

プログラミングをはじめて学ぶための スクリプト言語の開発と教育実践

倉光君郎
横浜国立大学大学院工学研究院
kimio@ynu.ac.jp

概要

本論文は、スクリプト言語 KonohaScript を用いたプログラミング教育の提案である。今日、プログラミング言語の高度化、複雑化が進み、プログラミング教育は難しくなっている。我々は、C 言語/Java の導入としてスクリプト言語を開発した。設計方針やスクリプト言語の特徴である対話的なプログラミングを活用したプログラミング言語を紹介する。

1 はじめに

現代社会では、ソフトウェアなしには社会活動が成り立たないほどソフトウェアは必要不可欠な要素となっている。また、ソフトウェアに対する理解を高め、さらにソフトウェア開発に参加できる人材を育てることは国際競争力に直結する。

ソフトウェアへの理解を高めるもっとも簡単な手段は、たぶん実際にプログラミングを体験し、ソフトウェア開発を体験してみることである。ソフトウェアは、ハードウェアに対し柔軟なものという語源からきており、この柔軟さはプログラミングを通して直感的な理解がえられる。さらに、プログラミングを通してアルゴリズムを学ぶことができれば、情報や構造に対する思考法の習得にもつながる。

幸い、急速な進歩を遂げたコンピュータ技術により、コンピュータ機材の価格は大幅に低下し、教育機材にかかるコストは障壁ではなくなっている。しかしながら、このような経済的な環境にも関わらず、プログラミング教育の成果はあまり変化していない。学生にとってプログラミングが学びやすくなったと言えない。

この原因のひとつとして、ソフトウェア技術の発展にともない、ソフトウェア開発のツール類が極めて高度に専門化したことが考えられる。プログラミング言語も、ツールの一部として、極めて複雑なものに進化してきた。学生にとっても高度化、さらに一部は自動化が進んだツールを使うことは、基本的なステップの本質的な理解を妨げる結果になっている。同時に、教員の立場からみても教材として扱うことがより難しくなっている。

我々は、大学など高等教育において、次世代のソフトウェア人材を育てるための学びやすいプログラミング言語が必要であると考え、そこで、オープンソース開発された KonohaScript をプログラミング教育の実践経験や学生たちからのフィードバックを含め、教育向けのアップデートを行ってきた。本稿では、KonohaScript の、特に教育向けの言語設計を紹介し、同時にスクリプトを用いたプログラミング教育の活動事例を報告する。

本論文の構成は以下のとおりである。第 2 節では、現在のプログラミング教育が抱える言語選択の問題について述べる。第 3 節では、新しい教育向けの言語設計とその指針を述べる。第 4 節では簡単に言語と設計を紹介する。第 5 節では、教育現場での事例を報告する。第 6 節では関連研究を概観し、第 7 節では本論文を総括する。

2 言語選択の問題

今日の高等教育では、理工系学部を中心に広範囲にプログラミング教育が行われている。プログラミング教育には、まずプログラミング言語が必要となる。教員は、講義や演習で用いるプログラミング言語を選択する余地が生じる。しかしながら、カリキュラムや学習内容の実用面から、プログラミング言語選択の余地は限られている。

2.1 カリキュラムの立場から

現在のプログラミング教育では、プログラミング言語として C 言語や Java が多く採用されている。逆に、カリキュラムの一部として、プログラミング教育が導入されている場合（つまり純粋な教養科目でない場合）、C 言語や Java をまったく教えずともよいという大学は極めて少数であると考えられる。

まず、情報系学科の場合、いくつかの科目群（コンピュータアーキテクチャ、アルゴリズム、ソフトウェア工学、コンパイル構成法）を縦断的に学ぶため、プログラミング言語の選択はカリキュラムの視点から重要である。たとえば、オペレーティングシステムやアーキテクチャを理解するため、システム言語（C 言語）は必要となるなど制約がある。それ

表 1: Java 言語の進化の変遷

リリース年	追加機能
1997	Java 1.1 インナークラス リフレクション
1998	Java 1.2 コレクション strictfp
2002	Java 1.4 assert 文
2004	Java 5 総称型 アノテーション 自動ボックス化 列挙型 for 文 拡張
2011	Java 7 ダイヤモンド演算子 switch 文 拡張

