はじめに

情報科学とは何か

- 「情報」に関する科学的な基礎
  - 様々な分野に現われる「情報」を扱うための科学
    - cf. 数学は様々な分野の「数理」を、物理は「物」を扱うための科学
    - 生物の理解: 個体→分子→情報
    - ロボット工学: 機械→制御→ソフトウェア
  - 科目「情報」の土台部分
  - 情報系の学科で学ぶことの入り口

授業形式・評価

- クラス指定
- 講義(2/3) + 演習(1/3)
- 講義はスライド・板書を用いる
- 演習はプログラミングを通して行う
  - プログラミングが主目的ではない!
- レポート課題
- 期末試験
- 成績 = 1/2試験 + 3/10レポート + 2/10出席
  - 出席は投票システム（後述）への投票回数で測る
  - 練習問題確認プログラムの報告は出席として加味

教科書

- 必ず手に入れるように

- サポートページ
  - http://lecture.ecc.u-tokyo.ac.jp/johzu/joho-kagaku/text/

授業計画

- 第1部 必要最小限のプログラミング
  - 第1章 数の計算と関数
  - 第2章 配列による画像の表示
  - 第3章 条件分岐と繰り返し
  - 第6章 数値計算（第2部より）
  - 第4章 関数から計算へ
  - 第7章 パターン認識（第2部より）
- 第2部 プログラミングを通して学ぶ情報科学の諸概念
  - 第5章 アルゴリズムと計算量
  - 第8章 レコードとオブジェクト
  - 第9章 データ構造と再帰
  - 第10章 いろいろなプログラミング言語

ご利益

- 情報システムが何ができるのか?が分かる
  - 「ゲノムデータベースから検索をするのにかかる時間は?」
  - 「ロボットに滑らかに動きをさせるのが大変なのは何故?」
  - 実際にやらせてみないと分からない!
- 情報科学の基礎知識がいくつか分かる
  - アルゴリズム/数値計算/パターン認識
- プログラミングの入門ができる(副作用)
  - 言語の違いは大きな問題ではない!

プログラミング言語Ruby

- 概念を理解するためにプログラミング演習を行う
  - 手段であり目的ではない
  - 簡単に試せるのも情報関係の特徴
- プログラミング言語 Ruby を用いる
  - 正確には irb という対話型の処理系～簡単に試せる
  - 実際には極めて実用的な言語(だが、この授業では深入りしない)
  - 産業界・科学技術研究開発で使われている他のプログラミング言語を学ぶ際の土台にもなる
  - 日本発のオープンソースソフトウェア

irb と isrb

- グラフィックス表示の機能 show を使うためには、irb の代わりに isrb を用いる。
- ただし、ECC の Mac OS X の端末の前でしか使えない。(リモートから使うとエラーになる。)

URL

- この授業のページ

  lecture.ecc.u-tokyo.ac.jp/~shagiya/
- 情報科学共通資料

  http://lecture.ecc.u-tokyo.ac.jp/johzu/joho-kagaku/text/
- 山口和紀先生のRuby入門とプログラム練習

  lecture.ecc.u-tokyo.ac.jp/~yamaguch/johokagaku/2010/ruby-primer.html

投票システム

- 目的
  - 出席状況の確認 ⇒ 成績
  - 授業の理解度の確認
  - 演習の進捗状況の確認
- 成績について
  - 授業中に質問に対して答えてもらう
  - 答えた回数を成績に加味する（全体の1/5）
    - 答えた内容は問わない

投票システム

- ダウンロード
  - lecture.ecc.u-tokyo.ac.jp/~shagiya/ の
    投票クライアント をクリック
  - ファイルに保存
    - ファイル名を指定しなければ、c.rb というファイルができる
    - ディレクトリに注意
    - よくわからなければ、ホームディレクトリに置けばよい
- 投票
  - ターミナルを開く
  - c.rb があるディレクトリに cd して、以下のコマンドを実行

    $ ruby c.rb 選択肢番号
  - 同じ質問に何回も投票した場合、 最終の投票のみ有効

科目「情報」は面白かったか

1. 面白かった
2. ふつう
3. つまらなかった

科目「情報」は難しかったか

1. 難しかった
2. ふつう
3. やさしかった

科目「情報」は役に立つと思うか

1. 役に立つと思う
2. どちらとも言えない
3. 役に立たないと思う

プログラミングの経験

1. 自由にプログラムを書くことができる
2. プログラムをしたことがある
3. プログラムをしたことがない

高校の情報

1. 情報Aを履修
2. 情報Bを履修
3. 情報Cを履修
4. 何を履修したかわからない
5. 未履修

練習問題確認プログラム

- http://lecture.ecc.u-tokyo.ac.jp/johzu/joho-kagaku/text/
- http://lecture.ecc.u-tokyo.ac.jp/johzu/joho-kagaku/text/appendices.pdf
- http://lecture.ecc.u-tokyo.ac.jp/johzu/joho-kagaku/text/code/

