一般人にプログラミングの重要さを伝える

原田 康徳[1]

概要：教養としてのプログラミングを一般人に対して教える重要性を述べる．一般人向けにプログラミングによるコンピュータ入門講座を開いたのでその内容を報告する．

キーワード：プログラミング教育

Everyone Should Learn Programming

Yasunori Harada[1]

Abstract: I describe the importance of programming as general education. I report a lecture on introduction to computer for non-professional people.

1. はじめに

プログラミング教育がブームである [1]．政府の成長戦略でも，IT 人材の育成として義務教育からプログラミング教育を推進するとしている [2]．たしかに，今後コンピュータを中心とした仕事がますます増え，プログラムができる人材の重要性は高まることが目に見えている．しかし，そういった時代の流れに対し，「やらなければ乗り遅れる」といった脅して教えるのでよいのだろうか．一番の懸念は，教える側も教えられる側も無理矢理やらされることで，嫌いになってしまう人が出てしまうのではないかということである．

子供にプログラミング教育をする以前の問題として，一般人に対してプログラミングの重要性を伝えすべての子供たちに教えるべき内容であるというコンセンサスを広めて行かなければならない．この努力をしないまま「やらなければ乗り遅れる」で強引に導入しようとすれば，かなりの抵抗にあう可能性もある．

未来は，技術者と資本家によってのみ作られて行くべきではない．IT 人材育成という観点ではなく，IT 化社会で豊かに生きるための教養としてプログラミングを教えるべきである．豊かに生きるとは，すぐに役に立つプログラミングという視点よりも，コンピュータとは何か，人間とコンピュータとはどこが違っているのか，そこからコンピュータの楽しさと可能性を感じられるか，そういったことを考えられる視点を持つということである．

コンピュータとは何かを言葉だけで伝えることは非常に難しい．その理由の一つは，コンピュータは前例がない特異な発明だからである．多くの発明は，ほぼすべて，それまでに存在していた技術からの差分で説明できる．たとえば写真の発明の前には絵画があったし，映画は動く写真と言えば良かった．しかし，コンピュータは，あまりに特殊でなにかの差分で説明することは不可能である．前例となるような発明が存在しないのである．さらにコンピュータは「すごさ」が桁違いである．自動車が馬車より速く遠くまで行けるようになったとしても，せいぜい 100 倍程度である．ところが，コンピュータのすごさは記憶容量も計算速度も億や兆の比である．このように前例がなく，すごさも桁違いであるということから，コンピュータとは何かを言葉だけで説明することが非常に難しいのである．

一方，人類の進歩にとって，文字による知識伝達は欠かせない．文字で記述されたものを読むことで，先人の経験と同じ時間をなぞる必要がなく，先に進める．世の中の大半が文字による知識伝達で上手く行っているので，コンピュータも本を読んだだけで理解できると思う人もいるか

[1] NTT コミュニケーション科学基礎研究所

もしれない．この誤解もコンピュータの理解を下げている．

文字で理解できないということは，体験として伝えなければ伝わらないということである．それが，コンピュータとは何かを理解するにはためにプログラミングをすべき理由なのである．コンピュータとは何かが理解できて初めて，IT 化社会の未来について語れるようになる．

さて，このように一般人に教養として広めるプログラミングにはどのような性質が求められているだろうか．プログラミングのどの性質を強調し，どの性質は隠してもよいのだろうか．

2. 知って欲しいこと

教養として，どの範囲までの理解を求めるのかは，コンピュータの専門家の間でも意見が分かれるところであろう．非常に重要なことであっても，その説明が難しければ諦めなければならないだろうし，説明が簡単であれば広く取り入れた方がよいであろう．

著者が重要だと思うことを述べる．

2.1 小さな変化の積み重ね

コンピュータと人間とが一番大きく違うのは，コンピュータは小さな変化の積み重ねで動いている，ということである．たとえば，「この部屋から東京駅に行く」という命令を考えたとしよう．人間の場合は，駅に行って，電車を乗り換えて，といったプランをさっと立てられる．ところがコンピュータにやらせるには，まず「部屋のドアをあける」というところから始めなければならない．そうしなければドアにぶつかる．本当は人間もドアをあけなければならないのだけれど，そういう当たり前のことは考えなくてもよい．この最もコンピュータの基本である「小さな変化の積み重ね」を知るということは，人間がいかに常識を持って賢く処理しているかということを知ることでもある．

