Ruby を用いて情報科学の原理を教える

久野　靖*

2009.9.7

1　はじめに

□ 竹内先生の講演 --- 「すごい人材を (多いに越したことはないけれども) 少数精鋭で育成する」にはどうすべきか (たぶん)

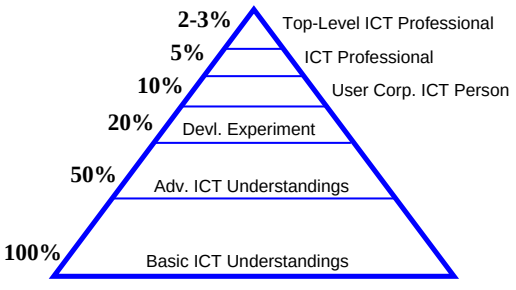
□ もう 1 つ必要なこと --- 「世の中一般の人を『ある程度分かっている』人にする」

- 初等中等教育 (小・中・高) における情報教育
- 大学 1 年次における情報教育 (一般情報教育)
- 情報系学科以外の (理系?) 専門に進む学生に対する教育 ←☆

□ なぜ「世の中一般の人を『ある程度分かっている』人にする」必要?

- 山は「頂上だけ」を作ることはできない←裾野の広がりが不可欠
- 「すごい人材」が生まれてもそれを社会が評価できないと不幸
- 「すごい人材」だけでなく「できる人材」「普通の人材」の階層があってはじめて IT 業界は成り立っているはず
- 「できる人材」「普通の人材」を正しく評価できる人はさらに多数必要

□ 情報技術に対する理解の「ピラミッド構造」が必要



□ 現在「世の中の人が『全然分かっていない』」ことの弊害…

- 「コンピュータだから何でもできるんだろう/正しいんだろう」(思考停止) →「これはできるがこれはできない」という説明を受け入れてくれない
- 「何やら面倒なことだが専門家に金を払えばシステムができてくる」(思考停止) →「なんでこんなに金が掛かるの」「安いに越したことはない」(品質ということばを受け入れてくれない)
- 「自分がやっている『本当の仕事』(業務知識?) よりずっと下賤なことであり、自分が理解する必要はない」(思考停止) →まっとうな要求仕様ができるはずがない

□ このような「思考停止」が世の中の隅々まで広まっている→ 「IT 業界は 7K」(きつい、給料が安い、暗い = 新 3K) + (休暇が取れない、帰れない、結婚できない、化粧が乗らない)

- 業界もこのようなイメージを助長する面があったのでは? (派遣ベース)
- 世の中のイメージが悪い → 優秀な人が (人ほど) この進路を選ばない

□ だからやっぱり、「世の中一般の人を『ある程度分かっている』人にする」べきでは?

- 初等中等教育 (小・中・高) における情報教育 --- 努力中 (文献参照)
- 大学 1 年次における情報教育 (一般情報教育) --- 努力中 (文献参照)
- 情報系学科以外の (理系?) 専門に進む学生に対する教育 ← 今日のお話

2　東大 1 年次科目「情報科学」

2.1　「情報科学」の概要

□ 東京大学の 1 年次情報教育科目→前期・後期各 1 科目

□ 前期 → 「情報」(必修) (いわゆる一般情報教育の科目)

- (他の多くの大学と異なり) ソフトの操作方法教育を目標としない
- 情報の人間的側面 (表現、認知、伝達)
- 情報の社会的側面 (情報システム、法、技術と社会)
- 情報の問題解決側面 (データ、計算モデル、複雑さ)

*筑波大学ビジネス科学研究科

- 「難しい」という意見が多いらしい…(本当に難しいのかどうかは?)

□ 後期 → 「情報科学」(理系:クラス指定、文系:選択)(かなり特殊)

- 以前の「計算機プログラミングI」(選択)を発展的に解消(2006〜)
- 「情報科学の基本概念や思考方法をプログラミングを通して習得」
- 「これらの概念の学習の手立てとしてRubyによるプログラミングを使用」

□ 興味深い点

- 情報科学の基本概念や思考方法を理系(できれば)全員に学んでもらう
- →理系全員を「ある程度分かっている人」にすることを目指している(かも)?
- Rubyの採用(2007〜)
- →伝統的にはCとかJavaだったがRubyだとどうか?

2.2 経緯ないし歴史

□ 〜2005: 計算機プログラミングI

- 普通の「プログラミング入門科目」
- 言語はJavaが標準、具体的内容構成は担当教員に任されていた

□ 2006: 「情報科学」試行(この年からクラス指定)

- 内容構成を設計→「標準スライド」(東大の関係教員が分担作成)
- 言語は「RubyまたはJava」→久野はJavaで実施、スライドと別に資料作成

□ 2007: 「情報科学」本格実施

- 内容構成は「標準スライド」の改訂(試行に比べて易しくした)
- 久野はこの年からRubyで実施、Java版の資料をRubyに書き換え
- →これを改訂して「Rubyによる情報科学入門」を編纂

□ 2008: 「情報科学」本格実施2年目

- 「標準スライド」に加えて増原英彦先生が「標準テキスト」を執筆
- 標準スライド/テキストでは独自画像/アニメーション表示ライブラリ使用
- 久野は前年度資料(〜書籍原稿)を改訂して使用

□ 「扱う内容範囲は定めるが扱い方や順序は各担当に任せる」という東大様の方針のおかげで、好きにやらせていただいている

- 問題: 履修率の低さ --- クラス指定(他の選択科目はない)にも関わらず、1類(理・工学部)で50%、2・3類(生命・農・医)で10〜20%程度→ 先に述べた「思考停止」のせい? → 鶏と卵

3 授業設計と内容

3.1 授業設計

□ 内容範囲は標準スライドをカバーするものとする(科目の前提)

- 順番については「コードにした時に易しい」ものが先に来るように

□ 各内容は(原則として)Rubyプログラムで確認・追体験できるようにする

- 確認・追体験の内容が演習問題となる→小レポートとして提出
- 演習の数は多くなるので、難易度を広くとり学生の選択に任せる

□ 上記が可能になるように「Rubyプログラムが書ける」ことも目標

- → できれば、プログラミングの楽しさも知って欲しい

□ 各回(90分)、おおむね2つ程度のトピックを紹介

- 各回の資料は授業の数日前にPDFで公開(印刷して持参を推奨)
- 1つ目について講義/説明した後、対応する例題を「そのまま」打ち込み動作確認してもらう
- 続いてそれを多少手直しするとできる演習をやってもらう(A課題) →この演習内容を「当日レポート」として出席代わりに提出してもらう
- ここまででだいたい50〜60分(説明20分+演習30分)
- 頃会いを見計らって演習を中断し、2つ目のトピックを講義/説明
- これに対応する演習から2つ選んで次回までに提出してもらう(B課題)
- その解説は次回講義の冒頭に行う(「問題提示→考えてもらう→解説」の順にしたいため)

