The Ruby Programming Language

CHAPTER 3

Datatypes and Objects

3.1.1 Integer Literals

```
0
123
12345678901234567890

1_000_000_000     # One billion (or 1,000 million in the UK)

0377           # Octal representation of 255
0b1111_1111    # Binary representation of 255
0xFF           # Hexadecimal representation of 255
```

3.1.2 Floating-Point Literals

```
0.0
-3.14
6.02e23       # This means 6.02 &#xD7; 10<superscript>23</superscript>
1_000_000.01  # One million and a little bit more
```

3.1.3 Arithmetic in Ruby

```
x = 5/2     # result is 2
y = 5.0/2   # result is 2.5
z = 5/2.0   # result is 2.5

x = 5%2     # result is 1

x = 1.5%0.4   # result is 0.3

x**4          # This is the same thing as x*x*x*x
x**-1         # The same thing as 1/x
x**(1/3.0)    # The cube root of x
x**(1/4)      # Oops! Integer division means this is x**0, which is always 1
x**(1.0/4.0)  # This is the fourth-root of x

even = (x[0] == 0)  # A number is even if the least-significant bit is 0
```

3.1.4 Binary Floating-Point and Rounding Errors

```
0.4 - 0.3 == 0.1    # Evaluates to false in most implementations
```

3.2.1.1 Single-quoted string literals

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

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

'This string literal ends with a single backslash: \\'
'This is a backslash-quote: \\\''
'Two backslashes: \\\\'

'a\b' == 'a\\b'

'This is a long string literal \
that includes a backslash and a newline'

message =
'These three literals are '\
'concatenated into one by the interpreter. '\
'The resulting string contains no newlines.'
```

3.2.1.2 Double-quoted string literals

```
"\t\"This quote begins with a tab and ends with a newline\"\n"
"\\"  # A single backslash

"360 degrees=#{2*Math::PI} radians" # "360 degrees=6.28318530717959 radians"

$salutation = 'hello'     # Define a global variable

"My phone #: 555-1234"                # No escape needed
"Use \#{ to interpolate expressions"  # Escape #{ with backslash

sprintf("pi is about %.4f", Math::PI) # Returns "pi is about 3.1416"

"pi is about %.4f" % Math::PI # Same as example above
"%s: %f" % ["pi", Math::PI]   # Array on righthand side for multiple args

"This string literal
has two lines \
but is written on three"
```

```
"This string has three lines.\r\n" \
"It is written as three adjacent literals\r\n" \
"separated by escaped newlines\r\n"
```

3.2.1.3 Unicode escapes

```
"\u00D7"       # => "&#xD7;": leading zeros cannot be dropped
"\u20ac"       # => "&#x20AC;": lowercase letters are okay

"\u{A5}"      # => "&#xA5;": same as "\u00A5"
"\u{3C0}"     # Greek lowercase pi: same as "\u03C0"
"\u{10ffff}"  # The largest Unicode codepoint

money = "\u{20AC A3 A5}"  # => "&#x20AC;&#xA3;&#xA5;"

money = "\u{20AC 20 A3 20 A5}"  # => "&#x20AC; &#xA3; &#xA5;"
```

3.2.1.4 Arbitrary delimiters for string literals

```
%q(Don't worry about escaping ' characters!)
%Q|"How are you?", he said|
%-This string literal ends with a newline\n-  # Q omitted in this one

%q_This string literal contains \_underscores\__
%Q!Just use a _different_ delimiter\!!