Twitter

- とりあえず

#ishagiya

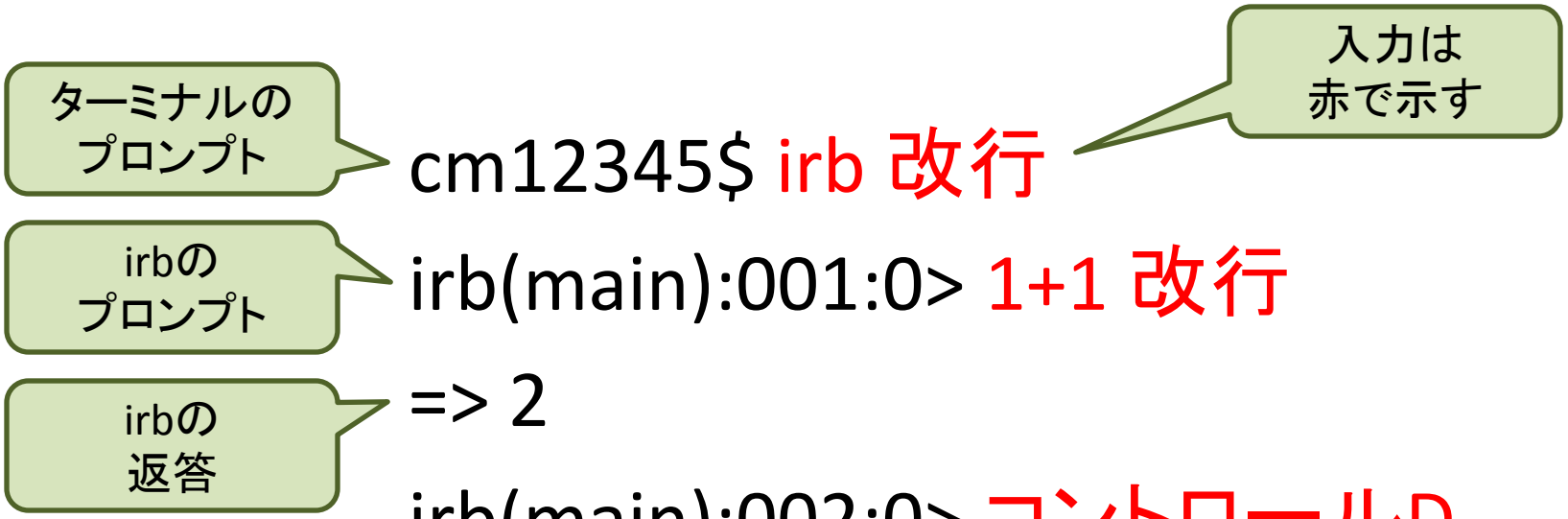
- レスポンスの保証はない。

@hagiya

数の計算と関数

コンピュータとの対話

- ターミナルの起動 ⇒ irbの起動 ⇒ 数式の入力



- 今回は、指示されるまで、irbを使いながら…

数式の計算 --- 電卓がわり

以下、
改行は省略

7を2で割った
余り

```
irb(main):003:0> 7 – 2
=> 5
rb(main):004:0> 7 * 2
=> 14
irb(main):005:0> 7 / 2
=> 3
```

```
irb(main):006:0> 7 % 2
=> 1
irb(main):007:0> 7 ** 2
=> 49
```

7の2乗

空白を入れても入れなくても
ただし、全角にしないように

ここには空白を
入れてはダメ

電卓がわり

```
irb(main):009:0> 7 - 2 * 3
=> 1
irb(main):010:0> (7 - 2) * 3
=> 15

irb(main):012:0> 7.0 / 2
=> 3.5
irb(main):013:0> 7 / 2.0
=> 3.5
```

17 - 17/3*3 の値は

1. 0.0
2. 0
3. 2
4. 15.111111111111
5. 16

56 の16乗として間違っているのは

1. 56 ** 16
2. (7 * 8) ** 16
3. 7 * 8 ** 16
4. 56 ** 4 ** 2
5. 56 ** (4 ** 2)