□ 全部で 14 回

- ＃ 1〜＃ 11 --- Ruby により実施 (上記の通り)、B 課題は＃ 8 まで
- ＃ 12 --- 「さまざまな言語」(各種言語の紹介、Java を演習)
- ＃ 13(グラフィクス、GUI)、＃ 14(クラス階層設計) --- Java/出席自由 (強い型の言語も多少やりたいため)

□ 以下では＃ 1〜＃ 11 の部分 (Ruby で情報科学) の部分について内容を紹介

3.2 ＃ 1: アルゴリズムとプログラム/数値の表現

□ 計算の手順がアルゴリズムで、それをコンピュータで実行できるように表記したものがプログラムだ、という話 (前期の復習)

- Ruby の「三角形の面積」プログラムを提示しそのまま動かさせる。

```
def triarea(w, h)
  s = (w * h) / 2.0
  return s
end
```

- 動いたら「2 数の和」「円柱の体積」など同様の (単純計算の) プログラムを書かせる
- すべての例題はメソッドの形で書き、エディタでファイルに打ち込ませ、irb でロードして引数をつけて呼び出し実行させる
- irb で直接式を計算させるのは「わざと」避けている ← 少し複雑になったら結局ファイルに書いて繰り返し修正するので
- Java に比べると、入力する行数が圧倒的に少なく、おまじないが少ないため、最初の演習に掛かる労力は非常に少なくなった。これはさすがだと思った。
- 「エディタでファイルに打ち込む」というのを丁寧に説明する必要はあった (しかし多少苦労した学生も 1 週間たてばまったく OK)
- 「プログラムが記述した通りに動くととても嬉しい」「非常に細かい (1 文字違っただけでも動かない) のは大変」という感想が多く出される

□ 数値の表現について、整数の 2 の補数表現と、実数の浮動少数点表現の説明。数の有限性/有限精度、誤差 (情報落ち、桁落ち) の説明

- Java のときは整数があふれて負になるとかを体験する課題を出していたが、Ruby は勝手に多倍長にするためこれができない。しかたないので時間計測法を提示して「ある値から先は計算時間が長くなるが、そのある値はいくつか調べよ」という課題にした
- 浮動少数点の誤差を実際に観察する課題。非常に緻密に調べてくれた学生もいるし、適当にやって誤差がありました、程度の学生もいた
- 全体として、単なる計算のコードだけでできる課題なので初回の B 課題にぴったり
- いろいろ実験してくれている。「コンピュータの計算が正確でないことを知って大変びっくりした」「やってみるとコンピュータの計算のしくみが分かる」という感想が聞かれる

3.3 ＃ 2: 制御構造/数値積分

□ 前述の通り、課題として出した整数演算時間の段差と実数の誤差の問題について解説

□ 制御構造がないと何もできないので、とにかくまず if 文を教える。

- 例題は「絶対値の計算」で、1 つの動作に複数の書き方があることを強調

```
def abs1(x)
  if x < 0
    result = -x
  else
    result = x
  end
  return result
end

def abs2(x)
  if x < 0
    return -x
  else
    return x
  end
end

def abs3(x)
  result = x
  if x < 0 then result = -x end
  return result
end
```

- これを打って動かした後「2 数のより大きいものを打ち出す」「3 数の〃」を演習してもらう。3 つ、4 つになると初心者はそれなりに苦労するので楽しんでもらえる
- 制御構造があると「プログラムをしているんだ」という気分になるという感想がある

□ if 文が終わったら while 文を説明し、「積分の公式を知らなくても積分が求められる」と言って数値積分の概念を説明する

- 例題は while 文を使って 2 次関数の定積分 (正解が分かっている) を求める→誤差がバラバラ→dx=W/N を累計しても誤差のためループが N+1 回になったりすることを示す→回数を決めてループしカウンタから x を求める→計数ループを教える→刻みを小さくするとそれに応じて安定に誤差が減っていることを示す

```
def integ1(a, b, n)
  dx = (b - a) / n;
  s = 0.0
  x = a
  while x < b
    y = x**2        # 関数 f(x) の計算
    s = s + y * dx
    x = x + dx
  end
  return s
end

def integ2(a, b, n)
  dx = (b - a) / n;
  s = 0.0
  n.times do |i|
    x = a + i * (b - a) / n;
    y = x**2        # 関数 f(x) の計算
    s = s + y * dx
  end
  return s
end
```

- 単調増加な関数で左端公式 (?) だと常に小さいことを説明し、ヒント (中点公式、台形公式、両者の混合の考え方) を示して次回までの演習問題としてやらせる
- 計数ループは for 文も一応教えるが、サンプルコードでは「数値.times do ... end」が多い (好みと、step も使いたいから)
- 積分は理系にとってはなじみあるテーマなので分かりやすいという感想が多い

3.4 # 3: 制御構造/f(x)=0 の求解/データ型

□ 前回課題の説明として、シンプソンの公式による積分の説明と打ち切り誤差の話をする。テーラ展開式をそのまま計算すると誤差でめちゃくちゃになる話もする

□ 制御構造の続きとして fizzbuzz 問題を取り上げ、計数ループの中に if が入るパターンを演習してもらう。また素数判定/素数列挙の問題も提示

- fizzbuzz 問題: 「1 から 100 まで打ち出すが、3 の倍数では代わりに fizz、5 の倍数では buzz、両者の公倍数では fizzbuzz と打ち出す」
- 「世界のナベアツ」のようなバリエーションも可能
- この問題自体は分かりやすいので楽しんでもらえている感じ

□ f(x)=0 の解も数値的に解けるよという話をしてから、単調増加関数を前提として区間 2 分法の説明とニュートン法の説明を (あらすじだけ) して、そのコードを自力で演習として書いてもらう