プログラムを書いて動かす，という作業には，大きな問題を小さな変化に分解する必要がある．どれくらいまで小さくすればコンピュータが扱えるようになるのか，基本単位のサイズに対する直感は非常に重要である．

2.2 コンピュータには正解も間違えもない

コンピュータにとっては与えられた命令を単純にこなすだけである．命令に正解や間違えは存在しない．人間が想像した動きとは違うことを間違えと言うかもしれないが，それは人間の思い違いということでもある．

大人が子供に「私の言っていることがどうしてわからないの」と言って怒ることがあるが，コンピュータに対しては，わからないように言っている自分が悪いのである．

また「コンピュータは間違えない」というありがちな認識も誤解である．

便利で，色々と自動的にやってくれるコンピュータ（本当はそういうソフトウェアが動いているだけ）が増えてきて，逆にこういったコンピュータの元々の性質は見えにくくなっている．

2.3 アルゴリズムは？

プログラミング教育の基本としてアルゴリズムが重視されているが，それについても考えてみたい．そもそも，アルゴリズムがなぜ必要かというと，コンピュータは小さな変化の積み重ねで動き，与えられた指示だけを正確に実行し，それ以上のこともそれ以下のことも全くしない，というところから来ている．それらを知った上で，さらに効率を上げるということになって初めてアルゴリズムを考える必要が出てくる．アルゴリズムを教えるよりも前に，上記 2 つを理解することの方が先なのである．

入門だからアルゴリズムの簡単なものを教えればよいというのは間違っている．こんな面倒なものを最初にやらなきゃいけないとするから，コンピュータは専門家に任せておけば良いとなってしまうのである．たとえば，決まった経路でロボットを動かすための命令（たとえば，前，右，前，前，…といったもの）を並べることがプログラミングの入門であると教えているところもあるが，これはパズルを解いていることではあるけれど，それを知ることがコンピュータの本質に迫っているとは思えない．

優先順位からすると，アルゴリズムを教えるのは相当低いと考える（もちろん，コンピュータの専門家を育成する場合はまったく話しが別である）．

2.4 物理法則と同じこと・違うこと

情報とは何かということ．体感で知っている物理法則とはどこまでが同じで，どういうところが違うのか．一瞬で内容が全体に伝わり，全く同じものが簡単に複製できて，それを簡単に消滅させることも可能である．

一方，プログラムを微分方程式に対応させて考えると，プログラムの実行と微分方程式の解である物理現象とで似ている部分もある．どちらも，すぐ先に起こることは予想がしやすいが，遠い未来に起こることを予想するのが非常に難しい．単純な方程式（プログラム）からは想像できない複雑な現象が生成されることもある．また，プログラムの実行にある解釈を与えて，物理現象をシミュレーションする，ということの面白さにつながっている．

2.5 プログラミング言語の階層

プログラミングで，別のプログラミング言語を作ることができる，という能力がまた実に不思議である．世の中にプログラミングに似た記述，たとえば料理のレシピや楽譜など，は色々とあるが，言語の上に別の言語を作るというのは無い．たとえでは説明できない．

しかし，この技術が現代の複雑で大規模な情報システム

を支えていることは確かであり，コンピュータの可能性を知る上でも非常に重要な考えである．次の時代には，コンピュータがプログラムを作ることが普通に行われるようになるだろう．そういったことが直感的にどういうことであるか理解できることは重要である．

3. ビスケットによるコンピュータ入門

上記のようなことを念頭において，ビスケット [3], [4], [5], [6] というビジュアルプログラミング言語を用いて，一般の大人を対象としたコンピュータ入門講座を開催した．

ビスケットはメガネと呼ばれる書き換えルールによってプログラミングするビジュアルプログラミング言語である．書き換えをアニメーションとして確認できる．ビスケットは子供にプログラミングの楽しさと可能性を伝えるために開発された．

図 1 はビスケットのプログラムとその実行例である．メガネは左側の図形を右側に書き換える，という意味である．(a) は三角が左と右とでズレておかれているので，この書き換えの結果三角はまっすぐ進むというアニメーションになる．図左側で矢印が三角の動きを示している．メガネの丸のなかには 2 つ以上の絵を入れることができて，その図形の配置に近ければ，という条件になる．(b) は「三角が四角にぶつかると」という条件になり，それは「四角が分解した絵に変わる」となる．この書き換えを連続して適用するので，左側のアニメーションで三角が進んで，いずれ四角にぶつかり分解する．

