

入門用プログラミング言語としての Scala — 教程と環境 —

Scala as the Introductory Programming Language: Learning Environment

平岡 信之*

Nobuyuki HIRAOKA

1. はじめに

筆者は、プログラミング入門教育に携わっており、先の前論文[1]で報告したように入門教育に適したプログラミング言語に関する調査を行い、今般Scala言語[scala-lang.org, 2]を採用して教材準備を開始した。その前提条件として、実習授業で扱うべき事項の取舍選択についても考察を行った。また、実習を行なうための具体的な言語処理系とその周辺のソフトウェアについて調査を行った。ソフトウェアウェアの選定はネット上に公開されている情報及びそれに関するディスカッションを参考に行い、実習を行なうに際しての障害や困難を避けるべく運用上の問題点を洗い出すため、実機上で環境整備を行いその動作テストを行った。Scalaは比較的新しい言語であり、入門教育での活用例として参考になる事例は乏しいが、主に子供を対象としたインドでの事例[wiki.kogics.net/sf:kalpana-center]を参考にしつつ、日本での大学教育及び筆者の置かれた環境に適用可能な方法論と道具を模索した。なお、授業は文系の情報系学部の大学1年生を対象とし、Windowsベースの個人所有のノートPCを用いて、使用するソフトウェアは無料のものに限定し、2時限/週×半期のコースであることを前提とした。本論文では言語の決定、教材準備、環境整備にあたって調査、或

いは考察した事項について報告し、新たな教程開発のケーススタディの1つとして、同じ志を持って情報教育を行う方々の参考に供したい。

2. Scala 言語について

2.1 Scala を使う理由

2.1.1 関数型パラダイムに向けて

ハードウェア技術が、曲がり角に来ている。スーパーコンピュータ、画像処理等の特殊な分野を除いて、20世紀のコンピュータは単一プロセッサの性能向上により発展を続けていた。しかし近年のMPUのクロック周波数はGHz単位となり技術的限界に近づいており、現在は、マルチコア化によりその限界の克服が図られている。コア数が2、4、8といった低並列度のアーキテクチャでは、プロセスのスケジューリングをOSが適切に行うことでマルチコアの効果は得られるが、高並列度のコンピュータで性能を低下させないためには、アプリケーションそのものを並列対応に作る必要があるになってくる。プログラムの適切な並列化は今後のコンパイラ技術に負うとしても、そもそも互いに依存関係のない処理に順序をつけて逐次的に書くことを強要してきた従来のプログラミング言語では処理の並行性が記述出来ず、並列時代に対応したソフトウェアが作れない。その問題を

*企業情報学部准教授

打破するためには、

a 既存の言語に並行性を記述する言語要素を付加する

b 新たな言語体系を作る

という2つの方策が考えられる。このうち a については、ある程度粒度の高い並行性の記述のためにThreadやActorといった言語要素が既存言語でもすでに使われているが、付加的言語要素で粒度の低い並行性記述を容易にした例はまだ見かけない。b の方向としては、関数型言語が有望だと考えられる。例えば純関数型言語であるHaskell言語[haskell.org]では、関数は原則として副作用がなく実行順序も問われない。ということは処理系が勝手に複数プロセッサに割り当てても結果に影響しない。副作用を伴う処理についてはモナドとして特別扱いすることによりそれに関連する処理については実行順序を維持する、という方針を取ることができる。昨今のHaskellの認知度の向上は、原則逐次から原則並行へのパラダイム転換の予兆だと考えられる。

しかし、プログラミング言語が進化しただけでは、プログラミングは変化しない。プログラムは人が書くものであり、先進的な言語を使ってもプログラマが旧来の考え方でプログラムを書いているという場面は頻繁にある（これは筆者がCやJavaの経験者にRubyを教える授業でもよく経験している）し、そもそも新しい言語をプログラマが（例えば難しいなどの理由で）選択しなければ、話は進まない。そうすると、教育の場で現時点で行うべき適切なことは、（古い考え方を教えるのではないことは言うまでもないこととして）このあと何年かは使う機会がないかも知れない先進的な言語を教えることよりは、継続的に使う機会の多い言語で、関数的に書く考え方（「関数脳」と呼ばれることもある）を身につけさせること、だと考え、関数的に書きやすい言語を選ぶことが肝要だということになる。いわば、逐次型から関数型への橋渡しのできる言語が必要になるということである。

2.1.2 型システムの意味

型システムについて、前論文の段階では、入門者が型制約との遭遇をいつどんな形で行うべきかを明確にせず、言語の選択の最終判断も保留し

ていたが、ここであらためて、型システムの功罪について整理しておこう。

例えば動的型付け言語の1つであるRuby言語は、型に関して言えば以下のような理由で好まれている。

a 手軽に開発ができ、記述もコンパクト。

b ダックタイピングにより拡張が容易。

一方、静的型付け言語は、次のような短所（上記の裏返しとして）と長所があると考えられている。

a' 記述量が増える（書くときも読むときも負担になる）。

b' 拡張の際に型の再設計が負担になる。

c コンパイル時（昨今は入力しながらオンザフライでも行うが）の型チェックにより実行時エラーを最小限にできる。

d 実行時のオーバーヘッドが減らせる。

これに対して、Scalaや、型システムについてそのお手本となったHaskellについて調査すると以下のことがわかってくる。

a'' 型推論機構を適切に活用すれば記述量はさほど増えない。現実には型推論機構がどう働くかを（それぞれの箇所において有効か否かも含めて）正確に理解し予測するのは困難であるが、略式の記法で書いてみてエラーが出たら型名の指定を後から挿入するというようなスタンスでそれなりに心地よくコーディングが可能である。

b'' Scalaでは構造的部分型（Haskellでは代数データ型がそれに相当する）によりダックタイピングに代わることは実現できる

さらに、Scalaを実際に使用した経験から、以下の点が重要な事項として判明した。

e コーディングの際に入力・編集システム（REPLやIDE）が変数や関数の型を認識していることにより、その型に応じて候補を絞り込んだ補完が可能になり、コーディングの負担は大きく軽減できる。

実際には動的型付の言語でも、プログラマはその変数や関数の型を意識しながら設計やコーディングを行っており、読者（自分自身も含めて）のためにコメントとして付記することを習慣としているプログラマも多い。ならばそれを一歩進めて、型情報を適切に活用できるよう処理系に伝えるよ

うな文法を持っているほうが、処理系が処理の効率化や安全化のために活躍できる可能性を増やすことができる。

なお、ここで処理系と呼ぶものは、狭義にはコンパイラのことだと認識されている。コンパイラの元来の役割は、ソースコードに書かれた手順を、ターゲットマシンの命令列に変換することである。型システムを持つ言語のコンパイラは、それに加えて、型情報を最大限に活用してプログラムの正当性の検査を行い、起こりうるエラーを予測して知らせてくれる。それに加えて昨今では、開発支援環境が言語の文法やライブラリの構成を認識していてIDEという形でプログラマの負担を軽減する仕組みが整っており、これも含めて広義の言語処理系と見なして語るができるだろう。このとき、ソースコードの内容は、読者（人間）に向けたもの、処理系に向けたもの、ターゲットマシンに向けたもの（インストラクションに変換されるもの）と分類するならば前2者へのメッセージの比重が高まってきているのが昨今のプログラミングの動向だと考えられる。

