ARTICLE

初学者向けプログラミング自己学習教材におけるデザイン原則

久野　靖
電気通信大学

高校プログラミング教育の現状

我が国の大学入学試験では，2025 年 1 月の大学入学共通テストからプログラミングを必須内容として含んだ科目「情報 I」が出題され，国立大学受験者は原則としてこれを受験するようになった[1]．また，国公立の 2 次試験や私立大学の試験でも教科「情報」で受験できるところが増えて行く方向である．

2023 年度から 2024 年度にかけては，このような状況を受けて，「情報」を出題予定の多くの大学が体験会やサンプル問題の公開を行い，受験産業による「情報 I」の模擬試験も活発に行われている．

筆者が所属する本会情報入試委員会では，上記の状況を関心を持って注視してきたが，2024 年夏頃になって，次のことが課題ではないかという意見が多く見られるようになってきた．

- 他分野の問題の平均点と比較して，プログラミング分野の問題の平均点がはっきりと低い☆1．
- 他分野にくらべて，プログラミング分野の点数のばらつきが大きく，「まったくできない人」もいる．
- 長めの問題では，最初に（得点して当然と考えられる）基本的な問いが置かれることが多いが，プログラミング分野ではその最初の問いから不正解の人がいる．

これらの原因は，きちんと調査をしなければ決定的なことは言えない．しかし，我々のこれまでの「情報」科目にかかわる経験から，現状では，次のことが大きな要因になっていると予想された．

(1) 情報科を教えている教員の中に，プログラミングを苦手としたり，積極的に教えたいと思わない人が，一定数いる．

(2)「情報 I」の教科書のプログラミングの説明は必要最小限であり，割り当てられている時間も多くない．

(3) これらの結果，多くの生徒が「プログラムが書ける」以前の状態に置かれ，そこから抜け出す手段もない状態にある．

(1) については，情報科の科目が「情報 A/B/C」だったころ，プログラミングを内容として含む「情報 B」の開講比率（教科書販売数から分かる）が 10% ほど，「情報の科学／社会と情報」だった 2021 年度までだと同じく「情報の科学」が 20% ほどだったことが傍証といえる（科目の選択は担当教員だけで決めるわけではないが）．また，2023 年 2 ～ 4 月に実施された「情報 I」高校教員意識調査[2]（550 名対象 Web アンケート）でも，「コンピュータとプログラミング」分野を「嫌いまたは苦手」と解答した教員は 20% 弱いた．積極的に教えたくない先生の場合，授業は「教科書どおり，通り一辺」となることが考えられる．

(2) については，「情報 I」の内容が多いことが根底にあり，200 ページ余りの教科書のうちでプログラミングの説明には 10 ～ 15 ページ程度が割かれているにすぎない．プログラミング入門書なら 200 ページ程度以上あるのを思えば，少ない分量である．意欲ある先生であれば説明を工夫したり副読本や自作教材を使い，実習に時間をかけるなどで補うかもしれないが，そうでない先生に教わる生徒の場合はか

☆1「情報 I」は 2018 年告示学習指導要領では，情報社会の問題解決，コミュニケーションと情報デザイン，コンピュータとプログラミング，情報通信ネットワークとデータの活用，の 4 分野から成る．

なり困難が予想される．

これらの結果が(3) につながると，我々は考えた．

高校生初心者用プログラミング教材の試み

前章で述べた課題は一朝一夕に解決できるものではないし，教員の助けなしに 1 人で学ぶ生徒が必要とする支援についての調査・研究も十分ではないが，とりあえずそのような生徒にいくらかでも助けになることを目指し,「プログラミングの基本トレーニング―短冊型問題による『書ける力』の獲得―」と銘打った自習教材 (以下「本教材」と記す) を作成した．この教材は，情報入試委員会のサイトや Print On Demand 出版／電子ブックで公開している[☆2]．

本教材は「1: 変数と代入」「2A: 基本的な枝別れ」「2B: else や elif を含む枝分かれ」「3A: for 文による繰り返し」「3B: while 文による繰り返し」「4: 制御構造の様々な組み合わせ」「5: 配列の参照」の 7 レッスンに各 3 問（計 21 問）の練習問題と，それに先立つ概念等の説明が中心となっている．

