

# Ruby プログラミングへの準備

プログラミング演習 I 担当 時井、松村

## 概要

プログラミング演習 I を受講する予定の人向けの入門として、簡単なプログラムの実行の方法と間違いの直し方などについて説明する。

## 1 プログラミングの前に

プログラミング演習は、基本的に全学計算機システムの Windows にログインして、図 1 に示したエディタ Meadow を利用してプログラムを作成する。

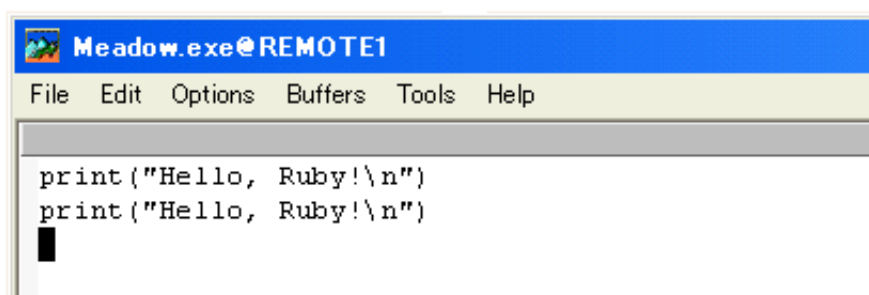


図 1: Meadow の画面

したがって、エディタの操作には習熟していなければならない。そして最後に、この演習では比較的多くのキーボード入力を要求される。ある程度、キーボード入力を速くできるようになればならない。

以上まとめると、プログラミング演習 I を受講するにあたって要求されることは、

- エディタ Meadow 操作の習熟
- 迅速なキーボード入力

である。これらは、テキストや参考書を使って何度も復習することでクリアできるはずである。2 学期までは時間があるので頑張ってみてほしい。

## 2 – Ruby プログラミング超入門 –

### 2.1 作業フォルダの準備

まずは、プログラミング演習 I の作業用フォルダとして、Z ドライブの下に progI という名前のフォルダを作成しよう。作成手順は次のようになる。

- 全学計算機システムの Windows にログインする。
- まず、[スタート] メニュー-[検索の開始] ボックスに `cmd` と入力し、「プログラム」の欄から `cmd` を選択する。
- コマンドプロンプトが起動し、`C:\Users\**** >` と表示されるので、この後ろに `Z:` と入力し、[Enter] キーを押す。
- コマンドプロンプトに `Z:\>` と表示されるので、この後ろに `mkdir progI` と入力し、[Enter] キーを押す。(注意: `\` は、実際画面上では、`¥` (エンマーク) である。また、`mkdir` と `progI` の間にはスペースを入れる。)

コマンドプロンプトは、キーボードからコマンド (command: 命令) を入力することによってファイル操作やプログラムの実行を行うことができる。どんなコマンドがあるかは、`help` と入力し、[Enter] キーを押して調べてみよう。コマンドの使い方は、`help` コマンド名 (注意: `help` とコマンド名の間にはスペースを入れる。) で見ることができる。コマンド `mkdir` は、`make directory` の略で、ディレクトリ (Windows では、フォルダ) を作成することを意味する。使い方は、「`mkdir` フォルダ名」である。`mkdir progI` とすると、現在のフォルダの下に、`progI` という名前のフォルダが作成される。

#### 演習

1. コマンドプロンプトを起動する。
2. `help` と入力し、[Enter] キーを押して、どんなコマンドがあるか確かめてみよう。
3. `help mkdir` と入力し、コマンド `mkdir` の使い方を調べてみよう。
4. 下記にあげたコマンドの使い方を調べてみよう。
  - `cd`
  - `dir`
  - `move`
  - `del`
  - `copy`
5. `mkdir progI` と入力し、フォルダ `progI` を作成してみよう。

## 2.2 プログラムの作成と実行

一般にプログラムといった場合、計算する手順を人間にもわかりやすいことばを用いて記述したものと、実際にコンピュータが計算できる形式になっているものの両方を指す。特に、前者のことをソースプログラム (source program)<sup>1</sup>という。まずは、その例を示そう。(注意：このテキスト内の\ (バックスラッシュ) は、キーボードの¥ (エンマーク) を使って入力する。)

```
1 print("Hello, Ruby!\n")
2 print("Hello, Ruby!\n")
```

