-NOT-expression*

29 *logical-OR-expression* ::
30 *logical-AND-expression*
31 | *logical-OR-expression* **||** *logical-AND-expression*

1 *logical-AND-expression* ::
2 *equality-expression*
3 | *logical-AND-expression* && *equality-expression*

4 **Semantics**

5 A *logical-NOT-expression* or a *keyword-NOT-expression* of the form **not** *keyword-NOT-expression*
6 is evaluated as follows:

- 7 a) If it is of the form **not** *keyword-NOT-expression*, evaluate the *keyword-NOT-expression*. Let
8 *X* be the resulting value.
- 9 b) If it is a *logical-NOT-expression*, evaluate its *method-invocation-without-parentheses* or
10 *unary-expression*. Let *X* be the resulting value.
- 11 c) If *X* is a true value, the value of the *keyword-NOT-expression* or the *logical-NOT-expression*
12 is **false**.
- 13 d) Otherwise, the value of the *keyword-NOT-expression* or the *logical-NOT-expression* is **true**.

14 Instead of the above process, a conforming processor may evaluate a *logical-NOT-expression* as
15 follows:

- 16 a) Evaluate the *unary-expression* or the *method-invocation-without-parentheses*. Let *V* be the
17 resulting value.
- 18 b) Create an empty list of arguments *L*. Invoke the method **!@** on *V* with *L* as the list of
19 arguments. The resulting value is the value of the *logical-NOT-expression*.

20 In this case, the processor shall:

- 21 • include the operator **!@** in *operator-method-name*.
- 22 • define an instance method **!@** in the class **Object** or one of its superclasses, if any. The
23 method **!@** shall not take any arguments. The method **!@** shall return **true** if the receiver
24 is **false** or **nil**, and shall return **false** otherwise.

25 A *logical-AND-expression* of the form *logical-AND-expression* && *equality-expression* or a *keyword-*
26 *AND-expression* is evaluated as follows:

- 27 a) Evaluate the *expression* or the *logical-AND-expression*. Let *X* be the resulting value.
- 28 b) If *X* is a true value, evaluate the *keyword-NOT-expression* or *equality-expression*. Let
29 *Y* be the resulting value. The value of the *keyword-AND-expression* or the *logical-AND-*
30 *expression* is *Y*.
- 31 c) Otherwise, the value of the *keyword-AND-expression* or the *logical-AND-expression* is *X*.

32 A *keyword-OR-expression* or a *logical-OR-expression* of the form *logical-OR-expression* || *logical-*
33 *AND-expression* is evaluated as follows:

- 1 a) Evaluate the *expression* or the *logical-OR-expression*. Let X be the resulting value.
- 2 b) If X is a false value, evaluate the *keyword-NOT-expression* or the *logical-AND-expression*.
3 Let Y be the resulting value. The value of the *keyword-OR-expression* or *logical-OR-*
4 *expression* is Y .
- 5 c) Otherwise, the value of the *keyword-OR-expression* or *logical-OR-expression* is X .

6 11.2 Method invocation expressions

7 Syntax

```

8  primary-method-invocation ::
9      super-with-optional-argument
10     | indexing-method-invocation
11     | method-only-identifier
12     | method-identifier ( [no whitespace here] argument-with-parentheses )? block?
13     | primary-expression [no line-terminator here]
14         . method-name ( [no whitespace here] argument-with-parentheses )? block?
15     | primary-expression [no line-terminator here]
16         :: method-name [no whitespace here] argument-with-parentheses block?
17     | primary-expression [no line-terminator here] :: method-name-without-constant
18     block?

19  indexing-method-invocation ::
20      primary-expression [no line-terminator here] optional-whitespace?
21      [ indexing-argument-list? ]

22  optional-whitespace ::
23      [ whitespace here ]

24  method-name-without-constant ::
25      method-name but not constant-identifier

26  method-invocation-without-parentheses ::
27      command
28      | chained-command-with-do-block
29      | chained-command-with-do-block ( . | :: ) method-name argument
30      | return-with-argument
31      | break-with-argument
32      | next-with-argument

33  command ::
34      super-with-argument
35      | yield-with-argument
36      | method-identifier argument
37      | primary-expression [no line-terminator here] ( . | :: ) method-name argument

```

```

1  chained-command-with-do-block ::
2      command-with-do-block chained-method-invocation *

3  chained-method-invocation ::
4      ( . | :: ) method-name
5      | ( . | :: ) method-name [no whitespace here]
6      [lookahead  $\notin$  { { } } ] argument-with-parentheses

7  command-with-do-block ::
8      super-with-argument-and-do-block
9      | method-identifier argument do-block
10     | primary-expression [no line-terminator here] ( . | :: ) method-name argument
11     do-block

```

12 The *primary-expression* of a *primary-method-invocation*, *command*, and *indexing-method-invocation*
13 shall not be a *jump-expression*.

14 If the *argument-with-parentheses* of a *primary-method-invocation* occurs, and the *block-argument*
15 of the *argument* of the *argument-with-parentheses* occurs, the *block* of the *primary-method-*
16 *invocation* shall not occur.

17 If the *argument* of a *command-with-do-block* occurs, and the *block-argument* of the *argument-*
18 *in-parentheses* of the *argument* (see §11.2.1) occurs, the *do-block* of the *command-with-do-block*
19 shall not occur.

20 The *optional-whitespace* of an *indexing-method-invocation* shall not occur if its *primary-expression*
21 is any of the following construct:

- 22 • A *primary-method-invocation* of the form *method-only-identifier*, *method-identifier*, *primary-*
23 *expression* . *method-name*, or *primary-expression* :: *method-name-without-constant*
- 24 • A *method-invocation-without-parentheses* of the form *chained-command-with-do-block* which
25 satisfies all of the following conditions:
 - 26 a) Let *M* be the *chained-command-with-do-block*. One or more *chained-method-invocation*
27 of *M* occurs.
 - 28 b) Let *I* be the last *chained-method-invocation* of *M*, in the order they appear in program
29 text. *I* is of the form (. | ::) *method-name*.

30 Semantics

31 A *primary-method-invocation* is evaluated as follows:

- 32 a) If the *primary-method-invocation* is a *super-with-optional-argument* or an *indexing-method-*
33 *invocation*, evaluate it. The resulting value is the value of the *primary-method-invocation*.
- 34 b) • If the *primary-method-invocation* is a *method-only-identifier*, let *O* be the current self
35 and let *M* be the *method-only-identifier*. Create an empty list of arguments *L*.

- 1 • If the *method-identifier* of the *primary-method-invocation* occurs:
 - 2 1) Let O be the current self and let M be the *method-identifier*.
 - 3 2) If the *argument-with-parentheses* occurs, construct a list of arguments and a block
4 from the *argument-with-parentheses* as described in §11.2.1. Let L be the resulting
5 list. Let B be the resulting block, if any.

6 If the *argument-with-parentheses* does not occur, create an empty list of arguments
7 L .
 - 8 3) If the *block* occurs, let B be the *block*.
- 9 • If the $.$ of the *primary-method-invocation* occurs:
 - 10 1) Evaluate the *primary-expression* and let O be the resulting value. Let M be the
11 *method-name*.
 - 12 2) If the *argument-with-parentheses* occurs, construct a list of arguments and a block
13 from the *argument-with-parentheses* as described in §11.2.1. Let L be the resulting
14 list. Let B be the resulting block, if any.

15 If the *argument-with-parentheses* does not occur, create an empty list of arguments
16 L .
 - 17 3) If the *block* occurs, let B be the *block*.
- 18 • If the $::$ and *method-name* of the *primary-method-invocation* occur:
 - 19 1) Evaluate the *primary-expression* and let O be the resulting value. Let M be the
20 *method-name*.
 - 21 2) Construct a list of arguments and a block from the *argument-with-parentheses* as
22 described in §11.2.1. Let L be the resulting list. Let B be the resulting block, if
23 any.
 - 24 3) If the *block* occurs, let B be the *block*.
- 25 • If the $::$ and *method-name-without-constant* of the *primary-method-invocation* occur:
 - 26 1) Evaluate the *primary-expression* and let O be the resulting value. Let M be the
27 *method-name-without-constant*.
 - 28 2) Create an empty list of arguments L .
 - 29 3) If the *block* occurs, let B be the *block*.
- 30 c) Invoke the method M on O with L as the list of arguments and B , if any, as the block. (see
31 §13.3.3). The resulting value is the value of the *primary-method-invocation*.

32 An *indexing-method-invocation* is evaluated as follows:

- 1 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 2 b) If the *indexing-argument-list* occurs, construct a list of arguments from the *indexing-*
3 *argument-list* as described in §11.2.1. Let L be the resulting list.
- 4 c) If the *indexing-argument-list* does not occur, Create an empty list of arguments L .
- 5 d) Invoke the method `[]` on O with L as the list of arguments. The resulting value is the value
6 of the *indexing-method-invocation*.

7 A *method-invocation-without-parentheses* is evaluated as follows:

- 8 • If the *method-invocation-without-parentheses* is a *command*, evaluate it. The resulting value
9 is the value of the *method-invocation-without-parentheses*.
- 10 • If the *method-invocation-without-parentheses* is a *return-with-argument*, *break-with-argument*
11 or *next-with-argument*, evaluate it (see §11.4.1.3).
- 12 • If the *chained-command-with-do-block* of the *method-invocation-without-parentheses* occurs:
 - 13 a) Evaluate the *chained-command-with-do-block*. Let V be the resulting value.
 - 14 b) If the *method-name* and the *argument* of the *method-invocation-without-parentheses*
15 occur:
 - 16 1) Let M be the *method-name*.
 - 17 2) Construct a list of arguments from the *argument* as described in §11.2.1 and let L
18 be the resulting list. If the *block-argument* of the *argument-in-parentheses* of the
19 *argument* occurs, let B be the *block* to which the *block-argument* corresponds.
 - 20 3) Invoke the method M on V with L as the list of arguments and B , if any, as the
21 block.
 - 22 4) Replace V with the resulting value.
 - 23 c) The value of the *method-invocation-without-parentheses* is V .

24 A *command* is evaluated as follows:

- 25 a) If the *command* is a *super-with-argument* or a *yield-with-argument*, evaluate it.
- 26 b) Otherwise:
 - 27 1) If the *method-identifier* of the *command* occurs:
 - 28 i) Let O be the current self and let M be the *method-identifier*.
 - 29 ii) Construct a list of arguments from the *argument* as described in §11.2.1 and let L
30 be the resulting list.
 - 31 If the *block-argument* of the *argument-in-parentheses* of the *argument* occurs, let
32 B be the *block* to which the *block-argument* corresponds.

- 1 2) If the *primary-expression*, *method-name*, and the *argument* of the *command* occurs:
- 2 i) Evaluate the *primary-expression*. Let O be the resulting value. Let M be the
- 3 *method-name*.
- 4 ii) Construct a list of arguments from the *argument* as described in §11.2.1 and let L
- 5 be the resulting list.
- 6 If the *block-argument* of the *argument-in-parentheses* of the *argument* occurs, let
- 7 B be the *block* to which the *block-argument* corresponds.
- 8 3) Invoke the method M on O with L as the list of arguments and B , if any, as the block.
- 9 The resulting value is the value of the *command*.
- 10 A *chained-command-with-do-block* is evaluated as follows:
- 11 a) Evaluate the *command-with-do-block* and let V be the resulting value.
- 12 b) For each *chained-method-invocation*, in the order they appears in the program text, take
- 13 the following steps:
- 14 1) Let M be the *method-name* of the *chained-method-invocation*.
- 15 2) If the *argument-with-parentheses* occurs, construct a list of arguments and a block from
- 16 the *argument-with-parentheses* as described in §11.2.1 and let L be the resulting list.
- 17 Let B be the resulting block, if any.
- 18 If the *argument-with-parentheses* does not occur, create an empty list of arguments L .
- 19 3) Invoke the method M on V with L as the list of arguments and B , if any, as the block.
- 20 4) Replace V with the resulting value.
- 21 c) The value of the *chained-command-with-do-block* is V .
- 22 A *command-with-do-block* is evaluated as follows:
- 23 • If the *command-with-do-block* is a *super-with-argument-and-do-block*, evaluate it. The re-
- 24 sulting value is the value of the *command-with-do-block*.
- 25 • Otherwise:
- 26 a) If the *method-identifier* of the *command* occurs, let O be the current self and let M be
- 27 the *method-name*.
- 28 If the *method-identifier* of the *command* does not occur, evaluate the *primary-expression*,
- 29 let O be the resulting value and let M be the *method-name*.
- 30 b) Construct a list of arguments from the *arguments* of the *command-with-do-block* and
- 31 let L be the resulting list.
- 32 c) Invoke the method M on O with L as the list of arguments and the *do-block* as the
- 33 block. The resulting value is the value of the *command-with-do-block*.

1 11.2.1 Method arguments

2 Syntax

```
3  indexing-argument-list ::
4      command
5      | operator-expression-list ,?
6      | operator-expression-list , splatting-argument
7      | association-list ,?
8      | splatting-argument

9  splatting-argument ::
10     * operator-expression

11  operator-expression-list ::
12     operator-expression ( , operator-expression )*

13  argument-with-parentheses ::
14     ( )
15     | ( argument-in-parentheses )
16     | ( operator-expression-list , chained-command-with-do-block )
17     | ( chained-command-with-do-block )

18  argument ::
19     [no line-terminator here] [lookahead  $\notin$  { { } }] optional-whitespace?
20     argument-in-parentheses

21  argument-in-parentheses ::
22     command
23     | ( operator-expression-list | association-list )
24       ( , splatting-argument )? ( , block-argument )?
25     | operator-expression-list , association-list
26       ( , splatting-argument )? ( , block-argument )?
27     | splatting-argument ( , block-argument )?
28     | block-argument

29  block-argument ::
30     & operator-expression
```

31 The *operator-expression* of a *splatting-argument*, *operator-expression-list*, and *block-argument*
32 shall not be a *jump-expression*.

33 If the *operator-expression-list* of an *argument-in-parentheses* occurs, the first *operator-expression*
34 of the *operator-expression-list* is called the **first argument**.

35 If a *splatting-argument* is the first argument, *whitespaces* shall not occur between its * and

1 *operator-expression*. If a *block-argument* is the first argument, *whitespaces* shall not occur be-
2 tween its *&* and *operator-expression*.

3 If the first argument of an *argument* is other than the following constructs, the *optional-*
4 *whitespace* shall occur.

- 5 • A *variable-reference* of the form *global-variable-identifier*, *class-variable-identifier* or *instance-*
6 *variable-identifier* (see §11.4.3).
- 7 • A *single-quoted-string* or *double-quoted-string* (see §8.5.5.2).
- 8 • A *symbol-literal*, or a *dynamic-symbol* of the form : [no *whitespace* here] *single-quoted-string*
9 or : [no *whitespace* here] *double-quoted-string* (see §8.5.5.5).
- 10 • An *external-command-execution* of the form *backquoted-external-command-execution* (see
11 §8.5.5.2.6).
- 12 • A *scoped-constant-reference* whose *primary-expression* occurs and the *primary-expression*
13 is any of these constructs.
- 14 • A *primary-method-invocation* whose *primary-expression* occurs and the *primary-expression*
15 is any of these constructs.

16 Semantics

17 The list of arguments used for method invocation is constructed from *indexing-argument-list*,
18 *splatting-argument*, *argument-with-parentheses*, or *argument*.

19 An *indexing-argument-list* is processed as follows:

- 20 a) Create an empty list of arguments *L*.
- 21 b) Evaluate the *command*, *operator-expressions* of *operator-expression-lists*, and the *association-*
22 *list* and append their values to *L* in the order they appear in the program text.
- 23 c) If the *splatting-argument* occurs, construct a list of arguments from it and concatenate the
24 resulting list to *L*.

25 A *splatting-argument* is processed as follows:

- 26 a) Create an empty list of arguments *L*.
- 27 b) Evaluate the *operator-expression*. Let *V* be the resulting value.
- 28 c) If *V* is not an instance of the class **Array**, the behavior is implementation dependent.
- 29 d) Append each element of *V*, in the indexing order, to *L*.

30 An *argument-with-parentheses* is processed as follows:

- 31 a) Create an empty list of arguments *L*.

1 b) If the *argument-in-parentheses* occurs, construct a list of arguments from it and concatenate
2 the resulting list to *L*. If *block-argument* of *argument-in-parentheses* occurs, the *block* to
3 which the *block-argument* corresponds is the block which is passed to the method invocation
4 with *L*.

5 c) If the *operator-expression-list* occurs, for each *operator-expression* of the *operator-expression-*
6 *list*, in the order they appears in the program text, take the following steps:

7 1) Evaluate the *operator-expression*. Let *V* be the resulting value.

8 2) Append *V* to *L*.

9 d) If the *chained-command-with-do-block* occurs, evaluate it. Append the resulting value to *L*.

10 An *argument* is processed as follows:

11 a) Evaluate the *argument-in-parentheses*.

12 b) Let *L* be the resulting list.

13 An *argument-in-parentheses* is processed as follows:

14 a) Create an empty list of arguments *L*.

15 b) If the *command* occurs, evaluate it. Append the resulting value to *L*.

16 c) If the *operator-expression-list* occurs, for each *operator-expression* of the *operator-expression-*
17 *list*, in the order they appears in the program text, take the following steps:

18 1) Evaluate the *operator-expression*. Let *V* be the resulting value.

19 2) Append *V* to *L*.

20 d) If the *association-list* occurs, evaluate it. Append the resulting value to *L*.

21 e) If the *splatting-argument* occurs, construct a list of arguments from it and concatenate the
22 resulting list to *L*.

23 f) If the *block-argument* occurs, construct a *block* which is passed to a method invocation as
24 described below.

25 A block which is passed to a method invocation is constructed from the *block-argument* as
26 follows:

27 a) Evaluate the *operator-expression*. Let *P* be the resulting value.

28 b) If *P* is not an instance of the class **Proc**, the behavior is implementation dependent.

29 c) Otherwise, the resulting block is the block which *P* represents.

30 11.2.2 Blocks

31 Syntax

```

1  block ::
2      brace-block
3      | do-block

4  brace-block ::
5      { block-formal-argument? block-body }

6  do-block ::
7      do block-formal-argument? block-body end

8  block-formal-argument ::
9      | |
10     | ||
11     | | block-formal-argument-list |

12 block-formal-argument-list ::
13     left-hand-side
14     | multiple-left-hand-side

15 block-body ::
16     compound-statement

```

Whether the *left-hand-side* (see §11.3.1.3) in the *block-formal-argument-list* is allowed to be of the following forms is implementation defined.

- *constant-identifier*
- *global-variable-identifier*
- *instance-variable-identifier*
- *class-variable-identifier*
- *primary-expression* [*indexing-argument-list?*]
- *primary-expression* (. | ::) (*local-variable-identifier* | *constant-identifier*)
- :: *constant-identifier*

Whether the *grouped-left-hand-side* in the *block-formal-argument-list* is allowed to be the following form is implementation defined.

- ((*multiple-left-hand-side-item* ,)+);

Semantics

A *block* is a sequence of statements or expressions passed to a method invocation.

1 A *block* can be called either by a *yield-expression* (see §11.2.4) or by invoking the method `call`
 2 on an instance of the class `Proc` which is created by an invocation of the method `Proc.new` to
 3 which the block is passed (see §15.2.17.3.3).

4 A *block* can be called with arguments. If a *block* is called by a *yield-expression*, the arguments
 5 to the *yield-expression* are used as the arguments to the *block* call. If a *block* is called by an
 6 invocation of the method `call`, the arguments to the method invocation is used as the arguments
 7 to the *block* call.

8 A *block* is evaluated under the execution context as it exists just before the method invocation to
 9 which the *block* is passed. However, the changes of variable bindings in `[[local-variable-bindings]]`
 10 after the *block* is passed to the method invocation affect the execution context. Let E_b be the
 11 affected execution context.

12 Both the *do-block* and the *brace-block* of the *block* are evaluated as follows:

13 a) Let E_o be the current execution context. Let L be the list of arguments passed to the block.

14 b) Set the execution context to E_b .

15 c) Push an empty set of local variable bindings onto `[[local-variable-bindings]]`.

16 d) If the *block-formal-argument-list* in the *do-block* or the *brace-block* occurs:

17 • If the *block-formal-argument-list* is of the form *left-hand-side* or *grouped-left-hand-side*:

18 — If the length of L is 0, let X be `nil`.

19 — If the length of L is 1, let X be the only element of L .

20 — If the length of L is larger than 1, the result of this step is implementation dependent.
 21

22 — If the *block-formal-argument-list* is of the form *left-hand-side*, evaluate a *single-*
 23 *variable-assignment-expression* E (see §11.3.1.1.1), where the *variable* of E is the
 24 *left-hand-side* and the value of the *operator-expression* of E is X .

25 — If the *block-formal-argument-list* is of the form *grouped-left-hand-side*, evaluate
 26 a *many-to-many-assignment-expression* E (see §11.3.1.3), where the *multiple-left-*
 27 *hand-side* of E is the *grouped-left-hand-side* and the value of the *method-invocation-*
 28 *without-parentheses* or *operator-expression* of E is X .

29 • If the *block-formal-argument-list* is of the form *multiple-left-hand-side* and the *multiple-*
 30 *left-hand-side* is not a *grouped-left-hand-side*:

31 1) If the length of L is 1:

32 i) If the only element of L is not an instance of the class `Array`, the result of
 33 this step is implementation dependent.

34 ii) Create a list of arguments Y which contains the elements of L , preserving
 35 their order.

- 1 2) If the length of L is 0 or larger than 1, let Y be L .
- 2 3) Evaluate the *many-to-many-assignment-statement* E as described in §11.3.1.3,
3 where the *multiple-left-hand-side* of E is the *block-formal-argument-list* and the
4 list of arguments constructed from the *multiple-right-hand-side* of E is Y .
- 5 e) Evaluate the *block-body*. If the evaluation of the *block-body*:
 - 6 • is terminated by a *break-expression*:
 - 7 — If the method invocation with which *block* is passed has already terminated when
8 the *block* is called:
 - 9 1) Let S be an instance of the class `Symbol` with name `break`.
 - 10 2) If the *jump-argument* of the *break-expression* occurs, let V be the value of
11 the *jump-argument*. Otherwise, let V be `nil`.
 - 12 3) Raise a direct instance of the class `LocalJumpError` which has two instance
13 variable bindings, one named `@reason` with the value S and the other named
14 `@exit_value` with the value V .
 - 15 — Otherwise, restore the execution context to E_o and terminate Step i and take Step
16 j of the current method invocation (see §13.3.3).
 - 17 If the *jump-argument* of the *break-expression* occurs, the value of the current
18 method invocation is the value of the *jump-argument*. Otherwise, the value of the
19 current method invocation is `nil`.
 - 20 • is terminated by a *redo-expression*, repeat Step e.
 - 21 • is terminated by a *next-expression*:
 - 22 — If the *jump-argument* of the *next-expression* occurs, let V be the value of the
23 *jump-argument*.
 - 24 — Otherwise, let V be `nil`.
 - 25 • is terminated by a *return-expression*, remove the element from the top of `[[local-variable-`
26 bindings]] .
 - 27 • terminates otherwise, let V be the resulting value of the evaluation of the *block-body*.
- 28 f) Unless Step e is terminated by a *return-expression*, restore the execution context to E_o ,
29 even when an exception is raised and not handled in Step d or e.
- 30 g) The value of calling the *do-block* or the *brace-block* is V .

31 11.2.3 The super expression

32 Syntax

```

1  super-expression ::=
2      super-with-optional-argument
3      | super-with-argument
4      | super-with-argument-and-do-block

5  super-with-optional-argument ::
6      super ( [no whitespace here] argument-with-parentheses )? block?

7  super-with-argument ::
8      super argument

9  super-with-argument-and-do-block ::
10     super argument do-block

```

11 The *block-argument* of the *argument-in-parentheses* of the *argument* (see §11.2.1) of a *super-*
12 *with-argument-and-do-block* shall not occur.

13 Semantics

14 A *super-expression* is evaluated as follows:

15 a) If the current self is pushed by an *eigenclass-definition* (see §13.4.2), or an invocation of
16 one of the following methods, the behavior is implementation dependent:

- 17 • the method `class_eval` of the class `Module` (see §15.2.2.3.15)
- 18 • the method `module_eval` of the class `Module` (see §15.2.2.3.35)
- 19 • the method `instance_eval` of the class `Kernel` (see §15.3.1.2.18)

20 b) Let *A* be an empty list. Let *B* be the top of `[[block]]`.

21 • If the *super-expression* is a *super-with-optional-argument*, and neither the *argument-*
22 *with-parentheses* nor the *block* occurs, construct a list of arguments as follows:

23 1) Let *M* be the method which correspond to the current method invocation. Let *L*
24 be the *parameter-list* of the *method-parameter-part* of *M*. Let *S* be the set of local
25 variable bindings in `[[local-variable-bindings]]` which corresponds to the current
26 method invocation.

27 2) If the *mandatory-parameter-list* occurs in *L*, for each *mandatory-parameter* *p*, take
28 the following steps:

29 i) Let *v* be the value of the binding with name *p* in *S*.

30 ii) Append *v* to *A*.

- 1 3) If the *optional-parameter-list* occurs in L , for each *optional-parameter* p , take the
2 following steps:
 - 3 i) Let n be the *optional-parameter-name* of p .
 - 4 ii) Let v be the value of the binding with name n in S .
 - 5 iii) Append v to A .
- 6 4) If the *array-parameter* occurs in L :
 - 7 i) Let n be the *array-parameter-name* of the *array-parameter*.
 - 8 ii) Let v be the value of the binding with name n in S . Append each element of
9 v , in the indexing order, to A .
- 10 • If the *super-expression* is a *super-with-optional-argument* with either or both of the
11 *argument-with-parentheses* and the *block*:
 - 12 — If the *argument-with-parentheses* occurs, construct a list of arguments and a block
13 as described in §11.2.1. Let A be the resulting list. Let B be the resulting block,
14 if any.
 - 15 — If the *block* occurs, Let B be the *block*.
- 16 • If the *super-expression* is a *super-with-argument*, construct the list of arguments from
17 the *argument* as described in §11.2.1. Let A be the resulting list. If *block-argument* of
18 the *argument-in-parentheses* of *argument* occurs, let B be the *block* constructed from
19 the *block-argument*.
- 20 • If the *super-expression* is a *super-with-argument-and-do-block*, construct a list of argu-
21 ments from the *argument* as described in §11.2.1. Let A be the resulting list. Let B
22 be the *do-block*.
- 23 c) Determine the method to be invoked as follows:
 - 24 1) Let C be the current class or module. Let N be the top of $\llbracket \text{defined-method-name} \rrbracket$.
 - 25 2) Search for a method binding with name N from Step b in §13.3.4, assuming that C in
26 §13.3.4 to be C .
 - 27 3) If a binding is found and its value is not undef (see §13.1.1), let V be the value of the
28 binding.
 - 29 4) Otherwise, add a direct instance of the class `Symbol` with name N to the head of A ,
30 and invoke the method `method_missing` on the current self with A as arguments and
31 B as the block. Then, terminate the evaluation of the *super-expression*. The value of
32 the *super-expression* is the resulting value of the method invocation.
- 33 d) Take Step g, h, i, and j of §13.3.3, assuming that A , B , M , R , and V in §13.3.3 to be A , B ,
34 N , the current self, and V in this subclause respectively. The value of the *super-expression*
35 is the resulting value.

1 11.2.4 The yield expression

2 Syntax

```
3  yield-expression ::=
4      yield-with-optional-argument
5      | yield-with-argument

6  yield-with-optional-argument ::
7      yield-with-parentheses-and-argument
8      | yield-with-parentheses-without-argument
9      | yield

10 yield-with-parentheses-and-argument ::
11     yield [no whitespace here] ( argument-in-parentheses )

12 yield-with-parentheses-without-argument ::
13     yield [no whitespace here] ( )

14 yield-with-argument ::
15     yield argument
```

16 The *block-argument* of the *argument-in-parentheses* (see §11.2.1) of a *yield-with-parentheses-and-*
17 *argument* shall not occur.

18 The *block-argument* of the *argument-in-parentheses* of the *argument* (see §11.2.1) of a *yield-with-*
19 *argument* shall not occur.

20 Semantics

21 A *yield-expression* calls the block at the top of `[[block]]`.

22 A *yield-with-optional-argument* is evaluated as follows:

23 a) Let *B* be the top of `[[block]]`. If *B* is block-not-given:

24 1) Let *S* be a direct instance of the class `Symbol` with name `noreason`.

25 2) Let *V* be an implementation defined value.

26 3) Raise a direct instance of the class `LocalJumpError` which has two instance variable
27 bindings, one named `@reason` with the value *S* and the other named `@exit_value` with
28 the value *V*.

29 b) If the *yield-with-optional-argument* is of the form *yield-with-parentheses-and-argument*, cre-
30 ate a list of arguments from the *argument* as described in §11.2.1. Let *L* be the list.

- 1 c) If the *yield-with-optional-argument* is of the form *yield-with-parentheses-without-argument*
2 or *yield*, create an empty list of argument *L*.
- 3 d) Call *B* with *L* as described in §11.2.2.
- 4 e) The value of *yield-with-optional-argument* is the value of the block call.
- 5 A *yield-with-argument* is evaluated as follows:
 - 6 a) Let *B* be the top of `[[block]]`. If *B* is block-not-given:
 - 7 1) Let *S* be a direct instance of the class `Symbol` with name `noreason`.
 - 8 2) Let *V* be an implementation defined value.
 - 9 3) Raise a direct instance of the class `LocalJumpError` which has two instance variable
10 bindings, one named `@reason` with the value *S* and the other named `@exit_value` with
11 the value *V*.
 - 12 b) Create a list of arguments from the *argument* as described in §11.2.1. Let *L* be the list.
 - 13 c) Call *B* with *L* as described in §11.2.2.
 - 14 d) The value of the *yield-with-argument* is the value of the block call.

15 11.3 Operator expressions

16 Syntax

```

17  operator-expression ::
18      assignment-expression
19      | defined?-without-parentheses
20      | conditional-operator-expression

```

21 11.3.1 Assignments

22 Syntax

```

23  assignment ::=
24      assignment-expression
25      | assignment-statement

26  assignment-expression ::
27      single-assignment-expression
28      | abbreviated-assignment-expression
29      | assignment-with-rescue-modifier

30  assignment-statement ::
31      single-assignment-statement

```

1 | *abbreviated-assignment-statement*
2 | *multiple-assignment-statement*

3 Semantics

4 Assignments create or update variable bindings, or invoke a method whose name ends with =.

5 Evaluation of each construct is described below.

6 11.3.1.1 Single assignments

7 Syntax

8 *single-assignment* ::=
9 *single-assignment-expression*
10 | *single-assignment-statement*

11 *single-assignment-expression* ::
12 *single-variable-assignment-expression*
13 | *scoped-constant-assignment-expression*
14 | *single-indexing-assignment-expression*
15 | *single-method-assignment-expression*

16 *single-assignment-statement* ::
17 *single-variable-assignment-statement*
18 | *scoped-constant-assignment-statement*
19 | *single-indexing-assignment-statement*
20 | *single-method-assignment-statement*

21 11.3.1.1.1 Single variable assignments

22 Syntax

23 *single-variable-assignment* ::=
24 *single-variable-assignment-expression*
25 | *single-variable-assignment-statement*

26 *single-variable-assignment-expression* ::
27 *variable* [no line-terminator here] = *operator-expression*

28 *single-variable-assignment-statement* ::
29 *variable* [no line-terminator here] = *method-invocation-without-parentheses*

30 *scoped-constant-assignment* ::=
31 *scoped-constant-assignment-expression*

```

1      | scoped-constant-assignment-statement

2      scoped-constant-assignment-expression ::
3          primary-expression [no whitespace here] :: constant-identifier
4          [no line-terminator here] = operator-expression
5      | :: constant-identifier [no line-terminator here] = operator-expression

6      scoped-constant-assignment-statement ::
7          primary-expression [no whitespace here] :: constant-identifier
8          [no line-terminator here] = method-invocation-without-parentheses
9      | :: constant-identifier [no line-terminator here] = method-invocation-without-parentheses

```

Semantics

A *single-variable-assignment* is evaluated as follows:

- a) Evaluate the *operator-expression* or the *method-invocation-without-parentheses*. Let V be the resulting value.
- b) • If the *variable* is a *constant-identifier*:
 - 1) Let N be the *constant-identifier*.
 - 2) If a binding with name N exists in the set of bindings of constants of the current class or module, replace the value of the binding with V .
 - 3) Otherwise, create a variable binding with name N and value V in the set of bindings of constants of the current class or module.
- If the *variable* is a *global-variable-identifier*:
 - 1) Let N be the *global-variable-identifier*.
 - 2) If a binding with name N exists in $\llbracket \text{global-variable-bindings} \rrbracket$, replace the value of the binding with V .
 - 3) Otherwise, create a variable binding with name N and value V in $\llbracket \text{global-variable-bindings} \rrbracket$.
- If the *variable* is a *class-variable-identifier*:
 - 1) Let C be the first class or module in the list at the top of $\llbracket \text{class-module-list} \rrbracket$ which is not an eigenclass.

Let CS be the set of classes which consists of C and all the superclasses of C . Let MS be the set of modules which consists of all the modules in the included module lists of all classes in CS . Let CM be the union of CS and MS .

Let N be the *class-variable-identifier*.

- 1 2) If one of the classes or modules in CM has a binding with name N in the set of
2 bindings of class variables, let B be that binding.
- 3 If more than one class or module in CM has bindings with name N in the set
4 of bindings of class variables, let B be one of those bindings. Which binding is
5 selected is implementation defined.
- 6 Replace the value of B with V .
- 7 3) If none of the classes or modules in CM has a binding with name N in the set of
8 bindings of class variables, create a variable binding with name N and value V in
9 the set of bindings of class variables of C .
- 10 • If the *variable* is an *instance-variable-identifier*:
- 11 1) Let N be the *instance-variable-identifier*.
- 12 2) If a binding with name N exists in the set of bindings of instance variables of the
13 current self, replace the value of the binding with V .
- 14 3) Otherwise, create a variable binding with name N and value V in the set of
15 bindings of instance variables of the current self.
- 16 • If the *variable* is a *local-variable-identifier*:
- 17 1) Let N be the *local-variable-identifier*.
- 18 2) Search for a binding of a local variable with name N as described in §9.1.1.
- 19 3) If a binding is found, replace the value of the binding with V .
- 20 4) Otherwise, create a variable binding with name N and value V in the current set
21 of local variable bindings.
- 22 c) The value of the *single-variable-assignment* is V .
- 23 A *scoped-constant-assignment* is evaluated as follows:
- 24 a) If the *primary-expression* occurs, evaluate it and let M be the resulting value. Otherwise,
25 let M be the class **Object**.
- 26 b) If M is an instance of the class **Module**:
- 27 1) Let N be the *constant-identifier*.
- 28 2) Evaluate the *operator-expression* or the *method-invocation-without-parentheses*. Let V
29 be the resulting value.
- 30 3) Create a variable binding with name N and value V in the set of bindings of constants
31 of M .
- 32 4) The value of the *scoped-constant-assignment* is V .

1 c) If M is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.

2 11.3.1.1.2 Single indexing assignments

3 Syntax

4 $\text{single-indexing-assignment} ::=$
5 $\text{single-indexing-assignment-expression}$
6 | $\text{single-indexing-assignment-statement}$

7 $\text{single-indexing-assignment-expression} ::$
8 $\text{primary-expression}$ [no *line-terminator* here] [*indexing-argument-list?*]
9 [no *line-terminator* here] = $\text{operator-expression}$

10 $\text{single-indexing-assignment-statement} ::$
11 $\text{primary-expression}$ [no *line-terminator* here] [*indexing-argument-list?*]
12 [no *line-terminator* here] = $\text{method-invocation-without-parentheses}$

13 Semantics

14 A *single-indexing-assignment* is evaluated as follows:

- 15 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 16 b) Construct a list of arguments from the *indexing-argument-list* as described in §11.2.1. Let
17 L be the resulting list.
- 18 c) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let V be the
19 resulting value.
- 20 d) Append V to L .
- 21 e) Invoke the method `[]=` on O with L as the list of arguments.
- 22 f) The value of the *single-indexing-assignment* is V .

23 11.3.1.1.3 Single method assignments

24 Syntax

25 $\text{single-method-assignment} ::=$
26 $\text{single-method-assignment-expression}$
27 | $\text{single-method-assignment-statement}$

28 $\text{single-method-assignment-expression} ::$
29 $\text{primary-expression}$ [no *line-terminator* here] (\cdot | $::$) *local-variable-identifier*
30 [no *line-terminator* here] = $\text{operator-expression}$

```

1      | primary-expression [no line-terminator here] . constant-identifier
2      [no line-terminator here] = operator-expression

3  single-method-assignment-statement ::
4      primary-expression [no line-terminator here] ( . | :: ) local-variable-identifier
5      [no line-terminator here] = method-invocation-without-parentheses
6      | primary-expression [no line-terminator here] . constant-identifier
7      [no line-terminator here] = method-invocation-without-parentheses

```

8 Semantics

9 A *single-method-assignment* is evaluated as follows:

- 10 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 11 b) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let V be the
12 resulting value.
- 13 c) Let M be the *local-variable-identifier* or *constant-identifier*. Let N be the concatenation of
14 M and $=$.
- 15 d) Invoke the method whose name is N on O with a list of arguments which contains only one
16 value V .
- 17 e) The value of the *single-method-assignment* is V .

18 11.3.1.2 Abbreviated assignments

19 Syntax

```

20  abbreviated-assignment ::=
21      abbreviated-assignment-expression
22      | abbreviated-assignment-statement

23  abbreviated-assignment-expression ::
24      abbreviated-variable-assignment-expression
25      | abbreviated-indexing-assignment-expression
26      | abbreviated-method-assignment-expression

27  abbreviated-assignment-statement ::
28      abbreviated-variable-assignment-statement
29      | abbreviated-indexing-assignment-statement
30      | abbreviated-method-assignment-statement

```

31 11.3.1.2.1 Abbreviated variable assignments

32 Syntax

```

1  abbreviated-variable-assignment ::=
2      abbreviated-variable-assignment-expression
3      | abbreviated-variable-assignment-statement

4  abbreviated-variable-assignment-expression ::
5      variable [no line-terminator here] assignment-operator operator-expression

6  abbreviated-variable-assignment-statement ::
7      variable [no line-terminator here] assignment-operator
8      method-invocation-without-parentheses

```

9 Semantics

10 An *abbreviated-variable-assignment* is evaluated as follows:

- 11 a) Evaluate the *variable* as a variable reference (see §11.4.3). Let *V* be the resulting value.
- 12 b) Evaluate the *operator-expression* or the *method-invocation-without-parentheses*. Let *W* be
13 the resulting value.
- 14 c) Let *OP* be the *assignment-operator-name* of the *assignment-operator*.
- 15 d) Evaluate the *operator-expression* of the form *L OP R*, where the value of *L* is *V* and the
16 value of *R* is *W*. Let *X* be the resulting value.
- 17 e) Let *I* be the *variable* of the *abbreviated-variable-assignment-expression* or the *abbreviated-*
18 *variable-assignment-statement*.
- 19 f) Evaluate a *single-variable-assignment-expression* (see §11.3.1.1.1) where its *variable* is *I* and
20 the value of the *operator-expression* is *X*.
- 21 g) The value of the *abbreviated-variable-assignment* is *X*.

22 11.3.1.2.2 Abbreviated indexing assignments

23 Syntax

```

24 abbreviated-indexing-assignment ::=
25     abbreviated-indexing-assignment-expression
26     | abbreviated-indexing-assignment-statement

27 abbreviated-indexing-assignment-expression ::
28     primary-expression [no line-terminator here] [ indexing-argument-list? ]
29     [no line-terminator here] assignment-operator operator-expression

```

1 *abbreviated-indexing-assignment-statement* ::
2 *primary-expression* [no *line-terminator* here] [*indexing-argument-list*?]
3 [no *line-terminator* here] *assignment-operator* *method-invocation-without-parentheses*

4 **Semantics**

5 An *abbreviated-indexing-assignment* is evaluated as follows:

- 6 a) Evaluate the *primary-expression*. Let *O* be the resulting value.
- 7 b) Construct a list of arguments from the *indexing-argument-list* as described in §11.2.1. Let
8 *L* be the resulting list.
- 9 c) Invoke the method [] on *O* with *L* as the list of arguments. Let *V* be the resulting value.
- 10 d) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let *W* be the
11 resulting value.
- 12 e) Let *OP* be the *assignment-operator-name* of the *assignment-operator*.
- 13 f) Evaluate the *operator-expression* of the form *V OP W*. Let *X* be the resulting value.
- 14 g) Append *X* to *L*.
- 15 h) Invoke the method []= on *O* with *L* as the list of arguments.
- 16 i) The value of the *abbreviated-indexing-assignment* is *X*.

17 **11.3.1.2.3 Abbreviated method assignments**

18 **Syntax**

19 *abbreviated-method-assignment* ::=
20 *abbreviated-method-assignment-expression*
21 | *abbreviated-method-assignment-statement*

22 *abbreviated-method-assignment-expression* ::
23 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
24 [no *line-terminator* here] *assignment-operator* *operator-expression*
25 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
26 [no *line-terminator* here] *assignment-operator* *operator-expression*

27 *abbreviated-method-assignment-statement* ::
28 *primary-expression* [no *line-terminator* here] (. | ::) *local-variable-identifier*
29 [no *line-terminator* here] *assignment-operator* *method-invocation-without-parentheses*
30 | *primary-expression* [no *line-terminator* here] . *constant-identifier*
31 [no *line-terminator* here] *assignment-operator* *method-invocation-without-parentheses*

1 Semantics

2 An *abbreviated-method-assignment* is evaluated as follows:

- 3 a) Evaluate the *primary-expression*. Let O be the resulting value.
- 4 b) Create an empty list of arguments L . Invoke the method whose name is the *local-variable-identifier* on O with L as the list of arguments. Let V be the resulting value.
- 5
- 6 c) Evaluate the *operator-expression* or *method-invocation-without-parentheses*. Let W be the
- 7 resulting value.
- 8 d) Let OP be the *assignment-operator-name* of the *assignment-operator*.
- 9 e) Evaluate the *single-method-assignment* of the form $V\ OP\ W$. Let X be the resulting value.
- 10 f) Let M be the *local-variable-identifier* or the *constant-identifier*. Let N be the concatenation
- 11 of M and $=$.
- 12 g) Invoke the method whose name is N on O with X as the argument.
- 13 h) The value of the *abbreviated-method-assignment* is X .

14 11.3.1.3 Multiple assignments

15 Syntax

16 *multiple-assignment-statement* ::
17 *many-to-one-assignment-statement*
18 | *one-to-packing-assignment-statement*
19 | *many-to-many-assignment-statement*

20 *many-to-one-assignment-statement* ::
21 *left-hand-side* [no *line-terminator* here] = *multiple-right-hand-side*

22 *one-to-packing-assignment-statement* ::
23 *packing-left-hand-side* [no *line-terminator* here] =
24 (*method-invocation-without-parentheses* | *operator-expression*)

25 *many-to-many-assignment-statement* ::
26 *multiple-left-hand-side* [no *line-terminator* here] = *multiple-right-hand-side*
27 | (*multiple-left-hand-side* **but not** *packing-left-hand-side*)
28 [no *line-terminator* here] =
29 (*method-invocation-without-parentheses* | *operator-expression*)

30 *left-hand-side* ::
31 *variable*
32 | *primary-expression* [no *line-terminator* here] [*indexing-argument-list?*]
33 | *primary-expression* [no *line-terminator* here]

1 (. | ::) (*local-variable-identifier* | *constant-identifier*)
2 | :: *constant-identifier*

3 *multiple-left-hand-side* ::
4 (*multiple-left-hand-side-item* ,)+ *multiple-left-hand-side-item*?
5 | (*multiple-left-hand-side-item* ,)+ *packing-left-hand-side*?
6 | *packing-left-hand-side*
7 | *grouped-left-hand-side*

8 *packing-left-hand-side* ::
9 * *left-hand-side*?

10 *grouped-left-hand-side* ::
11 (*multiple-left-hand-side*)

12 *multiple-left-hand-side-item* ::
13 *left-hand-side*
14 | *grouped-left-hand-side*

15 *multiple-right-hand-side* ::
16 *operator-expression-list* (, *splatting-right-hand-side*)?
17 | *splatting-right-hand-side*

18 *splatting-right-hand-side* ::
19 *splatting-argument*

20 Any of the *operator-expressions* in a *multiple-assignment-statement* or *splatting-right-hand-side*
21 shall not be a *jump-expression*.

22 Semantics

23 A *many-to-one-assignment-statement* is evaluated as follows:

- 24 a) Construct a list of values from the *multiple-right-hand-side* (see below). Let *L* be the
25 resulting list.
- 26 b) If the length of *L* is 0 or 1, let *A* be an implementation defined value.
- 27 c) If the length of *L* is larger than 1, create a direct instance of the class **Array** and store the
28 elements of *L* in it, preserving their order. Let *A* be the instance of the class **Array**.
- 29 d) Evaluate a *single-variable-assignment-expression* (see §11.3.1.1.1) where its *variable* is the
30 *left-hand-side* and the value of its *operator-expression* is *A*.
- 31 e) The value of the *many-to-one-assignment-statement* is *A*.

32 A list of values is constructed from a *multiple-right-hand-side* as follows:

- 1 a) If the *operator-expression-list* occurs, evaluate its *operator-expressions* in the order they
2 appear in the program text. Let *L1* be a list which contains the resulting values, preserving
3 their order.
 - 4 b) If the *operator-expression-list* does not occur, create an empty list of values *L1*.
 - 5 c) If the *splatting-right-hand-side* occurs, construct a list of values from its *splatting-argument*
6 as described in §11.2.1 and let *L2* be the resulting list.
 - 7 d) If the *splatting-right-hand-side* does not occur, create an empty list of values *L2*.
 - 8 e) The result is the concatenation of *L1* and *L2*.
- 9 A *one-to-packing-assignment-statement* is evaluated as follows:
- 10 a) Evaluate the *method-invocation-without-parentheses* or the *operator-expression*. Let *V* be
11 the value.
 - 12 b) If *V* is an instance of the class **Array**, let *A* be an implementation defined value.
 - 13 c) If *V* is not an instance of the class **Array**, create an instance of the class **Array** *A* which
14 contains only one value *V*.
 - 15 d) If the *left-hand-side* of the *packing-left-hand-side* occurs, evaluate a *single-variable-assignment-*
16 *expression* (see §11.3.1.1.1) where its *variable* is the *left-hand-side* and the value of the
17 *operator-expression* is *A*.
 - 18 e) The value of the *one-to-packing-assignment-statement* is *A*.
- 19 A *many-to-many-assignment-statement* is evaluated as follows:
- 20 a) If the *multiple-right-hand-side* occurs, construct a list of values from it (see above) and let
21 *R* be the resulting list.
 - 22 b) If the *multiple-right-hand-side* does not occur:
 - 23 1) Evaluate the *method-invocation-without-parentheses* or the *operator-expression*. Let *V*
24 be the resulting value.
 - 25 2) If *V* is not an instance of the class **Array**, the behavior is implementation dependent.
 - 26 3) Create a list of arguments *R* which contains all the elements of *V*, preserving their
27 order.
 - 28 c) Create an empty list of variables *L*.
- 29 For each *multiple-left-hand-side-item*, in the order they appear in the program text, append
30 the *left-hand-side* or the *grouped-left-hand-side* of the *multiple-left-hand-side-item* to *L*.
- 31 If the *packing-left-hand-side* of the *multiple-left-hand-side* occurs, append it to *L*.
- 32 If the *multiple-left-hand-side* is a *grouped-left-hand-side*, append the *grouped-left-hand-side*
33 to *L*.

- 1 d) For each element L_i of L , in the same order in L , take the following steps:
 - 2 • Let i be the index of L_i within L . Let N_R be the number of elements of R .
 - 3 • If L_i is a *left-hand-side*:
 - 4 1) If i is larger than N_R , let V be **nil**.
 - 5 2) Otherwise, let V be the i th element of R .
 - 6 3) Evaluate the *single-variable-assignment* of the form $L_i = V$.
 - 7 • If L_i is a *packing-left-hand-side* and its *left-hand-side* occurs:
 - 8 1) If i is larger than N_R , create an empty direct instance of the class **Array**. Let A - 9 be the instance.
 - 10 2) Otherwise, create a direct instance of the class **Array** which contains elements in
 - 11 R whose index is equal to, or larger than i , in the same order they are store in R .
 - 12 let A be the instance.
 - 13 3) Evaluate a *single-variable-assignment-expression* (see §11.3.1.1.1) where its *vari-* - 14 *able* is the *left-hand-side* and the value of the *operator-expression* is A .
 - 15 • If L_i is a *grouped-left-hand-side*:
 - 16 1) If i is larger than N_R , let V be **nil**.
 - 17 2) Otherwise, let V be the i th element of R .
 - 18 3) Evaluate a *many-to-many-assignment-statement* where its *multiple-left-hand-side* - 19 is the *multiple-left-hand-side* of the *grouped-left-hand-side* and its *multiple-right-* - 20 *hand-side* is V .

21 11.3.1.4 Assignments with rescue modifiers

22 Syntax

```

23 assignment-with-rescue-modifier ::
24   left-hand-side [no line-terminator here] =
25   operator-expression1 rescue operator-expression2

```

26 Semantics

27 An *assignment-with-rescue-modifier* is evaluated as follows:

- 28 a) Evaluate the *operator-expression*₁. Let V be the resulting value.
- 29 If an exception is raised and not handled during the evaluation of the *operator-expression*₁,
- 30 and if the exception is an instance of the class **StandardError**, evaluate the *operator-*
- 31 *expression*₂ and let V be the resulting value.

- 1 b) Evaluate a *single-variable-assignment-expression* (see §11.3.1.1.1) where its *variable* is the
2 *left-hand-side* and the value of the *operator-expression* is *V*. The value of the *assignment-*
3 *with-rescue-modifier* is the resulting value of the evaluation.

4 11.3.2 Unary operators

5 Syntax

6 *unary-minus-expression* ::
7 *power-expression*₁
8 | - *power-expression*₂

9 *unary-expression* ::
10 *primary-expression*
11 | *logical-NOT-with-unary-expression*
12 | ~ *unary-expression*₁
13 | + *unary-expression*₂

14 If a *unary-minus-expression* of the form - *power-expression*₂ is the first argument (see §11.2.1),
15 *whitespaces* shall not occur between its - and *power-expression*₂.

16 If a *unary-expression* of the form + *unary-expression*₂ is the first argument (see §11.2.1), *whites-*
17 *paces* shall not occur between its + and *unary-expression*₂.

18 Semantics

19 See §11.1 for *logical-NOT-with-unary-expression*.

20 An *unary-expression* of the form ~ *unary-expression*₁ is evaluated as follows:

- 21 a) Evaluate the *unary-expression*₁. Let *X* be the resulting value.
22 b) Create an empty list of arguments *L*. Invoke the method ~ on *X* with *L* as the list of
23 arguments. The value of the *unary-expression* is the resulting value of the invocation.