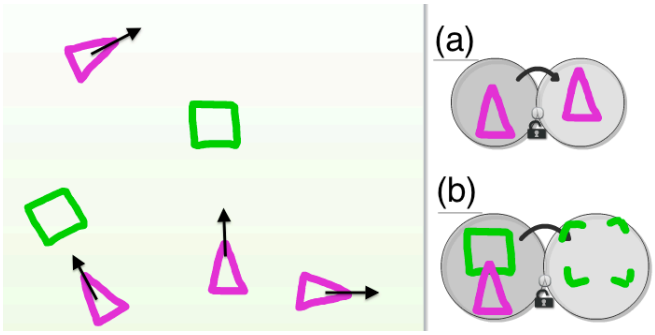

図 1 ビスケットのプログラミング

ビスケットは 2 節で述べたコンピュータについて知って欲しいことを教えるために非常にシンプルな設計になっている．

このビスケットでプログラムを学びつつ，コンピュータ科学に関する話題を交えた体験型講義を行った．参加者は，一般の大人 7 名＋小学生 1 名で，仕事等でコンピュータを使用することはあっても，プログラミングの経験はない人たちである．2 時間ずつ全 3 回で，以下の内容であった．

3.0.1 なぜコンピュータは二進法で計算するのか

コンピュータに計算を教える．これによって，人間とコンピュータの本質的な違いを知ることができる．

図 2 は 2 進数の加算と 1 の生成器である．加算は「0 と 1 が重なると 1 になる」(左上)「1 と 1 が重なると同じ場所に 0 ができ，隣に 1 ができる」(左下) であり，それぞれ 0+1=1, 1+1=10 という二進法の基本的な計算を定義している．生成器は「口が開くとき 1 が出来る」(右上)「口がとじる」(右下) というメガネでできている．

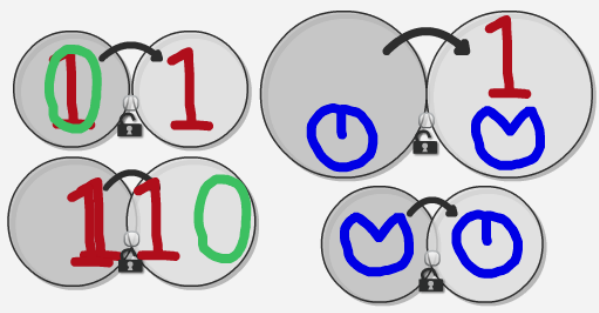

図 2 2 進数の加算と 1 の生成器

画面に口を閉じた生成器を置くと，口を開いて 1 を生成する (図 3)．

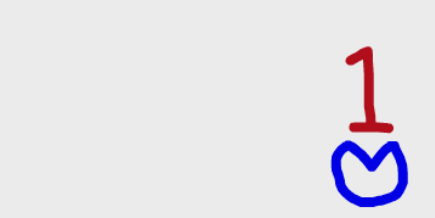

図 3 1 を生成したところ

次に，また口を閉じて開くと，いまの 1 の上に新たに 1 を重ねて生成する．重なった二つの 1 は左下のメガネにより 10 に書き換えられる (図 4)．

図 4 もう一つ 1 を生成して 10 になった

生成器はパクパクしながら連続して同じ位置に 1 を生成するので，たとえば 12 回 1 を生成すれば，1100 になる (図 5)．このようにして二進法のカウンタを作ることができた．

さらに，ふたつ隣にもう一つ 1 の生成器を置いてみる (図 6)．

これで動かすと (図 7)，5 回「1 1」が生成された結果 11001 (十進法で 25) が表示されている．これは 5 ずつ増えるカウンタを作ったことになる．別の場所に 1 のカウン

図 5 12 回 1 を生成した結果 1100 になる

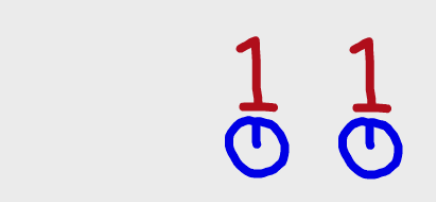

図 6 二つの 1 生成器

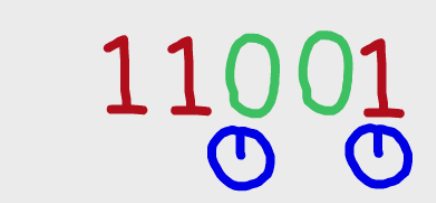

図 7 5 回生成されたのち 11001（25）になっている

タも置いておけば，何回生成されたかを数えることができるので，そこである数で止めるようにすれば，その数と 5 のかけ算を計算したことになる．

生成器の置く位置を変えれば，様々な数の倍数でカウントする．二進法の計算はたった二つのメガネだけで定義できていることが驚きである．二進法は非常に簡単な計算であることがわかる．

