Vol.2021-CE-159 No.10
2021/3/13

# プログラミング初学者にとって抽象化はなぜ難しいのか ―お絵かきプログラミングによる学習を分析する―

葛原志保[1]　米屋美雪[2]　伊地知咲希[1]　大坂祥子[3]　内田奈津子[4]

**概要**：プログラミング入門においては，プログラミングの概念を学び，原理を理解することが重要である．プログラミングの基礎知識には，制御構造と抽象化がある．この 2 つは，プログラミング初学者にとっては難しく，躓きやすい．初学者の学習を助けるための工夫は様々あるが，著者らは，絵を描く教材によりプログラミングを学んだ．本論文では，学習の際に躓いた抽象化を取り上げ，”初学者にとって抽象化はなぜ難しいのか”を学習者の視点から分析する．

**キーワード**：プログラミング入門，抽象化，一般情報教育

# Why Abstraction is Hard for Novice Learners of Programming?: Analysis of Learning through Drawing Programming

SHIHO KUZUHARA[1]　MIYUKI YONEYA[2]　SAKI IJICHI[1]
SHOKO OSAKA[3]　NATSUKO UCHIDA[4]

## 1. はじめに

技術が進歩し続ける今，人工知能が組み込まれたプログラムの中での処理を行う一方で，創造的な考えを持つ人間らしさが求められるようになってきている．こうした人間の強みを伸ばしていくことは，学校教育が長年目指してきたことでもあり，社会や産業の構造が変化し成熟社会に向かう中で，社会が求める人材像とも合致するものとなっている[1].

高度情報化社会において，様々な情報の波に流されず必要なものと不必要なものを判断する力が必要である．このような判断力を身に付けるには，論理的に考え答えを導き出す「プログラミング的思考」が大変重要となってくる．

2013 年 5 月には世界最先端 IT 国家創造宣言により，21 世紀の素養として初等教育からのプログラミング教区の必要性が宣言された [2]. 2016 年には，日本学術会議より「情報学の参照基準」が，2020 年に「情報教育課程の設計指針―初等教育から高等教育まで」が取りまとめられ，プログラミング入門はだれもが学習する時代になってきている[3].

しかし,「プログラミング的思考」を身に付けるためのプログラミング教育が浸透し始めている一方で，コーディングを覚えることが目的だという誤解も広がりつつある[1].

これからの社会においては，より身近なところにまでシステムが入り込み，システム開発の専門家だけでは太刀打ちできないため，現場の力が必要である．また，コンピュータやプログラミングに理解がないことで，専門家に丸投げするのではなく，専門家と一緒に議論が出来るその場の基礎的知識を備えた人材が求められている．

学生のうちにプログラミングの概念を理解することで，IT を用いた課題解決や新たな価値の創出など，社会に出てからの活躍の可能性を広げることができると考える．そのため，コーディングを覚えることが重要なのではなく，プログラミング学習を通して論理的思考や構造を学ぶことが大変重要である．

プログラミングを学習する上で基本的な学習事項として，分岐・繰り返し・抽象化がある．著者らは，お絵描きプロジェクトを通じてこれらの事項を学んだ．

本論文では,”抽象化”を取り上げる．抽象化とは，メソッド化することで，それを利用して様々な計算をすることが出来き，メソッド化された中の仕組みを毎回考えることなく利用することである．本論文では，お絵かきプロジェクトにおける，絵のパーツ化を抽象化と位置付ける．抽象化をおこなうことで，パーツの再利用が可能となり，より複雑な絵のプログラムが作れるようになる．

本論文では，学習の際につまずいた点を挙げながら,”なぜプログラミング初学者にとって抽象化は難しいのか”について学習者の視点から分析・考察し，授業の改善提案を行う．

## 2. 先行研究

大岩は，[4]で，抽象化を中心にした入門プログラミング教育の例を具体的に取り上げ，それを効率的に行うための方法を考察している．

1 フェリス女学院大学国際交流学部
2 フェリス女学院大学文学部コミュニケーション学科
3 フェリス女学院大学音楽学部
4 フェリス女学院大学情報センター

ⓒ2021 Information Processing Society of Japan

大岩は，情報を操作する手順を書き下すプログラミングが，コンピュータの進歩にともない大規模化したことから，複雑な手順の記述を人間が誤りなく作り出すためには，抽象化の概念を身につける必要がある，としている．

