ARTICLE

基般

プログラミング入門科目の指針と実践例(前編)

久野　靖
電気通信大学

大学におけるプログラミング入門科目の課題

情報社会の今日において，プログラミング教育は「コンピュータの基本原理にかかわってる」という理由からか，かなり多くの大学で教えられている．特に理工系の教育課程においては，「プログラムを組んで計算を行えることが必要な素養である」と課程設計者が考えるのか，初年次においてプログラミングを必修・準必修としている場合も多い．

しかし現実には，そのような科目はいろいろな問題を持つことがある．その最大のものはずばり，「プログラミング科目で単位を取得したのにプログラミングができない」ではないだろうか．筆者も，そのような現実をいくつか見聞したことがある．

それはなぜか，それを避けるためにはどうしたらよいか，というのが筆者にとっては長期にわたる問いだった．この問いに対する解を思いついたとして，それを実際にやってみるというのは簡単ではない……はずなのだが，好運にも「理工系大学の初年次情報教育の責任者」という職に就くこととなり，実際に「やってみる」ことができている(しかも大規模に)．本稿はその報告である（分量の都合から前編と後編に分けている)．

電気通信大学の初年次情報教育

電気通信大学は東京都調布市にある理工系の単科大学であり，初年次情報教育として前期に「コンピュータリテラシー」，後期に「基礎プログラミングおよび演習」(Fundamental Programming という英語名であり，以下 FP と記す）を，いずれも 2 単位の必修科目として開講している．

両科目とも全学必修のため受講人数は 800 名前後であり（過年度生を含み，既習得単位認定者は除く)，13 クラス(うち夜間課程 1）に分かれて実施している．初年次科目にしては珍しく，担当者すべてが本学の専任教職員となっているが，これは本学が情報系の比率の高い大学であることによっている．

このうち前期の科目については，その名称にもかかわらず，一部（15 回中 2 回）プログラミングを体験してもらう内容を盛り込み，好評を得ていることを別稿において報告した[1), 2)]．

しかし本題は後期の科目である．この科目は「簡単なプログラムの作成と読解ができる」ことを目標にしているが，筆者が 2016 年度に着任した時点では期末試験時においても「プログラミングができない」学生がかなり存在していた☆1．そこで 2017 年度から，教材や授業運営を大幅に見直し，「できない」学生をできるだけ減らすべく努力中である．その具体的な設計・実装は，筆者が上記の「問い」に対して「こうしたらよいはず／こうするべき」と考えてきた指針群に基づいている．

そこで以下では，指針を 1 つずつ挙げ，その内容，およびそれをこの科目でどう実現しているかについて説明させていただく(前編の内容)．後編では科目の具体的内容と個別の内容に即した配慮・工夫と結果について紹介する．

☆1 試験もそのことに配慮して，持ち込み可・資料閲覧可で，文法などの知識を問う設問なども交え，なんとか再履修が多すぎないようにしていた．

❑ 指針 1：離陸ファースト

離陸[3]とは「自分の考えたことをプログラムにして動かせる」ことを指す．それは最終目標であり指針ではないのでは，と思われるかもしれないがそうではない．

多くの教科書や授業では，プログラムの書き方の規則や対応する動作などを延々と説明してから「では書いて見ましょう」という段階に到達する．しかし「書いてみる」時点までに学んだ（というより単に提示された）内容が多ければ多いほど，学生はその中で何が大切で何はさほど重要でないか分からず，途方に暮れてしまう．そのような学生ができることは「例題をそのまま写して動かす」くらいであり，到底離陸はできない（図 -1 の下側の経路）．

そこで FP では，できる限り簡単な（しかし意味はある）例題をまず説明し，すぐにそれを動かしてもらい，それを土台に演習を行う．このために使用言語も C 言語から Ruby に変更した（科目の前半 2/3）．最初の例題を図 -2 に示す☆2．

初回はこの例題を一通り説明してから各自にも動かしてもらい，その程度でできる演習をすぐ実施する．演習の課題は「和・差・商・積」「剰余」「逆数」「円錐の体積」「8 乗」などである．たとえば最後のものは「乗算演算のみでその回数を少なく」と注記してあり，図 -3 が想定解となる．つまり，この程度のボキャブラリでも十分考えさせる演習は行える．

☆2 実行は irb（Ruby 用の read-eval-print loop）を用い，コードをファイルに書き load して実行する操作だけ教える．

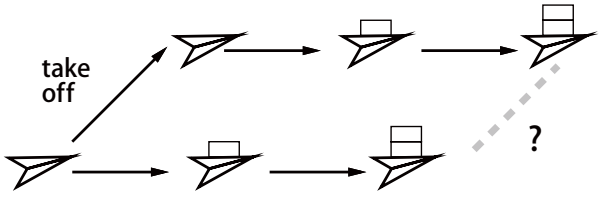

図 -1　離陸の概念

```
def triarea(w, h) #3 角形の面積
  s = (w * h) / 2.0
  return s
end
```