次に三進法のカウンタを作る課題をだす．このためには，新たに「2」という絵を描いて，それに関する計算を定義する．二進法であった「1+1 は 10」というのを「1+1 は 2」に修正する．さらに「2+0 は 2」「2+1 は 10」「2+2 は 11」の 3 つのメガネを追加する．二進法から三進法にしただけで，こんなに多くのメガネが必要になる．どちらも人間にとってはやりにくい計算であるけれど，コンピュータにとっては二進法の方が圧倒的に簡単である．

さらに，十進法の計算をさせるにはどうしたら良いか考えてみる．これは本当に作るのは大変である．絵として 0 から 9 までを用意して．それぞれの数同士の加算を定義する．メガネがざっと 50 個以上必要になるだろう．

実は，我々は小学 1 年で加算の九九を暗記したので，十進法の方が簡単に計算できるように感じているだけであって，実際の計算は十進法の方が遥かに面倒なのである．

実際に二進法，三進法，十進法を計算するプログラムを作って動かすことで，人間とコンピュータの本質的な違いがはっきりと理解できるようになる．さらに今まで漠然と「コンピュータは 0 と 1 で動いている」という文字からの知識が，より具体的なイメージをもって理解できるようになる．

3.0.2 シミュレーション

図 8 は感染のシミュレーションである．

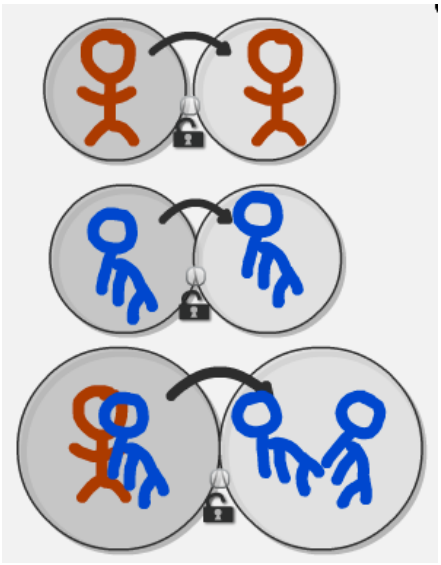

図 8 感染のシミュレーション

「健康な人は横に進む（1 段目）」，「病気の人は上に進む（2 段目）」，「健康な人と病気の人がぶつかると病気が感染する（3 段目）」である．

次に図 9 のように，大勢の健康な人のなかに一人だけ病気の人を入れる．プログラミングがわかる人なら，これを

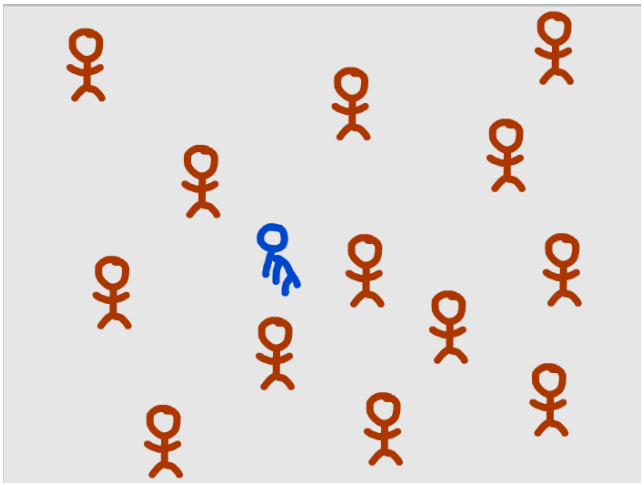

図 9 感染のシミュレーションの実行例 3

実行させるとどのような結果になるか予想がつくであろう．小学生でも感のいい子は理解できる．最初はゆっくりと感染するが一気に広がり全滅する．健康な人と病気の人の動く速度を調整すると，感染力を調整できる．