概念説明では，特に強調したい 13 項目を「基本原則」として目立たせた．最初の 3 つを示す．

1：順次実行　プログラムは原則として，書かれた順に実行される．

2：変数と代入　「変数 = ...」は，「=」の右辺で指定された値を，左辺の変数（変化し得る値を表す名前）に書き込む(代入する)．

3：複数回代入　同じ変数に 2 回以上代入した場合，最も最近に代入した値が有効となる．

これらからも分かるように，制御構造に入る前の「1: 変数と代入」にかなり重きを置き，丁寧に説明している．またこのレッスンでは短冊型問題(後述)とその解答方法についても説明している．

これら具体的な設計に加えて，以下のようなデザイン原則(特徴)を採用している．

原則 1：プログラムは書きもの

プログラミングを学ぶにあたっては，「書けるようになる」ことを目標としたい．これは「コンピュータで何かをするためには，それをするプログラムを書かなければならなかった」太古の時代には当然だったが，いまはそうではない．たくさんプログラムが読めればいいという人もいそうだし，穴埋め問題が解ければいいという人は非常に多くいる．しかし以下の理由から，「書ける」ことを目指して練習するのが，いまでも一番効率の良いプログラミングの学び方だと考える．

- コンピュータサイエンスは「さまざまなプログラムを作る」ことを中心に発展してきたので,「書く」姿勢でアプローチすると理解しやすいことが多い．
- 書くことを練習していくと，書いたプログラムが成果物として残るので，成果が振り返りやすく，動かして見られるので思い出しやすく，モチベーションとなりやすい．
- 自分で考えて書いたプログラムは，自分の知識の範囲内で作られ，自分の思考様式にとって自然な構成となるため，読むときも読みやすい．

最後の点は，逆に言えば自分の視点だけに固まってしまう恐れもあり，それを克服するために，意識して新しいものを取り入れて行く必要がある．

原則 2：離陸ファースト

離陸ファースト[3), 4)] は，筆者がここ 10 年ほど，プログラミング学習・教育で大事であると考えている原則である．離陸は次のように定義される．

離陸：自分の手でコードを書いて動かし，結果を見て手直しできるようになる．

それはいいことだが，一番最後に達成されることでは，と考える人もいるかと思う．

☆2 Amazon POD/Kindle, https://www.amazon.co.jp/dp/4802087667/, https://www.amazon.co.jp/dp/B0DMMPYWBF/
情報処理学会情報入試委員会 , https://sites.google.com/a.ipsj.or.jp/ipsjjn/home

そうではなく，これができるだけ早く達成されるべきだ，というのが「離陸ファースト」の考え方である．それを可能にするためには，各段階で学ぶことが十分簡潔であり，手間をかけずにいろいろ試してみられる必要がある．

世の中のプログラミング教材は，そうなっていないことが多い．たとえば，プログラミング言語の入門書には，使うものもあまり使わないものもとりまぜて，その言語のあらゆる機能を（段階を踏んではいるが）先にすべて説明しているものが多い．（言語でなく）プログラミングの入門書でも，著者がまず必要と思うことを１章分説明してしまい，それから「練習問題」がある，という構成は普通に見られる．

しかし，このようなテキストだと，いざ「最初の練習問題にとりかかろう」と思ったとき，山のように説明されたことの何を使えばいいのか分からない．そこで「解答」を見てなるほどと思い，先へ進んだとすれば，最後まで読了しても書けるようになっているとは思われない．

「離陸ファースト」ではこれらと対照的に，最初の例題は数行の簡潔なもので，それの理解と類題を書くのに必須のことだけを学び，それらに基づいて練習問題を解く．それが成功したら（離陸したら），もう少し新しいことを学び，また練習する．このようにして離陸状態を維持しながら学んでいくことで，必要な範囲をマスターする．

原則 3：コードを動かそうと言わない

これまで筆者は，前記の「離陸ファースト」で授業や講習をデザインするとき，必ず「すぐ動かしてみられる」環境をセットで用意してきた．そうすることで，練習問題の正誤がすぐ分かり，また「自分がプログラムを作れている」という大きなモチベーションが得られるからである．