24 An *unary-expression* of the form + *unary-expression*₂ is evaluated as follows:

- 25 a) Evaluate the *unary-expression*₂. Let *X* be the resulting value.
26 b) Create an empty list of arguments *L*. Invoke the method +@ on *X* with *L* as the list of
27 arguments. The value of the *unary-expression* is the resulting value of the invocation.
28 c) If the *unary-expression*₂ is a *numeric-literal* (see §8.5.5.1), instead of the above process, a
29 conforming processor may evaluate the *unary-expression* to the value of the *numeric-literal*.

30 An *unary-expression* of the form - *power-expression*₂ is evaluated as follows:

- 31 a) Evaluate the *power-expression*₂. Let *X* be the resulting value.

- 1 b) Create an empty list of arguments L . Invoke the method `-@` on X with L as the list of
2 arguments. The value of the *unary-expression* is the resulting value of the invocation.

3 11.3.2.1 The `defined?` expression

4 Syntax

5 *defined?-expression* ::=
6 *defined?-with-parentheses*
7 | *defined?-without-parentheses*

8 *defined?-with-parentheses* ::
9 `defined? (expression)`

10 *defined?-without-parentheses* ::
11 `defined? operator-expression`

12 Semantics

13 A *defined?-expression* is evaluated as follows:

14 a) If the *defined?-expression* is a *defined?-with-parentheses*, let E be the *expression*.

15 If the *defined?-expression* is a *defined?-without-parentheses*, let E be the *operator-expression*.

16 b) • If E is a *constant-identifier*:

17 1) Search for a binding of a constant with name E with the same evaluation steps
18 for *constant-identifier* as described in §11.4.3.1. However, a direct instance of the
19 class `NameError` shall not be raised when a binding is not found.

20 2) If a binding is found, the value of the *defined?-expression* is an implementation
21 defined value, which shall be a true value.

22 3) Otherwise, the value of the *defined?-expression* is `nil`.

23 • If E is a *global-variable-identifier*:

24 — If a binding with name E exists in `[[global-variable-bindings]]`, the value of the
25 *defined?-expression* is an implementation defined value, which shall be a true value.

26 — Otherwise, the value of the *defined?-expression* is `nil`.

27 • If E is a *class-variable-identifier*:

28 1) Let C be the current class or module. Let CS be the set of classes which consists
29 of C and all the superclasses of C . Let MS be the set of modules which consists
30 of all the modules in the included module lists of all classes in CS . Let CM be the
31 union of CS and MS .

- 2) If any of the classes or modules in *CM* has a binding with name *E* in the set of bindings of class variables, the value of the *defined?-expression* is an implementation defined value, which shall be a true value.
 - 3) Otherwise, the value of the *defined?-expression* is **nil**.
- If *E* is an *instance-variable-identifier*:
 - If a binding with name *E* exists in the set of bindings of instance variables of the current self, the value of the *defined?-expression* is an implementation defined value, which shall be a true value.
 - Otherwise, the value of the *defined?-expression* is **nil**.
 - If *E* is a *local-variable-identifier*:
 - 1) If the *local-variable-identifier* is a reference (see §9.1.2), the value of the *defined?-expression* is an implementation defined value, which shall be a true value.
 - 2) Otherwise, search for a method binding with name *E*, starting from the current class or module as described in §13.3.4.
 - If the binding is found and its value is not undef, the value of the *defined?-expression* is an implementation defined value, which shall be a true value.
 - Otherwise, the value of the *defined?-expression* is **nil**.
 - Otherwise, the value of the *defined?-expression* is implementation defined.

11.3.3 Binary operators

Syntax

equality-expression ::
 relational-expression
 | *relational-expression* <=> *relational-expression*
 | *relational-expression* == *relational-expression*
 | *relational-expression* === *relational-expression*
 | *relational-expression* != *relational-expression*
 | *relational-expression* =~ *relational-expression*
 | *relational-expression* !~ *relational-expression*

relational-expression ::
 bitwise-OR-expression
 | *relational-expression* > *bitwise-OR-expression*
 | *relational-expression* >= *bitwise-OR-expression*
 | *relational-expression* < *bitwise-OR-expression*
 | *relational-expression* <= *bitwise-OR-expression*

bitwise-OR-expression ::
 bitwise-AND-expression

```

1      | bitwise-OR-expression | bitwise-AND-expression
2      | bitwise-OR-expression ^ bitwise-AND-expression

3  bitwise-AND-expression ::
4      bitwise-shift-expression
5      | bitwise-AND-expression whitespace-before-operator? & bitwise-shift-expression

6  bitwise-shift-expression ::
7      additive-expression
8      | bitwise-shift-expression whitespace-before-operator? << additive-expression
9      | bitwise-shift-expression >> additive-expression

10 additive-expression ::
11     multiplicative-expression
12     | additive-expression whitespace-before-operator? + multiplicative-expression
13     | additive-expression whitespace-before-operator? - multiplicative-expression

14 multiplicative-expression ::
15     unary-minus-expression
16     | multiplicative-expression whitespace-before-operator? * unary-minus-expression
17     | multiplicative-expression whitespace-before-operator? / unary-minus-expression
18     | multiplicative-expression whitespace-before-operator? % unary-minus-expression

19 power-expression ::
20     unary-expression
21     | - ( numeric-literal ) ** power-expression
22     | unary-expression ** power-expression

23 binary-operator ::=
24     <=> | == | === | =~ | > | >= | < | <= | | | ^
25     | & | << | >> | + | - | * | / | % | **

```

26 If a *whitespace-before-operator* occurs, *whitespaces* shall not occur between the operator after the
27 *whitespace-before-operator* (i.e. *&*, *<<*, *+*, *-*, ***, */*, or *%*) and the nonterminal after that operator.

28 Semantics

29 An *operator-expression* of the form $x != y$ is evaluated as follows:

- 30 a) Evaluate x . Let X be the resulting value.
- 31 b) Evaluate y . Let Y be the resulting value.
- 32 c) Invoke the method `==` on X with a list of arguments which contains only one value Y . If
33 the resulting value is a true value, the value of the *operator-expression* is **false**. Otherwise,
34 the value of the *operator-expression* is **true**.

35 An *operator-expression* of the form $x !~ y$ is evaluated as follows:

- 1 a) Evaluate x . Let X be the resulting value.
- 2 b) Evaluate y . Let Y be the resulting value.
- 3 c) Invoke the method `=~` on X with a list of arguments which contains only one value Y . If
- 4 the resulting value is a true value, the value of the *operator-expression* is **false**. Otherwise,
- 5 the value of the *operator-expression* is **true**.

6 A conforming processor may include the operators `!=` and `!~` in *binary-operator*. In this case,

7 *operator-expressions* of the form `x != y` or `x !~ y` are evaluated as described below.

8 An *operator-expression* of the form x *binary-operator* y is evaluated as follows:

- 9 a) Evaluate x . Let X be the resulting value.
- 10 b) Evaluate y . Let Y be the resulting value.
- 11 c) Invoke the method whose name is the *binary-operator* on X with a list of arguments which
- 12 contains only one value Y . The value of the *operator-expression* is the resulting value of the
- 13 invocation.

14 11.4 Primary expressions

15 Syntax

```

16 primary-expression ::
17     class-definition
18     | eigenclass-definition
19     | module-definition
20     | method-definition
21     | singleton-method-definition
22     | yield-with-optional-argument
23     | if-expression
24     | unless-expression
25     | case-expression
26     | while-expression
27     | until-expression
28     | for-expression
29     | return-without-argument
30     | break-without-argument
31     | next-without-argument
32     | redo-expression
33     | retry-expression
34     | rescue-expression
35     | grouping-expression
36     | variable-reference
37     | scoped-constant-reference
38     | array-constructor
39     | hash-constructor
40     | literal
41     | defined?-with-parentheses
42     | primary-method-invocation

```

1 Semantics

2 See §13.2.2 for *class-definition*.

3 See §13.4.2 for *eigenclass-definition*.

4 See §13.1.2 for *module-definition*.

5 See §13.3.1 for *method-definition*.

6 See §13.4.3 for *singleton-method-definition*.

7 See §11.2.4 for *yield-with-optional-argument*.

8 See §11.3.2.1 for *defined?-with-parentheses*.

9 See §11.2 for *primary-method-invocation*.

10 11.4.1 Control structures

11 11.4.1.1 Conditional expressions

12 11.4.1.1.1 The if expression

13 Syntax

14 *if-expression* ::

15 **if** *expression* *then-clause* *elsif-clause** *else-clause*? **end**

16 *then-clause* ::

17 *separator* *compound-statement*

18 | *separator*? **then** *compound-statement*

19 *else-clause* ::

20 **else** *compound-statement*

21 *elsif-clause* ::

22 **elsif** *expression* *then-clause*

23 The *expression* of an *if-expression* or *elsif-clause* shall not be a *jump-expression*.

24 Semantics

25 The *if-expression* is evaluated as follows:

26 a) Evaluate *expression*. Let *V* be the resulting value.

- 1 b) If V is a true value, evaluate the *compound-statement* of the *then-clause*. The value of the
2 *if-expression* is the resulting value. In this case, *elsif-clauses* and the *else-clause*, if any, are
3 not evaluated.
- 4 c) If V is a false value, and if there is no *elsif-clause* and no *else-clause*, then the value of the
5 *if-expression* is **nil**.
- 6 d) If V is a false value, and if there is no *elsif-clause* but there is an *else-clause*, then evaluate
7 the *compound-statement* of the *else-clause*. The value of the *if-expression* is the resulting
8 value.
- 9 e) If V is a false value, and if there are one or more *elsif-clauses*, evaluate the sequence of
10 *elsif-clauses* as follows:
 - 11 1) Evaluate the *expression* of each *elsif-clause* in the order they appear in the program
12 text, until there is an *elsif-clause* for which *expression* evaluates to a true value. Let
13 T be this *elsif-clause*.
 - 14 2) If T exists, evaluate the *compound-expression* of its *then-clause*. The value of the *if*-
15 *expression* is the resulting value. Other *elsif-clauses* and an *else-clause* following T , if
16 any, are not evaluated.
 - 17 3) If T does not exist, and if there is an *else-clause*, then evaluate the *compound-statement*
18 of the *else-clause*. The value of the *if-expression* is the resulting value.
 - 19 4) If T does not exist, and if there is no *else-clause*, then the value of the *if-expression* is
20 **nil**.

21 11.4.1.1.2 The unless expression

22 Syntax

23 *unless-expression* ::
24 **unless** *expression then-clause else-clause?* **end**

25 The *expression* of an *unless-expression* shall not be a *jump-expression*.

26 Semantics

27 The *unless-expression* is evaluated as follows:

- 28 a) Evaluate the *expression*. Let V be the resulting value.
- 29 b) If V is a false value, evaluate the *compound-statement* of the *then-clause*. The value of the
30 *unless-expression* is the resulting value. In this case, the *else-clause*, if any, is not evaluated.
- 31 c) If V is a true value, and if there is no *else-clause*, then the value of the *unless-expression* is
32 **nil**.
- 33 d) If V is a true value, and if there is an *else-clause*, then evaluate the *compound-statement*
34 of the *else-clause*. The value of the *unless-expression* is the resulting value.

1 11.4.1.1.3 The case expression

2 Syntax

```
3  case-expression ::  
4      case-expression-with-expression  
5      | case-expression-without-expression  
  
6  case-expression-with-expression ::  
7      case expression separator-list? when-clause + else-clause? end  
  
8  case-expression-without-expression ::  
9      case separator-list? when-clause + else-clause? end  
  
10 when-clause ::  
11     when when-argument then-clause  
  
12 when-argument ::  
13     operator-expression-list ( , splatting-argument )?  
14     | splatting-argument
```

15 The *expression* of a *case-expression-with-expression* shall not be a *jump-expression*.

16 Semantics

17 A *case-expression* is evaluated as follows:

- 18 a) If the *case-expression* is a *case-expression-with-expression*, evaluate the *expression*. Let *V*
19 be the resulting value.
- 20 b) The meaning of the phrase “*O* is matching” in this step is defined as follows:
- 21 • If the *case-expression* is a *case-expression-with-expression*, invoke the method `===` on
22 *O* with a list of arguments which contains only one value *V*. *O* is matching if and only
23 if the resulting value is a true value.
 - 24 • If the *case-expression* is a *case-expression-without-expression*, *O* is matching if and only
25 if *O* is a true value.

26 Search the *when-clauses* in the order they appear in the program text for a matching *when-*
27 *clause* as follows:

- 28 1) If the *operator-expression-list* of the *when-argument* occurs:
- 29 • For each of its *operator-expressions*, evaluate it and test if the resulting value is
30 matching.
 - 31 • If a matching value is found, other *operator-expressions*, if any, are not evaluated.

- 1 2) If no matching value is found, and the *splatting-argument* occurs:
 - 2 • Construct a list of values from it as described in §11.2.1. For each element of the
 - 3 resulting list, in the indexing order, test if it is matching.
 - 4 • If a matching value is found, other values, if any, are not evaluated.
- 5 3) A *when-clause* is considered to be matching if and only if a matching value is found in
- 6 its *when-argument*. Later *when-clauses*, if any, are not tested in this case.
- 7 c) If one of the *when-clauses* is matching, evaluate the *compound-statement* of the *then-clause*
- 8 of this *when-clause*. The value of the *case-expression* is the resulting value.
- 9 d) If none of the *when-clauses* is matching, and if there is an *else-clause*, then evaluate the
- 10 *compound-statement* of the *else-clause*. The value of the *case-expression* is the resulting
- 11 value.
- 12 e) Otherwise, the value of the *case-expression* is **nil**.

13 11.4.1.1.4 Conditional operator

14 Syntax

15 *conditional-operator-expression* ::
 16 *range-constructor*
 17 | *range-constructor* ? *operator-expression*₁ : *operator-expression*₂

18 Semantics

19 A *conditional-operator-expression* of the form *range-constructor* ? *operator-expression*₁ : *operator-*
 20 *expression*₂ is evaluated as follows:

- 21 a) Evaluate the *range-constructor*.
- 22 b) If the resulting value is a true value, evaluate the *operator-expression*₁. The value of the
- 23 *conditional-operator-expression* is the resulting value of the evaluation.
- 24 c) Otherwise, evaluate the *operator-expression*₂. The value of the *conditional-operator-expression*
- 25 is the resulting value.

26 11.4.1.2 Iteration expressions

27 Syntax

28 *iteration-expression* ::=
 29 *while-expression*
 30 | *until-expression*
 31 | *for-expression*
 32 | *while-modifier-statement*
 33 | *until-modifier-statement*

1 Each *iteration-expression* has a **body**. The body of a *while-expression* or an *until-expression* is
2 its *compound-statement*. The body of a *while-modifier-statement* or an *until-modifier-statement*
3 is its *statement*.

4 See §12.4 for *while-modifier-statement*.

5 See §12.5 for *until-modifier-statement*.

6 11.4.1.2.1 The while expression

7 Syntax

8 *while-expression* ::
9 **while** *expression do-clause* **end**

10 *do-clause* ::
11 *separator compound-statement*
12 | **do** *compound-statement*

13 The *expression* of a *while-expression* shall not be a *jump-expression*.

14 Semantics

15 A *while-expression* is evaluated as follows:

- 16 a) Evaluate the *expression*. Let *V* be the resulting value.
- 17 b) If *V* is a false value, terminate the evaluation of the *while-expression*. The value of the
18 *while-expression* is **nil**.
- 19 c) Otherwise, evaluate the *compound-statement* of the *do-clause*. If this evaluation:
- 20 1) is terminated by a *break-expression*, terminate the evaluation of the *while-expression*.
- 21 If the *jump-argument* of the *break-expression* occurs, the value of the *while-expression*
22 is the value of the *jump-argument*. Otherwise, the value of the *while-expression* is **nil**.
- 23 2) is terminated by a *next-expression*, continue processing from Step a.
- 24 3) is terminated by a *redo-expression*, continue processing from Step c.
- 25 Otherwise, unless this evaluation is terminated by a *return-expression*, continue processing
26 from Step a.

27 11.4.1.2.2 The until expression

28 Syntax

1 *until-expression* ::
2 **until** *expression do-clause* **end**

3 The *expression* of an *until-expression* shall not be a *jump-expression*.

4 **Semantics**

5 An *until-expression* is evaluated as follows:

- 6 a) Evaluate the *expression*. Let *V* be the resulting value.
- 7 b) If *V* is a true value, terminate the evaluation of the *until-expression*. The value of the
8 *until-expression* is **nil**.
- 9 c) Otherwise, evaluate the *compound-statement* of the *do-clause*. If this evaluation:
- 10 1) is terminated by a *break-expression*, terminate the evaluation of the *until-expression*.
- 11 If the *jump-argument* of the *break-expression* occurs, the value of the *until-expression*
12 is the value of the *jump-argument*. Otherwise, the value of the *until-expression* is **nil**.
- 13 2) is terminated by a *next-expression*, continue processing from Step a.
- 14 3) is terminated by a *redo-expression*, continue processing from Step c.
- 15 Otherwise, unless this evaluation is terminated by a *return-expression*, continue processing
16 from Step a.

17 **11.4.1.2.3 The for expression**

18 **Syntax**

19 *for-expression* ::
20 **for** *for-variable in expression do-clause* **end**

21 *for-variable* ::
22 *left-hand-side*
23 | *multiple-left-hand-side*

24 The *expression* of a *for-expression* shall not be a *jump-expression*.

25 **Semantics**

26 A *for-expression* is evaluated as follows:

- 27 a) Evaluate the *expression*. Let *O* be the resulting value.

1 b) Let E be the *primary-method-invocation* of the form *primary-expression* [no *line-terminator*
 2 here] . **each do** | *block-formal-argument-list* | *block-body* **end**, where the value of the
 3 *primary-expression* is O , the *block-formal-argument-list* is the *for-variable*, the *block-body*
 4 is the *compound-statement* of the *do-clause*.

5 Evaluate E , but skip Step c of §11.2.2.

6 c) The value of the *for-expression* is the resulting value of the invocation.

7 11.4.1.3 Jump expressions

8 Syntax

9 *jump-expression* ::=
 10 *return-expression*
 11 | *break-expression*
 12 | *next-expression*
 13 | *redo-expression*
 14 | *retry-expression*

15 Semantics

16 Jump expressions are used to terminate the evaluation of a *method-body*, a *block-body*, the body
 17 of an *iteration-expression*, or the *compound-statement*₂ of a *rescue-clause*.

18 In this document, the **current block** or the **current iteration-expression** refers to either of
 19 the following:

- 20 • If the current method invocation exists, the current *block* or the current *iteration-expression*
 21 is the *block* or the *iteration-expression* whose evaluation started most recently among the
 22 *blocks* or *iteration-expressions* which are being evaluated on during the evaluation of the
 23 current method invocation.
- 24 • Otherwise, the current *block* or the current *iteration-expression* is the *block* or the *iteration-*
 25 *expression* whose evaluation started most recently among the *blocks* or *iteration-expressions*
 26 which are under evaluation.

27 11.4.1.3.1 The return expression

28 Syntax

29 *return-expression* ::=
 30 *return-without-argument*
 31 | *return-with-argument*

32 *return-without-argument* ::
 33 **return**

1 *return-with-argument* ::
2 **return** *jump-argument*

3 *jump-argument* ::
4 *argument*

5 The *block-argument* of the *argument-in-parentheses* of the *argument* (see §11.2.1) of a *jump-*
6 *argument* shall not occur.

7 **Semantics**

8 A *return-expression* is evaluated as follows:

9 a) Let *M* be the *method-body* which corresponds to the current method invocation. If such an
10 invocation does not exist, or has already terminated:

11 1) Let *S* be a direct instance of the class **Symbol** with name **return**.

12 2) If the *jump-argument* of the *return-expression* occurs, let *V* be the value of the *jump-*
13 *argument*. Otherwise, let *V* be **nil**.

14 3) Raise a direct instance of the class **LocalJumpError** which has two instance variable
15 bindings, one named **@reason** with the value *S* and the other named **@exit.value** with
16 the value *V*.

17 b) Evaluate the *jump-argument*, if any, as described below.

18 c) If there are *block-bodys* which include the *return-expression* and are included in *M*, terminate
19 the evaluations of such *block-bodys*, from innermost to outermost (see §11.2.2).

20 d) Terminate the evaluation of *M* (see §13.3.3).

21 A *jump-argument* is evaluated as follows:

22 a) If the *jump-argument* is a *splatting-argument*:

23 1) Construct a list of values from the *splatting-argument* as described in §11.2.1 and let
24 *L* be the resulting list.

25 2) If the length of *L* is 0 or 1, the value of the *jump-argument* is an implementation defined
26 value.

27 3) If the length of *L* is larger than 1, create a direct instance of the class **Array** and store
28 the elements of *L* in it, preserving their order. The value of the *jump-argument* is the
29 instance of the class **Array**.

30 b) Otherwise:

31 1) Construct a list of values from the *argument* as described in §11.2.1 and let *L* be the
32 resulting list.

- 1 2) If the length of L is 1, the value of the *jump-argument* is the only element of L .
- 2 3) If the length of L is larger than 1, create a direct instance of the class **Array** and store
- 3 the elements of L in it, preserving their order. The value of the *jump-argument* is the
- 4 instance of the class **Array**.

5 11.4.1.3.2 The break expression

6 Syntax

7 *break-expression* ::=

8 *break-without-argument*

9 | *break-with-argument*

10 *break-without-argument* ::

11 **break**

12 *break-with-argument* ::

13 **break** *jump-argument*

14 Semantics

15 A *break-expression* is evaluated as follows:

- 16 a) Evaluate the *jump-argument*, if any, as described in §11.4.1.3.1.
- 17 b) Let E be the current *block* or the current *iteration-expression*. If such a *block* or an *iteration-*
- 18 *expression* does not exist:
 - 19 1) Let S be a direct instance of the class **Symbol** with name **break**.
 - 20 2) If the *jump-argument* of the *break-expression* occurs, let V be the value of the *jump-*
 - 21 *argument*. Otherwise, let V be **nil**.
 - 22 3) Raise a direct instance of the class **LocalJumpError** which has two instance variable
 - 23 bindings, one named **@reason** with the value S and the other named **@exit_value** with
 - 24 the value V .
- 25 c) If E is a *block*, terminate the evaluation of the *block-body* of E (see §11.2.2).
- 26 d) If E is an *iteration-expression*, terminate the evaluation of the body of E (see §11.4.1.2).

27 11.4.1.3.3 The next expression

28 Syntax

29 *next-expression* ::=

30 *next-without-argument*

31 | *next-with-argument*

```

1  next-without-argument ::
2      next

3  next-with-argument ::
4      next jump-argument

```

5 Semantics

6 A *next-expression* is evaluated as follows:

- 7 a) Evaluate the *jump-argument*, if any, as described in §11.4.1.3.1.
- 8 b) Let E be the current *block* or the current *iteration-expression*. If such a *block* or an *iteration-expression* does not exist:
 - 9
 - 10 1) Let S be a direct instance of the class `Symbol` with name `next`.
 - 11 2) If the *jump-argument* of the *next-expression* occurs, let V be the value of the *jump-*
12 *argument*. Otherwise, let V be `nil`.
 - 13 3) Raise a direct instance of the class `LocalJumpError` which has two instance variable
14 bindings, one named `@reason` with the value S and the other named `@exit.value` with
15 the value V .
- 16 c) If E is a *block*, terminate the evaluation of the *block-body* of E (see §11.2.2).
- 17 d) If E is an *iteration-expression*, terminate the evaluation of the body of E (see §11.4.1.2).

18 11.4.1.3.4 The redo expression

19 Syntax

```

20 redo-expression ::
21     redo

```

22 Semantics

23 A *redo-expression* is evaluated as follows:

- 24 a) Let E be the current *block* or the current *iteration-expression*. If such a *block* or an *iteration-*
25 *expression* does not exist,
 - 26 1) Let S be a direct instance of the class `Symbol` with name `redo`.
 - 27 2) Raise a direct instance of the class `LocalJumpError` which has two instance variable
28 bindings, one named `@reason` with the value S and the other named `@exit.value` with
29 the value `nil`.

- 1 b) If E is a *block*, terminate the evaluation of the *block-body* of E (see §11.2.2).
- 2 c) If E is an *iteration-expression*, terminate the evaluation of the body of E (see §11.4.1.2).

3 11.4.1.3.5 The retry expression

4 Syntax

5 *retry-expression* ::
6 **retry**

7 Semantics

8 A *retry-expression* is evaluated as follows:

- 9 a) If the current method invocation exists, let M be the *method-body* which corresponds to
10 the current method invocation. Otherwise, let M be the *program*.
- 11 b) Let E be the innermost *rescue-clause* in M which encloses the *retry-expression*. If such a
12 *rescue-clause* does not exist, the behavior is implementation dependent.
- 13 c) Terminate the evaluation of the *compound-statement*₂ of E (see §11.4.1.4.1).

14 11.4.1.4 Exceptions

15 11.4.1.4.1 The rescue expression

16 Syntax

17 *rescue-expression* ::
18 **begin** *body-statement* **end**

19 *body-statement* ::
20 *compound-statement* *rescue-clause** *else-clause*? *ensure-clause*?

21 *rescue-clause* ::
22 **rescue** [no *line-terminator* here] *exception-class-list*?
23 *exception-variable-assignment*? *then-clause*

24 *exception-class-list* ::
25 *operator-expression*
26 | *multiple-right-hand-side*

27 *exception-variable-assignment* ::
28 **=>** *left-hand-side*

1 *ensure-clause* ::
2 **ensure** *compound-statement*

3 The *operator-expression* of an *exception-class-list* shall not be a *jump-expression*.

4 **Semantics**

5 The value of a *rescue-expression* is the value of the *body-statement*.

6 A *body-statement* is evaluated as follows:

7 a) Evaluate the *compound-statement* of the *body-statement*.

8 b) If no exception is raised, or all the raised exceptions are handled during Step a:

9 1) If the *else-clause* occurs, evaluate the *else-clause* as described in §11.4.1.1.1.

10 2) If the *ensure-clause* occurs, evaluate its *compound-statement*. The value of this evaluation is the value of the *ensure-clause*.

12 If both the *else-clause* and the *ensure-clause* occur, the value of the *body-statement* is the value of the *ensure-clause*. If only one of these clauses occurs, the value of the *body-statement* is the value of the occurring clause.

15 If neither the *else-clause* nor the *ensure-clause* occurs, the value of the *body-statement* is the value of its *compound-statement*.

17 c) If an exception is raised and not handled during Step a, test each *rescue-clause*, if any, in the order it occurs in the program text. The test determines whether the *rescue-clause* can handle the exception as follows:

20 1) Let *E* be the exception raised.

21 2) If the *exception-class-list* does not occur in the *rescue-clause*, and if *E* is an instance of the class **StandardError**, the *rescue-clause* handles *E*.

23 3) If the *exception-class-list* of the *rescue-clause* occurs:

24 • If the *exception-class-list* is of the form *operator-expression*, evaluate the *operator-expression*. Create an empty list of values, and append the value of the *operator-expression* to the list.

27 • If the *exception-class-list* is of the form *multiple-right-hand-side*, construct a list of values from the *multiple-right-hand-side* (see §11.3.1.3).

29 Let *L* be the list created by evaluating the *exception-class-list* as above. Compare each element *D* of *L* with *E* as follows:

31 • If *D* is neither the class **Exception** nor a subclass of the class **Exception**, raise a direct instance of the class **TypeError**.

1 • If E is an instance of D , the *rescue-clause* handles E . In this case, any remaining
2 *rescue-clauses* in the *body-statement* are not tested.

3 d) If a *rescue-clause* R which can handle E is found:

4 1) If the *exception-variable-assignment* of R occurs, evaluate it in the same way as a
5 *multiple-assignment-statement* of the form *left-hand-side* = *multiple-right-hand-side*
6 where the value of *multiple-right-hand-side* is E .

7 2) Evaluate the *compound-statement* of the *then-clause* of R . If this evaluation is termi-
8 nated by a *retry-expression*, continue processing from Step a. Otherwise, let V be the
9 value of this evaluation.

10 3) If the *ensure-clause* occurs, evaluate it. The value of the *body-statement* is the value
11 of the *ensure-clause*.

12 4) If the *ensure-clause* does not occur, the value of the *body-statement* is V .

13 e) If no *rescue-clause* occurs or if a *rescue-clause* which can handle E is not found, evaluate
14 the *ensure-clause*. In this case, the value of the *body-statement* is undefined.

15 The *ensure-clause* of a *body-statement*, if any, is always evaluated, even when the evaluation of
16 *body-statement* is terminated by a *jump-expression*.

17 11.4.2 Grouping expression

18 Syntax

19 *grouping-expression* ::
20 (*expression*)
21 | (*compound-statement*)

22 Semantics

23 A *grouping-expression* is evaluated as follows:

24 a) Evaluate the *expression* or the *compound-statement*.

25 b) The value of the *grouping-expression* is the resulting value.

26 11.4.3 Variable references

27 Syntax

28 *variable-reference* ::
29 *variable*
30 | *pseudo-variable*


```

1  variable ::
2      constant-identifier
3      | global-variable-identifier
4      | class-variable-identifier
5      | instance-variable-identifier
6      | local-variable-identifier

7  scoped-constant-reference ::
8      primary-expression [no whitespace here] :: constant-identifier
9      | :: constant-identifier

```

11.4.3.1 Constants

A *constant-identifier* is evaluated as follows:

- a) Let N be the *constant-identifier*.
- b) Search for a binding of a constant with name N as described below.

As soon as the binding is found in any of the following steps, the evaluation of the *constant-identifier* is terminated and the value of the *constant-identifier* is the value of the binding found.
- c) Let L be the top of $\llbracket \text{class-module-list} \rrbracket$. Search for a binding of a constant with name N in each element of L from start to end, including the first element, which is the current class or module, but except for the last element, which is the class `Object`.
- d) If a binding is not found, let C be the current class or module.

Let L be the included module list of C . Search each element of L in the reverse order for a binding of a constant with name N .
- e) If the binding is not found:
 - If C is a class:
 - 1) Let S be the direct superclass of C .
 - 2) If S is `nil`, create a direct instance of the class `Symbol` with name N , and let R be that instance. Invoke the method `const_missing` on the current class or module with R as the only argument.
 - 3) If S is not `nil`, search for a binding of a constant with name N in S .
 - 4) If the binding is not found, let L be the included module list of S and search each element of L in the reverse order for a binding of a constant with name N .
 - 5) If the binding is not found, let S be the direct superclass of S . Continue processing from Step e-2.

- 1 • If C is a module:
- 2 1) Search for a binding of a constant with name N in the class `Object`.
- 3 2) If the binding is not found, let L be the included module list of the class `Object`
- 4 and search each element of L in the reverse order for a binding of a constant with
- 5 name N .
- 6 3) If the binding is not found, create a direct instance of the class `Symbol` with name
- 7 N , and let R be that instance. Invoke the method `const_missing` on the current
- 8 class or module with R as the only argument.

9 **11.4.3.2 Scoped constants**

10 A *scoped-constant-reference* is evaluated as follows:

- 11 a) If the *primary-expression* occurs, evaluate it and let C be the resulting value. Otherwise,
- 12 let C be the class `Object`.
- 13 b) If C is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.
- 14 c) Otherwise:
- 15 1) Let N be the *constant-identifier*.
- 16 2) If a binding with name N exists in the set of bindings of constants of C , the value of
- 17 the *scoped-constant-reference* is the value of the binding.
- 18 3) Otherwise:
- 19 i) Let L be the included module list of C . Search each element of L in the reverse
- 20 order for a binding of a constant with name N .
- 21 ii) If the binding is found, the value of the *scoped-constant-reference* is the value of
- 22 the binding.
- 23 iii) Otherwise, search for a binding of a constant with name N from Step e of §11.4.3.1.

24 **11.4.3.3 Global variables**

25 A *global-variable-identifier* is evaluated as follows:

- 26 • Let N be the *global-variable-identifier*.
- 27 • If a binding with name N exists in `[[global-variable-bindings]]`, the value of *global-variable-*
- 28 *identifier* is the value of the binding.
- 29 • Otherwise, the value of *global-variable-identifier* is `nil`.

30 **11.4.3.4 Class variables**

31 A *class-variable-identifier* is evaluated as follows:

- 1 a) Let N be the *class-variable-identifier*. Let C be the first class or module in the list at the
2 top of $\llbracket \text{class-module-list} \rrbracket$ which is not an eigenclass.
- 3 b) Let CS be the set of classes which consists of C and all the superclasses of C . Let MS be
4 the set of modules which consists of all the modules in the included module list of all classes
5 in CS . Let CM be the union of CS and MS .
- 6 c) If a binding with name N exists in the set of bindings of class variables of only one of the
7 classes or modules in CM , let V be the value of the binding.
- 8 d) If more than two classes or modules in CM have a binding with name N in the set of
9 bindings of class variables, let V be the value of one of these bindings. Which binding is
10 selected is implementation dependent.
- 11 e) If none of the classes or modules in CM has a binding with name N in the set of bindings
12 of class variables, let S be a direct instance of the class `Symbol` with name N and raise a
13 direct instance of the class `NameError` which has S as its name property.
- 14 f) The value of the *class-variable-identifier* is V .

15 11.4.3.5 Instance variables

16 An *instance-variable-identifier* is evaluated as follows:

- 17 • Let N be the *instance-variable-identifier*.
- 18 • If a binding with name N exists in the set of bindings of instance variables of the current
19 self, the value of the *instance-variable-identifier* is the value of the binding.
- 20 • Otherwise, the value of the *instance-variable-identifier* is `nil`.

21 11.4.3.6 Local variables

22 This subclause describes a *local-variable-identifier* which is a reference to a local variable (see
23 §9.1.2).

24 A *local-variable-identifier* is evaluated as follows:

- 25 a) Let N be the *local-variable-identifier*.
- 26 b) Search for a binding of a local variable with name N as described in §9.1.1.
- 27 c) If a binding is found, the value of *local-variable-identifier* is the value of the binding.
- 28 d) Otherwise, the value of *local-variable-identifier* is `nil`.

29 11.4.3.7 Pseudo variables

30 Syntax

31 *pseudo-variable* ::
32 *nil*

```
1      | true
2      | false
3      | self
```

4 **11.4.3.7.1** `nil`

5 **Syntax**

```
6      nil ::
7          nil
```

8 **Semantics**

9 The pseudo variable `nil` evaluates to the only instance of the class `NilClass` (see §15.2.4).

10 **11.4.3.7.2** `true` and `false`

11 **Syntax**

```
12      true ::
13          true
```

```
14      false ::
15          false
```

16 **Semantics**

17 The pseudo variable `true` evaluates to the only instance of the class `TrueClass` (see §15.2.5).

18 The pseudo variable `false` evaluates to the only instance of the class `FalseClass` (see §15.2.6).

19 **11.4.3.7.3** `self`

20 **Syntax**

```
21      self ::
22          self
```

23 **Semantics**

24 The pseudo variable `self` evaluates to the value of the current self.

1 11.4.4 Object constructors

2 11.4.4.1 Array constructor

3 Syntax

4 *array-constructor* ::
5 [*indexing-argument-list*?]

6 Semantics

7 An *array-constructor* is evaluated as follows:

- 8 a) If there is an *indexing-argument-list*, construct a list of arguments from the *indexing-*
9 *argument-list* as described in §11.2.1. Let *L* be the resulting list.
- 10 b) Otherwise, create an empty list of values *L*.
- 11 c) Create a direct instance of the class **Array** which stores the values in *L* in the same order
12 they are stored in *L*. Let *O* be the instance.
- 13 d) The value of the *array-constructor* is *O*.

14 11.4.4.2 Hash constructor

15 Syntax

16 *hash-constructor* ::
17 { (*association-list* ,?)? }

18 *association-list* ::
19 *association* (, *association*)*

20 *association* ::
21 *association-key* => *association-value*

22 *association-key* ::
23 *operator-expression*

24 *association-value* ::
25 *operator-expression*

26 The *operator-expression* of an *association-key* or *association-value* shall not be a *jump-expression*.

1 Semantics

2 Both *hash-constructors* or *association-lists* evaluate to a direct instance of the class **Hash** (see
3 §15.2.13).

4 A *hash-constructor* is evaluated as follows:

- 5 a) If there is an *association-list*, evaluate the *association-list*. The value of the *hash-constructor*
6 is the resulting value.
- 7 b) Otherwise, create an empty direct instance of the class **Hash**. The value of the *hash-*
8 *constructor* is the resulting instance.

9 An *association-list* is evaluated as follows:

- 10 a) Create a direct instance of the class **Hash** H .
- 11 b) For each *association* A_i , in the order it appears in the program text, take the following
12 steps:
 - 13 1) Evaluate the *operator-expression* of the *association-key* of A_i . Let K_i be the resulting
14 value.
 - 15 2) Evaluate the *operator-expression* of the *association-value*. Let V_i be the resulting value.
 - 16 3) Store a pair of K_i and V_i in H , as if by invoking the method `[]=` on H with K_i and V_i
17 as the arguments.
- 18 c) The value of the *association-list* is H .

19 11.4.4.3 Range constructor

20 Syntax

21 *range-constructor* ::
22 *logical-OR-expression*
23 | *logical-OR-expression*₁ *range-operator* *logical-OR-expression*₂

24 *range-operator* ::
25 ..
26 | ...

27 Semantics

28 A *range-constructor* of the form *logical-OR-expression*₁ *range-operator* *logical-OR-expression*₂
29 is evaluated as follows:

- 30 a) Evaluate the *logical-OR-expression*₁. Let A be the resulting value.
- 31 b) Evaluate the *logical-OR-expression*₂. Let B be the resulting value.

1 c) If the *range-operator* is the terminal `..`, construct a list *L* which contains three arguments:
2 *A*, *B*, and `false`.

3 If the *range-operator* is the terminal `...`, construct a list *L* which contains three arguments:
4 *A*, *B*, and `true`.

5 d) Invoke the method `new` on the class `Range` with *L* as the list of arguments. The value of
6 the *range-constructor* is the resulting value.

7 11.4.5 Literals

8 See §8.5.5.1 for integer literals.

9 See §8.5.5.1 for float literals.

10 See §8.5.5.2 for string literals.

11 See §8.5.5.5 for symbol literals.

12 See §8.5.5.4 for regular expression literals.

13 12 Statements

14 Syntax

```
15 statement ::  
16     expression-statement  
17     | alias-statement  
18     | undef-statement  
19     | if-modifier-statement  
20     | unless-modifier-statement  
21     | while-modifier-statement  
22     | until-modifier-statement  
23     | rescue-modifier-statement  
24     | assignment-statement
```

25 See §13.3.6 for *alias-statement*.

26 See §13.3.7 for *undef-statement*.

27 See §11.3.1 for *assignment-statement*.

28 Semantics

29 See §11.3.1 for *assignment-statement*.

30 12.1 The expression statement

31 Syntax

1 *expression-statement* ::
2 *expression*

3 **Semantics**

4 An *expression-statement* is evaluated as follows:

- 5 a) Evaluate the *expression*.
6 b) The resulting value is the value of the *expression-statement*.

7 **12.2 The if modifier statement**

8 **Syntax**

9 *if-modifier-statement* ::
10 *statement* [no line-terminator here] **if** *expression*

11 The *expression* of an *if-modifier-statement* shall not be a *jump-expression*.

12 **Semantics**

13 An *if-modifier-statement* of the form *S if E*, where *S* is the *statement* and *E* is the *expression*,
14 is evaluated as follows:

- 15 a) Evaluate the *if-expression* of the form **if E then S end**.
16 b) The resulting value is the value of the *if-modifier-statement*.

17 **12.3 The unless modifier statement**

18 **Syntax**

19 *unless-modifier-statement* ::
20 *statement* [no line-terminator here] **unless** *expression*

21 The *expression* of an *unless-modifier-statement* shall not be a *jump-expression*.

22 **Semantics**

23 An *unless-modifier-statement* of the form *S unless E*, where *S* is the *statement* and *E* is the
24 *expression*, is evaluated as follows:

- 25 a) Evaluate the *unless-expression* of the form **unless E then S end**.
26 b) The resulting value is the value of the *unless-modifier-statement*.

1 12.4 The while modifier statement

2 Syntax

3 *while-modifier-statement* ::
4 *statement* [no line-terminator here] **while** *expression*

5 The *expression* of a *while-modifier-statement* shall not be a *jump-expression*.

6 Semantics

7 A *while-modifier-statement* of the form *S while E*, where *S* is the *statement* and *E* is the
8 *expression*, is evaluated as follows:

- 9 a) Evaluate the *while-expression* of the form **while** *E* **do** *S* **end**.
10 b) The resulting value is the value of the *while-modifier-statement*.

11 12.5 The until modifier statement

12 Syntax

13 *until-modifier-statement* ::
14 *statement* [no line-terminator here] **until** *expression*

15 The *expression* of an *until-modifier-statement* shall not be a *jump-expression*.

16 Semantics

17 An *until-modifier-statement* of the form *S until E*, where *S* is the *statement* and *E* is the
18 *expression*, is evaluated as follows:

- 19 a) Evaluate the *until-expression* of the form **until** *E* **do** *S* **end**.
20 b) The resulting value is the value of the *until-modifier-statement*.

21 12.6 The rescue modifier statement

22 Syntax

23 *rescue-modifier-statement* ::
24 *main-statement-of-rescue-modifier-statement* [no line-terminator here]
25 **rescue** *fallback-statement-of-rescue-modifier-statement*

26 *main-statement-of-rescue-modifier-statement* ::
27 *statement*

```

1  fallback-statement-of-rescue-modifier-statement ::
2      statement but not statement-not-allowed-in-fallback-statement

3  statements-not-allowed-in-fallback-statement ::
4      keyword-AND-expression
5      | keyword-OR-expression
6      | if-modifier-statement
7      | unless-modifier-statement
8      | while-modifier-statement
9      | until-modifier-statement
10     | rescue-modifier-statement

```

11 Semantics

12 A *rescue-modifier-statement* is evaluated as follows:

- 13 a) Evaluate the *main-statement-of-rescue-modifier-statement*. Let V be the resulting value.
- 14 b) If an instance of the class **StandardError** is raised and not handled in Step a, evalu-
15 ate *fallback-statement-of-rescue-modifier-statement*. The resulting value is the value of the
16 *rescue-modifier-statement*.
- 17 c) If no instances of the class **Exception** are raised in Step a, or all the instances of the class
18 **Exception** raised in Step a are handled in Step a, the value of the *rescue-modifier-statement*
19 is V .

20 13 Classes and modules

21 13.1 Modules

22 13.1.1 General description

23 Every module is an instance of the class **Module** (see §15.2.2). However, not every instance of
24 the class **Module** is a module because the class **Module** is a superclass of the class **Class**, an
25 instance of which is not a module, but a class.

26 Modules have the following attributes:

27 **Included module list:** An ordered list of modules included in the module. Module inclu-
28 sion is described in §13.1.3.

29 **Constants:** A set of bindings of constants.

30 A binding of a constant is created by the following program constructs:

- 31 • Assignments (see §11.3.1)
- 32 • Module definitions (see §13.1.2)

- Class definitions (see §13.2.2)

Class variables: A set of bindings of class variables. A binding of a class variable is created by an assignment (see §11.3.1).

Instance methods: A set of method bindings. A method binding is created by a method definition (see §13.3.1) or a singleton method definition (see §13.4.3). The value of a method binding may be **undef**, which is the flag indicating that a method cannot be invoked (see §13.3.7).

13.1.2 Module definition

Syntax

```
module-definition ::  
    module module-path module-body end  
  
module-path ::  
    top-module-path  
    | module-name  
    | nested-module-path  
  
module-name ::  
    constant-identifier  
  
top-module-path ::  
    :: module-name  
  
nested-module-path ::  
    primary-expression [no line-terminator here] :: module-name  
  
module-body ::  
    body-statement
```

Semantics

A *module-definition* is evaluated as follows:

- Determine the class or module in which a binding with name *module-name* is to be created or modified as follows:
 - If the *module-path* is of the form *top-module-path*, let *C* be the class **Object**.
 - If the *module-path* is of the form *module-name*, let *C* be the current class or module.
 - If the *module-path* is of the form *nested-module-path*, evaluate the *primary-expression*. If the resulting value is an instance of the class **Module**, let *C* be the instance. Otherwise, raise a direct instance of the class **TypeError**.

- 1 b) Let N be the *module-name*.
- 2 1) If a binding with name N exists in the set of bindings of constants of C , let B be this
- 3 binding. If the value of B is a module, let M be that module. Otherwise, raise a direct
- 4 instance of the class `TypeError`.
- 5 2) Otherwise, create a direct instance of the class `Module` and let M be that module.
- 6 Create a variable binding with name N and value M in the set of bindings of constants
- 7 of C .
- 8 c) Modify the execution context as follows:
 - 9 • Create a new list which has the same members as that of the list at the top of `[[class-`
 - 10 `module-list]]`, and add M to the head of the newly created list. Push the list onto
 - 11 `[[class-module-list]]`.
 - 12 • Push M onto `[[self]]`.
 - 13 • Push the public visibility onto `[[default-visibility]]`.
 - 14 • Push an empty set of bindings onto `[[local-variable-bindings]]`.
- 15 d) Evaluate the *module-body*. The value of the *module-definition* is the value of the *module-*
- 16 *body*.
- 17 e) Restore the execution context by removing the elements from the tops of `[[class-module-`
- 18 `list]]`, `[[self]]`, `[[default-visibility]]`, and `[[local-variable-bindings]]`, even when an exception is
- 19 raised and not handled during Step d.

20 13.1.3 Module inclusion

21 Modules and classes can be extended by including other modules into them. When a module is

22 included, the instance methods, the class variables, and the constants of the included module

23 are available to the including class or module (see §11.4.3.4, §13.3.3, and §11.4.3.1).

24 Modules and classes can include other modules by invoking the method `include` (see §15.2.2.3.27)

25 or the method `extend` (see §15.3.1.2.13).

26 A module M is included in another module N if and only if M is an element of the included

27 module list of N . A module M is included in a class C if and only if M is an element of the

28 included module list of C , or M is included in one of the superclasses of C .

29 13.2 Classes

30 13.2.1 General description

31 Every class is an instance of the class `Class` (see §15.2.3), which is a direct subclass of the class

32 `Module`.

33 Classes have the same set of attributes as modules. In addition, each class has a single direct

34 superclass.

1 13.2.2 Class definition

2 Syntax

3 *class-definition* ::
4 **class** *class-path* [no line-terminator here] *superclass* *class-body* **end**

5 *class-path* ::
6 *top-class-path*
7 | *class-name*
8 | *nested-class-path*

9 *class-name* ::
10 *constant-identifier*

11 *top-class-path* ::
12 :: *class-name*

13 *nested-class-path* ::
14 *primary-expression* [no line-terminator here] :: *class-name*

15 *superclass* ::
16 *separator*
17 | < *expression* *separator*

18 *class-body* ::
19 *body-statement*

20 Semantics

21 A *class-definition* is evaluated as follows:

22 a) Determine the class or module in which the binding with name *class-name* is to be created
23 or modified as follows:

- 24 • If the *class-path* is of the form *top-class-path*, let *M* be the class **Object**.
25 • If the *class-path* is of the form *class-name*, let *M* be the current class or module.
26 • If the *class-path* is of the form *nested-class-path*, evaluate the *primary-expression*. If
27 the resulting value is an instance of the class **Module**, let *M* be the instance. Otherwise,
28 raise a direct instance of the class **TypeError**.

29 b) Let *N* be the *class-name*.

- 30 1) If a binding with name *N* exists in the set of bindings of constants of *M*, let *B* be that
31 binding.

- 1 i) If the value of B is an instance of the class **Class**, let C be the instance. Otherwise,
2 raise a direct instance of the class **TypeError**.
- 3 ii) If the *superclass* occurs, evaluate it. If the resulting value does not correspond to
4 the direct superclass of C , raise a direct instance of the class **TypeError**.
- 5 2) Otherwise, create a direct instance of the class **Class**. Let C be that class.
- 6 i) If the *superclass* occurs, evaluate it. If the resulting value is not an instance
7 of the class **Class**, raise a direct instance of the class **TypeError**. If the value
8 of *superclass* is an eigenclass or the class **Class**, the behavior is implementation
9 dependent. Otherwise, let the direct superclass of C be the value of the *superclass*.
- 10 ii) If the *superclass* of the *class-definition* does not occur, let the direct superclass of
11 C be the class **Object**.
- 12 iii) Create an eigenclass, and associate it with C . The eigenclass shall have the eigen-
13 class of the direct superclass of C as one of its superclasses.
- 14 iv) Create a variables binding with name N and value C in the set of bindings of
15 constants of M .
- 16 c) Modify the execution context as follows:
 - 17 • Create a new list which has the same members as that of the list at the top of \llbracket class-
18 module-list \rrbracket , and add C to the head of the newly created list. Push the list onto
19 \llbracket class-module-list \rrbracket .
 - 20 • Push C onto \llbracket self \rrbracket .
 - 21 • Push the public visibility onto \llbracket default-visibility \rrbracket .
 - 22 • Push an empty set of bindings onto \llbracket local-variable-bindings \rrbracket .
- 23 d) Evaluate the *class-body*. The value of the *class-definition* is the value of the *class-body*.
- 24 e) Restore the execution context by removing the elements from the tops of \llbracket class-module-
25 list \rrbracket , \llbracket self \rrbracket , \llbracket default-visibility \rrbracket , and \llbracket local-variable-bindings \rrbracket , even when an exception is
26 raised and not handled during Step d.

27 13.2.3 Inheritance

28 A class inherits attributes of its superclasses. Inheritance means that a class implicitly contains
29 all attributes of its superclasses, as described below:

- 30 • Constants and class variables of superclasses can be referenced (see §11.4.3.1 and §11.4.3.4).
- 31 • Singleton methods of superclasses can be invoked (see §13.4).
- 32 • Instance methods defined in superclasses can be invoked on an instance of their subclasses
33 (see §13.3.3).

1 13.2.4 Instance creation

2 A direct instance of a class can be created by invoking the method `new` on the class (see
3 §15.2.3.2.3).

4 13.3 Methods

5 13.3.1 Method definition

6 Syntax

7 *method-definition* ::
8 `def` *method-name* [no *line-terminator* here] *method-parameter-part*
9 *method-body* `end`

10 *method-name* ::
11 *method-identifier*
12 | *operator-method-name*
13 | *reserved-word*

14 *method-body* ::
15 *body-statement*

16 The following constructs shall not occur in the *method-parameter-part* or the *method-body*:

- 17 • A *class-definition*.
- 18 • A *module-definition*.
- 19 • A *single-variable-assignment*, where its *variable* is a *constant-identifier*.
- 20 • A *scoped-constant-assignment*.
- 21 • A *multiple-assignment-statement*, where the form of any of the *left-hand-sides* which occurs
22 in it is any of the following:
 - 23 — *constant-identifier*
 - 24 — *primary-expression* [no *line-terminator* here] (`.` | `::`) (*local-variable-identifier* | *constant-*
25 *identifier*)
 - 26 — `::` *constant-identifier*.

27 However, those constructs may occur within an *eigenclass-definition* in the *method-parameter-*
28 *part* or the *method-body*.