2.1.3 Scalaに何を託すか

この先、どんな言語が主流になるか、未来のことは明確には見当がつかないが、プログラミング教育の姿を将来のプログラミング・パラダイムに対応できるよう橋渡しをしていくことが2012年の時点での教育者の役割であると考えたときに、以下の2点を重点的テーマとし、

- a 「関数脳」を育てる
- b ターゲットマシンとの対話から、処理系との対話へ

その実現の道具としてScalaを選んだ、というのが結論である。

2.2 Scala 言語の特徴

すでに述べたことの再掲になるが、Scala言語の特徴として、選択の際に決め手となった事項をここに整理しておく。

- a JVM (Java仮想マシン) 上で動作するバイトコードコンパイラ型言語 (JVM言語と呼ばれる) である。Java言語との親和性が考

慮されて作られており、Javaで、もしくはJava用に作られた既存ソフトウェア資産を利用することができる。教育プログラマー職場プロジェクト、のそれぞれの間に発生し得るネットワーク外部性の影響が良質の言語の普及を妨げることはありがちな話であるが、Javaとの親和性はその影響を大幅に軽減し言語の発展性にはプラスとなる。

- b Scala式の即時実行が可能なREPLを備える。入門時の学習困難を軽減するのに1ステップずつ実行し確認できることは重要である。
- c 純粋な関数型言語ではなく、再代入可能な変数もfor によるループ (Javaでいうforeach型のループ) も用意されている。これを積極的に使おうというのではないが、旧パラダイムから新パラダイムへの橋渡しとして位置づけることはできる。
- d 静的型付け言語である。これは前項で議論した通り。ただし実際には多くの場合で型指定を省略して処理系任せできるので初心者への負担は少ない。変数は宣言と同時に初期値 (val変数では最初で最後の値) の設定を行うのがScalaでは普通だが、この使い方をしていく限りは変数の型指定は不要である。高階関数に無名関数を渡す場合もその双方の間で型情報が伝播されるのでプログラマは指定する必要がない。入門者が扱う範囲の技術で (クラス階層の設計を始めるまでの段階で) 型指定を求められるのはdefで関数/メソッドを定義する場面だけだと言っていい。この場面では、

```
def twice(n: Int): Int = 式
```

というような雛形を見せて、関数定義とはこういうものだと覚える、という伝え方ができそうな範囲のオーバーヘッドだと考えられる。

一方、Scala言語の不便な点や学習者を混乱させがちな点もある。

- e 代入演算子が返す値は Unit型 (他言語でのvoidのようなもの)。これは、副作用のある関数はUnitを返すというScalaの原則に沿ったものであるが、代入した値をさらに別の変数

に代入したり数式の途中で得られる値を変数に代入しておくという使い方ができないのは、C、Ruby、Java等に慣れた感覚からは困惑の元になっており、また、プログラムを短く書く妨げにもなる。

- f プログラム内で `chdir()` に相当する操作 (カレントディレクトリの変更) ができない。これはJVMの制約である。シェルプログラミングの感覚でカレントディレクトリからの相対パスでファイルアクセスをするプログラム書法は避けるようにするしかない。
- g 省略記法や代替記法 (シンタックスシュガー) が数多く用意されている。これは書き手の自由度を高めるコード量を低減することができる便利な仕様である反面、学習者には混乱を増やす両刃の剣であると言える。書き手としては1つの書き方だけを覚えて使っていけばいい訳だが、プログラミングを学ぶ過程では読み手として既存のプログラムを読むことは重要であり、既存の (他人の書いた) プログラムを読む時に知らない文法に困惑するという状況を避けるためには、ある程度記法のバリエーションの概要は知っておく必要があるだろう。

3. 教程の概要

3.1 箱の中から外へ

Scalaはクラスベースのオブジェクト指向言語でると同時にスクリプト言語として使える言語でもある。この特徴を活用して、前半はスクリプトを使える環境で小規模なトイプログラムから始めて関数型のスクリプト言語としてのプログラミング感覚を習得し、後半は独立したプログラムとして動作させられる、オブジェクト指向型のプログラミングへ発展させていく。

3.2 扱う事項と遠ざけたい事項

Scalaでは 変数には `val`型と`var`型があり、`val`型は再代入が禁止されている。ワード単位のデータを主に扱う逐次型言語 (アセンブラを含む) では `var` 型に相当する変数を使って、その変数の値を次々に更新していくことでアルゴリズムを構築

していく、それがプログラミングであった。本課程ではその作り方を極力排除し、変数の使用は、特に技術的に避けられないケースを除いて `val`型に限定する。それにとまって、制御構造も、`while`型や`until`型のループは扱わないこととし、

- a 関数の呼び出し
- b 再帰
- c `for` による繰り返し
- d `if` による分岐

を扱う。また、関数脳の涵養のために、これに加えて、

- e `for~yield`による 内包表記
- f 関数的に用いる `if` 式
- g パラメータとして渡す無名関数
- h それを受ける高階関数

も扱う。実は `case`式による分岐とパターンマッチングは洗練されたなプログラミングのための有効な武器であるが、時間の制約があるため初級コースでは扱わないこととした。

前章で述べたように、Scalaが持つ記法のバリエーションについてもなるべく実例を見せるようにしておきたい。例えば、

- i 関数の引数は括弧を省略できる
- j メソッドチェーンは演算子のように書くこともできる
- k 型指定は省略してもいい箇所が多い
- l 無名関数の記法もバリエーションがある

といったことを示しておく。後半のポイントとしては、

- m オブジェクト指向の考え方
- n クラス定義の仕方と使い方
- o 型制約の扱い (の初歩)

といったことがはずせない。また、最後に、現実との折り合い方を知るという目的で、

- p ツールを使いこなすこと
 - ・ビルドツール
 - ・IDE

- q 様々な環境での動作
- r Java言語との関係

についても授業で扱っておくべきだろう。

4. 入門環境

4.1 利用可能な選択肢

プログラミングの入門環境の伝統的な姿として、グラフィクス付きインタプリタが古くから使われてきている。古くはBasicやLogoがそうであったし、最近ではProcessing[processing.org]、Squeak [squeak.org]などがそれに近い思想で作られ、広く活用されている。Scala言語を使う場合、初心者にとって敷居の低い入門環境として、調査した範囲では3つの選択肢があった。以下にコメントとともに並べる。

a ScalaのREPL

Read-Eval-Print Loop を実現するプログラムで、Scalaの式を入力するとその式を評価(eval)し、その結果の型と値を印字(print)して返す、その動作を無限に繰り返す。複数行の式にも対応し、また、入力中の言語要素の文脈や型を認識し、それに基づいて補完もする。事前にグラフィクス環境を生成するクラス群を用意し、REPL起動時に読み込ませておけば、グラフィクス付きインタプリタは実現できる。ただしこの目的のクラス群の実現例は見つけれない。

b Spde[spde.technically.us/Spde.html]