学習方法としては，制御構造を教えるタートルグラフィックスの命令を使ってイラストを描画する例を挙げている．四角形や三角形などの単純な図形を描画するところから始まり，徐々に繰り返し機能の導入や，単純図形を組み合わせた図形の描画をする．最後には，n 角形を描く関数を作り，それを動かす関数を作る．各命令ないし命令群をどこで分けるか，何を命令群としてまとめるか，という点や，繰り返しの内容はどのような命令群を使うかなどの検討において，抽象化の作業を伴っている．

そして大岩は，「教師に求められることは，学習者の能力範囲を常に意識して，その範囲で適切な課題設定を伝え，つまずいた時に解答を教えるのではなく，自分で考えるために役立つ最小の示唆を与えることである．」と述べている．そのためには，日本語で対象を正確に記述し，伝える能力が必要であるとし，プログラム自体も日本語で記述することで，効率的に入門プログラミング教育を行うことができると述べている．

このように，教育者の視点で抽象化の学習方法を考察した論文はあるが，学習者の視点で行われたものはない．また著者らは，実用言語 Ruby を用いて，実際にプログラミング言語に触れながら抽象化を学習したため，大岩の方法とは異なる．

## 3. どのように学んだか

### 3.1 授業の概要

著者らは，フェリス女学院大学国際交流学部国際交流専門科目「情報発信と世界」を履修した．において実施された． 1～4 年生までが履修可能であり，解放科目であるため音楽学部生や文学部生などの他学部の学生も履修可能である．

講義は，全 15 回で構成されており，図 1 に示した流れで，進められた．ただし，2020 年度は，オンライン授業となり，1 回目の講義の前に実習環境を整えるためのガイダンスの追加があった．

本講義は，前半と後半の二部構成となっている．

前半の 1～6 回では，プログラミングの基礎知識を学んだ．後半の 7～15 回では，グループに分かれてのプロジェクトを行った．

前半は，第 1 回から図形を生成する教材として，事前に用意された図形メソッド(図形ライブラリ)を利用し，簡単な絵を描いた．図形ライブラリは，丸，三角，四角，楕円によって構成されていた．

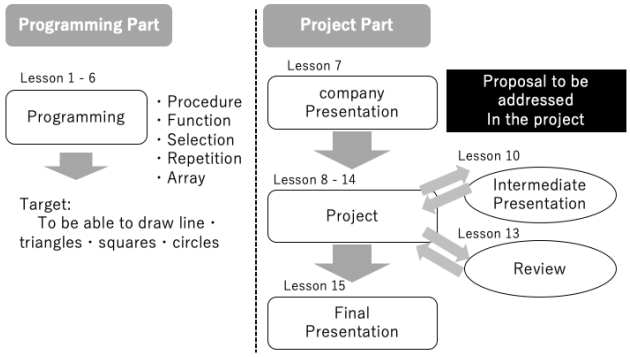



図 1 授業の構成と流れ
Figure 1 Class Structure and Flow

第２回から第５回まで図の生成を教材に，プログラミングの基本的知識を学んだ．前半の最後の回にあたる第６回目には，アニメーション作成の方法についても学習した．

表 1 前半 6 回の授業の構成
Table 1 Structure of Lessons: Lesson 1 - 6

| 回数 | 項目 | 内容 |
|---|---|---|
| 1 回目 | プログラミングの導入 | プログラムとは何であるかを理解し，簡単な絵を描く |
| 2 回目 | 分岐と繰り返し | 絵を描くことを通じて制御構造を理解し，そのプログラムを書く |
| 3 回目 | 制御構造と配列 | 配列について理解し，それを用いたプログラムを書く |
| 4 回目 | 手続き・関数と抽象化 | 手続き（メソッド）の理解と抽象化について理解し，各自の考えた絵を手続きを用いて書く |
| 5 回目 | 2 次元配列と画像 | 配列について理解を深め，2 次元配列と動画の仕組みについて理解する |
| 6 回目 | プログラミングまとめ | 5 回の学習内容を理解し，プロジェクトにつなげる |