さまざまなエラー

```
irb(main):001:0> 3/0
ZeroDivisionError: divided by 0
        from (irb):1:in `/'
        from (irb):1
irb(main):002:0> 7 - 2 3
SyntaxError: compile error
(irb):2: syntax error, unexpected tINTEGER, expecting $end
        from (irb):2
irb(main):003:0> (7 -
irb(main):004:1* 2) * 3)
SyntaxError: compile error
(irb):4: syntax error, unexpected ')', expecting $end
        from (irb):4
irb(main):005:0>
```

式の途中で改行すると
プロンプトが異なる

さまざまなエラー

```
irb(main):013:0> bm1(188.0, 104.0)
NoMethodError: undefined method 'bm1' for main:
  Object
    from (irb):13
```

わけがわからなくなったら

- ともかくコントロール C を押す
  - irb はトップレベルの戻る

数学関数

```
irb(main):003:0> include(Math)
=> Object
irb(main):004:0> sqrt(2)
=> 1.4142135623731
irb( main ):005:0> cos(3.141592/3)
=> 0.50000018867511
```

数学関数を使う準備
irbを起動し直す
たびに必要

黄金比の値は

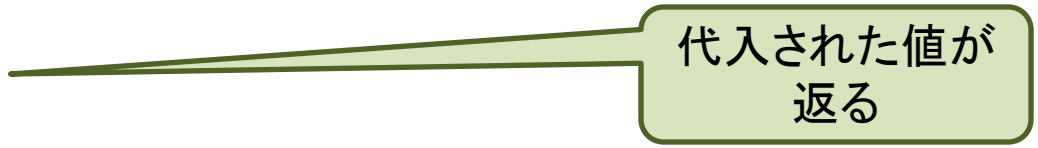
1. 1.61803398874989
2. 1.61803398874988
3. 1.61803398874987
4. 1.61803398874986
5. 1.61803398874985

$$\frac{1+\sqrt{5}}{2}$$

変数 --- 値に名前を付ける


```
irb(main):003:0> h=188.0
=> 188.0
irb(main):004:0> w=104.0
=> 104.0

irb(main):006:0> w / (h/100.0) ** 2
=> 29.4250792213671
```



変数を使うわけ

- 式の意味が理解しやすくなる

  w

  weight

  body_weight_in_pound

変数（局所変数）は、
小文字で始まる英数字列
アンダースコアは
小文字と考える

- 違う値で計算のやり直しができる

  irb(main):008:0> w=104.0-10

  => 94.0

  irb(main):009:0> w / (h/100.0) ** 2

  => 26.5957446808511

w=w-10
としてもよい

irbへの入力

- コントロールPもしくは上矢印を入力すると、直前の入力が復活する。
- コントロールB（もしくは左矢印）でカーソルは左に移動。
- コントロールF（もしくは右矢印）でカーソルは右に移動。
- 通常の文字はカーソル位置に挿入される。
- コントロールDでカーソル位置の文字が削除される。
- バックスペースでカーソルの直前の文字が削除される。

関数の定義 --- BMIを求める関数

```
irb(main):003:0> def bmi(height , weight)
irb(main):004:1>     weight / (height/100.0) ** 2
irb(main):005:1> end
=> nil

irb(main):007:0> bmi(188.0, 104.0)
=> 29.4250792213671
irb(main):008:0> 1.1*bmi(174.0, 119.0 * 0.454)
=> 19.6289470207425
```

練習1.2

1. 平面上の 2 点 (x, y) と (u, v) の距離を求める distance(x,y,u,v).
2. f フィート i インチをセンチメートルに変換する feet_to_cm(f,i). ただし、1 フィート = 12 インチ = 30.48 cm である。
3. p ポンド o オンスをキログラムに変換する pound_to_kg(p,o). 1 ポンド = 16 オンス = 0.4536 kg である。

進捗状況の確認

1. すべてできた
2. 二つできた
3. 一つできた
4. できない

解答

```
def  distance(x, y, u, v)
   sqrt((x-u)*(x-u) + (y-v)*(y-v))
end
def  feet_to_cm(f, i)
   30.48*f + 30.48/12*i
end
def  pound_to_kg(p, o)
   0.4536*p + 0.4536/16*o
end
```

ここまで

1. 理解した
2. よくわからないところがある
3. 全然わからない

ここで手を休めよう

関数を使う関数

関数も、
小文字で始まる英数字列
アンダースコアは
小文字と考える

```
irb(main):010:0> def bmi_yp(f,i,p,o)
irb(main):011:1>     bmi(feet_to_cm(f,i),
irb(main):012:2*         pound_to_kg(p,o))
irb(main):013:1> end
=> nil

irb(main):015:0> bmi_yp(5,11,170,0)
=> 23.7099441655737
```

