応用プログラム言語II / 演習II
#14
Rubyでもっと実用プログラミング

情報システム学科
坂本政祐
sakapon@sit.ac.jp

はじめに

- これまで見てきたように，Rubyの記述性の高さは素晴らしいものがあります。
- 今回は，まだ説明していなかった「正規表現」をまず説明。
- さらに，実用的なスクリプト，ライブラリを使ったスクリプトなどを紹介していきながら，どれだけRubyのスクリプトは楽しく書けるかを紹介していきます。

正規表現

正規表現

- 正規表現: 色んな可能性がありうる文字列を，記号などを用いてひとつの文字列として表現する方法。
  - 例1)「bar かも知れないし baz かも知れない」を正規表現で表すと: ba[rz]
  - 例2)「baの後ろの1文字は何でもよい」を正規表現で表すと: ba.
  - 例3)「最初はsで最後はoならその間はどんな文字が何文字あっても良い」を正規表現で表すと: s.*o

正規表現

- 正規表現はRubyが発明したわけではなく，それ以前からある。
- Perlが正規表現を採用して文字列処理が劇的に便利になった。
- ので，Rubyでも
  - 正規表現リテラル
  - いくつかのメソッド

  で正規表現が使えるようになっている。

正規表現の意義

- いま，
  - ある文字列に含まれる "bar" を "foo" に置換したい。
  - その文字列に含まれる "baz" も "foo" に置換したい。

  ということがしたいとする。
- もしこれが正規表現が使えなかったら:
  - いったん "bar" を "foo" に置換し，
  - 再度 "baz" を "foo" に置換しなければならない。二度手間。

正規表現の意義

- しかし，文字列置換メソッド String#gsub, String#sub では正規表現が使えるので，「bar でも baz でもどっちでも foo に換える」というのを一度に書くことができる。

regexp1.rb
```
str = "XXXbarYYYbazZZZ"

str2 = str.gsub(/ba[rz]/, "foo") #=> "XXXfooYYYfooZZZ"
```

  - ちなみに，破壊的メソッド String#gsub!, String#sub! もある。

正規表現の詳細

- 正規表現の詳細:
  - / / で囲まれたものは正規表現リテラル（Regexpオブジェクト）となる。
  - 正規表現ではメタ文字が使え，これにより複数の可能性を1つの記号などで表現できる。
  - Regexp#=~ メソッドが「マッチした位置もしくはnil」を返すので，if 文で使いやすい。

正規表現のメタ文字

applied-progII #14

メタ文字	持つ意味
[ ]	[ ] 内のいずれか1文字。[ ] 内では「-」で範囲指定もできる。
.	任意の1文字（改行を除く）
*	直前の表現の0回以上の繰り返し
+	直前の表現の1回以上の繰り返し
?	直前の表現の0回または1回（直前の表現があってもなくても良い）
( )	グループ化
a|b	a, bどちらか1つ
¥s	空白文字類。（半角スペース，タブ，改行，ラインフィード）[¥t¥n¥r¥f] と書くのと同じ。
¥w	英数字。[a-zA-Z0-9_] と書くのと同じ。
¥d	数字。[0-9] と書くのと同じ。
^	行頭。
$	行末。

この他のメタ文字は http://doc.ruby-lang.org/ja/1.8.7/doc/spec=2fregexp.html

正規表現の使い方

applied-progII #14

- 正規表現の使い方:
  - if で使う

regexp-if.rb
```
str = "sakamoto"

if /saka/ =~ str
  puts "#{str} は saka を含んでいます。"
else
  puts "#{str} は saka を含んでいません。"
end
```


正規表現の使い方

applied-progII #14

- （先ほども出てきましたが）String#gsub の第1引数は正規表現を書ける。

regexp-gsub.rb
```
url = "http://www.foo.com:8080/index.html"

p url.gsub(/:¥d+/, "") #=> "http://www.foo.com/index.html"
```

  - /:¥d+/ の意味:「:の後ろに数字が1文字以上連続して並んでいる部分」
  - gsub(/:¥d+/, "") の意味: // 内の正規表現にマッチする部分がもしあったらそれを"" （空文字列）に置換する。

正規表現の使い方

applied-progII #14

- String#scan でマッチした部分を配列化。もしくはブロックに渡してブロック内を繰り返し実行。

regexp-scan.rb
```
p "foobarbazfoobarbaz".scan(/ba./)
# => ["bar", "baz", "bar", "baz"]
```

regexp-scan2.rb
```
"foobarbazfoobarbaz".scan(/ba./) {|s| p s}
# => "bar"
#    "baz"
#    "bar"
#    "baz"
```


