08

基般

公立はこだて未来大学における初年度プログラミング教育

美馬義亮（公立はこだて未来大学システム情報科学部）

公立はこだて未来大学の紹介

公立はこだて未来大学（以下「はこだて未来大」）は，1 学部からなり，入学定員が 240 名の大学である．教育・研究分野としては，情報工学・科学を中心に，認知科学から数理科学まで幅広く社会の中に存在するテーマを対象とする．

はこだて未来大は，入学時は全学一括で学生募集を行う．1 年間の共通導入教育を行ったあと，2 年次進級時に情報システム，情報デザイン，複雑系，知能システムの各コースに配属され，さらに，2 年次に情報システムコースの一部は高度 ICT コースというコースに配属される（**図 -1**）．

コンピュータとそれを用いる人間を研究の対象とするこの大学では，全員がパーソナルコンピュータを所有することを前提とし，入学直後から，プログラミング教育を始める．2 学期制で教育がなされているが，1 年次にプログラミング技術の修得を要求する科目は，プログラミングの入門である情報表現入門（前期），センサを用いる情報表現基礎 I ならびに演習（後期前半），C 言語を学ぶプログラミング基礎（後期後半）の順に配置され，すべての学生が必修科目として履修を要求される．2 年次になると，Java を用いたオブジェクト指向プログラミングなどに学習内容が発展する．

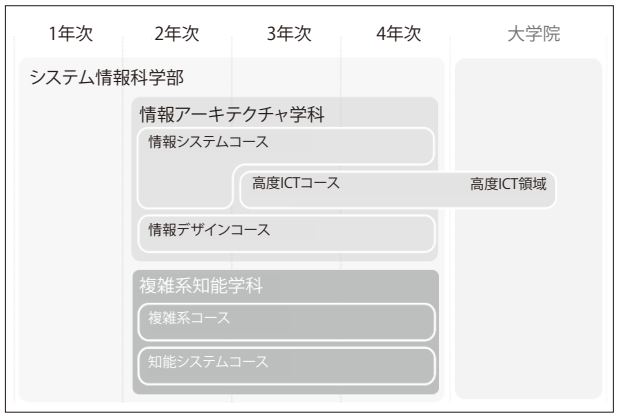

図 -1　はこだて未来大の学科構成（同大 Web ページより引用）

情報系の大学でのプログラミング教育において一般の大学と異なる面があるとすれば，教養の一端として，情報科学の諸概念にふれるというかかわりを超え，職業人としてのアイデンティティを構成する技能の 1 つとして，プログラミング技術へのより深い理解を持たせることである．

本稿では，このようなプログラミング教育の中で，最初に登場する 6 名の教員で取り組む「情報表現入門」という科目の狙い・内容・経験を紹介する．

「情報表現入門」の構成

Processing 言語とその特徴

本授業では，MIT（マサチューセッツ工科大学）で開発された Processing 言語 [1), 2)] を用いる．Processing を用いた主な理由は，プログラミング言語について，何も知らない受講者への障壁が比較的低いことである．特に，メソッドの名とパラメータの組みを列挙するだけで図形記述ができることは，プログラミングへの導入を容易にする．また，`draw()` というメソッドを記述したとき，このメソッドが定期的に呼び出される ActiveMode の存在は，アニメーションを自然に記述することを容易にしている．

さらに，Processing の文法は Java，C などの言語との類似度が高く，後続の授業への移行が容易である．インストールが容易で無料で利用できる点もまた重要な要素である（**図 -2**）．

以下の 3 行でプログラムが成立

```
void draw(){
 rect(mouseX, mouseY, 5, 5);
}
```

60 分の 1 秒ごとに，mouseX，mouseY として得られるマウスポインタの位置に大きさ 5 の四角を描き続ける

図 -2　Processing のプログラム例

3 部構成の課題

本科目は，1 週あたり 90 分授業を 3 コマ連続にとり実施している．この 3 コマを利用し，一斉授業と実習を組み合わせて，TA（ティーチングアシスタント）の協力も得ながら以下の 3 種類の課題を実施する．

A. アクションゲームの拡張：1 つ目の課題は，自分でデザインする「ブロック崩し」の作成である．ブロック崩しを作成するまえに，40 行程度のピンポンゲームを例として与える．このプログラムは，シンプルであるが，基本的な図形表示，変数，条件文，繰り返し，などの概念を含んでおり，一斉授業での説明，問題演習を経て理解する．

さらに，ブロック崩しのプログラムのヒントとするために，最初に示したピンポンゲームを自然に拡張したプログラム（**図 -3**）を提供する．このプログラムを解読することにより，配列，関数を含む Processing のクラス定義以外の主要な概念を 7 週目までに理解することになる．