### 3.2 教材について

第一回目から図形作成メソッドを利用して，丸・三角・四角・楕円を用いた，プログラミングの基礎知識であるループ・分岐・抽象化などの学習を行った．

第一回目の学習として以下の 3 つの手順で学習した．

1. プログラムについての理解(構造理解)
2. アルゴリズムについての理解(プログラミング言語での理解)
3. 実践として画像の生成(実践)

まず初めにプログラムやアルゴリズムについて，具体的に三角形の面積を求める簡単な式を用いた．手順を用意し，計算の際にはデータを与え計算をさせるというプログラムについての意味から学習した．次のステップとして，三角形の面積計算アルゴリズムを下記で示した Ruby プログラムに直してみる．

```
def     triarea(w, h)
  s = (w * h) / 2.0
  return s
end
```

ⓒ2021 Information Processing Society of Japan

三角形の面積を求める公式をプログラミング言語によって表現することで，メソッド化を学習した上で画像を生成する．

図 2 に第一回で扱ったテキストの例題を示す．座標と色の指定があり，この通りに図を生成するという課題内容である．

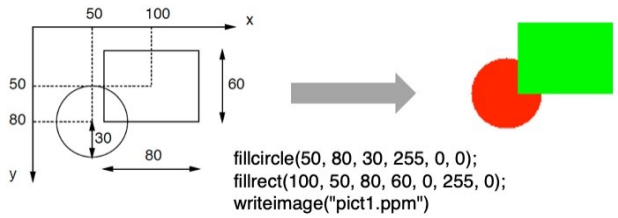



図 2 図形生成の例題

Figure 2 Figure Generation Example

また図 2 の次の例題として図 3 を示す．

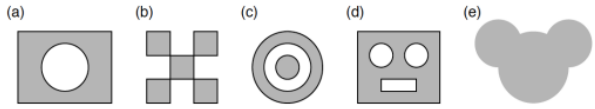



図 3 演習問題

Figure 3 Practice Problems

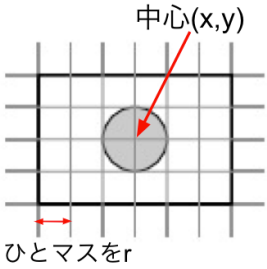
図 2 の例題で図が生成できたことを確認した後，大きさや色，図形の位置を自由に変更して確認する．次のステップとして図 3 の図形を自分で設計し，図形を書くプログラムを作成する．この際，カラーコードの存在を認識しておくべきである．このように一つずつ段階を経て学習することで，より実践的に理解を深めることができた．

### 3.3 テキストの内容と演習問題：抽象化

図 4 に示した例題を用いて，抽象化についての学習を行った．最初から複数の図を想定するのではなく，一つの図をメソッド化することを考える．

まず図を書くときに必要な設計図の作成を行う．第一に基準点となる x 座標・y 座標，一マスの大きさを設定することが重要である．メソッド化を行う上で，設計図をきちんと書くことが出来ていないと複雑な図形やパーツの使い回しができなくなってしまい，チームにおける活動に無駄な作業を強いることになる．

図 5 の例では，図 4 のメソッドを用いることにより，色や大きさの異なる複数の絵を生成することが可能となる．




```
def flag(x, y, r, r1, g1, b1, r2, g2, b2)
  fillrect(x, y, r*6, r*4, r1, g1, b1)
  fillcircle(x, y, r, r2, g2, b2)
end
```

図 4 抽象化の例題：旗

Figure 4 Examples of Abstraction: Flags

図 5 旗の例題の出力結果

Figure 5 Output of Example: Flags

## 4. 問題点の抽出方法

著者らは，次の 2 つの方法で，抽象化の問題点について情報収集を行った．

- 履修者に対してのアンケート

（主なアンケート項目）

・ 抽象化をいつ理解したか
・ いつの時点で抽象化されたコードを自力で書けるようになったか
・ 抽象化が難しかったか
・ 抽象化をどのように乗り越えたか
・ 抽象化について，どのような教材があったらいいと考えられるか

- 著者らによるディスカッション

アンケート調査は Google form にて，2021 年 2 月 8 日から 2 月 10 日まで行った．対象は，2018 年度から 2020 年度の授業履修者で，まだ在籍している 28 名とした．短い調査期間であったが，回答は 14 件あった．