Processing言語と同等のものをScalaでも実現できるよう作られたソフトウェアである。Spdeではリスト4-1のような枠組みで2つのメソッドを再定義することでプログラムを実装する。Processingによるプログラミング入門は和

書も含めて書籍が発行されており[3]、方法論としてある程度確立していると考えられる。ただ、PDE(Processing Design Environment)が、IDE的な機能(エディタースクリーン 文字出力エリア等)を備えているのに対して、Spdeでは完結したプログラムを、sbt(後述)でビルドした上で実行しグラフィクス窓に表示する機能のみ実現されており、入門教育で活用するためには単独では扱づらい。また、ビルドしてからの実行であることからREPLからの起動も容易ではなく、IDEと組み合わせる使うことが前提となるだろう。

c Kojo[kogics.net/sf:Kojo]

プログラミングと数学を学ぶための環境として開発されたもの。オープンソースソフトウェアとして公開されており、Windows版バイナリでも配布されている。Kojoの開発者にあたって啓発を受けた先発ソフトウェアとしてLogoやProcessingの名前が挙げられている。一見するとKojoは「タートルグラフィクス環境のScala版」だと認識できる。タートルグラフィクスだけでは次のステップに繋げていく際に敷居が高くなる可能性があるが、詳しく調査してみた結果、実際には、XY座標による描画や、描画された図形をオブジェクトとして扱う機能も実装されており、活用度が高いことがわかった。そこで、当面の入門コースの前半課程ではこのKojoを活用しいわば伝統的なタートルグラフィクス環境で進めて行くこととした。

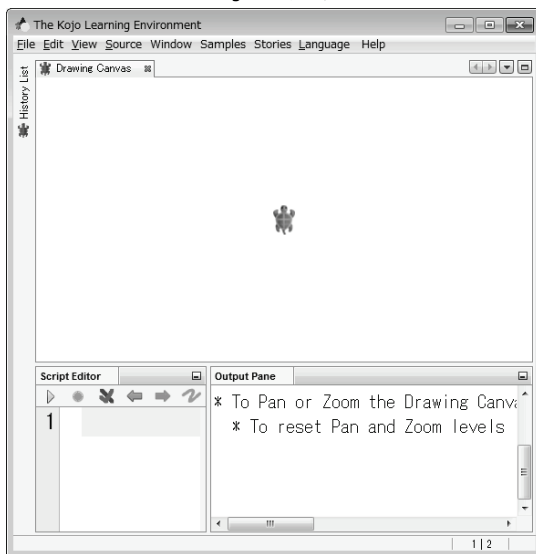
// リスト4-1 spdeでのプログラム例

```
class pde_sample extends processing.core.PApplet {  
  override def setup() {  
    // 初期化  
  }  
  override def draw() {  
    // 描画処理  
  }  
}
```

4.2 Kojo の特徴

起動直後のKojoの画面は図4-2に見るように、Drawing Canvas、Script Editor、Output Paneの3つのタイリングウィンドウ (paneとも呼ぶ) から成っており、その構成からその主要機能を推測することができる。すなわち、Scalaスクリプトをエディタペインで入力編集し、即時実行するのが基本機能で、Outputペインにはエラーメッセージ等が表示されるだけでなくコンソールとして入出力に用いることができる。初期画面でキャンバス中央に亀 (タートル) が表示されており、プログラミング学習はこの亀との対話から開始するよう仕組みられている。エディタ上端に並ぶツールバーに置かれたアイコンの個数は一桁で、その左端にある実行ボタン (AV機器のPlayボタンを擬えたオーソドックスな右向三角) に気づかせて、forwaad(数値) 等の関数を1つ2つ提示することで、学習者はKojoをとりあえず使い始めることができる。少ない努力で最初の一步を踏み出すには適した環境である。さらに以下のような機能を持つ、かなり盛り沢山のソフトウェアである。

図4-2 Kojo の起動画面



a IDEの代用機能：

エディタは、言語固有の支援機能を組み込んだIDEと同様に、変数名・関数名・仮引数列などの補完、ソースの整形、括弧類の対応関係の

提示などを行う。補完の基本操作 (Ctrl+SPACE、一般的なIDEと同じ) は起動時にOutput Paneに英語で書かれるメッセージの中で示されている。複数のソースに渡るプログラムのビルド・実行や、ワークスペースの切り替え機能はないが、実行したスクリプトは履歴として自動的にファイルに保管されて随時遡ることができるので、Kojoを一旦終了しても学習は継続的に行える。

b REPLの代用機能：

エディタ窓でコードの一部を選択反転した上で実行ボタンを押すと、その選択した部分を式として評価して結果の型と値をOutput Paneに印字する。即ち、コマンドプロンプトでREPL (Scalaコマンド) を利用するのが概ね同等のことができる。ただし、コマンドプロンプト上では、ファイル名を入力する際にその代用手段として、エクスプローラ等からファイルをドラッグしてコマンドプロンプト上でドロップするという手段が使えるがKojoの画面でその手段は使えない。

c TurtleモードとStagingモードの並立：

亀の前進後退(forward, back)と、方向転換(left,right)を基本操作とするタートルグラフィックスは、最初の一步として効果は高いが、その組み合わせだけで何かを作ろうとするとプログラミング技術習得としてはその効果が心もとない。Kojoではturtleモードとは別にStagingモードがあり、ここでは、xy座標を指定した図形が描画でき、さらにその描画したオブジェクトは後から位置変更・サイズ変更などを行なうことができる公開メソッドが用意されており、ゲーム感覚のプログラム作成が用意になる。なおこの分野については基本図形を組み合わせでより複雑な合成図形を作る、という操作を階層的に重ねて行って組み合わせでより大きな図形にしていくという手段も用意されている。ただしこれはKojo活用として先進的すぎて初心者には難しいと考えられるので、初級授業ではTurtleとStagingだけ使う。

d ZUI：

ZUI (Zoomable User Interfaces) を実装するライブラリの1つとしてPiccolo[www.cs.umd.

edu/hcil/jazz/] を使って拡大に耐える描画オブジェクトが使われている。そのため、大幅な拡大（ズームング）をして表示されても、画素の大きさも大きくなって図が荒れることがない。

e ガイド：

Kojoの機能とScala言語を使って、Kojo画面でハイパーテキスト的な動作をするプレゼンテーションを作成しデモすることができる。この機能を用いて、Kojoの使い方のガイドも作られている。独習環境としても使えるシステムとなっている。

