Swift によるプログラミング教育について

箕原辰夫[*1]
Email: minohara@cuc.ac.jp

*1: 千葉商科大学政策情報学部政策情報学科

◎Key Words　　Swift, プログラミング教育, Python, プログラミング言語

1. はじめに

非常勤講師として勤める慶應義塾大学藤沢湘南キャンパスにおいて、2019 年秋学期 1 コマの授業で Swift を使ったプログラミング教育の講義を行なった。その講義を通じて経験した、Swift によるプログラミング教育の意義と問題点について考察し、特に、Python を用いたプログラミング教育との比較を考える。Swift は、コンパイラとインタープリタの両方の開発環境を備えており、Mac OS X だけでなく、Ubuntu/Linux 上でもインタープリタが稼働するため、インタープリタを利用したプログラミング基礎教育を行なった。Apple Computer の方針に添って、過去との互換性をかなぐり捨てつつ言語仕様を進化させている Swift であるが、アプリケーションを開発するためのコンパイル言語としても機能させるために、複雑な制御構文などを導入している。その功罪について考察する。

2. 授業内容と授業環境

2.1 科目の位置付け

慶應義塾大学湘南藤沢キャンパスでは、1 年次に必修の科目として、「情報基礎 1」を春学期週 1 コマ 2 単位、「情報基礎 2」を秋学期週 1 コマ 2 単位で開講されている。この科目では、2019 年度までは JavaScript, HTML, CSS などを学生は学んでいた。なお、2020 年度からは、Python を学ぶことに変更されている。1 年次秋学期以降は、プログラミング系の科目を履修可能になっており、「情報基礎 1」の科目の履修を前提として、「オブジェクト指向プログラミング基礎」という科目を週 1 コマ 2 単位として春学期・秋学期同じ内容で担当している。なお、秋学期に担当した 2 コマの同名の科目のうち、1 コマは Python、もう 1 コマは Swift をプログラミング言語として扱うようにシラバスを構成した。学生は、同名の科目なので、どちらかの 1 コマだけしか選んで履修することしかできない。Swift を対象のプログラミング言語として選択したのは、2019 年度秋学期が最初であり、試行錯誤を伴うかなり実験的な授業になった。なお、学生は卒業時までにプログラミング系の授業を 4 単位取得する必要があり、この授業を受けても、更にもう 1 科目履修する必要がある。

2.2 各回の授業設計

毎回の授業は、基本的には、基礎プログラミング教育として、毎回教員がプロジェクタを使って、実際のプログラムを解説とともに入力して、授業回の中でいくつかのプログラム課題を出すという旧来からの方式に変更はない。ただし、Python の授業も同じだが、Swift の授業でも毎回の授業冒頭で小テストを行ない、理解度のチェックと出席確認を兼ねている。

各回の授業設計は、クラスの記述まで行なえるようなシラバスを立てたが、表にあるように、達成できたのは、リスト (配列) などを本格的に扱うところの導入までに留まった。後で説明する Swift 特有の、先進的な言語で採用されている仕組みを整理せずになんでもかんでも導入してしまっている言語設計に翻弄されてしまった形になっている。過去の PC Conference でも、Swift を使ったプログラミング教育に関する発表もあって、そのときに発表者と議論したが、Swift の言語のバージョンの移り変わりについて、どの教育者も悩まされている状況にある。Python も、最新版はかなりいろいろな要素が付け加わっているが、過去に記述されたプログラムがまったく動かないというような変更は Python 2 から Python 3 に替わるところだけだったので、Swift のように過去のプログラムがほぼ動かない 4.0 版以降の悩みからは解放されている。