式の途中で改行すると
プロンプトが異なる

ファイルに保存した関数定義 ---ファイルからの読み込み

#以下
行末まで
コメント

```
# BMI of a person with height (cm) and weight (kg)
def bmi(height , weight)
   weight / (height /100.0) ** 2
end
```

bmi.rb

```
irb(main):003:0> load("./bmi.rb")
=> true

irb(main):005:0> bmi(188.0, 104.0)
=> 29.4250792213671
```

ファイルを読み込むファイル

```
load ("./bmi.rb")
load ("./yardpound.rb")


def bmi_yp(f,i,p,o)
    bmi(feet_to_cm(f,i), pound_to_kg(p,o))
end
```

bmi_yp.rb

定数関数

```
# BMI of a person with height (cm) and weight (kg)
def bmi(height , weight)
   weight / ( height/100.0) ** 2
end
def k_height()       #K選手の身長
   188.0
end
def k_weight()       #K選手の体重
   104.0
end
```

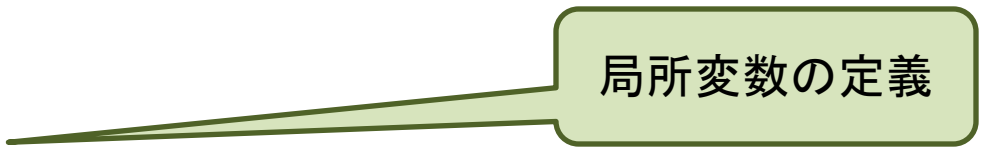
bmi.rb

定数関数

```
irb(main):004:0> load("./bmi.rb")
=> true
irb(main):005:0> k_weight()
=> 104.0
irb(main):006:0> bmi(k_height(), k_weight())
=> 29.4250792213671
```

局所変数

```
def heron(a,b,c)
    s = 0.5*(a+b+c)
    sqrt(s * (s-a) * (s-b) * (s-c))
end
```



heron.rb

エディタは

1. Emacs を使ったことがある
2. その他のエディタを使ったことがある
3. 使ったことがない
4. エディタって何？

何とか .rb という名前のファイルの

1. 作り方がわかる
2. .rb の指定方法がわからない
3. .rb って何？

Emacs

- Emacs (イーマックス) は歴史の長い代表的なテキスト・エディタである。
- コントロール・キーを使うのが由緒正しいが、とりあえず、メニューや矢印キーを使ってテキストを編集しよう。
- irb でロードする前にセーブを忘れずに。
- 自分の慣れたエディタがあれば、それを使ってもよいです。ただし、プレーン・テキストのファイルが作れないと駄目。

Emacs のコントロールキーの例

← … ^B (backward)

↑ … ^P (previous)

↓ … ^N (next)

→ … ^F (forward)

カーソルの下の文字の消去 … ^D (delete)

カーソルから行末までの消去 … ^K (kill)
（連続すると行の消去）

ペースト … ^Y (yank)

練習

- 定数関数 k_height と k_weight も定義されたファイル bmi.rb を作り、ロードして動作を確認せよ。
- ファイル heron.rb を作り、ロードして動作を確認せよ。

進捗状況の確認

1. すべてできた
2. bmi だけできた
3. heron だけできた
4. ファイルが作れない
5. ロードがうまくいかない
6. 関数が実行できない（エラーなど）

できてしまった人は、次のページの演習を
やってください。

練習1.9g)

- 二次方程式 $ax^2 + bx + c = 0$ に関して

  (a) 判別式 $b^2 - 4ac$ を求めるdet(a,b,c).

  (b) 解の1 つを求める solution1(a,b,c). (det を使って定義せよ。)

  (c) もう 1 つの解を求める solution2(a,b,c). (solution1 と solution2 の共通部分を1つの補助関数にできるか?)

  (d) 二次関数$f(x) = ax^2+bx+c$ の値を求める quadratic(a,b,c,x).