しかし，本教材では「教員の助けなしに１人で学習する（プログラミングが苦手な）生徒」が前提である．そこで「コードを動かしましょう」と言った場合，次の課題が予想される．

- 生徒は「コンピュータが苦手」というメンタルを持ち，そのため「コードを動かす」ことを促す教材は遠ざけてしまう可能性が大きい．
- PC や携帯で使ったことのないソフトを新たに動かし，その助けもないことは，大きなプレッシャーである．
- コードを動かすのに際して，誤りから抜け出せなかったり，操作に長い時間を要したりして，投げ出してしまう可能性がある．

これらのことを考慮した結果，本教材では「離陸ファースト」にもかかわらず，「コードを動かしましょう」とは一切言わないことにした．学習者は，テキストの説明と問題を読み，問題の解答を考え，解説でその正誤や難しい点を知る，という「慣れ親しんだやり方で」学ぶ．

ただし，プログラムを動かすことに前述の利点があることも確かなので，「コードを動かさないでよい」とも書かないことにした．そして，例題や問題の正解プログラム（使用言語は Python）は，目立つ囲みに入れて配置し，そのまま処理系に与えれば動くようにして，いつの時点でもコードを動かすようになってよいようにした[☆3]．

原則 4：短冊型問題の採用

本教材では「離陸ファースト」にもかかわらずコードを動かさないので，それに代わる「やってみる」が必要である．さまざまな教科のドリル型テキストでは通常，練習問題がこれに対応している．しかし，プログラミング入門が題材の場合に，それをどのような形にするか考える必要がある．

☆3 情報入試委員会で，プログラムと入力データは分ける方がよく，そのため入力命令を使ったらどうかという議論があった．しかし，データを用意せずプログラムだけで動く方が望ましいので，プログラム冒頭で入力用変数にデータを代入する方式にした（共通テストもこの方式での出題である）．本教材では，冒頭に並んでいる代入はそのような役割であることを，丁寧に説明している．

まず考えられるのは，共通テストと同様の，穴埋めの選択肢式問題である．これは，プログラミングの練習手段としてはすぐ後で述べるように欠点が多いが，共通テスト等がモチベーションである生徒たちにはよく使われている．利点としては，正誤の判定は明らかで，生徒も慣れていることが挙げられる．

しかし，穴埋め問題がプログラムを書く練習に適するだろうか．プログラムを書ける人は穴埋め問題を正解するだろう．しかし，穴埋め問題を正解する人が必ずプログラムを書けるかは疑問である．実際，同種の問題を出題する情報処理技術者試験の参考書では，プログラムとあまり関係なく「どのように穴を埋めたら正解になりやすいか」の解説が目立つ．生徒がそのような方向に行ったら，本教材としては失敗である．

また，穴埋め問題は「穴」の比率が問題全体に比して小さいので問題量に比べて場所を取り，1 度やったら覚えてしまい，反復練習が難しい．

大学等の授業における練習や試験では，プログラムをキー入力させるか白紙に書かせるのが普通である．確かにこれなら「書く」練習になる．

一方，この方法では正誤の判定が難しい．正解例を掲載しても，それと違う正しいコードは複数有り得るので，生徒に自分で採点させるのは困難である．コンピュータで実行させて正解出力と一致を調べる方法もあるが，「正解／不正解」の 1 ビットしか返さないので，練習用には向かない．

これらに対して筆者らが考案したのが「短冊型」[5), 6)]と呼ぶ問題形式である．短冊型では，正解プログラムを 1 行ずつバラバラにして，正解には使わない行を混ぜて並べ，頭に記号を付して選択肢とする（図 -1 上）．問題は文章として示され（図 -1 下），解答はプログラムとしては図 -2 のようになるが，それを図 -3 のように記号の並びで表したものを解答させる☆4．

短冊型では，穴埋めのようにプログラムの骨格を示してしまわないので，学習者は最初から「ゼロからプログラムを書く」メンタルを持つことができ，白紙にプログラムを書くのに近い部分まで力を見ることができる．問題は数行の文と選択肢で済み，1 回やったら覚えてしまうような弱点も少ない．正解かどうかは記号の列なので容易に判定できる☆5．