□ 基本データ型 (数値、文字、論理値、nil) と複合データ型 (配列、レコード、オブジェクト) の説明

- Ruby だと文字列がオブジェクトで文字型がないので苦しい。また、シンボル型があるのでどうしようかと思ったが説明してしまう (レコードを作る時に必要)
- 配列については沢山つかうので具体的例題で説明。配列内容の合計を取る程度
- 素数列挙の改良として割ってみるのを$\sqrt{}$までででやめる等を例示し、その後「素数を配列に取っておいて素数だけで割ってみる」「エラトステネスのふるい」の考えを示して作ってみる課題を説明
- このあたりで、繰り返し、枝分かれ、配列を使ったプログラムがそれなりに書けるようになった感じ

3.5 # 4: 手続き (関数) と副作用/レコードと画像

□ 前回課題の説明として素数列挙の問題を中心にプログラム例を解説

□ メソッドを分けることで「抽象化」がなされていたことを後づけで指摘。またグローバル変数の概念を説明し、グローバル変数を書き換えると副作用のあるメソッドになることを説明。続いて再帰関数の概念を GCD(最大公約数) で説明 (前にループによる最大公約数をやっているので)

```
def gcd(x, y)
  if x == y
    return x
  elsif x > y
    return gcd(x-y, y)
  else
    return gcd(x, y-x)
  end
end
```

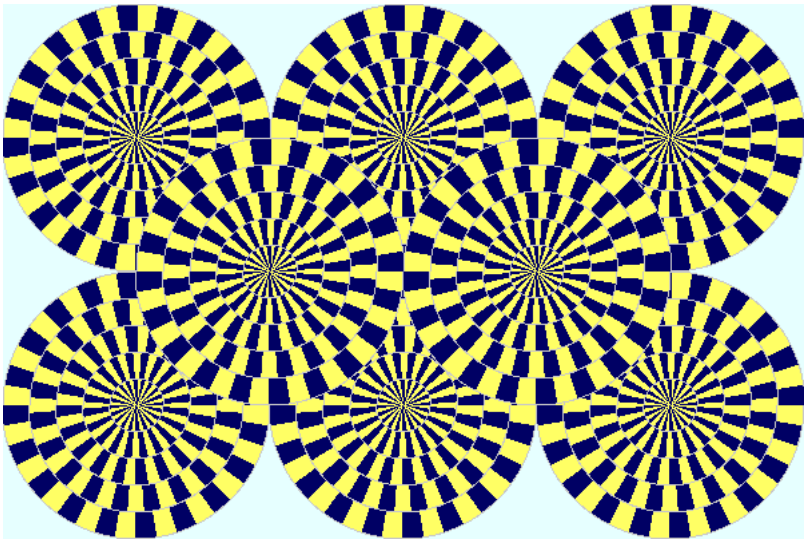
- 階乗、組み合わせの数、再帰的定義、非負整数の 2 進表現の 3 つについて再帰的定義を提示し、再帰メソッドを書かせる演習
- 例題と同様のパターンで書けばできるのですんなり再帰に入れるという印象

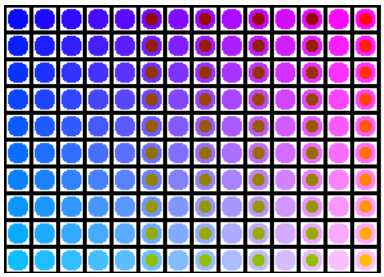
□ レコード型について解説し「Pixel = Struct.new (:r, :g, :b)」でピクセル値を表すレコードを例示。これの 2 次元配列で画像を表現できることを説明

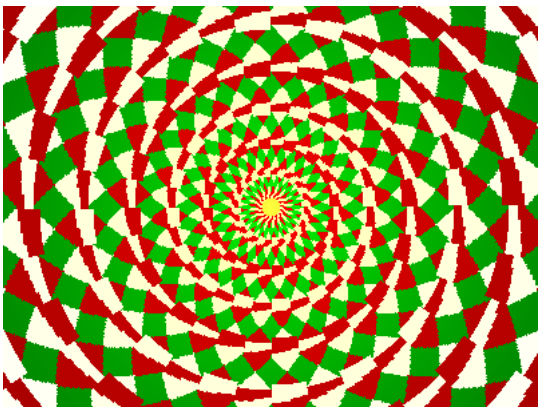
- `initimage`(画像データの初期化)、`writeimage`(ファイルへのPPM形式での書き出し)、`fillcircle`(円を塗りつぶし) を示し、この程度で画像が作れることを示したあと、長方形や直線などを作る課題と「美しい絵」を作る課題を提示

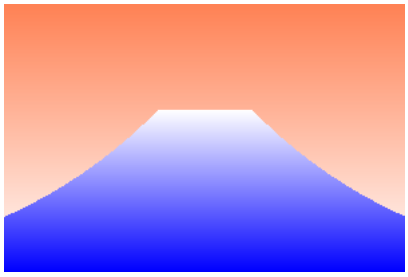
```
def mypicture
  initimage
  fillcircle(150, 30, 60, 255, 100, 70)
  fillcircle(190, 100, 30, 100, 200, 80)
  writeimage("t.ppm")
end
```

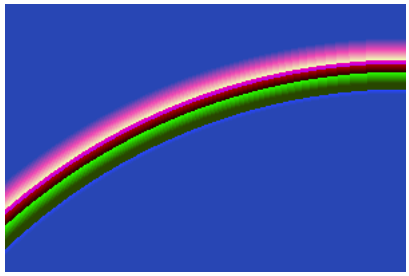
□ 「美しい画像」について、簡単な絵でも構わないとしたが、学生は皆いろいろ頑張ってくれた。