図 2: Hello, Ruby! と 2 行表示するプログラム

これは画面に

```
Hello, Ruby!
Hello, Ruby!
```

と表示するプログラムである。なお、左端にある数字はこのテキスト内で「何行目」というときに便利であるから付けたものである。実際のプログラムにはこのような数字をつけてはならない。

通常、プログラムを作るといった場合、このような形式のテキストファイルをエディタを使って作成する。例えば、hello.rb という名前にする。ここで注意しなければならないのは、サフィックス (suffix) を .rb としなければならないことである。

さて、ソースプログラムを書いただけでは何も起こらない。今度は書かれた内容を実行する。教育用システム上では ruby というコマンドが用意されているので、手順は次のようになる。

- コマンドプロンプトを起動する。
- コマンドプロンプトに、Z:\>と表示される。この文字列の後ろに、指定された文字列を入力する。(注意： \ は、実際画面上では、¥ (エンマーク) である。)
- まずは、フォルダの移動を行うために、cd progI と入力し、[Enter] キーを押す。
- 次に、ファイルの一覧を表示するために、dir と入力し、[Enter] キーを押す。すると、画面にフォルダ progI の中に保存したファイルの名前が表示されるので、ソースプログラム (例えば hello.rb) があることを確認する。
- プログラムの実行は、以下のように、ruby hello.rb などと入力する。(ruby ファイル名)

```
Z:\>cd progI
Z:\progI>dir
.....
Z:\progI>ruby hello.rb
```

すると、画面に

<sup>1</sup>単にソース、又はソースコードという場合もある。

```
Hello, Ruby!  
Hello, Ruby!
```

と表示される。

コマンド `cd` は、change directory の略で、指定したディレクトリ（Windows では、フォルダ）に移動することを意味する。使い方は、「`cd フォルダ名`」である。`cd progI` とすると、現在いるフォルダの下にある `progI` という名前のフォルダに移動することができる。コマンド `dir` は、list directory の略で、フォルダとファイルの一覧を表示することを意味する。`dir` コマンドを使用した結果、画面上で `<DIR>` と表示されているものが、フォルダで、それ以外はファイルである。

### （補足）ソースプログラムの解説

`print` は画面に表示せよ、という Ruby での命令である。表示する内容はその後ろに括弧で括って書いた `"Hello, Ruby!\n"` である。なお、`\n` は特殊な意味を持ち、「改行」を指示する。実際に表示されるのはダブルクォート（`"`）で括った中身である。

#### 演習

1. エディタ Meadow を起動する。
2. 図 2 のプログラムをエディタ Meadow を使って入力する。（注意：左端にある数字（行番号）は入力しないこと）
3. 一文字たりとも間違っていないことを確認し、`Z:\progI` の下に、`hello.rb` というファイル名をつけ保存せよ。
4. コマンドプロンプト上で、`cd progI` と入力し、`[Enter]` キーを押す。この操作により、`Z` ドライブの下のあるフォルダ `progI` に移動することができる。（注意：`cd` と `progI` の間にはスペースが必要）
5. コマンドプロンプト上で、以下の手順でプログラムを実行する。（`>`の後ろにつづけて指定された文字列を入力し、`[Enter]` キーを押す。（注意：`rubyhello.rb` の間にはスペースが必要）

```
Z:\>cd progI  
Z:\progI>dir  
.... ここにファイルの一覧が表示される.....  
  
Z:\progI>ruby hello.rb
```

画面に `Hello, Ruby!` と 2 行表示されたら成功である。もし、なにかメッセージが出た場合には間違いがあったと考えられる。エディタに戻って間違いを修正し、もう一度実行する。

## 2.3 エラーに対する対処

不幸 (幸運 ?) にして、上記の演習でエラーメッセージが出た人もいるかもしれない。ここではエラーメッセージが出た場合の対処方法について述べる。

まずは、エラーメッセージが出るようにしてみよう。図 2 の 2 行目の最後にある括弧 `)` を削除してみよう。その後、`ruby hello.rb` を実行する。すると図 3 のようなエラーメッセージが出力される。

```
hello.rb:2: syntax error, unexpected $end, expecting ')
```