次に図 10 のように病院をつくる．「病院に病気の人が入ると，病院から健康な人が出てくる」というものである．

画面に病院を建てても，病気の人が病院にぶつかること

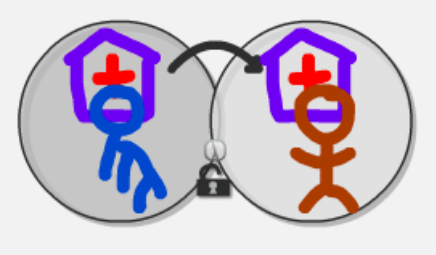

図 10 病院を建てる

は滅多に起きないので中々病気が治らない．図 9 で，病院を 4 軒くらい建てても，全滅されてしまうことがある．

病気がうつるメガネと治るメガネは，形は非常に良く似ている．しかし，片方は倍々で病気が増えて行くのに対して，片方は一人ずつしか病気を治せないというところが違う．うつった人は次は自分が感染源になるということが重要である．

これは指数関数と線形関数の違いであるという説明をしてもよい．線形関数の係数をいくら大きくしても，指数関数にはかなわないということを説明している．数学を使わずに指数関数のそういった性質が理解できる．マルチ商法の怖さの説明でもある．

これがコンピュータの強力さであると同時に怖さでもある．コンピュータウイルスは感染した後自分が次の感染源になってしまう．これは，コピーがまったく同じ動きをする，という基本的な性質から来ている．うわさ話が広まるのも同じ現象であり，そもそも「情報とは何か」と言っても良い．

最初の人の位置を少し変えたり，動きの速度を少し変えたりしただけで，全滅するまでの時間がまったく異なる．これを体験させることで，たとえば天気予報で先の方をあてるのが非常に難しいということにもつながっている．その他に，様々なシミュレーションの例も見せる．

3.0.3 プログラミング言語

オルゴールのプログラムを作る（図 11）．上から「音符はすすむ」「三角は一つジャンプ」「三角の下に丸があれば，その丸を消して次へすすむ」「四角は戻る」「ドはドの音を鳴らす」「停止で音符が消えて終了」である．オルゴールにするにはドの他の音がなる命令も作る必要があるが，ここでは省略し制御構造だけ作っている．

命令を並べる（図 12）．音符が進むとドの音を鳴らし，三角の下には丸が 3 個重なって置かれているので，丸をひとつずつ消して次の命令四角を実行する．四角は戻るので，音符は左に戻される．三回戻され，四回目に音を鳴らした後，三角の下には丸が無いので，後ろの命令（四角）がスキップされて，終了で停止する．

音符の種類を増やせば色々な曲を演奏できるし，繰り返し回数を変えたり，繰り返しが入れ子にさせたりできる．記号を並べ方で色々なパターンの演奏が可能となる．音符

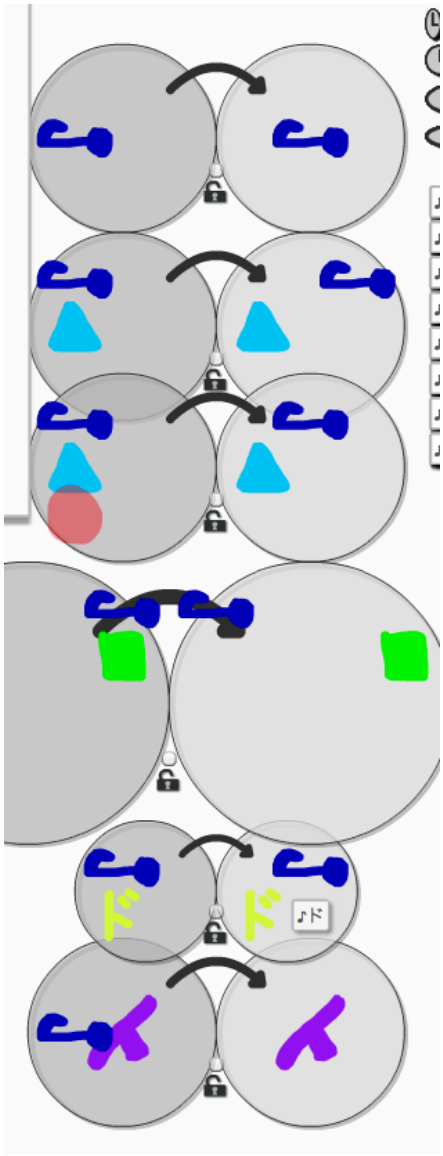

図 11 オルゴールのプログラム

図 12 オルゴールの命令をならべる

はプログラムカウンタであり，非常に小さな CPU を作ったと考えても良い．プログラミング言語はこのようにして作られている．

ここで，ビスケットがどのように作られているかという話をする．ビスケットは ActionScript で書かれている．ActionScript はコンパイルされて swf というフォーマットに変換される．このコンパイラは (たぶん)Java で書かれている．swf は FlashPlayer で解釈され実行される． FlashPlayer は（たぶん）C++で書かれており，実行する CPU

