Programming Ruby 1.9: Chapter 6: Standard Types

Numbers, Strings, and Ranges!

2011-05-18

David Grayson

Based on Chapter 6 in

Programming Ruby 1.9 by Dave Thomas

Integers

- Example: 0, 1, -2, 23902931391239192O
- `Integer` class: any integer
  - `Fixnum` subclass: only -2^N through 2^N-1 where N is 62 or 30
  - `Bignum` subclass
- Automatic conversion to appropriate class!

```
(2**62).class      # => Bignum
(2**62-1).class    # => Fixnum
```

Integer Notation

```
123456          => 123456    # Fixnum
0d123456        => 123456    # Fixnum
123_456         => 123456    # Fixnum – underscore ignored
-44             => -44       # Fixnum – negative
0x1FFB          => 8187      # Fixnum – hexadecimal
0377            => 255       # Fixnum – octal
0b10_1010       => 42        # Fixnum – negative binary
```

Floating Point Numbers

- `Float` in Ruby = `double` in native architecture
- Notation:

```
34e1            => 340.0      Float
1.9521          => 1.9521     Float
```

Rational and Complex

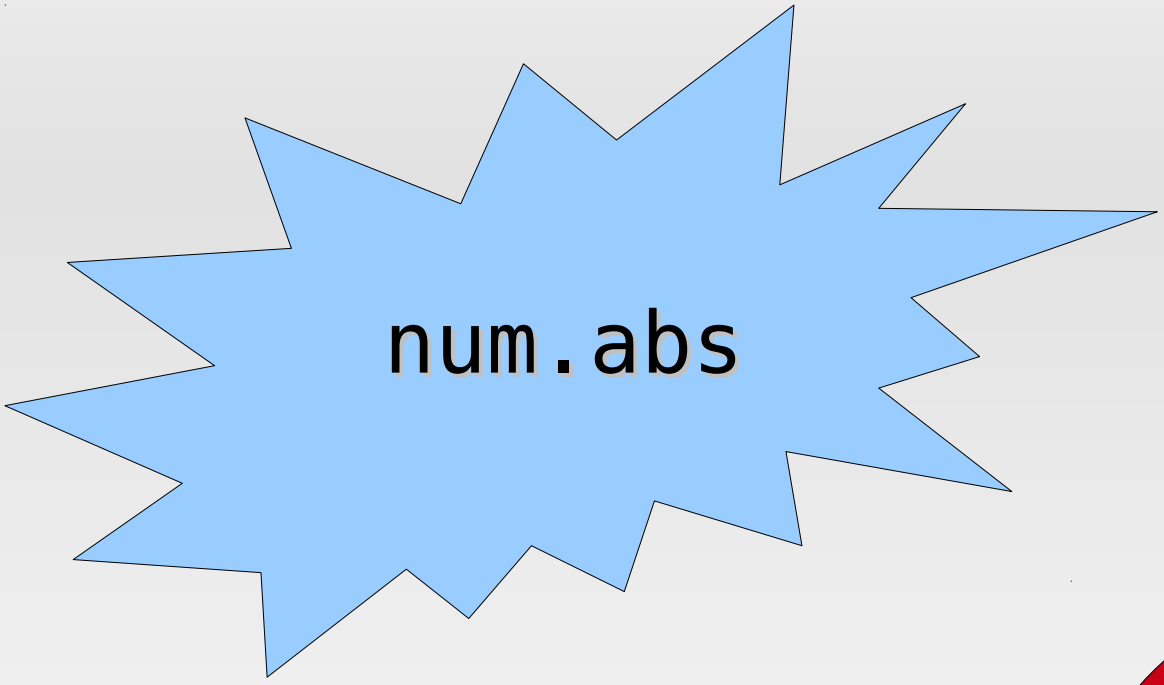
```
Rational(3, 4) * Rational(2, 3)    # => (1/2)
Rational("3/4") * Rational("2/3")  # => (1/2)

Complex(1, 2) * Complex(3, 4)      # => (-5+10i)
Complex("1+2i") * Complex("3+4i")  # => (-5+10i)
```

Automatic Simplification:

```
Rational(1, Rational(1, 6))  # => 6/1        Rational
Complex(0, Complex(0,1))     # => -1+0i      Complex
Rational(1, Complex(0,1))    # => 0/1-1/1*i  Complex!
```