表 1 各回の授業内容

授業回	実際に授業で扱った内容
第 1 回	開発環境の導入とオブジェクトモデルの説明
第 2 回	Swift のリテラル
第 3 回	文字列と構造的なリテラル
第 4 回	ターミナルへの入出力とオプショナル型
第 5 回	if 文による条件分岐
第 6 回	switch 文による条件分岐・その他の制御文
第 7 回	while 文と for 文による繰返し
第 8 回	範囲指定と for 文
第 9 回	その他の制御構文 (break 文, continue 文)
第 10 回	関数定義
第 11 回	整数論の関数・再帰関数
第 12 回	Swift 特有の関数呼出し
第 13 回	クロージャ・ジェネリック関数と内部関数
第 14 回	配列と典型的な操作
第 15 回	配列と高階関数・2 次元配列

これらの授業回以外に、エキストラの授業を 2 回行なった。1 回目のエキストラの授業は、Xcode を使った GUI アプリケーションの開発で、UIKit から SwiftUI[2]に変更されたものに対応するための授業で、もう 1 回は SwiftUI を用いて実際にイベント駆動でユーザの入力に対応する GUI アプリケーションのプログラミングの解説を予定していたが、学生の要望により、最後の課題の説明に終わった。課題としては、以下のようなプログラムを記述してもらった。最後の課題では、文字列の分解や辞書構造を利用する問題となっている。Python でも同様の問題を課題として出しているが、Swift では言語構造の解説に手間取り、出題個数がかなり限られたものになった。

- 完全数を素数とメルセンヌ数から高速に求める問題
- 入力された整数を素因数分解する問題
- floor 関数と ceil 関数を定義する問題
- 英語を単語分解して、単語ごとに日本語訳に変換する問題（簡単な語順の入れ替えも含む）

2.3 受講学生と授業評価

学生は、先修科目の情報基礎 1 および 2 で JavaScript や HTML, CSS を学んできているが、2019 年度までは情報基礎での扱いは Web のマッシュアップのためのツールとして学んでいるに過ぎず、プログラミング教育としては科目設計されていない状態であった。そのため、授業としては、プログラミングをほぼ経験してこなかった学生を対象に授業展開をせざるを得なかった。また、1 年次に情報基礎の受講が終わって、その内容をほぼ忘れてしまっている 2 年次以降の学生も多いため、「基礎」が科目名についたプログラミング系の科目では、最初からプログラミング教育を再度始めなければならない。

Swift の開発環境である Xcode[3]が Mac OS X でしか動かないこともあり、学生の履修者数は、33 名と限られたが、その中でも 4 名ほどは履修しただけで 1 回も授業に出てこず、毎回の授業の出席者数は 23〜25 名程度に限られていた。履修してきた学生の中には、Swift でアプリケーションを開発してきた学生もいたが、基礎プログラミング教育を受けてこなかったため、総じてアルゴリズムをプログラムに落とし込むのには慣れていない状態であった。また、学生の中には、授業が 4 時限・5 時限と連続していたので、4 時限の Python 版の授業を正規に履修し、5 時限の Swift 版の授業も参考程度に履修する学生もみられた。

毎回、スライドの作成に悩まされ、Swift に関しての過去の知識（Swift 4.0 でアプリケーションを作ってきた経験はあった）との差異によって、思うように授業は展開できないでいた。到達目標である自分でクラス定義してオブジェクトを生成するところまでは達成できず、1 コマでの授業の時間の短さと、過去との互換性をかなぐり捨て去る Apple Computer の方針を恨みながら、15 回の授業は終了した。ただし、学生からの授業の評価は例年の授業評価および Python 版での授業の授業評価と比べてそれほど悪くなかった。プログラミング基礎の部分は、学生は共通して学べたのではないかと考えている。