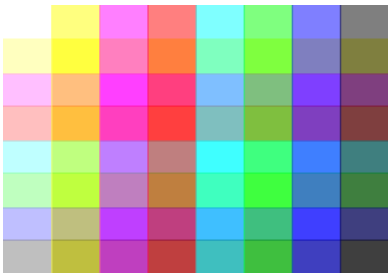


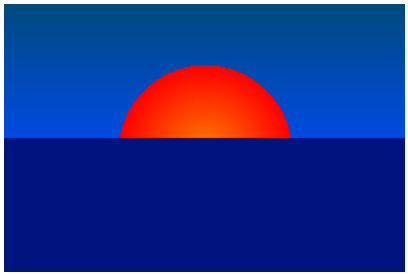


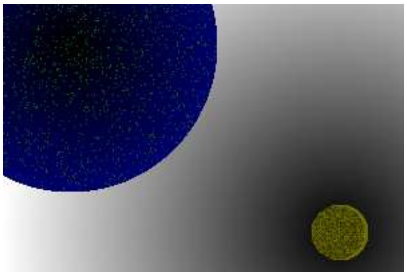


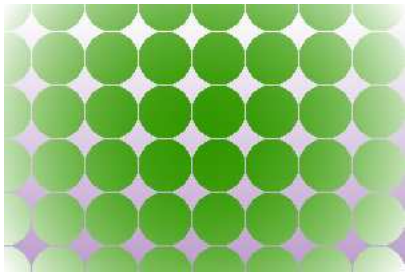


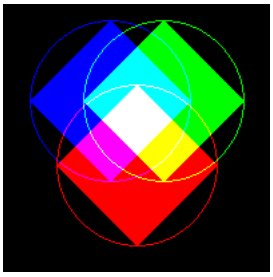


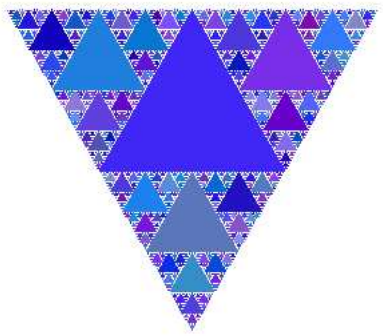


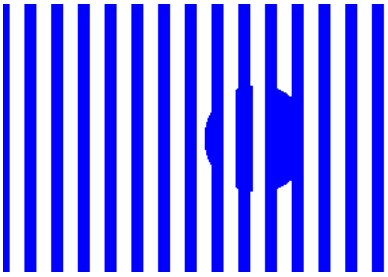


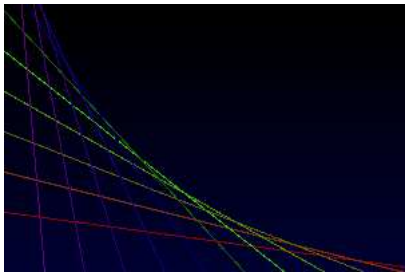


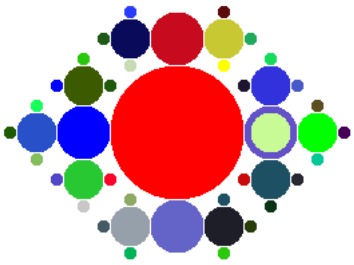


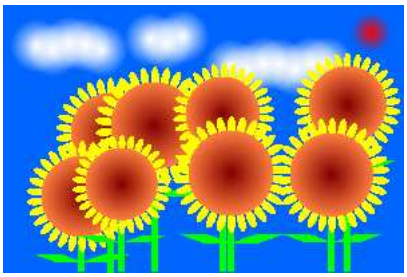


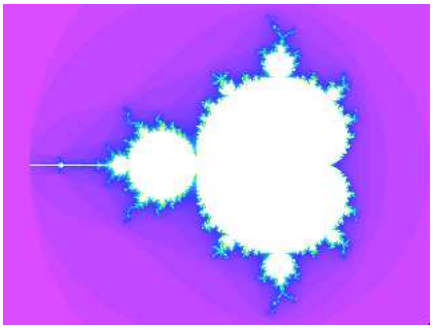


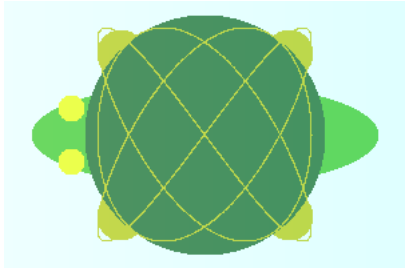


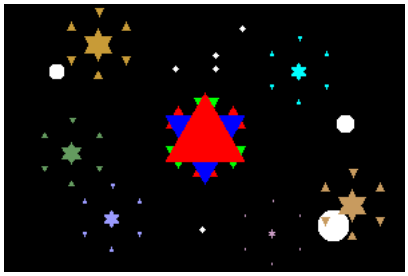


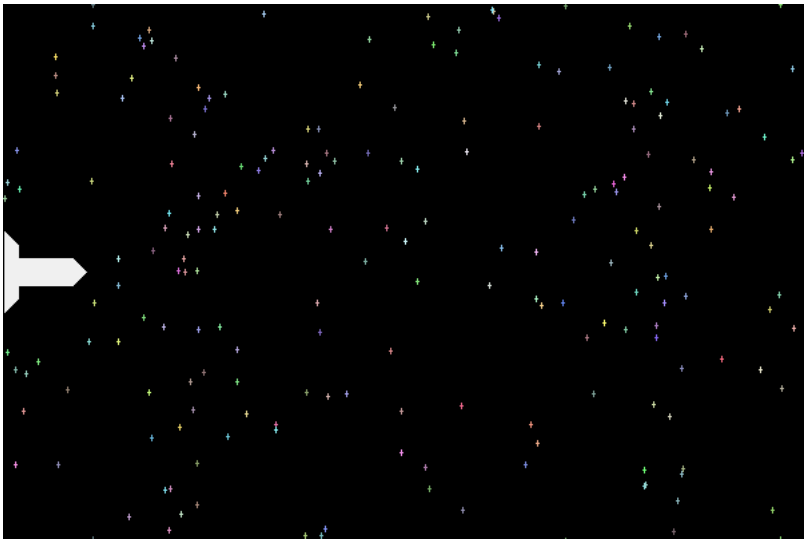


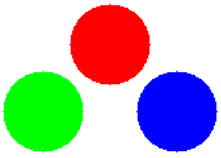


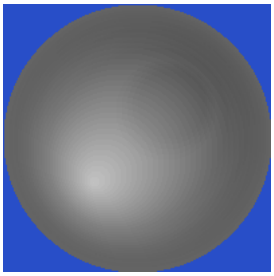


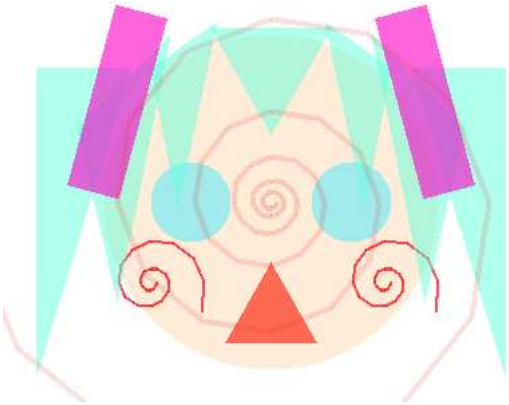


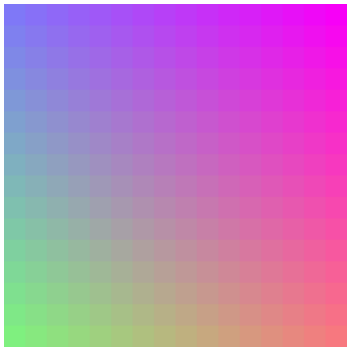


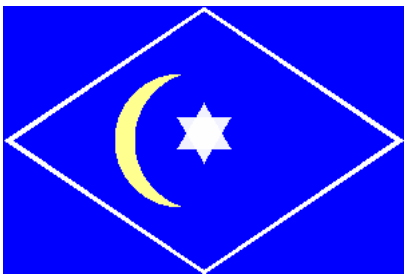


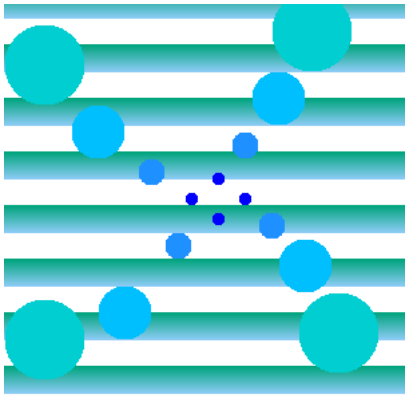


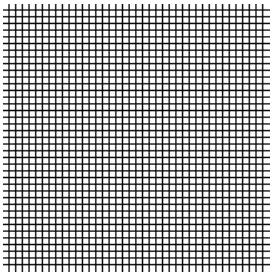


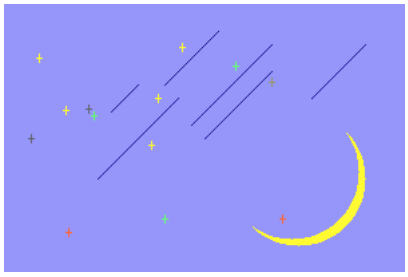


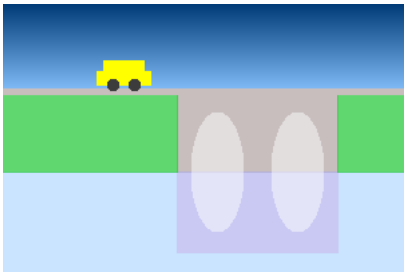