4.3 Kojoの問題点と今後の方策案

こうしてKojoは入門教育の重要なツールとなったが、一方で、問題点がないわけではない。現時点で把握している事項を挙げておく。

- a 日本語が使えない。メニューの言語として英語以外にスウェーデン語が選択できるが日本語は用意されていない。
- b グラフィックス系は関数型言語として実装されている。図形表示を前述のPiccoloが支えている訳だがそのPiccoloのオブジェクトはソースから直接使うことができないケースが多く、オブジェクトを生成したりプロパティにアクセスしたりする関数でラップされている。
- c Swingとの併用の問題：Kojoの亀の動作と次章に見るようなSwingベースのGUIを共存させると、結果的に並行して動作する2つ(以上)のスレッド間での適切な同期が崩れてしまうらしく、プログラムの実行がKojoごと停止してしまうケースが発生することがある。
- d タートルとStagingのライブラリ構成が中途半端タートル用のプログラムとStaging用のプログラムは、その要素を相互乗り入れ的に利用することが出来るが、機能の重複があったりする反面、例えばゲーム的なアプリケーションをKojoで作りたいと考えたときに用意されていないクラスやメソッドがあり、中途半端に感じることになる。

4.4 Kojoを活用した課程の概要

以下のような題材を用意し、順々にプログラミング技術を習得して行く。

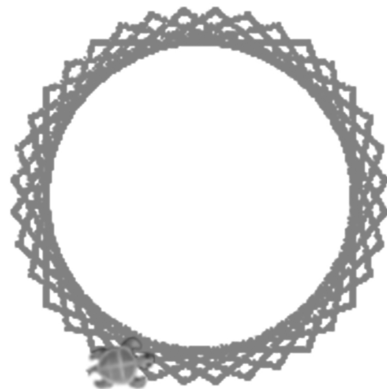
4.4.1 タートルグラフィックスの初歩

forward(steps:Int):Unit と left():Unit または left(angle:Int):Unit (rightも同様) を組み合わせるだけで、様々な図形を亀に描かせることができる。さらに、それに repeat(times:Int) (=>Unit):Unit を組み合わせ、表現力を増やすことができる。リスト4.4.1と図4.4.1はそのレベルでのプログラム例を示す。

```
// リスト 4.4.1 円のような形を描く

repeat(400) {
  forward(100)
  left(70)
}
```

図 4.4.1 円のような形



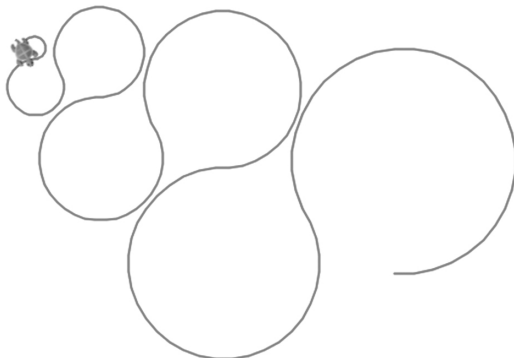
4.4.2 関数と再帰

関数の定義をここで学ぶ。関数を組み合わせてより多彩な図形を描く。その後、再帰の使い方を学ぶ。再帰は制御構造の1つと考えてもいいが、再帰が無限に続かずに適切に終了するために、最低限1つの引数があり、呼び出し毎に少しずつ一定方向に変化していき、それが限界値を超えたときにはそれ以上の再帰呼び出しをしない、という論理でプログラムを作る。再帰を使ってパラメータを少しずつ変化させていくプログラムの例はリスト4.4.2と図4.4.2に示す。

// リスト4-4-2 パラメータを漸減（と反転）しながらの再帰

```
def arc(len: Int, dir: Int) {
  repeat(30) {
    forward(len); left(10*dir)
  }
}
def arcs(len: Int, dir: Int) {
  if(len>1) {
    arc(len, dir)
    arcs(len-3, -dir)
  }
}
clear; right; arcs(20, 1)
```

図4-4-2 パラメータを漸減（と反転）しながらの再帰の例



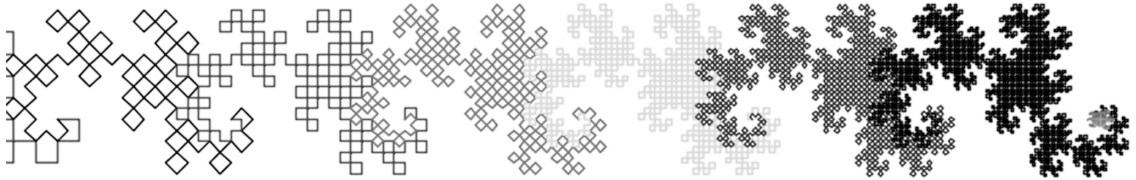
4.4.3 フラクタル

再帰の発展として、フラクタルの分野で古くから知られている図形の描画方法を紹介する。ここでは、少しのプログラムで亀に複雑な動作をさせることができることを見せ、プログラミングの可能性の深さを実感させることが狙いである。リスト4.4.3と図4.4.3にその例を示す。

// リスト 4-4-3 フラクタル関数の描画の例 ドラゴン曲線

```
val mlen=200
def dragon(level: Int) {
  val len=mlen.toDouble/
    Math.pow(2, level.toDouble/2)
  def dragon_(level: Int, dr: Int) {
    //println(level+"", "+dr)
    if(level<=0)
      forward(len)
    else {
      turn(45*dr)
      dragon_(level-1, 1)
      turn(-90*dr)
      dragon_(level-1, -1)
      turn(45*dr)
    }
  }
  dragon_(level, 1)
}
val colors=Array(green, blue, black, brown, red)
clear ; right
for(i<=0 to 12) {
  setPenColor(colors(i%colors.size))
  dragon(i)
}
```


図 4-4-3 ドラゴン曲線



4.4.4 並行動作

`newTurtle(x:Int, y:Int):Turtle` で亀を複数生成し、`runInBackground(=>Unit):Unit` で並行動作させる。プログラム例を 4.4.4 に示す (動きのあるプログラムなので図は省略する)。

**// リスト 4-4-4
複数の亀に並行動作をさせる例**

```
def tCirc(t:Turtle) {
  repeat(60) {
    t.forward(10)
    t.left(6)
  }
}
clear ; turtle0.invisible
val turtles=for(i <- 1 to 10)
  yield newTurtle(i*30, 0)
for(tt<-turtles) {
  runInBackground{ tCirc(tt) }
}
```

4.4.5 XY 座標とアニメーション

ここでは `Staging` モードを使う。 `Staging` では様々な描画オブジェクトが用意されている。ここでは再帰を用いずに `for` によるループを使い、複数の描画オブジェクトを並べたり、そのサイズや色を徐々に変化させたりすることを練習する。また乱数を紹介し、サイコロ程度のアプリケーションと、乱数を使ってオブジェクトの色をランダムに変化させることも紹介する。さらに、グラデーションも紹介する。 `Kojo` の `animation(=>Unit):Unit` という制御構造 (無限の繰り返しだがその各回毎に短い休止時間が入る) と、描画オブジェクトに対する変化を引き起こすメソッドを使い、簡単なアニメーションを作成する。リスト 4.4.5 にその例を示す。