B. グラフィック表示：2 つ目の課題は，「グラフ表示プログラム」の作成である．ファイルからテキストデータを読み出し，棒グラフやレーダチャートを作成するプログラムを作成したり，自分自身の学習時間の表示などを要求する．

C. アプリケーションの作成：3 つ目の課題では，役に立つプログラムの設計と実現を目指す．最近は，各自の学習時間の自覚を促すことも目的として自分専用スケジューラアプリケーションを作成する課題を出題している．

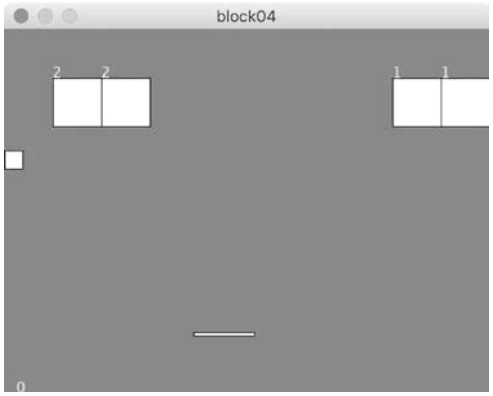

図 -3 「ブロック崩し」の画面

カテゴリ 1：	描画，変数，演算，データ型
カテゴリ 2：	条件式，論理演算
カテゴリ 3：	繰り返し
カテゴリ 4：	配列・関数の定義
カテゴリ 5：	関数の応用

表 -1　チェックテストの対象となる単元

最終発表の実施

課題については，設計計画，完成時などにクラス内で発表会を実施し，受講者同士が相互の批評などを行う．最終回には，各クラスの優秀者の発表を学年全員で共有する機会を設けるなど成績上位者のモチベーションを上げることにも配慮している．

基本概念の定着確認

課題に対応したプログラム作成を行うことに加えて，変数の使用，演算規則，条件式，繰り返し，配列，関数呼び出しなどの基本概念の理解を要求する．概念の獲得を確認する簡単な問題を 5 つのカテゴリに分け，1 回 3 ～ 4 題からなるチェックテスト（**表 -1**）を出題する．4 つのカテゴリに 2 度完答し，最後のカテゴリでは 1 度完答することを単位取得の必要条件としている．

チェックテスト受験機会は，1 つのカテゴリにつき 8 ～ 9 回程度与えられる．一般の問題については，2 度の完答に至るまで平均 2 回程度の不完全回答提出がなされ，配列・関数呼び出しのカテゴリについては，2 題完答するまでに平均約 3 度の不完全回答が発生している．理解が困難だと感じる受講者には，

授業時間外に，大学が認めた上級生がチューターとして学習相談を受け入れる仕組みがある．チェックテストの通過ができない者の数は年度ごとにばらつきがあり，これまでは，0名の年度もあれば，チェックテストの未通過により数パーセントの学生が落第してしまった年度もある．

以下に示すように，問題の難易度については，たとえ正解を得られなかった受講者でも，平易であるという印象を持つ程度の基礎的な問題としている．

関数の定義のチェックテスト例

問：整数M，Nが存在するときに，MとNが2つのint型の引数として与えられたとき，その二数の差の絶対値を求め，返り値として返すabs_diffという名の関数を定義しなさい．

平易な出題であっても，問題の解を与えるために，問題文とその解の関係をパターン（表面的な，当てずっぽうに近い単純な連想）として捉える（深い理解を伴わない）学習習慣を持つ受講生にとっては，複数問題に対し「完答」をすることは難しい．受講者がすでに理解したと信じていても，本質を理解していない場合には，何度も間違えることが多く見受けられ，完答を求めるチェックテストは理解度の指標として有効性が高い．

教育における留意事項

プログラミングを教えてみると，簡単と思える概念でも，簡単には理解してもらえないことが多いことに気付く．教師が教えたことが，受講者に理解されていないこと自体は，教育の現場ではよく見受けられるだろうが，この科目には，論理を積み上げ，問題解決に臨むという要素が強く，暗記を中心として知識構築を要求する科目と異なる特徴がいくつか存在する．また，「Fizz Buzz問題」[☆1]といわれる一見簡単に思えるプログラムが記述できない情報系大学の卒業生が，想像以上に多いという報告がなされている．

「できること」を「分かること」と誤解？

プログラミングの課題を出題すると，多くの受講者が工夫を凝らし，プログラミングに関しては不十分な理解でありながら，要求を満たしつつ動作するプログラムを作成できるものである．最近のプログラム開発環境は，インデンテーションも半自動的に行うので，一見美しく書かれたコードが提出され，動作も安定しているということであれば，教えている側としては，理解が進んだと考えたいところであるし，実際，そう考えがちである．