らをふまえ、最初のプログラミングを習う言語として、C/C++、Java が選択されることが多い。

一方、情報系を主体としない学科の場合は、プログラミング教育の言語選択の自由度はあがる。しかし、卒業研究などでシミュレーション、データ解析などで、プログラミング能力を必要とすることは増え、多くの教員からプログラミング教育への要望として実用性の高いスキルを求められる。その場合、やはり C 言語や Java など、実用性の高いプログラミング言語が選ばれることが少なくない。

2.2 最先端の技術開発の動向より

もうひとつ、今日のプログラミング言語は製品として提供されている。この製品の開発環境も少なからぬ影響を与えている。プログラミング言語は、ソフトウェア開発の中核技術として、北米のソフトウェア企業 (Microsoft, Oracle*, IBM, Google など) で積極的に開発されている。さらに、オープンソースコミュニティにおいても開発が盛んで、ユーザの要望を取り入れる形で常に機能的な進化が続いている。

たとえば、Java は 1990 年代は非常にコンパクトにまとまったオブジェクト指向言語であった。しかし、オブジェクト指向ソフトウェア工学の発展、ライバル言語となる C# の登場があり、言語仕様は大きく拡大してきた。表 1 は、簡単な Java 言語の機能的な進化を載せている。最先端の技術を駆使するソフトウェア開発者には使いやすいものとなったが、

これは教員[†]をはじめてプログラミングを学ぶ学生には必ずしも当てはまらない。同じことが、C++ から次世代仕様の C++0x でも発生しつつある。

一方、C 言語は、Java や C++ に比べると、言語進化は止まっている。そもそも言語設計が抽象化が少なく、文字列の扱いもすべてポインタ操作で行うという単純化された設計[‡]であるためである。C コンパイラは、GCC などオープンソースプロジェクトが開発している以外は、各種ベンダーは C++ のサブセットとして C 言語を提供している。多くの大学で導入している演習環境 Microsoft Visual Studio では、C 言語であっても、C++ を使うことになる。

2.3 スクリプト言語の活用

対話的なプログラミング環境を備えたスクリプト言語 (インタプリタ型言語) は、C 言語や Java のようなコンパイル言語に比べて習得しやすいと考える人は少なくない。これは、コーディングから実行までのサイクルが短く、コンパイルエラーによる初学者のつまづきが軽減されるだろうという予測にもとづいている。実際、高校生向けの情報科目では、インタプリタ型の BASIC を用いている。

また、Perl, Python, Ruby や JavaScript など、Web アプリケーション開発分野において、スクリプト言語の産業界での採用が増えている。さらに、今後、ますますアジャイルなソフトウェア開発スタイルが広がっていくと予想される。しかしながら、プログラミング教育の世界では、C/C++ や Java に比べると、実用性の評価はまだそれほど高くなく、米マサチューセッツ工科大学 (Python) や 東京大学教養学部 (Ruby) など、一部の大学で採用されることはあるが、それほど一般的な傾向となっていない。

ここで考えられるひとつの理由は、複数のプログラミング言語を学ぶコストの問題である。多くの大学では、C/C++、Java の習得は、他の科目との関係から必須であり、そのため複数の言語を学ぶことになる。スクリプト言語は、確かに習得しやすいかもしれないが、いくつも言語文法を授業で扱うと大きく混乱する学生も少なくない。さらにスクリプト言語の多くは、記号を多用した極めて実用的な文法を用い、変数スコープや型など言語システムも C/C++ と多くの点で異なる。表 2 は代表的なスクリプト言語と Java との違いをまとめている。

