Ruby 初級者向けレッスン第 27 回

okkez @ Ruby 関西

2009 年 04 月 25 日

演習の解答例と解説です。
よくわからないことは http://doc.okkez.net/ にある、るりまを参照してください。

演習

自分でコマンドラインアプリケーションを作ってみましょう。使用するライブラリは自由です。
初級者の人は、演習 0 からやってください。そうでない方はどこからはじめても自由です。

演習 0

引数を受け取るコマンドラインアプリケーションを作ってください。

- 指定された二つの数値を用いて四則演算の実行した結果を表示するスクリプトを書いてください。
- ゼロから指定した数値までの和を計算した結果を表示するスクリプトを書いてください。
- optparse ライブラリを使って上のスクリプトを書いて見てください。

解答例

まずは一つ目。

```
01: #!/usr/bin/ruby
02:
03: a = ARGV[0].to_i
04: b = ARGV[1].to_i
05:
06: puts a + b
07: puts a - b
08: puts a * b
09: puts a / b
```

これは簡単ですね。
知らなければハマるポイントはいくつかあります。

- ARGV が配列であること
- ARGV に入っている値は全て文字列であること

以上を踏まえて解答例を見ると String#to_i や ARGV[0] などを使用している理由がわかると思います。

二つ目。

```
01: #!/usr/bin/ruby
02:
03: puts (0..ARGV[0]).inject(:+)
```

これは中級者以上にはお馴染みの Enumerable#inject を使用してみました。Enumerable#inject については http://doc.okkez.net/187/view/method/Enumerable/i/inject を参照してください。

その他の解答例もありますが、ここでは割愛します。

上記の問題を optparse ライブラリを使用して実装してみた解答例です。

```
01: #!/usr/bin/ruby
02:
03: require 'optparse'
04:
05: def main
06:   a = b = nil
07:   op = '+'
08:   parser = OptionParser.new
09:   parser.on('-l', '--left-value=NUM', 'The left value') do |n|
10:     a = n.to_i
11:   end
12:   parser.on('-r', '--right-value=NUM', 'The right value') do |n|
13:     b = n.to_i
14:   end
15:   parser.on('-o', '--operator=[OP]', 'Operator +,-,*,/') do |o|
16:     op = o
17:   end
18:
19:   begin
20:     parser.parse!
21:   rescue => ex
22:     $stderr.puts ex.message
23:     $stderr.puts parser.help
24:     exit(1)
25:   end
26:
27:   if a.nil? or b.nil?
28:     $stderr.puts 'you must specify both -l and -r options'
```

```
29:     $stderr.puts parser.help
30:     exit(1)
31:   end
32:
33:   unless %w[+ - * /].include?(op)
34:     $stderr.puts "invalid operator: #{op}"
35:     $stderr.puts parser.help
36:     exit(1)
37:   end
38:
39:   puts a.__send__(op, b)
40: end
41:
42: if __FILE__ == $0
43:   main
44: end
```

例なのでこんな感じです。四則演算以外のことが出来ないようにしてあります。

optparse ライブラリについては http://doc.okkez.net/187/view/library/optparse を参照してください。

もう一つの問題でも optparse ライブラリを使用してみました。

```
01: #!/usr/bin/ruby
02:
03: require 'optparse'
04:
05: def main
06:   min = 0
07:   max = nil
08:   parser = OptionParser.new
09:   parser.on('--min=[NUM]', 'min value (default is 0)') do |n|
10:     min = n.to_i
11:   end
12:   parser.on('--max=NUM', 'max value') do |n|
13:     max = n.to_i
14:   end
15:
16:   def parser.error(message)
17:     $stderr.puts message
18:     $stderr.puts self.help
19:     exit(1)
20:   end
21:
22:   begin
```

```
23:     parser.parse!
24:   rescue => ex
25:     parser.error(ex.message)
26:   end
27:
28:   if max.nil?
29:     parser.error('must specify max value')
30:   end
31:
32:   if max < min
33:     parser.error('min value must be smaller then max value')
34:   end
35:
36:   puts (min..max).inject(:+)
37: end
38:
39: if __FILE__ == $0
40:   main
41: end
```

ゼロから指定された値までの和を求めるだけでは芸がなさすぎるので、指定された最小値から最大値までの和を計算するように格調してみました。

実質のロジックは一行だけで、他の部分はオプションの登録や実行前のチェックです。16 行目のメソッド定義は特異メソッドを定義しています。特異メソッドについては http://doc.okkez.net/187/view/spec/def を参照してください。ここでは、ある特定のオブジェクトでだけ使えるメソッドを定義したと考えておけば十分です。一つ前の解答例と、エラー処理の部分のコードを見比べてください。

