01

基般

プログラミング教育／学習の理念・特質・目標

久野 靖（筑波大学大学院ビジネス科学研究科）

教育か学習か？

筆者は最近，原稿等で，本稿のように「プログラミング教育／学習…」という語をよく用いるが，「なぜ教育／学習なのか，教育でいいのでは」と尋ねられることがある．確かに題名が長々しく見た目が悪くなるが，それでも単に「教育」だとどうしても，「受け身で教わることを消化していく」というイメージがあり（**図 -1**）[☆1]，それにとどまりたくない，という気持ちがあって「学習」をつけている．

その気持ちの由来を内省してみると，次のようになる．過去においては，プログラミングの技能は「情報技術者になる」「情報科学の研究手段として用いる」などの「必要」のために「専門家やその卵が」身に付けるものであり，そのためのトレーニングはいかにも（図 -1 のイメージでの）「教育」的だった．

しかし今日では，「コンピュータの本質を知る」「論理的指向力を養う」等のため（詳しくは次章），「一般の人が」プログラミングに接するようになった．そうなると，技能が目的ではなく，その「接する人」がより高みを目指すことに力点が置かれ，それで「学ぶ」「学習」を使いたくなるのである．

☆1 教育とは決して受け身なものだけではない，ということは十分承知していますが….

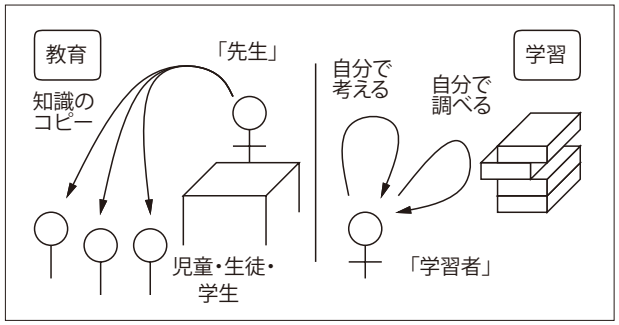

図 -1　教育と学習のイメージ

なぜプログラミングか？

それでは改めて，なぜプログラミングなのだろうか？　ここでは「専門家になるため」「一般の人のため」を合わせて思い付くものを挙げてみる（書籍[1]なども参考にした）．

ソフトウェア開発者が必要（V1）：今日では世の中のあらゆるところでコンピュータが使われており，これら多数のシステムを動かすソフトウェアを誰かが書かなければならない．そのような職に就く人は当然，プログラミングを学ぶ必要がある．

仕事の一環としてプログラミングが必要（V2）：コンピュータの広範な普及の結果，ソフトウェア開発者でなくても，自分の仕事のために必要なプログラムを作成することが増えてきている[☆2]．今後ますます，プログラミングを学んでいなければ就けない職業が増えると見込まれる．

ソフトウェア技術者との連携のため（V3）：ソフトウェアを発注するなどのかたちでソフトウェア技術者と連携する必要のある仕事は多い．そのような場合に，技術者と有効なコミュニケーションをとり，効果的な開発を行うには，プログラミングの経験や理解が有効だと考えられる．

コンピュータの原理理解のため（L1）：コンピュータの本質が何であり，何ができて，なぜ今日のように広まったのかについて真に理解するには，自分でプログラムを書いて動かす体験が必要である．これは，既存のプログラムを使うだけでは「そのプログラムでできること」しかできないのに対し，自分で

☆2 たとえばデータサイエンティストは，既存のソフトを使って分析するというより，自ら大量のデータを収集したり取り扱うためのコードを書くことが求められる職業だといえる．

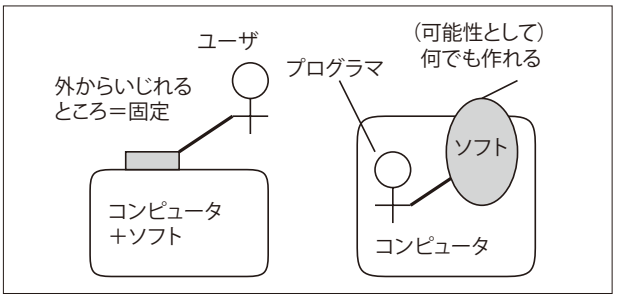

図 -2　ユーザとプログラマの違い

プログラムを書くことはある意味では自分がコンピュータになることであり「コンピュータができることは可能性として何でも作れる」からだと考える（**図 -2**）.