2.4 授業で用いた学生の開発環境

Swift 自体は、Mac OS X と Linux 上でしか稼働しないため、Windows のノート PC を利用して受講する学生はいなかった。受講者の全員が、MacBook Pro/Air のいずれかを持参して受講する形になった。また、基本的な開発環境として Xcode の統合開発環境をインストールする必要があり、このダウンロードとインストールについては、非常に大きなサイズで時間が掛かるために、予め第 1 回目の授業開始前に受講学生にメールを出して、インストールをしてから参加してもらうようにしてもらった。ただし、授業で使うのは、コンパイラの方ではなく、インタープリタの Swift の開発環境であったため、Xcode 自体は SwiftUI 等のアプリケーション開発のとき以外は使わず、授業での標準的な開発環境として、古くから Mac では用いられているプログラミング用のテキストエディタである BBEdit[4]を使うことにした。

Xcode を避けたのは、小さなプログラムを実行するのにも、アプリケーションの形で 1 つのプロジェクトにする必要があり、各プログラムを単独で実行できる機能がなかったこともある。一方 BBEdit は、プログラム毎にファイルに保存し、実行することができる。特に BBEdit は、今まで数年使い続けてきた TextWrangler（32bit モードでしか動かないので新しい Mac OS X では稼働しない）の後継の 64bit 版のシェアウェアになっており、30 日間の試用期間を過ぎても、フリーモードで使い続けることができることが決め手となった。VSCode でも Swift は開発することが可能だが、授業開始までの準備が間に合わなかったので採用しなかった。

なお、Swift のインタープリタは、コマンドラインから起動できる。BBEdit では設定することによって、標準のターミナルのアプリケーションから起動させることができる。コンパイラではなく、このインタープリタ（/usr/bin/swift）を利用することにした。

2.5 授業で用いたテキスト

参考図書は、Swift が 2019 年後半に 5.0〜5.1 版になったことによって、ほとんどそれまでの日本語のテキストは利用できなくなってしまった。特に、4.0 版よりも前の版で書かれているテキストは、ごく一部を除き、ほとんど使い物にならなくなってしまった。また、4.0 版に対応して修正された改訂版がでたテキストでも、以前の版の内

容が残っており、4.0 版以降では使えない内容が混じっているようなテキストもあった。結局、英語版の Apple Computer および swift.org から出されている“The Swift Programming Language”[1]のテキスト以外にはまるで参考にならなかった。ただし、荻原剛志氏の『詳解 Swift 第 4 版』[5]は非常に参考になった。だが、この本も 5.0 以降の版には対応していなかった。2019 年後半になってから 5.0 以降の版に対応した改訂版が出て、日本語で記述された Swift のテキストとしては、一番信頼がおけるものとなっている。授業用のテキストは、この『詳解 Swift 第 4 版』および英語版の The Swift Programming Language の内容を踏襲して、Keynote のスライドを PDF 版に直したものを以下の URL で一般公開することとした。ただし、後半部分がまだ未完成な状態のままである。

https://web.sfc.keio.ac.jp/~minohara/lectureslide/swift

3. Swift 特有の計算式・構文

Swift は、先進的なプログラミング言語で採用されている仕組みを整理せずになんでもかんでも導入してしまっている言語設計になっており、過去の 4.0 以前の版との互換性がまったくなくなっている状態である。そのため、数行のプログラムでも以前は動いていたが、現在の 5.0 以降の版ではまったく動かないという状況になっている。

3.1 インタープリタ言語として

Swift は元々コンパイラ言語であるので、C++/Java などと同様に厳格な型システムを導入している。しかしながら、インタープリタとしても動くようにしているために、柔軟な型推論の機構が入っている。その面では、厳密な型指定を行なう C++/Java に比べて、Swift は記述しやすい言語とは言える。以下のように型指定を省略して記述することが可能になっている。

```
let message: String = "Hello" // 型指定あり
let answer = "Hi" // 型推論によって省略
```

let による定数宣言と var による変数宣言だが、インタープリタでは実行時に警告がでる場合もあるが、かなり緩やかなものとなっている。しかしながら、Xcode の開発環境では、事前にチェックされ、var による変数宣言が行なわれた後にまったく自己参照代入などを行なっていない変数については、コンパイル前に、let による静的な定数宣言に置き換えろと執拗に迫ってくる。