演習 1

引数をいくつか取るコマンドラインアプリケーションを作ってください。以下の仕様を満たすようにしてください。

- CSV ファイルを分割します。分割数 n を指定できます。
- 読み込むファイル名を指定できます。
- 出力先ディレクトリを指定できます。
- 出力ファイル名のプレフィクスを指定できます。

解答例

分割方法はいくつか考えられますが、CSV ファイルなので行単位で分割することにしました。また、標準添付の csv ライブラリは遅い上にバグもあるので fastercsv gem をインストールして使用することにしました。たぶん、この問題の範囲ではバグが問題になることは無いと思いますが。

```
01: #!/usr/bin/ruby
02:
03: require 'rubygems'
04: require 'fastercsv'
05:
06: require 'optparse'
07:
08: def main
09:   num = nil
10:   input_file = nil
11:   output_dir = nil
12:   prefix = 'output'
13:   parser = OptionParser.new
14:   parser.on('-n', '--split-num=NUM', 'split NUM files') do |n|
15:     num = n.to_i
16:   end
17:   parser.on('-i', '--input-file=FILENAME', 'specify input file name') do |path|
18:     input_file = path
19:   end
20:   parser.on('-o', '--output-dir=DIRNAME', 'specify output directory name') do |path|
21:     output_dir = path
22:   end
23:   parser.on('-p', '--prefix=PREFIX', 'prefix of output files') do |str|
24:     prefix = str
25:   end
26:
27:   def parser.error(message)
28:     $stderr.puts message
29:     $stderr.puts self.help
30:     exit(1)
31:   end
32:
33:   begin
34:     parser.parse!
35:   rescue => ex
36:     parser.error(ex.message)
37:   end
38:
39:   if num.nil?
40:     parser.error('no --split-num')
41:   end
42:
43:   if input_file.nil?
```

```
44:     parser.error('no input file')
45:   end
46:
47:   if output_dir.nil?
48:     parser.error('no output file')
49:   end
50:
51:   array = FasterCSV.read(input_file)
52:   div, mod = array.size.divmod(num)
53:   array.each_slice(div).with_index{|v, i|
54:     FasterCSV.open("#{output_dir}/#{prefix}#{i}.csv", 'w+') do |csv|
55:       v.each do |row|
56:         csv << row
57:       end
58:     end
59:   }
60: end
61:
62: if __FILE__ == $0
63:   main
64: end
```

ほとんど fastercsv の使い方になるので、気になる人は fastercsv のリファレンスを参照してください。

演習 2

cat コマンドを実装してください。実装するオプションは、お任せします。
see also cat(1)

解答例

この解答例は以下の cat のバージョンに動作を大体合わせています。

```
$ cat --version
cat (GNU coreutils) 7.2
```