の機械語に変換される．このように何階層ものプログラミング言語の階層の一番上にこのビスケットで作られたオルゴール言語がある．

プログラミング言語の階層という複雑なシステムの作り方の発明のおかげで，コンピュータが現在のように高度に複雑に発展してきたといえる．

3.1 感想

これら 3 回の参加者からの感想を簡単に紹介する．(二進法)「こんな形で二進法とか三進法を習っていたらもっとすっと理解できたと思います．違う感覚のモノに触れるには感覚を使わなくては理解できないんだと思いました」「二進法などの概念を図で見るおもしろさ，加算による変化を視覚で感じる事のふしぎさを改めて知った」(シミュレーション)「プラスのベクトルとマイナスのベクトルが同時に存在している，という当たり前の事をあらためてとらえられる時間でした」「シミュレーションが体感できて，自分で作れるのは，難しくもあったがとても楽しかったです．これまで身近に使っていたパソコンの本来の可能性を感じました」「ほんの少しバランスが崩れると，結果が全く違う物になってしまうことも勉強になりました」「『こういう条件』と『そういう条件』を掛け合わせるとどうなるか？あるいは，『こうするためには？』条件 A と条件 B が必要，と考えることがコンピュータを動かすことなんだ！！と感じました．『わかる』より『感じる』ことなんですね」(プログラミング言語)「階層になってプログラムデータがつくられているというイメージが，3 回の講習中一番体感することができました．3 回を通して，ようやくメガネどうしの直接連動が見えてきたようで『プログラム』を理解するのはとても遠い気がします」「やっぱり私にはコンピュータがむいていないという思いが強くなりました」「『プログラムする』という行為が視覚化されているということは理解できた．約束事に慣れる前になんとなく出来てしまった，という感じ」

二進法，シミュレーションに関しては，実際に体験したことが非常に効果的だったことを表している．一方，プログラミング言語に関しては，そもそも難しい内容であり，前 2 回がすっと理解できた分だけ，より難しさが強調されていたようだ．しかし，プログラミング言語が難しいということが理解されたということは（プログラミング言語が簡単という誤解をされるよりは）ずっと良かったのかもしれない．

4. まとめ

子供向けにプログラミング教育の活動をいくら続けてても，年々改善はしているけれど中々大きくは広がらない．一般の大人に対する教育が非常に重要であると感じている．この大人向け講座は好評であり，内容を充実させて行きたいと考えている．

コンピュータのどういった性質を一般人に伝えて行くべきと考えるかは，コンピュータの専門家によって違ってくるであろう．ビスケットでは不得意なデータ構造やポインタに関係する内容にも伝えるべき重要なことが含まれている．

また，ビスケットによるプログラミング言語の説明は，まだ難しいし，重要なデータ構造であるスタックやキューをビスケットで表現することもできない．それらをビスケット自身を拡張するか，新しい入門用の言語を作る必要がある．

科学を一般人に伝えて行く「サイエンスコミュニケーション」の観点からも，プログラミングは重要である．ここで述べたようなシミュレーションは数学を使わずにプログラムによって科学の現象を説明することができる．どれくらいの精度を求めるかにもよるが，よりきちんとしたシミュレーションが出来る程度に簡略化したプログラミング言語があると，よりやり易くなるであろう．

参考文献

[1] The Royal Society: *Shutdown or restart? – The way forward for computing in UK schools*, 2012. 入手先 ⟨http://royalsociety.org/education/policy/computing-in-schools/report/⟩

[2] 産業競争力会議: 成長戦略 (素案), 平成 25 年 6 月. 入手先 ⟨http://www.kantei.go.jp/jp/singi/keizaisaisei/skkkaigi/dai11/siryou1-1.pdf⟩

[3] Yasunori Harada, Richard Potter : *Fuzzy Rewriting – Soft Program Semantics for Children –*, HCC 2003, IEEE. (2003/10/31)

[4] 原田康徳, 加藤美由紀, Richard Potter: Viscuit:柔軟な動作をするビジュアル言語, WISS 2003.

[5] 原田康徳，加藤美由紀: visucuit:柔らかい書き換えによるエンドユーザ向けアニメーション記述言語, 第 45 回プログラミングシンポジウム, 情報処理学会, 2004 年.

[6] 原田康徳: 体験型ワークショップ用ソフトウェアの開発, 第 50 回プログラミングシンポジウム, 情報処理学会, 2009 年.