3 設計目標

授業単位の時間数は、予習復習の時間を考慮に入れても、2 単位 60 時間程度と上限がある。この限

[†] コンピュータ科学分野で博士号をもつ教員であっても耳慣れない技術が少なくなく、仮に学生から質問されても正しく答えることが難しいことが少なくない。

[‡] C 言語の単純化された設計は、最初に習うプログラミング言語に適しているかどうかという疑問がある

*Sun Microsystems を買収

<pre>class Counter { // Java int cnt; Counter(int cnt) { this.cnt = cnt; } void count() { this.cnt++; } } Counter c = new Counter(0); c.count()</pre>	<pre>// JavaScript var Counter = function(num) { this.cnt = num; this.count = function() { this.cnt++; }; }; var c = new Counter(0); c.count();</pre>
<pre>class Counter: # Python def __init__(self, n): self.cnt = n def count(self): self.cnt += 1 c = Counter(0) c.count()</pre>	<pre>class Counter # Ruby def initialize(n) @cnt = n end def count @cnt = @cnt + 1 end end c = Counter.new(0) c.count()</pre>

図 1: Java, JavaScript, Python, Ruby, における Counter クラスによる文法比較

られた時間数で、より効率よくコーディングとプログラム実行の演習が行えるスクリプト言語は、プログラミング入門に用いやすい。そこで、現在の C/C++, Java を中心としたカリキュラム構成の中でプログラミング導入として用いる新しいスクリプト言語が必要と考えた。

具体的な設計目標は以下のとおりである。

- 学生の複数言語を学ぶ混乱を避けることができる C/C++, Java と 言語文法の互換性が高いことが必要である
- プログラミングの概念を学ぶことができる 型システム、変数スコープ、演算子などが同じである。また、必要に応じてオブジェクト指向プログラミングも可能である
- コンピュータの基本原理を学ぶ 過度な抽象化を行わず、整数や浮動小数点数などが低レベルでみることができる
- 本格的なソフトウェア開発の導入となる 必要に応じ、コンパイルや型検査、デバッグなど上位のソフトウェア開発スキルを例示できる

4 KonohaScript の言語設計

KonohaScript は、静的型付けのオブジェクト指向スクリプト言語である。C 言語で開発され、Windows, MacOS X, Linux など様々なプラットフォーム上で動作させることができる。次のサイトからオープンソースソフトウェアとして、ダウンロードすることができる。

<http://www.konohascript.org/>

我々は、KonohaScript を設計/開発するにあたり、プログラミング入門教育に活用できるように様々な設計上の配慮を行った。本節では、教育向けという視点から KonohaScript の言語設計を紹介する。

4.1 データ構造

KonohaScript の特徴は、静的な型付け言語である点である。したがって、変数は C/C++, Java と同様に変数宣言を使ってプログラミングを行うことができる。型の名前は、原則として Java 言語から採ってある。

```
boolean isdebug = true;
int n = 1;
String name = "naruto";
```

KonohaScript では、数値の高度な抽象化を行わず、CPU が提供する整数、浮動小数点数がみえるように設計されている。ただし、16 ビット整数/32 ビット整数など、細かい使い分けは行わず、64 ビット整数 (int)、倍精度浮動小数点数 (float) に統一されている。アーキテクチャや 2 進数演算を合わせて説明しやすいようになっている。

KonohaScript は、JST/CREST 「ディベンダブル組み込みオペレーティングシステム」プロジェクトの一部として開発されているため、先端の研究成果にもとづき複雑な概念や機構を導入している。我々は糖衣文法を加えることで、複雑な概念を理解することなく初学者にとってなじみやすい概念で利用できるように配慮している。たとえば、総称型を意識しなくても配列として利用できる。さらに、学生からのフィードバックも受けて、C スタイルの配列でもそのまま混乱なく書けるようになっている。

```
Array<int> a; // 総称型
int[] a; // 配列
```

表 2: KonohaScript 基本クラス

種類	クラス名
Top	Object
Data Value	Boolean Number Int Float String Bytes
Collection	Array<T> Iterator<T> Tuple Map <K,V>
Language	Class Method Func Translater System Script Context NameS- pace Exception
I/O	InputStream OutputStream Connection ResultSet
Misc	Regex Converter
Bottom	dyn

表 3: KonohaScript ステートメント

種類	ステートメント
分岐	if/else switch/case
ループ	while do/while for foreach*
例外	try/catch/finally
ジャンプ	break continue return throw
クラス	class/extends
デバッグ	print assert

```
int a[];    //C style
```