正規表現の使い方

applied-progII #14

- いったん =~ でマッチさせて，後から $番号 でマッチ部分を参照する。ひとつめの ( ) にマッチした部分が $1, ふたつめの ( ) にマッチした部分が $2 ...

regexp-matchdata.rb
```
url = "http://www.foo.com:8080/index.html"

/http:¥/¥/([¥w¥.-]+)(:¥d+)?/ =~ url

p $1                    #=> "www.foo.com"
p $2.sub(/:/, "").to_i #=> 8080
```

  - なお，上記のホスト名部分はわかりやすさ優先で [¥w¥.-]+ と書いてしまいましたが，ホスト名に _ を本当は使ってはいけないなどの理由で，本当は以下です。([a-zA-Z0-9][a-zA-Z0-9-]*[.])*[a-zA-Z0-9][a-zA-Z0-9-]+
  - なお，URIクラスならもっと簡単です（後述）。

正規表現が使えるメソッド

applied-progII #14

- 他に正規表現が使えるメソッド（マニュアル抜粋，一部改変）
  - String#slice(regexp, nth = 0)
    - 正規表現 regexp の nth 番目の括弧にマッチする最初の部分文字列を返します。nth を省略したときや 0 の場合は正規表現がマッチした部分文字列全体を返します。正規表現が self にマッチしなかった場合や nth に対応する括弧がないときは nil を返します。
  - String#index(regexp, pos = 0)
    - 文字列のインデックス pos から右に向かって regexp を検索し、最初に見つかった部分文字列の左端のインデックスを返します。見つからなければ nil を返します。pos が負の場合、文字列の末尾から数えた位置から探索します。
  - String#rindex(regexp, pos = self.size)
    - indexの逆版。末尾から検索。

正規表現が使えるメソッド

applied-progII #14

- 他に正規表現が使えるメソッド（マニュアル抜粋，一部改変）
  - String#partition(regexp)
    - regexp をセパレータとして、それが最初に登場する部分でレシーバを 3 つに分割し、[最初のセパレータより前の部分, セパレータ, それ以降の部分] の 3 要素の配列を返します。レシーバがセパレータを含まないときは、返り値の第 2 要素と第 3 要素が空文字列になります。
  - String#rpartition(regexp)
    - String#partitionの逆版。regexpが最後に登場する部分で分割。
  - String#split(regexp)
    - 正規表現 regexp で指定されたセパレータによって文字列を分割し、結果を文字列の配列で返します。特に、括弧によるグルーピングがあればそのグループにマッチした 文字列も結果の配列に含まれる 。

いろいろな実用スクリプト

日時関係

applied-progII #14

- ファイルから1行ずつ読み取って表示。
  - これを応用すれば行番号つき表示とか，1行ずつ加工するとかが簡単にできる。

fileread.rb
```
File.foreach("test.txt") do |gyou|
  print gyou
end
```

- File::readlines で1行ずつ配列化もできるので，Array#shuffleと合わせてランダムに行入替え。

fileread2.rb
```
print File.readlines("test.txt").shuffle
```

17

日時関係

applied-progII #14

- 現在時刻を "hh:mm:ss" 形式で表す。

hhmmss.rb
```
p Time.now.strftime("%H:%M:%S") #=> "10:45:23"
```

- 今日の日付を "YY-MM-DD" 形式で表す。

yymmdd.rb
```
p Time.now.strftime("%Y-%m-%d") #=> "2012-12-31"
```

18

日時関係

applied-progII #14

- ふたつの日付の間の日数を求める。もちろん月や年をまたいでも構わない。

date-diff.rb
```
require 'date'

date1 = Date.new(2012, 12, 1)
date2 = Date.new(2012, 11, 15)
p (date1 - date2).to_i #=> 16
```

- ある日付から n 日前は何月何日か。

date-diff2.rb
```
require 'date'

date1 = Date.new(2012, 12, 1)
p (date1 - 16).to_s #=> "2012-11-15"
```

19

日時関係

applied-progII #14

- 今日が国民の祝日かどうか。

holiday.rb
```
require 'date'
require 'date/holiday'

p Date.today.national_holiday? #=> true or false
```

20

日時関係

applied-progII #14

- 応用: ある年月日は日祝日かな?
  - 前回やったオープンクラスの考え方を使っています。

holiday2.rb
```
require 'date'
require 'date/holiday'

class Date
  def yasumi?
    return self.national_holiday? || self.wday == 0
  end
end

p Date.today.yasumi?
p Date.new(2012, 12, 24).yasumi?
p Date.new(2012, 12, 25).yasumi?
```