**// リスト 4-4-5
簡単なアニメーションの例**

```
import Staging._ ; clear ; turtle0.invisible()
val c=circle(0, 0, 100)
animate{
  c.translate(1, 0)
  if(c.position.x>300)
    c.setPosition(-300, 0)
}
```

5. 実用的プログラミングへの課程

プログラミング入門の後半過程は、実用的プログラミング技術を習得させる (或いは習得しようという気持ちになるよう仕向ける) ための仕掛けとして、実用技術の紹介、周辺ツールの紹介、様々な動作環境の紹介、という狙いで幅広い経験をさせつつ、その過程でコーディング技術にも慣れてもらおうと考えている。そのため、詳細な技術や長期的に継続して利用することは考慮せず、様々なプログラミングの世界を少ない努力で経験でき

ることに重点を置いて以下の各項目について検討を行う。

5.1 ファイル入出力

ScalaではJavaでの資産をそのまま使えるので、入出力に使えるライブラリは多数ある。その中で、入力に関して言えばScala標準の方法として `scala.io.Source` クラスを用いる方法が初学者にも使いやすい。(ただし、`scala` パッケージは標準で読み込まれるため、コーディング時にはサブパッケージ名から書き始めることが多いので、本稿でも適宜省略して書く。なお、一般的には `predefine` されていないクラスは、複数回使われるならば予め `import` して使われる。)

`fromFile`(ファイル名)、または第二引数としてエンコーディングを指定することもできるこのメソッドが `Iterator` 型を返す。これを用いて、

```
val f=io.Source.fromFile(
    "someFileName")
```

のあと、

```
f.getLines().map ...
for(line<-f) ...
```

のように各行のデータを使うことができる。ファイル全体を一度に読み込ませるなら、

```
f getlines() mkString("\n")
```

のようにも書ける。このレベルのテキストファイルの入力の方法は最初の段階で伝えておく必要があるだろう。

一方、ファイル出力については、Javaの資産を借用して、`java.io.PrintWriter` クラスを使う方法が簡便だと思われる。

```
val out=new java.io.
    PrintWriter("filename")
try{ for(... ) out.println(e) }
finally {out.close}
```

のように使う。ただし、文字データの出力結果を、人に見せたりレポートとして提出したりする際には、画面上でコピー&ペーストするという方法も使えるので、ファイル出力の方は必ずしもこの段階で扱う必要はないかも知れない。

5.2 高階関数によるコレクション処理

このプログラミング手法は、LispやAPLの時代

から脈々と使われてきた技術を、近年になってから関数型言語の世界で型情報つきで洗練させたものである。同種のデータの集合体を扱うためのデータ構造を集めたパッケージが `scala.collection` で、それに含まれる代表的なクラスが、`Seq`、`Map` (連想配列もしくは `Key-Value` 対に相当)、`Set` (集合) である。`Seq` は 1 次元の順序構造を持つデータで、そのサブクラスとして `List` と `Vector` がある。さらにJavaの配列と対応付けられた特殊な `Seq` 型クラスとして `Array` がある。このうち、`Map` は次節の例で扱うとし、まずは `Array` もしくは `List` を用いて、データ列に対する処理の練習を行なうこととする。この両者はアクセスの効率など使い方によって性能的な差は発生するが、集合体としての使い方はほぼ同等なので入門時にはどちらを使っても大差はない(本稿では `Array` で説明する)。

`Array` の生成は `Array(1, 2, 3)` のように要素を括弧内に列挙するのが最も直感的に分かりやすい。`new Array(...)` ではなく `Array(...)` で生成しており、ここではScalaのルールに従って `Array` クラスの `Apply` メソッドを呼び出して生成しているのだが、その説明は入門時には省略してもいいだろう。他に、別のコレクション型データから `toArray` メソッドで変換したり、`Array(...)` 式の引数に `Array(someList: _*)` のように展開する指示記号 `_*` をつけて呼び出して変換する方法もある。下記の実習のためのデータはテキストファイルで用意しておき前項の方法で読み込んで `Array` にするというのが手軽である。

`Seq` 型のデータ列に対する処理を書く記法として、Scalaでは、`for` (または `for~yield`) を使う書き方と、`Seq` をレシーバするメソッド呼び出しとして高階関数を呼び出す書き方が用意されている(実装としては前者が後者のシンタックスシュガーである)。入門の初期に `for` を使った簡単な繰り返し(回数を定めたループ等)を扱っているが、この段階では両方の書き方を提示しつつ、前者から後者への発想の橋渡しを目指したい。ここで扱う高階関数としては、

- a `foreach` (各要素に対する処理を順次)
- b `map` (各要素に対する処理結果を新たな `Array` にして返す)
- c `filter` (条件を満たす要素のみを `Array` にし

て返す)

- d zip, zippered (複数のSeqの対応する位置にある要素動詞を組にして新しいArrayを作る)
- e reduce と fold (縮約・集約 それぞれ右結合のものと左結合のものがある)
- f 関数引数をとらない関数として、sorted (並び換え)、reverse (反転)、zipWithIndex (要素と添字をタプルにしたArrayを作る) など
- g sortBy (書く要素に与えられた処理を施した結果を比較して並び替え)


などを紹介することとする。なお、高階関数を呼ぶ際に、オブジェクトとメソッド名をドットでつないだメソッドチェーン的な記法と、スペースを挟んで並べることによりメソッド名を演算子 (後置演算子または2項演算子) のように扱う記法があり、一方、それに与える無名関数の書き方にもバリエーションがあり、例えば要素を2倍 (または文字列の場合は2つ並べる) するような処理の場合は

- 1) 仮引数に名前をつけて `{(e) => e*2}` または `{e => e*2}` (1つならば括弧は省略できる)
- 2) スペースホルダを使って `{*2}` と書けるため、書き方には様々なバリエーションが発生する。リスト5-2に実例としてプログラム断片を示す。

5.3 Web スクレイピング

スクレイピングのためのライブラリやそれを支えるHTMLパーサもパッケージとして公開されているものがあり活用することができるが (ちなみにXMLパーサは言語に標準で搭載されている)、ここではもう少し無骨なアプローチをとる。

前述の `scala.io.Source` クラスに `fromURL` メソッドがあり、

```
io.Source.fromURL("http://www2. 
nagano.ac.jp/hiraoka/")
```

のようにファイルとほぼ同等の方法でデータを読

// リスト 5-2 コレクション操作の実例

```
// ファイル words.text の各行に何かの文字列が入っていると
val a=io.Source.fromFile("words.text").getLines.toArray

// 各行の長さを Array にする
for(e<-a) yield a.length
a.map {(e)->e.length}
a.map {_.length}

// 空白を含む行を選別する
for(e<a if e.contains(" ")) yield e
a.filter {(e)=>e.contains(" ")}
a.filter {_.contains(" ")}

// 各行の長さの合計値を出す
a.map {_.length}.foldLeft(0) {(r, e)=>r+e}
a.reduceLeft {(x, y)=>x.length+y.length}

// 最も長い行を求める
a.sortBy {_.length}.last
a.reduceLeft {(x, y)=>if (x.length>y.length)x else y}
```