表 2 は, KonohaScript がサポートしている基本クラスの一覧である。Java 言語に由来するクラスの場合は, そのフィールドやメソッド名も含めて, 可能な限り忠実にクローンする方針を採っている。次は, Java のプログラミング教科書によく出てくる print 文であるが, KonohaScript でも同じそのまま解釈して動作することができる。

```
System.out.println("hello,world\n");
```

4.2 文法

プログラミング言語は「ことば」である。現在, C 言語から始まり, C++, Java, C#と続く言語文法の系統はプログラミング言語の主流 (標準語) となっている。KonohaScript の設計では, 文法も可能な限り, 互換性を保ち, ことばとして混乱なく使うことを目指して設計されている。

表 3 は, KonohaScript のステートメント一覧である。*がついたステートメントは, KonohaScript 独自の意味論が定義されている。制御構造や例外処理, クラス定義など, 言語としてある程度表記法が確立している。KonohaScript では, C や Java と同じ文を採用するときは, 必ず同じ振る舞いを実現するようにしている。より高いレベルのソースコード互換性をもっている。

次は, 図 1 であげた Counter クラスの KonohaScript 版である。Java と完全にソースコード互換である。

```
class Counter {
    int count;
    Counter(int counter) {
        this.cnt = cnt;
    }
    void count() {
        this.cnt++;
    }
}

Counter c = new Counter(0);
c.count();
```

また, 教育現場では, Java のようにまずオブジェクト指向のクラス定義から入らなければならない言語は扱いにくいこともある。Konoha では, オブジェクト指向プログラミングの理解なしに, 関数など手続き言語として利用することができる。

```
int fact(int n) {
    if(n == 1) return 1;
    return n * fact(n-1);
}

print fact(10);
```

KonohaScript では, 演算子は文法の一部とみなし, 文法と同様に Java 言語が提供する演算子を忠実に再現する方針をとっている。例えば, KonohaScript は純粋なオブジェクト指向であり, 整数は不変なオブジェクトとして扱われるが, 前置 ++i と後置 i++ 区別を区別して, インクリメンタル演算子をエミュレーションしている。

- Java から継承した演算子. 条件演算子 (&&, ||, !), 三項演算子 (? :), 比較演算子 (==, !=, <, <=, >, >=), 型判定演算子 (instanceof), 四則演算子 (+, -, *, /, %), インクリメンタル演算子 (++i, --i) ビット演算子 (<<, >>, |, &, ^, ~)
- 読みやすさのための別名. 条件演算子 (and, or, not), 四則演算子 (mod),

4.3 対話的プログラミング環境

KonohaScript の特徴は, 実行時における評価 (eval) によるプログラム更新と実行が可能である点である。加えて, Python スタイルの対話的なシェルを備え, プログラムを対話的に動作させることができる。次は, 対話シェルによるプログラムの実行例である。

```
$ konoha
...
>>> int fibo(int n) {
        if(n < 3) return 1;
        return fibo(n-1) + fibo(n-2);
    }
>>> fibo(1)
1
>>>
```

本論文では、KonohaScript の振る舞いを表記するため、対話モードによる実行結果を用いる。>>> は、式やステートメントを入力するプロンプトであり、その評価結果は次の行に表示される。

```
>>> fibo(10)      // 式の入力
55                // 式の評価結果
```

KonohaScript は、インタプリタ風の対話的実行を可能にしているが、内部では専用のバイトコードに変換し仮想マシンで実行している。このとき、コンパイラ型言語と同じく、型検査などエラーの検出を行う。しかし、コンパイルエラーにより、コード生成を停止することを行わず、エラーのない箇所はそのまま実行できる。つまり、コンパイルエラーによってまったく動作しないことはない、コンパイルできる部分は常に実行できる。

```
int readnum(int seq) {
  if(seq == 0) {
    ins = new InputStream("seq.txt");
    seq = ins.readnum(); // エラー
    ins.close();
  }
  else {
    seq += 1;
  }
  return seq;
}
```

上の例では、エラーが発生するブロックのみ実行時例外に書き換えて、エラーが発生しないブロック(つまり seq が 0 でないとき)はいつも通り動作する。

```
>>> readnum(1)
2
>>> readnum(0)
* Script!! 例外
```