29 Semantics

30 A method is defined by a *method-definition* and has the *method-parameter-part* and the *method-*
31 *body* of the *method-definition*. In addition, a method has the following attributes:

Class module list: The list of classes and modules which is the top element of `[[class-module-list]]` when the method is defined.

Defined name: The name with which the method is defined.

Visibility: The visibility of the method (see §13.3.5).

A *method-definition* is evaluated as follows:

a) Let *N* be the *method-name*.

b) Create a method *U* defined by the *method-definition*. Initialize the attributes of *U* as follows:

- The class module list is the element at the top of `[[class-module-list]]`.

- The defined name is *N*.

- The visibility is:

- If the current class or module is an eigenclass, then the current visibility.

- Otherwise, if *N* is `initialize` or `initialize_copy`, then the private visibility.

- Otherwise, the current visibility.

c) If a method binding with name *N* exists in the set of bindings of instance methods of the current class or module, let *V* be the value of that binding.

1) If *V* is `undef`, the evaluation of the *method-definition* is implementation defined.

2) Replace the value of the binding with *U*.

d) Otherwise, create a method binding with name *N* and value *U* in the set of bindings of instance methods of the current class or module.

e) The value of the *method-definition* is implementation defined.

13.3.2 Method parameters

Syntax

method-parameter-part ::
 (*parameter-list?*)
 | *parameter-list?* *separator*

parameter-list ::
 mandatory-parameter-list , *optional-parameter-list?* ,
 array-parameter? , *block-parameter?*
 | *optional-parameter-list* , *array-parameter?* , *block-parameter?*


```

1      | array-parameter , block-parameter?
2      | block-parameter

3  mandatory-parameter-list ::
4      mandatory-parameter
5      | mandatory-parameter-list , mandatory-parameter

6  mandatory-parameter ::
7      local-variable-identifier

8  optional-parameter-list ::
9      optional-parameter
10     | optional-parameter-list , optional-parameter

11 optional-parameter ::
12     optional-parameter-name = default-parameter-expression

13 optional-parameter-name ::
14     local-variable-identifier

15 default-parameter-expression ::
16     operator-expression

17 array-parameter ::
18     * array-parameter-name
19     | *

20 array-parameter-name ::
21     local-variable-identifier

22 block-parameter ::
23     & block-parameter-name

24 block-parameter-name ::
25     local-variable-identifier

```

26 All the *local-variable-identifiers* of *mandatory-parameters*, *optional-parameter-names*, *array-*
27 *parameter-name*, and *block-parameter-name* of a *parameter-list* shall be pairwise different.

28 Semantics

29 There are four kinds of parameters as described below. How those parameters are bound to the
30 actual arguments is described in §13.3.3.

31 **Mandatory parameters:** These parameters are represented by *mandatory-parameters*.

For each mandatory parameter, a corresponding actual argument shall be given when the method is invoked.

Optional parameters: These parameters are represented by *optional-parameters*. Each optional parameter consists of a parameter name represented by *optional-parameter-name* and an expression represented by *default-parameter-expression*. For each optional parameter, when there is no corresponding argument in the list of arguments given to the method invocation, the value of the *default-parameter-expression* is used as the value of the argument.

An array parameter: This parameter is represented by *array-parameter-name*. Let N be the number of arguments, excluding a block argument, given to a method invocation. If N is more than the sum of the number of mandatory arguments and optional arguments, this parameter is bound to a direct instance of the class **Array** containing the extra arguments excluding a block argument. Otherwise, the parameter is bound to an empty direct instance of the class **Array**. If an *array-parameter* is of the form “*”, those extra arguments are ignored.

A block parameter: This parameter is represented by *block-parameter-name*. The parameter is bound to the block passed to the method invocation.

13.3.3 Method invocation

The way in which a list of arguments is created are described in §11.2.

Given the receiver R , the method name M , and the list of arguments A , take the following steps:

- a) If the method is invoked with a block, let B be the block. Otherwise, let B be block-not-given.
- b) Let C be the eigenclass of R if R has an eigenclass. Otherwise, let C be the class of R .
- c) Search for a method binding with name M , starting from C as described in §13.3.4.
- d) If a binding is found and its value is not undef, let V be the value of the binding.
- e) Otherwise, if M is **method_missing**, the behavior is implementation dependent. If M is not **method_missing**, add a direct instance of the class **Symbol** with name M to the head of A , and invoke the method **method_missing** on R with A as arguments and B as the block. Let O be the resulting value, and go to Step j.
- f) If the method is not invoked internally by a Ruby processor, check the visibility of V to see whether the method can be invoked (see §13.3.5). If the method cannot be invoked, add a direct instance of the class **Symbol** with name M to the head of A , and invoke the method **method_missing** on R with A as arguments and B as the block. Let O be the resulting value, and go to Step j.
- g) Modify the execution context as follows:
 - Push the class module list of V onto `[[class-module-list]]`.
 - Push R onto `[[self]]`.

- 1 • Push M onto $\llbracket \text{invoked-method-name} \rrbracket$.
- 2 • Push the public visibility to $\llbracket \text{default-visibility} \rrbracket$.
- 3 • Push the defined name of V onto $\llbracket \text{defined-method-name} \rrbracket$.
- 4 • Push B onto $\llbracket \text{block} \rrbracket$.
- 5 • Push an empty set of local variable bindings onto $\llbracket \text{local-variable-bindings} \rrbracket$.
- 6 h) Evaluate the *method-parameter-part* of V as follows:
 - 7 1) Let L be the *parameter-list* of the *method-parameter-part*.
 - 8 2) Let P_m , P_o , and P_a be the *mandatory-parameters* of the *mandatory-parameter-list*,
 9 the *optional-parameters* of the *optional-parameter-list*, and the *array-parameter* of L ,
 10 respectively. Let N_A , N_{P_m} , and N_{P_o} be the number of elements of A , P_m , and P_o
 11 respectively. If there are no *mandatory-parameters* or *optional-parameters*, let N_{P_m}
 12 and N_{P_o} be 0. Let S_b be the current set of local variable bindings.
 - 13 3) If N_A is smaller than N_{P_m} , raise a direct instance of the class **ArgumentError**.
 - 14 4) If the method does not have P_a and N_A is larger than the sum of N_{P_m} and N_{P_o} , raise
 15 a direct instance of the class **ArgumentError**.
 - 16 5) Otherwise, for each argument A_i in A , in the same order in A , take the following steps:
 - 17 i) Let P_i be the *mandatory-parameter* or the *optional-parameter* whose position in
 18 the L corresponds to the position of A_i in A .
 - 19 • If such P_i exists, let n be the *mandatory-parameter* if P_i is a mandatory
 20 parameter, or *optional-parameter-name* if P_i is an optional parameter. Create
 21 a variable binding with name n and value A_i in S_b .
 - 22 • If such P_i does not exist, i.e. if N_A is larger than the sum of N_{P_m} and N_{P_o} ,
 23 and P_a exists:
 - 24 I) Create a direct instance of the class **Array** X whose length is the number
 25 of extra arguments.
 - 26 II) Store each extra arguments into X , preserving the order in which they
 27 occur in the list of arguments.
 - 28 III) Let n be the *array-parameter-name* of P_a .
 - 29 IV) Create a variable binding with name n and value X in S_b .
 - 30 ii) If N_A is smaller than the sum of N_{P_m} and N_{P_o} :
 - 31 I) For each optional argument P_{O_i} to which no argument corresponds, evaluate
 32 the *default-parameter-expression* of P_{O_i} , and let V be the resulting value.

- 1 II) Let n be the *optional-parameter-name* of P_{O_i} .
- 2 III) Create a variable binding with name n and value V in S_b .
- 3 iii) If N_A is smaller than or equal to the sum of N_{P_m} and N_{P_o} , and P_a exists:
 - 4 I) Create an empty direct instance of the class **Array** V .
 - 5 II) Let n be the *array-parameter-name* of P_a .
 - 6 III) Create a variable binding with name n and value V in S_b .
 - 7 iv) If the *block-parameter* of L occurs, let D be the top of $\llbracket \text{block} \rrbracket$.
 - 8 I) If D is block-not-given, let V be **nil**.
 - 9 II) Otherwise, invoke the method **new** on the class **Proc** with an empty list of
 - 10 arguments and D as the block. Let V be the resulting value of the method
 - 11 invocation.
 - 12 III) Let n be the *block-parameter-name* of *block-parameter*.
 - 13 IV) Create a variable binding with name n and value V in S_b .
 - 14 i) Evaluate the *method-body* of V .
 - 15 • If the evaluation of the *method-body* is terminated by a *return-expression*:
 - 16 — If the *jump-argument* of the *return-expression* occurs, let O be the value of the
 - 17 *jump-argument*.
 - 18 — Otherwise, let O be **nil**.
 - 19 • Otherwise, let O be the resulting value of the evaluation.
 - 20 j) Restore the execution context by removing the elements from the tops of $\llbracket \text{class-module-list} \rrbracket$,
 - 21 $\llbracket \text{self} \rrbracket$, $\llbracket \text{invoked-method-name} \rrbracket$, $\llbracket \text{default-visibility} \rrbracket$, $\llbracket \text{defined-method-name} \rrbracket$, $\llbracket \text{block} \rrbracket$, and
 - 22 $\llbracket \text{local-variable-bindings} \rrbracket$, even when an exception is raised and not handled in Step i.
 - 23 k) The value of the method invocation is O .

24 The method invocation or the super expression (see Step d of §11.2.3) which corresponds to the
 25 set of items on the tops of all the attributes of the execution context modified in Step g, except
 26 $\llbracket \text{local-variable-bindings} \rrbracket$, is called the **current method invocation**.

27 13.3.4 Method lookup

28 Method lookup is the process by which a binding of an instance method is resolved.

29 Given a method name M and a class or a module C which is initially searched for the binding
 30 of the method, the method binding is resolved as follows:

- 1 a) If a method binding with name M exists in the set of bindings of instance methods of C ,
2 let B be that binding.
- 3 b) Otherwise, let L_m be the list of included modules of C . Search for a method binding with
4 name M in the set of bindings of instance methods of each module in L_m . Examine modules
5 in L_m in reverse order.
 - 6 1) If a binding is found, let B be this binding.
 - 7 2) Otherwise:
 - 8 • If the direct superclass of C is `nil`, the binding is considered not resolved.
 - 9 • Otherwise, replace C with the direct superclass of C , and continue processing from
10 Step a.
 - 11 c) B is the resolved method binding.

12 13.3.5 Method visibility

13 Methods are categorized into one of public, private, or protected methods according to the con-
14 ditions on which the method invocation is allowed. The attribute of a method which determines
15 these conditions is called the **visibility** of the method.

16 13.3.5.1 Public methods

17 A public method is a method whose visibility is the public visibility.

18 A public method can be invoked on an object anywhere within a program.

19 13.3.5.2 Private methods

20 A private method is a method whose visibility is the private visibility.

21 A private method can not be invoked with explicit receiver, i.e. method invocations of the
22 forms where *primary-expression* or *chained-method-invocation* occurs at the position which cor-
23 responds to the method receiver are not allowed, except for method invocations of the following
24 forms where the *primary-expression* is `self`.

- 25 • *single-method-assignment*
- 26 • *abbreviated-method-assignment*
- 27 • *single-indexing-assignment*
- 28 • *abbreviated-indexing-assignment*

29 13.3.5.3 Protected methods

30 A protected method is a method whose visibility is the protected visibility.

31 A protected method can be invoked if and only if the following condition holds:

- 1 • Let M be an instance of the class `Module` in which the binding of the method exists.
- 2 M is included in the current self, or M is the class of the current self or one of its superclasses.
- 3 If M is an eigenclass, whether the method can be invoked or not may be determined in a
- 4 implementation defined way.

5 13.3.5.4 Visibility change

6 The visibility of methods can be changed with built-in methods `public` (§15.2.2.3.38), `private`

7 (§15.2.2.3.36), and `protected` (§15.2.2.3.37), which are defined in the class `Module`.

8 13.3.6 The alias statement

9 Syntax

10 *alias-statement* ::

11 **alias** *new-name* *aliased-name*

12 *new-name* ::

13 *method-name*

14 | *symbol*

15 *aliased-name* ::

16 *method-name*

17 | *symbol*

18 Semantics

19 An *alias-statement* is evaluated as follows:

20 a) Evaluate the *new-name* as follows:

- 21 • If the *new-name* is of the form *method-name*, let N be the *method-name*.
- 22 • If the *new-name* is of the form *symbol*, evaluate it. Let N be the name of the resulting
- 23 instance of the class `Symbol`.

24 b) Evaluate the *aliased-name* as follows:

- 25 • If *aliased-name* is of the form *method-name*, let A be the *method-name*.
- 26 • If *aliased-name* is of the form *symbol*, evaluate it. Let A be the name of the resulting
- 27 instance of the class `Symbol`.

28 c) Let C be the current class or module.

29 d) Search for a method binding with name A , starting from C as described in §13.3.4.

- 1 e) If a binding is found and its value is not undef, let V be the value of the binding.
- 2 f) Otherwise, let S be a direct instance of the class **Symbol** with name A and raise a direct
3 instance of the class **NameError** which has S as its name property.
- 4 g) If a method binding with name N exists in the set of bindings of instance methods of the
5 current class or module, replace the value of the binding with V .
- 6 h) Otherwise, create a method binding with name N and value V in the set of bindings of
7 instance methods of the current class or module.
- 8 i) The value of *alias-statement* is **nil**.

9 **13.3.7 The undef statement**

10 **Syntax**

```

11  undef-statement ::
12      undef undef-list

13  undef-list ::
14      method-name-or-symbol ( , method-name-or-symbol )*

15  method-name-or-symbol ::
16      method-name
17      | symbol

```

18 **Semantics**

19 An *undef-statement* is evaluated as follows:

- 20 a) For each *method-name-or-symbol* of the *undef-list*, take the following steps:
 - 21 1) Let C be the current class or module.
 - 22 2) If the *method-name-or-symbol* is of the form *method-name*, let N be the *method-name*.
23 Otherwise, evaluate the *symbol*. Let N be the name of the resulting instance of the
24 class **Symbol**.
 - 25 3) Search for a method binding with name N , starting from C as described in §13.3.4.
 - 26 4) If a binding is found and its value is not undef:
 - 27 i) If the binding is found in C , replace the value of the binding with undef.
 - 28 ii) Otherwise, create a method binding with name N and value undef in the set of
29 bindings of instance methods of C .
 - 30 5) Otherwise, let S be a direct instance of the class **Symbol** with name N and raise a
31 direct instance of the class **NameError** which has S as its name property.

1 b) The value of *undef-statement* is **nil**.

2 13.4 Eigenclass

3 13.4.1 General description

4 An eigenclass is a class which is associated with a single object. An object, unless it is an
5 instances of the class **Class**, becomes associated with an eigenclass through a *singleton-method-*
6 *definition* or an *eigenclass-definition*. An instance of the class **Class** becomes associated with
7 an eigenclass when it is created.

8 The direct superclass of an eigenclass is implementation defined. However, an eigenclass shall
9 be a subclass of the class of the object with which it is associated.

10 NOTE 1 For example, the eigenclass of the class **Object** is a subclass of the class **Class** because the
11 class **Object** is a direct instance of the class **Class**. Therefore, the instance methods of the class **Class**
12 can be invoked on the class **Object**.

13 The eigenclass of a class whose direct superclass is not **nil** shall satisfy the following condition:

- 14 • Let E_c be the eigenclass of a class C , and let S be the direct superclass of C , and let E_s be
15 the eigenclass of S . Then, E_c have E_s as one of its superclasses.

16 NOTE 2 This requirement enables classes to inherit singleton methods from its superclasses. For
17 example, the eigenclass of the class **File** has the eigenclass of the class **I0** as its superclass. Thereby, the
18 class **File** inherits the singleton method **open** of the class **I0**.

19 The eigenclass of an object is unique in the sense that no two objects become associated with
20 the same eigenclass.

21 Although eigenclasses are instances of the class **Class**, they cannot create an instance of them-
22 selves. When the method **new** is invoked on an eigenclass, a direct instance of the class **TypeError**
23 shall be raised (see Step a of §15.2.3.2.3).

24 Whether an eigenclass can be a superclass of other classes is implementation dependent (see
25 Step b-2-i of §13.2.2 and Step c of §15.2.3.2.1).

26 Whether an eigenclass can have class variables or not is implementation defined.

27 13.4.2 Eigenclass definition

28 Syntax

29 *eigenclass-definition* ::
30 **class** << *expression separator eigenclass-body* **end**

31 *eigenclass-body* ::
32 *body-statement*

1 Semantics

2 An *eigenclass-definition* is evaluated as follows:

- 3 a) Evaluate the *expression*. Let O be the resulting value. A conforming processor may specify
4 the set of classes such that if O is an instance of one of the classes in the set, a direct
5 instance of the class `TypeError` is raised.
- 6 b) If O is one of `nil`, `true`, or `false`, let E be the class of O and go to Step e.
- 7 c) If O is not associated with an eigenclass, create a new eigenclass. Let E be the newly
8 created eigenclass.
- 9 d) If O is associated with an eigenclass, let E be that eigenclass.
- 10 e) Modify the execution context as follows:
 - 11 • Create a new list which consists of the same elements as the list at the top of `[[class-`
12 `module-list]]` and add E to the head of the newly created list. Push the list onto
13 `[[class-module-list]]`.
 - 14 • Push E onto `[[self]]`.
 - 15 • Push the public visibility onto `[[default-visibility]]`.
 - 16 • Push an empty set of bindings onto `[[local-variable-bindings]]`.
- 17 f) Evaluate the *eigenclass-body*. The value of the *eigenclass-definition* is the value of the
18 *eigenclass-body*.
- 19 g) Restore the execution context by removing the elements from the tops of `[[class-module-`
20 `list]]`, `[[self]]`, `[[default-visibility]]`, and `[[local-variable-bindings]]`, even when an exception is
21 raised and not handled during Step f.

22 13.4.3 Singleton method definition

23 Syntax

```
24 singleton-method-definition ::  
25   def singleton ( . | :: ) method-name [ no line-terminator here ]  
26     method-parameter-part method-body end  
  
27 singleton ::  
28   variable  
29   | pseudo-variable  
30   | ( expression )
```

31 Semantics

32 A *singleton-method-definition* is evaluated as follows:

- 1 a) Evaluate the *singleton*. Let S be the resulting value.
- 2 b) If S is one of `nil`, `true`, or `false`, let E be the class of O and go to Step e.
- 3 c) If S is not associated with an eigenclass, create a new eigenclass. Let E be the newly
4 created eigenclass.
- 5 d) If S is associated with an eigenclass, Let E be that eigenclass.
- 6 e) Let N be the *method-name*.
- 7 f) Create a method U defined by the *method-definition*. Initialize the attributes of U as
8 follows:
 - 9 • The class module list is the element at the top of `[[class-module-list]]`.
 - 10 • The defined name is N .
 - 11 • The visibility is the public visibility.
- 12 g) If a method binding with name N exists in the set of bindings of instance methods of E ,
13 let V be the value of that binding.
 - 14 1) If V is `undef`, the evaluation of the *singleton-method-definition* is implementation de-
15 fined.
 - 16 2) Replace the value of the binding with U .
- 17 h) Otherwise, create a method binding with name N and value U in the set of bindings of
18 instance methods of E .
- 19 i) The value of the *singleton-method-definition* is implementation defined.

20 14 Exceptions

21 If an instance of the class `Exception` is raised, the current evaluation process stops, and the
22 evaluation process is transferred to a program construct that can handle this exception.

23 14.1 Cause of exceptions

24 An exception is raised when:

- 25 • the method `raise` (see §15.3.1.1.13) is invoked.
- 26 • a certain exceptional condition occurs as described in various parts of this document.

27 Only instances of the class `Exception` shall be raised.

14.2 Exception handling

Exceptions are handled by a *body-statement*, an *assignment-with-rescue-modifier*, or a *rescue-modifier-statement*. These program constructs are called **exception handlers**. When an exception handler is handling an exception, the exception being handled is called the **current exception**.

When an exception is raised, it is handled by an exception handler. This exception handler is determined as follows:

- a) Let S be the innermost local variable scope which lexically encloses the location where the exception is raised, and which corresponds to one of a *program*, a *method-definition*, a *singleton-method-definition*, or a *block*.
- b) Test each exception handler in S which lexically encloses the location where the exception is raised from the innermost to the outermost.
 - An *assignment-with-rescue-modifier* is considered to handle the exception if the exception is an instance of the class `StandardError` (see §11.3.1.4), except when the exception is raised in its *operator-expression*₂. In this case, *assignment-with-rescue-modifier* does not handle the exception.
 - A *rescue-modifier-statement* is considered to handle the exception if the exception is an instance of the class `StandardError` (see §12.6), except when the exception is raised in its *fallback-statement-of-rescue-modifier-statement*. In this case, *rescue-modifier-statement* does not handle the exception.
 - A *body-statement* is considered to handle the exception if one of its *rescue-clauses* is considered to handle the exception (see §11.4.1.4.1), except when the exception is raised in one of its *rescue-clauses*, *else-clauses*, or *ensure-clauses*. In this case, *body-statement* does not handle the exception. If an *ensure-clause* of a *body-statement* occurs, it is evaluated even if the handler does not handle the exception (see §11.4.1.4.1).
- c) If an exception handler which can handle the exception is found in S , terminate the search for the exception handler. Continue evaluating the program as defined for the relevant construct (see §11.4.1.4.1, §11.3.1.4, and §12.6).
- d) If none of the exception handlers in S can handle the exception:
 - 1) If S corresponds to a *method-definition* or a *singleton-method-definition*, terminate Step h or Step i, and take Step j of the current method invocation. Continue the search from Step a, under the assumption that the exception is raised at the location where the method is invoked.
 - 2) If S corresponds to a *block*, terminate the evaluation of the current *block*. Continue the search from Step a, under the assumption that the exception is raised at the location where the block is called.
 - 3) Otherwise, terminate the evaluation of the *program*.

15 Built-in classes and modules

Built-in classes and modules are specified in this clause. Those classes and modules shall be defined in the class `Object` as constants.

15.1 General description

Each built-in class or module is specified by describing its attribute values, as described in §13.1 and §13.2.

When a clause specifying a built-in class or module contains a subclause titled “Included modules”, the built-in class or module shall include the modules listed in that subclause in the order of that listing.

Each subclause in the subclause titled “Singleton methods” with the title of the form *C.m* specifies the singleton method *m* of the class *C*.

Each subclause in the subclause titled “Instance methods” with the title of the form *C#m* specifies the instance method *m* of the class *C*.

The parameter specification of a method is described in the form of *method-parameter-part* (see §13.3.1).

EXAMPLE 1 The following example defines the parameter specification of a method `sample`.

```
sample( arg1, arg2, opt=expr, *ary, &blk )
```

For a singleton method, the method name is prefixed by the name of the class or the module, and a dot (`.`).

EXAMPLE 2 The following example defines the parameter specification of a singleton method `sample` of a class `SampleClass`:

```
SampleClass.sample( arg1, arg2, opt=expr, *ary, &blk )
```

Next to the parameter specification, the visibility and the behavior of the method are specified.

The visibility, which is any one of public, protected or private, is specified after the label named “Visibility:”.

The behavior, which is the steps which shall be taken while evaluating the *method-body* of the method (see Step i of §13.3.3), is specified after the label named “Behavior:”.

In these steps, a reference to the name of an argument in the parameter specification is considered to be the object bound to the local variables of the same name. The phrase “call the *block* with *X* as the argument” indicates that the block corresponding to the block parameter *block* shall be called as described in §11.2.2 with *X* as the argument to the block call. The phrase “return *X*” indicates that the evaluation of the *method-body* shall be terminated at that point, and *X* shall be the value of the *method-body*. The phrase “the name designated by *N*” means the result of the following steps:

- 1 a) If *N* is an instance of the class **Symbol**, the name of *N*.
- 2 b) If *N* is an instance of the class **String**, the content of *N*.
- 3 c) Otherwise, the behavior of the method is implementation dependent.

4 The class module list of an instance method of a built-in class or module shall be a list which
5 consists of two elements: the first is the built-in class or module; the second is the class **Object**.
6 The class module list of a singleton method of a built-in class or module shall be a list which
7 consists of two elements: the first is the eigenclass of the built-in class or module; the second is
8 the class **Object**.

9 A conforming processor may provide additional attributes and/or values: a specific initial
10 value for a predefined attribute whose initial value is not specified in this document, con-
11 stants, singleton methods, instance methods, and additional inclusion of modules into built-in
12 classes/modules.

13 **15.2 Built-in classes**

14 **15.2.1 Object**

15 The class **Object** is an implicit direct superclass for other classes; that is, if the direct superclass
16 of a class is not specified explicitly in the class definition, the direct superclass of the class is
17 the class **Object** (see §13.2.2).

18 All built-in classes and modules can be referenced through constants of the class **Object**.

19 **15.2.1.1 Direct superclass**

20 The value **nil**.

21 A conforming processor may define a sequence of superclasses of the class **Object**. However, the
22 direct superclass of the class at the top of the class hierarchy shall always be **nil**.

23 **15.2.1.2 Included modules**

24 The following module is included in the class **Object**.

- 25 • **Kernel**

26 **15.2.1.3 Constants**

27 The following constants are defined in the class **Object**.

28 **STDIN**: An implementation defined instance of the class **I0**, which shall be readable.

29 **STDOUT**: An implementation defined instance of the class **I0**, which shall be writable.

30 **STDERR**: An implementation defined instance of the class **I0**, which shall be writable.

31 Besides, every built-in class or module, including the class **Object** itself, shall be defined in the
32 class **Object** as a constant, whose name is the name of the class or module, and whose value is
33 the class or module.

1 15.2.1.4 Instance methods

2 15.2.1.4.1 Object#initialize

3 `initialize(*args)`

4 **Visibility:** private

5 **Behavior:** The method `initialize` is the default object initialization method, which is
6 invoked when an instance is created (see §13.2.4). It returns an implementation defined
7 value.

8 15.2.2 Module

9 All modules are instances of the class `Module`. Therefore, behaviors defined in the class `Module`
10 are shared by all modules.

11 The binary relation on the instances of the class `Module` denoted $A \sqsubset B$ is defined as follows:

- 12 • B is a module and B is included in A (see §13.1.3) or
13 • both A and B are classes and B is a superclass of A .

14 15.2.2.1 Direct superclass

15 The class `Object`

16 15.2.2.2 Singleton methods

17 15.2.2.2.1 Module.constants

18 `Module.constants`

19 **Visibility:** public

20 **Behavior:**

- 21 a) Create an empty direct instance of the class `Array`. Let A be the instance.
22 b) Let C be the current class or module. Let L be the list which consists of the same
23 elements as the list at the second element from the top of `[[class-module-list]]`, except
24 the last element, which is the class `Object`.

25 Let CS be the set of classes which consists of C and all the superclasses of C except
26 the class `Object`, but when C is the class `Object`, it shall be included in CS . Let MS
27 be the set of modules which consists of all the modules in the included module list of
28 all classes in CS . Let CM be the union of L , CS and MS .

- 29 c) For each class or module c in CM , and for each name N of a constant defined in c ,
30 take the following steps:

- 1 1) Let S be either a direct instance of the class **String** whose content is N or a
2 direct instance of the class **Symbol** whose name is N . Which of these classes of
3 instance is chosen as the value of S is implementation defined.
- 4 2) Unless A contains the element of the same name as S , when S is an instance of the
5 class **Symbol**, or the same content as S , when S is an instance of the class **String**,
6 append S to A .
- 7 d) Return A .

8 15.2.2.2.2 Module.nesting

9 Module.nesting

10 **Visibility:** public

11 **Behavior:** The method returns a new direct instance of the class **Array** which contains all
12 but the last element of the list at the second element from the top of the `[[class-module-list]]`
13 in the same order.

14 15.2.2.3 Instance methods

15 15.2.2.3.1 Module#<

16 <(*other*)

17 **Visibility:** public

18 **Behavior:** Let A be the *other*. Let R be the receiver of the method.

- 19 a) If A is not an instance of the class **Module**, raise a direct instance of the class **TypeError**.
- 20 b) If A and R is the same object, return **false**.
- 21 c) If $R \sqsubset A$, return **true**.
- 22 d) If $A \sqsubset R$, return **false**.
- 23 e) Otherwise, return **nil**.

24 15.2.2.3.2 Module#<=

25 <=(*other*)

26 **Visibility:** public

27 **Behavior:**

- 1 a) If the *other* and the receiver are the same object, return **true**.
2 b) Otherwise, the behavior is the same as the method `<` (see §15.2.2.3.1).

3 **15.2.2.3.3 Module#<=>**

4 `<=>(other)`

5 **Visibility:** public

6 **Behavior:** Let *A* be the *other*. Let *R* be the receiver of the method.

- 7 a) If *A* is not an instance of the class **Module**, return **nil**.
8 b) If *A* and *R* is the same object, return an instance of the class **Integer** whose value is
9 0.
10 c) If $R \sqsubset A$, return an instance of the class **Integer** whose value is -1.
11 d) If $A \sqsubset R$, return an instance of the class **Integer** whose value is 1.
12 e) Otherwise, return **nil**.

13 **15.2.2.3.4 Module#==**

14 `==(other)`

15 **Visibility:** public

16 **Behavior:** Same as the method `==` of the module **Kernel** (see §15.3.1.2.1).

17 **15.2.2.3.5 Module#===**

18 `===(object)`

19 **Visibility:** public

20 **Behavior:** The method behaves as if the method `kind_of?` were invoked on the *object*
21 with the receiver as the only argument (see §15.3.1.2.26).

22 **15.2.2.3.6 Module#>**

23 `>(other)`

24 **Visibility:** public

1 **Behavior:** Let A be the *other*. Let R be the receiver of the method.

2 a) If A is not an instance of the class `Module`, raise a direct instance of the class `TypeError`.

3 b) If A and R is the same object, return `false`.

4 c) If $R \sqsubset A$, return `false`.

5 d) If $A \sqsubset R$, return `true`.

6 e) Otherwise, return `nil`.

7 15.2.2.3.7 Module#>=

8 >=(*other*)

9 **Visibility:** public

10 **Behavior:**

- 11 a) If the *other* and the receiver are the same object, return `true`.
- 12 b) Otherwise, the behavior is the same as the method `>` (see §15.2.2.3.6).

13 15.2.2.3.8 Module#alias_method

14 alias_method(*new_name*, *aliased_name*)

15 **Visibility:** private

16 **Behavior:** Let C be the receiver of the method.

- 17 a) Let N be the name designated by the *new_name*. Let A be the name designated by
18 the *aliased_name*.
- 19 b) Take Step d through h of §13.3.6, assuming that A , C , and N in §13.3.6 to be A , C ,
20 and N in the above steps.
- 21 c) Return C .

22 15.2.2.3.9 Module#ancestors

23 ancestors

24 **Visibility:** public

25 **Behavior:**

- 1 a) Create an empty direct instance of the class `Array` A .
- 2 b) Let C be the receiver of the method.
- 3 c) If C is not an eigenclass, append C to A .
- 4 d) Append each element of the included module list of C , in the order in the list, to A .
- 5 e) If C is a class, replace C with the direct superclass of the current C .
- 6 f) If C is not `nil`, repeat from Step c.
- 7 g) Return A .

8 15.2.2.3.10 `Module#append_features`

9 `append_features(module)`

10 **Visibility:** private

11 **Behavior:** Let $L1$ and $L2$ be the included modules list of the receiver and the *module*
 12 respectively.

- 13 a) If the *module* and the receiver is the same object, the behavior is implementation
 14 dependent.
- 15 b) If the receiver is an element of $L2$, the behavior is implementation defined.
- 16 c) Otherwise, for each module M in $L1$, in the same order in $L1$, take the following steps:
 - 17 1) If M and the *module* are the same object, the behavior is implementation depen-
 18 dent.
 - 19 2) If M is not in $L2$, append M to the end of $L2$.
- 20 d) Append the receiver to $L2$.
- 21 e) Return the receiver.

22 15.2.2.3.11 `Module#attr`

23 `attr(symbol, writable=false)`

24 **Visibility:** private

25 **Behavior:** Let C be the method receiver.

- 26 a) If the *symbol* is not an instance of the class `Symbol`, the behavior is implementation
 27 dependent.

- 1 b) Let N be the name of the *symbol*.
- 2 c) If N is not of the form *local-variable-identifier* or *constant-identifier*, raise a direct
3 instance of the class `NameError` which has the *symbol* as its name property.
- 4 d) Define an instance method in C as if by evaluating the following method definition at
5 the location of the invocation. In the following method definition, N is N , and $@N$ is the
6 name which is N prefixed by “@”.

```
7               def N
8                @N
9               end
```

10

- 11 e) If the *writable* is `true`, define an instance method in C as if by evaluating the following
12 method definition at the location of the invocation. In the following method definition,
13 $N=$ is the name N postfixed by `=`, and $@N$ is the name which is N prefixed by “@”. The
14 choice of the parameter name is arbitrary, and `val` is chosen only for the expository
15 purpose.

```
16               def N=(val)
17                @N = val
18               end
```

19

- 20 f) Return `nil`.

21 15.2.2.3.12 Module#attr_accessor

```
22       attr_accessor(*symbol_list)
```

23 **Visibility:** private

24 **Behavior:**

- 25 a) For each element S of the *symbol_list*, invoke the method `attr` with S as the first
26 argument and `true` as the second argument (see §15.2.2.3.11).

- 27 b) Return `nil`.

28 15.2.2.3.13 Module#attr_reader

```
29       attr_reader(*symbol_list)
```

30 **Visibility:** private

31 **Behavior:**

1 a) For each element S of the *symbol_list*, invoke the method `attr` with S as the first
2 argument and `false` as the second argument (see §15.2.2.3.11).

3 b) Return `nil`.

4 15.2.2.3.14 Module#attr_writer

5 `attr_writer(*symbol_list)`

6 **Visibility:** private

7 **Behavior:**

8 a) For each element S of the *symbol_list*, invoke the method `attr` with S as the first
9 argument and `true` as the second argument, but skip Step d (see §15.2.2.3.11).

10 b) Return `nil`.

11 15.2.2.3.15 Module#class_eval

12 `class_eval(string = nil, &block)`

13 **Visibility:** public

14 **Behavior:**

15 a) Let M be the receiver.

16 b) If the *block* is given:

17 1) If the *string* is given, raise a direct instance of the class `ArgumentError`.

18 2) Call the *block* with implementation defined arguments as described in §11.2.2, and
19 let V be the resulting value. A conforming processor shall modify the execution
20 context just before Step d of §11.2.2 as follows:

21 • Create a new list which has the same members as those of the list at the top
22 of `[[class-module-list]]`, and add M to the head of the newly created list. Push
23 the list onto `[[class-module-list]]`.

24 • Push the receiver onto `[[self]]`.

25 • Push the public visibility onto `[[default-visibility]]`.

26 In Step d and e of §11.2.2, a conforming processor may ignore M which is added
27 to the head of the top of `[[class-module-list]]` as described above, except when
28 referring to the current class or module in a *method-definition* (see §13.3.1), an
29 *alias-statement* (see §13.3.6), or an *undef-statement* (see §13.3.7).

- 1 3) Return V .
- 2 c) If the *block* is not given:
 - 3 1) If the *string* is not an instance of the class **String**, the behavior is implementation

4 dependent.
 - 5 2) Modify the execution context as follows:
 - 6 • Create a new list which has the same members as those of the list at the top

7 of $\llbracket \text{class-module-list} \rrbracket$, and add M to the head of the newly created list. Push

8 the list onto $\llbracket \text{class-module-list} \rrbracket$.
 - 9 • Push the receiver onto $\llbracket \text{self} \rrbracket$.
 - 10 • Push the public visibility onto $\llbracket \text{default-visibility} \rrbracket$.
 - 11 3) Parse the content of the *string* as a *program* (see §10.1). If it fails, raise a direct

12 instance of the class **SyntaxError**.
 - 13 4) Evaluate the *program*. Let V be the resulting value of the evaluation.
 - 14 5) Restore the execution context by removing the elements from the tops of $\llbracket \text{class-}$

15 $\text{module-list} \rrbracket$, $\llbracket \text{self} \rrbracket$, and $\llbracket \text{default-visibility} \rrbracket$, even when an exception is raised and

16 not handled in Step c-3 or c-4.
 - 17 6) Return V .

18 In Step c-4, the *string* is evaluated under the new local variable scope in which references
 19 to *local-variable-identifiers* are resolved in the same way as in scopes created by *blocks* (see
 20 §9.1.1).

21 15.2.2.3.16 Module#class_variable_defined?

22 class_variable_defined?(*symbol*)

23 **Visibility:** public

24 **Behavior:** Let C be the receiver of the method.

- 25 a) Let N be the name designated by the *symbol*.
- 26 b) If N is not of the form *class-variable-identifier*, raise a direct instance of the class

27 **NameError** which has the *symbol* as its name property.
- 28 c) Search for a binding of the class variable with name N by taking Step b through d of

29 §11.4.3.4, assuming that C and N in §11.4.3.4 to be C and N in the above steps.
- 30 d) If a binding is found, return **true**.
- 31 e) Otherwise, return **false**.

1 **15.2.2.3.17** `Module#class_variable_get`

2 `class_variable_get(symbol)`

3 **Visibility:** private

4 **Behavior:** Let *C* be the receiver of the method.

- 5 a) Let *N* be the name designated by the *symbol*.
- 6 b) If *N* is not of the form *class-variable-identifier*, raise a direct instance of the class
7 `NameError` which has the *symbol* as its name property.
- 8 c) Search for a binding of the class variable with name *N* by taking Step b through d of
9 §11.4.3.4, assuming that *C* and *N* in §11.4.3.4 to be *C* and *N* in the above steps.
- 10 d) If a binding is found, return the value of the binding.
- 11 e) Otherwise, raise a direct instance of the class `NameError` which has the *symbol* as its
12 name property.

13 **15.2.2.3.18** `Module#class_variable_set`

14 `class_variable_set(symbol, obj)`

15 **Visibility:** private

16 **Behavior:** Let *C* be the receiver of the method.

- 17 a) Let *N* be the name designated by the *symbol*.
- 18 b) If *N* is not of the form *class-variable-identifier*, raise a direct instance of the class
19 `NameError` which has the *symbol* as its name property.
- 20 c) Search for a binding of the class variable with name *N* by taking Step b through d of
21 §11.4.3.4, assuming that *C* and *N* in §11.4.3.4 to be *C* and *N* in the above steps.
- 22 d) If a binding is found, replace the value of the binding with the *obj*.
- 23 e) Otherwise, create a variable binding with name *N* and value *obj* in the set of bindings
24 of class variables of *C*.

25 **15.2.2.3.19** `Module#class_variables`

26 `class_variables`

27 **Visibility:** public

Behavior: The method returns a direct instance of the class **Array** which consists of names of all class variables of the receiver. These names are represented by direct instances of either the class **String** or the class **Symbol**. Which of those classes is chosen is implementation defined.

15.2.2.3.20 **Module#const_defined?**

`const_defined?(symbol)`

Visibility: public

Behavior:

- a) Let *C* be the receiver of the method.
- b) Let *N* be the name designated by the *symbol*.
- c) If *N* is not of the form *constant-identifier*, raise a direct instance of the class **NameError** which has the *symbol* as its name property.
- d) If a binding with name *N* exists in the set of bindings of constants of *C*, return **true**.
- e) Otherwise, return **false**.

15.2.2.3.21 **Module#const_get**

`const_get(symbol)`

Visibility: public

Behavior:

- a) Let *N* be the name designated by the *symbol*.
- b) If *N* is not of the form *constant-identifier*, raise a direct instance of the class **NameError** which has the *symbol* as its name property.
- c) Search for a binding of a constant with name *N* from Step e of §11.4.3.1, assuming that *C* in §11.4.3.1 to be the receiver of the method.
- d) If a binding is found, return the value of the binding.
- e) Otherwise, return the value of the invocation of the method `const_missing` (See Step e-2 of §11.4.3.1).

15.2.2.3.22 **Module#const_missing**

1 `const_missing(symbol)`

2 **Visibility:** public

3 **Behavior:** The method `const_missing` is invoked when a binding of a constant does not
4 exist on a constant reference (see §11.4.3.1).

5 When the method is invoked, take the following steps:

- 6 a) Take Step a through c of §15.2.2.3.20.
- 7 b) Raise a direct instance of the class `NameError` which has the *symbol* as its name prop-
8 erty.

9 **15.2.2.3.23 Module#const_set**

10 `const_set(symbol, obj)`

11 **Visibility:** public

12 **Behavior:** Let *C* be the receiver of the method.

- 13 a) Let *N* be the name designated by the *symbol*.
- 14 b) If *N* is not of the form *constant-identifier*, raise a direct instance of the class `NameError`
15 which has the *symbol* as its name property.
- 16 c) If a binding with name *N* exists in the set of bindings of constants of *C*, replace the
17 value of the binding with the *obj*.
- 18 d) Otherwise, create a variable binding with *N* and value *obj* in the set of bindings of
19 constants of *C*.
- 20 e) Return the *obj*.

21 **15.2.2.3.24 Module#constants**

22 `constants`

23 **Visibility:** public

24 **Behavior:** The method returns a new direct instance of the class `Array` which consists
25 of names of all constants defined in the receiver. These names are represented by direct
26 instances of either the class `String` or the class `Symbol`. Which of those classes is chosen is
27 implementation defined.

28 **15.2.2.3.25 Module#extend_object**

1 `extend_object(object)`

2 **Visibility:** private

3 **Behavior:** Let *S* be the eigenclass of the *object*. The method behaves as if by invoking
4 the method `append_features` on the receiver with *S* as the only argument.

5 **15.2.2.3.26 Module#extended**

6 `extended(object)`

7 **Visibility:** private

8 **Behavior:** The method returns `nil`.

9 **15.2.2.3.27 Module#include**

10 `include(*module_list)`

11 **Visibility:** private

12 **Behavior:** Let *C* be the receiver of the method.

13 a) For each element *A* of the *module_list*, in the reverse order in the *module_list*, take the
14 following steps:

15 1) If *A* is not an instance of the class `Module`, raise a direct instance of the class
16 `TypeError`.

17 2) If *A* is an instance of the class `Class`, raise a direct instance of the class `TypeError`.

18 3) Invoke the method `append_features` on *A* with *C* as the only argument.

19 4) Invoke the method `included` on *A* with *C* as the only argument.

20 b) Return *C*.

21 **15.2.2.3.28 Module#include?**

22 `include?(module)`

23 **Visibility:** public

24 **Behavior:** Let *C* be the receiver of the method.

- 1 a) If the *module* is not an instance of the class **Module**, raise a direct instance of the class
- 2 **TypeError**.
- 3 b) If the *module* is an element of the included module list of *C*, return **true**.
- 4 c) Otherwise, if *C* is an instance of the class **Class**, and if the *module* is an element of
- 5 the included module list of one of the superclasses of *C*, then return **true**.
- 6 d) Otherwise, return **false**.

7 **15.2.2.3.29 Module#included**

8 **included**(*module*)

9 **Visibility:** private

10 **Behavior:** The method returns **nil**.

11 **15.2.2.3.30 Module#included_modules**

12 **included_modules**

13 **Visibility:** public

14 **Behavior:** Let *C* be the receiver of the method.

- 15 a) Create an empty direct instance of the class **Array** *A*.
- 16 b) Append each element of the included module list of *C*, in the reverse order, to *A*.
- 17 c) If *C* is an instance of the class **Class**, replace *C* with the direct superclass of the
- 18 current *C*
- 19 d) If *C* is not **nil**, repeat from Step b.
- 20 e) Return *A*.

21 **15.2.2.3.31 Module#initialize**

22 **initialize**(&*block*)

23 **Visibility:** private

24 **Behavior:**

- 25 a) If the *block* is given, call the *block* as if invoking the method **class_eval** of the class
- 26 **Module** on the receiver with no arguments and the *block* as the block.
- 27 b) Return an implementation defined value.

1 15.2.2.3.32 Module#initialize_copy

2 initialize_copy(*original*)

3 **Visibility:** private

4 **Behavior:**

- 5 a) Invoke the instance method `initialize_copy` defined in the module `Kernel` on the
6 receiver with the *original* as the argument.
- 7 b) If the receiver is associated with an eigenclass, let E_o be the eigenclass, and take the
8 following steps:
- 9 1) Create an eigenclass whose direct superclass is the direct superclass of E_o . Let E_n
10 be the eigenclass.
- 11 2) For each binding B_{v1} of the constants of E_o , create a variable binding with the
12 same name and value as B_{v1} in the set of bindings of constants of E_n .
- 13 3) For each binding B_{v2} of the class variables of E_o , create a variable binding with
14 the same name and value as B_{v2} in the set of bindings of class variables of E_n .
- 15 4) For each binding B_m of the instance methods of E_o , create a method binding with
16 the same name and value as B_m in the set of bindings of instance methods of E_n .
- 17 5) Associate the receiver with E_n .
- 18 c) If the receiver is an instance of the class `Class`, set the direct superclass of the receiver
19 to the direct superclass of the *original*.
- 20 d) Append each element of the included module list of the *original*, in the same order, to
21 the receiver.
- 22 e) For each binding B_{v3} of the constants of the *original*, create a variable binding with
23 the same name and value as B_{v3} in the set of bindings of constants of the receiver.
- 24 f) For each binding B_{v4} of the class variables of the *original*, create a variable binding
25 with the same name and value as B_{v4} in the set of bindings of class variables of the
26 receiver.
- 27 g) For each binding B_{m2} of the instance methods of the *original*, create a method binding
28 with the same name and value as B_{m2} in the set of bindings of instance methods of the
29 receiver.
- 30 h) Return an implementation defined value.

31 15.2.2.3.33 Module#instance_methods

1 `instance_methods(include_super=true)`

2 **Visibility:** public

3 **Behavior:** Let C be the receiver of the method.

4 a) Create an an empty direct instance of the class **Array**. Let A be the instance.

5 b) Let I be the set of bindings of instance methods of C . For each binding B of I , let N
6 be the name of B , and let V be the value of B , and take the following steps:

7 1) If V is undef, or the visibility of V is private, skip the next two steps.

8 2) Let S be either a direct instance of the class **String** whose content is N or a
9 direct instance of the class **Symbol** whose name is N . Which of the these classes of
10 instance is chosen as the value of S is implementation defined.

11 3) Unless A contains the element of the same name (if S is an instance of the class
12 **Symbol**) or the same content (if S is an instance of the class **String**) as S , append
13 S to A .

14 c) If the *include_super* is a true value:

15 1) For each module M in included module list of C , take Step b, assuming that C in
16 that step to be M .

17 2) Replace C with the direct superclass of C .

18 3) If C is not *nil*, repeat from Step b.

19 d) Return A .

20 **15.2.2.3.34 Module#method_defined?**

21 `method_defined?(symbol)`

22 **Visibility:** public

23 **Behavior:** Let C be the receiver of the method.

24 a) Let N be the name designated by the *symbol*.

25 b) Search for a binding of an instance method named N starting from C as described in
26 §13.3.4.

27 c) If a binding is found and its value is not undef, return **true**.

28 d) Otherwise, return **false**.

1 **15.2.2.3.35 Module#module_eval**

2 `module_eval(string = nil, &block)`

3 **Visibility:** public

4 **Behavior:** Same as the method `class_eval` (see §15.2.2.3.15)

5 **15.2.2.3.36 Module#private**

6 `private(*symbol-list)`

7 **Visibility:** private

8 **Behavior:** Same as the method `public` (see §15.2.2.3.38), except that the method changes
9 current visibility or visibilities of methods corresponding to each element of the *symbol-list*
10 to private.

11 **15.2.2.3.37 Module#protected**

12 `protected(*symbol-list)`

13 **Visibility:** private

14 **Behavior:** Same as the method `public` (see §15.2.2.3.38), except that the method changes
15 current visibility or visibilities of methods corresponding to each element of the *symbol-list*
16 to protected.

17 **15.2.2.3.38 Module#public**

18 `public(*symbol-list)`

19 **Visibility:** private

20 **Behavior:** Let *C* be the receiver of the method.

21 a) If the length of *symbol-list* is 0, change the current visibility to public and return *C*

22 b) Otherwise, for each element *S* of the *symbol-list*, take the following steps:

23 1) Let *N* be the name designated by *S*.

24 2) Search for a method binding with name *N* starting from *C* as described in §13.3.4.

25 3) If a binding is found and its value is not undef, let *V* the value of the binding.

- 1 4) Otherwise, raise a direct instance of the class **NameError** which has S as its name
2 property.
- 3 5) If C is the class or module in which the binding is found, change the visibility of
4 V to the public visibility.
- 5 6) Otherwise, define an instance method in C as if by evaluating the following method
6 definition. In the definition, N is N . The choice of the parameter name is arbitrary,
7 and **args** is chosen only for the expository purpose.

```

8                       def N(*args)
9                         super
10                       end
11

```

12 The attributes of the method created by the above definition are initialized as
13 follows:

- 14 i) The class module list is the element at the top of `[[class-module-list]]`.
- 15 ii) The defined name is the defined name of V .
- 16 iii) The visibility is the public visibility.
- 17 c) Return C .

18 **15.2.2.3.39 Module#remove_class_variable**

```

19       remove_class_variable( symbol )

```

20 **Visibility:** private

21 **Behavior:** Let C be the receiver of the method.

- 22 a) Let N be the name designated by the *symbol*.
- 23 b) If N is not of the form *class-variable-identifier*, raise a direct instance of the class
24 **NameError** which has the *symbol* as its name property.
- 25 c) If a binding with name N exists in the set of bindings of class variables of C , let V be
26 the value of the binding.
 - 27 1) Remove the binding from the set of bindings of class variables of C .
 - 28 2) Return V .
- 29 d) Otherwise, raise a direct instance of the class **NameError** which has the *symbol* as its
30 name property.

31 **15.2.2.3.40 Module#remove_const**

1 `remove_const(symbol)`

2 **Visibility:** private

3 **Behavior:** Let C be the receiver of the method.

4 a) Let N be the name designated by the *symbol*.

5 b) If N is not of the form *constant-identifier*, raise a direct instance of the class **NameError**
6 which has the *symbol* as its name property.

7 c) If a binding with name N exists in the set of bindings of constants of C , let V be the
8 value of the binding.

9 1) Remove the binding from the set of bindings of constants of C .

10 2) Return V .

11 d) Otherwise, raise a direct instance of the class **NameError** which has the *symbol* as its
12 name property.

13 **15.2.2.3.41 Module#remove_method**

14 `remove_method(*symbol_list)`

15 **Visibility:** private

16 **Behavior:** Let C be the receiver of the method.

17 a) For each element S of the *symbol_list*, take the following steps:

18 1) Let N be the name designated by S .

19 2) If a binding with name N exists in the set of bindings of instance methods of C ,
20 remove the binding from the set.

21 3) Otherwise, raise a direct instance of the class **NameError** which has S as its name
22 property. In this case, the remaining elements of the *symbol_list* are not processed.

23 b) Return C .

24 **15.2.2.3.42 Module#undef_method**

25 `undef_method(*symbol_list)`

26 **Visibility:** private

Behavior: Let *C* be the receiver of the method.

a) For each element *S* of the *symbol_list*, take following steps:

1) Let *N* be the name designated by *S*.