3.2 オプショナル型の記法

初修者が Swift で一番に躓く点は、オプショナル型の宣言の記法であろう。nil による変数に値がない状態を許すオプショナル宣言の導入は、その解凍の記法も伴って、煩わしい記述を必要とされる。例えば、ターミナルから文字列を入力する場合や、文字列を数値に変換する際には、このオプショナル型の戻り値が関数から返されるので、それらを解凍するための!演算子を記述する必要がある。例えば、以下の記述はターミナルから整数値を得るときの記法だが、変数宣言ではオプショナル型を示すための?指定子が必要で、値の評価の際には、!演算子を必要とする。これは、最終行のように 1 行で記述することができる。

```
var line: String? = readLine( ) // オプショナル型
var optvalue: Int? = Int( line! ) // オプショナル型
let value = optvalue! // 解凍して評価
let value = Int( readLine( )! )! // 1 行で同じ内容を記述
```

3.3 変数の省略と構文の句による修飾

switch 文では、case の変数にタプルとして値を受け取るが、その変数は参照しない場合、_（アンダーバー）変数による省略記法を用いることができる。for 文でのループ変数の値を参照しない場合は、_変数を用いることができる。for 文の_変数は、Python でも使用可能である。また、制御文においては、オプショナル型の定数に代入が行なわれたかどうかを条件判定に用いることができ、オプショナル束縛と呼ばれている。加えて、条件判定の際に、where 句を用いて、条件に合致する際だけ、変数に束縛させることが可能になっている。これらの修飾句などのバリエーションにより、旧来の制御文に掛けた時間の 2 倍近く制御文の説明に時間を要した。

3.4 関数定義およびクロージャとジェネリック関数

関数定義においては、Swift は悪しき Objective-C の属性を引き継いでいるため、関数の引数にタグをつけなければならないが、_（アンダーバー）による省略記法を用いて、そのタグを省略して実引数を指定することができる。Python では、逆にタグはキーワード引数としていて、その引数のデフォルト値を与えて、実引数を省略可能にするために用いられていて、こちらの方が使いやすい。

```
func square( x: Int ) -> Int { return x * x }
// 呼出しは、square( x: 23 )のようにして呼び出す
func square(_ x: Int ) -> Int { return x * x }
// 呼出しは、square( 23 )のようにして呼び出す
```

クロージャは、Python でいうところの無名関数として高階関数に対して引数として渡される関数であるが、Java の無名メソッドのようにクロージャの中にいろいろ記述できることや、クロージャ自体への引数などの省略記法があり、プログラムの可読性を低くしている。また、戻り値がある場合でも、その記述の仕方に多くのバリエーションがあり、どの記述に遭遇しても同じ動作をすることをプログラムの読者は認識しなければならない。例えば、names という文字列配列に対して降順に整列した配列を生成するために適用される sort(:by)高階関数に対して、以下のような省略形の記述をすることができる。

```
names.sorted(by: { (s1: String, s2: String) -> Bool in return s1 > s2 } ) // 省略無し
names.sorted(by: { (s1, s2) in return s1 > s2 } ) // 型推論による省略
names.sorted(by: { (s1, s2) in s1 > s2 } ) // return の省略
names.sorted(by: { $0 > $1 } ) // 簡略引数名の使用
names.sorted(by: > ) // 比較演算子だけの指定
```

ジェネリック関数は、C++/Java のテンプレートを引き継いだ形になっていて、複数の型に対して同じような処理を関数で行ないたいときに、T などの型名で型自体を引数にする方式である。下記の swapper 関数は引数に inout 修飾子がついており、参照渡しになっており、2 つの引数の変数の値を交換するものである。呼出し時に型名が与えられていないが、型推論によって T には Int が代入される。また、参照渡しの&演算子が付けられている。

```
func swapper<T>(_ a: inout T, _ b: inout T ) {
    let temp = a; a = b; b = temp
}
var (x, y) = (12, 56)
swapper( &x, &y )
```