## 5. 調査結果・分析・考察

まず，「抽象化ができないこととは？」について検討する．著者らは，調査を行うにあたって，抽象化について，以下の 2 つに分類する．

ⓒ2021 Information Processing Society of Japan

- 図 4(旗のメソッドの図)のように図形の座標(x, y)を中心として表現できることを，抽象化できるとする
- パーツの塊としてメソッドは作れるが，具体的な数値でしか表現できないことを，抽象化できないとする

以下で，抽象化ができる・できないについて．具体例を示す．

まず，図 6 に抽象化できている例を閉めず．表現したい絵のイメージを設計図に落とし，図形の座標を(x, y)を中心とし，方眼紙のひとマスを基準として，考えていることがわかる．この設計図を元に，コードを書き，絵を出力している．

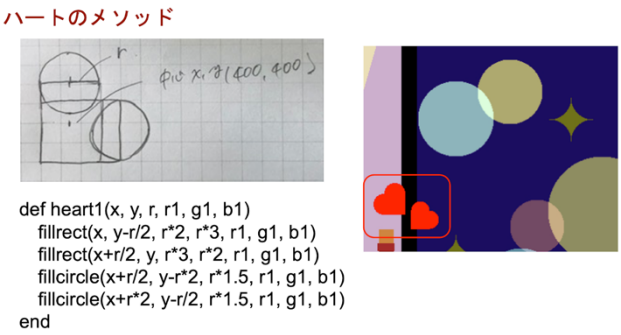
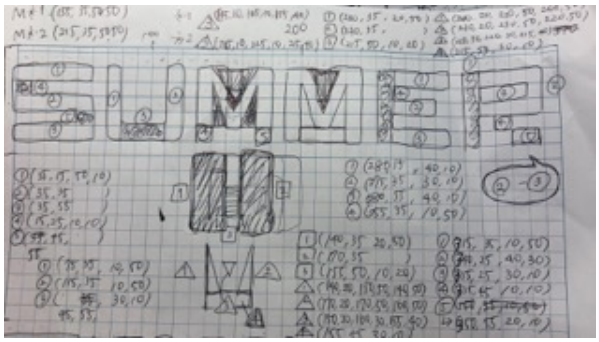



図 6 抽象化の良い例

Figure 6 Good Examples: abstraction

次に，抽象化できていない例を，図 7 に示す．図 7 の下の図では，考えた図をプログラミングのコードとしてまとめたメソッドとして作成されているが，具体的な数値でメソッドを作っていることがわかる．上の図からも，基準点を決めずに，具体的な数値を計算した設計図になっていることがわかる．



```
def summer
  fillrect(35,15,50,10,0,0,0,0)
  fillrect(35,35,50,10,0,0,0,0)
  fillrect(35,55,50,10,0,0,0,0)
  fillrect(15,25,10,10,0,0,0,0)
  fillrect(55,45,10,10,0,0,0,0)
  fillrect(75,35,10,50,0,0,0,0)
  fillrect(115,35,10,50,0,0,0,0)
  fillrect(95,55,30,10,0,0,0,0)
  fillrect(155,35,50,50,0,0,0,0)
  filltriangle(140,20,150,50,140,50,255,255,255,0)
  filltriangle(170,20,170,50,160,50,255,255,255,0)
  filltriangle(145,10,165,10,155,40,255,255,255,0)
  fillrect(155,55,30,10,255,255,255,0)
  fillrect(215,35,50,50,0,0,0,0)
  filltriangle(200,20,210,50,200,50,255,255,255,0)
  filltriangle(230,20,230,50,220,50,255,255,255,0)
  filltriangle(205,10,225,10,215,40,255,255,255,0)
  fillrect(215,55,30,10,255,255,255,0)
  fillrect(280,15,40,10,0,0,0,0)
  fillrect(275,35,30,10,0,0,0,0)
  fillrect(280,55,40,10,0,0,0,0)
  fillrect(255,35,10,50,0,0,0,0)
  fillrect(315,35,10,50,0,0,0,0)
  fillrect(340,25,40,30,0,0,0,0)
  fillrect(335,25,30,10,255,255,255,0)
  fillrect(335,45,10,10,0,0,0,0)
  fillrect(350,55,20,10,0,0,0,0)
  writeimagef("summer.ppm")
end
```

