Ruby について超簡単なガイド

降旗 大介

Ver.1.9

1 Introduction

Ruby とは，結構良くできているスクリプト言語，といえばいいのか．とりあえず簡単なことを簡単にできる，コンピュータ言語と思っていただけばよい．何かを数える，なんていう用途にはピッタリだ．

2 インストール

Windows の場合は，cygwin をインストールし，その内部コマンドとしてインストールするのが一番楽だろう．

3 ごく簡単な使い方

Ruby では，プログラムをテキストファイルとして作成して，それを読み込むという形をとるのがまずは簡単だ．そこで、まずはその形にそって、以下のように試してみよう．

1. まずは unix 環境が必要だ．多くの環境や阪大教育用端末では，cygwin がその代用になるだろうから，cygwin を立ち上げよう．こいつは一旦立ち上げたら作業が終わるまで使いっぱなしで構わない．
2. 次にテキストエディタを立ち上げよう．windows だとノートパッドでもよい．使える人は Emacs の方が便利だろう．
3. これから作業するディレクトリ (フォルダ) を決めよう．これから作るファイルが消えたりしないところにしよう．また，フォルダ名に日本語が入っていると cygwin がとても面倒なことになるので，そうでない場所にしよう．

   そして，cygwin の作業ディレクトリもそこに一致させよう．具体的には，cygwin の中で

   ```
   cd "フル path でのフォルダ名"
   ```

   とすればよい．
4. 最初のプログラムを書こう．テキストエディタで次のように一行だけ書いてみよう．

   ```
   p 3+5
   ```
5. せっかく書いたプログラムだ，名前をつけて保存しよう．エディタの機能で「save」とか「保存」を選べば名前をつけてファイルとして保存できる．ファイル名は今回は “test.rb” としておこう．
6. プログラムを動かしてみよう．cygwin の中で，

   ```
   ruby -w test.rb
   ```

   と打ってみよう．うまくいけば

   ```
   8
   ```

   という答えが返ってくるだろう．
7. あとはこの場合，`test.rb` ファイルの中身を適当に書き換えてまた `ruby -w test.rb` としてみることを繰り返せばいろいろチャレンジできる．

4 超簡単な文法解説

名称	解説, 注意	サンプル
数学したい	数学計算を行なう準備 (1 度でよい)	`include Math`
-	pp を使うための準備 (1 度でよい)	`require 'pp'`
表示 1	画面に内容を出す	`p ほにゃらら`
表示 2	p より見やすい	`pp ほにゃらら`
表示 3	丁寧に表示したい	`print ほにゃらら`
代入	変数に数字などを入れる	`a = 3.5`
掛け算	-	`a*b`
べき乗	-	`2**3` (8 になる)
絶対値	-	`a.abs` (a の絶対値)
切り上げ	-	`a.ceil` (a の切り上げ)
対数	自然対数	`log(a)`
指数	-	`exp(a)`
乱数	0 以上 1 未満	`rand()`
変数に足す	-	`a += 1` (a に 1 足す)
変数をコピー	-	`a = b` (a に b の中身をコピー)
文字列をコピー	文字列はちと特殊だよ	`a = b.dup`(a に b の中身をコピー)
文字列のある 1 文字	数え方が 1 ずれる	`str[3].chr`(左から 4 文字目の場合)
文字列の一部	数え方が 1 ずれる	`str[1..4]`(2〜5 文字目まで)
文字列一部削除	数え方が 1 ずれる	`s.slice!(3..6)`(4〜7 文字目を削除)
文字を整数に	-	`s.to_i`
文字を実数に	-	`s.to_f`
数字を文字に 1	便利なこともある	`a.to_s`
数字を文字に 2	() に 4 と入れると 4 進数表示に	`a.to_s(4)`
文字の数 1	全部で何文字？	`s.length`
文字の数 2	特定の文字の数	`s.count("a")`
単なる集合	-	`[a,b,3]` など
集合の要素	数え方が 1 ずれる	`a[3]` は 4 番目の要素を指す
集合に要素追加	-	`group.push(a)`
ペアの集合 (ハッシュ)	ペアを集めたもの	`hash = { "a" => "3", "b" => "2"}`
ペア集合中身の作業	ペアを一つずつ列挙して作業	`hash.each { |k, v| 作業 }`
最大, 最小値	集合の中身の最大, 最小値	`[3,5,2].min` など
数えながら繰返す	ループだね	`for i in 1..5 do ほにゃらら end`
数えながら繰返す 2	上と同じ動作	`1.upto(5){|i| ほにゃらら }`
数えながら繰返す 3	減らす方向に	`5.downto(1){|i| ほにゃらら }`
もしも	条件があうなら動作	`if ほにゃらら then ほい else ほい 2 end`
もしもループ	条件があう間はループ動作	`while ほにゃらら do ほい end`
一致してる?	比較する	`a == b`
関数作成	できると便利	`def ほにゃらら (引数) 中身 end`
関数の出力は	関数定義の中で使おう	`return 出力`
動作時のパラメータ	`ruby test.rb 3 10` とするなど	`ARGV[i]` で `i+1` 個目
コメント	メモ書きができるよ	`# ほにゃらら`