2) Take Step a-3 and a-4 of §13.3.7, assuming that *C* in §13.3.7 to be *C* and *N* in the above steps.

b) Return *C*.

15.2.3 Class

All classes are instances of the class **Class**. Therefore, behaviors defined in the class **Class** are shared by all classes.

A conforming processor shall undefine the instance methods **append_features** and **extend_object** of the class **Class**, as if by invoking the method **undef_method** on the class **Class** with instances of the class **Symbol** whoses names are “append_features” and “extend_object” as the arguments (see §15.2.2.3.42).

15.2.3.1 Direct superclass

The class **Module**

15.2.3.2 Instance methods

15.2.3.2.1 Class#initialize

```
initialize( superclass=Object, &block)
```

Visibility: private

Behavior:

a) If the receiver has its direct superclass, raise a direct instance of the class **TypeError**.

b) If the *superclass* is not an instance of the class **Class**, raise a direct instance of the class **TypeError**.

c) If the *superclass* is an eigenclass or the class **Class**, the behavior is implementation dependent.

d) Set the direct superclass of the receiver to the *superclass*.

e) Create an eigenclass, and associate it with the receiver. The eigenclass shall have the eigenclass of the *superclass* as one of its superclasses.

f) If the *block* is given, call the *block* as if invoking the method **class_eval** of the class **Module** on the receiver with no arguments and the *block* as the block.

g) Return an implementation defined value.

1 15.2.3.2.2 Class#initialize_copy

2 initialize_copy(*original*)

3 **Visibility:** private

4 **Behavior:**

- 5 a) If the direct superclass of the receiver has already been set, raise a direct instance of
6 the class `TypeError`.
- 7 b) If the receiver is an eigenclass, raise a direct instance of the class `TypeError`.
- 8 c) Invoke the instance method `initialize_copy` defined in the class `Module` on the re-
9 ceiver with the *original* as the argument.
- 10 d) Return an implementation defined value.

11 15.2.3.2.3 Class#new

12 new(**args*, &*block*)

13 **Visibility:** public

14 **Behavior:**

- 15 a) If the receiver is an eigenclass, raise a direct instance of the class `TypeError`.
- 16 b) Create a direct instance of the receiver which has no bindings of instance variables.
17 Let *O* be the newly created instance.
- 18 c) Invoke the method `initialize` on *O* with all the elements of the *args* as arguments
19 and the *block* as the block.
- 20 d) Return *O*.

21 15.2.3.2.4 Class#superclass

22 superclass

23 **Visibility:** public

24 **Behavior:** Let *C* be the receiver of the methods.

- 25 a) If *C* is an eigenclass, return an implementation defined value.
- 26 b) Otherwise, return the direct superclass of *C*.

1 15.2.4 NilClass

2 The class NilClass has only one instance, which is represented by the pseudo variable nil.

3 Instances of the class NilClass shall not be created by the method new of the class NilClass.
4 Therefore, a conforming processor shall undefine the singleton method new of the class NilClass,
5 as if by invoking the method undef_method on the eigenclass of the class NilClass with a direct
6 instance of the class Symbol whose name is “new” as the argument (see §15.2.2.3.42).

7 15.2.4.1 Direct superclass

8 The class Object

9 15.2.4.2 Instance methods

10 15.2.4.2.1 NilClass#&

11 &(other)

12 **Visibility:** public

13 **Behavior:** The method returns false.

14 15.2.4.2.2 NilClass#^

15 ^(other)

16 **Visibility:** public

17 **Behavior:**

18 a) If the other is a false value, return false.

19 b) Otherwise, return true.

20 15.2.4.2.3 NilClass#nil?

21 nil?

22 **Visibility:** public

23 **Behavior:** The method returns true.

24 15.2.4.2.4 NilClass#|

1 | (*other*)

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is a false value, return **false**.

5 b) Otherwise, return **true**.

6 **15.2.5 TrueClass**

7 The class **TrueClass** has only one instance, which is represented by the pseudo variable **true**.
8 **true** represents a logical true value.

9 Instances of the class **TrueClass** shall not be created by the method **new** of the class **TrueClass**.
10 Therefore, a conforming processor shall undefine the singleton method **new** of the class **TrueClass**,
11 as if by invoking the method **undef_method** on the eigenclass of the class **TrueClass** with a direct
12 instance of the class **Symbol** whose name is “new” as the argument (see §15.2.2.3.42).

13 **15.2.5.1 Direct superclass**

14 The class **Object**

15 **15.2.5.2 Instance methods**

16 **15.2.5.2.1 TrueClass#&**

17 &(*other*)

18 **Visibility:** public

19 **Behavior:**

20 a) If the *other* is a false value, return **false**.

21 b) Otherwise, return **true**.

22 **15.2.5.2.2 TrueClass#^**

23 ^(*other*)

24 **Visibility:** public

25 **Behavior:**

26 a) If the *other* is a false value, return **true**.

27 b) Otherwise, return **false**.

1 **15.2.5.2.3 TrueClass#to_s**

2 `to_s`

3 **Visibility:** public

4 **Behavior:** The method returns an instance of the class `String`, the content of which is
5 “true”.

6 **15.2.5.2.4 TrueClass#|**

7 `| (other)`

8 **Visibility:** public

9 **Behavior:** The method returns `true`.

10 **15.2.6 FalseClass**

11 The class `FalseClass` has only one instance, which is represented by the pseudo variable `false`.
12 `false` represents a logical false value.

13 Instances of the class `FalseClass` shall not be created by the method `new` of the class `FalseClass`.
14 Therefore, a conforming processor shall undefine the singleton method `new` of the class `FalseClass`,
15 as if by invoking the method `undef_method` on the eigenclass of the class `FalseClass` with a
16 direct instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.3.42).

17 **15.2.6.1 Direct superclass**

18 The class `Object`

19 **15.2.6.2 Instance methods**

20 **15.2.6.2.1 FalseClass#&**

21 `& (other)`

22 **Visibility:** public

23 **Behavior:** The method returns `false`.

24 **15.2.6.2.2 FalseClass#^**

25 `^ (other)`

26 **Visibility:** public

1 **Behavior:**

2 a) If the *other* is a false value, return **false**.

3 b) Otherwise, return **true**.

4 **15.2.6.2.3 FalseClass#to_s**

5 **to_s**

6 **Visibility:** public

7 **Behavior:** The method returns an instance of the class **String**, the content of which is
8 “false”.

9 **15.2.6.2.4 FalseClass#|**

10 **| (other)**

11 **Visibility:** public

12 **Behavior:**

13 a) If the *other* is a false value, return **false**.

14 b) Otherwise, return **true**.

15 **15.2.7 Numeric**

16 Instances of the class **Numeric** represent numbers. The class **Numeric** is a superclass of all the
17 other built-in classes which represent numbers.

18 The notation “the value of the instance *N* of the class **Numeric**” means the number which *N*
19 represent.

20 **15.2.7.1 Direct superclass**

21 The class **Object**

22 **15.2.7.2 Included modules**

23 The following module is included in the class **Numeric**.

24 • **Comparable**

25 **15.2.7.3 Instance methods**

26 **15.2.7.3.1 Numeric#+@**

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

```
+@

Visibility: public
Behavior: The method returns the receiver.

15.2.7.3.2 Numeric#-@

-@

Visibility: public
Behavior:
  a) Invoke the method coerce on the receiver with an instance of the class Integer whose
      value is 0 as the only argument. Let V be the resulting value.
    1) If V is an instance of the class Array which contains two elements, let F and S
       be the first and the second element of V respectively.
       i) Invoke the method - on F with S as the only argument.
       ii) Return the resulting value.
    2) Otherwise, raise a direct instance of the class TypeError.

15.2.7.3.3 Numeric#abs

abs

Visibility: public
Behavior:
  a) Invoke the method < on the receiver with an instance of the class Integer whose value
      is 0.
  b) If this invocation results in a true value, invoke the method -@ on the receiver and
      return the resulting value.
      Otherwise, return the receiver.

15.2.7.3.4 Numeric#coerce
```

1 `coerce(other)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the class of the receiver and the class of the *other* are the same class, let *X* and *Y*
5 be the *other* and the receiver, respectively.

6 b) Otherwise, let *X* and *Y* be instances of the class `Float` which are converted from the
7 *other* and the receiver, respectively. the *other* and the receiver are converted as follows:

8 1) Let *O* be the *other* or the receiver.

9 2) If *O* is an instance of the class `Float`, let *F* be *O*.

10 3) Otherwise:

11 i) If an invocation of the method `respond_to?` on *O* with a direct instance of
12 the class `Symbol` whose name is `to_f` as the argument results in a false value,
13 raise a direct instance of the class `TypeError`.

14 ii) Invoke the method `to_f` on *O* with no arguments, and let *F* be the resulting
15 value.

16 iii) If *F* is not an instance of the class `Float`, raise a direct instance of the class
17 `TypeError`.

18 4) If the value of *F* is `NaN`, the behavior is implementation dependent.

19 5) The converted value of *O* is *F*.

20 c) Create a direct instance of the class `Array` which consists of two elements: the first is
21 *X*; the second is *Y*.

22 d) Return the instance of the class `Array`.

23 15.2.8 Integer

24 Instances of the class `Integer` represent integers. The ranges of these integers are unbounded.

25 Instances of the class `Integer` shall not be created by the method `new` of the class `Integer`.
26 Therefore, a conforming processor shall undefine the singleton method `new` of the class `Integer`,
27 as if by invoking the method `undef_method` on the eigenclass of the class `Integer` with a direct
28 instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.3.42).

29 A conforming processor may define subclasses of the class `Integer` which differ only in the
30 ranges of the representing integer values. In this case, a conforming processor:

- 31 • shall define methods `+`, `-`, `*`, `/`, and `%` in all of these classes.

- 1 • shall not create a direct instance of the class `Integer`, but shall create a direct instance of
2 one of these subclasses, instead of the class `Integer`.

3 If a conforming processor does not define any subclass of the class `Integer`, it shall define
4 methods `+`, `-`, `*`, `/` and `%` in the class `Integer`.

5 **15.2.8.1 Direct superclass**

6 The class `Numeric`

7 **15.2.8.2 Instance methods**

8 **15.2.8.2.1 `Integer#+`**

9 `+(other)`

10 **Visibility:** public

11 **Behavior:**

12 a) If the *other* is an instance of the class `Integer`, return an instance of the class `Integer`
13 whose value is the sum of the values of the receiver and the *other*.

14 b) If the *other* is an instance of the class `Float`, let *R* be the value of the receiver as a
15 floating point number.

16 Return a direct instance of the class `Float` whose value is the sum of *R* and the value
17 of the *other*.

18 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
19 ment. Let *V* be the resulting value.

20 1) If *V* is an instance of the class `Array` which contains two elements, let *F* and *S*
21 be the first and the second element of *V* respectively.

22 i) Invoke the method `+` on *F* with *S* as the only argument.

23 ii) Return the resulting value.

24 2) Otherwise, raise a direct instance of the class `TypeError`.

25 **15.2.8.2.2 `Integer#-`**

26 `-(other)`

27 **Visibility:** public

28 **Behavior:**

- 1 a) If the *other* is an instance of the class **Integer**, return an instance of the class **Integer**
2 whose value is the result of subtracting the value of the *other* from the value of the
3 receiver.
- 4 b) If the *other* is an instance of the class **Float**, let R be the value of the receiver as a
5 floating point number.
6 Return a direct instance of the class **Float** whose value is the result of subtracting the
7 value of the *other* from R .
- 8 c) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
9 ment. Let V be the resulting value.
 - 10 1) If V is an instance of the class **Array** which contains two elements, let F and S
11 be the first and the second element of V respectively.
 - 12 i) Invoke the method **-** on F with S as the only argument.
 - 13 ii) Return the resulting value.
 - 14 2) Otherwise, raise a direct instance of the class **TypeError**.

15 **15.2.8.2.3 Integer#***

16 *****(*other*)

17 **Visibility:** public

18 **Behavior:**

- 19 a) If the *other* is an instance of the class **Integer**, return an instance of the class **Integer**
20 whose value is the result of multiplication of the values of the receiver and the *other*.
- 21 b) If the *other* is an instance of the class **Float**, let R be the value of the receiver as a
22 floating point number.
23 Return a direct instance of the class **Float** whose value is the result of multiplication
24 of R and the value of the *other*.
- 25 c) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
26 ment. Let V be the resulting value.
 - 27 1) If V is an instance of the class **Array** which contains two elements, let F and S
28 be the first and the second element of V respectively.
 - 29 i) Invoke the method ***** on F with S as the only argument.
 - 30 ii) Return the resulting value.
 - 31 2) Otherwise, raise a direct instance of the class **TypeError**.

1 **15.2.8.2.4 Integer# /**

2 */(other)*

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the *other* is an instance of the class **Integer**:
- 6 1) If the value of the *other* is 0, raise a direct instance of the class **ZeroDivisionError**.
- 7 2) Otherwise, let n be the value of the receiver divided by the value of the *other*.
8 Return an instance of the class **Integer** whose value is the largest integer smaller
9 than or equal to n .
- 10 b) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
11 ment. Let V be the resulting value.
- 12 1) If V is an instance of the class **Array** which contains two elements, let F and S
13 be the first and the second element of V respectively.
- 14 i) Invoke the method **/** on F with S as the only argument.
- 15 ii) Return the resulting value.
- 16 2) Otherwise, raise a direct instance of the class **TypeError**.

17 **15.2.8.2.5 Integer# %**

18 *%(other)*

19 **Visibility:** public

20 **Behavior:**

- 21 a) If the *other* is an instance of the class **Integer**:
- 22 1) If the value of the *other* is 0, raise a direct instance of the class **ZeroDivisionError**.
- 23 2) Otherwise, let x and y be the values of the receiver and the *other*.
- 24 i) Let t be the largest integer smaller than or equal to x divided by y .
- 25 ii) Let m be $x - t \times y$.
- 26 iii) If $m \times y < 0$, return an instance of the class **Integer** whose value is $m + y$.

- 1 iv) Otherwise, return an instance of the class **Integer** whose value is m .
- 2 b) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
3 ment. Let V be the resulting value.
- 4 1) If V is an instance of the class **Array** which contains two elements, let F and S
5 be the first and the second element of V respectively.
- 6 i) Invoke the method **%** on F with S as the only argument.
- 7 ii) Return the resulting value.
- 8 2) Otherwise, raise a direct instance of the class **TypeError**.

9 **15.2.8.2.6 Integer#<=>**

10 <=>(*other*)

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the *other* is an instance of the class **Integer**:
- 14 1) If the value of the receiver is larger than the value of the *other*, return an instance
15 of the class **Integer** whose value is 1.
- 16 2) If the values of the receiver and the *other* are the same integer, return an instance
17 of the class **Integer** whose value is 0.
- 18 3) If the value of the receiver is smaller than the value of the *other*, return an instance
19 of the class **Integer** whose value is -1.
- 20 b) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
21 ment. Let V be the resulting value.
- 22 1) If V is an instance of the class **Array** which contains two elements, let F and S
23 be the first and the second element of V respectively.
- 24 i) Invoke the method **<=>** on F with S as the only argument.
- 25 ii) If this invocation does not result in an instance of the class **Integer**, the
26 behavior is implementation dependent.
- 27 iii) Otherwise, return the value of this invocation.
- 28 2) Otherwise, return **nil**.

29 **15.2.8.2.7 Integer#==**

1 `==(other)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is an instance of the class **Integer**:

5 1) If the values of the receiver and the *other* are the same integer, return **true**.

6 2) Otherwise, return **false**.

7 b) Otherwise, invoke the method `==` on the *other* with the receiver as the argument.
8 Return the resulting value of this invocation.

9 **15.2.8.2.8 Integer#~**

10 `~`

11 **Visibility:** public

12 **Behavior:** The method returns an instance of the class **Integer** whose two's complement
13 representation is the one's complement of the two's complement representation of the re-
14 ceiver.

15 **15.2.8.2.9 Integer#&**

16 `&(other)`

17 **Visibility:** public

18 **Behavior:**

19 a) If the *other* is not an instance of the class **Integer**, the behavior is implementation
20 dependent.

21 b) Otherwise, return an instance of the class **Integer** whose two's complement represen-
22 tation is the bitwise AND of the two's complement representations of the receiver and
23 the *other*.

24 **15.2.8.2.10 Integer#|**

25 `|(other)`

26 **Visibility:** public

1 **Behavior:**

2 a) If the *other* is not an instance of the class **Integer**, the behavior is implementation

3 dependent.

4 b) Otherwise, return an instance of the class **Integer** whose two's complement repre-

5 sentation is the bitwise inclusive OR of the two's complement representations of the

6 receiver and the *other*.

7 **15.2.8.2.11 Integer#^**

8 $\wedge(\textit{other})$

9 **Visibility:** public

10 **Behavior:**

11 a) If the *other* is not an instance of the class **Integer**, the behavior is implementation

12 dependent.

13 b) Otherwise, return an instance of the class **Integer** whose two's complement repre-

14 sentation is the bitwise exclusive OR of the two's complement representations of the

15 receiver and the *other*.

16 **15.2.8.2.12 Integer#<<**

17 $\ll(\textit{other})$

18 **Visibility:** public

19 **Behavior:**

20 a) If the *other* is not an instance of the class **Integer**, the behavior is implementation

21 dependent.

22 b) Otherwise, let x and y be the values of the receiver and the *other*.

23 c) Return an instance of the class **Integer** whose value is the largest integer smaller than

24 or equal to $x \times 2^y$.

25 **15.2.8.2.13 Integer#>>**

26 $\gg(\textit{other})$

27 **Visibility:** public

28 **Behavior:**

- 1 a) If the *other* is not an instance of the class **Integer**, the behavior is implementation
2 dependent.
- 3 b) Otherwise, let x and y be the values of the receiver and the *other*.
- 4 c) Return an instance of the class **Integer** whose value is the largest integer smaller than
5 or equal to $x \times 2^{-y}$.

6 **15.2.8.2.14 Integer#ceil**

7 **ceil**

8 **Visibility:** public

9 **Behavior:** The method returns the receiver.

10 **15.2.8.2.15 Integer#downto**

11 **downto**(*num*, &*block*)

12 **Visibility:** public

13 **Behavior:**

- 14 a) If the *num* is not an instance of the class **Integer**, or the *block* is not given, the
15 behavior is implementation dependent.
- 16 b) Let i be the value of the receiver.
- 17 c) If i is smaller than the value of the *num*, return the receiver.
- 18 d) Call the *block* with an instance of the class **Integer** whose value is i .
- 19 e) Decrement i by 1 and continue processing from Step c.

20 **15.2.8.2.16 Integer#eql?**

21 **eql?**(*other*)

22 **Visibility:** public

23 **Behavior:**

- 24 a) If the *other* is not an instance of the class **Integer**, return **false**.
- 25 b) Otherwise, invoke the method **==** on the *other* with the receiver as the argument.
- 26 c) If this invocation results in a true value, return **true**. Otherwise, return **false**.

1 **15.2.8.2.17 Integer#floor**

2 floor

3 **Visibility:** public

4 **Behavior:** The method returns the receiver.

5 **15.2.8.2.18 Integer#hash**

6 hash

7 **Visibility:** public

8 **Behavior:** The method returns an implementation defined instance of the class **Integer**,
9 which satisfies the following condition:

- 10 a) Let I_1 and I_2 be instances of the class **Integer**.
- 11 b) Let H_1 and H_2 be the resulting values of invocations of the method **hash** on I_1 and I_2 ,
12 respectively.
- 13 c) The values of the H_1 and H_2 shall be the same integer, if and only if the values of I_1
14 and I_2 are the same integer.

15 **15.2.8.2.19 Integer#next**

16 next

17 **Visibility:** public

18 **Behavior:** The method returns an instance of the class **Integer**, whose value is the value
19 of the receiver plus 1.

20 **15.2.8.2.20 Integer#round**

21 round

22 **Visibility:** public

23 **Behavior:** The method returns the receiver.

24 **15.2.8.2.21 Integer#succ**

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

`succ`

Visibility: public

Behavior: Same as the method `next` (see §15.2.8.2.19).

15.2.8.2.22 Integer#times

`times(&block)`

Visibility: public

Behavior:

- a) If the *block* is not given, the behavior is implementation dependent.
- b) Let *i* be 0.
- c) If *i* is larger than or equal to the value of the receiver, return the receiver.
- d) Call the *block* with an instance of the class `Integer` whose value is *i*.
- e) Increment *i* by 1 and continue processing from Step c.

15.2.8.2.23 Integer#to_f

`to_f`

Visibility: public

Behavior: The method returns a direct instance of the class `Float` whose value is the value of the receiver as a floating point number.

15.2.8.2.24 Integer#to_i

`to_i`

Visibility: public

Behavior: The method returns the receiver.

15.2.8.2.25 Integer#truncate

1 **truncate**

2 **Visibility:** public

3 **Behavior:** The method returns the receiver.

4 **15.2.8.2.26 Integer#upto**

5 **upto**(*num*, &*block*)

6 **Visibility:** public

7 **Behavior:**

8 a) If the *num* is not an instance of the class **Integer**, or the *block* is not given, the
9 behavior is implementation dependent.

10 b) Let *i* be the value of the receiver.

11 c) If *i* is larger than the value of the *num*, return the receiver.

12 d) Call the *block* with an instance of the class **Integer** whose value is *i*.

13 e) Increment *i* by 1 and continue processing from Step c.

14 **15.2.9 Float**

15 Instances of the class **Float** represent floating point numbers. A conforming processor should
16 use the native binary floating point representation of the underlying platform.

17 When an arithmetic operation involving floating point numbers results in a value which cannot
18 be represented exactly as an instance of the class **Float**, how the result is rounded to fit in the
19 representation of an instance of the class **Float** is implementation defined.

20 If the underlying platform of a conforming processor supports IEC 60559:1989:

21 • The representation of an instance of the class **Float** should be the 64-bit double format as
22 specified in §3.2.2 of IEC 60559:1989.

23 • If an arithmetic operation involving floating point numbers results in NaN while invoking
24 a method of the class **Float**, the behavior of the method is implementation dependent.

25 Instances of the class **Float** shall not be created by the method **new** of the class **Float**. Therefore,
26 a conforming processor shall undefine the singleton method **new** of the class **Float**, as if by
27 invoking the method **undef_method** on the eigenclass of the class **Float** with a direct instance
28 of the class **Symbol** whose name is “new” as the argument (see §15.2.2.3.42).

29 **15.2.9.1 Direct superclass**

30 The class **Numeric**

1 15.2.9.2 Instance methods

2 15.2.9.2.1 Float#+

3 +(*other*)

4 **Visibility:** public

5 **Behavior:**

6 a) If the *other* is an instance of the class **Float**, return a direct instance of the class **Float**
7 whose value is the sum of the values of the receiver and the *other*.

8 b) If the *other* is an instance of the class **Integer**, let *R* be the value of the *other* as a
9 floating point number.

10 Return a direct instance of the class **Float** whose value is the sum of *R* and the value
11 of the receiver.

12 c) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
13 ment. Let *V* be the resulting value.

14 1) If *V* is an instance of the class **Array** which contains two elements, let *F* and *S*
15 be the first and the second element of *V* respectively.

16 i) Invoke the method **+** on *F* with *S* as the only argument.

17 ii) Return the resulting value.

18 2) Otherwise, raise a direct instance of the class **TypeError**.

19 15.2.9.2.2 Float#-

20 -(*other*)

21 **Visibility:** public

22 **Behavior:**

23 a) If the *other* is an instance of the class **Float**, return a direct instance of the class **Float**
24 whose value is the result of subtracting the value of the *other* from the value of the
25 receiver.

26 b) If the *other* is an instance of the class **Integer**, let *R* be the value of the *other* as a
27 floating point number.

28 Return a direct instance of the class **Float** whose value is the result of subtracting *R*
29 from the value of the receiver.

- 1 c) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
2 ment. Let V be the resulting value.
- 3 1) If V is an instance of the class **Array** which contains two elements, let F and S
4 be the first and the second element of V respectively.
- 5 i) Invoke the method **-** on F with S as the only argument.
- 6 ii) Return the resulting value.
- 7 2) Otherwise, raise a direct instance of the class **TypeError**.

8 15.2.9.2.3 **Float#***

9 *****(*other*)

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the *other* is an instance of the class **Float**, return a direct instance of the class **Float**
13 whose value is the result of multiplication of the values of the receiver and the *other*.
- 14 b) If the *other* is an instance of the class **Integer**, let R be the value of the *other* as a
15 floating point number.

16 Return a direct instance of the class **Float** whose value is the result of multiplication
17 of R and the value of the receiver.
- 18 c) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
19 ment. Let V be the resulting value.
- 20 1) If V is an instance of the class **Array** which contains two elements, let F and S
21 be the first and the second element of V respectively.
- 22 i) Invoke the method ***** on F with S as the only argument.
- 23 ii) Return the resulting value.
- 24 2) Otherwise, raise a direct instance of the class **TypeError**.

25 15.2.9.2.4 **Float#/**

26 **/**(*other*)

27 **Visibility:** public

28 **Behavior:**

- 1 a) If the *other* is an instance of the class `Float`, return a direct instance of the class `Float`
2 whose value is the value of the receiver divided by the value of the *other*.
- 3 b) If the *other* is an instance of the class `Integer`, let R be the value of the *other* as a
4 floating point number.
5 Return a direct instance of the class `Float` whose value is the value of the receiver
6 divided by R .
- 7 c) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
8 ment. Let V be the resulting value.
9 1) If V is an instance of the class `Array` which contains two elements, let F and S
10 be the first and the second element of V respectively.
11 i) Invoke the method `/` on F with S as the only argument.
12 ii) Return the resulting value.
13 2) Otherwise, raise a direct instance of the class `TypeError`.

14 15.2.9.2.5 `Float#%`

15 `%(other)`

16 **Visibility:** public

17 **Behavior:** In the following steps, binary operators `+`, `-`, and `*` represent floating point
18 arithmetic operations addition, subtraction, and multiplication which are used in the in-
19 stance methods `+`, `-`, and `*` of the class `Float`, respectively. The operator `*` has a higher
20 precedence than the operators `+` and `-`.

- 21 a) If the *other* is an instance of the class `Integer` or the class `Float`:
22 Let x be the value of the receiver.
23 1) If the *other* is an instance of the class `Float`, let y be the value of the *other*. If
24 the *other* is an instance of the class `Integer`, let y be the value of the *other* as a
25 floating point number.
26 i) Let t be the largest integer smaller than or equal to x divided by y .
27 ii) Let m be $x - t * y$.
28 iii) If $m * y < 0$, return a direct instance of the class `Float` whose value is $m +$
29 y .
30 iv) Otherwise, return a direct instance of the class `Float` whose value is m .
31 b) Otherwise, invoke the method `coerce` on the *other* with the receiver as the only argu-
32 ment. Let V be the resulting value.

- 1 1) If V is an instance of the class **Array** which contains two elements, let F and S
2 be the first and the second element of V respectively.
- 3 i) Invoke the method `%` on F with S as the only argument.
- 4 ii) Return the resulting value.
- 5 2) Otherwise, raise a direct instance of the class **TypeError**.

6 **15.2.9.2.6 Float#<=>**

7 `<=>(other)`

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the *other* is an instance of the class **Integer** or the class **Float**:
- 11 1) Let a be the value of the receiver. If the *other* is an instance of the class **Float**,
12 let b be the value of the *other*. Otherwise, let b be the value of the *other* as a
13 floating point number.
- 14 2) If a conforming processor supports IEC 60559:1989, and if a or b is NaN, then
15 return an implementation defined value.
- 16 3) If $a > b$, return an instance of the class **Integer** whose value is 1.
- 17 4) If $a = b$, return an instance of the class **Integer** whose value is 0.
- 18 5) If $a < b$, return an instance of the class **Integer** whose value is -1.
- 19 b) Otherwise, invoke the method **coerce** on the *other* with the receiver as the only argu-
20 ment. Let V be the resulting value.
- 21 1) If V is an instance of the class **Array** which contains two elements, let F and S
22 be the first and the second element of V respectively.
- 23 i) Invoke the method `<=>` on F with S as the only argument.
- 24 ii) If this invocation does not result in an instance of the class **Integer**, the
25 behavior is implementation dependent.
- 26 iii) Otherwise, return the value of this invocation.
- 27 2) Otherwise, return **nil**.

28 **15.2.9.2.7 Float#==**

1 `==(other)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is an instance of the class **Float**:

5 1) If a conforming processor supports IEC 60559:1989, and if the value of the receiver

6 is NaN, then return **false**.

7 2) If the values of the receiver and the *other* are the same number, return **true**.

8 3) Otherwise, return **false**.

9 b) If the *other* is an instance of the class **Integer**:

10 1) If the values of the receiver and the *other* are the mathematically the same, return

11 **true**.

12 2) Otherwise, return **false**.

13 c) Otherwise, invoke the method `==` on the *other* with the receiver as the argument and

14 return the resulting value of this invocation.

15 **15.2.9.2.8 Float#ceil**

16 `ceil`

17 **Visibility:** public

18 **Behavior:** The method returns an instance of the class **Integer** whose value is the smallest

19 integer larger than or equal to the value of the receiver.

20 **15.2.9.2.9 Float#finite?**

21 `finite?`

22 **Visibility:** public

23 **Behavior:**

24 a) If the value of the receiver is a finite number, return **true**.

25 b) Otherwise, return **false**.

26 **15.2.9.2.10 Float#floor**

1 **floor**

2 **Visibility:** public

3 **Behavior:** The method returns an instance of the class **Integer** whose value is the largest
4 integer smaller than or equal to the value of the receiver.

5 **15.2.9.2.11 Float#infinite?**

6 **infinite?**

7 **Visibility:** public

8 **Behavior:**

9 a) If the value of the receiver is the positive infinite, return an instance of the class **Integer**
10 whose value is 1.

11 b) If the value of the receiver is the negative infinite, return an instance of the class
12 **Integer** whose value is -1.

13 c) Otherwise, return **nil**.

14 **15.2.9.2.12 Float#round**

15 **round**

16 **Visibility:** public

17 **Behavior:** The method returns an instance of the class **Integer** whose value is the nearest
18 integer to the value of the receiver. If there are two integers equally distant from the value
19 of the receiver, the one which has the larger absolute value is chosen.

20 **15.2.9.2.13 Float#to_f**

21 **to_f**

22 **Visibility:** public

23 **Behavior:** The method returns the receiver.

24 **15.2.9.2.14 Float#to_i**

1 `to_i`

2 **Visibility:** public

3 **Behavior:** The method returns an instance of the class `Integer` whose value is the integer
4 part of the receiver.

5 **15.2.9.2.15 Float#truncate**

6 `truncate`

7 **Visibility:** public

8 **Behavior:** Same as the method `to_i` (see §15.2.9.2.14).

9 **15.2.10 String**

10 Instances of the class `String` represent sequences of characters.

11 An instance of the class `String` which does not contain any character is said to be *empty*. An
12 instance of the class `String` shall be empty when it is created by Step b of the method `new` of
13 the class `Class`.

14 The notation “an instance of the class `Object` which represents the character *C*” means either
15 of the following:

- 16 • An instance of the class `Integer` whose value is the character code of *C*.
- 17 • An instance of the class `String` whose content is the single character *C*.

18 A conforming processor shall choose one of the above representations and use the same repre-
19 sentation wherever this notation is used.

20 The notation “the *n*th character of a string” means the character whose index is *n* counted up
21 from 0.

22 **15.2.10.1 Direct superclass**

23 The class `Object`

24 **15.2.10.2 Included modules**

25 The following modules are included in the class `String`.

- 26 • `Comparable`

27 **15.2.10.3 Instance methods**

28 **15.2.10.3.1 String#***

1 *****(*num*)

2 **Visibility:** public

3 **Behavior:**

4 a) If the *num* is not an instance of the class **Integer**, the behavior is implementation
5 dependent.

6 b) Let *n* be the value of the *num*.

7 c) If *n* is smaller than 0, raise a direct instance of the class **ArgumentError**.

8 d) Otherwise, let *C* be the content of the receiver.

9 e) Create a direct instance of the class **String** *S* the content of which is *C* repeated *n*
10 times.

11 f) Return *S*.

12 **15.2.10.3.2 String#+**

13 **+**(*other*)

14 **Visibility:** public

15 **Behavior:**

16 a) If the *other* is not an instance of the class **String**, the behavior is implementation
17 dependent.

18 b) Let *S* and *O* be the contents of the receiver and the *other* respectively.

19 c) Return a newly created instance of the class **String** the content of which is the con-
20 catenation of *S* and *O*.

21 **15.2.10.3.3 String#<=>**

22 **<=>**(*other*)

23 **Visibility:** public

24 **Behavior:**

25 a) If the *other* is not an instance of the class **String**, the behavior is implementation
26 dependent.

- 1 b) Let $S1$ and $S2$ be the contents of the receiver and the *other* respectively.
- 2 c) If both $S1$ and $S2$ are empty, return an instance of the class **Integer** whose value is 0.
- 3 d) If $S1$ is empty, return an instance of the class **Integer** whose value is -1.
- 4 e) If $S2$ is empty, return an instance of the class **Integer** whose value is 1.
- 5 f) Let a , b be the character codes of the first characters of $S1$ and $S2$ respectively.
 - 6 1) If $a > b$, return an instance of the class **Integer** whose value is 1.
 - 7 2) If $a < b$, return an instance of the class **Integer** whose value is -1.
 - 8 3) Otherwise, replace $S1$ and $S2$ with $S1$ and $S2$ excluding their first characters,
 - 9 respectively. Continue processing from Step c.

10 15.2.10.3.4 String#==

11 ==(*other*)

12 **Visibility:** public

13 **Behavior:**

- 14 a) If the *other* is not an instance of the class **String**, the behavior is implementation
 - 15 dependent.
- 16 b) If the *other* is an instance of the class **String**:
 - 17 1) If the content of the receiver and the *other* is the same, return **true**.
 - 18 2) Otherwise, return **false**.

19 15.2.10.3.5 String# =~

20 =~ (*regexp*)

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the *regexp* is not an instance of the class **Regexp**, the behavior is implementation
 - 24 dependent.
- 25 b) Otherwise, behave as if the method **match** is invoked on the *regexp* with the receiver
 - 26 as the argument (see §15.2.15.6.7).

1 **15.2.10.3.6 String#[]**

2 [](*args)

3 **Visibility:** public

4 **Behavior:**

5 a) If the length of the *args* is 0 or larger than 2, raise a direct instance of the class
6 **ArgumentError**.

7 b) Let *P* be the first element of the *args*. Let *n* be the length of the receiver.

8 c) If *P* is an instance of the class **Integer**, let *b* be the value of *P*.

9 1) If the length of the *args* is 1:

10 i) If *b* is smaller than 0, increment *b* by *n*. If *b* is still smaller than 0, return
11 **nil**.

12 ii) If $b \geq n$, return **nil**.

13 iii) Create an instance of the class **Object** which represents the *b*th character of
14 the receiver and return this instance.

15 2) If the length of the *args* is 2:

16 i) If the last element of the *args* is an instance of the class **Integer**, let *l* be the
17 value of the instance. Otherwise, the behavior is implementation dependent.

18 ii) If *l* is smaller than 0, or *b* is larger than *n*, return **nil**.

19 iii) If *b* is smaller than 0, increment *b* by *n*. If *b* is still smaller than 0, return
20 **nil**.

21 iv) If $b + l$ is larger than *n*, let *l* be $n - b$.

22 v) If *l* is smaller than or equal to 0, create an empty direct instance of the class
23 **String** and return the instance.

24 vi) Otherwise, create a direct instance of the class **String** whose content is the
25 $(n-l)$ characters of the receiver, from the *b*th index, preserving their order.
26 Return the instance.

27 d) If *P* is an instance of the class **Regexp**:

28 1) If the length of the *args* is 1, let *i* be 0.

29 2) If the length of the *args* is 2, and the last element of *args* is an instance of the
30 class **Integer**, let *i* be the value of the instance. Otherwise, the behavior is
31 implementation dependent.

- 1 3) Match the pattern of P against the content of the receiver. (see §15.2.15.3 and
2 Step 15.2.15.4). Let M be the result of the matching process.
- 3 4) If M is `nil`, return `nil`.
- 4 5) If i is larger than the length of the match result of M , return `nil`.
- 5 6) If i is smaller than 0, increment i by the length of the match result of M . If i is
6 still smaller than or equal to 0, return `nil`.
- 7 7) Let m be the i th element of the match result of M . Create a direct instance of the
8 class `String` whose content is the first element of m and return the instance.
- 9 e) If P is an instance of the class `String`:
- 10 1) If the length of the *args* is 2, the behavior is implementation dependent.
- 11 2) If the receiver includes the content of P as a substring, create a direct instance of
12 the class `String` whose content is equal to the content P and return the instance.
- 13 3) Otherwise, return `nil`.
- 14 f) Otherwise, the behavior is implementation dependent.

15 15.2.10.3.7 String#capitalize

16 capitalize

17 **Visibility:** public

18 **Behavior:** The method returns a newly created instance of the class `String` which contains
19 all the characters of the receiver, except:

- 20 • If the first character of the receiver is a lower-case character, the first character of the
21 resulting instance is the corresponding upper-case character.
- 22 • If the i th character of the receiver (where $i > 0$) is an upper case character, the i th
23 character of the resulting instance is the corresponding lower-case character.

24 15.2.10.3.8 String#capitalize!

25 capitalize!

26 **Visibility:** public

27 **Behavior:**

- 28 a) Let s be the content of the instance of the class `String` returned when the method
29 **capitalize** is invoked on the receiver.

- 1 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
2 content of the receiver to *s*, and return the receiver.

3 15.2.10.3.9 String#chomp

4 chomp(*rs* = "\n")

5 **Visibility:** public

6 **Behavior:**

- 7 a) If the *rs* is **nil**, return a newly created instance of the class **String** whose content is
8 the same as the receiver.
- 9 b) If the receiver is empty, return a newly created empty instance of the class **String**.
- 10 c) If *rs* is not an instance of the class **String**, the behavior is implementation dependent.
- 11 d) Otherwise, return a newly created instance of the class **String** whose content is the
12 same as the receiver, except the following characters:
- 13 1) If the *rs* consists of only one character 0x0a, the *line-terminator* on the end, if
14 any, is excluded.
- 15 2) If the *rs* is empty, the sequence of *line-terminators* on the end, if any, is excluded.
- 16 3) Otherwise, if the receiver ends with the content of *rs*, this sequence of the charac-
17 ters at the end of the receiver is excluded.

18 15.2.10.3.10 String#chomp!

19 chomp!(*rs* = "\n")

20 **Visibility:** public

21 **Behavior:**

- 22 a) Let *s* be the content of the instance of the class **String** returned when the method
23 **chomp** is invoked on the receiver with the *rs* as the argument.
- 24 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
25 content of the receiver to *s*, and return the receiver.

26 15.2.10.3.11 String#chop

1

2

3

4

5
6
7

8

9

10

11

12
13

14
15

16

17

18

19
20
21

22

23

24

25

26
27

chop

Visibility: public

Behavior:

- a) If the receiver is empty, return a newly created empty instance of the class **String**.
- b) Otherwise, return a newly created instance of the class **String** whose content is the receiver without the last character. If the last character is 0x0a, and the character just before the 0x0a is 0x0d, the 0x0d is also dropped.

15.2.10.3.12 String#chop!

chop!

Visibility: public

Behavior:

- a) Let *s* be the content of the instance of the class **String** returned when the method **chop** is invoked on the receiver.
- b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the content of the receiver to *s*, and return the receiver.

15.2.10.3.13 String#downcase

downcase

Visibility: public

Behavior: The method returns a newly created instance of the class **String** which contains all the characters of the receiver, with the upper-case characters replaced with the corresponding lower-case characters.

15.2.10.3.14 String#downcase!

downcase!

Visibility: public

Behavior:

- a) Let *s* be the content of the instance of the class **String** returned when the method **downcase** is invoked on the receiver.

- 1 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
2 content of the receiver to *s*, and return the receiver.

3 **15.2.10.3.15 String#each_line**

4 **each_line**(&*block*)

5 **Visibility:** public

6 **Behavior:** Let *s* be the content of the receiver. Let *c* be the first character of *s*.

- 7 a) If the *block* is not given, the behavior is implementation dependent.
- 8 b) Find the first 0x0a in *s* from *c*. If there is such a 0x0a:
- 9 1) Let *d* be that 0x0a.
- 10 2) Create a direct instance of the class **String** *S* whose content is the sequence of
11 the characters from *c* to *d*.
- 12 3) Call the *block* with *S* as the argument.
- 13 4) If *d* is the last character of *s*, return the receiver. Otherwise, let new *c* be the
14 character just after *d* and continue processing from Step b.
- 15 c) If there is not such a 0x0a, create a direct instance of the class **String** whose content
16 is the sequence of the characters from *c* to the last character of *s*. Call the *block* with
17 this instance as the argument.
- 18 d) Return the receiver.

19 **15.2.10.3.16 String#empty?**

20 **empty?**

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the receiver is empty, return **true**.
- 24 b) Otherwise, return **false**.

25 **15.2.10.3.17 String#eq?**

1 `eql?(other)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the *other* is an instance of the class **String**:

5 1) If the contents of the receiver and the *other* are the same, return **true**.

6 2) Otherwise, return **false**.

7 b) If the *other* is not an instance of the class **String**, return **false**.

8 **15.2.10.3.18 String#gsub**

9 `gsub(*args, &block)`

10 **Visibility:** public

11 **Behavior:**

12 a) If the length of the *args* is 0 or larger than 2, or the length of the *args* is 1 and the
13 *block* is not given, raise a direct instance of the class **ArgumentError**.

14 b) Let *P* be the first element of the *args*. If *P* is not an instance of the class **Regexp**, or
15 the length of the *args* is 2 and the last element of the *args* is not an instance of the
16 class **String**, the behavior is implementation dependent.

17 c) Let *S* be the content of the receiver, and let *l* be the length of *S*.

18 d) Let *L* be an empty list and let *n* be an integer 0.

19 e) Match the pattern of *P* against *S* at the offset *n* (see §15.2.15.3 and Step 15.2.15.4).
20 Let *M* be the result of the matching process.

21 f) If *M* is **nil**, append to *L* the substring of *S* beginning at the *n*th character up to the
22 last character of *S*.

23 g) Otherwise:

24 1) If the length of the *args* is 1:

25 i) Call the *block* with a direct instance of the class **String** whose content is the
26 matched substring of *M* as the argument.

27 ii) Let *V* be the resulting value of this call. If *V* is not an instance of the class
28 **String**, the behavior is implementation dependent.

- 1 2) Let *pre* be the pre-match of *M*. Append to *L* the substring of *pre* beginning at the
- 2 *n*th character up to the last character of *pre*, unless *n* is larger than the offset of
- 3 the last character of *pre*.
- 4 3) If the length of the *args* is 1, append the content of *V* to *L*. If the length of the
- 5 *args* is 2, append to *L* the content of the last element of the *args*.
- 6 4) Let *post* be the post-match of *M*. Let *i* be the offset of the first character of *post*
- 7 within *S*.
- 8 i) If *i* is equal to *n*, i.e. if *P* matched an empty string:
- 9 I) Append to *L* a string whose content is the *i*th character of *S*.
- 10 II) Increment *n* by 1.
- 11 ii) Otherwise, replace *n* with *i*.
- 12 5) If *n* < *l*, continue processing from Step e.
- 13 h) Create a direct instance of the class **String** whose content is the concatenation of all
- 14 the elements of *L*, and return the instance.

15 15.2.10.3.19 String#gsub!

16 gsub!(*args, &block)

17 **Visibility:** public

18 **Behavior:**

- 19 a) Let *s* be the content of the instance of the class **String** returned when the method
- 20 gsub is invoked on the receiver with the same arguments.
- 21 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
- 22 content of the receiver to *s*, and return the receiver.

23 15.2.10.3.20 String#hash

24 hash

25 **Visibility:** public

26 **Behavior:** The method returns an implementation defined instance of the class **Integer**

27 which satisfies the following condition:

- 28 a) Let *S*₁ and *S*₂ be two distinct instances of the class **String**.

- 1 b) Let H_1 and H_2 be the resulting values of the invocations of the method **hash** on S_1 and
2 S_2 respectively.
- 3 c) If and only if S_1 and S_2 has the same content, the values of H_1 and H_2 shall be the
4 same integer.

5 **15.2.10.3.21 String#include?**

6 **include?**(*obj*)

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the *obj* is an instance of the class **Integer**:

10 If the receiver includes the character whose character code is the *obj*, return **true**.
11 Otherwise, return **false**.

- 12 b) If the *obj* is an instance of the class **String**:

13 If there exists a substring of the receiver whose sequence of characters is the same as
14 the *obj*, return **true**. Otherwise, return **false**.

- 15 c) Otherwise, the behavior is implementation dependent.

16 **15.2.10.3.22 String#initialize**

17 **initialize**(*str*="")

18 **Visibility:** private

19 **Behavior:**

- 20 a) If the *str* is not an instance of the class **String**, the behavior is implementation de-
21 pendent.

- 22 b) Otherwise, initialize the content of the receiver to the same sequence of characters as
23 the content of the *str*.

- 24 c) Return an implementation defined value.

25 **15.2.10.3.23 String#initialize_copy**

26 **initialize_copy**(*original*)

27 **Visibility:** private

- 1 **Behavior:**
- 2 a) If the *original* is not an instance of the class **String**, the behavior is implementation
- 3 dependent.
- 4 b) Change the content of the receiver to the content of the *original*.
- 5 c) Return an implementation defined value.

6 **15.2.10.3.24 String#intern**

7 **intern**

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the length of the receiver is 0, or the receiver contains 0x00, raise a direct instance
- 11 of the class **ArgumentError**.
- 12 b) Otherwise, return a direct instance of the class **Symbol** whose name is the content of
- 13 the receiver.

14 **15.2.10.3.25 String#length**

15 **length**

16 **Visibility:** public

17 **Behavior:** The method returns the number of characters of the content of the receiver.

18 **15.2.10.3.26 String#match**

19 **match(*regexp*)**

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the *regexp* is an instance of the class **Regexp**, let *R* be the *regexp*.
- 23 b) If the *regexp* is an instance of the class **String**, create a direct instance of the class
- 24 **Regexp** as if the method **new** is invoked on the class **Regexp** with the *regexp* as the
- 25 argument. Let *R* be the instance of the class **Regexp**.
- 26 c) Otherwise, the behavior is implementation dependent.

- 1 d) Invoke the method `match` on *R* with the receiver as the argument.
- 2 e) Return the resulting value of the invocation.

3 15.2.10.3.27 String#replace

4 `replace(other)`

5 **Visibility:** public

6 **Behavior:** Same as the method `initialize_copy` (see §15.2.10.3.23).

7 15.2.10.3.28 String#reverse

8 `reverse`

9 **Visibility:** public

10 **Behavior:** The method returns a newly created instance of the class `String` which contains
11 all the characters of the content of the receiver in the reverse order.

12 15.2.10.3.29 String#reverse!

13 `reverse!`

14 **Visibility:** public

15 **Behavior:**

16 a) Change the content of the receiver to the content of the resulting instance of the class
17 `String` when the method `reverse` is invoked on the receiver.

18 b) Return the receiver.

19 15.2.10.3.30 String#scan

20 `scan(reg, &block)`

21 **Visibility:** public

22 **Behavior:**

23 a) If the *reg* is not an instance of the class `Regexp`, the behavior is implementation de-
24 pendent.

25 b) If the *block* is not given, create an empty direct instance of the class `Array` *A*.

- 1 c) Let S be the content of the receiver, and let l be the length of S .
- 2 d) Let n be an integer 0.
- 3 e) Match the pattern of the *reg* against S at the offset n (see §15.2.15.3 and Step 15.2.15.4).
- 4 Let M be the result of the matching process.
- 5 f) If M is not **nil**:
 - 6 1) Let L be the match result of M .
 - 7 2) If the length of L is 1, create a direct instance of the class **String** V whose content
 - 8 is the matched substring of M .
 - 9 3) If the length of L is larger than 1:
 - 10 i) Create an empty direct instance of the class **Array** V .
 - 11 ii) Except for the first element, for each element e of L , in the same order in the
 - 12 list, append to V a direct instance of the class **String** whose content is the
 - 13 first element of e .
 - 14 4) If the *block* is given, call the *block* with V as the argument. Otherwise, append V
 - 15 to A .
 - 16 5) Let *post* be the post-match of M . Let i be the offset of the first character of *post*
 - 17 within S .
 - 18 i) If i and n are the same, i.e. if the *reg* matched the empty string, increment
 - 19 n by 1.
 - 20 ii) Otherwise, replace n with i .
 - 21 6) If $n < l$, continue processing from Step e.
 - 22 g) If the *block* is given, return the receiver. Otherwise, return A .

23 15.2.10.3.31 String#size

24 **size**

25 **Visibility:** public

26 **Behavior:** Same as the method **length** (see §15.2.10.3.25).

27 15.2.10.3.32 String#slice

1 `slice(*args)`

2 **Visibility:** public

3 **Behavior:** Same as the method [] (see §15.2.10.3.6).

4 **15.2.10.3.33 String#split**

5 `split(sep)`

6 **Visibility:** public

7 **Behavior:**

- 8 a) If the *sep* is not an instance of the class **Regexp**, the behavior is implementation de-
9 pendent.
- 10 b) Create an empty direct instance of the class **Array** *A*.
- 11 c) Let *S* be the content of the receiver, and let *l* be the length of *S*.
- 12 d) Let both *sp* and *bp* be 0, and let *was-empty* be false.
- 13 e) Match the pattern of the *sep* against *S* at the offset *sp* (see §15.2.15.3 and Step
14 15.2.15.4). Let *M* be the result of the matching process.
- 15 f) If *M* is **nil**, append to *A* a direct instance of the class **String** whose content is the
16 substring of *S* beginning at the *sp*th character up to the last character of *S*.
- 17 g) Otherwise:
- 18 1) If the matched substring of *M* is an empty string:
- 19 i) If *was-empty* is true, append to *A* a direct instance of the class **String** whose
20 content is the *bp*th character of *S*.
- 21 ii) Otherwise, increment *sp* by 1. If *sp* < *l*, replace *was-empty* with true and
22 continue processing from Step e.
- 23 2) Otherwise, replace *was-empty* with false. Let *pre* be the pre-match of *M*. Append
24 to *A* an instance of the class **String** whose content is the substring of *pre* beginning
25 at the *bp*th character up to the last character of *pre*, unless *bp* is larger than the
26 offset of the last character of *pre*.
- 27 3) Let *L* be the match result of *M*.
- 28 4) If the length of *L* is larger than 1, except for the first element, for each element *e*
29 of *L*, in the same order in the list, take the following steps:

- 1 i) Let c be the first element of e .
- 2 ii) If c is not `nil`, append to A a direct instance of the class `String` whose
- 3 content is c .
- 4 5) Let $post$ be the post-match of M , and replace both sp and bp with the offset of
- 5 the first character of $post$.
- 6 6) If $sp > l$, continue processing from Step e.
- 7 h) If the last element of A is an instance of the class `String` whose content is empty,
- 8 remove the element. Repeat this step until the this condition does not hold.
- 9 i) return A .