# XML uses paired angle brackets:
%<<book><title>Ruby in a Nutshell</title></book>>  # This works
# Expressions use paired, nested parens:
%((1+(2*3)) = #{(1+(2*3))})                        # This works, too
%(A mismatched paren \( must be escaped)           # Escape needed here
```

3.2.1.5 Here documents

```
document = <<HERE        # This is how we begin a here document
This is a string literal.
It has two lines and abruptly ends...
HERE

greeting = <<HERE + <<THERE + "World"
Hello
HERE
There
THERE
```

```
empty = <<END
END

document = <<'THIS IS THE END, MY ONLY FRIEND, THE END'
    .
    . lots and lots of text goes here
    . with no escaping at all.
    .
THIS IS THE END, MY ONLY FRIEND, THE END

document = <<-"# # #"    # This is the only place we can put a comment
<html><head><title>#{title}</title></head>
<body>
<h1>#{title}</h1>
#{body}
</body>
</html>
# # #
```

3.2.1.6 Backtick command execution

```
`ls`

%x[ls]

if windows
  listcmd = 'dir'
else
  listcmd = 'ls'
end
listing = `#{listcmd}`

listing = Kernel.`(listcmd)
```

3.2.1.7 String literals and mutability

```
10.times { puts "test".object_id }
```

3.2.2 Character Literals

```
?A   # Character literal for the ASCII character A
?"   # Character literal for the double-quote character
??   # Character literal for the question mark character
```

```
?\u20AC == ?&#x20AC;     # => true: Ruby 1.9 only
?&#x20AC; == "\u20AC"    # => true

?\t      # Character literal for the TAB character
?\C-x    # Character literal for Ctrl-X
?\111    # Literal for character whose encoding is 0111 (octal)

planet = "Earth"
"Hello" + " " + planet    # Produces "Hello Earth"

"Hello planet #" + planet_number.to_s  # to_s converts to a string

"Hello planet ##{planet_number}"

greeting = "Hello"
greeting << " " << "World"
puts greeting   # Outputs "Hello World"

alphabet = "A"
alphabet << ?B   # Alphabet is now "AB"
alphabet << 67   # And now it is "ABC"
alphabet << 256  # Error in Ruby 1.8: codes must be >=0 and < 256

ellipsis = '.'*3    # Evaluates to '...'

a = 0;
"#{a=a+1} " * 3   # Returns "1 1 1 ", not "1 2 3 "
```

3.2.3 String Operators

```
planet = "Earth"
"Hello" + " " + planet    # Produces "Hello Earth"

"Hello planet #" + planet_number.to_s  # to_s converts to a string

"Hello planet ##{planet_number}"

greeting = "Hello"
greeting << " " << "World"
puts greeting   # Outputs "Hello World"

alphabet = "A"
alphabet << ?B   # Alphabet is now "AB"
alphabet << 67   # And now it is "ABC"
alphabet << 256  # Error in Ruby 1.8: codes must be >=0 and < 256

ellipsis = '.'*3    # Evaluates to '...'
```

```
a = 0;
"#{a=a+1} " * 3   # Returns "1 1 1 ", not "1 2 3 "
```

3.2.4 Accessing Characters and Substrings

```
s = 'hello';   # Ruby 1.8
s[0]           # 104: the ASCII character code for the first character 'h'
s[s.length-1]  # 111: the character code of the last character 'o'
s[-1]          # 111: another way of accessing the last character
s[-2]          # 108: the second-to-last character
s[-s.length]   # 104: another way of accessing the first character
s[s.length]    # nil: there is no character at that index

s = 'hello';   # Ruby 1.9
s[0]           # 'h': the first character of the string, as a string
s[s.length-1]  # 'o': the last character 'o'
s[-1]          # 'o': another way of accessing the last character
s[-2]          # 'l': the second-to-last character
s[-s.length]   # 'h': another way of accessing the first character
s[s.length]    # nil: there is no character at that index

s[0] = ?H        # Replace first character with a capital H
s[-1] = ?O       # Replace last character with a capital O
s[s.length] = ?! # ERROR! Can't assign beyond the end of the string

s = "hello"      # Begin with a greeting
s[-1] = ""       # Delete the last character; s is now "hell"
s[-1] = "p!"     # Change new last character and add one; s is now "help!"