☆4 上の正解例のように，1 つの選択肢が複数回現れることもある．またこの問題のように，複数の解答が考えられる場合もあるが，回答者はそのうちの 1 つを解答する．完全に一致しなくても編集距離などで「近い」場合は部分点を与えてもよい．

☆5 誤答の場合は解説を見て学ぶことができるが，その機会や情報の量は穴埋め式より豊富に得られる．

選択肢

```
ア x = 12
イ y = 3
ウ print(z)
エ z = x + y
オ z = x - y
カ z = x
キ z = z + 1
ク z = z + x
ケ x = x * 2
```

問題

変数 x に 12 を格納する。その後、x を 3 倍した結果を書き出す。

図 -1　短冊型問題の選択肢と問題の例

```
x = 12
z = x
z = z + x
z = z + x
print(z)
```

```
x = 12
z = x
x = x * 2
z = z + x
print(z)
```

図 -2　短冊型問題の正解となるプログラム例

正解例

```
アカククウ
アカケクウ
```

図 -3　短冊型問題の正解例

短冊型の問題を解く操作は「写経」[☆6] よりやさしいので，初心者レベルでも対応できる．よりレベルが高い学習者については，書きたいプログラムが長くなって短冊型の選択肢が 30 以上になるようなら「白紙」での練習に移る必要がある（しかしそれは共通テストに出るレベルを超えていると考える）．

本教材では最初から最後まで短冊型の問題を並べることで，コードは動かさないものの，最初から「自分の手でコードを書いて動かし，結果を見て手直しできるようになる」という，離陸ファーストの考え方に近いものが実現できたと考える．

もちろん，まったく白紙にプログラムを書くのと比べれば，使える短冊が限られ，思い浮かべたコードが作れない場合もあるという弱点は存在するが，短冊型問題にはそれを補うだけのメリットがあると考える．

その他の補足

本教材では問題はすべて短冊型で，「変数 data に，10 個の非負整数が並んだ配列を格納する．続いて，data に入った値の最小値を求め，書き出す」のような文章題である．7 つのレッスンそれぞれに 3 問があり，選択肢はレッスンごとに 3 問共通の 1 群にまとめられているので，紙面と問題を見る時間の節約になる．そして，各問ごとに説明，正解プログラム（複数ある場合はその説明も），実行例，正解となる記号の並び（複数正解があればそのすべて）が付されている．

これにより，プログラムの実行なしでも，「離陸ファースト」によって学習者がテキストの範囲のコードが書けるようになることを期待している．

ここまで学習者として生徒を想定してきたが，情報科を教える教員で「離陸」していない人にも，本教材などで学んでプログラムを書ける人になってもらえれば，喜ばしい．やはり分かっている人が教えるのに優る方法はないと思うからである．

本教材はまだ公開したばかりで，評価はこれからという段階である．にもかかわらず，複数の先生方から好評をいただき，要望をいただいて本稿のような背景説明まで書かせていただいた．ここに感謝します．

☆6 例題プログラムをそのまま打ち込んで動かさせること．プログラミングの入門教育では多く見られる．

参考文献

1) 角田博保：大学情報入試の概要，情報処理，Vol.65, No.2, pp.e1-e5 (Feb 2024).
2) みんなのコード：2022 年度プログラミング教育・高校「情報 I」実態調査報告書教員の意識調査（高校・情報担当教員）単純集計結果 , https://speakerdeck.com/codeforeveryone/programmingeducationreport2022-high
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).
4) 久野　靖：プログラミング入門科目の指針と実践例（前編），情報処理，Vol.60, No.3, pp.244-247 (Feb 2019).
5) 角田博保，久野　靖：短冊型問題：プログラミングの技能を評価可能な試験出題形式，情報処理学会夏のプログラミングシンポジウム 2016：教育・学習，pp.73-80 (2016).
6) Nakayama, Y., Kuno, Y. and Kakuda, H.: Split-Paper Testing: A Novel Approach to Evaluate Programming Performance, Journal of Information Processing, Vol.28, pp.733-743 (Nov. 2020).

（2024 年 11 月 19 日受付）

久野　靖（正会員） skuno@acm.org

1984 年東京工業大学理工学研究科情報科学専攻単位取得退学．同大学助手，筑波大学講師，助教授，教授，電気通信大学情報理工学研究科教授を経て，同大学特命教授．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．