図 3: エラーメッセージの例

最初の `"hello.rb"` は対象となるファイル名である。2 はそのファイル内での行番号であり、2 行目ではじめてエラーであることがわかったことを意味する。エラーの内容は、

```
syntax error, unexpected $end, expecting ')
```

である。文字通り解釈するならば、「構文の誤り、予期しない `$end`、`' )` (という記号) を期待する」ということになる。

エラーメッセージを読んだらそれをヒントに実際に間違っているところを自分で発見して修正を行う必要がある。

### 演習

実際に前頁に記述したように、図 2 のプログラムの 2 行目の終わりの括弧 `)` を抜いて、実行してみよう。また、1 行目の終わりの括弧を抜くとどのようなエラーメッセージが表示されるか確認してみよう。

## 2.4 インデント

プログラムの間違いをなくすために、可読性を高めることが重要である。そのためには、インデントを適切に行なう必要がある。インデントは、スペースを入力しても良いが `tab` キーを用いて行なうと便利である。

```
1 x = 10
2
3 if x == 10
4   print("x は 10 です\n")
5 else
6   print("x は 10 以外です\n")
7 end
```

図 4: sample0.rb

プログラム `sample0.rb` では、4 行目では左端から 2 つ空白があったのちに `print` と書かれている。人がプログラムを読む際に、プログラムの区切りをわかりやすくするため、わざと 2 つ分の空

白を入れている。このように空白を入れることを字下げあるいはインデント(indent)という。プログラムを読みやすくすることは間違いなどの混入を防ぐ意味でも非常に重要である。

**Meadow の Ruby mode** Meadow を利用して、Ruby のソースプログラム (サフィックスが .rb のファイル) を編集している場合には、Ruby mode という特別のモードになっていて、各行で tab キーを押すと適切な位置までインデントしてくれる。また、C-x h とした後に ESC C-\ とすると、ファイル全体に対して一気にインデントを行ってくれる。途中何箇所かインデントを忘れてしまったというような場合には便利である。

#### 演習

- sample0.rb の 4 行目をインデントせずに入力した後、カーソルを行頭に移動して tab キーを何度か押してみよ。また、行の途中で同様の操作をするとどうなるか。
- 4 行目と 6 行目をインデントせずに入力した後、C-x h とし、ESC C-\ としてみよ。

以上の操作は sample0.rb 程度のプログラムではありがた味を感じないかも知れないが、プログラムが複雑になればなるほど便利に感じられるであろう。次の節の練習問題でも是非試してもらいたい。

### 3 Ruby で遊ぶ

プログラミング (programming, プログラムを組む) の上達のコツは、とにかくプログラムを入力して実行することで、動作を理解することである。プログラミングに対する理解力は人それぞれであるので、とにかく、新しい考え方、やり方に触れたときは、その方法に慣れるまで繰り返し練習をすることが重要である。繰り返し練習をして考え方に馴染んでくれば授業でやっている内容も理解できるようになる。

#### 演習

1. 図 2 を改造してみよう。print 中のダブルクォートで括られた中身を適当な違う文字列にしたり、改行を示す \n を削除したりしてみよう。プログラムを修正したら保存して、実行をするという手順を忘れずに。
2. 図 5 から、Ruby のプログラムをいくつか挙げる。それぞれ入力し、コンパイルし、実行してみよう。これらも適宜改造してみたい。なお、ソースプログラムのサフィックス (suffix) は、.rb でなければならない。

```
print("H")
print("e")
print("l")
print("l")
print("o\n")
print("Hello\n")
```

☒ 5: sample1.rb

```
puts("Hello")
```

☒ 6: sample2.rb

```
print(2 + 3, "\n")
print(2 - 3, "\n")
print(2 * 3, "\n")
print(2 / 3, "\n")
print(2 ** 3, "\n")
print(Math.sin(3.14), "\n")
print(Math.sqrt(2), "\n")
```