  各定義をquadratic.rbという名前のファイルに格納せよ。

1.6　定義のまとめ

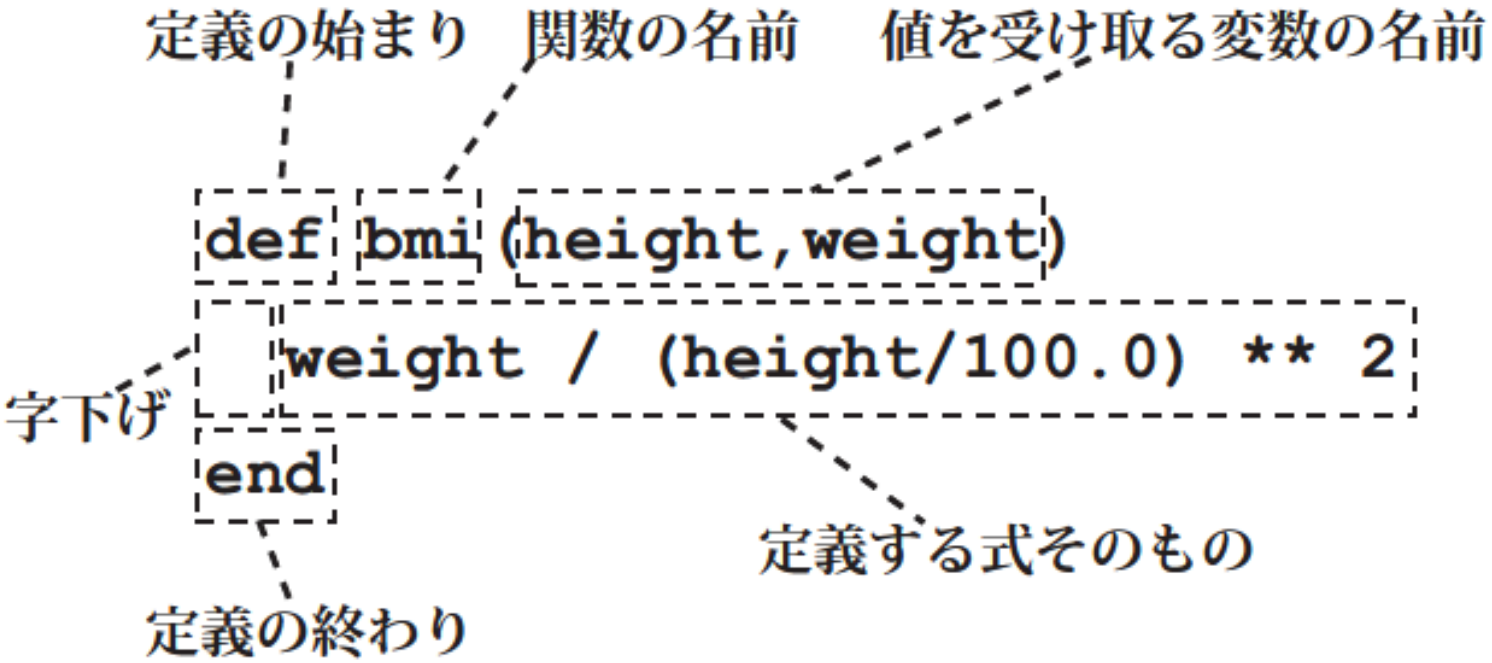
この章で紹介した式や命令をまとておこう。

`include(Math)`： `cos` や `sqrt` などの数学関数を使う前に、実行しておかなければいけない命令。

変数名 = 式：右辺の式を計算した値を、左辺に書かれた名前の変数にしまう**代入命令**。

関数名(式$_1$,...,式$_n$)：　式$_1$,...,式$_n$を計算した値を、関数に渡す**関数呼び出し式**。関数に渡される式の値のことを**引数**(ひきすう)という。関数名は `sqrt` のような予め定義されている関数でも、`bmi` のような自分で定義した関数でもよい。

def 関数名 (変数名$_1$, . . . , 変数名$_n$) ↵ 式 ↵ end : **関数定義**。少し複雑なので、具体例で説明する。



def と end は、定義の範囲を示している。**関数の名前**(ここでは bmi) は、変数名と同様、好きな名前を付けることができる。次の (height,weight) は、関数が値を受け取るために使う変数名である。

練習問題確認プログラム

- 練習1.3と1.9と1.10をやる
- check.rb と ex01.rb をダウンロード
- ターミナルの中で以下のように実行する

  $ ruby check.rb ex01.rb
- うまくできたことを確かめたら以下を実行

  $ ruby check.rb --report ex01.rb
  - 結果、ex01-091021-204855.tgz というようなファイルが作られる
  - これをメールに添付して送る
  - メールの本体には学生証番号と名前を忘れずに

練習問題確認プログラム

- ex01.rb（練習1.3と1.9と1.10）の結果を以下のアドレスに送れ。

  is-komaba@lyon.is.s.u-tokyo.ac.jp

- Subject:欄は、

  is-komaba

  としてください。

- ✍切: 10月XX日
  - 練習問題確認プログラムの結果は出席点の一部とします。結果の内容は問いません。（習得状況を知りたい。）

笹田先生のクラス（火曜日）は
ko1+johokagaku@atdot.net