s = "hello"
s[0,2]          # "he"
s[-1,1]         # "o": returns a string, not the character code ?o
s[0,0]          # "": a zero-length substring is always empty
s[0,10]         # "hello": returns all the characters that are available
s[s.length,1]   # "": there is an empty string immediately beyond the end
s[s.length+1,1] # nil: it is an error to read past that
s[0,-1]         # nil: negative lengths don't make any sense

s = "hello"
s[0,1] = "H"              # Replace first letter with a capital letter
s[s.length,0] = " world"  # Append by assigning beyond the end of the string
s[5,0] = ","              # Insert a comma, without deleting anything
s[5,6] = ""               # Delete with no insertion; s == "Hellod"

s = "hello"
s[2..3]          # "ll": characters 2 and 3
s[-3..-1]        # "llo": negative indexes work, too
s[0..0]          # "h": this Range includes one character index
s[0...0]         # "": this Range is empty
```

```
s[2..1]           # "": this Range is also empty
s[7..10]          # nil: this Range is outside the string bounds
s[-2..-1] = "p!"     # Replacement: s becomes "help!"
s[0...0] = "Please " # Insertion: s becomes "Please help!"
s[6..10] = ""        # Deletion: s becomes "Please!"

s = "hello"       # Start with the word "hello"
while(s["l"])     # While the string contains the substring "l"
  s["l"] = "L";   # Replace first occurrence of "l" with "L"
end               # Now we have "heLLo"

s[/[aeiou]/] = '*'      # Replace first vowel with an asterisk
```

3.2.5 Iterating Strings

```
s = "&#xA5;1000"
s.each_char {|x| print "#{x} " }         # Prints "&#xA5; 1 0 0 0". Ruby 1.9
0.upto(s.size-1) {|i| print "#{s[i]} "}  # Inefficient with multibyte chars
```

3.2.6.1 Multibyte characters in Ruby 1.9

```
# -*- coding: utf-8 -*-   # Specify Unicode UTF-8 characters

# This is a string literal containing a multibyte multiplication character
s = "2&#xD7;2=4"

# The string contains 6 bytes which encode 5 characters
s.length        # => 5: Characters:  '2'   '&#xD7;'   '2'   '='   '4'
s.bytesize      # => 6: Bytes (hex): 32   c3 97  32    3d    34

# -*- coding: utf-8 -*-
s = "2&#xD7;2=4"     # Note multibyte multiplication character
s.encoding      # => <Encoding: UTF-8>
t = "2+2=4"     # All characters are in the ASCII subset of UTF-8
t.encoding      # => <Encoding: ASCII-8BIT>

text = stream.readline.force_encoding("utf-8")
bytes = text.dup.force_encoding(nil)   # nil encoding means binary

s = "\xa4".force_encoding("utf-8")  # This is not a valid UTF-8 string
s.valid_encoding?                   # => false

# -*- coding: utf-8 -*-
euro1 = "\u20AC"                    # Start with the Unicode Euro character
puts euro1                          # Prints "&#x20AC;"
euro1.encoding                      # => <Encoding:UTF-8>
euro1.bytesize                      # => 3
```

```
euro2 = euro1.encode("iso-8859-15")  # Transcode to Latin-15
puts euro2.inspect                   # Prints "\xA4"
euro2.encoding                       # => <Encoding:iso-8859-15>
euro2.bytesize                       # => 1

euro3 = euro2.encode("utf-8")        # Transcode back to UTF-8
euro1 == euro3                       # => true

# Interpret a byte as an iso-8859-15 codepoint, and transcode to UTF-8
byte = "\xA4"
char = byte.encode("utf-8", "iso-8859-15")

text = bytes.encode(to, from)
text = bytes.dup.force_encoding(from).encode(to)

# The iso-8859-1 encoding doesn't have a Euro sign, so this raises an
exception
"\u20AC".encode("iso-8859-1")
```

3.2.6.2 The Encoding class