図 7 抽象化の悪い例

Figure 7 Bad Examples: abstraction

前述のアンケートから，「抽象化についていつの時点で理解できましたか？」の回答を，図 8 に示す．

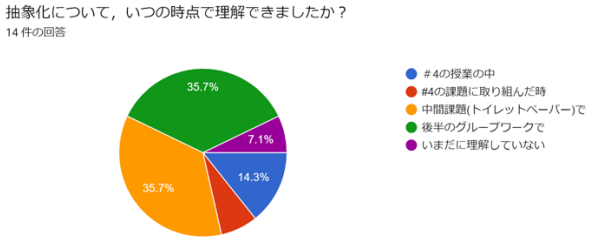



図 8 円グラフ：「抽象化についていつの時点で理解できましたか？」

Figure 8 Pie chart: “When you can understand abstraction?”

回答者のうち 57.2%が，グループワークに入る前までに理解できたと回答している．中間課題の提出物を調べたところ，プログラムが読めた提出物のうち約 25%が，抽象化できていなかった．できているものも，テキストや友達の助言などを元に完成させたと推測でき，自分の力でできたと自信を持って回答できるものがいたのではないかと推察する．

次に，アンケートの「抽象化について，いつの時点でプログラムを自分の力で書けるようになりましたか？」という問いについての結果を，図 9 に示す．

ⓒ2021 Information Processing Society of Japan

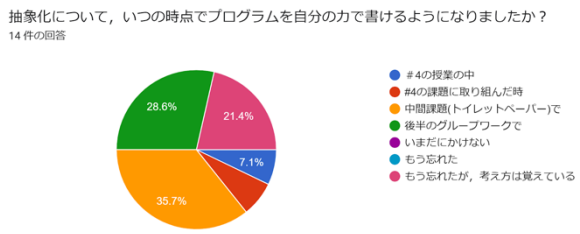



図 9 円グラフ：「抽象化についていつの時点でプログラムを自分の力で書けるようになりましたか？」
Figure 9 Pie chart: “When you can write the program abstracted by yourself?”

この結果から，約半数の学生は，前半の授業で習得していたが，課題やグループワークなど，より手を動かす機会で習得していることが読み取れる．

次に，なぜ抽象化ができないかの原因いついて考える．アンケートの「抽象化のどういう点が難しかったか？」で，14 件中の 4 件が，作図が出来ないこと(設計ができないこと)を挙げた．

また，筆者らの議論でも同様に，作図の難しさが挙げられた．この原因は，頂点や中心点からなる図形の構造が理解できていないことである．

具体的には，自分の書きたい図形を書く際に，いくつかのパーツの中心点や頂点をどこにするかということ，その中心点や頂点をどこに置くかということが問題となっている．実際に多くの設計図に見られた状況として，座標が書かれていない，イラストの例がある．その一例を図 10 に示す．

