Swift によるプログラミング教育について

箕原辰夫

千葉商科大学政策情報学部

（慶應義塾大学環境情報学部非常勤講師）

CIEC PC Conference 2020 Online

発表概要

- 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（総合政策学部および環境情報学部）での担当科目
- 講義名「オブジェクト指向プログラミング基礎」
- 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。その内容の紹介
- Swiftのプログラム記述方法の特徴を紹介
- Swiftによるプログラミング教育の意義と問題点について、特に、Pythonを用いたプログラミング教育との比較から考察

科目の位置付け

- 科目名「オブジェクト指向プログラミング基礎」
- 慶應義塾大学の湘南藤沢キャンパス（SFC）にある社会系の総合政策学部と情報系の環境情報学部の主に2年次以降の学生を対象とした科目
- 両学部の学生は、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る。なお、2020年度からは、Pythonを学ぶことに変更されている。
- 「オブジェクト指向プログラミング基礎」では、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。
- iPhone/iPad向けのアプリケーション開発を将来行ないたいと考えている学生の需要はあるだろうと推測した。

各回の講義内容

- 基本的に、プログラミング言語を学ぶ基礎科目としての講義内容を踏襲している
- Swift特有の記述の仕方のバリエーションが多くて、その部分で足を引っ張られてしまった
- 到底、クラス定義までは辿り着けなかった→「基礎」がついていない次の科目が必要
- アプリケーション開発の部分は、授業回とは別の回を1回だけ設けて、新しいSwift UIについて授業を行なった

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の型とリテラル
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

講義の課題内容

- 完全数を素数とメルセンヌ数から高速に求める問題
  - アルゴリズムによって高速に解が得られることを体感させる
  - Swiftでは、整数は64bit符号付きの表現になっているので、そこまでの完全数しか求められない
- 入力された整数を素因数分解する問題
  - 配列リストを返す関数の定義が必要
- floor関数とceil関数を定義する問題
  - Swiftの数学関数は、昔ながらのC言語の<math>ライブラリベースになっていて、正規のドキュメントに細かな仕様は書かれていない
- 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）
  - 辞書構造を利用して、単語の日本語訳を引き出す必要がある

学生の受講前の状況

- 2019年度までは、「情報基礎」で学ぶJavaScript/HTML/CSS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ない
- 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再教育する必要性があった
- 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

学生の受講・開発環境

- 学生の手持ちのMacBook Air/Proで受講する（2019年度までは湘南藤沢キャンパスでは標準の環境）
- Swiftの開発環境であるXcodeは、Mac OS Xでしか動かない、Xcodeは1GB以上あるので、授業開始前にダウンロード・インストールをしてもらった
- Swiftのインタープリタ自体は、Mac OS X以外にもLinux/Ubuntu, CentOSで稼働させることができる（今回はSAの学生以外はすべてMac OS X）
- Xcodeは、プログラムごとに１つのプロジェクトを作成しなければならないために、授業の標準環境としては、使用を避けた
- Swiftのインタープリタと、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を利用した。VSCodeの利用は、準備時間がなくて使用を避けた
- Swiftインタープリタは、シェル（ターミナル）から起動可能

Swiftの特徴

- Mac OS Xで稼働するアプリケーション、iOS（iPhone/iPad）で稼働するアプリケーションの標準開発プログラミング言語
- コンパイラだけでなく、インタープリタも利用することができる
- swift.orgという団体で、一般にも公開されている（Windows版はない）
- NextStepからのObjective-Cのライブラリをまだ引摺っているところがある（Objective-Cは記号だらけの記述しにくい最悪の言語であった）
- 最初は、Java言語に似た設計であったが、版が2.0→3.0→4.0と進むに従って、Pythonなどの先進のプログラミング言語の要素を採り入れてきた
- Apple Computerの製品なので、過去の製品との互換性をかなぐり捨てており、最新の5.0版以降では、3.0や4.0までの記述方法ではエラーになってしまい、コンパイルできない
- 過去にCIECのPCCでも3.0から4.0に置き換わるときに授業で採用して、劇的に変わったので教えた内容が使い物にならないという発表が、このセッションであった

参考としたテキストと授業スライド

- 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、ほとんど使い物にならなかった（エラーになってしまう）
- Apple Computerおよびswift.orgから出されている“The Swift Programming Language”のテキストだけが頼り（WebとiBookで閲覧可能）
- 荻原剛志氏の『詳解 Swift 第4版』は非常に参考になったが、5.0版以降に対応した改訂版が出たのは、授業後半ぐらいだった

- 作成した授業スライドは、一般で観れるようにしている（ただし、途中までしかできていない）
  - https://web.sfc.keio.ac.jp/~minohara/lectureslide/swift

授業講評

- もともと、Swift 4.0でアプリケーション開発をしていたので、4.0版までの文法の知識があったが、5.0版で動かない部分が出てきたので、そのキャッチアップに追われる
- GUIのライブラリもUIKitから、Swift UIに授業期間直前に替わってしまい、そのキャッチアップに追われる
- 前のコマの4時限がPythonでの授業で、次の曜日にJavaScriptの授業だったので、教員側が少し混乱する
- 学生の受講者は33名で、実質出席者は23〜25名程度
- 毎回の授業開始直後に小テストをやっているのが好評、学生は自分の理解度が確かめられたとのこと
- 授業中に作ったプログラムは、LMSで授業後に公開した
- クラス定義まで進めなかったが、もともとのプログラミング基礎の科目がなかったので、プログラミングの基礎が学べたと学生から評価がされた