21

巨大な数

applied-progII #14

- 巨大な整数を扱える。(Cだとオーバフロー)
  - これは，そういう特別なメソッドがあるわけじゃなく，巨大な数になるような計算をすると，自動的にFixnumオブジェクトからBignumオブジェクトに切り替わってくれるということ。

bignum.rb
```
# 2の200乗
p 2**200
# => 1606938044258990275541962092341162602522202993782792835301376

# 100! (100の階乗)
p (1..100).to_a.inject(1){|prod, i| prod * i}
# => 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

22

分数

applied-progII #14

- 有理数を扱う(分数を分数のまま扱える)

rational.rb
```
require 'rational'

p Rational(1, 3) * 2              #=> Rational(2, 3)
p Rational(1, 3) + Rational(1, 5) #=> Rational(8, 15)
```

23

ファイル名の変更

applied-progII #14

- カレントディレクトリにある，拡張子が .ppt のファイルすべてに対して，「kougi」という文字列がファイル名に含まれていたらその部分を「lecture」に変更。

file-rename.rb
```
Dir.glob("*.ppt").each do |fname|
  if /kougi/ =~ fname
    new_fname = fname.sub(/kougi/, "lecture")
    File.rename(fname, new_fname)
  end
end
```

24

XMLファイルのパース

applied-progII #14

- XMLファイルを解析
  - 以下のようなXML

movies.xml
```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
  <movie>
    <title>タイタニック</title>
    <point>10</point>
    <review>長い</review>
  </movie>
</movies>
```

  - 解析するRubyコードの例

xml-parse.rb
```
require 'rexml/document'

doc = REXML::Document.new(File.open("movies.xml"))
puts doc.elements["/movies/movie/title"].text #=> タイタニック
```

25

XMLファイルのパース

applied-progII #14

- あるエレメント（movie）が複数あっても…

movies2.xml
```
<?xml version="1.0" encoding="UTF-8"?>
<movies>
  <movie>
    <title>タイタニック</title>
    <point>10</point>
    <review>長い</review>
  </movie>

  <movie>
    <title>コマンドー</title>
    <point>10</point>
    <review>シュワちゃん最高！</review>
  </movie>
</movies>
```

26

XMLファイルのパース

applied-progII #14

- eachメソッドで列挙できる。

xml-parse2.rb
```
require 'rexml/document'

doc = REXML::Document.new(File.open("movies2.xml"))
doc.elements.each("/movies/movie") do |elem|
  puts elem.elements["title"].text
  #=> タイタニック
  #=> コマンドー
end
```

Webクライアント

applied-progII #14

- あるURLからHTMLをゲット

web-client.rb
```
require 'net/http'
require 'uri'

html = Net::HTTP.get URI.parse('http://sakamotoken.sit.ac.jp/')
print html
```

  - 画像もOK（悪用しないでね）

web-client2.rb
```
require 'net/http'
require 'uri'

img = 'http://sakamotoken.sit.ac.jp/sakamotoken200x91_bback.png'
save_fname = File.basename(img) #=> "sakamotoken200x91_bback.png"
File.open(save_fname, 'w'){|f|
  f.write Net::HTTP.get(URI.parse(img))
}
```

28

Webサーバ

applied-progII #14

- WEBrickによるWebサーバ

webrick.rb
```
require 'webrick'
include WEBrick

s = HTTPServer.new(
  :Port => 80,
  :DocumentRoot => File.join(Dir::pwd, "www")
)
trap("INT"){ s.shutdown }
s.start
```

  - ruby webrick.rb で起動
  - 起動したディレクトリに www というディレクトリを作って，その中に適当な中身で index.html を置く。
  - Webブラウザから http://localhost/ にアクセス。

29

URLパースの本当のやり方

applied-progII #14

- URLからホスト名やポート番号を抜き出す。

uri-parse.rb
```
require 'uri'