この解答例は 1.8.7, 1.9.1 で動くはずです。

```
01: #!/usr/bin/ruby
02: # -*- coding: utf-8 -*-
03:
04: require 'optparse'
05:
```

```
06: def main
07:   options = {
08:     :number_nonblank  => false,
09:     :number           => false,
10:     :squeeze_blank    => false,
11:     :show_nonprinting => false,
12:     :binary           => false,
13:     :show_ends        => false,
14:     :show_tabs        => false,
15:   }
16:   parser = OptionParser.new
17:   parser.on('-b', '--number-nonblank', ' 空白でない行に番号を付ける。初めの行を
1 行目とする。') do |v|
18:     options[:number_nonblank] = true
19:   end
20:   parser.on('-e', "‘ -vE ’ と同じ。") do |v|
21:     options[:show_nonprinting] = true
22:     options[:show_ends] = true
23:   end
24:   parser.on('-n', '--number', ' すべての行に番号を付ける。初めの行を 1 行目とする。
') do |v|
25:     options[:number] = true
26:   end
27:   parser.on('-s', '--squeeze-blank', "連続した空白行を、1 つの空白行にまとめる。") do |v|
28:     options[:squeeze_blank] = true
29:   end
30:   parser.on('-t', "-vT ’ と同じ。") do |v|
31:     options[:show_nonprinting] = true
32:     options[:show_tabs] = true
33:   end
34:   parser.on('-u', "何もしない。 Unix との互換性のために存在する。") do |v|
35:     # do nothing
36:   end
37:   parser.on('-v', '--show-nonprinting', <<-STR) do |v|
38: <LFD> と <TAB> とを除く制御文字を ‘ ^ ’ 表記を使って表示する。高位
39: ビットがセットされている文字は、前に ‘ M- ’ を置いて表わす。
40:     STR
41:     options[:show_nonprinting] = true
42:   end
43:   parser.on('-A', '--show-all', "‘ -vET ’ と同じ。") do |v|
44:     options[:show_nonprinting] = true
45:     options[:show_ends] = true
46:     options[:show_tabs] = true
```

```
47:   end
48:   parser.on('-B', '--binary', <<-STR) do ||
49: DOS プラットフォームのみ。ファイルの読み書きをバイナリモードで行う。
50:     STR
51:     options[:binary] = true
52:   end
53:   parser.on('-E', '--show-ends', "各行の最後に ‘ $ ’ を表示する。") do |v|
54:     options[:show_ends] = true
55:   end
56:   parser.on('-T', '--show-tabs', "<TAB> 文字を ‘ ^I ’ と表示する。") do |v|
57:     options[:show_tabs] = true
58:   end
59:   parser.on('--help', '標準出力に使用方法のメッセージを出力して正常終了する。') do
60:     puts parser.help
61:     exit(0)
62:   end
63:   parser.on('--version', '標準出力にバージョン情報を出力して正常終了する。') do
64:     parser.version = '0.0.1'
65:     puts parser.version
66:     exit(0)
67:   end
68:
69:   def parser.error(message)
70:     $stderr.puts message
71:     $stderr.puts self.help
72:     exit(1)
73:   end
74:
75:   begin
76:     parser.parse!
77:   rescue => ex
78:     parser.errpr(ex.message)
79:   end
80:
81:   str = ARGF.read
82:   [
83:    :squeeze_blank,
84:    :number_nonblank,
85:    :number,
86:    :show_nonprinting,
87:    :show_ends,
88:    :show_tabs,
89:    :binary,
```

```
90:   ].each do |m|
91:     next if m == :number && options[:number_nonblank]
92:     str = __send__(m, str) if options[m]
93:   end
94:   print str
95: end
96:
97: def number_nonblank(str)
98:   num = 0
99:   s = str.lines.to_a.size.to_s.size
100:   str.lines.map{|line|
101:     if /^$/ === line
102:       line
103:     else
104:       num += 1
105:       "\t%0#{s}d\t%s" % [num, line]
106:     end
107:   }.join
108: end
109:
110: def number(str)
111:   s = str.lines.to_a.size.to_s.size
112:   str.lines.with_index.map{|line, i|
113:     "\t%0#{s}d\t%s" % [i+1, line]
114:   }.join
115: end
116:
117: def squeeze_blank(str)
118:   str.gsub(/(:?\n){2,}/, "\n\n")
119: end
120:
121: # 0x00 - 0x1F (+ 0x40) see ascii(7)
122: def show_nonprinting(str)
123:   (0x00..0x1f).each do |v|
124:     next if v == 0x09 # \t
125:     next if v == 0x0a # \n
126:     str = str.gsub(/#{v.chr}/, "^#{(v + 0x40).chr}")
127:   end
128:   str
129: end
130:
131: def binay(str)
132:   # do nothing
```

```
133: end
134:
135: def show_ends(str)
136:   str.gsub(/^(.*)$/, '\1$')
137: end
138:
139: def show_tabs(str)
140:   str.gsub(/\t/, "^I")
141: end
142:
143: if __FILE__ == $0
144:   main
145: end
```

正常終了、異常終了については http://doc.okkez.net/187/view/method/Kernel/m/exit を参照してください。

各オプションごとにメソッドを実装し、それらを組み合わせて指定されたオプションの動作を実現するという方向で実装しました。cat のオプションで -b を指定すると -n が効かないという仕様があったので、解答例のような実装 (line.82-93) になっています。

各オプションごとのメソッドについてはそんなに変わったことはしていませんが、メソッドごとにポイントを説明しておきます。

squeeze_blank
: 二つ以上連続する空行を一つにまとめる。正規表現の量指定子について調べるとよいです。

number_nonblank
: 空行以外に行番号を付ける。

number
: 行番号を付ける。Enumerator について調べるとよいです。

show_nonprinting
: コメントにも書いてあるとおり ascii(7) を見て実装しました。コードとコメントを見れば何をしているかは分かると思います。

show_ends
: 改行にマッチしたものを $\n に置換するだけです。

show_tabs
: タブ文字にマッチしたものを I に置換するだけです。

binary
: なにもしません。