10 **15.2.10.3.34 String#sub**

11 `sub(*args, &block)`

12 **Visibility:** public

13 **Behavior:**

- 14 a) If the length of the $args$ is 1 and the $block$ is given, or the length of the $args$ is 2:
 - 15 1) If the first element of the $args$ is not an instance of the class `Regexp`, the behavior
 - 16 is implementation dependent.
 - 17 2) Match the pattern of the first element of the $args$ against the content of the receiver
 - 18 (see §15.2.15.3 and Step 15.2.15.4). Let M be the result of the matching process.
 - 19 3) If M is `nil`, create a direct instance of the class `String` whose content is the same
 - 20 as the receiver and return the instance.
 - 21 4) Otherwise:
 - 22 i) If the length of the $args$ is 1, call the $block$ with a direct instance of the class
 - 23 `String` whose content is the matched substring of M as the argument. Let S
 - 24 be the resulting value of this call. If S is not an instance of the class `String`,
 - 25 the behavior is implementation dependent.
 - 26 ii) If the length of the $args$ is 2, let S be the last element of the $args$. If S is not
 - 27 an instance of the class `String`, the behavior is implementation dependent.
 - 28 iii) Create a direct instance of the class `String` whose content is the concatenation
 - 29 of pre-match of M , the content of S , and post-match of M , and return the
 - 30 instance.
- 31 b) Otherwise, raise a direct instance of the class `ArgumentError`.

1 15.2.10.3.35 String#sub!

2 sub! (*args, &block)

3 **Visibility:** public

4 **Behavior:**

5 a) Let *s* be the content of the instance of the class **String** returned when the method **sub**
6 is invoked on the receiver with the same arguments.

7 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
8 content of the receiver to *s*, and return the receiver.

9 15.2.10.3.36 String#upcase

10 upcase

11 **Visibility:** public

12 **Behavior:** The method returns a newly created instance of the class **String** which con-
13 tains all the characters of the receiver, with all the lower-case characters replaced with the
14 corresponding upper-case characters.

15 15.2.10.3.37 String#upcase!

16 upcase!

17 **Visibility:** public

18 **Behavior:**

19 a) Let *s* be the content of the instance of the class **String** returned when the method
20 **upcase** is invoked on the receiver.

21 b) If the content of the receiver and *s* are the same, return **nil**. Otherwise, change the
22 content of the receiver to *s*, and return the receiver.

23 15.2.10.3.38 String#to_i

24 to_i (base=10)

25 **Visibility:** public

26 **Behavior:**

- 1 a) If the *base* is not an instance of the class **Integer** whose value is 2, 8, 10, nor 16, the
2 behavior is implementation dependent. Otherwise, let *b* be the value of the *base*.
- 3 b) If the receiver is empty, return an instance of the class **Integer** whose value is 0.
- 4 c) Let *i* be 0. Increment *i* by 1 while the *i*th character of the receiver is a *whitespace*.
- 5 d) If the *i*th character of the receiver is “+” or “-”, increment *i* by 1.
- 6 e) If the *i*th character of the receiver is “0”, and any of the following conditions holds,
7 increment *i* by 2:

8 Let *c* be the character of the receiver whose index is *i* plus 1.

9 • *b* is 2, and *c* is “b” or “B”.
10 • *b* is 8, and *c* is “o” or “O”.
11 • *b* is 10, and *c* is “d” or “D”.
12 • *b* is 16, and *c* is “x” or “X”.
- 13 f) Let *s* be a sequence of the following characters of the receiver from the *i*th index:

14 • If *b* is 2, *binary-digit* and “_”.
15 • If *b* is 8, *octal-digit* and “_”.
16 • If *b* is 10, *decimal-digit* and “_”.
17 • If *b* is 16, *hexadecimal-digit* and “_”.
- 18 g) If the length of *s* is 0, return an instance of the class **Integer** whose value is 0.
- 19 h) If *s* starts with “_”, or *s* contains successive “_”s, the behavior is implementation
20 dependent.
- 21 i) Let *n* be the value of *s*, computed in base *b*.

22 If the “-” occurs in Step d, return an instance of the class **Integer** whose value is $-n$.
23 Otherwise, return an instance of the class **Integer** whose value is *n*.

24 15.2.10.3.39 String#to_f

25 to_f

26 **Visibility:** public

27 **Behavior:**

- 28 a) If the receiver is empty, return a direct instance of the class **Float** whose value is 0.0.

- b) If the receiver starts with the sequence of the characters which is a *float-literal*, return a direct instance of the class `Float` whose value is the value of the *float-literal* (see §8.5.5.1).
- c) If the receiver starts with the sequence of the characters which is a *digit-decimal-integer-literal*, return a direct instance of the class `Float` whose value is the value of the *digit-decimal-integer-literal* as a floating point number (see §8.5.5.1).
- d) Otherwise, return a direct instance of the class `Float` whose value is implementation defined.

15.2.10.3.40 `String#to_s`

`to_s`

Visibility: public

Behavior: The method returns the receiver.

15.2.10.3.41 `String#to_sym`

`to_sym`

Visibility: public

Behavior: Same as the method `intern` (see §15.2.10.3.24).

15.2.11 `Symbol`

Instances of the class `Symbol` represent names (see §8.5.5.5). No two instances of the class `Symbol` have the same name.

Instances of the class `Symbol` shall not be created by the method `new` of the class `Symbol`. Therefore, a conforming processor shall undefine the singleton method `new` of the class `Symbol`, as if by invoking the method `undef_method` on the eigenclass of the class `Symbol` with a direct instance of the class `Symbol` whose name is “new” as the argument (see §15.2.2.3.42).

15.2.11.1 `Direct superclass`

The class `Object`

15.2.11.2 `Instance methods`

15.2.11.2.1 `Symbol#===`

`===(other)`

Visibility: public

1 **Behavior:** Same as the method `==` of the module `Kernel` (see §15.3.1.2.1).

2 15.2.11.2.2 Symbol#id2name

3 id2name

4 **Visibility:** public

5 **Behavior:** The method returns an instance of the class `String`, the content of which
6 represents the name of the receiver.

7 15.2.11.2.3 Symbol#to_s

8 to_s

9 **Visibility:** public

10 **Behavior:** Same as the method `id2name` (see §15.2.11.2.2).

11 15.2.11.2.4 Symbol#to_sym

12 to_sym

13 **Visibility:** public

14 **Behavior:** The method returns the receiver.

15 15.2.12 Array

16 Instances of the class `Array` represent arrays, which are unbounded. An instance of the class
17 `Array` which has no element is said to be *empty*. The number of elements in an instance of the
18 class `Array` is called its *length*.

19 Instances of the class `Array` shall be empty when they are created by Step b of the method `new`
20 of the class `Class`.

21 Elements of an instance of the class `Array` has their indexes counted up from 0.

22 Given an array *A*, operations *append*, *prepend*, *remove* are defined as follows:

23 **append:** To append an object *O* to *A* is defined as follows:

24 Insert *O* after the last element of *A*.

25 Appending an object to *A* increases its length by 1.

26 **prepend:** To prepend an object *O* to *A* is defined as follows:

1 Insert O to the first index of A . Original elements of A are moved toward the end of A by
2 one position.

3 Prepending an object to A increases its length by 1.

4 **remove:** To remove an element X from A is defined as follows:

5 a) Remove X from A .

6 b) If X is not the last element of A , move the elements after X toward the head of A by
7 one position.

8 Removing an object to A decreases its length by 1.

9 **15.2.12.1 Direct superclass**

10 The class `Object`

11 **15.2.12.2 Included modules**

12 The following module is included in the class `Array`.

13 • `Enumerable`

14 **15.2.12.3 Singleton methods**

15 **15.2.12.3.1 `Array.[]`**

16 `Array.[] (*items)`

17 **Visibility:** public

18 **Behavior:** The method returns a newly created instance of the class `Array` which contains
19 the elements of the *items*, preserving their order.

20 **15.2.12.4 Instance methods**

21 **15.2.12.4.1 `Array#*`**

22 `*(num)`

23 **Visibility:** public

24 **Behavior:**

25 a) If the *num* is not an instance of the class `Integer`, the behavior is implementation
26 dependent.

27 b) If the value of the *num* is smaller than 0, raise a direct instance of the class `ArgumentError`.

- 1 c) If the value of the *num* is 0, return an empty direct instance of the class **Array**.
- 2 d) Otherwise, create a direct instance of the class **Array** *A* and repeat the following for
- 3 the *num* times:
- 4 Append all the elements of the receiver to *A*, preserving their order.
- 5 e) Return *A*.

6 **15.2.12.4.2 Array#+**

7 **+(*other*)**

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the *other* is an instance of the class **Array**, let *A* be the *other*. Otherwise, the
- 11 behavior is implementation dependent.
- 12 b) Create an empty direct instance of the class **Array** *R*.
- 13 c) For each element of the receiver, in the indexing order, append the element to *R*. Then,
- 14 for each element of *A*, in the indexing order, append the element to *R*.
- 15 d) Return *R*.

16 **15.2.12.4.3 Array#<<**

17 **<<(*obj*)**

18 **Visibility:** public

19 **Behavior:** The method appends the *obj* to the receiver and return the receiver.

20 **15.2.12.4.4 Array#[]**

21 **[] (**args*)**

22 **Visibility:** public

23 **Behavior:**

- 24 a) Let *n* be the length of the receiver.
- 25 b) If the length of the *args* is 0, raise a direct instance of the class **ArgumentError**.

- 1 c) If the length of the *args* is 1:
 - 2 1) If the only argument is an instance of the class **Integer**, let *k* be the value of the
 - 3 only argument. Otherwise, the behavior is implementation dependent.
 - 4 2) If $k < 0$, increment *k* by *n*. If *k* is still smaller than 0, return **nil**.
 - 5 3) If $k \geq n$, return **nil**.
 - 6 4) Otherwise, return the *k*th element of the receiver.
- 7 d) If the length of the *args* is 2:
 - 8 1) If the elements of the *args* are instances of the class **Integer**, let *b* and *l* be the
 - 9 values of the first and the last element of the *args*, respectively. Otherwise, the
 - 10 behavior is implementation dependent.
 - 11 2) If $b < 0$, increment *b* by *n*. If *b* is still smaller than 0, return **nil**.
 - 12 3) If $b = n$, create an empty direct instance of the class **Array** and return this instance.
 - 13 4) If $b > n$ or $l < 0$, return **nil**.
 - 14 5) If $l > n - b$, let new *l* be $n - b$.
 - 15 6) Create an empty direct instance of the class **Array** *A*. Append the *l* elements of
 - 16 the receiver to *A*, from the *b*th index, preserving their order. Return *A*.
- 17 e) If the length of the *args* is larger than 2, raise a direct instance of the class **ArgumentError**.

18 15.2.12.4.5 **Array#[]=**

19 [] = (*args)

20 **Visibility:** public

21 **Behavior:**

- 22 a) Let *n* be the length of the receiver.
- 23 b) If the length of the *args* is smaller than 2, raise a direct instance of the class **ArgumentError**.
- 24 c) If the length of the *args* is 2:
 - 25 1) If the first element of the *args* is an instance of the class **Integer**, let *k* be the
 - 26 value of the element and let *V* be the last element of the *args*. Otherwise, the
 - 27 behavior is implementation dependent.
 - 28 2) If $k < 0$, increment *k* by *n*. If *k* is still smaller than 0, raise a direct instance of
 - 29 the class **IndexError**.

- 1 3) If $k < n$, replace the k th element of the receiver with V .
- 2 4) Otherwise, expand the length of the receiver to $k + 1$. The last element of the
- 3 receiver is V . If $k > n$, the elements whose index is from n to $k - 1$ is `nil`.
- 4 5) Return V .
- 5 d) If the length of the *args* is 3, the behavior is implementation dependent.
- 6 e) If the length of the *args* is larger than 3, raise a direct instance of the class `ArgumentError`.

7 **15.2.12.4.6 Array#clear**

8 `clear`

9 **Visibility:** public

10 **Behavior:** The method removes all the elements from the receiver and return the receiver.

11 **15.2.12.4.7 Array#collect!**

12 `collect!(&block)`

13 **Visibility:** public

14 **Behavior:**

- 15 a) If the *block* is given:
 - 16 1) For each element of the receiver in the indexing order, call the *block* with the
 - 17 element as the only argument and replace the element with the resulting value.
 - 18 2) Return the receiver.
- 19 b) If the *block* is not given, the behavior is implementation dependent.

20 **15.2.12.4.8 Array#concat**

21 `concat(other)`

22 **Visibility:** public

23 **Behavior:**

- 24 a) If the *other* is not an instance of the class `Array`, the behavior is implementation
- 25 dependent.

- 1 b) Otherwise, append all the elements of the *other* to the receiver, preserving their order.
- 2 c) Return the receiver.

3 15.2.12.4.9 **Array#each**

4 `each(&block)`

5 **Visibility:** public

6 **Behavior:**

- 7 a) If the *block* is given:

- 8 1) For each element of the receiver in the indexing order, call the *block* with the
- 9 element as the only argument.

- 10 2) Return the receiver.

- 11 b) If the *block* is not given, the behavior is implementation dependent.

12 15.2.12.4.10 **Array#each_index**

13 `each_index(&block)`

14 **Visibility:** public

15 **Behavior:**

- 16 a) If the *block* is given:

- 17 1) For each element of the receiver in the indexing order, call the *block* with an
- 18 argument, which is an instance of the class **Integer** whose value is the index of
- 19 the element.

- 20 2) Return the receiver.

- 21 b) If the *block* is not given, the behavior is implementation dependent.

22 15.2.12.4.11 **Array#empty?**

23 `empty?`

24 **Visibility:** public

25 **Behavior:**

- 1 a) If the receiver is empty, return **true**.
- 2 b) Otherwise, return **false**.

3 **15.2.12.4.12 Array#first**

4 **first**(**args*)

5 **Visibility:** public

6 **Behavior:**

- 7 a) If the length of the *args* is 0:
 - 8 1) If the receiver is empty, return **nil**.
 - 9 2) Otherwise, return the first element of the receiver.
- 10 b) If the length of the *args* is 1:
 - 11 1) If the only argument is not an instance of the class **Integer**, the behavior is
 - 12 implementation dependent. Otherwise, let *n* be the value of the only argument.
 - 13 2) If *n* is smaller than 0, raise a direct instance of the class **ArgumentError**.
 - 14 3) Otherwise, let *N* be the smaller of *n* and the length of the receiver.
 - 15 4) Return a newly created instance of the class **Array** which contains the first *N*
 - 16 elements of the receiver, preserving their order.
- 17 c) If the length of *args* is larger than 1, raise a direct instance of the class **ArgumentError**.

18 **15.2.12.4.13 Array#initialize**

19 **initialize**(*size=0, obj=nil, &block*)

20 **Visibility:** private

21 **Behavior:**

- 22 a) If the *size* is not an instance of the class **Integer**, the behavior is implementation
- 23 dependent. Otherwise, let *n* be the value of the *size*.
- 24 b) If *n* is smaller than 0, raise a direct instance of the class **ArgumentError**.
- 25 c) If *n* is 0, return an implementation defined value.
- 26 d) If *n* is larger than 0:

- 1) If the *block* is given:
 - i) Let k be 0.
 - ii) Call the block with an argument, which is an instance of the class **Integer** whose value is k . Append the resulting value of this call to the receiver.
 - iii) Increase k by 1. If k is equal to n , terminate this process. Otherwise, repeat from Step d-1-ii.
- 2) Otherwise, append the *obj* to the receiver n times.
- 3) Return an implementation defined value.

15.2.12.4.14 **Array#initialize_copy**

```
initialize_copy(original)
```

Visibility: private

Behavior:

- a) If the *original* is not an instance of the class **Array**, the behavior is implementation dependent.
- b) Remove all the elements from the receiver.
- c) Append all the elements of the *original* to the receiver, preserving their order.
- d) Return an implementation defined value.

15.2.12.4.15 **Array#join**

```
join(sep=nil)
```

Visibility: public

Behavior:

- a) If the *sep* is neither **nil** nor an instance of the class **String**, the behavior is implementation dependent.
- b) Let S be an empty direct instance of the class **String**.
- c) For each element X of the receiver, in the indexing order:
 - 1) If the *sep* is not **nil**, and X is not the first element of the receiver, append the content of the *sep* to S .

- 1 2) If X is an instance of the class **String**, append the content of X to S .
- 2 3) If X is an instance of the class **Array**:
- 3 i) If X is the receiver, i.e. if the receiver contains itself, append an implementa-
4 tion defined sequence of characters to S .
- 5 ii) Otherwise, append the content of the instance of the class **String** returned as
6 if by the invocation of the method **join** on X with the *sep* as the argument.
- 7 4) Otherwise, the behavior is implementation dependent.
- 8 d) Return S .

9 **15.2.12.4.16 Array#last**

10 **last**(**args*)

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the length of the *args* is 0:
- 14 1) If the receiver is empty, return **nil**.
- 15 2) Otherwise, return the last element of the receiver.
- 16 b) If the length of the *args* is 1:
- 17 1) If the only argument is not an instance of the class **Integer**, the behavior is
18 implementation dependent. Otherwise, let n be the value of the only argument.
- 19 2) If n is smaller than 0, raise a direct instance of the class **ArgumentError**.
- 20 3) Otherwise, let N be the smaller of n and the length of the receiver.

21 Return a newly created instance of the class **Array** which contains the last N
22 elements of the receiver, preserving their order.
- 23 c) If the length of *args* is larger than 1, raise a direct instance of the class **ArgumentError**.

24 **15.2.12.4.17 Array#length**

25 **length**

26 **Visibility:** public

27 **Behavior:** The method returns the number of elements of the receiver.

1 **15.2.12.4.18 Array#map!**

2 `map!(&block)`

3 **Visibility:** public

4 **Behavior:** Same as the method `collect!` (see §15.2.12.4.7).

5 **15.2.12.4.19 Array#pop**

6 `pop`

7 **Visibility:** public

8 **Behavior:**

9 a) If the receiver is empty, return `nil`.

10 b) Otherwise, remove the last element from the receiver and return that element.

11 **15.2.12.4.20 Array#push**

12 `push(*items)`

13 **Visibility:** public

14 **Behavior:**

15 a) For each element of the *items*, in the indexing order, append it to the receiver.

16 b) Return the receiver.

17 **15.2.12.4.21 Array#replace**

18 `replace(other)`

19 **Visibility:** public

20 **Behavior:** Same as the method `initialize_copy` (see §15.2.12.4.14).

21 **15.2.12.4.22 Array#reverse**

1	reverse
2	Visibility: public
3	Behavior: The method returns a newly created instance of the class Array which contains
4	all the elements of the receiver in the reverse order.
5	15.2.12.4.23 Array#reverse!
6	reverse!
7	Visibility: public
8	Behavior: The method reverses the order of the elements of the receiver and return the
9	receiver.
10	15.2.12.4.24 Array#shift
11	shift
12	Visibility: public
13	Behavior:
14	a) If the receiver is empty, return nil .
15	b) Otherwise, remove the first element from the receiver and return that element.
16	15.2.12.4.25 Array#size
17	size
18	Visibility: public
19	Behavior: Same as the method length (see §15.2.12.4.17).
20	15.2.12.4.26 Array#slice
21	slice(*args)
22	Visibility: public
23	Behavior: Same as the method [] (see §15.2.12.4.4).

1 **15.2.12.4.27 Array#unshift**

2 `unshift(*items)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) For each element of the *items*, in the reverse indexing order, prepend it to the receiver.
- 6 b) Return the receiver.

7 **15.2.13 Hash**

8 Instances of the class **Hash** represent hashes, which are sets of key/value pairs.

9 An instance of the class **Hash** which has no key/value pair is said to be **empty**. Instances of
10 the class **Hash** shall be empty when they are created by Step b of the method **new** of the class
11 **Class**.

12 A hash cannot contain more than one key/value pair for each key.

13 An instance of the class **Hash** has the following property:

14 **default value or proc:** Either of the followings:

- 15 • A default value, which is returned by the method `[]` when the specified key is not
16 found in the hash.
- 17 • A default proc, which is called to generate the return value of the method `[]` when the
18 specified key is not found in the hash.

19 An instance of the class **Hash** shall not have both a default value and a default proc simul-
20 taneously.

21 Given two keys K_1 and K_2 , the notation “ $K_1 \equiv K_2$ ” means that all of the following conditions
22 hold:

- 23 • An invocation of the method `eq1?` on K_1 with K_2 as the only argument evaluates to a true
24 value.
- 25 • Let H_1 and H_2 be the results of invocations of the method `hash` on K_1 and K_2 , respectively.

26 H_1 and H_2 are the instances of the class **Integer** which represents the same integer.

27 A conforming processor may define a certain range of integers, and when the values of H_1 or
28 H_2 lies outside of this range, an implementation shall convert H_1 or H_2 to another instance
29 of the class **Integer** whose value is within the range. Let I_1 and I_2 be each of the resulting
30 instances respectively.

31 The values of I_1 and I_2 are the same integer.

1 If H_1 or H_2 is not an instance of the class `Integer`, whether K_1 and K_2 are considered to
2 be the same is implementation dependent.

3 Note that $K_1 \equiv K_2$ is not equivalent to $K_2 \equiv K_1$.

4 **15.2.13.1 Direct superclass**

5 The class `Object`

6 **15.2.13.2 Included modules**

7 The following module is included in the class `Hash`.

- 8 • `Enumerable`

9 **15.2.13.3 Instance methods**

10 **15.2.13.3.1 Hash#==**

11 `==(other)`

12 **Visibility:** public

13 **Behavior:**

14 a) If the *other* is not an instance of the class `Hash`, the behavior is implementation de-
15 pendent.

16 b) If all of the following conditions hold, return `true`:

- 17 • The receiver and the *other* have the same number of key/value pairs.
- 18 • For each key/value pair P in the receiver, the *other* has a corresponding key/value
19 pair Q which satisfies the following conditions:

20 — The key of $P \equiv$ the key of Q .

21 — An invocation of the method `==` on the value of P with the value of Q results
22 in a true value.

23 c) Otherwise, return `false`.

24 **15.2.13.3.2 Hash#[]**

25 `[] (key)`

26 **Visibility:** public

- 1 **Behavior:**
- 2 a) If the receiver has a key/value pair P where the $key \equiv$ the key of P , return the value
- 3 of P .
- 4 b) Otherwise, invoke the method `default` on the receiver with the key as the argument
- 5 and return the resulting value.

6 **15.2.13.3.3 Hash#[]=**

7 [] =(key , $value$)

8 **Visibility:** public

9 **Behavior:**

- 10 a) If the receiver has a key/value pair P where the $key \equiv$ the key of P , replace the value
- 11 of P with the $value$.
- 12 b) Otherwise:
- 13 1) If the key is a direct instance of the class `String`, create a copy of the key , i.e.
- 14 create a direct instance of the class `String` K whose content is the same as the
- 15 key .
- 16 2) If the key is not an instance of the class `String`, let K be the key .
- 17 3) If the key is an instance of a subclass of the class `String`, whether to create a copy
- 18 or not is implementation defined.
- 19 4) Store a pair of K and the $value$ into the receiver.
- 20 c) Return the $value$.

21 **15.2.13.3.4 Hash#clear**

22 clear

23 **Visibility:** public

24 **Behavior:**

- 25 a) Remove all the key/value pairs from the receiver.
- 26 b) Return the receiver.

27 **15.2.13.3.5 Hash#default**

```
1  default(*args)
```

2 **Visibility:** public

3 **Behavior:**

4 a) If the length of the *args* is larger than 1, raise a direct instance of the class **ArgumentError**.

5 b) If the receiver has the default value, return the value.

6 c) If the receiver has the default proc:

7 1) If the length of the *args* is 0, return **nil**.

8 2) If the length of the *args* is 1, invoke the method **call** on the default proc of the

9 receiver with two arguments, the receiver and the only element of the *args*. Return

10 the resulting value of this invocation.

11 d) Otherwise, return **nil**.

12 15.2.13.3.6 Hash#default=

```
13  default=(value)
```

14 **Visibility:** public

15 **Behavior:**

16 a) If the receiver has the default proc, remove the default proc.

17 b) Set the default value of the receiver to the *value*.

18 c) Return the *value*.

19 15.2.13.3.7 Hash#default_proc

```
20  default_proc
```

21 **Visibility:** public

22 **Behavior:**

23 a) If the receiver has the default proc, return the default proc.

24 b) Otherwise, return **nil**.

1 15.2.13.3.8 Hash#delete

2 delete(*key*, &*block*)

3 **Visibility:** public

4 **Behavior:**

5 a) If the receiver has a key/value pair *P* where the *key* \equiv the key of *P*, remove *P* from
6 the receiver and return the value of *P*.

7 b) Otherwise:

8 1) If the *block* is given, call the *block* with the *key* as the argument. Return the
9 resulting value of this call.

10 2) Otherwise, return `nil`.

11 15.2.13.3.9 Hash#each

12 each(&*block*)

13 **Visibility:** public

14 **Behavior:**

15 a) If the *block* is given, call the *block* for each key/value pair of the receiver with an
16 instance of the class **Array** as the argument, which contains two elements, the key
17 and the value of that pair. The order of key/value pairs are implementation defined.
18 Return the receiver.

19 b) If the *block* is not given, the behavior is implementation dependent.

20 15.2.13.3.10 Hash#each_key

21 each_key(&*block*)

22 **Visibility:** public

23 **Behavior:**

24 a) If the *block* is given, for each key/value pair of the receiver, in an implementation
25 defined order, call the *block* with the key of the pair as the argument. Return the
26 receiver.

27 b) If the *block* is not given, the behavior is implementation dependent.

1 **15.2.13.3.11 Hash#each_value**

2 `each_value(&block)`

3 **Visibility:** public

4 **Behavior:**

5 a) If the *block* is given, call the *block* for each key/value pair of the receiver, with the
6 value as the argument, in an implementation defined order. Return the receiver.

7 b) If the *block* is not given, the behavior is implementation dependent.

8 **15.2.13.3.12 Hash#empty?**

9 `empty?`

10 **Visibility:** public

11 **Behavior:**

12 a) If the receiver is empty, return **true**.

13 b) Otherwise, return **false**.

14 **15.2.13.3.13 Hash#has_key?**

15 `has_key?(key)`

16 **Visibility:** public

17 **Behavior:**

18 a) If the receiver has a key/value pair *P* where the *key* \equiv the key of *P*, return **true**.

19 b) Otherwise, return **false**.

20 **15.2.13.3.14 Hash#has_value?**

21 `has_value?(value)`

22 **Visibility:** public

23 **Behavior:**

1 a) If the receiver has a key/value pair whose value holds the following condition, return
2 **true**.

3 • An invocation of the method `==` on the value with the *value* as the argument result
4 in a true value.

5 b) Otherwise, return **false**.

6 **15.2.13.3.15 Hash#include?**

7 **include?**(*key*)

8 **Visibility:** public

9 **Behavior:** Same as the method `has_key?` (see §15.2.13.3.13).

10 **15.2.13.3.16 Hash#initialize**

11 **initialize**(**args*, &*block*)

12 **Visibility:** private

13 **Behavior:**

14 a) If the *block* is given, and the length of the *args* is not 0, raise a direct instance of the
15 class **ArgumentError**.

16 b) If the *block* is given and the length of the *args* is 0, set the default proc of the receiver
17 to a direct instance of the class **Proc** which represents the *block*.

18 c) If the *block* is not given:

19 1) If the length of the *args* is 0, let *D* be **nil**.

20 2) If the length of the *args* is 1, let *D* be the only argument.

21 3) If the length of the *args* is larger than 1, raise a direct instance of the class
22 **ArgumentError**.

23 4) Set the default value of the receiver to *D*.

24 d) Return an implementation defined value.

25 **15.2.13.3.17 Hash#initialize_copy**

1 `initialize_copy(original)`

2 **Visibility:** private

3 **Behavior:**

4 a) If the *original* is not an instance of the class **Hash**, the behavior is implementation
5 dependent.

6 b) Remove all the key/value pairs from the receiver.

7 c) For each key/value pair *P* of the *original*, in an implementation defined order, store *P*
8 in the receiver.

9 d) Remove the default value and the default proc from the receiver.

10 e) If the *original* has a default value, set the default value of the receiver to that value.

11 f) If the *original* has a default proc, set the default proc of the receiver to that proc.

12 g) Return an implementation defined value.

13 **15.2.13.3.18 Hash#key?**

14 `key?(key)`

15 **Visibility:** public

16 **Behavior:** Same as the method `has_key?` (see §15.2.13.3.13).

17 **15.2.13.3.19 Hash#keys**

18 `keys`

19 **Visibility:** public

20 **Behavior:** The method returns a newly created instance of the class **Array** whose content
21 is the keys of the receiver. The order of the keys stored in somewhere are implementation
22 defined.

23 **15.2.13.3.20 Hash#length**

24 `length`

25 **Visibility:** public

26 **Behavior:** The method returns an instance of the class **Integer** whose value is the number
27 of key/value pairs stored in the receiver.

1 **15.2.13.3.21 Hash#member?**

2 `member?(key)`

3 **Visibility:** public

4 **Behavior:** Same as the method `has_key?` (see §15.2.13.3.13).

5 **15.2.13.3.22 Hash#merge**

6 `merge(other, &block)`

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the *other* is not an instance of the class `Hash`, the behavior is implementation de-
10 pendent.
- 11 b) Otherwise, create a direct instance of the class `Hash` *H* which has the same key/value
12 pairs as the receiver.
- 13 c) For each key/value pair *P* of the *other*, in an implementation defined order:
- 14 1) If the *block* is given:
- 15 i) If *H* has the key/value pair *Q* where the key of *P* \equiv the key of *Q*, call the
16 *block* with three arguments, the key of *P*, the value of *Q*, and the value of *P*.
17 Store the key/value pair whose key is the key of *P* and whose value is the
18 resulting value of this call.
- 19 ii) Otherwise, store *P* in *H*.
- 20 2) If the *block* is not given, store *P* in *H*.
- 21 d) Return *H*.

22 **15.2.13.3.23 Hash#replace**

23 `replace(other)`

24 **Visibility:** public

25 **Behavior:** Same as the method `initialize_copy` (see §15.2.13.3.17).

26 **15.2.13.3.24 Hash#shift**

1 **shift**

2 **Visibility:** public

3 **Behavior:**

4 a) If the receiver has no key/value pairs:

5 1) If the receiver has the default proc, invoke the method **call** on the default proc

6 with two arguments, the receiver and **nil**. Return the resulting value of this call.

7 2) If the receiver has the default value, return the value.

8 3) Otherwise, return **nil**.

9 b) Otherwise, choose a key/value pair *P* and remove *P* from the receiver. Return a newly

10 created instance of the class **Array** which contains two elements, the key and the value

11 of *P*.

12 Which pair is chosen is implementation defined.

13 **15.2.13.3.25 Hash#size**

14 **size**

15 **Visibility:** public

16 **Behavior:** Same as the method **length** (see §15.2.13.3.20).

17 **15.2.13.3.26 Hash#store**

18 **store**(*key*, *value*)

19 **Visibility:** public

20 **Behavior:** Same as the method **[]=** (see §15.2.13.3.3).

21 **15.2.13.3.27 Hash#value?**

22 **value?**(*value*)

23 **Visibility:** public

24 **Behavior:** Same as the method **has_value?** (see §15.2.13.3.14).

1 15.2.13.3.28 Hash#values

2 values

3 **Visibility:** public

4 **Behavior:** The method returns a newly created instance of the class **Array** which contains
5 all the values of the receiver. The order of the values stored are implementation defined.

6 15.2.14 Range

7 Instances of the class **Range** represent ranges between two values, the start and end point.

8 An instance of the class **Range** has the following properties:

9 **start point:** The value at the start of the range.

10 **end point:** The value at the end of the range.

11 **exclusive flag:** If this is **true**, the end point is excluded from the range. Otherwise, the
12 end point is included in the range.

13 When the method **clone** (see §15.3.1.2.8) or the method **dup** (see §15.3.1.2.9) of the class **Kernel**
14 is invoked on an instance of the class **Range**, those properties shall be copied from the receiver
15 to the resulting value.

16 15.2.14.1 Direct superclass

17 The class **Object**

18 15.2.14.2 Included modules

19 The following module is included in the class **Range**.

- 20 • **Enumerable**

21 15.2.14.3 Instance methods

22 15.2.14.3.1 Range#==

23 **==(other)**

24 **Visibility:** public

25 **Behavior:**

26 a) If all of the following conditions hold, return **true**:

- 27 • the *other* is an instance of the class **Range**.

- 1 • Let S be the start point of the *other*. Invocation of the method `==` on the start
- 2 point of the receiver with S as the argument results in a true value.
- 3 • Let E be the end point of the *other*. Invocation of the method `==` on the end point
- 4 of the receiver with E as the argument results in a true value.
- 5 • The exclusive flag of the receiver and the one of the *other* are the same boolean
- 6 value.
- 7 b) Otherwise, return **false**.

8 15.2.14.3.2 Range#===

9 `===(obj)`

10 **Visibility:** public

11 **Behavior:**

- 12 a) If neither the start point of the receiver nor the end point of the receiver is an instance
- 13 of the class **Numeric**, the behavior is implementation dependent.
- 14 b) Invoke the method `<=>` on the start point of the receiver with the *obj* as the argument.
- 15 Let S be the result of this invocation.
- 16 1) If S is not an instance of the class **Integer**, the behavior is implementation de-
- 17 pendent.
- 18 2) If the value of S is larger than 0, return **false**.
- 19 c) Invoke the method `<=>` on the *obj* with the end point of the receiver as the argument.
- 20 Let E be the result of this invocation.
- 21 • If E is not an instance of the class **Integer**, the behavior is implementation de-
- 22 pendent.
- 23 • If the exclusive flag of the receiver is **true**, and the value of E is smaller than 0,
- 24 return **true**.
- 25 • If the exclusive flag of the receiver is **false**, and the value of E is smaller than or
- 26 equal to 0, return **true**.
- 27 • Otherwise, return **false**.

28 15.2.14.3.3 Range#begin

29 `begin`

30 **Visibility:** public

1 **Behavior:** The method returns the start point of the receiver.

2 15.2.14.3.4 Range#each

3 each(&block)

4 **Visibility:** public

5 **Behavior:**

- 6 a) If the *block* is not given, the behavior is implementation dependent.
- 7 b) If an invocation of the method `respond_to?` on the start point of the receiver with a
8 direct instance of the class `Symbol` whose name is `succ` as the argument results in a
9 false value, raise a direct instance of the class `TypeError`.
- 10 c) Let *V* be the start point of the receiver.
- 11 d) Invoke the method `<=>` on *V* with the end point of the receiver as the argument. Let
12 *C* be the resulting value.
- 13 1) If *C* is not an instance of the class `Integer`, the behavior is implementation
14 dependent.
- 15 2) If the value of *C* is larger than 0, return the receiver.
- 16 3) If the value of *C* is 0:
- 17 i) If the exclusive flag of the receiver is `true`, return the receiver.
- 18 ii) If the exclusive flag of the receiver is `false`, call the *block* with *V* as the
19 argument, then, return the receiver.
- 20 e) Call the *block* with *V* as the argument.
- 21 f) Invoke the method `succ` on *V* with no argument, and let new *V* be the resulting value.
- 22 Continue processing from Step d.

23 15.2.14.3.5 Range#end

24 end

25 **Visibility:** public

26 **Behavior:** The method returns the end point of the receiver.

27 15.2.14.3.6 Range#exclude_end?

1 `exclude_end?`

2 **Visibility:** public

3 **Behavior:** The method returns the exclusive flag of the receiver.

4 **15.2.14.3.7 Range#first**

5 `first`

6 **Visibility:** public

7 **Behavior:** Same as the method `begin` (see §15.2.14.3.3).

8 **15.2.14.3.8 Range#include?**

9 `include?(obj)`

10 **Visibility:** public

11 **Behavior:** Same as the method `===` (see §15.2.14.3.2).

12 **15.2.14.3.9 Range#initialize**

13 `initialize(left, right, exclusive=false)`

14 **Visibility:** public

15 **Behavior:**

16 a) Invoke the method `<=>` on the *left* with the *right* as the argument. If an exception
17 is raised and not handled during this invocation, raise a direct instance of the class
18 **ArgumentError**. If the result of this invocation is not an instance of the class **Integer**,
19 the behavior is implementation dependent.

20 b) If the *exclusive* is a true value, let *F* be **true**. Otherwise, let *F* be **false**.

21 c) Set the start point, end point, and exclusive flag of the receiver to the *left*, the *right*,
22 and *F*, respectively.

23 d) Return an implementation defined value.

24 **15.2.14.3.10 Range#last**

1 `last`

2 **Visibility:** public

3 **Behavior:** Same as the method `end` (see §15.2.14.3.5).

4 **15.2.14.3.11 Range#member?**

5 `member?(obj)`

6 **Visibility:** public

7 **Behavior:** Same as the method `===` (see §15.2.14.3.2).

8 **15.2.15 Regexp**

9 Instances of the class **Regexp** represent regular expressions, and have the following properties.

10 **pattern:** A *pattern* of the regular expression (see §15.2.15.3). The default value of this
11 property is empty.

12 If the value of this property is empty when a method is invoked on an instance of the class
13 **Regexp**, except for the invocation of the method `initialize`, the behavior of the invoked
14 method is implementation dependent.

15 **ignorecase:** A boolean value which denotes whether a match is performed in the case
16 insensitive manner. The default value of this property is false.

17 **multiline:** A boolean value which denotes whether the pattern “.” matches a *line-*
18 *terminator* (see §15.2.15.3). The default value of this property is false.

19 **15.2.15.1 Direct superclass**

20 The class **Object**

21 **15.2.15.2 Constants**

22 The following constants are defined in the class **Regexp**.

23 **IGNORECASE:** An instance of the class **Integer** whose value is 2^n , where the integer n
24 is an implementation defined value. The value of this constant shall be different from that
25 of **MULTILINE** described below.

26 **MULTILINE:** An instance of the class **Integer** whose value is 2^m , where the integer m
27 is an implementation defined value.

28 The above constants are used to set the `ignorecase` and `multiline` properties of an instance of
29 the class **Regexp** (see §15.2.15.6.1).

1 15.2.15.3 Patterns

2 Syntax

3 *pattern* ::
4 *alternative*₁
5 | *pattern*₁ | *alternative*₂

6 *alternative* ::
7 [empty]
8 | *alternative*₃ *term*

9 *term* ::
10 *anchor*
11 | *atom*₁
12 | *atom*₂ *quantifier*

13 *anchor* ::
14 ^ | \$

15 *quantifier* ::
16 * | + | ?

17 *atom* ::
18 *pattern-character*
19 | *grouping*
20 | .
21 | *atom-escape-sequence*

22 *pattern-character* ::
23 *source-character* **but not** *regexp-meta-character*

24 *regexp-meta-character* ::
25 | | . | * | + | ^ | ? | (|)
26 | *future-reserved-meta-character*

27 *future-reserved-meta-character* ::
28 [|] | { | }

29 *grouping* ::
30 (*pattern*)

31 *atom-escape-sequence* ::
32 *decimal-escape-sequence*
33 | *regexp-character-escape-sequence*

```

1  decimal-escape-sequence ::
2      \ decimal-digit-without-zero

3  regexp-character-escape-sequence ::
4      regexp-escape-sequence
5      | regexp-non-escaped-sequence
6      | hex-escape-sequence
7      | regexp-octal-escape-sequence
8      | regexp-control-escape-sequence

9  regexp-escape-sequence ::
10     \ regexp-escaped-character

11 regexp-escaped-character ::
12     n | t | r | f | v | a | e | b

13 regexp-non-escaped-sequence ::
14     \ regexp-non-escaped-character

15 regexp-non-escaped-character ::
16     source-character but not regexp-escaped-character

17 regexp-octal-escape-sequence ::
18     octal-escape-sequence but not decimal-escape-sequence

19 regexp-control-escape-sequence ::
20     \ ( C - | c ) regexp-control-escaped-character

21 regexp-control-escaped-character ::
22     regexp-character-escape-sequence
23     | ?
24     | source-character but not ( \ | ? )

```

25 *future-reserved-meta-characters* are reserved for the extension of the pattern of regular expres-
26 sions.

27 Semantics

28 A *pattern* matches the following string:

- 29 a) If the *pattern* is an *alternative*₁, it matches the string which the *alternative*₁ matches.
- 30 b) If the *pattern* is a *pattern*₁ | *alternative*₂, it matches the string which either the *pattern*₁ or
31 the *alternative*₂ matches.

32 An *alternative* matches the following string:

- 1 a) If the *alternative* is [empty], it matches an empty string.
- 2 b) If the *alternative* is an *alternative₃ term*, it matches the concatenation of two strings, the
3 one which the *alternative₃* matches and the other which the *term* matches, in this order.
- 4 A *term* matches the following string:
 - 5 a) If the *term* is an *atom₁*, it matches the string which the *atom₁* matches.
 - 6 b) If the *term* is an *atom₂ quantifier*, it matches a string as follows:
 - 7 1) If the *quantifier* is *, it matches zero or more strings which the *atom₂* matches.
 - 8 2) If the *quantifier* is +, it matches one or more strings which the *atom₂* matches.
 - 9 3) If the *quantifier* is ?, it matches at most one string which the *atom₂* matches.
 - 10 c) If the *term* is an *anchor*, it matches the position within the string *S* which the *pattern* is
11 matched against, as follows:
 - 12 1) If the *anchor* is ^, it matches the beginning of *S* or the position just after a *line-*
13 *terminator* which is followed by at least one character.
 - 14 2) If the *anchor* is \$, it matches the end of *S* or the position just before a *line-terminator*.
- 15 An *atom* matches the following string:
 - 16 a) If the *atom* is a *pattern-character*, it matches a single character *C* represented by the
17 *pattern-character*. If the *atom* occurs in the pattern of an instance of the class **Regex**
18 whose ignorecase property is true, it also matches a corresponding uppercase character of
19 *C*, if *C* is a lowercase character, or a corresponding lowercase character of *C*, if *C* is an
20 uppercase character.
 - 21 b) If the *atom* is a *grouping*, it matches the string which the *grouping* matches.
 - 22 c) If the *atom* is “.”, it matches any character except for a *line-terminator*. If the *atom* occurs
23 in the pattern of an instance of the class **Regex** whose multiline property is true, it also
24 matches a *line-terminator*.
 - 25 d) If the *atom* is an *atom-escape-sequence*, it matches the string which the *atom-escape-*
26 *sequence* matches.
- 27 A *grouping* matches the string which the *pattern* matches.
- 28 An *atom-escape-sequence* matches the following string:
 - 29 a) If the *atom-escape-sequence* is a *decimal-escape-sequence*, it matches the string which the
30 *decimal-escape-sequence* matches.
 - 31 b) If the *atom-escape-sequence* is a *regex-character-escape-sequence*, it matches a string of
32 length one, the content of which is the character which the *regex-character-escape-sequence*
33 represents.

1 A *decimal-escape-sequence* matches the following string:

- 2 a) Let i be an integer represented by *decimal-digit-without-zero*.
- 3 b) Let G be the i th *grouping* in the *pattern*, counted from 1, in the order of the occurrence of
4 “(” of *groupings* from the left of the *pattern*.
- 5 c) If the *decimal-escape-sequence* occurs before G within the *pattern*, it does not match any
6 string.
- 7 d) If G matches any string, the *decimal-escape-sequence* matches the same string.
- 8 e) Otherwise, the *decimal-escape-sequence* does not match any string.

9 A *regex-character-escape-sequence* represents a character as follows:

- 10 • A *regex-escape-sequence* represents a character as shown in Table 1 in §8.5.5.2.2.
- 11 • A *regex-non-escaped-sequence* represents a *regex-non-escaped-character*.
- 12 • A *hex-escape-sequence* represents a character as described in §8.5.5.2.2.
- 13 • A *regex-octal-escape-sequence* is interpreted in the same way as an *octal-escape-sequence*
14 (see §8.5.5.2.2).
- 15 • A *regex-control-escape-sequence* represents a character, the code of which is computed by
16 taking bitwise AND of 0x9f and the code of the character represented by the *regex-control-*
17 *escaped-character*, except when the *regex-control-escaped-character* is ?, in which case, the
18 *regex-control-escape-sequence* represents a character whose code is 127.

19 15.2.15.4 Matching process

20 A *pattern* P is considered to successfully match the given string S , if there exists a substring of
21 S (including S itself) which P matches, satisfying the following conditions.

- 22 a) If P is of the form $pattern_1 \mid alternative_2$, and if both the $pattern_1$ and the $alternative_2$
23 matches some substrings, the matched substring is the one among the substrings matched
24 by the $pattern_1$ which meets conditions b and c as follows.
- 25 b) If P matches more than one substrings, the substring which begins earliest in S is the
26 matched substring. This condition takes precedence over the condition c.
- 27 c) If there are more than one substrings which satisfy the condition b, the longest one is the
28 matched substring.

29 These conditions are applied to any substring matched by any sub-pattern of P as much as
30 possible in the way that the resulting substring matched by P still satisfies the conditions a, b
31 and c.

32 When a numerical offset is specified, P is matched against the part of S which begins at the
33 offset. Note, however, that if the match succeeds, the string property of the resulting instance
34 of the class `MatchData` is S , not the part of S which begins at the offset, as described below.

1 A matching process returns either an instance of the class `MatchData` (see §15.2.16) if the match
2 succeeds or `nil` if the match failed.

3 An instance of the class `MatchData` is created as follows:

4 a) Let B be the substring of S which P matched.

5 b) Create a direct instance of the class `MatchData`, and let M be the instance.

6 c) Set the string of M to S .

7 d) Create a new empty list L .

8 e) Let O be an ordered pair whose first element is B and whose second element is the offset
9 of the first character of B within S , counted from 0. Append O to L .

10 f) For each *grouping* G in P , in the order of the occurrence of its “(” within P , take the
11 following steps:

12 1) If G contributed to the match of P , let B be the substring which G matched. Let O
13 be an ordered pair whose first element is B and whose second element is the offset of
14 the first character of B within S , counted from 0. Append O to L .

15 2) Otherwise, append to L an ordered pair whose elements are both `nil`.

16 g) Set the match result of M to L .

17 h) M is the instance of the class `MatchData` returned by the matching process.

18 A matching process creates or updates a local variable binding with name “~”, which is specifi-
19 cally used by the method `Regexp.last_match` (see §15.2.15.5.3), as follows:

20 a) Let M be the value which the matching process returns.

21 b) If the binding for the name “~” can be resolved by the process described in §9.1.1 as if “~”
22 were a *local-variable-identifier*, replace the value of the binding with M .

23 c) Otherwise, create a local variable binding with name “~” and value M in the uppermost non-
24 block element of `[[local-variable-bindings]]` where the non-block element means the element
25 which does not correspond to a *block*.

26 A conforming processor may name the binding other than “~”; however, it shall not be of the
27 form *local-variable-identifier*.

28 15.2.15.5 Singleton methods

29 15.2.15.5.1 Regexp.compile

30 `Regexp.compile(*args)`

1 **Visibility:** public

2 **Behavior:** Same as the method `new` (see §15.2.3.2.3).

3 15.2.15.5.2 `Regexp.escape`

4 `Regexp.escape(string)`

5 **Visibility:** public

6 **Behavior:**

- 7 a) If the *string* is not an instance of the class `String`, the behavior is implementation
8 dependent.
- 9 b) Let *S* be the content of the *string*.
- 10 c) Return a new instance of the class `String` whose content is same as *S*, except that every
11 occurrences of characters on the left of Table 3 are replaced with the corresponding
12 sequences of characters on the right of the Table 3.

13 15.2.15.5.3 `Regexp.last_match`

14 `Regexp.last_match(*index)`

15 **Visibility:** public

16 **Behavior:**

- 17 a) Search for a binding of a local variable with name “~” as described in §9.1.1 as if “~”
18 were a *local-variable-identifier*.
- 19 b) If the binding is found and its value is an instance of the class `MatchData`, let *M* be
20 the instance. Otherwise, return `nil`.
- 21 c) If the length of the *index* is 0, return *M*.
- 22 d) If the length of the *index* is larger than 1, raise a direct instance of the class `ArgumentError`.
- 23 e) If the length of the *index* is 1, let *A* be the only argument.
- 24 f) If *A* is not an instance of the class `Integer`, the behavior of the method is implemen-
25 tation dependent.
- 26 g) Let *R* be the result returned as if by invoking the method `[]` on *M* with *A* as the only
27 argument (see §15.2.16.2.1).
- 28 h) Return *R*.

Table 3 – Regexp escaped characters

Characters replaced	Escaped sequence
0x0a	\n
0x09	\t
0x0d	\r
0x0c	\f
0x20	\0x20
#	\#
\$	\\$
(\(
)	\)
*	*
+	\+
-	\-
.	\.
?	\?
[\[
\	\\
]	\]
^	\^
{	\{
	\
}	\}

1 **15.2.15.5.4 Regexp.quote**

2 Regexp.quote

3 **Visibility:** public

4 **Behavior:** Same as the method `escape` (see §15.2.15.5.2).

5 **15.2.15.6 Instance methods**

6 **15.2.15.6.1 Regexp#initialize**

7 `initialize(source, flag)`

8 **Visibility:** private

9 **Behavior:**

- 1 a) If the *source* is an instance of the class **Regex**, let *S* be the pattern of the *source*.
2 If the *source* is an instance of the class **String**, let *S* be the content of the *source*.
3 Otherwise, the behavior is implementation dependent.
- 4 b) If *S* cannot be derived from the *pattern* (§15.2.15.3), raise a direct instance of the class
5 **RegexError**.
- 6 c) Set the pattern of the receiver to *S*.
- 7 d) If the *flag* is an instance of the class **Integer**, let *n* be its value.
 - 8 1) If computing bitwise AND of the value of **Regex::IGNORECASE** and *n* results in
9 non-zero value, set the ignorecase property of the receiver to true.
 - 10 2) If computing bitwise AND of the value of **Regex::MULTILINE** and *n* results in
11 non-zero value, set the multiline property of the receiver to true.
- 12 e) If the *flag* is true value other than an instance of the class **Integer**, set the ignorecase
13 property of the receiver to true.
- 14 f) Return an implementation defined value.

15 15.2.15.6.2 **Regex#initialize_copy**

```
16 initialize_copy(original)
```

17 **Visibility:** private

18 **Behavior:**

- 19 a) If the *original* is not an instance of the class of the receiver, raise a direct instance of
20 the class **TypeError**.
- 21 b) Set the pattern of the receiver to the pattern of the *original*.
- 22 c) Set the ignorecase property of the receiver to the ignorecase property of the *original*.
- 23 d) Set the multiline property of the receiver to the multiline property of the *original*.
- 24 e) Return an implementation defined value.

25 15.2.15.6.3 **Regex#==**

```
26 ==( other )
```

27 **Visibility:** public

28 **Behavior:**

- 1 a) If the *other* is not an instance of the class **Regexp**, return **false**.
- 2 b) If the corresponding properties of the receiver and the *other* are the same, return **true**.
- 3 c) Otherwise, return **false**.