み出すことができる。これに対して、

```
getLines mkString
```

したものの即ちファイル全体を1つの文字列にしたものを `s:String` とする。

HTMLの文法の基礎 (`img` タグ) 正規表現の基礎 (文字クラスと量指定子) を教えた上で、タグを抽出する正規表現

```
val re="""(?:<img [^>]*src= [
"([^\"]+)""".r
```

を使い、

```
re.findAllIn(s)
```

でこの正規表現にマッチする文字列を `List` や `Array` として取り出すことができる。或は、

```
re.findAllIn(s).matchData.map{
  m=>m.group(1)}
```

のように `matchData` を使って、このページから引用されるすべての画像データ (の `URL`) を抽出することができる。Aタグの `HREF` パラメータにマッチする正規表現として、

```
val re2="""(?:<a [^>]*href= [
"([^\"]+)""".r
```

を使えば、ページ内のリンクを抽出できる。なお、`(?i)` は `case-insensitive` のマッチを指示するフラグである。

こうして正規表現を活用して、リンクとそのタグの内容である文字列を抽出して `Map` として保存したり、再帰的にリンク先を読みだして解析することでサイトマップを形成するなど、応用の方法は幾つか考えられるだろう。

5.4 GUI ツールキット

`Visual Basic`、`Smalltalk` (とその後継) を例外として、殆どの実用プログラミング言語は、内蔵もしくは添付のGUIキットを持たず、サードパーティの製品をユーザが選択して使う形になっている。Javaでは `AWT`、`Swing`、`SWT`、`GWT` などが使われて来ている。その中で `Swing` の認知度が高い。`Swing` に関しては、Scala用にAPIを再設計したラッパーが `scala.swing` パッケージとして用意されており、これを標準として使うことに問題はなさそうである。ちなみに `Kojo` もGUIの外枠は `scala.swing` で書かれている。

`Swing` を使ったGUI型プログラムを動作させる

方法は以下の2つがある。

a スクリプトとして動作させる：

この時は `scala.swing.Frame` オブジェクト (以下、本節では `ScalaSwing` 以外のキャピタライズされた名前は `scala.swing` パッケージのクラス名を表す) を1つ生成し、

```
val f=new Frame()
```

最小限の属性設定をすれば

```
f.visible=true
```

画面に最小限の窓が現れる。`REPL` や `Kojo` の中からはこの形で動作確認ができる。なお、`scala.swing` ではオブジェクトの属性設定 (等のメソッド呼び出し) は変数に記憶させておいて後からレシーバ、セクタの記法で呼び出す以外に、インスタンス生成式 (`new` クラス名) の後ろにブロックを置き、その中で呼び出すこともできる。このブロックの中ではレシーバは不要であり、このブロックはインスタンス生成直後に実行される。

```
new Frame {
  visible=true
}
```

b 独立したアプリケーション (を実現するクラス) として動作させる：

この時は、`SimpleSwingApplication` のサブクラスとしてアプリケーションを定義し、その中のメソッドとして `top` を定義しそこで上記の `Frame` もしくは `MainFrame` オブジェクトを生成させる。

```
object app extends
```

```
SimpleSwingApplication {
```

```
  def top = new MainFrame {
```

```
  }
```

```
}
```

こうして生成した `Frame` の中にウィジェット (コントロール部品) を配置することでGUIをデザインして行く。ただし、`Frame` の中には単一の部品しか置けないので、間に複数の部品を配置できるコンテナとして `Panel` 抽象クラスを具現化したオブジェクトを置き、その `Panel` 類の中に部品を配置する (もしくはさらに階層的にコンテナを差し挟んで複雑な構造を作ることにもできる)。この部品の包含関係を扱うメンバー

がComponent型 または List[Component]型のcontentsフィールドで、代入（前者）または追加（後者）により関係を設定する。

```
new Frame {
  contents = new FlowPanel {
    contents += new Button("left")
    contents += new Button("right")
  }
}
```

配置した部品がインタフェースとして機能するためには、その部品に対してユーザが行った操作（ボタンを押すなど）がボタン押下に対する応答（リアクション）を設定する。ここではeventを扱う（eventに対する応答をプログラムするため、まず、eventパッケージの全クラスをimportしておき、

```
import scala.swing.event._
```

Buttonオブジェクトの reactionsフィールドにブロックを追加する。

```
reactions += {
  case イベント => 動作
  case イベント => 動作
} // という文法で記述する
```

イベントの書き方としては2種類ある。

e : クラス名

eは適当な名前前のcase変数（何でもい）でイベントオブジェクトがこのeに（対応する動作の間だけ）設定される。例えばButtonオブジェクトの場合 対応する event クラス名は ButtonClickedが使われる。

```
ButtonClicked(src)
```

ここで、src は適当な名前前のcase変数（やはり何でもい）で、ここにイベントの発生源であるオブジェクトが設定される。

キャンバスに2次元図形を描画することは、Kojoの中でも実現出来ているので、Swing環境であらためて扱う必要はないだろうが、もし必要ならCanvasの機能を持つウィジェットとして、(java.awt.Canvasを使う例もよく見かけるが) Panel（を具現化したオブジェクト）を使うことができる。例えば

```
object p Extends swing.Panel {
  override def paintComponent(
    g: swing.Graphics2D) = {
    ...
  }
}
frame.contents = p
のように生成すればいい。
```

5.5 ビルドツール

昔のプログラマーには、（今の呼び方で言う）ビルドツールとしてはmake+Makefileが業界標準であったが、Java時代の昨今はMaven[maven.apache.org]とAnt[ant.apache.org]が主流となっている。これらは、パッケージ管理システムの機能も併せ持っており、リポジトリからビルドに必要なパッケージを見つけ出してダウンロードしてくれる。一方、Scalaに特化したビルドツールとしてSbt[scala-sbt.org]がある。WindowsではSbtは.batと.jarの2つのファイルから成っており、これをScala処理系のbinフォルダに置くことでインストールは完了する（後の節に補足あり）。Sbtに依存する周辺ツールも多くあるので、Scala処理系のインストールのついでに一緒に入れておくもの、という認識でいいだろう。想定するプログラムの規模がせいぜい数個のクラスであり特殊なライブラリを自発的に使用することを想定しない初心者範囲ならば、これで充分である。

前項のSwingの例で、スクリプトから独立アプリケーションへの展開を図ることができるが、独立アプリケーションをコマンドラインから起動する方法は、一般には

```
scalac ソースファイル名
```

```
scala (mainメソッドを持つ) クラス名
```

の順で呼ぶが、単純なソースツリーならば sbt run でも起動できる。もう少し規模の大きいプログラムを作成した時に、それを実行可能なjarファイルとして1つにまとめる操作を行なうという際にも Sbtの活用を経験する機会があるだろう。このとき、Sbt設定ファイルまたは mainClassとなるソース・ファイルに mainClassが何であるかを（前者の場合はパッケージ名からつないだクラス