しかし，よく見てみると変数名が不統一であるとか，無駄なコードが多いなどの不自然さを持つ場合もあり，作者に聞いてみると，各所から集めたコードを寄せ集めて縫合したような場合も少なくはない．

多くの学習は真似ることから始まるため，どこかに既存コードの流用部分があってもよい．むしろ，プログラミングの学習においてはサンプルコードをよく理解し，それらを利用し展開することが重要である．逆に，学習理解が不十分な場合は，受講者は明解な説明ができず，そのプログラムを拡張することも難しい．

しかし，プログラムが動作をした場合には，十分に理解できぬまま作成した場合でも，理解度の確認は難しく，残念ながら教員や出題者の多くは，その作者がプログラムを「作成できた」と認めることが多い．結果的に，学習者自身の判断が甘い場合には，残念ながら「できた（一見動作するように見える）」ことを「分かった（安定して何度も作れる）」ことと誤解していることが多いと考える．

☆1 Fizz Buzz問題とは複数の人が集まり，順番に数字を数えるが，3の倍数のときには「Fizz」，5の倍数のときは「Buzz」，15の倍数のときは「Fizz Buzz」と唱えるというゲームを模したプログラムを作成する問題である．標準入出力，繰り返し，条件文を単純に組み合わせれば解決するはずだが，完答率がそれほど高くないことが不思議だとされる．

プログラミング理解の関門は「抽象概念」

学会の研究会などで，プログラミング教育の場で工夫を行っている人たちに，学習者がつまずく単元を聞いてみると，多くが繰り返しの単元が最初の関門であるという．筆者の経験からも，やはり繰り返しの記述でのつまずきが多い．

数年前，C 言語の教育後，1 学年 250 人程度の受講者からの回答によると，「十分理解しており人にも（その概念が）説明できる」と回答した履修者の割合は，if 文 50%，繰り返し 26%，関数 23% であった．推測すると，繰り返しや関数呼び出しなどは，それらの動作記述を単純に並べて記述すれば済むことが多く，（特に，保守性などの概念も知らない初心者には）あえて抽象化を含む概念を理解することのメリットが感じにくいことにも原因があると考える．

困難さの正体を推測する

さらに考察を進めると，プログラミングの繰り返しの記述では，まったく同じ字面のプログラムが繰り返し実行されている．しかし，実行時には一般に少しずつ挙動の異なる動作を要求される．たとえば，（1 回目は 1 を表示し，2 回目は 2 と表示するなど）何回目の実行なのかの回数を表示する場合は，変数を用いて，その変数をインクリメントすることで実現が可能になる．

この場合には，同じ字面のプログラムとして記述しながらも異なる動作を行う文が用いられることになり，その振舞いを思い通りにするため，初学者の多くは試行錯誤により強引にねじ伏せようとするように見える．慣れたプログラマには平易な作業に感じられても，抽象化を伴う思考方法に慣れない人たちには，理解の難しい問題となり得る．

受講者の「分からなさ」を理解すること

「教える側」にもある種の問題が存在する場合があると考える．それは，教員の多くが，「プログラミングを苦労することなく理解できた人たち」であることである．ある分野に対する理解度が高く研究者になるような人は，その分野の基礎的な概念をつかみ取るのに苦労したと感じる人は少ないと考えられる．

たとえば，筆者の場合であれば，for 文の理解に苦しんでいる人を見ても，何がどう分からないのか見当がつかないということがあった．「自分が問題を理解すること」と「問題を理解できない他者を理解すること」は異質のことであろうが，教える側は，まずその異質さに気付かなければならない．このようなことは，一朝一夕に解決できるような問題ではないが，心の隅にはいつも忘れずにいることは必要だろうと思う．

本科目設計のポイント

はこだて未来大における「情報表現入門」は，情報工学分野の専門家を目指す学生たちが，意欲を持ってプログラミングを学べる学習環境を提供すべく，毎年，担当教員が集まり状況を確認しつつ改善を加えてきたものである．

ここでは，（1）大学内のプログラミング教育の接続性の重視，（2）プログラミング学習の動機付けとなる題材や活動の重視，（3）専門教育としてのプログラミングにかかわる基本概念の定着，などを目標として授業設計をしてきた方針を設計者の 1 人の個人的な意見として紹介した．

参考文献
1）Reas, C. and Fry, B.，船田　功 訳：Processing を始めよう，オライリー・ジャパン (2011).
2）美馬義亮：情報表現入門，未来大出版会 (2014).
（2015 年 12 月 29 日受付）

美馬義亮（正会員）■ mima@fun.ac.jp

日本アイ・ビー・エム東京基礎研究所を経て，2000 年より，公立はこだて未来大学に所属．2012 年千葉大学工学研究科デザイン科学専攻博士課程修了，博士（工学）．芸術情報，教育工学に興味を持つ．