☒ 7: sample3.rb

```
print("Hello\n")
print("Hello" + "ruby\n")
print("Hello" + " ruby" + " world!\n")
```

☒ 8: sample4.rb

```
print("Hello\n")
print("Hello" * 4 + "\n")
```

☒ 9: sample5.rb

```
a = 3
print(a, "\n")
```

図 10: sample6.rb

```
a = 3
b = 4
print("a = ", a, ",b = ", b, "\n")
```

図 11: sample7.rb

```
a = 2
b = 3

print(a, " + ", b, " = ", a + b, "\n")
print(a, " - ", b, " = ", a - b, "\n")
print(a, " * ", b, " = ", a * b, "\n")
print(a, " / ", b, " = ", a / b, "\n")
```

図 12: sample8.rb

```
i = 1
print("i = ", i, "\n")
i = i + 1
print("i = ", i, "\n")
i = i + 1
print("i = ", i, "\n")
```

図 13: sample9.rb

```
# 1 から 30 までの数を順番に表示するプログラム

i = 1
while i <= 30
  print(i, "\n")
  i = i + 1
end
```

図 14: sample10.rb



```
i = 0
while i < 10
  print(i + 1, "回目\n")
  i = i + 1
end
```

図 15: sample11.rb

```
print("Hello!")
print("Hello!")
print("Hello!")
print("-----\n")

i = 0
while i < 3
  print("Hello!!")
  i = i + 1
end
```

図 16: sample12.rb

```
# *を表示するプログラム
i = 1
while i <= 10
  j = 1
  while j <= i
    print("*")
    j = j + 1
  end
  print("\n")
  i = i + 1
end
```

図 17: sample13.rb

```
# 1 から 10 までの和を計算し、表示するプログラム
total = 0

total = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
print("total = ", total, "\n")

sum = 0
i = 1
while i <= 10
  sum = sum + i
  i = i + 1
end
print("sum = ", sum, "\n")
```

図 18: sample14.rb

```
sum = 0
i = 0
x = [10, 20, 35, 40, 50]

while i < x.length
  sum = sum + x[i]
  i = i + 1
end
print("sum = ", sum, "\n")
```

図 19: sample15.rb

```
puts("ジュース販売機です。1 本 120 円です")
puts("お金を入れてください")
n = gets.chomp.to_i

if n > 120
  print("ジュース 1 本とおつり", n - 120, " 円です。\\n")
elsif n == 120
  print("ジュース 1 本です\\n")
else
  print(120 - n, "円足りません\\n")
end
```

図 20: sample16.rb

```
sushi = {"かつば巻き" => 100, "いくら" => 300, "たまご" => 200}

print(sushi["かつば巻き"], "\\n")
print(sushi["いくら"], "\\n")
print(sushi["たまご"], "\\n")
```

図 21: sample17.rb

```
def hello(name)
  print(name, "さん、こんにちは。\\n")
end

hello("うに")
hello("かに")
hello("おに")
```

図 22: sample18.rb

(注) 画像ファイル <http://klis.tsukuba.ac.jp/klib/Subjects/ProgI/text/zu1.gif> も sample19.rb と同じフォルダに保存する。ひつじ何匹と表示されたら、適当な数値を入力し、Enter キーを押す。

```
require 'sdl'

print("ひつじ何匹?")
n = gets.chomp.to_i

SDL.init(SDL::INIT_VIDEO)
screen = SDL.setVideoMode(300, 600, 16, SDL::SWSURFACE )

#ひつじ表示
interval = 1
i = 0
while i < n
  loop_start = SDL.getTicks/1000.0
  x = i % 3 * 100
  y = 500 - (i / 3)*50

  chara = SDL::Surface.load('zu1.gif')
  screen.put(chara, x, y)
  screen.updateRect(0, 0, 0, 0)

  print("ひつじが", i + 1, "匹...");
  sleep(1.8)

  if i >= 29
    print("ひ... つ... じ... が....")
    break
  end

  i = i + 1
end

print("\n <-.... <-..... <-... ムニヤムニヤ... <-..... <-.....\n")
sleep(6)

print("\n\n\n")
print("おやすみなさい\n")
```

図 23: sample19.rb