座標が書かれていないと、パーツの位置や大きさが変わってしまい、自分の作りたい絵とは異なる絵を作ってしまう．

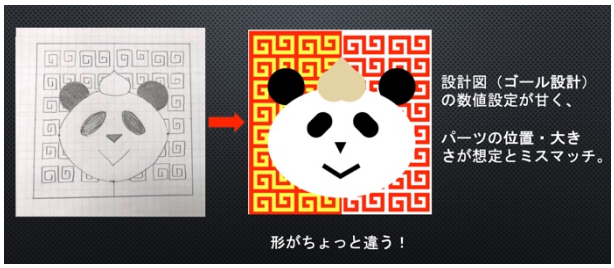



図 10 座標が書かれていないイラストの例
Figure 10 Example of Illust: coordinates are not written

この原因を乗り越えるには，書きたい図形に近づけるように何度もコードを書き直すことではなく，まず初めに，座標や中心点を意識した設計図を作ることが重要である．

その他の原因として，コードの実行時のエラーに対応しづらいことがある．

著者らが，経験した，主なエラーは，次の 2 点である．

- メソッドの引数が足りないことに対するエラー
- 間接的ではあるが，分岐やループの文法のエラー

これには，コード自体を修正する必要があるが，エラーメッセージの内容の理解ができないことから，対応ができないケースがあったことがあげられる．

その他，エラーではないが，自分の描きたい絵とは異なった絵が出るケースがある．これは，絵が出力されることから，結果をみて修正することができる．

著者らが行なったディスカッションからは，プログラミング初学者がエラーに対応しづらい要因について，次の 2 点があった．

- 図形ライブラリの構造を十分に理解できていない
- 自分の描きたい絵の構造を理解できていても，ある一部の中心点や頂点の位置を間違えて捉えていること

コンピュータに不慣れな著者らは，最初に学んだ「fillcircle(x, y, rad, r=0, g=0, b=0, a=0.0)」は，何度も講義にて使ってきたため，徐々に慣れ抵抗は軽減された．しかし，「filltriangle(x0, y0, x1, y1, x2, y2, r=0, g=0, b=0, a=0.0)」を使うことはあまりなかったことから，抵抗のある学生が多くいた．

それゆえに，新たな図形のメソッド(たとえば，三角形)を利用する際に，頂点の数として，6 個の要素を入力する必要があり，それがどのような構造になっているかについて十分に理解できていなかったことが推察される．

２つ目の原因は，自分の描きたい絵への図形の構造を理解し，図形の設計図を書けたとしても，ある一部の中心点や頂点の位置を間違えて捉えてしまうことである．

具体的には，自分の描きたい絵の構造を理解し，設計図を書けたとしても，複雑な絵を書く場合には，一部分の図形の基準点に注意がいかず，図 12 のように意図しないものが出力される．このような場合，複数回の試作と調整を重ねることで，完成に至ることができる．完成図を，図 13 に示す．

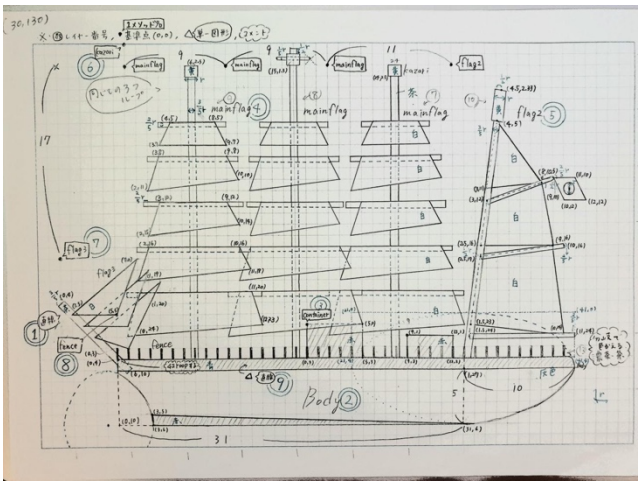


図 11 完成図を作れた設計図
Figure 11 Design to make completed figure

図 12 失敗例
Figure 12 Failure Example

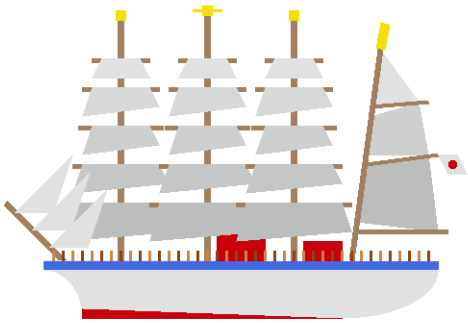


図 13 完成図
Figure 13 Completed Figure

これらの原因を乗り越えるには，以下の点が必要となる．

- 新たな図形のメソッドを何度も使うことで，そのメソッドの構造を十分に理解すること
- 設計図をきっちり書くこと
- 基準点を定めること
- パーツごとの試作とそれらの統合を段階的に進めること
- エラーコードに出された言葉の意味を自分自身で理解すること
- エラーコードからエラーがある部分を自分自身で探せるようになること

これらの項目があやふやな学生にとっては，抽象化は難しく，理解を難しくしていたと推察する．

## 6. 授業の改善提案

アンケートや授業の成果物にて，グループワークが終了した時点で，全学生が抽象化ができたことがわかった．しかし，プログラミング初学者である全学生が抽象化を難しいと感じずに，より早く抽象化ができた方が良いであろう．