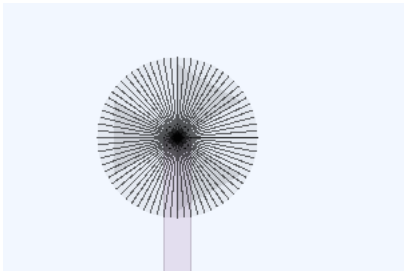


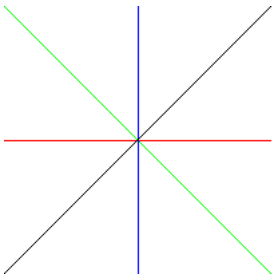


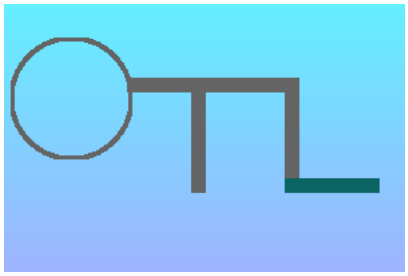


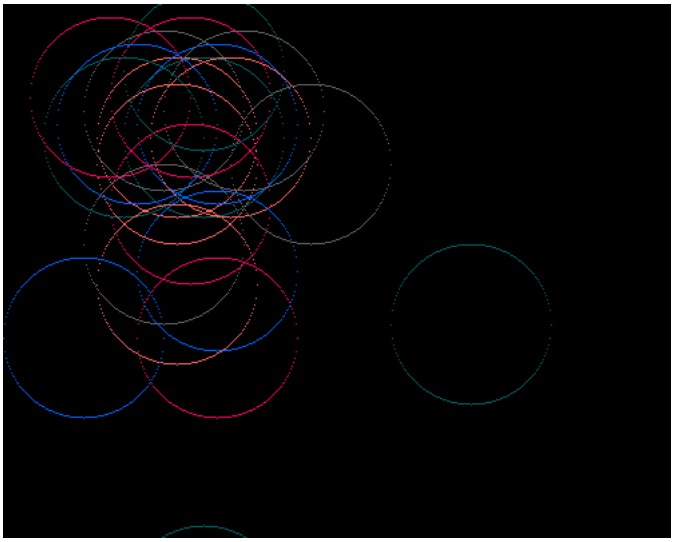


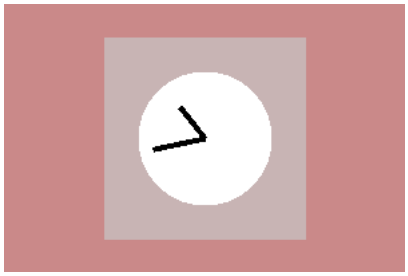












3.6 # 5: さまざまな整列手法/時間計算量

□ 配列の整列について # 3 で出した課題の解説として単純選択法、バブルソートを示し、N を変化させて時間計測を行う演習

□ マージソート、クイックソートを解説

□ 時間計算量の考え方を解説

□ ビンソート、基数ソートの考え方を解説し実装する課題を提示

□ その他の題材として最大公約数のユークリッド互除法、フィボナッチのループ版と 22 行列版、組み合わせのパスカル三角形版を考え方だけ示し実装と時間計算量の計測/見当を課題として提示

- 整列はテーマが分かりやすいためか納得してもらった感じ
- 計算量については計測してみると納得という感じで、一応計算量の考察ができるようになる

3.7 # 6: 連立方程式/常微分方程式の数値解法

□ 前回課題の各種アルゴリズムの実装例と時間計算量分析

□ 連立一次方程式のガウス消去法を説明し、Gauss-Jordan に直す演習。また Jacobi 法の原理と実装例を説明しテスト実行による収束の検討と Gauss-Seidel 法に直す演習

- 配列操作のロジックなので、このあたりにまだ慣れていない人はかなり苦労していた

□ 常微分方程式とは何か整理した後、Euler 法、Runge-Kutta 法（2 次）の考え方とコード、および Runge-Kutta 法（4 次）の考え方と公式までを示し、精度を検討するかまたは 4 次の実装をする演習

- 誤差の検討はできるが、アルゴリズムについてはこのようなものという程度の理解?
- この回は数値計算ばかりなのでつまらない? （2007 年度は半分を乱数・ランダムアルゴリズムにしていたのでより良かったかも知れない）