Numeric functions



How Numbers Interact

1) Operations between numbers of same class
   => same class (usually)
2) Mix different classes => more general class
3) (Integer or Rational) + Float => Float

```
1 + 2                   # => 3
1 + 2.0                 # => 3.0
1.0 + Complex(1,2)      # => (2.0+2i)
1 + Rational(2,3)       # => 5/3
1.0 + Rational(2,3)     # => 1.6666666666666665

1 / 2.0                 # => 0.5
1 / 2                   # => 0     confusing?  use mathn
```

Looping Using Numbers

```
3.times { … }
1.upto(5) { |i| … }
99.downto(95) { |i| … }
50.step(80, 5) { |i| … }
10.downto(7).with_index { |num, index| … }
```

Strings

"Ruby strings are simply sequences of characters." – Section 6.2

In Ruby, a string consists of an arbitrary sequence of bytes (typically representing characters) along with an encoding.

String Notation

- Single-quoted strings support only two escape sequences:

```
'escape using "\\"'   # => esacpe using "\"
'That\'s right'       # => That's right
```

- Double-quoted strings have a boatload of escape sequences

Ruby code inside strings

```
"Seconds/day: #{24*60*60}"     # => Seconds/day: 86400
"Safe level is #$SAFE"         # => Save level is 0
"Stored in #@@table_name"
"Score: #@score"

"now is #{def the(a);'the '+a;end; the('time')} for fun"
  => now is the time for fun
```

Wacky String Literals

```
%q/single 'quoted'/             # => single 'quoted'
%Q!double "quoted"!             # => double "quoted"
%Q{Seconds/hour: #{60*60}}      # => Seconds/hour: 3600
%{Seconds/hour: #{60*60}}       # => Seconds/hour: 3600
```

- Delimeter can be almost any non-alphanumeric, single byte character:

```
%$#$SAFE$                       # => 0
%^_^                            # => _

puts %Q
Hello world
```

Here Documents

```
string = <<END
This is how you easily make
a many line string.
END

    string = <<-ENDER
With a dash, the ending
can be indented
    ENDER
```

Working with Strings

```
/01.mp3   | 3:45 | Fats     Waller     | Ain't Misbehavin'
/02.mp3   | 2:48 | Louis    Armstrong  | Wonderful World
/03.mp3   | 4:09 | Strength in Numbers | Texas Red
```

```
Song = Struct.new(:title, :name, :length)

songs = []
File.open("songdata") do |song_file|
  song_file.each do |line|
    file, length, name, title = line.chomp.split(/\s*\|\s*/)
    name.squeeze!(" ")
    mins, secs = length.scan(/\d+/)
    songs << Song.new(title, name, mins.to_i*60+secs.to_i)
  end
end

songs.each do |song|
  puts song
end
```

```
#<struct Song title="Ain't Misbehavin'", name="Fats Waller", length=225>
#<struct Song title="Wonderful World", name="Louis Armstrong", length=168>
#<struct Song title="Texas Red", name="Strength in Numbers", length=249>
```

Ranges

- Examples:
  - January to June
  - 0 to 9
  - rare to well done
  - lines 50 through 67

```
1..10     # includes endpoints, 1 ≤ x ≤ 10
1...10    # excludes upper endpoint, 1 ≤ x < 10
```

Ranges as Sequences

```
(1..5).to_a                 # => [1, 2, 3, 4, 5]
('bar'..'bat').to_a         # => ["bar", "bas", "bat"]
enum = ('bar'..'bat').to_enum
enum.next                   # => "bar"
enum.next                   # => "bas"


digits = (0..9)
digits.min                      # => 0
digits.max                      # => 9
digits.include?(5)              # => true
digits.reject {|i| i < 5}  # => [5, 6, 7, 8, 9]
digits.inject(:+)               # => 45
```

Ranges as Intervals

```
(1..10)    === 5         # => true
(1..10)    === 15        # => false
(1..10)    === 3.14159   # => true
('a'..'j') === 'c'       # => true
(Date.new(2011,7,2)..Date.new(2011,7,4))
    === Date.today       # => false
```

```
person_age = 16.30
case person_age
  when 16..17 then puts "You can drive now!"
  when 21..23 then puts "You can drink at PT's now!"
  else             puts "Hi."
end
```

Ranges as Conditions

```
while line = gets
  puts line if (line=~/start/)..(line=~/end/)
end
```

The End

For more information, see Chapter 6 in Programming Ruby 1.9 by Dave Thomas.