4. Python によるプログラミング教育との比較

Swift の基本的な言語構造は、Python と Java の中間的なもの、あるいはその両者の特長を取り込んだものになっているため、これまで両方のプログラミング言語を教えていた経験からは、基本構造の共通性から教えやすいものになっている。しかしながら、型指定が必要なコンパイル言語を基本としていること、省略記法の多さから、プログラミング初修者の学生にとっては非常に覚えにくい言語であると考えられる。

インタープリタは初等プログラミング教育に必須であり、Java のように簡単な計算結果を得るためだけに公開クラスを作成し、その中にクラスメソッドである main メソッドをシグネチャがあう形で定義しなければならないのは馬鹿げている。簡単な例えば、「45 * 67」のような計算を行なうときに、インタープリタでは、その数式をそのまま記述すれば、計算結果を表示してくれる。Swift や Python のインタープリタでは、この当たり前の環境が実現されている。初等プログラミング教育では、このようなインタープリタの環境からプログラミングを始めるべきであり、前提となるクラス作成や main 関数などの定義などは、クラス・関数の概念を学んでから使うべきである。

クロージャの省略記法については、プログラミング初学者にとっては、プログラムの可読性を低くする効果しかない。行き過ぎた省略形は、記述されたプログラムへの理解を阻害する。ある自然言語について、方言を全部覚えておかないと口述された内容がまったく理解できないということと同じである。Python のように、一定の書き方で記述でき、記述の仕方のバリエーションを少なくする方が、プログラムの可読性はあがる。

Python は、タプル・文字列・リスト（配列）にスライス記法とインデックス記法が統一して使える。要素を持つ構造型のリテラルやデータに対して、同じ記法が使えるのは無駄のない言語設計と思える。また、これらの構造型のデータに適用されるジェネリック関数も、元々型指定をしない Python の扱いの方が優れている。Swift は、Java や C++の多相形（テンプレート）を引摺っており、ジェネリック関数を型指定で宣言する必要があるのは不便であると感じる。

5. おわりに

近年は、Rust などのコンパイル型プログラミング言語なども人気であるが、それは C/C++/Java 言語でプログラミング基礎教育を受けて来た人達が先進的な要素が加わった言語として用いているのではないかと思われる。Python, Go, Swift などは、インタープリタ型のプログラミング言語であるが、プログラミング基礎教育においては、1 行からプログラムが記述できる、あるいは計算式による電卓機能を持ったインタープリタ型のプログラミング言語から始めるべきである。Python が全世界的に標準的なプログラミング基礎教育言語になっているのとは対照的に、Swift は所謂 Apple Computer 系のアプリケーションを作るための言語として位置づけられているが、プログラミング基礎教育に使えなくもないことがわかった。ただし、圧倒的なユーザ数を誇る Windows 上でも動く Swift のインタープリタの開発が普及には必要と思われる。Mac OS X 以外では、Ubuntu や Cent OS しか稼働しないので、他の Linux のディストリビューションも含め、Windows 10 でも動作する環境が構築されること、および今後は言語仕様が劇的に変化されないことをプログラミング教育者としては渇望している。

参考文献

(1) Apple Computer：“The Swift Programming Language Swift 5.0”, iBook, pp.1207 (2019) same as swift.org：“The Swift Programming Language”, https://docs.swift.org/swift-book/

(2) Apple Computer：“SwiftUI”, https://developer.apple.com/documentation/swiftui (2019).

(3) Apple Computer: “Xcode”, https://developer.apple.com/xcode/ (2020).

(4) Bare Bones Software: “BBEdit”, http://www.barebones.com/products/bbedit/ (2020).

(5) 荻原剛志：“詳解 Swift 第 4 版”, pp.568, SB クリエイティブ, ISBN 978-4797395181 (2017).