それゆえに，5 章での結論を踏まえて，著者らが議論した結果，授業の流れと授業で使用する教材の修正を考えた．

まず，授業で使用するテキストは，文章が長く，初学者にとって，誰もわかりやすい表記にはなっていない．そのため，授業では，補助教材として，色分けされたパワーポイントや追加資料を使用した．著者らが使用してきた初等中等教育の教科書は，フルカラーであったことから，できれば，色分けされたテキストがあると，なおよい．

次に，著者らが学んだ授業では，授業の後半にあるグループワークは，授業を受けている全学生の理解度が深まるため，変えなくてもいいと考えている．

ここでは，前半について提案していきたい．第 4 回目の授業を第 2 回目の授業に入れることを提案したい．

第 4 回目の授業は，簡単な絵で利用されている抽象化を学べる授業になっている．第 4 回目の授業よりも先に抽象化を学ぶことで，より早く抽象化が出来た方が良いであろう．

授業の各回の構成は，単元を数式で捉えることと丸や四角形のみしか現れないメソッドを活用して取り組む課題の 2 段階となっていた．1 回目の授業では，具体的な座標から，順次処理とプログラムの書き方，実行の仕方などの手順を覚えることが目的であった．しかし，具体的な数値で進める例題は，抽象化を考えることを困難にしていると考えた．

毎回の授業にて，複数の抽象化されたメソッドを作ることで，眼鏡や円を二つ重ねたドーナツといった汎用性のある形をシンプルに作ることを経験することで，反復練習にもつながる．

こうすることで，抽象化ができない人でも徐々に抽象化ができるようになると考えた．

例えば，図 3 は，具体的な数値をいれて，1 つの絵を出力する課題であった．この課題を抽象的に捉え，汎用的に活用できるよう，取り組むことで，汎用的に利用できる意義や再利用についての意識が高まり，定着につながるのではないかと考える．

4 回目の演習問題では，図 14 に取り組む．しかし，この問題では，各図形の中心座標に分数を用いてなければならず，誰もが学びやすい問題にはなっていない．

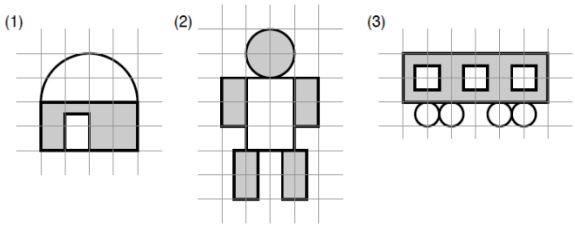



図 14 演習問題(4 回目)
Figure 14 The Forth Practice Problems

この絵自体には問題はないが，数学が苦手な対象者にも取り組みやすい課題とするのであれば，より計算が用意に

ⓒ2021 Information Processing Society of Japan

なるよう方眼紙の使い方を工夫し，整数で扱えるようにする方が良いと考える．

上述したことから，著者らは，表 1 の 4 回目の内容を 2 回目に移動することを提案したい．表 2 に，6 回の構成を修正したものを示す．

表 2 構成の修正案
Table 2 Proposal to Modify the Structure

| 回数 | 項目 | 内容 |
|---|---|---|
| 1 回目 | プログラミングの導入 | プログラムとは何であるかを理解し，簡単な絵を描く |
| 2 回目 | 手続き・関数と抽象化 | 手続き（メソッド）の理解と抽象化について理解し，各自の考えた絵を手続きを用いて書く |
| 3 回目 | 分岐と繰り返し | 絵を描くことを通じて制御構造を理解し，そのプログラムを書く |
| 4 回目 | 制御構造と配列 | 配列について理解し，それを用いたプログラムを書く |
| 5 回目 | 2 次元配列と画像 | 配列について理解を深め，2 次元配列と動画の仕組みについて理解する |
| 6 回目 | プログラミングまとめ | 5 回の学習内容を理解し，プロジェクトにつなげる |

授業の流れと授業で使用する教材の修正を行うことにより，初学者が抽象化を難しいと感じずに，より早く抽象化が出来るようになると期待する．

## 7. まとめ

本論文では，プログラミング入門の講義全 15 回を振り返って、なぜプログラミング初学者にとって抽象化が難しいのかについて、初学者の躓く点や困難だった点を分析し、構造理解が不十分だということが分かった。それに基づき、よりプログラミング初学者にとって抽象化が理解しやすいように改善提案を行った。