```
Encoding::ASCII_8BIT     # Also ::BINARY
Encoding::UTF_8          # UTF-8-encoded Unicode characters
Encoding::EUC_JP         # EUC-encoded Japanese
Encoding::SHIFT_JIS      # Japanese: also ::SJIS, ::WINDOWS_31J, ::CP932

encoding = Encoding.find("utf-8")
```

3.2.6.3 Multibyte characters in Ruby 1.8

```
$KCODE = "u"        # Specify Unicode UTF-8, or start Ruby with -Ku option
require "jcode"     # Load multibyte character support

mb = "2\303\2272=4" # This is "2&#xD7;2=4" with a Unicode multiplication sign
mb.length           # => 6: there are 6 bytes in this string
mb.jlength          # => 5: but only 5 characters
mb.mbchar?          # => 1: position of the first multibyte char, or nil
mb.each_byte do |c| # Iterate through the bytes of the string.
  print c, " "      # c is Fixnum
end                 # Outputs "50 195 151 50 61 52 "
mb.each_char do |c| # Iterate through the characters of the string
  print c, " "      # c is a String with jlength 1 and variable length
end                 # Outputs "2 &#xD7; 2 = 4 "
```

3.3 Arrays

```
[1, 2, 3]         # An array that holds three Fixnum objects
[-10...0, 0..10,] # An array of two ranges; trailing commas are allowed
[[1,2],[3,4],[5]] # An array of nested arrays
[x+y, x-y, x*y]   # Array elements can be arbitrary expressions
[]                # The empty array has size 0

words = %w[this is a test]  # Same as: ['this', 'is', 'a', 'test']
open = %w| ( [ { < |        # Same as: ['(', '[', '{', '<']
white = %W(\s \t \r \n)     # Same as: ["\s", "\t", "\r", "\n"]

empty = Array.new       # []: returns a new empty array
nils = Array.new(3)     # [nil, nil, nil]: new array with 3 nil elements
zeros = Array.new(4, 0) # [0, 0, 0, 0]: new array with 4 0 elements
copy = Array.new(nils)  # Make a new copy of an existing array
count = Array.new(3) {|i| i+1}  # [1,2,3]: 3 elements computed from index

a = [0, 1, 4, 9, 16]   # Array holds the squares of the indexes
a[0]        # First element is 0
a[-1]       # Last element is 16
a[-2]       # Second to last element is 9
a[a.size-1] # Another way to query the last element
a[-a.size]  # Another way to query the first element
a[8]        # Querying beyond the end returns nil
a[-8]       # Querying before the start returns nil, too

a[0] = "zero"   # a is ["zero", 1, 4, 9, 16]
a[-1] = 1..16   # a is ["zero", 1, 4, 9, 1..16]
a[8] = 64       # a is ["zero", 1, 4, 9, 1..16, nil, nil, nil, 64]
a[-9] = 81      # Error: can't assign before the start of an array

a = ('a'..'e').to_a   # Range converted to ['a', 'b', 'c', 'd', 'e']
a[0,0]                # []: this subarray has zero elements
a[1,1]                # ['b']: a one-element array
a[-2,2]               # ['d','e']: the last two elements of the array
a[0..2]               # ['a', 'b', 'c']: the first three elements
a[-2..-1]             # ['d','e']: the last two elements of the array
a[0...-1]             # ['a', 'b', 'c', 'd']: all but the last element

a[0,2] = ['A', 'B']      # a becomes ['A', 'B', 'c', 'd', 'e']
a[2...5]=['C', 'D', 'E'] # a becomes ['A', 'B', 'C', 'D', 'E']
a[0,0] = [1,2,3]         # Insert elements at the beginning of a
a[0..2] = []             # Delete those elements
a[-1,1] = ['Z']          # Replace last element with another
a[-1,1] = 'Z'            # For single elements, the array is optional
a[-2,2] = nil  # Delete last 2 elements in 1.8; replace with nil in 1.9

a = [1, 2, 3] + [4, 5]    # [1, 2, 3, 4, 5]
a = a + [[6, 7, 8]]       # [1, 2, 3, 4, 5, [6, 7, 8]]
a =
```