図 -2　最初の例題：3 角形の面積

すなわち「離陸ファースト」では，まず簡単な内容で離陸してもらい，離陸の状態を維持したままで徐々に内容を学び増やしていく（図 -1 の上側の経路）．こうすれば，学んだことはすぐ試せ，演習問題で考える機会が持てるため，身につきやすい．

❑ 指針 2：多様な水準の演習問題

プログラミングに限らず何事も，演習するときにはその難しさ（負荷）が適切でなければ成長（学び）は起こらない（図 -4）．しかし，プログラミングの授業では，全員に同じ演習問題に取り組ませるものが多い．もちろん，プログラミングの腕前（や素養）は人により大きく違っている．そのため，その 1 つの問題は多くの（下手をするとすべての）学生にとってやさしすぎか難しすぎであり，効果的な演習とならない．

そこで FP では毎回，テキストの演習問題の数を多くし（1 回あたり 10 個程度），好きなものを 1 つ以上選んで実施し，結果をレポートとして提出するよう求めている．問題はおおむねやさしいものから難しいものの順で並べてあるので，実力のある学生はさっさと後の問題まで進み（明らかに簡単だと思えばパスしてよい），そうでない学生はやさしい問題に取り組む．これにより，多くの学生が自分に合った難しさの問題で演習をできるようにしている．

手抜きでやさしい問題ばかり選ばれると危惧されるかもしれないが，実際には実力ある学生は簡単な問題は「つまらない」ため，多くの学生が能力一杯く

```
def pow8(x)       #x の 8 乗
    x2 = x*x; x4 = x2*x2; return x4*x4
end
```

図 -3　x の 8 乗のコード例

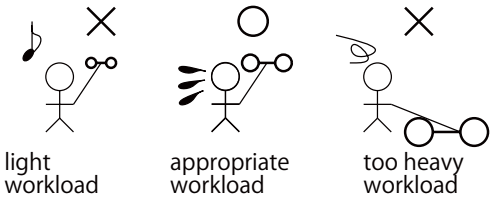

図 -4　適切な負荷

らいの課題にチャレンジしている（体調その他の都合によりやさしい問題を選ぶことは認めている）．

取り組む問題が異なるのは不公平という意見もありそうだが，科目の目的は有効な学びであり公平ではないし，それぞれの実力に応じた練習をすることが公平だともいえる．

❑ 指針３：演習の重視

この科目は週１回 90 分の授業であり，授業時に講義をしていたら演習の時間はほとんどない．筆者が着任した 2016 年時点の授業はほぼそういう状況で，演習ゼロで授業終了となることも多くあった．そこで起こるのは次のことである☆3．

> 学生は時点 X までは「やさしい内容だと馬鹿にして」演習せず，時点 X で突然「内容が難しすぎて落ちこぼれた」と気づく．

これを防ぐにはとにかく「全員が毎回演習する」科目設計が必要である．そこで，テキストと講義ビデオを用意してあらかじめ公開し，予習を義務づけた上で，担当教員には「最低限の説明にとどめる」よう依頼することで演習時間を確保した．

また，各回とも演習問題を「1 つ以上」選んでプログラム・解説・考察を含むレポートとして提出することとした．その点数合計は 50 点あり（試験も 50 点満点），レポートが出ていなければ単位は取れない．演習してプログラムが作れていなければレポートは出せないので，学生は毎回演習に取り組むようになった．

レポートは「説明や考察を重視」するとしたので，プログラムだけ写して出しても 0 点であり，自分で書いた（少なくとも説明してもらって理解した）ものでなければ意味がない．元々学生がプログラムを写すのは「難しすぎて書けない」からであり，やさしい問題（初回の例では 2 つの数値の和など）が選べるようにすればまず写すことはないというのが筆者の経験である（そもそもテキストには毎回「前回演習問題の解説」を掲載しており，写すとすればそれを写している．しかしそれでも説明や考察は自力で書かないとレポートにならない）．

☆3 もちろんこの「時点 X」は学生によってまちまちである．

❑ 指針４：苦手な人の学びを促す

多くのプログラミングの科目では，前述の「同一の問題」のせいもあり，できる学生が労せずしてよい成績を取り，苦手な学生は否定的な感情を持つ．初年次の必修科目としては，苦手な学生が最大限成長できることを目標にするべきではないだろうか．

このため FP では毎回のレポートを ABCD の 4 段階で評価し，D は不受理 C は明らかな不備，A は特に優秀，それ意外は B（通常）とし，すべて B のとき 50 点（満点）とした．苦手な学生も普通にレポートを出していれば満点となるため，前向きに取り組んでもらえる効果があったと考える．

A は 50 点から上積みされる（上限 59 点，単位は 60 点以上）ため，できる学生は必死に A を取ろうとするが，A は 2 ～ 3% にとどめるように担当教員に依頼している．この採点方針では基本的に B をつければよいので，大量のレポートを毎週評価する担当教員の負担軽減にもなっている☆4．