同時に、コンパイラの機能を用いて実行することなしにスクリプトの検証ができる。これらの特徴を用いることで、スクリプト言語からコンパイル言語への橋渡しがスムーズになるように設計されている。

```
$ konoha -c readnum.k
- (readnum.k:8) (info) suppose ins has ty,
InputStream
- (readnum.k:9) (error) undefined method:
InputStream.readnum
```

5 実践と経験

我々は、大学学部におけるプログラミング教育において実際に教育活動に用いながら、KonohaScript の言語設計を行ってきた。以下、実際に KonohaScript を用いた授業の一覧である。

- 高大連携教育プログラム (2008 年) 高校生を対象としたプログラミングの紹介講義

- プログラミング演習 (2009 年) 20 名程度の小クラスの演習形式の講義。プログラミング未経験者を対象に関数からオブジェクト指向まで学習

- アルゴリズムとデータ構造 (2011 年) 80 名程度の大クラス向けの講義形式の授業。

5.1 手続きの理解と試行錯誤

高等学校で数学を学んだ学生は、プログラミングは手続きであることを最初に理解しなければならない。この違いは、端的に言えば、 $i = i + 1$ のような式は高校数学では成り立たないが、プログラミングでは手続きとして自然な記述であるという点に尽きる。この違いは、概念的に区別するのは難しく、やはりプログラムの実行をとまらねて体得することが一番わかりやすい。

KonohaScript の対話的プログラミング環境を用いれば、1 ステップずつ式や文を実行し、その結果を確認することができる。これにより、学生はプログラムとその振る舞いを対応付けて理解しやすい効果がある。また、授業中に入力してすぐに動作結果が得られ、注意力が散漫となる複雑な IDE やコンパイラ機能が途中で必要としない。

```
>>> int n = 0
>>> n = n + 1
1
>>> n = n + 1
2
```

プログラミングは、必ずしも最初のコーディングで期待する結果が得られるものではない。必ず、試行錯誤を繰り返しながら答えに近づいてゆくことになる。KonohaScript では、関数を再定義することも可能であり、予想と異なる結果をみて直すことができる。

```
>>> int fact(int n) {
      if(n == 0) return n;
      return fact(n-1) * n;
    }
>>> fact(5)
0 // おかしい!!
>>> int fact(int n) {
      if(n == 1) return 1;
      return fact(n-1) * n;
    }
>>> fact(5)
120
```

5.2 実例：バブルソート

ここでは、アルゴリズムとデータ構造における簡単な講義例を紹介する。

バブルソートは、近辺の値を比較して順序を入れ替えるソートアルゴリズムである。プログラム自体は単純であるが、最終的に正しくソートされることがわかりにくい。KonohaScript を用いれば、アル