4 **15.2.15.6.4 Regexp#===**

5 **===**(*string*)

6 **Visibility:** public

7 **Behavior:**

- 8 a) If the *string* is not an instance of the class **String**, the behavior is implementation dependent.
- 9
- 10 b) Let *S* be the content of the *string*.
- 11 c) Match the pattern of the receiver against *S* (see §15.2.15.3 and Step 15.2.15.4). Let *M*
- 12 be the result of the matching process.
- 13 d) If *M* is an instance of the class **MatchData**, return **true**.
- 14 e) Otherwise, return **false**.

15 **15.2.15.6.5 Regexp#=~**

16 **=~**(*string*)

17 **Visibility:** public

18 **Behavior:**

- 19 a) If the *string* is not an instance of the class **String**, the behavior is implementation dependent.
- 20
- 21 b) Let *S* be the content of the *string*.
- 22 c) Match the pattern of the receiver against *S* (see §15.2.15.3 and Step 15.2.15.4). Let *M*
- 23 be the result of the matching process.
- 24 d) If *M* is **nil** return **nil**.
- 25 e) If *M* is an instance of the class **MatchData**, let *P* be first element of the match result
- 26 property of *M*, and let *i* be the second element of *P*.
- 27 f) Return an instance of the class **Integer** whose value is *i*.

1 **15.2.15.6.6 Regexp#casefold?**

2 **casefold?**

3 **Visibility:** public

4 **Behavior:** The method returns the value of the ignorecase property of the receiver.

5 **15.2.15.6.7 Regexp#match**

6 **match(*string*)**

7 **Visibility:** public

8 **Behavior:**

9 a) If the *string* is not an instance of the class **String**, the behavior is implementation
10 dependent.

11 b) Let *S* be the content of the *string*.

12 c) Match the pattern of the receiver against *S* (see §15.2.15.3 and Step 15.2.15.4). Let *M*
13 be the result of the matching process.

14 d) Return *M*.

15 **15.2.15.6.8 Regexp#source**

16 **source**

17 **Visibility:** public

18 **Behavior:** The method returns a direct instance of the class **String** whose content is the
19 pattern of the receiver.

20 **15.2.16 MatchData**

21 Instances of the class **MatchData** represent results of successful matches of instances of the class
22 **Regexp** against instances of the class **String**.

23 An instance of the class **MatchData** has the properties called ***string*** and ***match result***, which
24 are initialized as described in §15.2.15.4. Elements of the match result are indexed by integers
25 starting from 0.

26 Given an instance of the class **MatchData** *M*, the ***matched substring***, ***pre-match*** and ***post-***
27 ***match*** of *M* are defined as follows:

1 Let S be the string of M . Let F be the first element of the match result of M . Let B and O be
 2 the first portion (the substring matched) and the second portion (offset of that substring) of F .
 3 Let i be the sum of O and the length of B .

4 **matched substring:** The matched substring of M is B .

5 **pre-match:** The pre-match of M is a part of S , from the first up to, but not including the
 6 O th character of S .

7 **post-match:** The post-match of M is a part of S , from the i th up to the last character of
 8 S .

9 **15.2.16.1 Direct superclass**

10 The class `Object`

11 **15.2.16.2 Instance methods**

12 **15.2.16.2.1 MatchData#[]**

13 `[] (*args)`

14 **Visibility:** public

15 **Behavior:** The method behaves as if the method `to_a` were invoked on the receiver (see
 16 §15.2.16.2.12), and then, the method `[]` were invoked on the resulting instance of the class
 17 `Array` with the same arguments passed to an invocation of this method (see §15.2.12.4.4).

18 **15.2.16.2.2 MatchData#begin**

19 `begin(index)`

20 **Visibility:** public

21 **Behavior:**

22 a) If the *index* is not an instance of the class `Integer`, the behavior is implementation
 23 dependent.

24 b) Let L be the match result of the receiver, and let i be the value of the *index*.

25 c) If i is smaller than 0 or equal to, or larger than the number of elements of L , raise a
 26 direct instance of the class `IndexError`.

27 d) Otherwise, return the second portion of the i th element of L .

28 **15.2.16.2.3 MatchData#captures**

1 **captures**

2 **Visibility:** public

3 **Behavior:**

4 a) Let L be the match result of the receiver.

5 b) Create an empty direct instance of the class **Array** A .

6 c) Except for the first element, for each element e of L , in the same order in the list,
7 append to A a direct instance of the class **String** whose content is the first portion of
8 e .

9 d) Return A .

10 **15.2.16.2.4 MatchData#end**

11 **end(*index*)**

12 **Visibility:** public

13 **Behavior:**

14 a) If the *index* is not an instance of the class **Integer**, the behavior is implementation
15 dependent.

16 b) Let L be the match result of the receiver, and let i be the value of the *index*.

17 c) If i is smaller than 0 or equal to, or larger than the number of elements of L , raise a
18 direct instance of the class **IndexError**.

19 d) Let F and S be the first and the second portions of the i th element of L .

20 e) If F is **nil**, return **nil**.

21 f) Otherwise, let f be the length of F . Return an instance of the class **Integer** whose
22 value is the sum of S and f .

23 **15.2.16.2.5 MatchData#initialize_copy**

24 **initialize_copy(*original*)**

25 **Visibility:** private

26 **Behavior:**

- 1 a) If the *original* is not an instance of the class of the receiver, raise a direct instance of
2 the class **TypeError**.
- 3 b) Set the string property of the receiver to the string property of the *original*.
- 4 c) Set the match result property of the receiver to the match result property of the
5 *original*.
- 6 d) Return an implementation defined value.

7 **15.2.16.2.6 MatchData#length**

8 **length**

9 **Visibility:** public

10 **Behavior:** The method returns the number of elements of the match result of the receiver.

11 **15.2.16.2.7 MatchData#offset**

12 **offset**(*index*)

13 **Visibility:** public

14 **Behavior:**

- 15 a) If the *index* is not an instance of the class **Integer**, the behavior is implementation
16 dependent.
- 17 b) Let L be the match result of the receiver, and let i be the value of the *index*.
- 18 c) If i is smaller than 0 or equal to, or larger than the number of elements of L , raise a
19 direct instance of the class **IndexError**.
- 20 d) Let S and b be the first and the second portions of the i th element of L . Let e be the
21 sum of b and the length of S .
- 22 e) Return a new instance of the class **Array** which contains two instances of the class
23 **Integer**, the one whose value is b and the other whose value is e , in this order.

24 **15.2.16.2.8 MatchData#post_match**

25 **post_match**

26 **Visibility:** public

27 **Behavior:** The method returns an instance of the class **String** the content of which is the
28 post-match of the receiver.

1 15.2.16.2.9 MatchData#pre_match

2 pre_match

3 **Visibility:** public

4 **Behavior:** The method returns an instance of the class **String** the content of which is the
5 pre-match of the receiver.

6 15.2.16.2.10 MatchData#size

7 size

8 **Visibility:** public

9 **Behavior:** Same as the method **length** (see §15.2.16.2.6).

10 15.2.16.2.11 MatchData#string

11 string

12 **Visibility:** public

13 **Behavior:** The method returns an instance of the class **String** the content of which is the
14 string of the receiver.

15 15.2.16.2.12 MatchData#to_a

16 to_a

17 **Visibility:** public

18 **Behavior:**

19 a) Let L be the match result of the receiver.

20 b) Create an empty direct instance of the class **Array** A .

21 c) For each element e of L , in the same order in the list, append to A an instance of the
22 class **String** whose content is the first portion of e .

23 d) Return A .

24 15.2.16.2.13 MatchData#to_s

1 `to_s`

2 **Visibility:** public

3 **Behavior:** The method returns an instance of the class `String` the content of which is the
4 matched substring of the receiver.

5 **15.2.17 Proc**

6 Instances of the class `Proc` represent *blocks*.

7 An instance of the class `Proc` has the following property.

8 **block:** The block represented by the instance.

9 **15.2.17.1 Direct superclass**

10 The class `Object`

11 **15.2.17.2 Singleton methods**

12 **15.2.17.2.1 Proc.new**

13 `Proc.new(&block)`

14 **Visibility:** public

15 **Behavior:**

16 a) If the *block* is given, let *B* be the *block*.

17 b) Otherwise:

18 1) If the top of `[[block]]` is block-not-given, raise a direct instance of the class `ArgumentError`.

19 2) Otherwise, let *B* be the top of `[[block]]`.

20 c) Create a new instance of the class `Proc` which has *B* as its block.

21 d) Return the instance.

22 **15.2.17.3 Instance methods**

23 **15.2.17.3.1 Proc#[]**

24 `[](*args)`

1 **Visibility:** public

2 **Behavior:** Same as the method `call` (see §15.2.17.3.3).

3 **15.2.17.3.2 Proc#arity**

4 arity

5 **Visibility:** public

6 **Behavior:** Let B be the block represented by the receiver.

7 a) If a *block-formal-argument* does not occur in B , return an instance of the class `Integer`
8 whose value is implementation defined.

9 b) If a *block-formal-argument* occurs in B :

10 1) If a *block-formal-argument-list* does not occur in the *block-formal-argument*, return
11 an instance of the class `Integer` whose value is 0.

12 2) If a *block-formal-argument-list* occurs in the *block-formal-argument*:

13 i) If the *block-formal-argument-list* is of the form *left-hand-side*, return an in-
14 stance of the class `Integer` whose value is 1.

15 ii) If the *block-formal-argument-list* is of the form *multiple-left-hand-side*:

16 I) If the *multiple-left-hand-side* is of the form *grouped-left-hand-side*, return
17 an instance of the class `Integer` whose value is implementation defined.

18 II) If the *multiple-left-hand-side* is of the form *packing-left-hand-side*, return
19 -1.

20 III) Otherwise, let n be the number of *multiple-left-hand-side-items* of the
21 *multiple-left-hand-side*.

22 IV) If the *multiple-left-hand-side* ends with a *packing-left-hand-side*, return
23 an instance of the class `Integer` whose value is $-(n+1)$.

24 V) Otherwise, return an instance of the class `Integer` whose value is n .

25 **15.2.17.3.3 Proc#call**

26 `call(*args)`

27 **Visibility:** public

28 **Behavior:** Let B be the block of the receiver. Let L be an empty list.

- 1 a) Append each element of the *args*, in the indexing order, to *L*.
- 2 b) Call *B* with *L* as the arguments (see §11.2.2). Let *V* be the result of the call.
- 3 c) Return *V*.

4 15.2.17.3.4 Proc#clone

5 clone

6 **Visibility:** public

7 **Behavior:**

- 8 a) Create a direct instance of the class of the receiver which has no bindings of instance
9 variables. Let *O* be the newly created instance.
- 10 b) For each binding *B* of the instance variables of the receiver, create a variable binding
11 with the same name and value as *B* in the set of bindings of instance variables of *O*.
- 12 c) If the receiver is associated with an eigenclass, let *E_o* be the eigenclass, and take the
13 following steps:
 - 14 1) Create an eigenclass whose direct superclass is the direct superclass of *E_o*. Let *E_n*
15 be the eigenclass.
 - 16 2) For each binding *B_{v1}* of the constants of *E_o*, create a variable binding with the
17 same name and value as *B_{v1}* in the set of bindings of constants of *E_n*.
 - 18 3) For each binding *B_{v2}* of the class variables of *E_o*, create a variable binding with
19 the same name and value as *B_{v2}* in the set of bindings of class variables of *E_n*.
 - 20 4) For each binding *B_m* of the instance methods of *E_o*, create a method binding with
21 the same name and value as *B_m* in the set of bindings of instance methods of *E_n*.
 - 22 5) Associate *O* with *E_n*.
- 23 d) Set the block of *O* to the block of the receiver.
- 24 e) Return *O*.

25 15.2.17.3.5 Proc#dup

26 dup

27 **Visibility:** public

28 **Behavior:**

- 1 a) Create a direct instance of the class of the receiver which has no bindings of instance
- 2 variables. Let *O* be the newly created instance.
- 3 b) Set the block of *O* to the block of the receiver.
- 4 c) Return *O*.

5 **15.2.18 Struct**

6 The class **Struct** is a generator of a structure type which is a class defining a set of fields
7 and methods for accessing these fields. Fields are indexed by integers starting from 0 (see
8 §15.2.18.2.1). An instance of a generated class has values for the set of fields. Those values can
9 be referenced and updated with accessor methods for their fields.

10 **15.2.18.1 Direct superclass**

11 The class **Object**

12 **15.2.18.2 Singleton methods**

13 **15.2.18.2.1 Struct.new**

14 **Struct.new**(*string*, **symbol_list*)

15 **Visibility:** public

16 **Behavior:** The method creates a class defining a set of fields and accessor methods for
17 these fields.

18 When the method is invoked, take the following steps:

- 19 a) Create a direct instance of the class **Class** which has the class **Struct** as its direct
- 20 superclass. Let *C* be that class.
- 21 b) If the *string* is not an instance of the class **String** or the class **Symbol**, the behavior is
- 22 implementation dependent.
- 23 c) If the *string* is an instance of the class **String**, let *N* be the content of the instance.
 - 24 1) If *N* is not of the form *constant-identifier*, raise a direct instance of the class
 - 25 **ArgumentError** error.
 - 26 2) Otherwise,
 - 27 i) If the binding with name *N* exists in the set of bindings of constants in the
 - 28 class **Struct**, replace the value of the binding with *C*.
 - 29 ii) Otherwise, create a constant binding in the class **Struct** with name *N* and
 - 30 value *C*.
- 31 d) If the *string* is an instance of the class **Symbol**, prepend the instance to the *symbol_list*.

- 1 e) Let i be 0.
- 2 f) For each element S of the *symbol_list*, take the following steps:
 - 3 1) Let N be the name designated by S .
 - 4 2) Define a field, which is named N and is indexed by i , in C .
 - 5 3) If N is of the form *local-variable-identifier* or *constant-identifier*:
 - 6 i) Define a method named N in C which takes no arguments, and when invoked,

7 returns the value of the field named N .
 - 8 ii) Define a method named $N=$ (i.e. N postfixed by “=”) in C which takes one

9 argument, and when invoked, sets the value of the field named N to the given

10 argument and returns the argument.
 - 11 4) Increment i by 1.
- 12 g) Return C .

13 Singleton methods of classes created by the **Struct.new**

14 Classes created by the method **Struct.new** are equipped with public singleton methods **new**,
 15 **[]**, and **members**. The following describes those methods, assuming that the name of a class
 16 created by the method **Struct.new** is C .

17 $C.new(*args)$

18 **Visibility:** public

19 **Behavior:**

- 20 a) Create a direct instance of the class with the set of fields the receiver defines. Let I be
 21 the instance.
- 22 b) Invoke the method **initialize** on I with the *args* as the list of arguments.
- 23 c) Return I .

24 $C.[](*args)$

25 **Visibility:** public

26 **Behavior:** Same as the method **new** described above.

1	<hr/> <i>C</i> .members <hr/>
2	Visibility: public
3	Behavior:
4	a) Create a direct instance of the class Array which contains instances of the class String ,
5	each of which represents a field name of the receiver. Let <i>A</i> be the instance of the class
6	Array .
7	The elements in <i>A</i> are arranged in the indexing order of corresponding fields.
8	b) Return <i>A</i> .
9	15.2.18.3 Instance methods
10	15.2.18.3.1 Struct#==
11	<hr/> ==(<i>other</i>) <hr/>
12	Visibility: public
13	Behavior:
14	a) If the <i>other</i> and the receiver is the same object, return true .
15	b) If the class of the <i>other</i> and that of the receiver are different, return false .
16	c) Otherwise, for each field named <i>f</i> of the receiver, take the following steps:
17	1) Let <i>R</i> and <i>O</i> be the values of the fields named <i>f</i> of the receiver and the <i>other</i>
18	respectively.
19	2) If <i>R</i> and <i>O</i> are not the same object,
20	i) Invoke the method == on <i>R</i> with <i>O</i> as the only argument. Let <i>V</i> be the
21	resulting value of the invocation.
22	ii) If <i>V</i> is a false value, return false .
23	d) Return true .
24	15.2.18.3.2 Struct#[]
25	<hr/> [] (<i>name</i>) <hr/>
26	Visibility: public

Behavior:

a) If the *name* is an instance of the class **Symbol** or the class **String**:

1) Let *N* be the name designated by the *name*.

2) If the receiver has the field named *N*, return the value of the field.

3) Otherwise, let *S* be an instance of the class **Symbol** with name *N* and raise a direct instance of the class **NameError** which has *S* as its name property.

b) If the *name* is an instance of the class **Integer**, let *i* be the value of the *name*. Let *n* be the number of the fields of the receiver.

1) If *i* is negative, replace *i* with $n + i$.

2) If *i* is still negative or *i* equal or larger than *n*, raise a direct instance of the class **IndexError**.

3) Otherwise, return the value of the field whose index is *i*.

c) Otherwise, the behavior of the method is implementation dependent.

15.2.18.3.3 Struct#[[]=

[[]=(*name*, *obj*)

Visibility: public

Behavior:

a) If the *name* is an instance of the class **Symbol** or an instance of the class **String**:

1) Let *N* be the name designated by the *name*.

2) If the receiver has the field named *N*,

i) Replace the value of the field with the *obj*,

ii) Return the *obj*.

3) Otherwise, let *S* be an instance of the class **Symbol** with name *N* and raise a direct instance of the class **NameError** which has *S* as its name property.

b) If the *string* is an instance of the class **Integer**, let *i* be the value of the *name*. Let *n* be the number of the fields of the receiver.

1) If *i* is negative, replace *i* with $n + i$.

- 1 2) If i is still negative or i equal or larger than n , raise a direct instance of the class
- 2 `IndexError`.
- 3 3) Otherwise,
- 4 i) Replace the value of the field whose index is i with the *obj*
- 5 ii) Return the *obj*.
- 6 c) Otherwise, the behavior of the method is implementation dependent.

7 **15.2.18.3.4 Struct#each**

8 `each(&block)`

9 **Visibility:** public

10 **Behavior:**

- 11 a) If the *block* is not given, the behavior is implementation dependent.
- 12 b) For each field of the receiver, in the indexing order, call the *block* with the value of the
- 13 field as the only argument.
- 14 c) Return the receiver.

15 **15.2.18.3.5 Struct#each_pair**

16 `each_pair(&block)`

17 **Visibility:** public

18 **Behavior:**

- 19 a) If the *block* is not given, the behavior is implementation dependent.
- 20 b) For each field of the receiver, in the indexing order, take the following steps:
 - 21 1) Let N and V be the name and the value of the field respectively. Let S be an
 - 22 instance of the class `Symbol` with name N .
 - 23 2) Call the *block* with the list of arguments which contains S and V in this order.
- 24 c) Return the receiver.

25 **15.2.18.3.6 Struct#members**

1	members
2	Visibility: public
3	Behavior: Same as the method members described in §15.2.18.2.1.
4	15.2.18.3.7 Struct#select
5	select(&block)
6	Visibility: public
7	Behavior:
8	a) If the <i>block</i> is not given, the behavior is implementation dependent.
9	b) Create an empty instance the class Array . Let <i>A</i> be the instance.
10	c) For each field of the receiver, in the indexing order, take the following steps:
11	1) Let <i>V</i> be the value of the field.
12	2) Call the <i>block</i> with <i>V</i> as the only argument. Let <i>R</i> be the resulting value of the
13	call.
14	3) If <i>R</i> is a true value, append <i>V</i> to <i>A</i> .
15	d) Return <i>A</i> .
16	15.2.18.3.8 Struct#initialize
17	initialize(*args)
18	Visibility: private
19	Behavior: Let N_a be the length of the <i>args</i> , and let N_f be the number of the fields of the
20	receiver.
21	a) If N_a is larger than N_f , raise a direct instance of the class ArgumentError .
22	b) Otherwise, for each field <i>f</i> of the receiver, let <i>i</i> be the index of <i>f</i> , and set the value of <i>f</i>
23	to the <i>i</i> th element of the <i>args</i> , or to nil when <i>i</i> is equal to or larger than N_a .
24	c) Return an implementation defined value.
25	15.2.18.3.9 Struct#initialize_copy

1 `initialize_copy(original)`

2 **Visibility:** private

3 **Behavior:**

4 a) If the receiver and the *original* are the same object, return an implementation defined
5 value.

6 b) If the *original* is not an instance of the class of the receiver, raise a direct instance of
7 the class `TypeError`.

8 c) If the number of the fields of the receiver and the number of the fields of the *original*
9 are different, raise a direct instance of the class `TypeError`.

10 d) For each field *f* of the *original*, let *i* be the index of *f*, and set the value of the *i*th field
11 of the receiver to the value of *f*.

12 e) Return an implementation defined value.

13 **15.2.19 Time**

14 Instances of the class `Time` represent dates and times.

15 An instance of the class `Time` holds the following data.

16 **Microseconds:** Microseconds since January 1, 1970 00:00 UTC. Microseconds is an inte-
17 ger whose range is implementation defined. If an out of range value is given as microsec-
18 onds when creating an instance of the class `Time`, a direct instance of either of the class
19 `ArgumentError` or the class `RangeError` shall be raised.

20 **Time zone:** The time zone.

21 **15.2.19.1 Direct superclass**

22 The class `Object`

23 **15.2.19.2 Time computation**

24 Mathematical functions introduced in this subclause are used throughout the descriptions in
25 §15.2.19. These functions are assumed to compute exact mathematical results using mathemat-
26 ical real numbers.

27 **15.2.19.2.1 Day**

The number of microseconds of a day is computed as follows:

$$MicroSecPerDay = 24 \times 60 \times 60 \times 10^6$$

The number of days since January 1, 1970 00:00 UTC which corresponds to microseconds t is computed as follows:

$$Day(t) = floor\left(\frac{t}{MicroSecPerDay}\right)$$

$floor(t)$ = The integer x such that $x \leq t < x + 1$

The weekday which corresponds to microseconds t is computed as follows:

$$WeekDay(t) = (Day(t) + 4) \text{ modulo } 7$$

1 15.2.19.2.2 Year

2 A year has 365 days, except for leap years, which have 366 days. Leap years are those which
3 are either:

- 4 • divisible by 4 and not divisible by 100, or
- 5 • divisible by 400.

The number of days from January 1, 1970 00:00 UTC to the beginning of a year y is computed as follows:

$$DayFromYear(y) = 365 \times (y - 1970) + floor\left(\frac{y - 1969}{4}\right) - floor\left(\frac{y - 1901}{100}\right) + floor\left(\frac{y - 1601}{400}\right)$$

Microseconds elapsed since January 1, 1970 00:00 UTC until the beginning of y is computed as follows:

$$MicroSecFromYear(y) = DayFromYear(y) \times MicroSecPerDay$$

The year number y which corresponds to microseconds t measured from January 1, 1970 00:00 UTC is computed as follows.

$$YearFromTime(t) = y \text{ such that, } MicroSecFromYear(y - 1) < t \leq MicroSecFromYear(y)$$

The number of days from the beginning of the year for the given microseconds t is computed as follows.

$$DayWithinYear(t) = Day(t) - DayFromYear(YearFromTime(t))$$

6 15.2.19.2.3 Month

7 Months have usual number of days. Leap years have the extra day in February. Each month is
8 identified by the number in the range 1 to 12, in the order from January to December.

The month number which corresponds to microseconds t measured from January 1, 1970 00:00 UTC is computed as follows.

$$MonthFromTime(t) = \begin{cases} 1 & \text{if } 0 \leq DayWithinYear(t) < 31 \\ 2 & \text{if } 31 \leq DayWithinYear(t) < 59 \\ 3 & \text{if } 59 + LeapYear(t) \leq DayWithinYear(t) < 90 + LeapYear(t) \\ 4 & \text{if } 90 + LeapYear(t) \leq DayWithinYear(t) < 120 + LeapYear(t) \\ 5 & \text{if } 120 + LeapYear(t) \leq DayWithinYear(t) < 151 + LeapYear(t) \\ 6 & \text{if } 151 + LeapYear(t) \leq DayWithinYear(t) < 180 + LeapYear(t) \\ 7 & \text{if } 181 + LeapYear(t) \leq DayWithinYear(t) < 212 + LeapYear(t) \\ 8 & \text{if } 212 + LeapYear(t) \leq DayWithinYear(t) < 243 + LeapYear(t) \\ 9 & \text{if } 243 + LeapYear(t) \leq DayWithinYear(t) < 273 + LeapYear(t) \\ 10 & \text{if } 273 + LeapYear(t) \leq DayWithinYear(t) < 304 + LeapYear(t) \\ 11 & \text{if } 304 + LeapYear(t) \leq DayWithinYear(t) < 334 + LeapYear(t) \\ 12 & \text{if } 334 + LeapYear(t) \leq DayWithinYear(t) < 365 + LeapYear(t) \end{cases}$$

$$LeapYear(t) = \begin{cases} 1 & \text{if } YearFromTime(t) \text{ is a leap year} \\ 0 & \text{otherwise} \end{cases}$$

1 15.2.19.2.4 Days of month

The day of the month which corresponds to microseconds t measured from January 1, 1970 00:00 UTC is computed as follows.

$$DayWithinMonth(t) = \begin{cases} DayWithinYear(t) + 1 & \text{if } MonthFromTime(t) = 1 \\ DayWithinYear(t) - 30 & \text{if } MonthFromTime(t) = 2 \\ DayWithinYear(t) - 58 - LeapYear(t) & \text{if } MonthFromTime(t) = 3 \\ DayWithinYear(t) - 89 - LeapYear(t) & \text{if } MonthFromTime(t) = 4 \\ DayWithinYear(t) - 119 - LeapYear(t) & \text{if } MonthFromTime(t) = 5 \\ DayWithinYear(t) - 150 - LeapYear(t) & \text{if } MonthFromTime(t) = 6 \\ DayWithinYear(t) - 180 - LeapYear(t) & \text{if } MonthFromTime(t) = 7 \\ DayWithinYear(t) - 211 - LeapYear(t) & \text{if } MonthFromTime(t) = 8 \\ DayWithinYear(t) - 242 - LeapYear(t) & \text{if } MonthFromTime(t) = 9 \\ DayWithinYear(t) - 272 - LeapYear(t) & \text{if } MonthFromTime(t) = 10 \\ DayWithinYear(t) - 303 - LeapYear(t) & \text{if } MonthFromTime(t) = 11 \\ DayWithinYear(t) - 333 - LeapYear(t) & \text{if } MonthFromTime(t) = 12 \end{cases}$$

2 15.2.19.2.5 Hours, Minutes, and Seconds

The number of microseconds in an hour, a minute, a second are as follows:

$$MicroSecPerHour = 60 \times 60 \times 10^6$$

$$MicroSecPerMinute = 60 \times 10^6$$

$$MicroSecPerSecond = 10^6$$

The hour, the minute, and the second which correspond to microseconds t measured from January 1, 1970 00:00 UTC are computed as follows.

$$\begin{aligned} \text{HourFromTime}(t) &= \text{floor}\left(\frac{t}{\text{MicroSecPerHour}}\right) \text{ modulo } 24 \\ \text{MinuteFromTime}(t) &= \text{floor}\left(\frac{t}{\text{MicroSecPerMinute}}\right) \text{ modulo } 60 \\ \text{SecondFromTime}(t) &= \text{floor}\left(\frac{t}{\text{MicroSecPerSecond}}\right) \text{ modulo } 60 \end{aligned}$$

1 15.2.19.3 Time zone and Local time

2 The current time zone is determined from time zone information provided by the underlying
3 system. If the system does not provide information on the current time zone, the time zone of
4 an instance of the class **Time** is implementation defined.

The local time for an instance of the class **Time** is computed from its microseconds t and time zone z as follows.

$$\begin{aligned} \text{LocalTime} &= t + \text{ZoneOffset}(z) \\ \text{ZoneOffset}(z) &= \text{UTC offset of } z \text{ measured in microseconds} \end{aligned}$$

5 15.2.19.4 Daylight saving time

6 On system where it is possible to determine the daylight saving time for each time zone, a
7 conforming processor should adjust the microseconds of an instance of the class **Time** if that
8 microseconds falls within the daylight saving time of the time zone of the instance. An algorithm
9 used for the adjustment is implementation defined.

10 15.2.19.5 Singleton methods

11 15.2.19.5.1 Time.at

12 **Time.at**(**args*)

13 **Visibility:** public

14 **Behavior:**

15 a) If the length of the *args* is 0 or larger than 2, raise a direct instance of the class
16 **ArgumentError**.

17 b) If the length of the *args* is 1, let A be the only argument.

18 1) If A is an instance of the class **Time**, return a new instance of the class **Time** which
19 represents the same time and has the same time zone as A .

20 2) If A is an instance of the class **Integer** or an instance of the class **Float**:

- 1 i) If A is an instance of the class **Integer**, let N_S be the value of A . Let N_M be
2 0.
- 3 ii) If A is an instance of the class **Float**, let F be the value of A . Let N_S be the
4 largest integer such that $N_S \leq F$. Let N_M be an integer which is the result
5 of computing $(F - N_S) \times 10^6$, rounded off the first decimal place.
- 6 iii) Create a direct instance of the class **Time** which represents the time at $N_S \times$
7 $10^6 + N_M$ microseconds since January 1, 1970 00:00 UTC, with the current
8 local time zone.
- 9 iv) Return the resulting instance.
- 10 3) Otherwise, the behavior is implementation dependent.
- 11 c) If the length of the *args* is 2, let S and M be the first and second element of the *args*.
- 12 1) i) If S is an instance of the class **Integer**, let N_S be the value of A .
- 13 ii) If S is an instance of the class **Float**, let F be the value of A . If F is positive,
14 let N_M be the largest integer such that $N \leq F$. Otherwise, let N_M be the
15 smallest integer such that $N \geq F$.
- 16 iii) Otherwise, the behavior is implementation dependent.
- 17 2) Compute an integer which corresponds to M in the same way as S as described
18 in Step c-1-i and c-1-ii. Let N_M be the integer.
- 19 3) Create a direct instance of the class **Time** which represents the time at $N_S \times 10^6 +$
20 N_M microseconds since January 1, 1970 00:00 UTC, with the current local time
21 zone.
- 22 4) Return the resulting instance.

23 **15.2.19.5.2 Time.gm**

24 **Time.gm**(*year*, *month*=1, *day*=0, *hour*=0, *min*=0, *sec*=0, *usec*=0)

25 **Visibility:** public

26 **Behavior:**

- 27 a) Compute an integer value for the *year*, *day*, *hour*, *min*, *sec*, and *usec* as described
28 below. Let Y , D , H , Min , S , and U be integers thus converted.

29 An integer I is determined from the given object O as follows:

- 30 1) If O is an instance of the class **Integer**, let I be the value of O .
- 31 2) If O is an instance of the class **Float**, let I be the integral part of the value of O .

1 3) If O is an instance of the class **String**:

2 i) If the content of O is a sequence of *decimal-digits*, let I be the value of those
3 sequence of digits computed using base 10.

4 ii) Otherwise, the behavior is implementation dependent.

5 4) Otherwise, the behavior is implementation dependent.

6 b) Compute an integer value from the *month* as follows:

7 1) If the *month* is not an instance of the class **String**, the behavior is implementation
8 dependent.

9 2) If the content of the *month* is the same as one of the names of the months in
10 the upper row on Table 4, ignoring the differences in case, let Mon be the integer
11 which corresponds to the *month* in the lower row on the same table.

12 3) If the first character of the *month* is *decimal-digit*, compute an integer value from
13 the *month* as in Step a. Let Mon be the resulting integer.

14 4) Otherwise, raise a direct instance of the class **ArgumentError**.

15 c) If Y is an integer such that $0 \leq Y \leq 38$, replace Y with $2000 + Y$.

16 d) If Y is an integer such that $69 \leq Y \leq 138$, replace Y with $1900 + Y$.

17 e) If each integer computed above is outside the range as listed below, raise a direct
18 instance of the class **ArgumentError**.

19 • $1 \leq Mon \leq 12$

20 • $1 \leq D \leq 31$

21 • $0 \leq H \leq 23$

22 • $0 \leq Min \leq 59$

23 • $0 \leq S \leq 60$

24 Whether any conditions are placed on Y is implementation defined.

25 f) Let t be an integer which satisfies all of the following equations.

26 • $YearFromTime(t) = Y$

27 • $MonthFromTime(t) = Mon$

28 • $DayWithinMonth(t) = 1$

g) Compute microseconds T as follows.

$$T = t + D \times \text{MicroSecPerDay} + H \times \text{MicroSecPerHour} + \\ \text{Min} \times \text{MicroSecPerMinute} + S \times 10^6 + U$$

h) Create a direct instance of the class **Time** which represents the time at T since January 1, 1970 00:00 UTC, with the UTC time zone.

i) Return the resulting instance.

Table 4 – The names of months and corresponding integer

1	2	3	4	5	6	7	8	9	10	11	12
Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec

15.2.19.5.3 **Time.local**

```
Time.local( year, month=1, day=0, hour=0, min=0, sec=0, usec=0 )
```

Visibility: public

Behavior: Same as the method **Time.gm** (see §15.2.19.5.2), except that the method returns a direct instance of the class **Time** which has the current local time zone as its time zone.

15.2.19.5.4 **Time.mktime**

```
Time.mktime( year, month=1, day=0, hour=0, min=0, sec=0, usec=0 )
```

Visibility: public

Behavior: Same as the method **Time.local** (see §15.2.19.5.3).

15.2.19.5.5 **Time.now**

```
Time.now
```

Visibility: public

Behavior: This method returns a direct instance of the class **Time** which represents the current time with the current time zone.

The behavior of this method is the same as the method **new** (see §15.2.3.2.3).

15.2.19.5.6 **Time.utc**

1 `Time.utc(year, month=1, day=0, hour=0, min=0, sec=0, usec=0)`

2 **Visibility:** public

3 **Behavior:** Same as the method `Time.gm` (see §15.2.19.5.2).

4 **15.2.19.6 Instance methods**

5 **15.2.19.6.1 Time#+**

6 `+(offset)`

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the *offset* is not an instance of the class `Integer` or the class `Float`, the behavior is
10 implementation dependent.
- 11 b) Let *V* be the value of the *offset*.
- 12 c) Let *o* be an integer which is the result of computing $V \times 10^6$, rounded of the first
13 decimal place, if any.
- 14 d) Let *t* and *z* be the microseconds and time zone of the receiver.
- 15 e) Create a direct instance of the class `Time` which represents the time at $(t + o)$ microsec-
16 onds since January 1, 1970 00:00 UTC, with *z* as its time zone.
- 17 f) Return the resulting instance.

18 **15.2.19.6.2 Time#-**

19 `-(offset)`

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the *offset* is not an instance of the class `Integer` or the class `Float`, the behavior is
23 implementation dependent.
- 24 b) Let *V* be the value of the *offset*.
- 25 c) Let *o* be an integer which is the result of computing $V \times 10^6$, rounded of the first
26 decimal place, if any.

- 1 d) Let t and z be the microseconds and time zone of the receiver.
- 2 e) Create a direct instance of the class **Time** which represents the time at $t-o$ microseconds
- 3 since January 1, 1970 00:00 UTC, with z as its time zone.
- 4 f) Return the resulting instance.

5 **15.2.19.6.3 Time#<=>**

6 **<=>(other)**

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the *other* is not an instance of the class **Time**, return **nil**.
- 10 b) Otherwise, let T_r and T_o be microseconds of the receiver and the *other*, respectively.
 - 11 1) If $T_r > T_o$, return an instance of the class **Integer** whose value is 1.
 - 12 2) If $T_r = T_o$, return an instance of the class **Integer** whose value is 0.
 - 13 3) If $T_r < T_o$, return an instance of the class **Integer** whose value is -1.

14 **15.2.19.6.4 Time#asctime**

15 **asctime**

16 **Visibility:** public

17 **Behavior:**

- 18 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- 19 b) Let W be the name of the day of the week in the second row on Table 5 which
- 20 corresponds to $WeekDay(t)$ in the upper row on the same table.
- 21 c) Let Mon be the name of the month in the second row on Table 4 which corresponds
- 22 to $MonthFromTime(t)$ in the upper row on the same table.
- d) Let D , Min , S , and Y be as follows:

$$D = DayWithinMonth(t)$$

$$H = HourFromTime(t)$$

$$M = MinuteFromTime(t)$$

$$S = SecondFromTime(t)$$

$$Y = YearFromTime(t)$$

- e) Create a direct instance of the class `String`, the content of which is the following sequence of characters:

W Mon D H:M:S Y<line-terminator>

- f) Return the resulting instance.

Table 5 – The names of the days of the week corresponding integer

0	1	2	3	4	5	6
Sun	Mon	Tue	Wed	Thu	Fly	Sat

15.2.19.6.5 Time#ctime

`ctime`

Visibility: public

Behavior: Same as the method `asctime` (see §15.2.19.6.4).

15.2.19.6.6 Time#day

`day`

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- b) Compute *DayWithinMonth*(t).
- c) Return an instance of the class `Integer` whose value is the result of Step b.

15.2.19.6.7 Time#dst?

`dst?`

Visibility: public

Behavior: Let T and Z be the microseconds and time zone of the receiver.

- a) If T falls within the daylight saving time of Z , return `true`.
- b) Otherwise, return `false`.

1 **15.2.19.6.8 Time#getgm**

2 getgm

3 **Visibility:** public

4 **Behavior:** Same as the method `getutc` (see §15.2.19.6.10).

5 **15.2.19.6.9 Time#getlocal**

6 getlocal

7 **Visibility:** public

8 **Behavior:** The method returns a new instance of the class `Time` which has the same
9 microseconds as the receiver, but has current local time zone as its time zone.

10 **15.2.19.6.10 Time#getutc**

11 getutc

12 **Visibility:** public

13 **Behavior:** The method returns a new instance of the class `Time` which has the same
14 microseconds as the receiver, but has UTC as its time zone.

15 **15.2.19.6.11 Time#gmt?**

16 gmt?

17 **Visibility:** public

18 **Behavior:** Same as the method `utc?` (see §15.2.19.6.26).

19 **15.2.19.6.12 Time#gmt_offset**

20 gmt_offset

21 **Visibility:** public

22 **Behavior:** Same as the method `utc_offset` (see §15.2.19.6.27).

23 **15.2.19.6.13 Time#gmtime**

1 `gmtime`

2 **Visibility:** public

3 **Behavior:** Same as the method `utc` (see §15.2.19.6.25).

4 **15.2.19.6.14 Time#gmtoff**

5 `gmtoff`

6 **Visibility:** public

7 **Behavior:** Same as the method `utc_offset` (see §15.2.19.6.27).

8 **15.2.19.6.15 Time#hour**

9 `hour`

10 **Visibility:** public

11 **Behavior:**

12 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.

13 b) Compute *HourFromTime*(t).

14 c) Return an instance of the class `Integer` whose value is the result of Step b.

15 **15.2.19.6.16 Time#localtime**

16 `localtime`

17 **Visibility:** public

18 **Behavior:**

19 a) Change the time zone of the receiver to the current local time zone.

20 b) Return the receiver.

21 **15.2.19.6.17 Time#mday**

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

mday

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- b) Compute *DayWithinMonth*(t).
- c) Return an instance of the class **Integer** whose value is the result of Step b.

15.2.19.6.18 Time#min

min

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- b) Compute *MinuteFromTime*(t).
- c) Return an instance of the class **Integer** whose value is the result of Step b.

15.2.19.6.19 Time#mon

mon

Visibility: public

Behavior:

- a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- b) Compute *MonthFromTime*(t).
- c) Return an instance of the class **Integer** whose value is the result of Step b.

15.2.19.6.20 Time#month

month

Visibility: public

Behavior: Same as the method **mon** (see §15.2.19.6.19).

1 **15.2.19.6.21 Time#sec**

2 **sec**

3 **Visibility:** public

4 **Behavior:**

- 5 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- 6 b) Compute *SecondFromTime*(t).
- 7 c) Return an instance of the class **Integer** whose value is the result of Step b.

8 **15.2.19.6.22 Time#to_f**

9 **to_f**

10 **Visibility:** public

11 **Behavior:** Let t the microseconds of the receiver.

- 12 a) Compute $t/10^6$.
- 13 b) Create a direct instance of the class **Float** whose value is the result of Step a.
- 14 c) Return the resulting instance.

15 **15.2.19.6.23 Time#to_i**

16 **to_i**

17 **Visibility:** public

18 **Behavior:** Let t the microseconds of the receiver.

- 19 a) Compute *floor*($t/10^6$).
- 20 b) Return an instance of the class **Integer** whose value is the result of Step a.

21 **15.2.19.6.24 Time#usec**

22 **usec**

23 **Visibility:** public

- 1 **Behavior:**
- 2 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.
- 3 b) Compute t modulo 10^6 .
- 4 c) Return the resulting instance.

5 **15.2.19.6.25 Time#utc**

6 **utc**

7 **Visibility:** public

8 **Behavior:**

- 9 a) Change the time zone of the receiver to UTC.
- 10 b) Return the receiver.

11 **15.2.19.6.26 Time#utc?**

12 **utc?**

13 **Visibility:** public

14 **Behavior:** Let Z be the time zone of the receiver.

- 15 a) If Z is UTC, return **true**.
- 16 b) Otherwise, return **false**.

17 **15.2.19.6.27 Time#utc_offset**

18 **utc_offset**

19 **Visibility:** public

20 **Behavior:** Let Z be the time zone of the receiver.

- 21 a) Compute $\text{floor}(\text{ZoneOffset}(Z)/10^6)$.
- 22 b) Return an instance of the class **Integer** whose value is the result of Step a.

23 **15.2.19.6.28 Time#wday**

1 **wday**

2 **Visibility:** public

3 **Behavior:**

4 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.

5 b) Compute *WeekDay*(t).

6 c) Return an instance of the class **Integer** whose value is the result of Step b

7 **15.2.19.6.29 Time#yday**

8 **yday**

9 **Visibility:** public

10 **Behavior:**

11 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.

12 b) Compute *DayWithinYear*(t).

13 c) Return an instance of the class **Integer** whose value is the result of Step b.

14 **15.2.19.6.30 Time#year**

15 **year**

16 **Visibility:** public

17 **Behavior:**

18 a) Compute the local time from the receiver (see §15.2.19.3). Let t be the result.

19 b) Compute *YearFromTime*(t).

20 c) Return an instance of the class **Integer** whose value is the result of Step b.

21 **15.2.19.6.31 Time#zone**

22 **zone**

23 **Visibility:** public

- 1 **Behavior:** Let Z be the time zone of the receiver.
- 2 a) Create a direct instance of the class **String**, the content of which represents Z . The
- 3 exact content of the instance is implementation dependent.
- 4 b) Return the resulting instance.

5 **15.2.19.6.32 Time#initialize**

6 **initialize**

7 **Visibility:** private

8 **Behavior:**

- 9 a) Set the microseconds of the receiver to microseconds elapsed since January 1, 1970
- 10 00:00 UTC.
- 11 b) Set the time zone of the receiver to the current time zone.
- 12 c) Return an implementation defined value.

13 **15.2.19.6.33 Time#initialize_copy**

14 **initialize_copy**(*original*)

15 **Visibility:** private

16 **Behavior:**

- 17 a) If the *original* is not an instance of the class **Time**, raise a direct instance of the class
- 18 **TypeError**.
- 19 b) Set the microseconds of the receiver to the microseconds of the *original*.
- 20 c) Set the time zone of the receiver to the time zone of the *original*.
- 21 d) Return an implementation defined value.

22 **15.2.20 IO**

23 An instance of the class **IO** represents a stream, which is a source and/or a sink of data.

24 An instance of the class **IO** has the following properties:

25 **readability flag:** A boolean value which denotes whether the stream can handle input

26 operations.

27 An instance of the class **IO** is said to be readable if and only if this flag is true.

Reading from a stream which is not readable raises a direct instance of the class `IOError`.

writability flag: A boolean value which denotes whether the stream can handle output operations.

An instance of the class `IO` is said to be writable if and only if this flag is true.

Writing to a stream which is not writable raises a direct instance of the class `IOError`.

openness flag: A boolean value which denotes whether the stream is open.

An instance of the class `IO` is said to be open if and only if this flag is true. An instance of the class `IO` is said to be closed if and only if this flag is false.

A closed stream is neither readable nor writable. Reading from or writing to a stream which is not open raises an instance of the class `IOError`.

buffering flag: A boolean value which denotes whether the data to be written to the stream is buffered.

When this flag is true, a conforming processor may delay the output to the receiver until the instance methods `flush` or `close` is invoked.

A conforming processor may raise an instance of the class `SystemCallError` during the execution of instance methods of the class `IO`.

In the following description of the methods of the class `IO`, a **byte** means an integer from 0 to 255.

15.2.20.1 Direct superclass

The class `Object`

15.2.20.2 Included modules

The following module is included in the class `IO`.

- `Enumerable`

15.2.20.3 Singleton methods

15.2.20.3.1 `IO.open`

`IO.open(*args, &block)`

Visibility: public

Behavior:

- Invoke the method `new` on the receiver with all the elements of the *args* as the arguments. Let *I* be the resulting value.

- 1 b) If the *block* is not given, return *I*.
- 2 c) Otherwise, call the *block* with *I* as the argument. Let *V* be the resulting value.
- 3 d) Invoke the method `close` on *I* with no arguments, even when an exception is raised
- 4 and not handled in Step c.
- 5 e) Return *V*.

6 **15.2.20.4 Instance methods**

7 **15.2.20.4.1 IO#close**

8 `close`

9 **Visibility:** public

10 **Behavior:**

- 11 a) If the receiver is closed, raise a direct instance of the class `IOError`.
- 12 b) If the buffering flag of the receiver is true, and the receiver is buffering any output,
- 13 immediately write all the buffered data to the stream which the receiver represents.
- 14 c) Set the openness flag of the receiver to false.
- 15 d) Return `nil`.

16 **15.2.20.4.2 IO#closed?**

17 `closed?`

18 **Visibility:** public

19 **Behavior:**

- 20 a) If the receiver is closed, return `true`.
- 21 b) Otherwise, return `false`.

22 **15.2.20.4.3 IO#each**

23 `each(&block)`

24 **Visibility:** public

25 **Behavior:**

- 1 a) If the *block* is not given, the behavior is implementation dependent.
- 2 b) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 3 c) If the receiver has reached its end, return the receiver.
- 4 d) Otherwise, read characters from the receiver until `0x0a` is read or the receiver reaches
5 its end.
- 6 e) Call the *block* with an argument, a direct instance of the class `String` whose content
7 is the sequence of characters read in Step d.
- 8 f) Continue processing from Step c.

9 15.2.20.4.4 IO#each_byte

10 `each_byte(&block)`

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the *block* is not given, the behavior is implementation dependent.
- 14 b) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 15 c) If the receiver has reached its end, return the receiver.
- 16 d) Otherwise, read a single byte from the receiver. Call the *block* with an argument, an
17 instance of the class `Integer` whose value is the byte.
- 18 Continue processing from Step c.

19 15.2.20.4.5 IO#each_line

20 `each_line(&block)`

21 **Visibility:** public

22 **Behavior:** Same as the method `each` (see §15.2.20.4.3).

23 15.2.20.4.6 IO#eof?

24 `eof?`

25 **Visibility:** public

26 **Behavior:**

- 1 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 2 b) If the receiver has reached its end, return `true`. Otherwise, return `false`.

3 **15.2.20.4.7 IO#flush**

4 **flush**

5 **Visibility:** public

6 **Behavior:**

- 7 a) If the receiver is not writable, raise a direct instance of the class `IOError`.
- 8 b) If the buffering flag of the receiver is true, and the receiver is buffering any output,
9 immediately write all the buffered data to the stream which the receiver represents.
- 10 c) Return the receiver.

11 **15.2.20.4.8 IO#getc**

12 **getc**

13 **Visibility:** public

14 **Behavior:**

- 15 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 16 b) If the receiver has reached its end, return `nil`.
- 17 c) Otherwise, read a character from the receiver. Return an instance of the class `Object`
18 which represents the character.

19 **15.2.20.4.9 IO#gets**

20 **gets**

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the receiver is not readable, raise a direct instance of the class `IOError`.
- 24 b) If the receiver has reached its end, return `nil`.
- 25 c) Otherwise, read characters from the receiver until `0x0a` is read or the receiver reaches
26 its end.

- 1 d) Return a direct instance of the class **String** whose content is the sequence of characters
2 read in Step c.

3 **15.2.20.4.10 IO#initialize_copy**

4 initialize_copy(*original*)

5 **Visibility:** private

6 **Behavior:** The behavior of the method is implementation dependent.

7 **15.2.20.4.11 IO#print**

8 print(**args*)

9 **Visibility:** public

10 **Behavior:**

11 a) For each element of the *args* in the indexing order:

- 12 1) If the element is **nil**, let *V* be an instance of the class **String** whose content is
13 “nil”.

14 A conforming processor may let *V* be an empty instance of the class **String**.

- 15 2) If the element is not **nil**, let *V* be the element.

- 16 3) Invoke the method **write** on the receiver with *V* as the argument.

17 b) Return **nil**.

18 **15.2.20.4.12 IO#putc**

19 putc(*obj*)

20 **Visibility:** public

21 **Behavior:**

- 22 a) If the *obj* is not an instance of the class **Integer** or an instance of the class **String**, the
23 behavior is implementation dependent. If the *obj* is an instance of the class **Integer**
24 whose value is smaller than 0 or larger than 255, the behavior is implementation de-
25 pendent.

- 26 b) If the *obj* is an instance of the class **Integer**, create a direct instance of the class **String**
27 *S* whose content is a single character, whose character code is the integer represented
28 by *obj*.

- 1 c) If the *obj* is an instance of the class **String**, create a direct instance of the class **String**
- 2 *S* whose content is the first character of the *obj*.
- 3 d) Invoke the method **write** on the receiver with *S* as the argument.
- 4 e) Return the *obj*.

5 15.2.20.4.13 IO#puts

6 **puts**(**args*)

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the length of the *args* is 0, Invoke the method **write** on the receiver with an argument,
- 10 which is a direct instance of the class **String** whose content is a single character 0x0a.
- 11 b) Otherwise, for each element *E* of the *args* in the indexing order:
 - 12 1) If *E* is **nil**:
 - 13 i) let *S* be an instance of the class **String** whose content is “nil”.
 - 14 A conforming processor may let *S* be an empty instance of the class **String**.
 - 15 ii) Invoke the method **write** on the receiver with *S* as the argument.
 - 16 iii) Invoke the method **write** on the receiver with an argument, which is an
 - 17 instance of the class **String** whose content is a single character 0x0a.
 - 18 2) If *E* is an instance of the class **Array**, for each element *X* of *E* in the indexing
 - 19 order:
 - 20 i) If *X* is the same object as *E*, i.e. if *E* contains itself, invoke the method
 - 21 **write** on the receiver with an instance of the class **String**, whose content is
 - 22 implementation defined.
 - 23 ii) Otherwise, invoke the method **write** on the receiver with *X* as the argument.
 - 24 3) If *E* is an instance of the class **String**:
 - 25 i) Invoke the method **write** on the receiver with *E* as the argument.
 - 26 ii) If the last character of *E* is not 0x0a, invoke the method **write** on the receiver
 - 27 with an argument, which is an instance of the class **String** whose content is
 - 28 a single character 0x0a.
- 29 c) Return **nil**.