3.8 # 7: オブジェクト指向・乱数

□ オブジェクト指向とはプログラムが扱うデータを「もの」として考える「考え方」であると説明し、クラス、インスタンス、インスタンス変数、インスタンスメソッドなどの言語機構/概念を解説

- 例題として「犬」のクラスを示しその変更を演習として提示

```
class Dog
  def initialize(name)
    @name = name; @speed = 0.0; @count = 3
  end
  def talk
    puts('my name is ' + @name)
  end
  def addspeed(d)
    @speed = @speed + d
    puts('speed = ' + @speed.to_s)
  end
  def setcount(c)
    @count = c
  end
  def bark
    @count.times do puts('Vow! ') end
  end
end
```

- 「有理数」クラスを示しその演算追加と「複素数」クラスの作成を演習として提示
- 犬のクラスとかは簡単だったという意見が多い
- クラスを何に使うのか納得してもらえるのかと思ったが、分かったという意見も多い

□ 乱数と擬似乱数、ランダムアルゴリズム、モンテカルロ法による数値積分について説明し適当な関数の積分と試行数による精度分析を課題として提示

□ 乱数を使ったゲームの例として「数当て」を説明し、任意のゲームを作ってにみるという課題を提示

```
def kazuate
  a = (rand(10000)+10000).to_s[1..4]; count = 0
  while true do
    printf("your guess? ")
    s = gets; hit = 0; blow = 0
    4.times do |i|
      4.times do |j|
        if s[i] == a[j] then
          if i == j then hit = hit + 1
          else           blow = blow + 1
          end
        end
      end
    end
    if hit == 4 then puts "you win!"; return end
    count = count + 1
    if count > 9 then
      puts "you lose! answer = #{a}."; return
    end
    puts "hit = #{hit}, blow = #{blow}."
  end
end
```

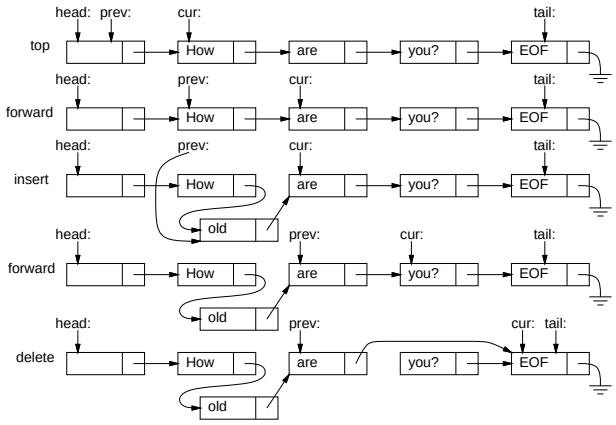
□ ゲームを作りたい人はかなり頑張ってゲームを作り提出してくれた (文字だけでやる RPG とか)

- ゲームを作ってみると「作れるものが作れるようになった」という感じを持った人がそれなりにいる

3.9 # 8: 動的データ構造/表と探索

□ 動的 (再帰的?) データ構造について説明し、単リストとその操作を説明

```
class Buffer
  Cell = Struct.new(:data, :next)
  def initialize
    @tail = @cur = Cell.new("EOF", nil)
    @head = @prev = Cell.new("", @cur)
  end
  def atend
    return @cur == @tail
  end
  def top
    @prev = @head; @cur = @head.next
  end
  def forward
    if atend then return end
    @prev = @cur; @cur = @cur.next
  end
  def insert(s)
    @prev.next = Cell.new(s, @cur); @prev = @prev.next
  end
  def print
    puts("  " + @cur.data)
  end
```



□ 演習として「現在行削除」「現在行と次の行の交換」「1 つ戻る」「全体の順序逆転」から 1 つ作ってもらう

- おまけとしてファイル読み書き、置換コマンドの実装を説明 (本当にエディタとして使えるようにするために不可欠)
- 単リストによる行バッファはかなり「面白い!」という反応が多かった→課題提出も多く人気があった
- バッファという「抽象データ型」を作っているという感じもする

□ 表と探索の概念を説明し、線形探索、2 分探索木の実装例を時間計測例とともに示す。時間計算量の検討を課題として提示

- Ruby の Hash について表の一種として説明し、具体的な記法や使い方を示したあと、これの時間計測も課題として提示
- こちらはあんまり人気はなかった

3.10 # 9: 抽象構文木/動的分配/字句解析/再帰下降解析

□ この回から A 課題 (当日課題) だけになる

□ 式木/抽象構文木の概念を説明し、簡単な式木を組み立てて評価する例題を提示した後、演算の追加や制御構造の追加を演習

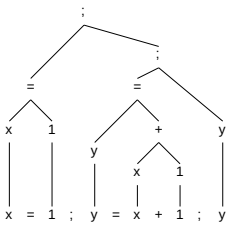
```
$vars = {}
class Add
  def initialize(l, r) @left = l; @right = r  end
  def exec() return @left.exec + @right.exec end
  def to_s()
    return '(' + @left.to_s + ' + ' + @right.to_s + ')'
  end
end
class Mul
  def initialize(l, r) @left = l; @right = r end
  def exec() return @left.exec * @right.exec end
  def to_s()
    return '(' + @left.to_s + ' * ' + @right.to_s + ')'
  end
end
class Lit
  def initialize(v) @left = v end
  def exec() return @left end
  def to_s() return @left.to_s end
end
class Var
  def initialize(v) @left = v end
  def exec() return $vars[@left] end
  def to_s() return @left.to_s end
end
def test
  $vars['x'] = 5
  e = Add.new(Var.new('x'), Lit.new(1))
  puts(e); puts(e.exec)
  e = Add.new(Lit.new(3),
              Mul.new(Var.new('x'), Lit.new(2)))
  puts(e); puts(e.exec)
  e = Mul.new(Add.new(Var.new('x'), Lit.new(3)),
              Lit.new(2))
  puts(e); puts(e.exec)
end

  irb(main):004:0> test
  (x + 1)
  6
  (3 + (x * 2))
  13
  ((x + 3) * 2)
  16
```

```
  => nil
```