名を、後者の場合は`import sbt._`した上で該当クラスの中に) 指示しておくが必要になる。

5.6 IDE

プログラミングを (Kojitoよりは本格的に) 行なうとしたらエディタが必要となる。原理的にはWindowsのおまけであるメモ帳でも編集は可能であるが、機能があまりに貧弱である。せめてエディタの中でScalaの入力支援をするものを探すとしたら、老舗の Emacs や vi (それぞれ派生も含めて) と Sublime Text、Aptana などがあげられる。しかし前者は初心者には敷居が高く、後者は日本語の問題などを抱えており、決定版となるエディタにはまだ遭遇できずにいる。使い方を教えるコストを考慮すると、IDEで作業するという選択が無難なようだ。

Scalaが使えるIDEとしてメジャーなもの (日本語対応もされ日本でも親しまれているもの) は Eclipse、Netbeans、IDEAの3つである (他に EclipseベースのType Safeなどもあるがまだ普及はしていない)。それぞれ、Javaを標準として作られているIDEなので、Scala用のプラグインの導入が必要となる。IDEは編集・実行・テストも含めて開発の様々なフェーズを支援する多機能なツールであるが、当面はエディタと実行ボタンだけを認識するだけでもいいだろう。コマンドでプログラムを起動する経験を一回したら、そのままIDEに移行することにしよう。これまでの経緯と、普及の度合いの両方の観点から、(他の2つとの詳細な比較はパスして) 仮にEclipseを選ぶこととするが、Eclipse+Scalaプラグインを起動して、Scalaでの開発がJavaと違うところは、最初にScalaプロジェクトを新規作成するところだけと考えていいだろう。このときパースペクティブもScalaパースペクティブになるが、特にJavaの場合と根本的な違いはないようである。

5.7 フレームワークの活用

ここではWebアプリケーション作成のためのフレームワークに限定して考える。Scalaで使えるフルスタックのフレームワークとして、現時点でメジャーなものは Lift [liftweb.net] と Play! [playframework.org]の2つである。

5.7.1 Lift

Lift はインストールして使い始めるまでは簡単な操作で実現できる。

ダウンロード、解凍展開、シェルで (またはWindowsの場合はコマンドプロンプトでという意味だが以下ではその注釈は省略する) そのディレクトリに移動して、

```
sbt updateで 必要なライブラリを自動インストールし、
```

```
sbt コンソールに入り
```

```
jetty-run で起動 (Webサーバが起動する)
```

```
jetty-stop でサーバ終了
```

ここまでは簡単であるが、アプリケーション作成の段階で、Mavenのコマンド`mvn`をパラメータを多数つけて呼び出す必要がある。また、動作のしくみがRails的なMVC分離構造でない独自の構造を持つため、入門で使うには若干敷居が高いと思われる。

5.7.2 Play!

Play! はバージョン2が最近公開され (Play2と呼ばれる)、Scala向けに洗練されたものになっている。ただ、2になってからMVCのMの部分 (モデル) が大きく変更されている。Play2を使う際にインストールは不要で、解凍したディレクトリ上で以下のシェル上の操作を行うことでアプリケーションを起動できる。ただし、カレントディレクトリ変更後も`play`コマンドが呼び出せるよう、適切なPATH変数の設定 (後述) は必要である。

```
play appName # これで appNameと同名のディレクトリが作られ
```

```
# その下にアプリケーションの雛形を含むツリーが作られる
```

```
cd appName
```

```
play run # または 単にplayコマンドでplayコンソールに入り、
```

```
# 設定等をした上で run サブコマンド
```

この状態で (Webフレームワークの一般的な姿として) 開発用Webサーバがシェルのサブプロセスとして起動しており、その起動時のメッセージにポート番号等の (localhost上での) URLが提示されるのでそのURLを用いて同一マシン上のブラウザからアプリケーションにアクセスできる。

Play2では、Railsと同様にMVCが分離したアプリケーションツリーを用いる。アプリケーションを作る際にチェックする必要があるファイルとして以下のものがある。

- a `conf/routes` : クライアントから呼び出されたリクエストと、それに呼応して呼び出すべきプログラム (一般にcontroller) とお対応関係を規定する
- b `app/controller/*.scala` : `routes` から呼び出されるメソッドを複数持っている、Controllerクラスを継承したクラスが定義されている。この中に多くのメソッドを定義する。各メソッドは Actionオブジェクトを返す形になっており、その中で、Responseオブジェクトが生成される。Responseオブジェクトに渡す値として下のcのファイルがパースされる。
- c `app/views/*.scala.html` : ここにあるファイルはPlay2独自のテンプレートエンジンの記法になっており、このファイルで、アクションを実行した後にクライアントの窓に表示される内容が規定される。
- d アクションの中で適宜 Model (データ・モデルの実態) やりとりを行なう。

Play2はRailsなどと違い Scaffold機能は添付しておらず、また、モデルの設計のためにSQLによるデータ定義が必要なので、このモデルの指定・構築は煩雑となる。初歩の段階では、データをサーバにストアしない程度の簡単なアプリケーションにとどめておくのが無難だろう。

5.8 タブレット端末を対象としたクロス開発

他のOSで、Windows環境から、実機をつないで検証することが容易なもの、と限定すればAndroidが最有力である。Androidアプリケーションを作る際にはAndroid SDKがインストールされている必要がある。最終的にはAndroid用実行形式である `.dex` を作った上で、それをパッケージングしてアプリケーションファイル `.apk` を作成するのだが、その作業は何らかのビルドツールに委ねることになる。

具体的な方法の1つは、Sbtにプラグイン [github.com/jberkel/android-plugin] を組み込む方法である。コーディングは勿論スクラッチから書いてもいいが、ひな形から始めるのが楽である。そのひな形を生成するシステムは別途調達が必要で、`android-app.g8` [github.com/jberkel/android-app.g8] が使えそうである。こういった手順はEclipse上でもなぞることができる。

6. 学生の環境整備

個人持ちのノートPCを使っての実習授業の場合、各受講者のPCに実習用ソフトウェアをインストールする必要がある。この過程を複雑にしていると授業の時間をサポートに浪費する結果になる。

6.1 標準的な方法

インストーラが添付されている、或いは単に解凍展開すればそのディレクトリ上で起動できるソフトウェアは、ダウンロードサイトまたは開発元のトップページのURLと、その中での道案内を大まかに伝えた上で「インストールしておいてね」と言えば通じる時代にほぼなっている。Scala処理系とKojoはインストーラがあり、CUIで使用するscalaコマンドについてはScalaインストーラがPATH設定まで行なうので特に考慮すべき事項はない。Sbtは.batと.jarの2ファイルなので、展開したものをそのまま、Scala処理系のbinディレクトリにコピーすることでPATHの設定に関する心配はなく使える。Eclipseについては、土台となるEclipse Classicを展開した上で、日本語で (を) 使うためのプラグイン群 [mergedoc.sourceforge.jp]、Scalaを使うためのプラグイン [scala-ide.org]、またEclipse上でAndroid開発を行なうならばADTプラグイン [<http://developer.android.com/tools/sdk/eclipse-adt.html>] を組み込む必要がある (その操作方法は個別に違うのでここでは省略する)、こういった作業をパソコン初心者に委ねるとすると指導が大変なので組込済のフォルダツリーをアーカイブ化したものを提供するのが適切だろう。