uri = URI("http://sakamotoken.sit.ac.jp:80/index.html")
p uri.host #=> "sakamotoken.sit.ac.jp"
p uri.port #=> 80
```

30

evalで何でも動的に

applied-progII #14

- 禁断のeval
  - 動的に作った文字列をRubyスクリプトとしてその場で評価できてしまう。

eval.rb
```
command = 'puts "Hello, World."'
eval command #=> Hello, World.
```

  - メソッドを動的に作る。

eval2.rb
```
str = "def mult(x, y); return x*y; end"
eval str
```

  - メソッド名すら動的に決められる。

eval3.rb
```
method_name = STDIN.gets
str = "def #{method_name.chomp}(x, y); return x*y; end"
eval str
```

31

その他

applied-progII #14

- このほかにも，組み込みライブラリ・標準添付ライブラリで以下のようなことができる。
  - csvファイルの読み書き。
  - JSONファイルの読み書き。
  - YAMLファイルの読み書き。
  - マルチスレッド。
  - ユニットテスト。
  - 複素数を扱う。
  - メールの送受信。
  - ソケット。
  - win32ole。Windowsアプリを操縦できる。

32

Ruby用ライブラリ・Rubyバインディング

Rubyでもライブラリプログラミング

- ここまで見てきたのはRubyの組み込みライブラリもしくは標準添付ライブラリでできること（＝Rubyをインストールすればどんな環境でもできる）。
- 一方，他の言語でもそうであるように，あるいはこの講義でも以前やったように，Ruby用の様々なフリー／オープンソースのライブラリが公開されている。
  - もともとRuby用に開発されたものもあれば，
  - C/C++用に開発されたライブラリを，Rubyらしく書けると嬉しいなということでRubyに結びつけた（バインディング）ものもある。

Rubyでもライブラリプログラミング

- ただし，これらのライブラリはインストールが必要なので，残念ながらこの部屋では試すことができません。でもせっかくなので紹介します。

PDF生成ライブラリ

- hpdf: RubyでPDFを作るライブラリ。
  - もうWordは要らないね！特に沢山自動的に処理したいようなとき。
  - 実はこの講義の出席バーコードシートもこれで作っています。
  - A4を想定したPDF上の任意の座標に，線を引いたり文字を置いたり画像を置いたりできる。
    - もちろん他の用紙サイズもOK。年賀状も作れちゃいますね！

PDF生成ライブラリ

generate-pdf.rb

```
require "hpdf"
require "kconv"

pdf = HPDFDoc.new
pdf.use_jp_fonts
pdf.use_jp_encodings
encoder = pdf.get_encoder("90ms-RKSJ-H")
font = pdf.get_font("MS-Mincyo", "90ms-RKSJ-H")

page = pdf.add_page
page.set_size(HPDFDoc::HPDF_PAGE_SIZE_A4, HPDFDoc::HPDF_PAGE_PORTRAIT)
page.set_rgb_fill(0, 0, 0) # 文字色
page.set_font_and_size(font, 32)

page.begin_text
page.text_out(100, 700, "こんにちは，世界！".tosjis)
page.end_text

pdf.save_to_file("hello-world.pdf")
```


PDF生成ライブラリ

- 生成されたPDF

xls操作ライブラリ

- Spreadsheet: RubyからExcelのxlsファイルを直接いじれるライブラリ。事務処理がいろいろ捗る。
  - Excelがインストールされていなくてもok。
  - 新規にxlsファイルを生成することができる。
  - 既存のxlsファイルを読み書きすることができる。

xls操作ライブラリ

generate-xls.rb

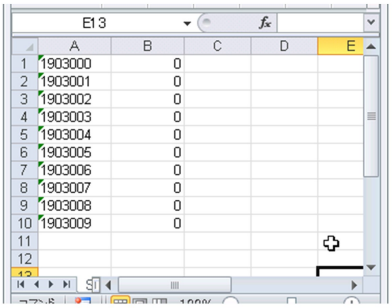
```
require 'rubygems'
require 'spreadsheet'

book = Spreadsheet::Workbook.new
sheet = book.create_worksheet
sheet.name = 'Sheet1'
10.times do |n|
  sheet[n, 0] = sprintf("1903%03d", n)
  sheet[n, 1] = 0
end
book.write('seiseki-temple.xls')
```

xls操作ライブラリ

applied-progII #14

- 生成されたxls



41

画像処理ライブラリ

applied-progII #14

- Ruby/OpenCV: OpenCVという超有名画像処理ライブラリをRubyから操るライブラリ。
  - OpenCVはもともとわかりやすいAPIだが，それでもやっぱりC/C++ベースなので，画像処理以外の部分が面倒くさい。
    - ファイル名を動的に生成するとか，
    - カレントディレクトリにあるすべてのjpgファイルに対して，とか。
  - そういう部分をRubyで書けるとやっぱり楽。

42

画像処理ライブラリ

applied-progII #14

ruby-opencv.rb

```ruby
require 'opencv'
include OpenCV

orig_img = IplImage.load('Lenna.jpg', CV_LOAD_IMAGE_GRAYSCALE)

thre_img = orig_img.adaptive_threshold(255, :block_size => 25)

GUI::Window.new('original').show orig_img
GUI::Window.new('threshold').show thre_img
GUI::wait_key
GUI::Window.destroy_all
```

43

ちなみにCで書くと

opencv.c