❑ 指針５：プログラムが書けるという目標の明示

世の中の多くのプログラミング科目では（そして 2016 年度の FP も），プログラムが書けなくても単位が取れる．これは試験で，知識問題（構文規則の知識などを問う），穴埋め問題（プログラムが書けなくてもパターンで正解し得ることが多い）を採用するためである．このような「逃げ道」があると，学生が何がなんでもプログラミングできるようになろうとは考えない可能性がある．

FP では試験を CBT（Computer Based Test）で実施するため，問題形式として「短冊問題」を採用している．これはプログラミングの設問に対して，正解プログラムを 1 行ずつばらばらにして誤答と混ぜて選択肢とし，これらの選択肢から拾って画面上で

☆4 しかし多くの教員がまじめにレポートを見てくれて，一定量のコメントを返している．フィードバックがあることは学生にとって大きなモチベーションとなっている．

並べることで正解を構成させるやり方である．

この出題形式は，プログラムが実際に書けない人が正解できる可能性は非常に小さい．毎回の授業において，この形式の「確認問題」を 2 問ずつ提示し，試験時にはその類題を出題することとしている．これにより，多くの学生は「プログラムが書けなければ単位は取れない」と（正しく）認識し，プログラミングの技能の向上に努めるようになっている．

❑ 指針 6：プログラミングに唯一の正解はない

今日の日本の教育は「問題には唯一の正解がありそこに早く到達すると勝ち」という強い刷り込みを作っている．これは社会に出てからさまざまな問題を解決する上でマイナスである．そしてこの刷り込みは，プログラミングの科目においても「テキストの例題の丸暗記」「誰かの正解のコピー」のような意味のない行為につながっている．

FP ではこの刷り込みを打破するため「プログラムの書き方は一通りではない」「自分でよいと思う書き方を各自で見つける」ことを繰り返し述べており，またそれを体現する演習問題を題材としている．

たとえば図 -5 は，「2 数のうち大きいもの」を返すコードである．どちらも動作は同じだが，好みを尋ねると max2a の方が好みという学生が多い．しかし，「2 数」を「3 数」に拡張すると（演習問題になっている），max2a を土台にしたものは if の入れ子になり複雑になるが，max2b を土台にしたものは同じ形の if を並べるだけで簡潔になる．このような例をなるべく多く取り上げることで，書き方の多様性に触れ，それらに対する自分の考え方を持たせるようにしている．

```
def max2a(x, y)
  if x > y then return x else return y end
end
def max2b(x, y)
  max = x
  if y > max then max = y end
  return max
end
```

図 -5　2 数の最大を実現する 2 種類のコード

❑ 指針 7：プログラムは自分の頭で作り出す

プログラミングの科目では，テキストにある例題プログラムがそのまま再現できれば単位がとれるようなものもある．しかしそれでは，実際にプログラミングが必要になったときに「まだない（必要な）プログラムを作り出す」ことができず，役に立たない．

そこで FP では，例題は最小限にとどめ，その例題を元にして「新たな」プログラムを「自分の頭で」考え作り出すことを重視した．各回ともこの方針は共通しているが，その中でも特にこの点を重視した内容が「画像の作成」「動画の作成」である．

これらはそれぞれ「2 次元配列」「プログラム構造の整理」の単元になるが，課題としては自分（たち）の考えた画像や動画を生成するプログラムを作成してもらう．

画像や動画は自分で「このようなもの」と思いつくのが容易であり，かつ個性が出しやすいので，これを課題とすることで自然に「自分の作品を自分で設計し開発する」形となる．実際にこれらの課題は学生が特に熱心に取り組んだという印象である．

本稿では初年次プログラミング教育に関する問題意識と科目の設計指針について説明した．後編では具体的なカリキュラムと内容ごとの工夫，および実施結果について述べる．

参考文献

1）久野　靖：電気通信大学における「コンピュータリテラシー」科目，情報処理，Vol.59, No.10, pp.934-938 (Oct. 2018).
2）久野　靖，江木啓訓，赤澤紀子，竹内純人，笹倉理子，木本真紀子：コンピュータサイエンス入門教育の題材としてのアセンブリ言語プログラミング，情報処理学会誌教育とコンピュータ，Vol.4, No.2, pp.23-36 (June 2018).
3）久野　靖：プログラミング教育／学習の理念・特質・目標，情報処理 , Vol.57, No.4, pp.340-343 (Apr. 2016).

（2018 年 12 月 19 日受付）

久野　靖（正会員） y-kuno@uec.ac.jp

1984 年東京工業大学理工学研究科情報科学専攻博士後期課程単位取得退学．同大学助手，筑波大学講師，助教授，教授を経て現在，電気通信大学情報理工学研究科教授．筑波大学名誉教授．理学博士．プログラミング言語，プログラミング教育，情報教育に関心を持つ．