論理的思考を身に付ける（L2）：コンピュータはプログラムの指示通りに動作するので，プログラムを書く際には起こり得るすべての場合を想定し，それぞれの場合について厳密に動作を記述する必要がある．このため，プログラミングを経験することは筋道立てて系統的に考える練習となる.

問題解決と能動学習の題材として（L3）：コンピュータは強力なツールであり，多くの問題に「解答を導く」「人間に代わって作業してくれる」という両面から解決策を提供し得る．このため，プログラミングを通じて自分が持つ問題を解決することを（自分の問題としての〜能動的な）学びの題材としやすい.

答えが 1 つだけでない題材として（L4）：我が国の初等中等教育は（特に大学受験前後で）「想定された唯一の正解を当てる」ことに偏重し過ぎており，これが個人が社会に出て答えのない問題に取り組む際の妨げとなっている．プログラムは同じ動作を実現する複数の記述が可能であり，「解の多様性」を身を持って学べる題材である.

自己実現／表現の手段として（E1）：自分で考えてプログラムを作ることは，自分のアイディアを表現しかたちにすることであり，強力な表現手段である．そしてプログラムを完成させることは，自信や達成感をもたらしてくれる.

もの作りと創造力のため（E2）：ソフトウェアは実際に動いて役立つものなので，プログラミングは（コスト・安全性・場所・機材・身体能力等に大きく制約されることなく）もの作りの活動が行える題材となり，創造力を育む機会を提供してくれる.

思考を外部化した成果物として（E3）：プログラムは「自分が考えたこと」を厳密かつコンパクトに外部化したものであり，それを自分や他人が見て検討することは「個人が考えたことを論理性・客観性を持ってグループで検討する」経験となる.

楽しく熱中できる題材として（X1）：人間は楽しんで熱中するとき最もよく学ぶ．プログラミングはコンピュータに指示して思い通りに動かすという点で，多くの人にとって楽しく熱中できる題材であり，熱中して学ぶ貴重な体験を提供してくれる.

試行錯誤の経験を積む場として（X2）：実社会における課題には，少し考えただけでは解決せず，さまざまな可能性を試行錯誤していく必要があるものも多い．プログラミングは，他人に気兼ねせずいくらでも失敗してみられる場として，貴重な試行錯誤の経験を与えてくれる.

上記はおおまかに「職業的必要性（V1 〜 3）」「教養（L1 〜 4）」「表現力・創造力（E1 〜 3）」「価値ある体験（X1 〜 2）」に分類できる．筆者はいずれも今後の我が国を担う人たちにとって欠かせないことだと考えているが，実際にプログラミング教育／学習を実現する際には，漫然と全部を目指すのではなく，これらの理念・特質のうちどれ（複数でもよい）に力点を置くかを決め，それに合わせて設計・配慮する必要がある（各学校段階の分担案については後述）.

目標は「離陸」

本章ではプログラミング教育／学習を行う際の具体的な目標設定について検討する．筆者はこれまでにさまざまなプログラミング教育／学習の活動を（直接・間接に）見聞してきたが，それらの中で「うまく行っていない」と思えたものはおおむね，次のような特性があったと感じている.