□ 類似したノードクラスを継承により短く定義できることを示す

□ ループ文なども同様に定義できることを示す




```
class Assign < Node
  def initialize(l, r) super; @op = '=' end
  def exec()
    v = @right.exec; $vars[@left.getleft] = v; return v
  end
end
class Seq < Node
  def initialize(l, r) super; @op = ';' end
  def exec() @left.exec; return @right.exec end
end
class Loop < Node
  def initialize(l, r) super; @op = 'L' end
  def exec()
    v=0; @left.exec.times do v=@right.exec end; return v
  end
end

def test1
 e =
  Seq.new(
   Assign.new(Var.new('n'), Lit.new(5)),
   Seq.new(
    Assign.new(Var.new('x'), Lit.new(1)),
    Seq.new(
     Loop.new(
      Var.new('n'),
      Seq.new(
       Assign.new(Var.new('x'),
        Mul.new(Var.new('x'), Var.new('n'))),
       Assign.new(Var.new('n'),
        Sub.new(Var.new('n'), Lit.new(1))))),
     Var.new('x'))))
 puts(e)
 return e.exec
end
```

```
  irb(main):006:0> test1
  (((n$)=(5#));(((x$)=(1#));(((n$)L(((x$)=
  ((x$)*(n$)));((n$)=((n$)-(1#)))));(x$))))
  => 120
```

□ 演習で動かしてみてもらうのはここまで

□ この先の話題は説明だけ

- 字句解析器の考え方とコード例
- BNF、構文解析木
- 再帰下降解析器の考え方とコード例

□ 内容をこれでも減らしているが、まだ「多い」「難しい」との意見多数…

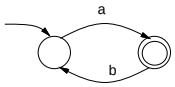
- 今年どうするかは考え中…

3.11 # 10: スタックとキュー/状態空間の探索

□ 抽象データ型としてのスタックとその実現 (配列、リスト)。抽象データ型としてのキューとその実現 (配列、リスト)。いずれも実例は配列版のみ示し、連結リスト版を実装する演習課題

- 式木を深さ優先 (スタック)、幅優先 (キュー) でたどってみせる
- JR 路線図をたどる課題、エディタバッファを 2 つのスタックで実装する課題

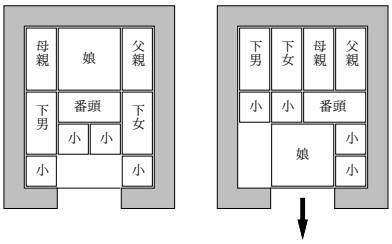
□ 有限オートマトンの復習/オートマトンを Ruby で実装




```
$atm = [{'a' => 1},
        {'b' => 0, :final => true}]
def accept(s)
  cur = 0
  s.length.times do |i|
    k = $atm[cur][s[i..i]];
    if k == nil then return false else cur = k end
  end
  return $atm[cur][:final] == true
end
```

- Hash のリテラルを使うことで「まんま」書ける感じ→ Ruby の利点
- いくつかオートマトンを示して受理するものは何か考えさせる。また Ruby で動かして確認させる

□ パズルやゲームが状態空間探索として定式化できることを説明し、例として箱入り娘を解くコードを提示 (64 行)



- 状態空間の探索については納得したという意見が多かった
- オートマトンは前期にもやっているが、改めて分かったという意見

3.12 # 11: 動的計画法/パターン認識

□ 動的計画法→パターン認識の前フリとして標準スライドで指定

- 動的計画法の考え方を「フィボナッチ数列」「部屋割り問題」を題材として説明
- 多くの学生にとって (ほぼ全員)「考え方は分かっても自分で作れるようになる気がしない」。課題も一番易しいものの提出ばかりだった

□ 文字列のアラインメント→できればこのプログラムを自力で作れるようになって欲しいところだが、授業 1 回では難しいかも

□ パターン認識、forward algorithm、Viterbi アルゴリズム→一応説明してコードも掲載しているが、これも 1 回では時間がない感じ

- しかし「情報科学」の内容の最終回ということで意見をもらうと、色々なものがあるのが分かったという好意的な反応が多かった

4 まとめ

□ Ruby の採用について→手順を書くという点では Java などと変わらない (当然)

- Ruby だからいい点…短く書ける、irb で配列リテラル等を打ち込める (文字からデータ構造を作る部分のコードが要らない)
- Ruby だと困る点…グラフィクスがない、勝手に多倍長など仕掛けが埋もれている
- ブロックとイテレータなど、書きやすいのは確かだが、書いたものが何を意味しているのか説明しようとするとちょっと大変
- 強い型検査は個人的には体験してほしいところ…

□ 内容はまだハードなのでもう少し易しくした方がいいかと思っている (が、骨がある方が面白いと思ってもらえる面も…)

- 学生さんの感想 (下記) を見ていると、やはりプログラミングを学んでいるという意識が強い。それはプログラミングが魅力的だからと思うことにして… :-)
- 各回の題材についてはそれなりに興味を持ってもらえていると思う (いかにも情報科学なので学生もそう思ってくれている?)

□ 冒頭の課題:「世の中一般の人を『ある程度分かっている』人にする」についてはどうか?

- この講義を取った人は「プログラミングとはこういうもの」ということはかなり分かった状態になっていると思う
- この講義を取った人は「情報科学とはこういうもの」ということや、その主要概念 (アルゴリズム、計算量、データ構造、抽象化、…) をかなり分かった状態になっていると思う
- ソフトウェア工学や開発プロセスのようなさらに上位のものは…→ あれば望ましいが、1 科目に詰め込むのは無理がありそう

□ これを紹介するとすぐ「東大だから」と言われてしまいますが…

- どの学校でも、そこに適したレベルで「情報科学の原理」は取り上げることができると考えるし、それが大学としてのレベルであるべきだとも考える

5 学生アンケートから

□ すべてのレポートで 3 問ずつアンケートに回答してもらっている

- だいたいは、2 個の内容についての質問+感想・要望
- 2007 年度に「科目全体としての」アンケートを取った→下記

□ Q. この講義は「情報科学」の学習を目的とし、その確認手段として「Ruby によるプログラミング」を演習して頂いていますが、本クラスにおけるその両者の位置づけやバランス、難易度についてどう思いますか。

- 一部非常に難しい演習もありましたが、それを除けばバランスは取れていると思います。
- 情報科学としてはバランスはちょうど良かったと思います。難易度は途中で急に上がったような…
- こういうのはよくわかりませんが、個人的には小難しい話を延々とするよりはこの講義のように実習を多めに取り入れるほうがいいと思います。ただ難易度はもうちょっと低くてもいい気がします。
- 講義の難易度が途中から著しくあがった気がします …
- 情報科学についても、Ruby によるプログラミングも、そこそこ理解できた気がします。比較的易しい内容や、プログラミングに関してはほとんど理解できなかった内容もありました。Ruby を通して情報科学の学習を行うというのは出来たと思います。
- 情報科学の考え方が理解できても、いざそれを Ruby で表現するとなると大きな困難が伴い、大変苦戦し