1 **15.2.20.4.14 IO#read**

2 `read(length=nil)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the receiver is not readable, raise a direct instance of the class **IOError**.
- 6 b) If the receiver has reached its end:
- 7 1) If the *length* is **nil**, create an empty instance of the class **String** and return that
- 8 instance.
- 9 2) If the *length* is not **nil**, return **nil**.
- 10 c) Otherwise:
- 11 1) If the *length* is **nil**, read characters from the receiver until the receiver reaches its
- 12 end.
- 13 2) If the *length* is an instance of the class **Integer**, let *N* be the value of the *length*.
- 14 Otherwise, the behavior is implementation dependent.
- 15 3) If *N* is smaller than 0, raise a direct instance of the class **ArgumentError**.
- 16 4) Read bytes from the receiver until *N* bytes are read or the receiver reaches its end.
- 17 d) Return a direct instance of the class **String** whose content is the sequence of characters
- 18 read in Step c.

19 **15.2.20.4.15 IO#readchar**

20 `readchar`

21 **Visibility:** public

22 **Behavior:**

- 23 a) If the receiver is not readable, raise a direct instance of the class **IOError**.
- 24 b) If the receiver has reached its end, raise a direct instance of the class **EOFError**.
- 25 c) Otherwise, read a character from the receiver. Return an instance of the class **Object**
- 26 which represents the character.

27 **15.2.20.4.16 IO#readline**

1 **readline**

2 **Visibility:** public

3 **Behavior:**

4 a) If the receiver is not readable, raise a direct instance of the class **IOError**.

5 b) If the receiver has reached its end, raise a direct instance of the class **EOFError**.

6 c) Otherwise, read characters from the receiver until 0x0a is read or the receiver reaches
7 its end.

8 d) Return a direct instance of the class **String** whose content is the sequence of characters
9 read in Step c.

10 **15.2.20.4.17 IO#readlines**

11 **readlines**

12 **Visibility:** public

13 **Behavior:**

14 a) If the receiver is not readable, raise a direct instance of the class **IOError**.

15 b) Create an empty direct instance of the class **Array** *A*.

16 c) If the receiver has reached to its end, return *A*.

17 d) Otherwise, read characters from the receiver until 0x0a is read or the receiver reaches
18 its end.

19 e) Append a direct instance of the class **String** whose content is the sequence of characters
20 read in Step d to *A*.

21 f) Continue processing from Step c.

22 **15.2.20.4.18 IO#sync**

23 **sync**

24 **Visibility:** public

25 **Behavior:**

26 a) If the receiver is closed, raise a direct instance of the class **IOError**.

27 b) If the buffering flag of the receiver is true, return **false**. Otherwise, return **true**.

1 15.2.20.4.19 IO#sync=

2 sync=(*bool*)

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the receiver is closed, raise a direct instance of the class **IOError**.
- 6 b) If the *bool* is a true value, set the buffering flag of the receiver to false. If the *bool* is a
7 false value, set the buffering flag of the receiver to true.
- 8 c) Return the *bool*.

9 15.2.20.4.20 IO#write

10 write(*str*)

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the *str* is not an instance of the class **String**, the behavior is implementation de-
14 pendent.
- 15 b) If the *str* is empty, return an instance of the class **Integer** whose value is 0.
- 16 c) If the receiver is not writable, raise a direct instance of the class **IOError**.
- 17 d) Write all the characters in the *str* to the stream which the receiver represents, preserving
18 their order.
- 19 e) Return an instance of the class **Integer**, whose value is implementation defined.

20 15.2.21 File

21 Instances of the class **File** represent opened files.

22 A conforming processor may raise an instance of the class **SystemCallError** during the execution
23 of the methods of the class **File** if the underlying system reports an error.

24 A **path** of a file is a sequence of characters which represents the location of the file. The correct
25 syntax of paths is implementation defined.

26 An instance of the class **File** has the following property:

27 **path:** The path of the file.

1 **15.2.21.1 Direct superclass**

2 The class IO

3 **15.2.21.2 Singleton methods**

4 **15.2.21.2.1 File.exist?**

5 `File.exist?(path)`

6 **Visibility:** public

7 **Behavior:**

8 a) If the file specified by the *path* exists, return **true**.

9 b) Otherwise, return **false**.

10 **15.2.21.3 Instance methods**

11 **15.2.21.3.1 File#initialize**

12 `initialize(path, mode="r")`

13 **Visibility:** private

14 **Behavior:**

15 a) If the *path* is not an instance of the class **String**, the behavior is implementation
16 dependent.

17 b) If the *mode* is not an instance of the class **String** whose content is a single character
18 “r” or “w”, the behavior is implementation dependent.

19 c) Open the file specified by the *path* in an implementation defined way, and associate it
20 with the receiver.

21 d) Set the path of the receiver to the content of the *path*.

22 e) Set the openness flag and the buffering flag of the receiver to true.

23 f) Set the readability flag and the writability flag of the receiver as follows:

24 1) If the *mode* is an instance of the class **String** whose content is a single character
25 “r”, set the readability flag of the receiver to true and set the writability flag of
26 the receiver to false.

27 2) If the *mode* is an instance of the class **String** whose content is a single character
28 “w”, set the readability flag of the receiver to false and set the writability flag of
29 the receiver to true.

1 g) Return an implementation defined value.

2 **15.2.21.3.2 File#path**

3 path

4 **Visibility:** public

5 **Behavior:** The method returns a direct instance of the class **String** whose content is the
6 path of the receiver.

7 **15.2.22 Exception**

8 Instances of the class **Exception** represent exceptions. The class **Exception** is a superclass of
9 all the other exception classes.

10 Instances of the class **Exception** have the following property.

11 **message:** An object returned by the method **to_s** (see §15.2.22.4.3).

12 When the method **clone** (see §15.3.1.2.8) or the method **dup** (see §15.3.1.2.9) of the class **Kernel**
13 is invoked on an instance of the class **Exception**, the message property shall be copied from the
14 receiver to the resulting value.

15 **15.2.22.1 Direct superclass**

16 The class **Object**

17 **15.2.22.2 Built-in exception classes**

18 This document defines several built-in subclasses of the class **Exception**. Figure 1 shows the
19 list of these subclasses and their class hierarchy. A conforming processor shall raise instances
20 of these built-in subclasses in various erroneous conditions as described in this document. The
21 class hierarchy shown in Figure 1 is used to handle an exception (see §14).

22 **15.2.22.3 Singleton methods**

23 **15.2.22.3.1 Exception.exception**

24 **Exception.exception(*args, &block)**

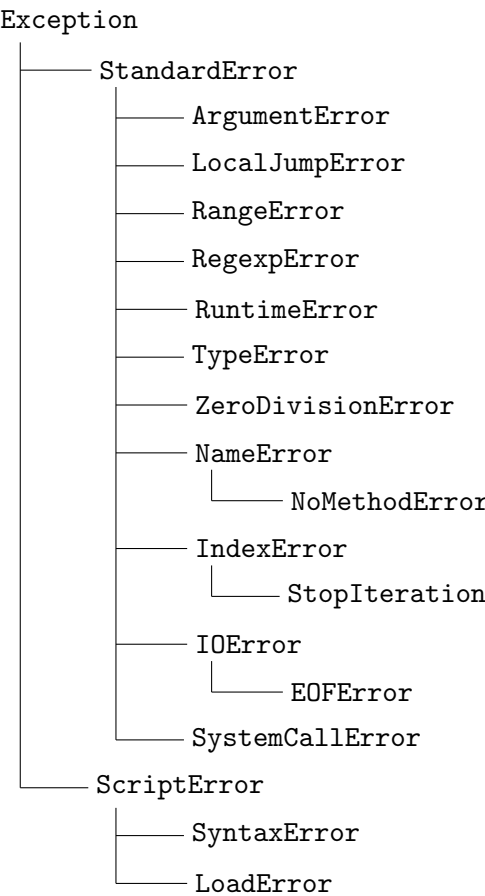
25 **Visibility:** public

26 **Behavior:** Same as the method **new** (see §15.2.3.2.3).

27 **15.2.22.4 Instance methods**

28 **15.2.22.4.1 Exception#exception**

Figure 1 – The exception class hierarchy



`exception(*string)`

Visibility: public

Behavior:

- a) If the length of the *string* is 0, return the receiver.
- b) If the length of the *string* is 1:
 - 1) If the only argument is the same object as the receiver, return the receiver.
 - 2) Otherwise let *M* be the argument.
 - i) Create a direct instance of the class of the receiver. Let *E* be the instance.
 - ii) Set the message property of *E* to *M*.
 - iii) Return *E*.
- c) If the length of the *string* is larger than 1, raise a direct instance of the class **ArgumentError**.

1 **15.2.22.4.2 Exception#message**

2 `message`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Invoke the method `to_s` on the receiver with no arguments.
- 6 b) Return the resulting value of the invocation.

7 **15.2.22.4.3 Exception#to_s**

8 `to_s`

9 **Visibility:** public

10 **Behavior:**

- 11 a) Let M be the message property of the receiver.
- 12 b) If M is `nil`, return an implementation defined value.
- 13 c) If M is not an instance of the class `String`, the behavior is implementation dependent.
- 14 d) Otherwise, return M .

15 **15.2.22.4.4 Exception#initialize**

16 `initialize(message = nil)`

17 **Visibility:** private

18 **Behavior:**

- 19 a) Set the message property of the receiver to the *message*.
- 20 b) Return an implementation defined value.

21 **15.2.23 StandardError**

22 Instances of the class `StandardError` represent standard errors, which can be handled in a
23 *rescue-clause* without a *exception-class-list* (see §11.4.1.4.1).

24 **15.2.23.1 Direct superclass**

25 The class `Exception`

1 15.2.24 ArgumentError

2 Instances of the class `ArgumentError` represent argument errors.

3 15.2.24.1 Direct superclass

4 The class `StandardError`

5 15.2.25 LocalJumpError

6 Instances of the class `LocalJumpError` represent errors which occur while evaluating *blocks* and
7 *jump-expressions*.

8 15.2.25.1 Direct superclass

9 The class `StandardError`

10 15.2.25.2 Instance methods

11 15.2.25.2.1 LocalJumpError#exit_value

12 `exit_value`

13 **Visibility:** public

14 **Behavior:** The method returns the value of the instance variable `@exit_value` of the
15 receiver.

16 15.2.25.2.2 LocalJumpError#reason

17 `reason`

18 **Visibility:** public

19 **Behavior:** The method returns the value of the instance variable `@reason` of the receiver.

20 15.2.26 RangeError

21 Instances of the class `RangeError` represent range errors.

22 15.2.26.1 Direct superclass

23 The class `StandardError`

24 15.2.27 RegexpError

25 Instances of the class `RegexpError` represent regular expression errors.

26 15.2.27.1 Direct superclass

27 The class `StandardError`

1 **15.2.28 RuntimeError**

2 Instances of the class `RuntimeError` represent runtime errors, which are raised by the method
3 `raise` of the class `Kernel` by default (see §15.3.1.1.13).

4 **15.2.28.1 Direct superclass**

5 The class `StandardError`

6 **15.2.29 TypeError**

7 Instances of the class `TypeError` represent type errors.

8 **15.2.29.1 Direct superclass**

9 The class `StandardError`

10 **15.2.30 ZeroDivisionError**

11 Instances of the class `ZeroDivisionError` represent zero division errors.

12 **15.2.30.1 Direct superclass**

13 The class `StandardError`

14 **15.2.31 NameError**

15 Instances of the class `NameError` represent errors which occur while resolving names to values.

16 Instances of the class `NameError` have the following property.

17 **name:** The name a reference to which causes this exception to be raised.

18 When the method `clone` (see §15.3.1.2.8) or the method `dup` (see §15.3.1.2.9) of the class `Kernel`
19 is invoked on an instance of the class `NameError`, the `name` property shall be copied from the
20 receiver to the resulting value.

21 **15.2.31.1 Direct superclass**

22 The class `StandardError`

23 **15.2.31.2 Instance methods**

24 **15.2.31.2.1 NameError#name**

25 **name**

26 **Visibility:** public

27 **Behavior:** The method returns the `name` property of the receiver.

15.2.31.2.2 `NameError#initialize`

```
initialize( message=nil, name=nil )
```

Visibility: public

Behavior:

- a) Set the name property of the receiver to the *name*.
- b) Invoke the method `initialize` defined in the class `Exception`, which is a superclass of the class `NameError`, as if *super-with-argument* were evaluated with the *message* as the *argument* of the *super-with-argument*.
- c) Return an implementation defined value.

15.2.32 `NoMethodError`

Instances of the class `NoMethodError` represent errors which occur while invoking methods which do not exist or which cannot be invoked.

Instances of the class `NoMethodError` have properties called ***name*** (see §15.2.31) and ***arguments***. The values of these properties are set in the method `initialize` (see §15.2.32.2.2).

When the method `clone` (see §15.3.1.2.8) or the method `dup` (see §15.3.1.2.9) of the class `Kernel` is invoked on an instance of the class `NoMethodError`, those properties shall be copied from the receiver to the resulting value.

15.2.32.1 `Direct superclass`

The class `NameError`

15.2.32.2 `Instance methods`

15.2.32.2.1 `NoMethodError#args`

```
args
```

Visibility: public

Behavior: The method returns the value of the arguments property of the receiver.

15.2.32.2.2 `NoMethodError#initialize`

```
initialize( message=nil, name=nil, args=nil )
```

Visibility: public

1 **Behavior:**

- 2 a) Set the arguments property of the receiver to the *args*.
- 3 b) Perform all the steps of the method `initialize` described in §15.2.31.2.2.
- 4 c) Return an implementation defined value.

5 **15.2.33 IndexError**

6 Instances of the class `IndexError` represent index errors.

7 **15.2.33.1 Direct superclass**

8 The class `StandardError`

9 **15.2.34 StopIteration**

10 Instances of the class `StopIteration` represent exceptions, which are raised to terminate the
11 method `loop` of the class `Kernel` (see §15.3.1.1.8).

12 **15.2.34.1 Direct superclass**

13 The class `IndexError`

14 **15.2.35 IOError**

15 Instances of the class `IOError` represent input/output errors.

16 **15.2.35.1 Direct superclass**

17 The class `StandardError`

18 **15.2.36 EOFError**

19 Instances of the class `EOFError` represent errors which occur when a stream has reached its end.

20 **15.2.36.1 Direct superclass**

21 The class `IOError`

22 **15.2.37 SystemCallError**

23 Instances of the class `SystemCallError` represent errors which occur while invoking the instance
24 methods of the class `IO`.

25 **15.2.37.1 Direct superclass**

26 The class `StandardError`

27 **15.2.38 ScriptError**

28 Instances of the class `ScriptError` represent programming errors.

1 **15.2.38.1 Direct superclass**

2 The class `Exception`

3 **15.2.39 SyntaxError**

4 Instances of the class `SyntaxError` represent syntax errors.

5 **15.2.39.1 Direct superclass**

6 The class `ScriptError`

7 **15.2.40 LoadError**

8 Instances of the class `LoadError` represent errors which occur while loading external programs
9 (see §15.3.1.1.14).

10 **15.2.40.1 Direct superclass**

11 The class `ScriptError`

12 **15.3 Built-in modules**

13 **15.3.1 Kernel**

14 The module `Kernel` is included in the class `Object`. Unless overridden, instance methods defined
15 in the module `Kernel` can be invoked on any instance of the class `Object`.

16 **15.3.1.1 Singleton methods**

17 **15.3.1.1.1 Kernel.‘**

18 `Kernel.‘ (string)`

19 The method `‘` is invoked in the form described in §8.5.5.2.6.

20 **Visibility:** public

21 **Behavior:** The method `‘` executes an external command corresponding to the *string*. The
22 external command executed by the method is implementation defined.

23 When the method is invoked, take the following steps:

- 24 a) If the *string* is not an instance of the class `String`, the behavior is implementation
25 dependent.
- 26 b) Execute the command which corresponds to the content of the *string*. Let *R* be the
27 output of the command.
- 28 c) Create a direct instance of the class `String` whose content is *R*, and return the instance.

1 **15.3.1.1.2 Kernel.block_given?**

2 `Kernel.block_given?`

3 **Visibility:** public

4 **Behavior:**

5 a) If the top of `[[block]]` is block-not-given, return **false**.

6 b) Otherwise, return **true**.

7 **15.3.1.1.3 Kernel.eval**

8 `Kernel.eval(string)`

9 **Visibility:** public

10 **Behavior:**

11 a) If the *string* is not an instance of the class **String**, the behavior is implementation
12 dependent.

13 b) Parse the content of the *string* as a *program* (see §10.1). If it fails, raise a direct instance
14 of the class **SyntaxError**.

15 c) Evaluate the *program*. Let *V* be the resulting value of the evaluation.

16 d) Return *V*.

17 In Step c, the *string* is evaluated under the new local variable scope in which references to
18 *local-variable-identifiers* are resolved in the same way as in scopes created by *blocks* (see
19 §9.1.1).

20 **15.3.1.1.4 Kernel.global_variables**

21 `Kernel.global_variables`

22 **Visibility:** public

23 **Behavior:** The method returns a new direct instance of the class **Array** which consists
24 of names of all the global variables. These names are represented by instances of either
25 the class **String** or the class **Symbol**. Which of those classes is chosen is implementation
26 defined.

27 **15.3.1.1.5 Kernel.iterator?**

1 `Kernel.iterator?`

2 **Visibility:** public

3 **Behavior:** Same as the method `Kernel.block_given?` (see §15.3.1.1.2).

4 **15.3.1.1.6 Kernel.lambda**

5 `Kernel.lambda(&block)`

6 **Visibility:** public

7 **Behavior:** The method creates an instance of the class `Proc` as `Proc.new` does (see §15.2.17.2.1).
8 However, the way in which the *block* is evaluated differs from the one described in §11.2.2
9 except when the *block* is called by a *yield-expression*.

10 The differences are as follows.

11 a) Before Step d of §11.2.2, the number of arguments is checked as follows:

12 1) Let A be the list of arguments passed to the block. Let N_a be the length of A .

13 2) If the *block-formal-argument-list* is of the form *left-hand-side*, and if N_a is not 1,
14 the behavior is implementation dependent.

15 3) If the *block-formal-argument-list* is of the form *multiple-left-hand-side*:

16 i) If the *multiple-left-hand-side* is not of the form *grouped-left-hand-side* or *packing-*
17 *left-hand-side*:

18 I) Let N_p be the number of *multiple-left-hand-side-items* of the *multiple-*
19 *left-hand-side*.

20 II) If $N_a < N_p$, raise a direct instance of the class `ArgumentError`.

21 III) If a *packing-left-hand-side* does not occur, and if $N_a > N_p$, raise a direct
22 instance of the class `ArgumentError`.

23 ii) If the *multiple-left-hand-side* is of the form *grouped-left-hand-side*, and if N_a
24 is not 1, the behavior is implementation dependent.

25 b) In Step e of §11.2.2, when the evaluation of the block associated with a `lambda` invo-
26 cation is terminated by a *return-expression* or *break-expression*, the execution context
27 is restored to E_o (i.e. the saved execution context), and the evaluation of the `lambda`
28 invocation is terminated.

29 The value of the `lambda` invocation is determined as follows:

30 1) If the *jump-argument* of the *return-expression* or the *break-expression* occurs, the
31 value of the `lambda` invocation is the value of the *jump-argument*.

2) Otherwise, the value of the lambda invocation is `nil`.

15.3.1.1.7 `Kernel.local_variables`

`Kernel.local_variables`

Visibility: public

Behavior: The method returns a new direct instance of the class `Array` which contains all the names of local variable bindings which meet the following conditions.

- The name of the binding is of the form *local-variable-identifier*.
- The binding can be resolved in the scope of local variables which includes the point of invocations of this method by the process described in §9.1.1.

In the instance of the class `Array` returned by the method, names of the local variables are represented by instances of either the class `String` or the class `Symbol`. Which of those classes is chosen is implementation defined.

15.3.1.1.8 `Kernel.loop`

`Kernel.loop(&block)`

Visibility: public

Behavior:

- If the *block* is not given, the behavior is implementation dependent.
- Otherwise, repeat calling the *block*.
- If a direct instance of the class `StopIteration` is raised and not handled in Step b, handle the exception and return `nil`.

15.3.1.1.9 `Kernel.method_missing`

`Kernel.method_missing(symbol, *args)`

Visibility: public

Behavior:

- If the *symbol* is not an instance of the class `Symbol`, raise a direct instance of the class `ArgumentError`.

- 1 b) Otherwise, raise a direct instance of the class `NoMethodError` which has the *symbol*
2 as its name property and the *args* as its arguments property. A conforming processor
3 may raise a direct instance of the class `NameError` which has the *symbol* as its name
4 property instead of `NoMethodError` if the method is invoked in Step e of §13.3.3 during
5 an evaluation of a *local-variable-identifier* as a method invocation.

6 15.3.1.1.10 Kernel.p

7 `Kernel.p(*args)`

8 **Visibility:** public

9 **Behavior:**

- 10 a) For each element *E* of the *args*, in the indexing order, take the following steps:
- 11 1) Invoke the method `inspect` on *E* with no arguments and let *X* be the resulting
12 value of this invocation.
- 13 2) If *X* is not an instance of the class `String`, the behavior is implementation depen-
14 dent.
- 15 3) Invoke the method `write` on `Object::STDOUT` with *X* as the argument.
- 16 4) Invoke the method `write` on `Object::STDOUT` with an argument, which is a direct
17 instance of the class `String` whose content is a single character 0x0a.
- 18 b) Return `nil`. A conforming processor may return the *args* instead of `nil`.

19 15.3.1.1.11 Kernel.print

20 `Kernel.print(*args)`

21 **Visibility:** public

22 **Behavior:** The method behaves as if the method `print` of the class `IO` (see §15.2.20.4.11)
23 were invoked on `Object::STDOUT` with the same arguments.

24 15.3.1.1.12 Kernel.puts

25 `Kernel.puts(*args)`

26 **Visibility:** public

27 **Behavior:** The method behaves as if the method `puts` of the class `IO` (see §15.2.20.4.13)
28 were invoked on `Object::STDOUT` with the same arguments.

1 15.3.1.1.13 Kernel.raise

2 Kernel.raise(*args)

3 **Visibility:** public

4 **Behavior:**

- 5 a) If the length of the *args* is larger than 2, the behavior is implementation dependent.
- 6 b) If the length of the *args* is 0:
- 7 1) If the location of the method invocation is within an *operator-expression*₂ of an
- 8 *assignment-with-rescue-modifier*, a *fallback-statement-of-rescue-modifier-statement*,
- 9 or a *rescue-clause*, let *E* be the current exception (see §14.2).
- 10 2) Otherwise, invoke the method **new** on the class **RuntimeError** with no argument.
- 11 Let *E* be the resulting value.
- 12 c) If the length of the *args* is 1, let *A* be the only argument.
- 13 1) If *A* is an instance of the class **String**, invoke the method **new** on the class
- 14 **RuntimeError** with *A* as the only argument. Let *E* be the resulting instance.
- 15 2) Otherwise, invoke the method **exception** on *A*. Let *E* be the resulting value.
- 16 3) If *E* is not an instance of the class **Exception**, raise a direct instance of the class
- 17 **TypeError**.
- 18 d) If the length of the *args* is 2, let *F* and *S* be the first and the second argument,
- 19 respectively.
- 20 1) Invoke **exception** on *F* with *S* as the only argument. Let *E* be the resulting
- 21 value.
- 22 2) If *E* is not an instance of the class **Exception**, raise a direct instance of the class
- 23 **TypeError**.
- 24 e) Raise *E*.

25 15.3.1.1.14 Kernel.require

26 Kernel.require(string)

27 **Visibility:** public

28 **Behavior:** The method **require** evaluates the external program *P* corresponding to the

29 *string*. The way in which *P* is determined from the *string* is implementation defined.

30 When the method is invoked, take the following steps:

- 1 a) If the *string* is not an instance of the class **String**, the behavior is implementation
2 dependent.
- 3 b) Search for the external program *P* corresponding to the *string*.
- 4 c) If the program does not exist, raise a direct instance of the class **LoadError**.
- 5 d) If *P* cannot be derived from the *program* (§10.1), raise a direct instance of the class
6 **SyntaxError**.
- 7 e) Change the state of the execution context temporarily for the evaluation of *P* as follows:
 - 8 1) `[[self]]` contains only one object which is the object at the bottom of `[[self]]` in the
9 current execution context.
 - 10 2) `[[class-module-list]]` contains only one list whose only element is the class **Object**.
 - 11 3) `[[default-visibility]]` contains only one visibility, which is the private visibility.
 - 12 4) All the other attributes of the execution context are empty logical stacks.
- 13 f) Evaluate *P* under the execution context set up in Step e.
- 14 g) Restore the state of the execution context as it is just before Step e, even when an
15 exception is raised and not handled during the evaluation of *P*.

16 Note that the evaluation of *P* affects the restored execution context if it changes the
17 attributes of objects in the original execution context.
- 18 h) Unless an exception is raised and not handled in Step f, return **true**.

19 15.3.1.2 Instance methods

20 15.3.1.2.1 Kernel#==

21 ==(*other*)

22 **Visibility:** public

23 **Behavior:**

- 24 a) If the receiver and the *other* is the same object, return **true**.
- 25 b) Otherwise, return **false**.

26 15.3.1.2.2 Kernel#===

27 ===(*other*)

1 **Visibility:** public

2 **Behavior:**

3 a) If the receiver and the *other* is the same object, return **true**.

4 b) Otherwise, invoke the method **==** on the receiver with the *other* as the only argument.
5 Let *V* be the resulting value.

6 c) If *V* is a true value, return **true**. Otherwise, return **false**.

7 **15.3.1.2.3 Kernel#__id__**

8 __id__

9 **Visibility:** public

10 **Behavior:** Same as the method **object_id** (see §15.3.1.2.33).

11 **15.3.1.2.4 Kernel#__send__**

12 __send__(*symbol*, **args*, &*block*)

13 **Visibility:** public

14 **Behavior:** Same as the method **send** (see §15.3.1.2.44).

15 **15.3.1.2.5 Kernel#‘**

16 ‘ (*string*)

17 The method **‘** is invoked in the form described in §8.5.5.2.6.

18 **Visibility:** private

19 **Behavior:** Same as the method **Kernel.‘** (see §15.3.1.1.1).

20 **15.3.1.2.6 Kernel#block_given?**

21 block_given?

22 **Visibility:** private

23 **Behavior:** Same as the method **Kernel.block_given?** (see §15.3.1.1.2).

1 **15.3.1.2.7 Kernel#class**

2 **class**

3 **Visibility:** public

4 **Behavior:** The method returns the class of the receiver.

5 **15.3.1.2.8 Kernel#clone**

6 **clone**

7 **Visibility:** public

8 **Behavior:**

- 9 a) If the receiver is an instance of one of the following classes: **NilClass**, **TrueClass**,
10 **FalseClass**, **Integer**, **Float**, or **Symbol**, then raise a direct instance of the class
11 **TypeError**.
- 12 b) Create a direct instance of the class of the receiver which has no bindings of instance
13 variables. Let O be the newly created instance.
- 14 c) For each binding B of the instance variables of the receiver, create a variable binding
15 with the same name and value as B in the set of bindings of instance variables of O .
- 16 d) If the receiver is associated with an eigenclass, let E_o be the eigenclass, and take the
17 following steps:
- 18 1) Create an eigenclass whose direct superclass is the direct superclass of E_o . Let E_n
19 be the eigenclass.
- 20 2) For each binding B_{v1} of the constants of E_o , create a variable binding with the
21 same name and value as B_{v1} in the set of bindings of constants of E_n .
- 22 3) For each binding B_{v2} of the class variables of E_o , create a variable binding with
23 the same name and value as B_{v2} in the set of bindings of class variables of E_n .
- 24 4) For each binding B_m of the instance methods of E_o , create a method binding with
25 the same name and value as B_m in the set of bindings of instance methods of E_n .
- 26 5) Associate O with E_n .
- 27 e) Invoke the method **initialize_copy** on O with the receiver as the argument.
- 28 f) Return O .

29 **15.3.1.2.9 Kernel#dup**

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

`dup`

Visibility: public

Behavior:

- a) If the receiver is an instance of one of the following classes: `NilClass`, `TrueClass`, `FalseClass`, `Integer`, `Float`, or `Symbol`, then raise a direct instance of the class `TypeError`.
- b) Create a direct instance of the class of the receiver which has no bindings of instance variables. Let *O* be the newly created instance.
- c) For each binding *B* of the instance variables of the receiver, create a variable binding with the same name and value as *B* in the set of bindings of instance variables of *O*.
- d) Invoke the method `initialize_copy` on *O* with the receiver as the argument.
- e) Return *O*.

15.3.1.2.10 Kernel#eql?

`eql?(other)`

Visibility: public

Behavior: Same as the method `==` (see §15.3.1.2.1).

15.3.1.2.11 Kernel#equal?

`equal?(other)`

Visibility: public

Behavior: Same as the method `==` (see §15.3.1.2.1).

15.3.1.2.12 Kernel#eval

`eval(string)`

Visibility: private

Behavior: Same as the method `Kernel.eval` (see §15.3.1.1.3).

15.3.1.2.13 Kernel#extend

1 `extend(*module_list)`

2 **Visibility:** public

3 **Behavior:** Let R be the receiver of the method.

4 a) If the length of the *module_list* is 0, raise a direct instance of the class **ArgumentError**.

5 b) For each element A of the *module_list*, take the following steps:

6 1) If A is not an instance of the class **Module**, raise a direct instance of the class
7 **TypeError**.

8 2) If A is an instance of the class **Class**, raise a direct instance of the class **TypeError**.

9 3) Invoke the method `extend_object` on A with R as the only argument.

10 4) Invoke the method `extended` on A with R as the only argument.

11 c) Return R .

12 **15.3.1.2.14 Kernel#global_variables**

13 `global_variables`

14 **Visibility:** private

15 **Behavior:** Same as the method `Kernel.global_variables` (see §15.3.1.1.4).

16 **15.3.1.2.15 Kernel#hash**

17 `hash`

18 **Visibility:** public

19 **Behavior:** The method returns an instance of the class **Integer**. When invoked on the
20 same object, the method shall always return an instance of the class **Integer** whose values
21 is same.

22 When a conforming processor overrides the method `eq1?` (see §15.3.1.2.10), it shall override
23 the method `hash` in the same class or module in which the method `eq1?` is overridden in
24 such a way that, if an invocation of the method `eq1?` on an object with an argument returns
25 a true value, invocations of the method `hash` on the object and the argument return the
26 instances of the class **Integer** with the same value.

27 **15.3.1.2.16 Kernel#initialize_copy**

1 `initialize_copy(original)`

2 **Visibility:** private

3 **Behavior:** The method `initialize_copy` is invoked when an object is created by the
4 method `clone` (see §15.3.1.2.8) or the method `dup` (see §15.3.1.2.9).

5 When the method is invoked, take the following steps:

6 a) If the classes of the receiver and the *original* are not the same class, raise a direct
7 instance of the class `TypeError`.

8 b) Return an implementation defined value.

9 **15.3.1.2.17 Kernel#inspect**

10 `inspect`

11 **Visibility:** public

12 **Behavior:** The method returns an instance of the class `String`, the content of which
13 represents the state of the receiver. The content of the resulting instance of the class
14 `String` is implementation defined.

15 **15.3.1.2.18 Kernel#instance_eval**

16 `instance_eval(string = nil, &block)`

17 **Visibility:** public

18 **Behavior:**

19 a) If the receiver is an instance of one of the implementation defined set of classes as
20 described in Step a of §13.4.2, or if the receiver is one of `nil`, `true`, or `false`, then the
21 behavior is implementation dependent.

22 b) If the receiver is not associated with an eigenclass, create a new eigenclass. Let *M* be
23 the newly created eigenclass.

24 c) If the receiver is associated with an eigenclass, let *M* be that eigenclass.

25 d) Take Step b through the last step of the method `class_eval` of the class `Module` (see
26 §15.2.2.3.15).

27 **15.3.1.2.19 Kernel#instance_of?**

1 `instance_of?(module)`

2 **Visibility:** public

3 **Behavior:** Let *C* be the class of the receiver.

4 a) If the *module* is not an instance of the class **Class** or the class **Module**, raise a direct
5 instance of the class **TypeError**.

6 b) If the *module* and *C* are the same object, return **true**.

7 c) Otherwise, return **false**.

8 **15.3.1.2.20 Kernel#instance_variable_defined?**

9 `instance_variable_defined?(symbol)`

10 **Visibility:** public

11 **Behavior:**

12 a) Let *N* be the name designated by the *symbol*.

13 b) If *N* is not of the form *instance-variable-identifier*, raise a direct instance of the class
14 **NameError** which has the *symbol* as its name property.

15 c) If a binding of an instance variable with name *N* exists in the set of bindings of instance
16 variables of the receiver, return **true**.

17 d) Otherwise, return **false**.

18 **15.3.1.2.21 Kernel#instance_variable_get**

19 `instance_variable_get(symbol)`

20 **Visibility:** public

21 **Behavior:**

22 a) Let *N* be the name designated by the *symbol*.

23 b) If *N* is not of the form *instance-variable-identifier*, raise a direct instance of the class
24 **NameError** which has the *symbol* as its name property.

25 c) If a binding of an instance variable with name *N* exists in the set of bindings of instance
26 variables of the receiver, return the value of the binding.

27 d) Otherwise, return **nil**.

1 **15.3.1.2.22 Kernel#instance_variable_set**

2 `instance_variable_set(symbol, obj)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Let *N* be the name designated by the *symbol*.
- 6 b) If *N* is not of the form *instance-variable-identifier*, raise a direct instance of the class
7 **NameError** which has the *symbol* as its name property.
- 8 c) If a binding of an instance variable with name *N* exists in the set of bindings of instance
9 variables of the receiver, replace the value of the binding with the *obj*.
- 10 d) Otherwise, create a variable binding with name *N* and value *obj* in the set of bindings
11 of instance variables of the receiver.
- 12 e) Return the *obj*.

13 **15.3.1.2.23 Kernel#instance_variables**

14 `instance_variables`

15 **Visibility:** public

16 **Behavior:** The method returns a direct instance of the class **Array** which consists of names
17 of all the instance variables of the receiver. These names are represented by instances of
18 either the class **String** or the class **Symbol**. Which of those classes is chosen is implemen-
19 tation defined.

20 **15.3.1.2.24 Kernel#is_a?**

21 `is_a?(module)`

22 **Visibility:** public

23 **Behavior:**

- 24 a) If the *module* is not an instance of the class **Class** or the class **Module**, raise a direct
25 instance of the class **TypeError**.
- 26 b) Let *C* be the class of the receiver.
- 27 c) If the *module* is an instance of the class **Class** and one of the following conditions
28 holds, return **true**.

- 1 • The *module* and *C* are the same object.
- 2 • The *module* is a superclass of *C*.
- 3 • The *module* and the eigenclass of the receiver are the same object.
- 4 d) If the *module* is an instance of the class `Module` and is included in *C* or one of the
- 5 superclasses of *C*, return `true`.
- 6 e) Otherwise, return `false`.

7 **15.3.1.2.25 Kernel#iterator?**

8 `iterator?`

9 **Visibility:** private

10 **Behavior:** Same as the method `Kernel.iterator?` (see §15.3.1.1.5).

11 **15.3.1.2.26 Kernel#kind_of?**

12 `kind_of?(module)`

13 **Visibility:** public

14 **Behavior:** Same as the method `is_a?` (see §15.3.1.2.24).

15 **15.3.1.2.27 Kernel#lambda**

16 `lambda(&block)`

17 **Visibility:** private

18 **Behavior:** Same as the method `Kernel.lambda` (see §15.3.1.1.6).

19 **15.3.1.2.28 Kernel#local_variables**

20 `local_variables`

21 **Visibility:** private

22 **Behavior:** Same as the method `Kernel.local_variables` (see §15.3.1.1.7).

23 **15.3.1.2.29 Kernel#loop**

1 `loop(&block)`

2 **Visibility:** private

3 **Behavior:** Same as the method `Kernel.loop` (see §15.3.1.1.8).

4 **15.3.1.2.30 Kernel#method_missing**

5 `method_missing(symbol, *args)`

6 **Visibility:** private

7 **Behavior:** Same as the method `Kernel.method_missing` (see §15.3.1.1.9).

8 **15.3.1.2.31 Kernel#methods**

9 `methods(all=true)`

10 **Visibility:** public

11 **Behavior:** Let *C* be the class of the receiver.

12 a) If the *all* is a true value, the method behaves as if the method `instance_methods` were
13 invoked on *C* with no arguments (see §15.2.2.3.33).

14 b) If the *all* is a false value, the method behaves as if the method `singleton_methods`
15 were invoked on the receiver with `false` as the only argument (see §15.3.1.2.45).

16 **15.3.1.2.32 Kernel#nil?**

17 `nil?`

18 **Visibility:** public

19 **Behavior:**

20 a) If the receiver is `nil`, return `true`.

21 b) Otherwise, return `false`.

22 **15.3.1.2.33 Kernel#object_id**

1 `object_id`

2 **Visibility:** public

3 **Behavior:** The method returns an instance of the class `Integer` with the same value
4 whenever it is invoked on the same object. When invoked on two distinct objects, the
5 method returns an instance of the class `Integer` with different value for each invocation.

6 **15.3.1.2.34 Kernel#p**

7 `p(*args)`

8 **Visibility:** private

9 **Behavior:** Same as the method `Kernel.p` (see §15.3.1.1.10).

10 **15.3.1.2.35 Kernel#print**

11 `print(*args)`

12 **Visibility:** private

13 **Behavior:** Same as the method `Kernel.print` (see §15.3.1.1.11).

14 **15.3.1.2.36 Kernel#private_methods**

15 `private_methods(all=true)`

16 **Visibility:** public

17 **Behavior:**

- 18 a) Create an empty direct instance of the class `Array` *A*.
- 19 b) If the receiver is associated with an eigenclass, let *C* be the eigenclass.
- 20 c) Let *I* be the set of bindings of instance methods of *C*.

21 For each binding *B* of *I*, let *N* and *V* be the name and the value of *B* respectively, and
22 take the following steps:

- 23 1) If *V* is undef, or the visibility of *V* is not private, skip the next two steps.
- 24 2) Let *S* be either a direct instance of the class `String` whose content is *N* or a
25 direct instance of the class `Symbol` whose name is *N*. Which of these classes of
26 instance is chosen as the value of *S* is implementation defined.

- 1 3) Unless A contains the element of the same name (if S is an instance of the class
- 2 `Symbol`) or the same content (if S is an instance of the class `String`) as S , append
- 3 S to A .
- 4 d) For each module M in included module list of C , take Step c, assuming that C in that
- 5 step to be M .
- 6 e) Replace C with the class of the receiver, and take Step c.
- 7 f) If the *all* is a true value:
 - 8 1) Take Step d.
 - 9 2) Replace C with the direct superclass of current C .
 - 10 3) If C is not `nil`, take Step c, and then, repeat from Step f-1.
- 11 g) Return A .

12 **15.3.1.2.37 Kernel#protected_methods**

13 `protected_methods(all=true)`

14 **Visibility:** public

15 **Behavior:** Same as the method `private_methods` (see §15.3.1.2.36), except that the method

16 returns a direct instance of the class `Array` which contains names of protected methods.

17 **15.3.1.2.38 Kernel#public_methods**

18 `public_methods`

19 **Visibility:** public

20 **Behavior:** Same as the method `private_methods` (see §15.3.1.2.36), except that the method

21 returns a direct instance of the class `Array` which contains names of public methods.

22 **15.3.1.2.39 Kernel#puts**

23 `puts(*args)`

24 **Visibility:** private

25 **Behavior:** Same as the method `Kernel.puts` (see §15.3.1.1.12).

26 **15.3.1.2.40 Kernel#raise**

1 `raise(*args)`

2 **Visibility:** private

3 **Behavior:** Same as the method `Kernel.raise` (see §15.3.1.1.13).

4 **15.3.1.2.41 Kernel#remove_instance_variable**

5 `remove_instance_variable(symbol)`

6 **Visibility:** private

7 **Behavior:**

- 8 a) Let N be the name designated by the *symbol*.
- 9 b) If N is not of the form *instance-variable-identifier*, raise a direct instance of the class
10 **NameError** which has the *symbol* as its name property.
- 11 c) If a binding of an instance variable with name N exists in the set of bindings of instance
12 variables of the receiver, let V be the value of the binding.
- 13 1) Remove the binding from the set of bindings of instance variables of the receiver.
- 14 2) Return V .
- 15 d) Otherwise, raise a direct instance of the class **NameError** which has the *symbol* as its
16 name property.

17 **15.3.1.2.42 Kernel#require**

18 `require(*args)`

19 **Visibility:** private

20 **Behavior:** Same as the method `Kernel.require` (see §15.3.1.1.14).

21 **15.3.1.2.43 Kernel#respond_to?**

22 `respond_to?(symbol, include_private=false)`

23 **Visibility:** public

24 **Behavior:**

- 1 a) Let N be the name designated by the *symbol*.
- 2 b) Search for a binding of an instance method named N starting from the receiver of the
3 method as described in §13.3.4.
- 4 c) If a binding is found, let V be the value of the binding.
 - 5 1) If V is undef, return **false**.
 - 6 2) If the visibility of V is private:
 - 7 i) If the *include_private* is a true value, return **true**.
 - 8 ii) Otherwise, return **false**.
 - 9 3) Otherwise, return **true**.
- 10 d) Otherwise, return **false**.

11 15.3.1.2.44 Kernel#send

12 **send**(*symbol*, **args*, &*block*)

13 **Visibility:** public

14 **Behavior:**

- 15 a) Let N be the name designated by the *symbol*.
- 16 b) Invoke the method named N on the receiver with the *args* as arguments and the *block*
17 as the block, if any.
- 18 c) Return the resulting value of the invocation.

19 15.3.1.2.45 Kernel#singleton_methods

20 **singleton_methods**(*all*=true)

21 **Visibility:** public

22 **Behavior:** Let E be the eigenclass of the receiver.

- 23 a) Create an empty direct instance of the class **Array** A .
- 24 b) Let I be the set of bindings of instance methods of E .

25 For each binding B of I , let N and V be the name and the value of B respectively, and
26 take the following steps:

- 1) If V is undef, or the visibility of V is private, skip the next two steps.
- 2) Let S be either a direct instance of the class **String** whose content is N or a direct instance of the class **Symbol** whose name is N . Which of these classes of instance is chosen as the value of S is implementation defined.
- 3) Unless A contains the element of the same name (if S is an instance of the class **Symbol**) or the same content (if S is an instance of the class **String**), append S to A .
- c) If the *all* is a true value, for each module M in included module list of E , take Step b, assuming that E in that step to be M .
- d) Return A .

15.3.1.2.46 Kernel#to_s

to_s

Visibility: public

Behavior: The method returns an instance of the class **String**, the content of which is the string representation of the receiver. The content of the resulting instance of the class **String** is implementation defined.

15.3.2 Enumerable

The module **Enumerable** provides methods which iterates over the elements of the object using the method **each**.

In the following description of the methods of the module **Enumerable**, an *element* of the receiver means one of the values which is yielded by the method **each**.

15.3.2.1 Instance methods

15.3.2.1.1 Enumerable#all?

all?(&block)

Visibility: public

Behavior:

- a) Invoke the method **each** on the receiver.
 - b) For each element X which the method **each** yields:
 - 1) If the *block* is given, call the *block* with X as the argument.
- If this call results in a false value, return **false**.

- 1 2) If the *block* is not given, and *X* is a false value, return **false**.
- 2 c) Return **true**.

3 **15.3.2.1.2 Enumerable#any?**

4 **any?(&block)**

5 **Visibility:** public

6 **Behavior:**

- 7 a) Invoke the method **each** on the receiver.
- 8 b) For each element *X* which **each** yields:
- 9 1) If the *block* is given, call the *block* with *X* as the argument.
- 10 If this call results in a true value, return **true**.
- 11 2) If the *block* is not given, and *X* is a true value, return **true**.
- 12 c) Return **false**.

13 **15.3.2.1.3 Enumerable#collect**

14 **collect(&block)**

15 **Visibility:** public

16 **Behavior:**

- 17 a) If the *block* is not given, the behavior is implementation dependent.
- 18 b) Create an empty direct instance of the class **Array** *A*.
- 19 c) Invoke the method **each** on the receiver.
- 20 d) For each element *X* which **each** yields, call the *block* with *X* as the argument and
- 21 append the resulting value to *A*.
- 22 e) Return *A*.

23 **15.3.2.1.4 Enumerable#detect**

1 `detect(ifnone=nil, &block)`

2 **Visibility:** public

3 **Behavior:**

- 4 a) If the *block* is not given, the behavior is implementation dependent.
- 5 b) Invoke the method **each** on the receiver.
- 6 c) For each element *X* which **each** yields, call the *block* with *X* as the argument. If this
- 7 call results in a true value, return *X*.
- 8 d) Return the *ifnone*.

9 **15.3.2.1.5 Enumerable#each_with_index**

10 `each_with_index(&block)`

11 **Visibility:** public

12 **Behavior:**

- 13 a) If the *block* is not given, the behavior is implementation dependent.
- 14 b) Let *i* be 0.
- 15 c) Invoke the method **each** on the receiver.
- 16 d) For each element *X* which **each** yields:
- 17 1) Call the *block* with *X* and *i* as the arguments.
- 18 2) Increase *i* by 1.
- 19 e) Return the receiver.

20 **15.3.2.1.6 Enumerable#entries**

21 `entries`

22 **Visibility:** public

23 **Behavior:**

- 24 a) Create an empty direct instance of the class **Array** *A*.

- 1 b) Invoke the method **each** on the receiver.
- 2 c) For each element *X* which **each** yields, append *X* to *A*.
- 3 d) Return *A*.

4 **15.3.2.1.7 Enumerable#find**

5 **find**(*ifnone=nil*, &*block*)

6 **Visibility:** public

7 **Behavior:** Same as the method **detect** (see §15.3.2.1.4).

8 **15.3.2.1.8 Enumerable#find_all**

9 **find_all**(&*block*)

10 **Visibility:** public

11 **Behavior:**

- 12 a) If the *block* is not given, the behavior is implementation dependent.
- 13 b) Create an empty direct instance of the class **Array** *A*.
- 14 c) Invoke the method **each** on the receiver.
- 15 d) For each element *X* which **each** yields, call the *block* with *X* as the argument. If this
- 16 call results in a true value, append the element to *A*.
- 17 e) Return *A*.

18 **15.3.2.1.9 Enumerable#grep**

19 **grep**(*pattern*, &*block*)

20 **Visibility:** public

21 **Behavior:**

- 22 a) Create an empty direct instance of the class **Array** *A*.
- 23 b) Invoke the method **each** on the receiver.
- 24 c) For each element *X* which **each** yields, invoke the method **===** on the *pattern* with *X*
- 25 as the argument.
- 26 If this invocation results in a true value:

- 1 1) If the *block* is given, call the *block* with *X* as the argument and append the resulting
2 value to *A*.
- 3 2) Otherwise, append *X* to *A*.
- 4 d) Return *A*.

5 15.3.2.1.10 Enumerable#include?

6 include?(*obj*)

7 **Visibility:** public

8 **Behavior:**

- 9 a) Invoke the method **each** on the receiver.
- 10 b) For each element *X* which **each** yields, invoke the method **==** on *X* with the *obj* as the
11 argument. If this invocation results in a true value, return **true**.
- 12 c) Return **false**.

13 15.3.2.1.11 Enumerable#inject

14 inject(**args*, &*block*)

15 **Visibility:** public

16 **Behavior:**

- 17 a) If the *block* is not given, the behavior is implementation dependent.
- 18 b) If the length of the *args* is 2, the behavior is implementation dependent. If the length
19 of the *args* is smaller than 0 or larger than 2, raise a direct instance of the class
20 **ArgumentError**.
- 21 c) Invoke the method **each** on the receiver. If the method **each** does not yield any element,
22 return **nil**.
- 23 d) For each element *X* which **each** yields:
 - 24 1) If *X* is the first element, and the length of the *args* is 0, let *V* be *X*.
 - 25 2) If *X* is the first element, and the length of the *args* is 1, call the *block* with two
26 arguments, which are the only element of the *args* and *X*. Let *V* be the resulting
27 value of this call.
 - 28 3) If *X* is not the first element, call the *block* with *V* and *X* as the arguments. Let
29 new *V* be the resulting value of this call.

1 e) Return *V*.

2 **15.3.2.1.12 Enumerable#map**

3 map(&*block*)

4 **Visibility:** public

5 **Behavior:** Same as the method `collect` (see §15.3.2.1.3).

6 **15.3.2.1.13 Enumerable#max**

7 max(&*block*)

8 **Visibility:** public

9 **Behavior:**

- 10 a) Invoke the method `each` on the receiver.
- 11 b) If the method `each` does not yield any elements, return `nil`.
- 12 c) For each element *X* which the method `each` yields:
- 13 1) If *X* is the first element, let *V* be *X*.
- 14 2) Otherwise, if the *block* is given:
- 15 i) Call the *block* with *X* and *V* as the arguments. Let *D* be the result of this
- 16 call.
- 17 ii) If *D* is not an instance of the class `Integer`, the behavior is implementation
- 18 dependent.
- 19 iii) If the value of *D* is larger than 0, let new *V* be *X*.
- 20 If the *block* is not given:
- 21 i) Invoke the method `<=>` on *X* with *V* as the argument. Let *D* be the result
- 22 of this invocation.
- 23 ii) If *D* is not an instance of the class `Integer`, the behavior is implementation
- 24 dependent.
- 25 iii) If the value of *D* is larger than 0, let new *V* be *X*.
- 26 d) Return *V*.

1 **15.3.2.1.14 Enumerable#min**

2 `min(&block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Invoke the method `each` on the receiver.
- 6 b) If the method `each` does not yield any elements, return `nil`.
- 7 c) For each element X which the method `each` yields:
- 8 1) If X is the first element, let V be X .
- 9 2) Otherwise, if the *block* is given:
- 10 i) Call the *block* with X and V as the arguments. Let D be the result of this
- 11 call.
- 12 ii) If D is not an instance of the class `Integer`, the behavior is implementation
- 13 dependent.
- 14 iii) If the value of D is smaller than 0, let new V be X .
- 15 If the *block* is not given:
- 16 i) Invoke the method `<=>` on X with V as the argument. Let D be the result
- 17 of this invocation.
- 18 ii) If D is not an instance of the class `Integer`, the behavior is implementation
- 19 dependent.
- 20 iii) If the value of D is smaller than 0, let new V be X .
- 21 d) Return V .

22 **15.3.2.1.15 Enumerable#member?**

23 `member?(obj)`

24 **Visibility:** public

25 **Behavior:** Same as the method `include?` (see §15.3.2.1.10).

26 **15.3.2.1.16 Enumerable#partition**

1 `partition(&block)`

2 **Visibility:** public

3 **Behavior:**

4 a) If the *block* is not given, the behavior is implementation dependent.

5 b) Create two empty instances of the class **Array** *T* and *F*.

6 c) Invoke the method **each** on the receiver.

7 d) For each element *X* which **each** yields, call the *block* with *X* as the argument.

8 If this call results in a true value, append *X* to *T*. If this call results in a false value,
9 append *X* to *F*.

10 e) Return a newly created instance the class **Array**, which contains only *T* and *F* in this
11 order.