6.2 PATH 変数の設定

Sbtのような軽量のソフトについては、すでにPATHに入れてあるソフトウェアに便乗する戦略でいいが、Play2のようにCUI動作する大きなソフトウェアの場合、このプログラムが置かれた場所をPATH変数に組み込んだシェルを起動する必要がある。Windowsではユーザ環境変数とシステム環境変数を設定するためのGUI操作が広く知られてはいるが教室でその操作を実施することは煩雑であるとともに、様々なソフトを積極的に導入する学習者においては、ソフトウェア間の名前の衝突や環境変数の長さの制約によるトラブルも発生し得る。これを回避するためには、特定のパス名をパス変数の前に挿入した状態でコマンドプロンプトを起動するようなショートカットを用意するという方策が有効である。具体的には、ショートカットのプロパティを表示させたときに「リンク先」として表示されるプログラム（ここではcmd.exe）の引数として /k オプションの後にPATH変数を変更するsetコマンドを文字列として与えるというやり方である。各ユーザがPlay2（な

ど）をどこに展開するかを統一することは困難なので、カレントフォルダをPATHに挿入した上でこのショートカットを作成するようなリスト6.1のようなスクリプトを配布することにする。

なお、このスクリプトはVBScriptで書かれているが、拡張子は.vbsである。Scala言語に関して啓蒙する筈の論文の最後にレトロな言語によるハックが登場するのは皮肉なことであるが、JavaやScalaはこのようなOS配下のデバイス操作、特にWindows独自のものになると、その発展の経緯の影響もあるのだろうが、不得意分野になるようで、適切な道具立てを発見することができなかった。そこで、ここではWindows Scripting Host(WSH)を活用することとした。WSHで使える言語はVBScriptとJScriptがあるが、JScriptの標準的な拡張子がJavaScriptと同じ.jsで、JavaScript用に何かの設定をしているユーザの環境ではダブルクリックで動作できない可能性があるため(.vbsはそういう混乱を引き起こすケースは見受けられない)、ここではVBScriptで書いた。

リスト 6-1 crtShortcut.vbs

```
'
' カレントフォルダを PATH に挿入した上で
' コマンドプロンプトを起動するショートカットを作成する
Option Explicit
Dim lnkName, ws, curDir, lnk
lnkName = "playCmd.lnk"
Set ws = WScript.CreateObject("WScript.Shell")
curDir = ws.CurrentDirectory
Set lnk = ws.CreateShortcut(lnkName)
lnk.TargetPath = "%windir%\system32\cmd.exe"
lnk.Arguments = "/k ""path=" & curDir & ";%path%"" "
lnk.WorkingDirectory = curDir
lnk.Description = "Play2 を使うための""コマンドプロンプト"
lnk.save
WScript.Echo lnkName & "を作成しました"
Set ws = nothing
```


6.3 プロキシのあるネットワーク環境

多くの大学でユーザのWeb利用を管理できるように学外へのアクセスをプロキシ経由に限定している。Webブラウザでのプロキシ利用の設定方法は昨今はかなり洗練されてきたが、他のソフトウェア（特に開発系）については、個別の対応を必要とするのが現状である。開発系のソフトウェアがネットアクセスを行なうのはそれぞれのソフトウェアがネット上の何らかのリポジトリから情報を引き出す時である。今回扱ったソフトで言えば、Eclipseが拡張機能のインストールの時に使うし、Android SDKも同様にリポジトリを使う。Eclipseのプロキシ設定はGUIベースであり、Android SDKマネージャでも（Eclipseのメニューから起動できるので同じソフトの一部のように見えるのだがそれに反して）プロキシ設定は別のメニューで用意されている。学生は大学と自宅などプロキシ利用方針の違う場所を行き来してノートPCを使うため、これらの設定を固定しておく訳に行かず、何らかの方法で各ソフトウェアのプロキシ設定を切り替えることが必要になる。しかしこれを自動で切り替える機能を各ソフトウェアが内蔵している訳ではないため、様々なソフトウェアに共通で流用できる簡潔な方法は見つけられていない。こういったGUI系のソフトについては、当面はどのメニューのどこで変更するという情報を提示して学習者各自で手作業で切り替えてもらうしかないさそうである。なお、リポジトリへのアクセスにhttpsを用いる設定をしているソフトが、httpsのプロキシを使ってアクセスに失敗することがあるので、強制的にhttp（httpsでなく）を使うよう設定する必要があるケースがある。5.4章で紹介したSbtに関しては、やはりライブラリのリポジトリを検索するためにhttpを使う。Sbtにプロキシ経由のアクセスをさせるためには、起動のためのjava

コマンドに

```
-Dhttp.proxyHost=xxxxxx -Dhttp.proxyPort=9999
```

のようにパラメータを挟めばいい。ネット環境によりプロキシ使用を切替える必要があるなら、環境に応じたsbt.batファイル（に代わるもの）を用意して同じ場所に置けばいい。conscript, giter8といった、プログラム本体がjarファイルで提供されているツールは概してこの方法で対応できる。

7. むすび

以上に見て頂いたように、プログラミングという営みの未来を見据えた上で入門教育向けのプログラミング言語の選定と教程および環境の準備を行い、授業を開始できる状態にすることができた。その過程で考察または検証を通じて得た知見をここに報告した。プログラミング言語は、言語の1カテゴリとして、使う人（に加えて使わせる人や教える人も）が文化として育てていくものである。その中で教える人の影響力は強いものがあるだろう。その立場としての自覚から、ともすれば個人の慣れや好みによるバイアスがかかりやすい領域で、そのバイアスを極力排除した議論を行ったつもりである。1つの実習授業を行うためには様々な準備が必要となる。上記のようになんとか作り上げた教程と環境だが、継ぎ接ぎだらけの未完成的な印象があるのは否定できないだろう。しかし変化の激しい技術の世界のことであり、完成を求めて開始を遅らせるよりは、不完全でも教程の実施を開始して、経験を積みながら改良を重ねて行くことが重要だろう。本稿を見て頂いた方からのご意見も頂きながら、よりよい教程に改める作業は今後も続けて行きたいと考えている。

[参考文献]

- 1 平岡信之「入門教育向けプログラミング言語に関する調査研究」長野大学紀要、2012
- 2 Martin Odersky ほか「Scala スケーラブルプログラミング第2版」インプレスジャパン、2011
- 3 Casey Reas, Ben Fry 「Processingをはじめよう」オライリー・ジャパン、2011
(その他、ソフトウェアに関する情報源はURLまたはドメイン名を本文中に[]で囲んで示した。)