（a）プログラミング言語の文法や機能を逐一，順番

に説明していく．

(b) 1つの例題を選んで丁寧に説明し，試験もその例題から出す．

(c) 例題をその通りに打ち込んで動かすことにだけ注力する．

(a) については，そもそもプログラミング言語のテキストは言語を機能別に説明していくことが普通なので，そのようなテキストを教科書として採用し，その順番に説明していくと自然にそうなってしまう．すると学生はその説明を暗記して試験に対応しようとするので，プログラムを書くということにはあまり意識が行かず，書けるようにならない．

(b) はこの「書けるようにならない」問題を克服しようとして生まれたように思える．しかしこの方法だと，学生はその1つの例題の字面と説明を暗記して試験を乗り切ろうとするので，その例題の「再現」だけはできても，自力でコードを書くことはおぼつかなくなりやすい．

(c) は，プログラミングに対してあまり多くの経験を持たない者が教える場合に，コードの内容やその意味を十分説明できないので，学んだというかたちを作ることに注力する結果，起こりやすい．この場合も学生はコードを十分理解していないので，書けるようにならない．

これらの知見から，筆者はプログラミング教育／学習においてはまず次のことを目標にするべきだと考えるようになった．

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．

なぜ「離陸」が大切かというと，これができれば（手直ししたものをさらに動かして…と続くことで）自力でコードを改良していけるからである（**図-3**）．前章で挙げた項目の大半は「離陸」していて初めて成り立つものであり，この意味でも「離陸」が最重要な目標だといえる．これは前記のような状況を考えると，野心的な目標に思えるかもしれないが，決してそうではない．

たとえば，2～3個の数値を入力し，それを元に計算を行い，結果を表示する，という数行のコードであっても，「どのような計算をするか」については無限の選択肢があり，コードの書き手がその中から「自分の意思で」選択している[☆3]．授業の場面でそのことを指摘し，実際にさまざまな計算を行うように求め，学ぶ側が実際にあれこれ試して意外な結果に遭遇し，コンピュータの杓子定規さに触れることで「離陸」が実現できると考える．

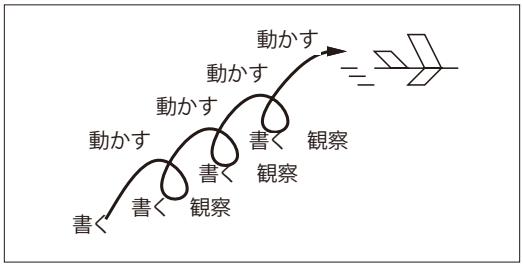

図-3 「離陸」の概念

一度簡単なコードで「離陸」が実現できたら，その状態を維持しつつ，制御構造，関数など，次第に高度な（そして「離陸」している者にとって使いでがある）機能に進んでいくのがよい．その場合，それらの機能を使うことで自然にこなせる課題を示し，自分で試しながら（唯一の正解ではなく）自分なりの解に到達することを求めることで，「離陸」を維持していけるというのが筆者の経験である[☆4]．

このような報告に対し，「学生による違いが大きく，全員にとって適切な課題を出すのは困難」という指摘を受けることがある．これについては筆者は，各段階においてやさしいものから高度なものまで複数の問題を用意し，学生に選択してもらっている．

そうすると常にやさしい課題ばかり選ばれないかという危惧もあると思うが，学生も学ぶことの価値や大切さは認識しているので，「自分が学んで成長するためには適切な水準の問題を選ぶことが大切」であることを呼びかければ十分従ってもらえるというのが筆者の経験である[☆5]．

☆3 多くの授業ではこのような例題を「簡単なもの」としてすぐに通過してしまうが，それは「離陸」の機会を奪ってしまう大変もったいない行為ではないだろうか．

☆4 多くの授業ではたくさんのことを教えた後で「さあ，では離陸してください」とやっているが，それではその間に教わったことが全部重荷になってしまい，浮かび上がれない者が増えるという印象である．

☆5 もちろん「離陸」できていることが必須である．「離陸」できていれば，前章の「楽しむ」「熱中」の要素が働くことで，むしろ積極的に難しい課題に取り組んでもらえることも多い．

学校段階ごとの分担

現在の我が国では，小学校・中学校・高等学校・大学における体系的なプログラミング学習は（情報・工業・商業などの専門高校や大学の情報系・理工系を除けば）ほとんど実現されていない．そのため，どの学校段階であっても，最初の「離陸」からやる必要があり，そのために「楽しさ」に基づく学びが不可欠である．