```c
#include <opencv2/imgproc/imgproc_c.h>
#include <opencv2/highgui/highgui_c.h>

int main( int argc, char **argv )
{
    IplImage *orig_img;
    IplImage *thre_img;

    orig_img = cvLoadImage("Lenna.jpg", CV_LOAD_IMAGE_GRAYSCALE);

    cvAdaptiveThreshold(orig_img, thre_img, 255, 0, 0, 25, 50.0);

    cvNamedWindow("original", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("threshold", CV_WINDOW_AUTOSIZE);

    cvShowImage("original", orig_img);
    cvShowImage("threshold", thre_img);

    cvWaitKey(0);

    cvReleaseImage(&orig_img);
    cvReleaseImage(&thre_img);

    cvDestroyWindow("original");
    cvDestroyWindow("thoreshold");

    return 0;
}
```

44

まとめ

applied-progII #14

- Rubyに慣れてくると，「たぶんこう書けるだろうな」というのがだいたいその通りになるし，最悪そうならなかったとしてもオープンクラスによってどんどん自分の思う通りにメソッドを追加していける。
- この3回はRubyに焦点を当てたが，他のモダンなLL（例えばPython）でも記述性の高さは似たようなものである。
- ちょっとしたツールはどんどんRubyやPythonで作って作業効率を激アップさせよう。

45

今日の課題

applied-progII #14

- 課題1: 正規表現を使って，何か意味のあるスクリプトを書いてください。
- 課題2: オープンクラスの機能を使って，既存のクラスに何か自分で独自のメソッドを追加してください。意味のある（実際に使いみちのありそうな）メソッドにしてください。
- なお，前回の課題1の提出がまだの人は，引き続きそれも取り組んで今日いっしょに出して下さい。

46

最終レポート課題

47

最終レポート課題

applied-progII #14

- 最終レポート課題:
  - この講義でこれまでに取り扱った内容に関係することなら何でも良いので，何か1つプログラムを作ってください。
    - 規模は問いません。意味のあるプログラムになっているなら，ワンライナー（1行プログラム）でも構わないということです。
    - オリジナリティを重視します。
    - #08「ライブラリで楽しよう」という回がありましたが，それに関係するという意味では，#08で紹介した以外のライブラリを使ったプログラムでも構いません。もちろん，紹介したライブラリを使ったプログラムでも構いません。

48

最終レポート課題

applied-progII #14

- 最終レポート課題(続き):
  - 提出〆切: 2013年1月28日(月) 17:00まで。
  - 提出方法: 今回はE-mailに添付して提出して下さい。
    - To: sakapon@sit.ac.jp
    - Subject: 応用プログラム言語IIレポート
    - Visual Studio で作ったプログラムは, プロジェクトのフォルダー式をzip圧縮して添付してください。ただし, Gmail 経由だと exe ファイルを含んでいると送れない場合があります。その場合は, 他のアカウントから送るか, もしくは lzh 圧縮でも構いません。
    - それ以外のシステムや言語で作ったプログラムは, ソースコードのみで構いません。(例: Ruby, Brainfuck)

49

最終レポート課題

applied-progII #14

- 最終レポート課題(続き):
  - 自作したものであるなら, 既発表のものでも構いません(例えば, 自身のブログで過去に発表したソースや, 過去にネットで自分で公開した自作フリーソフトのソース等)。この講義の課題として作ったものは駄目です。

50

今週の落穂拾い

applied-progII #14

- evalは少し紹介しましたが, Rubyにはこの他にも様々なリフレクションの仕組みがあります。
- また, こういった仕組みを利用できるため, Rubyは「内部DSL」を記述することに優れた言語とも言われています。
- DSL: Domain Specific Languageというのは「ある分野に特化した言語」。例えばロボットの操縦を言語で書きたいとして, それを例えばC言語の文法に縛られて書くのは嫌なわけです。ロボットにはロボット特有の特性があるでしょうから, 関数と変数しか使えない言語ですべてを記述したくないわけです。

51

今週の落穂拾い

applied-progII #14

- そういう時に, ロボット動作言語を書きたいわけですが, しかしゼロから作ってそのパーサ/コンパイラを作るのもまた大変。
- そういったときに, 「ロボットの動作を記述してるように見えるんだけど, 実はそれはRubyの実行文になっている」というのが内部DSLです。
- Ruby中級になったな, と思ったら是非DSLも調べてみてください。結構感動的ですよ。(そして, C/C++を使う気がどんどん失せてきますよ(苦笑)。)

52