```
a + 9                 # Error: righthand side must be an array

['a', 'b', 'c', 'b', 'a'] - ['b', 'c', 'd']    # ['a', 'a']

a = []        # Start with an empty array
a << 1        # a is [1]
a << 2 << 3   # a is [1, 2, 3]
a << [4,5,6]  # a is [1, 2, 3, [4, 5, 6]]

a = [0] * 8    # [0, 0, 0, 0, 0, 0, 0, 0]

a = [1, 1, 2, 2, 3, 3, 4]
b = [5, 5, 4, 4, 3, 3, 2]
a | b    # [1, 2, 3, 4, 5]: duplicates are removed
b | a    # [5, 4, 3, 2, 1]: elements are the same, but order is different
a & b    # [2, 3, 4]
b & a    # [4, 3, 2]

a = ('A'..'Z').to_a    # Begin with an array of letters
a.each {|x| print x }  # Print the alphabet, one letter at a time
```

3.4 Hashes

```
# This hash will map the names of digits to the digits themselves
numbers = Hash.new     # Create a new, empty, hash object
numbers["one"] = 1     # Map the String "one" to the Fixnum 1
numbers["two"] = 2     # Note that we are using array notation here
numbers["three"] = 3

sum = numbers["one"] + numbers["two"]  # Retrieve values like this
```

3.4.1 Hash Literals

```
numbers = { "one" => 1, "two" => 2, "three" => 3 }

numbers = { :one => 1, :two => 2, :three => 3 }

numbers = { :one, 1, :two, 2, :three, 3 } # Same, but harder to read

numbers = { :one => 1, :two => 2, } # Extra comma ignored

numbers = { one: 1, two: 2, three: 3 }
```

3.5 Ranges

```
1..10      # The integers 1 through 10, including 10
1.0...10.0 # The numbers between 1.0 and 10.0, excluding 10.0 itself

cold_war = 1945..1989
cold_war.include? birthdate.year

r = 'a'..'c'
r.each {|l| print "[#{l}]"}     # Prints "[a][b][c]"
r.step(2) { |l| print "[#{l}]"} # Prints "[a][c]"
r.to_a                          # => ['a','b','c']: Enumerable defines to_a

1..3.to_a    # Tries to call to_a on the number 3
(1..3).to_a  # => [1,2,3]
```

3.5.1 Testing Membership in a Range

```
begin <= x <= end

begin <= x < end

r = 0...100      # The range of integers 0 through 99
r.member? 50     # => true: 50 is a member of the range
r.include? 100   # => false: 100 is excluded from the range
r.include? 99.9  # => true: 99.9 is less than 100

triples = "AAA".."ZZZ"
triples.include? "ABC"        # true; fast in 1.8 and slow in 1.9
triples.include? "ABCD"       # true in 1.8, false in 1.9
triples.cover?   "ABCD"       # true and fast in 1.9
triples.to_a.include? "ABCD"  # false and slow in 1.8 and 1.9
```

3.6 Symbols

```
:symbol                  # A Symbol literal
:"symbol"                # The same literal
:'another long symbol'   # Quotes are useful for symbols with spaces
s = "string"
sym = :"#{s}"            # The Symbol :string

%s["]     # Same as :'"'

o.respond_to? :each

name = :size
if o.respond_to? name
  o.send(name)
```

```
end

str = "string"     # Begin with a string
sym = str.intern   # Convert to a symbol
sym = str.to_sym   # Another way to do the same thing
str = sym.to_s     # Convert back to a string
str = sym.id2name  # Another way to do it
```

3.7 True, False, and Nil

```
o == nil   # Is o nil?
o.n
il?     # Another way to test
```

3.8.1 Object References

```
s = "Ruby" # Create a String object. Store a reference to it in s.
t = s      # Copy the reference to t. s and t both refer to the same object.
t[-1] = "" # Modify the object through the reference in t.
print s    # Access the modified object through s. Prints "Rub".
t = "Java" # t now refers to a different object.
print s,t  # Prints "RubJava".
```

3.8.2 Object Lifetime

```
myObject = myClass.new
```

3.8.4 Object Class and Object Type

```
o = "test"  # This is a value
o.class     # Returns an object representing the String class

o.class                       # String: o is a String object
o.class.superclass            # Object: superclass of String is Object
o.class.superclass.superclass # nil: Object has no superclass