しかし，そのようなかたちだと各学校段階ごとの違いが見えにくい．そこでここでは，「仮に小学校から大学まで切れ目なくプログラミングを学ぶ機会が提供できるなら」という仮定のもとで，各学校段階ごとに何を理念とすべきかを提案する（**表 -1**）．大学については初年度の共通教育までを考え（専門課程については学部学科ごとの違いが大きく一概に言えないため），情報系，（情報系以外の）理工系，それ以外で分けた．高校については普通科で全員が学ぶ部分と選択科目で分けた[☆6]．職業科・高等専門学校については大学の情報系／理工系／その他のいずれかに準じればよいものと考えている．

小学校：L1 E1 E2 E3 X1 — プログラミング体験を通じて「コンピュータとはどのようなものか」を知ることが重要である．また，図画工作や音楽などと同様，表現手段・創造力を養う手段としても扱う．熱中して取り組み，楽しい体験を持たせることがこの先の学習に備える上で大切．
中学校：L1 L2 L3 L4 E1 E2 E3 X1 — コンピュータについて知ることも重要だが，併せて論理的思考・問題解決・答えが 1 つでない問題への取り組みを学ぶことも重要．表現手段，創造力，熱中しての取り組みも引き続き重視する．
高校 必履修：V2 L2 L3 E1 E2 X2 — 仕事の上で必要ならソフトウェアが作れることと，論理的思考や問題解決のためのプログラミングを，表現や試行錯誤と合わせて取り上げる．
高校 選択：V2 V3 L3 L4 E2 E3 X2 — 必修部分に加えて，専門家と連携，多数の正解，グループでの検討などの側面まで，より踏み込んで学ぶ．
大学 情報系：V1 V2 V3 L2 L3 E1 X1 — 開発者から発注側まで職業的観点を一通り学ぶ．また論理的思考，問題解決，自己実現，熱中できるなどの点も盛り込む．グループで作るなどの部分（E2 E3）は専門課程にゆだねればよいと考える．
大学 理工系：V1 V2 V3 L2 L3 E2 E3 X1 — 情報系でなくても現状ではソフトウェア開発者への進路も考慮する．専門課程は情報技術が中心ではないので，初年度段階でもの作り的視点を経験させたい．
大学 一般：V2 V3 L4 E1 E2 E3 X1 — 情報系・理工系以外では，複数の解を持つ題材として，また自己表現やもの作りの体験手段としてプログラミングを取り入れる（汎用スキル指向）．

表 -1　各学校段階ごとの理念の選択

コードは美しく

そしてもう 1 つ，今後追加したい理念に「コードは美しく」がある[2]．これは，ソフトウェア工学の観点からも必要であるし[3]，価値あるコードで世の中に貢献している人の多くは「美しく」書くことにこだわっていると思うからである．ただ，この理念は天下り的に規則を紹介しても身に付かないと思う．筆者は今は学生に「自分で美しいと思う指針を考えよう」と呼びかけるにとどめているが，よりうまくこの理念を学んでもらう方法は，今後の課題としたい．

☆6 現在検討中の次期学習指導要領では，情報の科学的な理解に重点を置く必履修科目とその履修を前提とした選択科目の 2 科目体制が想定されている．本稿では進学しない場合は必履修科目まで，する場合は選択科目までを学ぶとして考えた．

参考文献

1) 神谷加代 著，竹林　暁監修：子どもにプログラミングを学ばせる 6 つの理由，インプレス (2015).
2) Kernighan, B., Bentley, J. and まつもとゆきひろ他 著，久野禎子，久野　靖 訳：ビューティフルコード，オライリー (2015).
3) McConell, S. : The Code Complete 2nd ed., Microsoft Press (2004).

(2015 年 12 月 28 日受付)

久野　靖（正会員）■ kuno@gssm.otsuka.tsukuba.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．同年同大学理学部情報科学科助手．筑波大学講師，助教授を経て現在，同大学ビジネスサイエンス系教授．理学博士．プログラミング言語，ユーザインタフェース，情報教育に関心を持つ．