12 **15.3.2.1.17 Enumerable#reject**

13 `reject(&block)`

14 **Visibility:** public

15 **Behavior:**

16 a) If the *block* is not given, the behavior is implementation dependent.

17 b) Create an empty direct instance of the class **Array** *A*.

18 c) Invoke the method **each** on the receiver.

19 d) For each element *X* which **each** yields, call the *block* with *X* as the argument. If this
20 call results in a false value, append the element to *A*.

21 e) Return *A*.

22 **15.3.2.1.18 Enumerable#select**

23 `select(&block)`

24 **Visibility:** public

25 **Behavior:** Same as the method **find_all** (see §15.3.2.1.8).

1 **15.3.2.1.19 Enumerable#sort**

2 `sort(&block)`

3 **Visibility:** public

4 **Behavior:**

- 5 a) Create an empty direct instance of the class **Array** *A*.
- 6 b) Invoke the method **each** on the receiver.
- 7 c) Insert all the elements which the method **each** yields into *A*. For any two elements E_i
8 and E_j of *A*, all of the following conditions shall hold:
- 9 1) Let i and j be the index of E_i and E_j , respectively.
- 10 2) If the *block* is given:
- 11 i) Suppose the *block* is called with E_i and E_j as the arguments.
- 12 ii) If this invocation does not result in an instance of the class **Integer**, the
13 behavior is implementation dependent.
- 14 iii) If this invocation results in an instance of the class **Integer** whose value is
15 larger than 0, j shall be larger than i .
- 16 iv) If this invocation results in an instance of the class **Integer** whose value is
17 smaller than 0, i shall be larger than j .
- 18 3) If the *block* is not given:
- 19 i) Suppose the method `<=>` is invoked on E_i with E_j as the argument.
- 20 ii) If this invocation does not result in an instance of the class **Integer**, the
21 behavior is implementation dependent.
- 22 iii) If this invocation results in an instance of the class **Integer** whose value is
23 larger than 0, j shall be larger than i .
- 24 iv) If this invocation results in an instance of the class **Integer** whose value is
25 smaller than 0, i shall be larger than j .
- 26 d) Return *A*.

27 **15.3.2.1.20 Enumerable#to_a**

28 `to_a`

1 **Visibility:** public

2 **Behavior:** Same as the method `entries` (see §15.3.2.1.6).

3 **15.3.3 Comparable**

4 The module `Comparable` provides methods which compare the receiver and an argument using
5 the method `<=>`.

6 **15.3.3.1 Instance methods**

7 **15.3.3.1.1 Comparable#<**

8 `<(other)`

9 **Visibility:** public

10 **Behavior:**

11 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
12 resulting value of this invocation.

13 b) If *I* is not an instance of the class `Integer`, the behavior is implementation dependent.

14 c) If the value of *I* is smaller than 0, return `true`. Otherwise, return `false`.

15 **15.3.3.1.2 Comparable#<=**

16 `<=(other)`

17 **Visibility:** public

18 **Behavior:**

19 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
20 resulting value of this invocation.

21 b) If *I* is not an instance of the class `Integer`, the behavior is implementation dependent.

22 c) If the value of *I* is smaller than or equal to 0, return `true`. Otherwise, return `false`.

23 **15.3.3.1.3 Comparable#==**

24 `==(other)`

25 **Visibility:** public

- 1 **Behavior:**
- 2 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *I* be the
- 3 resulting value of this invocation.
- 4 b) If *I* is not an instance of the class `Integer`, the behavior is implementation dependent.
- 5 c) If the value of *I* is 0, return `true`. Otherwise, return `false`.

6 15.3.3.1.4 Comparable#>

7 >(*other*)

8 **Visibility:** public

9 **Behavior:**

- 10 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *N* be the
- 11 resulting value of this invocation.
- 12 b) If *I* is not an instance of the class `Integer`, the behavior is implementation dependent.
- 13 c) If the value of *I* is larger than 0, return `true`. Otherwise, return `false`.

14 15.3.3.1.5 Comparable#>=

15 >=(*other*)

16 **Visibility:** public

17 **Behavior:**

- 18 a) Invoke the method `<=>` on the receiver with the *other* as the argument. Let *N* be the
- 19 resulting value of this invocation.
- 20 b) If *I* is not an instance of the class `Integer`, the behavior is implementation dependent.
- 21 c) If the value of *I* is larger than or equal to 0, return `true`. Otherwise, return `false`.

22 15.3.3.1.6 Comparable#between?

23 between?(*left*, *right*)

24 **Visibility:** public

25 **Behavior:**

1 a) Invoke the method `<=>` on the receiver with the *left* as the argument. Let I_1 be the
2 resulting value of this invocation.

3 1) If I_1 is not an instance of the class `Integer`, the behavior is implementation de-
4 pendent.

5 2) If the value of I_1 is smaller than 0, return `false`.

6 b) Invoke the method `<=>` on the receiver with the *right* as the argument. Let I_2 be the
7 resulting value of this invocation.

8 1) If I_2 is not an instance of the class `Integer`, the behavior is implementation de-
9 pendent.

10 2) If the value of I_2 is larger than 0, return `false`. Otherwise, return `true`.

Annex A

(informative)

Grammar Summary

A.1 Lexical structure

A.1.1 Source text

see §8.1

source-character ::
[any character in ISO/IEC 646]

A.1.2 Line terminators

see §8.2

```
line-terminator ::
    0xd? 0xa
```

A.1.3 Whitespace

see §8.3

```
whitespace ::
    0x09 | 0x0b | 0x0c | 0x0d | 0x20 | \ 0x0d? 0x0a
```

A.1.4 Comments

see §8.4

```
comment ::
    single-line-comment
  | multi-line-comment
```

```
single-line-comment ::
    # comment-content?
```

```
comment-content ::
    line-content
```

```

1  line-content ::
2      source-character +

3  multi-line-comment ::
4      multi-line-comment-begin-line multi-line-comment-line?
5      multi-line-comment-end-line

6  multi-line-comment-begin-line ::
7      [ beginning of a line ] =begin rest-of-begin-end-line? line-terminator

8  multi-line-comment-end-line ::
9      [ beginning of a line ] =end rest-of-begin-end-line?
10     ( line-terminator | [ end of a program ] )

11 rest-of-begin-end-line ::
12     whitespace + comment-content

13 line ::
14     comment-content line-terminator

15 multi-line-comment-line ::
16     line but not multi-line-comment-end-line

```

17 A.1.5 Tokens

18 see §8.5

```

19 token ::
20     reserved-word
21     | identifier
22     | punctuator
23     | operator
24     | literal

```

25 A.1.5.1 Reserved words

26 see §8.5.1

```

27 reserved-word ::
28     __LINE__ | __ENCODING__ | __FILE__ | BEGIN | END | alias | and | begin
29     | break | case | class | def | defined? | do | else | elsif | end
30     | ensure | for | false | if | in | module | next | nil | not | or | redo
31     | rescue | retry | return | self | super | then | true | undef | unless
32     | until | when | while | yield

```

1 A.1.5.2 Identifiers

2 see §8.5.2

3 *identifier* ::

4 *local-variable-identifier*

5 | *global-variable-identifier*

6 | *class-variable-identifier*

7 | *instance-variable-identifier*

8 | *constant-identifier*

9 | *method-identifier*

10 *local-variable-identifier* ::

11 (*lowercase-character* | *_*) *identifier-character**

12 *global-variable-identifier* ::

13 \$ *identifier-start-character* *identifier-character**

14 *class-variable-identifier* ::

15 @@ *identifier-start-character* *identifier-character**

16 *instance-variable-identifier* ::

17 @ *identifier-start-character* *identifier-character**

18 *constant-identifier* ::

19 *uppercase-character* *identifier-character**

20 *method-identifier* ::

21 *method-only-identifier*

22 | *assignment-like-method-identifier*

23 | *constant-identifier*

24 | *local-variable-identifier*

25 *method-only-identifier* ::

26 (*constant-identifier* | *local-variable-identifier*) (! | ?)

27 *assignment-like-method-identifier* ::

28 (*constant-identifier* | *local-variable-identifier*) =

29 *identifier-character* ::

30 *lowercase-character*

31 | *uppercase-character*

32 | *decimal-digit*

33 | *-*


```

1  identifier-start-character ::
2      lowercase-character
3      | uppercase-character
4      | _

5  uppercase-character ::
6      A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R
7      | S | T | U | V | W | X | Y | Z

8  lowercase-character ::
9      a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r
10     | s | t | u | v | w | x | y | z

11 decimal-digit ::
12     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

13 A.1.5.3 Punctuators

14 see §8.5.3

```

15 punctuator ::
16     [ | ] | ( | ) | { | } | :: | , | ; | .. | ... | ? | : | =>

```

17 A.1.5.4 Operators

18 see §8.5.4

```

19 operator ::
20     operator-method-name
21     | assignment-operator

22 operator-method-name ::
23     ^ | & | | | <=> | == | === | !~ | =~ | > | >= | < | <= | << | >> | +
24     | - | * | / | % | ** | ~ | +@ | -@ | [] | []= | '

25 assignment-operator ::
26     assignment-operator-name =

27 assignment-operator-name ::
28     + | - | * | ** | / | ^ | % | << | >> | & | && | || | |

```

1 A.1.5.5 Literals

2 see §8.5.5

3 *literal* ::
4 *numeric-literal*
5 | *string-literal*
6 | *array-literal*
7 | *regular-expression-literal*
8 | *symbol*

9 A.1.5.5.1 Numeric literals

10 see §8.5.5.1

11 *numeric-literal* ::
12 *signed-number*
13 | *unsigned-number*

14 *unsigned-number* ::
15 *integer-literal*
16 | *float-literal*

17 *integer-literal* ::
18 *decimal-integer-literal*
19 | *binary-integer-literal*
20 | *octal-integer-literal*
21 | *hexadecimal-integer-literal*

22 *decimal-integer-literal* ::
23 *digit-decimal-integer-literal*
24 | *prefixed-decimal-integer-literal*

25 *digit-decimal-integer-literal* ::
26 0
27 | *decimal-digit-without-zero* (*_?* *decimal-digit*)*

28 *prefixed-decimal-integer-literal* ::
29 0 (*d* | *D*) *digit-decimal-part*

30 *digit-decimal-part* ::
31 *decimal-digit* (*_?* *decimal-digit*)*

32 *binary-integer-literal* ::
33 0 (*b* | *B*) *binary-digit* (*_?* *binary-digit*)*

```

1  octal-integer-literal ::
2      0 ( _ | o | 0 )? octal-digit ( _? octal-digit )*

3  hexadecimal-integer-literal ::
4      0 ( x | X ) hexadecimal-digit ( _? hexadecimal-digit )*

5  float-literal ::
6      decimal-float-literal
7      | exponent-float-literal

8  decimal-float-literal ::
9      digit-decimal-integer-literal . digit-decimal-part

10 exponent-float-literal ::
11     base-part exponent-part

12 base-part ::
13     decimal-float-literal
14     | digit-decimal-integer-literal

15 exponent-part ::
16     ( e | E ) ( + | - )? digit-decimal-part

17 signed-number ::
18     ( + | - ) unsigned-number

19 decimal-digit-without-zero ::
20     1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

21 octal-digit ::
22     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

23 binary-digit ::
24     0 | 1

25 hexadecimal-digit ::
26     decimal-digit | a | b | c | d | e | f | A | B | C | D | E | F

```

27 A.1.5.5.2 String literals

28 see §8.5.5.2

```

29 string-literal ::
30     single-quoted-string

```

1 | *double-quoted-string*
2 | *quoted-non-expanded-literal-string*
3 | *quoted-expanded-literal-string*
4 | *here-document*
5 | *external-command-execution*

6 **A.1.5.5.2.1 Single quoted strings**

7 see §8.5.5.2.1

8 *single-quoted-string* ::
9 ' *single-quoted-string-character** '

10 *single-quoted-string-character* ::
11 *non-escaped-single-quoted-string-character*
12 | *single-quoted-escape-sequence*

13 *single-quoted-escape-sequence* ::
14 *single-escape-character-sequence*
15 | *non-escaped-single-quoted-string-character-sequence*

16 *single-escape-character-sequence* ::
17 \ *single-escaped-character*

18 *non-escaped-single-quoted-string-character-sequence* ::
19 \ *non-escaped-single-quoted-string-character*

20 *single-escaped-character* ::
21 ' | \

22 *non-escaped-single-quoted-string-character* ::
23 *source-character* **but not** *single-escaped-character*

24 **A.1.5.5.2.2 Double quoted strings**

25 see §8.5.5.2.2

26 *double-quoted-string* ::
27 " *double-quoted-string-character** "

28 *double-quoted-string-character* ::
29 *source-character* **but not** (" | \)
30 | *double-escape-sequence*
31 | *interpolated-character-sequence*

```

1  double-escape-sequence ::
2      simple-escape-sequence
3      | non-escaped-sequence
4      | line-terminator-escape-sequence
5      | octal-escape-sequence
6      | hex-escape-sequence
7      | control-escape-sequence

8  simple-escape-sequence ::
9      \ double-escaped-character

10 non-escaped-sequence ::
11     \ non-escaped-double-quoted-string-character

12 line-terminator-escape-sequence ::
13     \ line-terminator

14 non-escaped-double-quoted-string-character ::
15     source-character but not ( double-escaped-character | line-terminator )

16 double-escaped-character ::
17     \ | n | t | r | f | v | a | e | b | s

18 octal-escape-sequence ::
19     \ octal-digit ( octal-digit octal-digit? )?

20 hex-escape-sequence ::
21     \ x hexadecimal-digit hexadecimal-digit?

22 control-escape-sequence ::
23     \ ( C - | c ) control-escaped-character

24 control-escaped-character ::
25     double-escape-sequence
26     | ?
27     | source-character but not ( \ | ? )

28 interpolated-character-sequence ::
29     # global-variable-identifier
30     | # class-variable-identifier
31     | # instance-variable-identifier
32     | # { compound-statement }

```

1 A.1.5.5.2.3 Quoted non-expanded literal strings

2 see §8.5.5.2.3

3 *quoted-non-expanded-literal-string* ::
4 %q *literal-beginning-delimiter* *non-expanded-literal-string** *literal-ending-delimiter*

5 *non-expanded-literal-string* ::
6 *non-expanded-literal-character*
7 | *non-expanded-delimited-string*

8 *non-expanded-delimited-string* ::
9 *literal-beginning-delimiter* *non-expanded-literal-string** *literal-ending-delimiter*

10 *non-expanded-literal-character* ::
11 *non-escaped-literal-character*
12 | *non-expanded-literal-escape-sequence*

13 *non-escaped-literal-character* ::
14 *source-character* **but not** *quoted-literal-escape-character*

15 *non-expanded-literal-escape-sequence* ::
16 *non-expanded-literal-escape-character-sequence*
17 | *non-escaped-non-expanded-literal-character-sequence*

18 *non-expanded-literal-escape-character-sequence* ::
19 \ *non-expanded-literal-escaped-character*

20 *non-expanded-literal-escaped-character* ::
21 *literal-beginning-delimiter*
22 | *literal-ending-delimiter*
23 | \

24 *quoted-literal-escape-character* ::
25 *non-expanded-literal-escaped-character*

26 *non-escaped-non-expanded-literal-character-sequence* ::
27 \ *non-escaped-non-expanded-literal-character*

28 *non-escaped-non-expanded-literal-character* ::
29 *source-character* **but not** *non-expanded-literal-escaped-character*

1 A.1.5.5.2.4 Quoted expanded literal strings

2 see §8.5.5.2.4

3 *quoted-expanded-literal-string* ::
4 % Q? *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

5 *expanded-literal-string* ::
6 *expanded-literal-character*
7 | *expanded-delimited-string*

8 *expanded-literal-character* ::
9 *non-escaped-literal-character*
10 | *double-escape-sequence*
11 | *interpolated-character-sequence*

12 *expanded-delimited-string* ::
13 *literal-beginning-delimiter* *expanded-literal-string** *literal-ending-delimiter*

14 *literal-beginning-delimiter* ::
15 *source-character* **but not** *alpha-numeric-character-or-separator*

16 *alpha-numeric-character-or-separator* ::
17 *whitespace*
18 | *line-terminator*
19 | *uppercase-character*
20 | *lowercase-character*
21 | *decimal-digit*

22 *literal-ending-delimiter* ::
23 [depending on the *literal-beginning-delimiter*]

24 *matching-literal-beginning-delimiter* ::
25 (| { | < | [

26 A.1.5.5.2.5 Here documents

27 see §8.5.5.2.5

28 *here-document* ::
29 *heredoc-start-line* *heredoc-body* *heredoc-end-line*

30 *heredoc-start-line* ::
31 *heredoc-signifier* *rest-of-line*

```

1  heredoc-signifier ::
2      << heredoc-delimiter-specifier

3  rest-of-line ::
4      line-content? line-terminator

5  heredoc-body ::
6      heredoc-body-line*

7  heredoc-body-line ::
8      line but not heredoc-end-line

9  heredoc-delimiter-specifier ::
10     -? heredoc-delimiter

11  heredoc-delimiter ::
12     non-quoted-delimiter
13     | single-quoted-delimiter
14     | double-quoted-delimiter
15     | command-quoted-delimiter

16  non-quoted-delimiter ::
17     non-quoted-delimiter-identifier

18  non-quoted-delimiter-identifier ::
19     identifier-character*

20  single-quoted-delimiter ::
21     ' single-quoted-delimiter-identifier* '

22  single-quoted-delimiter-identifier ::
23     source-character but not '

24  double-quoted-delimiter ::
25     " double-quoted-delimiter-identifier* "

26  double-quoted-delimiter-identifier ::
27     source-character but not "

28  command-quoted-delimiter ::
29     ` command-quoted-delimiter-identifier* `

30  command-quoted-delimiter-identifier ::
31     source-character but not `

```



```

1  heredoc-end-line ::
2      indented-heredoc-end-line
3      | non-indented-heredoc-end-line

4  indented-heredoc-end-line ::
5      [ beginning of a line ] whitespace* heredoc-delimiter-identifier line-terminator

6  non-indented-heredoc-end-line ::
7      [ beginning of a line ] heredoc-delimiter-identifier line-terminator

8  heredoc-delimiter-identifier ::
9      non-quoted-delimiter-identifier
10     | single-quoted-delimiter-identifier
11     | double-quoted-delimiter-identifier
12     | command-quoted-delimiter-identifier

```

13 **A.1.5.5.2.6 External command execution**

14 see §8.5.5.2.6

```

15  external-command-execution ::
16      backquoted-external-command-execution
17      | quoted-external-command-execution

18  backquoted-external-command-execution ::
19      ‘ double-quoted-string-character* ‘

20  quoted-external-command-execution ::
21      %x literal-beginning-delimiter expanded-literal-string* literal-ending-delimiter

```

22 **A.1.5.5.3 Array literals**

23 see §8.5.5.3

```

24  array-literal ::
25      quoted-non-expanded-array-constructor
26      | quoted-expanded-array-constructor

27  quoted-non-expanded-array-constructor ::
28      %w literal-beginning-delimiter non-expanded-array-content literal-ending-delimiter

29  non-expanded-array-content ::
30      quoted-array-item-separator-list? non-expanded-array-item-list?
31      quoted-array-item-separator-list?

```

```

1  non-expanded-array-item-list ::
2      non-expanded-array-item ( quoted-array-item-separator-list non-expanded-array-item )*

3  quoted-array-item-separator-list ::
4      quoted-array-item-separator +

5  quoted-array-item-separator ::
6      whitespace
7      | line-terminator

8  non-expanded-array-item ::
9      non-expanded-array-item-character +

10 non-expanded-array-item-character ::
11     non-escaped-array-item-character
12     | non-expanded-array-escape-sequence

13 non-escaped-array-item-character ::
14     non-escaped-array-character
15     | matching-literal-delimiter

16 non-escaped-array-character ::
17     non-escaped-literal-character but not quoted-array-item-separator

18 matching-literal-delimiter ::
19     ( | { | < | [ | ) | } | > | ]

20 non-expanded-array-escape-sequence ::
21     non-expanded-literal-escape-sequence but not escaped-quoted-array-item-separator
22     | escaped-quoted-array-item-separator

23 escaped-quoted-array-item-separator ::
24     \ quoted-array-item-separator

25 quoted-expanded-array-constructor ::
26     %W literal-beginning-delimiter expanded-array-content literal-ending-delimiter

27 expanded-array-content ::
28     quoted-array-item-separator-list? expanded-array-item-list?
29     quoted-array-item-separator-list?

30 expanded-array-item-list ::
31     expanded-array-item ( quoted-array-item-separator-list expanded-array-item )*

```

```

1  expanded-array-item ::
2      expanded-array-item-character +

3  expanded-array-item-character ::
4      non-escaped-array-item-character
5      | expanded-array-escape-sequence
6      | interpolated-character-sequence

7  expanded-array-escape-sequence ::
8      double-escape-sequence but not escaped-quoted-array-item-separator
9      | escaped-quoted-array-item-separator

```

10 **A.1.5.5.4 Regular expression literals**

11 see §8.5.5.4

```

12 regular-expression-literal ::
13     / [no whitespace here] regular-expression-body / regular-expression-option*
14     | %r literal-beginning-delimiter expanded-literal-string*
15         literal-ending-delimiter regular-expression-option*

16 regular-expression-body ::
17     regular-expression-character*

18 regular-expression-character ::
19     source-character but not ( / | \ )
20     | \ \
21     | line-terminator-escape-sequence
22     | interpolated-character-sequence

23 regular-expression-option ::
24     i | m

```

25 **A.2 Program structure**

26 **A.2.1 Program**

27 see §10.1

```

28 program ::
29     compound-statement

```

A.3 Expressions

A.3.1 Logical expressions

see §11.1

keyword-logical-expression ::
 keyword-NOT-expression
 | *keyword-AND-expression*
 | *keyword-OR-expression*

keyword-NOT-expression ::
 method-invocation-without-parentheses
 | *operator-expression*
 | *logical-NOT-with-method-invocation-without-parentheses*
 | **not** *keyword-NOT-expression*

logical-NOT-expression ::=
 logical-NOT-with-method-invocation-without-parentheses
 | *logical-NOT-with-unary-expression*

logical-NOT-with-method-invocation-without-parentheses ::
 ! *method-invocation-without-parentheses*

logical-NOT-with-unary-expression ::
 ! *unary-expression*

keyword-AND-expression ::
 expression **and** *keyword-NOT-expression*

keyword-OR-expression ::
 expression **or** *keyword-NOT-expression*

logical-OR-expression ::
 logical-AND-expression
 | *logical-OR-expression* || *logical-AND-expression*

logical-AND-expression ::
 equality-expression
 | *logical-AND-expression* && *equality-expression*

A.3.2 Method invocation expressions

see §11.2

```

1  primary-method-invocation ::
2      super-with-optional-argument
3      | indexing-method-invocation
4      | method-only-identifier
5      | method-identifier ( [no whitespace here] argument-with-parentheses )? block?
6      | primary-expression [no line-terminator here]
7          . method-name ( [no whitespace here] argument-with-parentheses )? block?
8      | primary-expression [no line-terminator here]
9          :: method-name [no whitespace here] argument-with-parentheses block?
10     | primary-expression [no line-terminator here] :: method-name-without-constant
11     block?

12 indexing-method-invocation ::
13     primary-expression [no line-terminator here] optional-whitespace?
14     [ indexing-argument-list? ]

15 optional-whitespace ::
16     [ whitespace here ]

17 method-name-without-constant ::
18     method-name but not constant-identifier

19 method-invocation-without-parentheses ::
20     command
21     | chained-command-with-do-block
22     | chained-command-with-do-block ( . | :: ) method-name argument
23     | return-with-argument
24     | break-with-argument
25     | next-with-argument

26 command ::
27     super-with-argument
28     | yield-without-parentheses
29     | method-identifier argument
30     | primary-expression [no line-terminator here] ( . | :: ) method-name argument

31 chained-command-with-do-block ::
32     command-with-do-block chained-method-invocation *

33 chained-method-invocation ::
34     ( . | :: ) method-name
35     | ( . | :: ) method-name [no whitespace here]
36     [lookahead ∉ { { } ] argument-with-parentheses

37 command-with-do-block ::
38     super-with-argument-and-do-block

```

```

1      | method-identifier argument do-block
2      | primary-expression [no line-terminator here] ( . | :: ) method-name argument
3      do-block

```

4 **A.3.2.1 Method arguments**

5 see §11.2.1

```

6      indexing-argument-list ::
7          command
8          | operator-expression-list ,?
9          | operator-expression-list , splatting-argument
10         | association-list ,?
11         | splatting-argument

12      splatting-argument ::
13          * no-whitespace-after-symbol? operator-expression

14      no-whitespace-after-symbol ::
15          [no whitespace here]

16      operator-expression-list ::
17          operator-expression ( , operator-expression )*

18      argument-with-parentheses ::
19          ( )
20          | ( argument-in-parentheses )
21          | ( operator-expression-list , chained-command-with-do-block )
22          | ( chained-command-with-do-block )

23      argument ::
24          [no line-terminator here] [lookahead  $\notin$  { { }] optional-whitespace?
25          argument-in-parentheses

26      argument-in-parentheses ::
27          command
28          | ( operator-expression-list | association-list )
29             ( , splatting-argument )? ( , block-argument )?
30          | operator-expression-list , association-list
31             ( , splatting-argument )? ( , block-argument )?
32          | splatting-argument ( , block-argument )?
33          | block-argument

34      block-argument ::
35          & no-whitespace-after-symbol? operator-expression

```

1 A.3.2.2 Blocks

2 see §11.2.2

```
3  block ::  
4      brace-block  
5      | do-block  
  
6  brace-block ::  
7      { block-formal-argument? block-body }  
  
8  do-block ::  
9      do block-formal-argument? block-body end  
  
10 block-formal-argument ::  
11     | |  
12     | ||  
13     | | block-formal-argument-list |  
  
14 block-formal-argument-list ::  
15     left-hand-side  
16     | multiple-left-hand-side  
  
17 block-body ::  
18     compound-statement
```

19 A.3.2.3 The super expression

20 see §11.2.3

```
21 super-expression ::=  
22     super-with-optional-argument  
23     | super-with-argument  
24     | super-with-argument-and-do-block  
  
25 super-with-optional-argument ::  
26     super ( [no whitespace here] argument-with-parentheses )? block?  
  
27 super-with-argument ::  
28     super argument  
  
29 super-with-argument-and-do-block ::  
30     super argument do-block
```

1 **A.3.2.4 The yield expression**

2 see §11.2.4

3 *yield-expression* ::=
4 *yield-with-parentheses*
5 | *yield-without-parentheses*

6 *yield-with-parentheses* ::
7 *yield-with-parentheses-and-argument*
8 | *yield-with-parentheses-without-argument*
9 | **yield**

10 *yield-with-parentheses-and-argument* ::
11 **yield** [no whitespace here] (*argument-in-parentheses*)

12 *yield-with-parentheses-without-argument* ::
13 **yield** [no whitespace here] ()

14 *yield-without-parentheses* ::
15 **yield** *argument*

16 **A.3.3 Operator expressions**

17 see §11.3

18 *operator-expression* ::
19 *assignment-expression*
20 | *defined?-without-parentheses*
21 | *conditional-operator-expression*

22 **A.3.3.1 Assignments**

23 see §11.3.1

24 *assignment* ::=
25 *assignment-expression*
26 | *assignment-statement*

27 *assignment-expression* ::
28 *single-assignment-expression*
29 | *abbreviated-assignment-expression*
30 | *assignment-with-rescue-modifier*


```

1  assignment-statement ::
2      single-assignment-statement
3      | abbreviated-assignment-statement
4      | multiple-assignment-statement

```

5 **A.3.3.1.1 Single assignments**

6 see §11.3.1.1

```

7  single-assignment ::=
8      single-assignment-expression
9      | single-assignment-statement

10 single-assignment-expression ::
11     single-variable-assignment-expression
12     | scoped-constant-assignment-expression
13     | single-indexing-assignment-expression
14     | single-method-assignment-expression

15 single-assignment-statement ::
16     single-variable-assignment-statement
17     | scoped-constant-assignment-statement
18     | single-indexing-assignment-statement
19     | single-method-assignment-statement

```

20 **A.3.3.1.1.1 Single variable assignments**

21 see §11.3.1.1.1

```

22 single-variable-assignment ::=
23     single-variable-assignment-expression
24     | single-variable-assignment-statement

25 single-variable-assignment-expression ::
26     variable [no line-terminator here] = operator-expression

27 single-variable-assignment-statement ::
28     variable [no line-terminator here] = method-invocation-without-parentheses

29 scoped-constant-assignment ::=
30     scoped-constant-assignment-expression
31     | scoped-constant-assignment-statement

```

```

1  scoped-constant-assignment-expression ::
2      primary-expression [no whitespace here] :: constant-identifier
3      [no line-terminator here] = operator-expression
4      | :: constant-identifier [no line-terminator here] = operator-expression

5  scoped-constant-assignment-statement ::
6      primary-expression [no whitespace here] :: constant-identifier
7      [no line-terminator here] = method-invocation-without-parentheses
8      | :: constant-identifier [no line-terminator here] = method-invocation-without-parentheses

```

9 **A.3.3.1.1.2 Single indexing assignments**

10 see §11.3.1.1.2

```

11  single-indexing-assignment ::=
12      single-indexing-assignment-expression
13      | single-indexing-assignment-statement

14  single-indexing-assignment-expression ::
15      primary-expression [no line-terminator here] [ indexing-argument-list? ]
16      [no line-terminator here] = operator-expression

17  single-indexing-assignment-statement ::
18      primary-expression [no line-terminator here] [ indexing-argument-list? ]
19      [no line-terminator here] = method-invocation-without-parentheses

```

20 **A.3.3.1.1.3 Single method assignments**

21 see §11.3.1.1.3

```

22  single-method-assignment ::=
23      single-method-assignment-expression
24      | single-method-assignment-statement

25  single-method-assignment-expression ::
26      primary-expression [no line-terminator here] ( . | :: ) local-variable-identifier
27      [no line-terminator here] = operator-expression
28      | primary-expression [no line-terminator here] . constant-identifier
29      [no line-terminator here] = operator-expression

30  single-method-assignment-statement ::
31      primary-expression [no line-terminator here] ( . | :: ) local-variable-identifier
32      [no line-terminator here] = method-invocation-without-parentheses
33      | primary-expression [no line-terminator here] . constant-identifier
34      [no line-terminator here] = method-invocation-without-parentheses

```

1 A.3.3.1.2 Abbreviated assignments

2 see §11.3.1.2

3 *abbreviated-assignment ::=*
4 *abbreviated-assignment-expression*
5 | *abbreviated-assignment-statement*

6 *abbreviated-assignment-expression ::*
7 *abbreviated-variable-assignment-expression*
8 | *abbreviated-indexing-assignment-expression*
9 | *abbreviated-method-assignment-expression*

10 *abbreviated-assignment-statement ::*
11 *abbreviated-variable-assignment-statement*
12 | *abbreviated-indexing-assignment-statement*
13 | *abbreviated-method-assignment-statement*

14 A.3.3.1.2.1 Abbreviated variable assignments

15 see §11.3.1.2.1

16 *abbreviated-variable-assignment ::=*
17 *abbreviated-variable-assignment-expression*
18 | *abbreviated-variable-assignment-statement*

19 *abbreviated-variable-assignment-expression ::*
20 *variable* [no line-terminator here] *assignment-operator operator-expression*

21 *abbreviated-variable-assignment-statement ::*
22 *variable* [no line-terminator here] *assignment-operator*
23 *method-invocation-without-parentheses*

24 A.3.3.1.2.2 Abbreviated indexing assignments

25 see §11.3.1.2.2

26 *abbreviated-indexing-assignment ::=*
27 *abbreviated-indexing-assignment-expression*
28 | *abbreviated-indexing-assignment-statement*

```

1  abbreviated-indexing-assignment-expression ::
2      primary-expression [no line-terminator here] [ indexing-argument-list? ]
3      [no line-terminator here] assignment-operator operator-expression

4  abbreviated-indexing-assignment-statement ::
5      primary-expression [no line-terminator here] [ indexing-argument-list? ]
6      [no line-terminator here] assignment-operator method-invocation-without-parentheses

```

7 **A.3.3.1.2.3 Abbreviated method assignments**

8 see §11.3.1.2.3

```

9  abbreviated-method-assignment ::=
10     abbreviated-method-assignment-expression
11     | abbreviated-method-assignment-statement

12  abbreviated-method-assignment-expression ::
13     primary-expression [no line-terminator here] ( . | :: ) local-variable-identifier
14     [no line-terminator here] assignment-operator operator-expression
15     | primary-expression [no line-terminator here] . constant-identifier
16     [no line-terminator here] assignment-operator operator-expression

17  abbreviated-method-assignment-statement ::
18     primary-expression [no line-terminator here] ( . | :: ) local-variable-identifier
19     [no line-terminator here] assignment-operator method-invocation-without-parentheses
20     | primary-expression [no line-terminator here] . constant-identifier
21     [no line-terminator here] assignment-operator method-invocation-without-parentheses

```

22 **A.3.3.1.3 Multiple assignments**

23 see §11.3.1.3

```

24  multiple-assignment-statement ::
25     many-to-one-assignment-statement
26     | one-to-packing-assignment-statement
27     | many-to-many-assignment-statement

28  many-to-one-assignment-statement ::
29     left-hand-side [no line-terminator here] = multiple-right-hand-side

30  one-to-packing-assignment-statement ::
31     packing-left-hand-side [no line-terminator here] =
32     ( method-invocation-without-parentheses | operator-expression )

```

```

1  many-to-many-assignment-statement ::
2      multiple-left-hand-side [no line-terminator here] = multiple-right-hand-side
3      | ( multiple-left-hand-side but not packing-left-hand-side )
4          [no line-terminator here] =
5          ( method-invocation-without-parentheses | operator-expression )

6  left-hand-side ::
7      variable
8      | primary-expression [no line-terminator here] [ indexing-argument-list? ]
9      | primary-expression [no line-terminator here]
10         ( . | :: ) ( local-variable-identifier | constant-identifier )
11         | :: constant-identifier

12 multiple-left-hand-side ::
13     ( multiple-left-hand-side-item , )+ multiple-left-hand-side-item?
14     | ( multiple-left-hand-side-item , )+ packing-left-hand-side?
15     | packing-left-hand-side
16     | grouped-left-hand-side

17 packing-left-hand-side ::
18     * left-hand-side?

19 grouped-left-hand-side ::
20     ( multiple-left-hand-side )

21 multiple-left-hand-side-item ::
22     left-hand-side
23     | grouped-left-hand-side

24 multiple-right-hand-side ::
25     operator-expression-list ( , splatting-right-hand-side )?
26     | splatting-right-hand-side

27 splatting-right-hand-side ::
28     splatting-argument

```

29 A.3.3.1.4 Assignments with rescue modifiers

30 see §11.3.1.4

```

31 assignment-with-rescue-modifier ::
32     left-hand-side [no line-terminator here] =
33     operator-expression1 rescue operator-expression2

```

1 **A.3.3.2 Unary operators**

2 see §11.3.2

3 *unary-minus-expression* ::
4 *power-expression*₁
5 | - *no-whitespace-after-symbol?* *power-expression*₂

6 *unary-expression* ::
7 *primary-expression*
8 | *logical-NOT-with-unary-expression*
9 | ~ *unary-expression*₁
10 | + *no-whitespace-after-symbol?* *unary-expression*₂

11 **A.3.3.2.1 The defined? expression**

12 see §11.3.2.1

13 *defined?-expression* ::=
14 *defined?-with-parentheses*
15 | *defined?-without-parentheses*

16 *defined?-with-parentheses* ::
17 *defined?* (*expression*)

18 *defined?-without-parentheses* ::
19 *defined?* *operator-expression*

20 **A.3.3.3 Binary operators**

21 see §11.3.3

22 *equality-expression* ::
23 *relational-expression*
24 | *relational-expression* <=> *relational-expression*
25 | *relational-expression* == *relational-expression*
26 | *relational-expression* === *relational-expression*
27 | *relational-expression* != *relational-expression*
28 | *relational-expression* =~ *relational-expression*
29 | *relational-expression* !~ *relational-expression*

30 *relational-expression* ::
31 *bitwise-OR-expression*
32 | *relational-expression* > *bitwise-OR-expression*
33 | *relational-expression* >= *bitwise-OR-expression*

```

1      | relational-expression < bitwise-OR-expression
2      | relational-expression <= bitwise-OR-expression

3  bitwise-OR-expression ::
4      bitwise-AND-expression
5      | bitwise-OR-expression | bitwise-AND-expression
6      | bitwise-OR-expression ^ bitwise-AND-expression

7  bitwise-AND-expression ::
8      bitwise-shift-expression
9      | bitwise-AND-expression whitespace-before-operator? & no-whitespace-after-operator?
10     bitwise-shift-expression

11 bitwise-shift-expression ::
12     additive-expression
13     | bitwise-shift-expression whitespace-before-operator? << no-whitespace-after-operator?
14     additive-expression
15     | bitwise-shift-expression >> additive-expression

16 additive-expression ::
17     multiplicative-expression
18     | additive-expression whitespace-before-operator? + no-whitespace-after-operator?
19     multiplicative-expression
20     | additive-expression whitespace-before-operator? - no-whitespace-after-operator?
21     multiplicative-expression

22 multiplicative-expression ::
23     unary-minus-expression
24     | multiplicative-expression whitespace-before-operator? * no-whitespace-after-operator?
25     unary-minus-expression
26     | multiplicative-expression whitespace-before-operator? / no-whitespace-after-operator?
27     unary-minus-expression
28     | multiplicative-expression whitespace-before-operator? % no-whitespace-after-operator?
29     unary-minus-expression

30 power-expression ::
31     unary-expression
32     | - ( numeric-literal ) ** power-expression
33     | unary-expression ** power-expression

34 binary-operator ::=
35     <=> | == | === | =~ | > | >= | < | <= | | | ^
36     | & | << | >> | + | - | * | / | % | **

37 whitespace-before-operator ::
38     [ whitespace here ]

```

1 *no-whitespace-after-operator* ::
2 [*no whitespace* here]

3 **A.3.4 Primary expressions**

4 see §11.4

5 *primary-expression* ::
6 *class-definition*
7 | *eigenclass-definition*
8 | *module-definition*
9 | *method-definition*
10 | *singleton-method-definition*
11 | *yield-with-parentheses*
12 | *if-expression*
13 | *unless-expression*
14 | *case-expression*
15 | *while-expression*
16 | *until-expression*
17 | *for-expression*
18 | *return-without-argument*
19 | *break-without-argument*
20 | *next-without-argument*
21 | *redo-expression*
22 | *retry-expression*
23 | *rescue-expression*
24 | *grouping-expression*
25 | *variable-reference*
26 | *scoped-constant-reference*
27 | *array-constructor*
28 | *hash-constructor*
29 | *literal*
30 | *defined?-with-parentheses*
31 | *primary-method-invocation*

32 **A.3.4.0.0.1 The if expression**

33 see §11.4.1.1.1

34 *if-expression* ::
35 *if expression then-clause elsif-clause* else-clause? end*

36 *then-clause* ::
37 *separator compound-statement*
38 | *separator? then compound-statement*

1 *else-clause* ::
2 **else** *compound-statement*

3 *elsif-clause* ::
4 **elsif** *expression then-clause*

5 **A.3.4.0.0.2 The unless expression**

6 see §11.4.1.1.2

7 *unless-expression* ::
8 **unless** *expression then-clause else-clause?* **end**

9 **A.3.4.0.0.3 The case expression**

10 see §11.4.1.1.3

11 *case-expression* ::
12 *case-expression-with-expression*
13 | *case-expression-without-expression*

14 *case-expression-with-expression* ::
15 **case** *expression separator-list? when-clause + else-clause?* **end**

16 *case-expression-without-expression* ::
17 **case** *separator-list? when-clause + else-clause?* **end**

18 *when-clause* ::
19 **when** *when-argument then-clause*

20 *when-argument* ::
21 *operator-expression-list* (, *splatting-argument*)?
22 | *splatting-argument*

23 **A.3.4.0.0.4 Conditional operator**

24 see §11.4.1.1.4

25 *conditional-operator-expression* ::
26 *range-constructor*
27 | *range-constructor* ? *operator-expression*₁ : *operator-expression*₂

1 **A.3.4.0.1 Iteration expressions**

2 see §11.4.1.2

3 *iteration-expression* ::=
4 *while-expression*
5 | *until-expression*
6 | *for-expression*
7 | *while-modifier-statement*
8 | *until-modifier-statement*

9 **A.3.4.0.1.1 The while expression**

10 see §11.4.1.2.1

11 *while-expression* ::
12 **while** *expression do-clause* **end**

13 *do-clause* ::
14 *separator compound-statement*
15 | **do** *compound-statement*

16 **A.3.4.0.1.2 The until expression**

17 see §11.4.1.2.2

18 *until-expression* ::
19 **until** *expression do-clause* **end**

20 **A.3.4.0.1.3 The for expression**

21 see §11.4.1.2.3

22 *for-expression* ::
23 **for** *for-variable in expression do-clause* **end**

24 *for-variable* ::
25 *left-hand-side*
26 | *multiple-left-hand-side*

1 **A.3.4.0.2 Jump expressions**

2 see §11.4.1.3

```
3  jump-expression ::=
4      return-expression
5      | break-expression
6      | next-expression
7      | redo-expression
8      | retry-expression
```

9 **A.3.4.0.2.1 The return expression**

10 see §11.4.1.3.1

```
11  return-expression ::=
12      return-without-argument
13      | return-with-argument
```

```
14  return-without-argument ::
15      return
```

```
16  return-with-argument ::
17      return jump-argument
```

```
18  jump-argument ::
19      argument
```

20 **A.3.4.0.2.2 The break expression**

21 see §11.4.1.3.2

```
22  break-expression ::=
23      break-without-argument
24      | break-with-argument
```

```
25  break-without-argument ::
26      break
```

```
27  break-with-argument ::
28      break jump-argument
```

1 **A.3.4.0.2.3 The next expression**

2 see §11.4.1.3.3

3 *next-expression* ::=
4 *next-without-argument*
5 | *next-with-argument*

6 *next-without-argument* ::
7 **next**

8 *next-with-argument* ::
9 **next** *jump-argument*

10 **A.3.4.0.2.4 The redo expression**

11 see §11.4.1.3.4

12 *redo-expression* ::
13 **redo**

14 **A.3.4.0.2.5 The retry expression**

15 see §11.4.1.3.5

16 *retry-expression* ::
17 **retry**

18 **A.3.4.0.2.6 The rescue expression**

19 see §11.4.1.4.1

20 *rescue-expression* ::
21 **begin** *body-statement* **end**

22 *body-statement* ::
23 *compound-statement* *rescue-clause** *else-clause*? *ensure-clause*?

24 *rescue-clause* ::
25 **rescue** [no *line-terminator* here] *exception-class-list*?
26 *exception-variable-assignment*? *then-clause*

```

1  exception-class-list ::
2      operator-expression
3      | multiple-right-hand-side

4  exception-variable-assignment ::
5      => left-hand-side

6  ensure-clause ::
7      ensure compound-statement

```

8 **A.3.4.1 Grouping expression**

9 see §11.4.2

```

10 grouping-expression ::
11     ( expression )
12     | ( compound-statement )

```

13 **A.3.4.2 Variable references**

14 see §11.4.3

```

15 variable-reference ::
16     variable
17     | pseudo-variable

```

```

18 variable ::
19     constant-identifier
20     | global-variable-identifier
21     | class-variable-identifier
22     | instance-variable-identifier
23     | local-variable-identifier

```

```

24 scoped-constant-reference ::
25     primary-expression [no whitespace here] :: constant-identifier
26     | :: constant-identifier

```

27 **A.3.4.2.1 Pseudo variables**

28 see §11.4.3.7

```

29 pseudo-variable ::
30     nil

```

1 | *true*
2 | *false*
3 | *self*

4 **A.3.4.2.1.1 nil**

5 see §11.4.3.7.1

6 *nil* ::
7 *nil*

8 **A.3.4.2.1.2 true and false**

9 see §11.4.3.7.2

10 *true* ::
11 *true*

12 *false* ::
13 *false*

14 **A.3.4.2.1.3 self**

15 see §11.4.3.7.3

16 *self* ::
17 *self*

18 **A.3.4.2.2 Array constructor**

19 see §11.4.4.1

20 *array-constructor* ::
21 [*indexing-argument-list*?]

22 **A.3.4.2.3 Hash constructor**

23 see §11.4.4.2

24 *hash-constructor* ::
25 { (*association-list* ,?)? }

```

1  association-list ::
2      association ( , association )*

3  association ::
4      association-key => association-value

5  association-key ::
6      operator-expression

7  association-value ::
8      operator-expression

```

9 **A.3.4.2.4 Range constructor**

10 see §11.4.4.3

```

11  range-constructor ::
12      logical-OR-expression
13      | logical-OR-expression1 range-operator logical-OR-expression2

14  range-operator ::
15      ..
16      | ...

```

17 **A.4 Statements**

18 **A.4.1 The expression statement**

19 see §12.1

```

20  expression-statement ::
21      expression

```

22 **A.4.2 The if modifier statement**

23 see §12.2

```

24  if-modifier-statement ::
25      statement [no line-terminator here] if expression

```

1 **A.4.3 The unless modifier statement**

2 see §12.3

3 *unless-modifier-statement* ::
4 *statement* [no line-terminator here] **unless** *expression*

5 **A.4.4 The while modifier statement**

6 see §12.4

7 *while-modifier-statement* ::
8 *statement* [no line-terminator here] **while** *expression*

9 **A.4.5 The until modifier statement**

10 see §12.5

11 *until-modifier-statement* ::
12 *statement* [no line-terminator here] **until** *expression*

13 **A.5 Classes and modules**

14 **A.5.0.1 Module definition**

15 see §13.1.2

16 *module-definition* ::
17 **module** *module-path* *module-body* **end**

18 *module-path* ::
19 *top-module-path*
20 | *module-name*
21 | *nested-module-path*

22 *module-name* ::
23 *constant-identifier*

24 *top-module-path* ::
25 **::** *module-name*

26 *nested-module-path* ::
27 *primary-expression* [no line-terminator here] **::** *module-name*

1 *module-body* ::
2 *body-statement*

3 **A.5.0.2 Class definition**

4 see §13.2.2

5 *class-definition* ::
6 **class** *class-path* [no *line-terminator* here] *superclass class-body* **end**

7 *class-path* ::
8 *top-class-path*
9 | *class-name*
10 | *nested-class-path*

11 *class-name* ::
12 *constant-identifier*

13 *top-class-path* ::
14 :: *class-name*

15 *nested-class-path* ::
16 *primary-expression* [no *line-terminator* here] :: *class-name*

17 *superclass* ::
18 *separator*
19 | < *expression separator*

20 *class-body* ::
21 *body-statement*

22 **A.5.0.3 Method definition**

23 see §13.3.1

24 *method-definition* ::
25 **def** *method-name* [no *line-terminator* here] *method-parameter-part*
26 *method-body* **end**

27 *method-name* ::
28 *method-identifier*
29 | *operator-method-name*
30 | *reserved-word*

1 *method-body* ::
2 *body-statement*

3 **A.5.0.4 Method parameters**

4 see §13.3.2

5 *method-parameter-part* ::
6 (*parameter-list?*)
7 | *parameter-list?* *separator*

8 *parameter-list* ::
9 *mandatory-parameter-list* , *optional-parameter-list?* ,
10 *array-parameter?* , *block-parameter?*
11 | *optional-parameter-list* , *array-parameter?* , *block-parameter?*
12 | *array-parameter* , *block-parameter?*
13 | *block-parameter*

14 *mandatory-parameter-list* ::
15 *mandatory-parameter*
16 | *mandatory-parameter-list* , *mandatory-parameter*

17 *mandatory-parameter* ::
18 *local-variable-identifier*

19 *optional-parameter-list* ::
20 *optional-parameter*
21 | *optional-parameter-list* , *optional-parameter*

22 *optional-parameter* ::
23 *optional-parameter-name* = *default-parameter-expression*

24 *optional-parameter-name* ::
25 *local-variable-identifier*

26 *default-parameter-expression* ::
27 *operator-expression*

28 *array-parameter* ::
29 * *array-parameter-name*
30 | *

31 *array-parameter-name* ::
32 *local-variable-identifier*

1 *block-parameter* ::
2 & *block-parameter-name*

3 *block-parameter-name* ::
4 *local-variable-identifier*

5 **A.5.0.5 The alias statement**

6 see §13.3.6

7 *alias-statement* ::
8 **alias** *new-name* *aliased-name*

9 *new-name* ::
10 *method-name*
11 | *symbol*

12 *aliased-name* ::
13 *method-name*
14 | *symbol*

15 **A.5.0.6 The undef statement**

16 see §13.3.7

17 *undef-statement* ::
18 **undef** *undef-list*

19 *undef-list* ::
20 *method-name-or-symbol* (, *method-name-or-symbol*)*

21 *method-name-or-symbol* ::
22 *method-name*
23 | *symbol*

24 **A.5.0.7 Eigenclass definition**

25 see §13.4.2

26 *eigenclass-definition* ::
27 **class** << *expression* *separator* *eigenclass-body* **end**

1 *eigenclass-body* ::
2 *body-statement*

3 **A.5.0.7.1 Patterns**

4 see §15.2.15.3

5 *pattern* ::
6 *alternative*₁
7 | *pattern*₁ | *alternative*₂

8 *alternative* ::
9 [*empty*]
10 | *alternative*₃ *term*

11 *term* ::
12 *anchor*
13 | *atom*₁
14 | *atom*₂ *quantifier*

15 *anchor* ::
16 ^ | \$

17 *quantifier* ::
18 * | + | ?

19 *atom* ::
20 *pattern-character*
21 | *grouping*
22 | .
23 | *atom-escape-sequence*

24 *pattern-character* ::
25 *source-character* **but not** *regexp-meta-character*

26 *regexp-meta-character* ::
27 | | . | * | + | ^ | ? | (|)
28 | *future-reserved-meta-character*

29 *future-reserved-meta-character* ::
30 [|] | { | }

31 *grouping* ::
32 (*pattern*)

```

1  atom-escape-sequence ::
2      decimal-escape-sequence
3      | regexp-character-escape-sequence

4  decimal-escape-sequence ::
5      \ decimal-digit-without-zero

6  regexp-character-escape-sequence ::
7      regexp-escape-sequence
8      | regexp-non-escaped-sequence
9      | hex-escape-sequence
10     | regexp-octal-escape-sequence
11     | regexp-control-escape-sequence

12 regexp-escape-sequence ::
13     \ regexp-escaped-character

14 regexp-escaped-character ::
15     n | t | r | f | v | a | e | b

16 regexp-non-escaped-sequence ::
17     \ regexp-non-escaped-character

18 regexp-non-escaped-character ::
19     source-character but not regexp-escaped-character

20 regexp-octal-escape-sequence ::
21     octal-escape-sequence but not decimal-escape-sequence

22 regexp-control-escape-sequence ::
23     \ ( C - | c ) regexp-control-escaped-character

24 regexp-control-escaped-character ::
25     regexp-character-escape-sequence
26     | ?
27     | source-character but not ( \ | ? )

```