注意: 大文字小文字は区別される．変数名を大文字で書くと中身変更不可．大域変数は `$` で始める．

5 簡単なサンプル

5.1 エントロピー計算

例えば，5 つの値を取る確率が

x	1	2	3	4	5
$P(X=x)$	1/20	1/5	1/2	1/5	1/20

となるような確率変数 X の情報源のエントロピー

$$H(X) = \sum_{x \in \chi} P(X=x) \log\left(\frac{1}{P(X=x)}\right)$$

を求めるには，次のようなプログラムが書ける (二段表記しているので注意)．手で計算するより簡単だし、なにより計算ミスがなくて確実だ．なお，底が 2 の対数関数を lg と表記する慣習があるようなので，覚えておくと楽かも．

```
include Math

def lg(p)
  return log(p)/log(2.0)
end

def entropy(array)
  h = 0.0
  array.each{|p|
    h -= p * lg(p)
    }
  return h
end

X = [0.05, 0.2, 0.5, 0.2, 0.05]
print "H(X)␣=␣", entropy(X),"␣bit\n"
```

5.2 Newton 法

$$\begin{cases} x^2 + y^2 = 1, \\ \sqrt{3}x - y = 0, \end{cases}$$

の数値解を計算するために、

$$\begin{pmatrix} x^+ \\ y^+ \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \frac{1}{2x + 2\sqrt{3}y} \begin{pmatrix} 1 & 2y \\ \sqrt{3} & -2x \end{pmatrix} \begin{pmatrix} x^2 + y^2 - 1 \\ \sqrt{3}x - y \end{pmatrix}$$

という反復式をつかう Newton 法を ruby で何も考えずにベタで書いてみよう．面倒なので 10 回反復と決め打ちして、例えば以下のようなプログラムができる．ずいぶんかっこわるいが, 簡単にプログラムが書けて気楽だ．

```
include Math

SQ = sqrt(3.0)

def f(arr)
  x,y = arr
  return x**2 + y**2 - 1.0, SQ*x - y
end

def newton(arr)
  x,y = arr
  fx,fy = f(arr)
  d = 2.0* (x + SQ*y)
  dx = (fx+2*y*fy)/d
  dy = (SQ*fx-2*x*fy)/d
  return x-dx, y-dy
end

ini = [2.0,0.0]
p ini

a = ini
for i in (1..10)
  b = newton(a)
  p b
  a = b
end
```

6　ペア集合の each ってのがわかんないんだけど?

これはペア集合の中身であるペアを一つ取り出して仮の名前をつけ，指定した作業を行なったら次のペアを取り出して同様のことを繰り返すというものだ．ペアを列挙して操作するのにはピッタリの機能で，数学科の学生にはとても便利だ．具体例で見た方がわかりやすいだろう．例えば，

```
1 require 'pp'
2 a = 1.5
3 b = 2.5
4 hash = {1=>2, 2=>4, 3=>8, 4=>16, 5=>32}
5
6 hash.each {|left,right|
7   print "modify:␣(", left, ",␣", right, ")␣->␣(", a*left, ",␣", b*right, ")\n"
8   }
```

というプログラムを考えてみよう．これは数学っぽく書けば $\{(1,2),(2,4),(3,8),(4,16),(5,32)\}$ という集合に hash という名前をつけ，その中身のペアを一つ取り出したときに仮に $(\mathrm{left},\mathrm{right})$ という名前で呼んで処理をしていることに相当する．

実際にこのプログラムを動かしてみると，

```
modify: (1, 2) -> (1.5, 5.0)
modify: (2, 4) -> (3.0, 10.0)
modify: (3, 8) -> (4.5, 20.0)
modify: (4, 16) -> (6.0, 40.0)
modify: (5, 32) -> (7.5, 80.0)
```

のような結果が出力される．だいたいこれでやっていることがわかるだろう．ちなみに、古い Ruby ではペア集合 each から中身がどういう順番で取り出されるかは決まっていないので，そこは気をつけておこう．

備考: 単純な集合などにもこの .each という機能は利用できる．プログラム上、これは大変便利な機能なので覚えておくといいだろう．

7　マニュアルや，入門コースとか無いの?

もちろんありますデスよ．ついでに，ruby についての本家 Web も示しておきましょう．

本家 http://www.ruby-lang.org/ja/

公式マニュアル http://www.ruby-lang.org/ja/man/html/

チュートリアル (入門コース)
http://www1.tf.chiba-u.jp/~shin/tutorial/index.rb

Ruby ではじめるプログラミング (これもわかりやすいよ)
http://jp.rubyist.net/magazine/?0002-FirstProgramming

マニュアルをダウンロードしよう
http://www.ruby-lang.org/ja/documentation/