プログラミング初学者且つ文系の学生にとっては、文字や数字のみでプログラミング入門の学習を行うことは大変困難である。そのため、本論文で取り上げたレベルの教材を使うことが適切であった。また，プログラミングを用いて自由な絵を生成することで、論理的な思考だけでなく，発想力や創造力を鍛えることができた．

プログラミング構築能力を養う上で、設計図を描くことは重要であることがわかった。導入から数字のみで行うのではなく、お絵描きプログラミングを行う際は、方眼紙を用いて設計図に起こすことで、一マスの大きさや基準点などを意識しながら、対象がどのような構造で作られているのかを可視化することができる。可視化することで完成図をイメージしやすく、構造理解にも繋がることから，お絵かき教材は有効であることがわかった．可視化により，構造を理解することで論理的思考に慣れ、他者に自分の考えを伝えることも、他者の考えを的確に受け止めることもできるため、高度なコミュニケーションが可能になることもわかった。また，仕組みを理解していることで，問題発生時にも迅速な対応ができ、社会においても目的を達成するための最適な方法を導き出すことにも役立つと考える．

大岩は，「論理思考教育を行なおうとすると，プログラミング活動の中で，アルゴリズム構築が一番役に立つ部分である．従来のプログラミング授業では，ここに到達する前に，文法理解で終ってしまっていたために，社会に出て役に立つ教育が行なわれてこなかったのである.」と述べている[5]。

[1]では，「こうした「プログラミング的思考」は、急速な技術革新の中でプログラミングや情報技術の在り方がどのように変化していっても、普遍的に求められる力であると考えられる。 また、特定のコーディングを学ぶことではなく、「プログラミング的思考」を身に付けることは、情報技術が人間の生活にますます身近なものとなる中で、それらのサービスを受け身で享受するだけではなく、その働きを理解して、自分が設定した目的のために使いこなし、よりよい人生や社会づくりに生かしていくために必要である。言い換えれば、「プログラミング的思考」は、プログラミングに携わる職業を目指す子供たちだけではなく、どのような進路を選択しどのような職業に就くとしても、これからの時代において共通に求められる力であると言える。」と述べられており，プログラミングは，これらの知識として全ての人に求められていることから義務教育となった．

Society5.0 で実現する情報社会では，知識や情報を共有し、分野横断的な連携が求められている[6]．必要な情報を得て分析する過程で，プログラミング的思考を社会に出る前に養うことは大変重要であり，高等教育においても学ぶ必要があり，誰もが学びやすく有益な教材が求められている．

本論文が，貢献できることを期待する．

## 参考文献

[1] 文部科学省 教育課程部会教育課程企画特別部会：小学校段階におけるプログラミング教育のあり方について(オンライン)，入手先 <https://www.mext.go.jp/b_menu/shingi/chukyo/chukyo3/053/siryo/__icsFiles/afieldfile/2016/07/08/1373901_12.pdf>(参照 2021-02-10).
[2] 首相官邸：世界最先端 IT 国家創造宣言 (2013 年 6 月 14 日)(オンライン)，入手先 ⟨http://www.kantei.go.jp/jp/singi/it2/kettei/pdf/20130614/siryou5.pdf⟩(参照 2020-08-23).
[3] 日本学術会議：情報教育課程の設計指針―初等教育から高等教育まで― (オンライン)，入手先 ⟨http://www.scj.go.jp/ja/info/kohyo/kohyo-24- h200925-abstract.html⟩(参照 2021-02-10).
[4] 大岩元：プログラミング入門教育と受講者モデル–抽象化の教育–，研究報告コンピュータと教育，CE-136(10)(2016).
[5] 大岩元：プログラミング教育の社会的意義－初等教育の視点から考えるー，情報教育シンポジウム 2016 論文集(2016).
[6] 内閣府：科学技術政策 Society 5.0(オンライン)，入手先< https://www8.cao.go.jp/cstp/society5_0/index.html>(参照 2021-02-10).

ⓒ2021 Information Processing Society of Japan