ました。個人的な意見としては情報科学の学習が主目的であれば、プログラミングに関してはもうちょっと易しくしてもよかったような気がします。

- バランスはちょうど良いと思いました。難易度は、後半が結構高かったです。
- この講義では演習の方に重点を置いていたような気がします。原理的なものについては理解できても、実際プログラムを作ろうとすると難しいことが多かったです。
- 内容は理解できても、ルビー化できないときが多々ありました。難しいです。
- 少し課題が大変なときもありましたが、全体としてちょうどいい難度だったと思います。 はじめてのプログラミングでしたが、楽しくなんとかついていくことができました。
- この授業はプログラムの実習が中心だったので、もう少し，解説に力を入れてほしかったと思います。
- この講義は、位置付け、バランス、難易度のいずれも程良い感じの良い講義だと思います。効率良く情報科学の内容が身についたように思います。
- Ruby は簡素な言語だなあと思いました。内容は時間の指数関数的に難しくなっていった気がします。
- Ruby プログラミングの難易度が高すぎる気がします。初学者にはきついです。
- どちらかというと、Ruby プログラミングがメインだったような感じがしないでもないです。ただ、プログラミングができる=情報科学が学べている、というきもするし、なんともいえないです。
- 後半はプログラミング未経験者には無理でしょう。継承や演算子オーバーロード (実質) がいきなり出てくるのは……(苦笑
- 情報科学を学んでいるというより、どっちかというとほとんど「Ruby でのプログラミング」を学んでいるという感じでした。ただ、そっちのほうがよかったので、たくさん学べてうれしかったですが。難易度はクラスぐらいまではよかったのですが、動的データ構造あたりからはかなりきつく、おもちゃ言語では力尽きました。
- 情報科学を学ぶために Ruby を利用するのは最初疑問でしたが、まぁありかなって思います。
- ちょうど良かったです。時々与えられる自由な課題が大変でしたが、やりがいがありました。

□ Q. 講義開始時の自分を振り返り、今回までの講義で自分がどんなことを学んだか簡単にまとめて見てください (全体的な感想も含めてどうぞ)。

- 自分はプログラミングの経験はありましたが、Ruby を新たに習得することができました・実用的なプログラミングができるぐらいのアルゴリズムは学べたと思います。・でも結局完全には理解できていない理論はあります。そして、あやふやなアルゴリズムもあります。試験までに復習しないと…
- Ruby が書けるようになったのがよかったですね。課題も楽しかったです。
- Ruby でちょっとしたプログラムをつくれるようになってくらいであまり進歩を感じませんが、プログラミングに取り組む中で「こういうモノがあって、こういう使い方が出来る」というようなことをたくさん見た気がします。
- エディタバッファとかは今後使いそうです。後半の授業は難しかったですが、得るものも多かったです。
- とりあえず、プログラミング自体初めて扱うものだったので、新鮮でした。なかなか複雑なもので、自分で新しいプログラムを作るのはかなり難しそうですが、この講義で基礎を学べたと思います。
- 講義開始時に比べて、パソコンがどのようなロジックで動いているのかがよく理解できるようになった。またごく簡単なものに関してだけであるが、プログラムを書いたり、書いてあるものを理解できるようになった。あと、記号とか打つのが速くなりました。
- 講義開始時は再帰的なものがとても苦手だったのですが、現在ではある程度扱えるようになりました。
- プログラムの計算量について結構理解できまるようになりました。同じことをやるのでも、プログラムの種類によってかかる時間がかなり違っているということには驚かざるを得ません。
- 数学に似てても全く異なる分野でした。使いこなせるのは html どまり…
- まずプログラミングという、色々な手法を組み合わせて、一つの動作をおこすものが作れるんだとわかりました。そういうものを自分で作ってうごかすのは、楽しかったです。
- この情報科学の授業で学んだ事で最も興味深かったのは誤差と計算量でした。 浮動小数点、桁落ち、情報落ちの仕組みを理解する事で，これまで原因が分からなかった乗除計算の時に微小な誤差が生じる仕組みが理解できました。また、同じ目的のプログラムでも計算方法が違う事で計算時間に非常に大きな差が出る事は実際にプログラムを動かす事で実感する事ができたと思います。
- ソートの辺りからは (OO はなんとなく分かっていましたが) 知らないことばかりで、いろいろ新しく勉強させてもらいました。
- アルゴリズムの組み立て方など
- データ型やクラスなどの概念は分かったと思います

が、プログラミング言語としてのRubyはまだまだ分からないことだらけです。あと、/の逆向きの記号がまだ打てません。なので、上のプログラムも/(の逆)nがありません。

- プログラミングというものを習得し、コンピュータの動くしくみの一部を垣間見た。というのが、「学んだこと」に対して真っ先に思い浮かんだものです。
- アルゴリズム関連弱かったので、経験できてよかったです。
- 講義開始時はプログラムなんて全く分からず、もちろんソースなんか見てもただの意味不明な羅列にしかすぎなかったのですが、現在ではソースを見て何をしているのかなんとなくは理解できるようになりました。数学的な話題 (連立方程式とか微分方程式とか) はほとんど忘れてしまっていますが、基本的な構文、クラスの使い方、配列、ハッシュ等は大方使えるようになりました。興味深い内容も多くあり、楽しくまなぶことができました。(字句解析器はきつかったですが)
- 内容が量的に結構大目で大変でしたが、プログラムがどんな風に組み立てられているのがが少しわかって面白かったです。
- ものごとの表現方法はいろいろあるのだと思いました。

6 参考文献

□ 情報処理学会，情報専門学科におけるカリキュラム標準J07，http://www.ipsj.or.jp/12kyoiku/，2009．

□ 情報処理学会，日本の情報教育・情報処理教育に関する提言2005，http://www.ipsj.or.jp/12kyoiku/，2005．

□ 情報処理学会，2005年後半から2006年初頭にかけての事件と情報教育の関連に関するコメント，http://www.ipsj.or.jp/12kyoiku/，2006．

□ 兼宗，久野，ドリトルで学ぶプログラミング，イーテキスト研究所，2008．

□ 久野，Rubyによる情報科学入門，近代科学社，2008．

□ Tim Bellほか著，兼宗監訳，コンピュータを使わない情報教育アンプラグドコンピュータサイエンス，イーテキスト研究所，2007．