# Ruby 1.9 only
Object.superclass             # BasicObject: Object has a superclass in 1.9
BasicObject.superclass        # nil: BasicObject has no superclass

o.class == String       # true if is o a String
```

```
o.instance_of? String   # true if o is a String

x = 1                    # This is the value we're working with
x.instance_of? Fixnum    # true: is an instance of Fixnum
x.instance_of? Numeric   # false: instance_of? doesn't check inheritance
x.is_a? Fixnum           # true: x is a Fixnum
x.is_a? Integer          # true: x is an Integer
x.is_a? Numeric          # true: x is a Numeric
x.is_a? Comparable       # true: works with mixin modules, too
x.is_a? Object           # true for any value of x

Numeric === x            # true: x is_a Numeric

o.respond_to? :"<<"  # true if o has an << operator

o.respond_to? :"<<" and not o.is_a? Numeric
```

3.8.5.1 The equal? method

```
a = "Ruby"       # One reference to one String object
b = c = "Ruby"   # Two references to another String object
a.equal?(b)      # false: a and b are different objects
b.equal?(c)      # true: b and c refer to the same object

a.object_id == b.object_id   # Works like a.equal?(b)
```

3.8.5.2 The == operator

```
a = "Ruby"    # One String object
b = "Ruby"    # A different String object with the same content
a.equal?(b)   # false: a and b do not refer to the same object
a == b        # true: but these two distinct objects have equal values
```

3.8.5.3 The eql? method

```
1 == 1.0    # true: Fixnum and Float objects can be ==
1.eql?(1.0) # false: but they are never eql!
```

3.8.5.4 The === operator

```
(1..10) === 5    # true: 5 is in the range 1..10
/\d+/ === "123"  # true: the string matches the regular expression
String === "s"   # true: "s" is an instance of the class String
:s === "s"       # true in Ruby 1.9
```

3.8.6 Object Order

```
1 <=> 5     # -1
5 <=> 5     # 0
9 <=> 5     # 1
"1" <=> 5   # nil: integers and strings are not comparable
< Less than
<= Less than or equal
== Equal
>= Greater than or equal
> Greater than
1.between?(0,10)  # true: 0 <= 1 <= 10

nan = 0.0/0.0;     # zero divided by zero is not-a-number
nan < 0            # false: it is not less than zero
nan > 0            # false: it is not greater than zero
nan == 0           # false: it is not equal to zero
nan == nan         # false: it is not even equal to itself!
nan.equal?(nan)    # this is true, of course
```

3.8.7.2 Implicit conversions

```
# Ruby 1.8 only
e = Exception.new("not really an exception")
msg = "Error: " + e  # String concatenation with an Exception
```

3.8.7.4 Arithmetic operator type coercions

```
1.1.coerce(1)      # [1.0, 1.1]: coerce Fixnum to Float
require "rational" # Use Rational numbers
r = Rational(1,3)  # One third as a Rational number
r.coerce(2)        # [Rational(2,1), Rational(1,3)]: Fixnum to Rational
```

3.8.7.5 Boolean type conversions

```ruby
if x != nil   # Expression "x != nil" returns true or false to the if
  puts x      # Print x if it is defined
end

if x       # If x is non-nil
  puts x   # Then print it
end
```

3.8.9 Marshaling Objects

```ruby
def deepcopy(o)
  Marshal.load(Marshal.dump(o))
end
```

3.8.10 Freezing Objects

```ruby
s = "ice"      # Strings are mutable objects
s.freeze       # Make this string immutable
s.frozen?      # true: it has been frozen
s.upcase!      # TypeError: can't modify frozen string
s[0] = "ni"    # TypeError: can't modify frozen string
```

3.8.11 Tainting Objects

```ruby
s = "untrusted"   # Objects are normally untainted
s.taint           # Mark this untrusted object as tainted
s.tainted?        # true: it is tainted
s.upcase.tainted? # true: derived objects are tainted
s[3,4].tainted?   # true: substrings are tainted
```