Q1. プログラミングは好きですか？

YES 28 名 NO 13 名

**Q2. 将来、プログラミングは必要となると思いま
すか？**

YES 27 名 NO 1 名 わからない 14 名

Q3. KonohaScript による演習課題への参加

参加 37 名 不参加 5 名

表 4: アンケート結果

ゴリゾムの一部を切り出して、振る舞いを見せることができる。

次の例は、バブルソートの 1 回分を bubble 関数として切り出したものである。配列の要素 0 に着目すれば、bubble を 1 回実行するたびに、泡のように上っていく様子がわかる。

```
>>> int[] bubble(int[] a) {
    int i;
    for(i = 0; i < |a| - 1; i++) {
        if(a[i] > a[i+1]) {
            swap(a, i, i+1);
        }
    }
    return a;
}

>>> a
[5, 3, 1, 9, 8, 7, 4, 0, 2, 6]
>>> bubble(a)
[3, 1, 5, 8, 7, 4, 0, 2, 6, 9]
>>> bubble(a)
[3, 1, 5, 8, 0, 4, 2, 6, 7, 9]
>>> bubble(a)
[1, 3, 5, 0, 4, 2, 6, 7, 8, 9]
>>> bubble(a)
[1, 3, 0, 4, 2, 5, 6, 7, 8, 9]
>>> bubble(a)
[1, 0, 3, 2, 4, 5, 6, 7, 8, 9]
>>> bubble(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> bubble(a) // これ以上、必要なし
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

なお、この状態ではバブルソートは完成していない。授業中に学生にプログラムを完成させる宿題を出すこともできるし、教員がプロジェクトに投影しながら完成させることもできる。

5.3 学生からの評価

最後に、「アルゴリズムとデータ構造」を履修した学生からの評価を報告したい。主なアンケート項目とその結果は、表 4 にまとめている。以下、自由形式のコメントとあわせた考察である。

- **Q1: プログラミングが好きか？** 教員の認識に比べ、プログラミングが好きな学生の比率が高い。代表的な好きな理由は動いたときの感動。嫌いな理由は、思いどおり動かないことで気

が滅入るなど。本来、授業前と授業後で比較できるようにアンケートを取るべきであったが、KonohaScript による演習で好きになったとの回答もあった。

- **Q2: 学習動機** プログラミングが嫌いと回答した学生のうち、3 名は書けるようになりたいと考えている。それ以外の 10 名は積極的にプログラミングに関わりたくないと考えていた。
- **Q3: KonohaScript の評価** 今回は、KonohaScript による課題提出を任意としたが、多くの学生が KonohaScript による課題に挑戦したことがわかる。自由形式の回答からも、KonohaScript への否定的な意見はなかった。(一般に主流言語以外で授業を行うと、C 言語や Java で授業をやってほしいという要望がくることが少なからずある。)

6 関連研究

多くのプログラミング言語が、教育用途として開発が行われてきた。初學者、特に子供向けのプログラミング教育用として代表的な言語だけでも、LOGO 言語に始まり、Alan Kay らによる Squeak[2]、MIT メディアラボの Scratch[3]、Microsoft Xbox Kodu[4] などがある。ただし、これらの言語は低年齢層へのプログラミング教育を前提に設計されており、高等教育のカリキュラムの一部として利用しにくい。本研究では、C/C++、Java をベースとしたカリキュラムで用いることを目標に新しい言語の提案を行った。

7 むすびに

現在、プログラミング言語の入門として C 言語や Java が採用されるケースが多い。これは情報カリキュラム上、避けられない言語選択と言えるが、初學者には難しいという課題が残っている。我々は、C 言語や Java と互換性の高い文法をもつスクリプト言語 KonohaScript の開発を行い、スクリプト言語の対話性を活かした教育活動を行っている。プログラミング演習やアルゴリズムの講義経験を通し、より限られた単位時間数の中でより多くのプログラミング経験がたまることが示されてきた。

我々は、オープンソースプロジェクトとして KonohaScript の開発を行っている。今後は、Konoha を用いた教材開発に関心をもっていただける方々を広く募り、教育経験を活かした言語設計へのフィードバック、プログラミングの教材づくりから教育方法まで広く議論していきたいと考えている。

謝辞 Konoha 言語は、IPA 未踏ソフトウェア創造事業「軽量オントロジレポジトリの開発」に始

まり、横浜国立大学学長裁量経費による米ジョージア工科大学在外研究、総務省 SCOPE-R 若手先端 IT 研究者育成型研究開発「意味型を備えたユビキタスバーチャルマシンの開発」科学技術振興機構 JST/CREST「実用化を目指したディペンダブル組込みオペレーティングシステム領域」等の研究補助を受けて開発を行ってきました。

参考文献

- [1] Kuramitsu, K.: Konoha: implementing a static scripting language with dynamic behaviors, *Workshop on Self-Sustaining Systems*, S3 '10, ACM, p p. 21–29 (2010).
- [2] Ingalls, D., Kaehler, T., Maloney, J. and Wallace, S. and Alan K.,: Back to the future: the story of Squeak, a practical Smalltalk written in itself, *SIGPLAN Notice*, 32(10), pp. 318–326. (1997)
- [3] Cheng, J. and Randall, A. Z. and Sweredoski, M. J. and Baldi, P.: SCRATCH: a protein structure and structural feature prediction server, *Nucleic Acids Research*, 33(2) pp. 72–76, 2
- [4] Stolee, Kathryn T. and Fristoe, Teale, Expressing computer science concepts through Kodu game lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pp. 99–104, (2011).