Swiftの変数の型指定

- Swiftは、コンパイル言語としての特徴も兼ね備えているので、実行前（コンパイル時）における型推論が可能になっていなければならない
- インタープリタ言語でもあるので、C++/Javaよりは型指定を厳格に指定しなくても、ある程度、柔軟に型推論してくれる
- 例：

  let message: String = "Hello" // 型指定あり

  let answer = "Hi" // 型推論によって省略

- Xcodeの開発環境では、執拗に、「これ変数なのか！？おまえの使い方だと後で代入がないから、定数にしておくべきじゃないのか！」と注意してきて、コンパイルさせてくれない（Swiftインタープリタは警告表示するだけ）
  - 定数の場合：**let** value = 10 // これ以降代入ができない
  - 変数の場合：**var** value = 10

オプショナル宣言（オプショナルを伴う型）

- nilによる変数に値がない状態を許すオプショナル宣言
- オプショナル宣言を伴う型指定は、型名?という形で記述する
- オプショナルは、変数にラップが掛かった状態と考えられる。これを解凍するには、!演算子を用いる（not演算子とは異なり、式あるいは変数名の後に付ける）
- 関数の戻り値として、オプショナル型の値を戻すものが多い（端末からの入力や型変換など、値がないときの対応のため）
- C#にも導入されていると思うが、Swiftほどは頻繁に使われていない
- 例：

```
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
```

制御文での変数代入

- if, while, switch文などの条件式の部分にlet節を伴ったオプショナル型の代入式（束縛：bindingと呼ばれている）を混入させることができる
- 代入結果がnilの場合は、falseという形で評価される
- 例：

```
var s : String = "1234" // 文字列の値
if let num = Int( s ) {  print( num! ) } // 変換できた場合
else { print( "\(s) は整数に変換できません" ) }
```

- switch文のcaseのところで、変数に代入ができる
- 例：

```
let anotherPoint = (2, 0)
switch anotherPoint {
case (let x, 0): print("on the x-axis with an x value of \(x)")
case (0, let y): print("on the y-axis with a y value of \(y)")
case let (x, y):  print("somewhere else at (\(x), \(y))")
}
```

制御文での修飾句

- 制御文で代入式（束縛）が導入されるときに、更に、代入された変数についての制限をwhere修飾句で入れることができる
- 例：

```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
case let (x, y) where x == y:
   print("(\(x), \(y)) is on the line x == y")
case let (x, y) where x == -y:
   print("(\(x), \(y)) is on the line x == -y")
case let (x, y):
   print("(\(x), \(y)) is just some arbitrary point")
}
```

関数の位置仮引数の指定

- 関数の定義の際の仮引数は、キーワードとしても利用されている。関数を呼び出すときは、キーワードを指定して実引数を与える必要がある。
- 位置だけの仮引数を利用した場合は、アンダーバー（_）を付ける必要がある（仮引数の変数名の前に空白を入れて、その前に指定する）
- 例：

```
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

- 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。
- 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認識しなければならない。
- 例：

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"] // 文字列の配列

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

- 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
- 下記の例では、型は明示的に渡されていないが、自動的に型推論によって、TにIntが代入されている
- また、下記の例では仮引数にinoutの修飾語があるが、これは「参照渡し」を意味している。そのため、実引数である、2つの変数には、参照渡し（C言語でのポインタ・C++言語での参照）を示すための&演算子が使われている
- 例：

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

言語構造の違い

- 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、Swiftは型指定が必要なコンパイル言語を基本としていること、および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。
- Pythonも最近の版では、C/C++/Swiftが持っているような代入式も入ってきており、全体的には両者の書き方は似通ってきているが、記述能力の高さについては、SwiftはPythonに劣る。その大きな原因は、型指定の部分にある。
- Pythonにおいて型指定（アノテーション）はあくまでもインタープリタに対する型ヒントであり、間違っていても動く。 x : int = "ABC"でもOK

ジェネリック関数について

- C++/Java/Swiftのジェネリック関数よりも、インタープリタベースでPythonの実行時に型が対応していれば実行できる形で記述できる多相型の方が記述しやすい
  - Swiftの場合

```
func square<T: Numeric>(_ x: T ) -> T { return x * x }
```

  - Pythonの場合

```
def square( x ): return x * x
```

構造型（シーケンス型）についての統一記法

- 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。演算子も同じ意味を持っている（加算、整数との乗算、in演算子など）。関数も同じものが適用できる。
- 例：

```
numlist = [ 12, 23, 34, 45, 56 ]  # list
print( numlist[ 2 ], numlist[ 1: 3 ] )
alphabet = "ABCDEFGHIJKLMN" # string
print( alphabet[ 2 ], alphabet[ 1: 3 ] )
tup = ( 12, 23, 34, 45, 56 ) # tuple
print( tup[ 2 ], tup[ 1: 3 ] )
```

- Swiftでは、C++/Javaと同様に、文字列には、そのままでは、配列と同じ記法（範囲指定）が使えない。面倒な型変換をする必要がある。タプルには、範囲指定が使えない。
- 例：

```
let numlist = [ 12, 23, 34, 45, 56 ]  // list
print( numlist[ 2 ], numlist[ 1...3 ] )
let alphabet = "ABCDEFGHIJKLMN" // string
print( alphabet[ 2 ], String(alphabet[ String.Index( encodedOffset: 1 ) ... String.Index( encodedOffset: 3 ) ] ) )
let tup = ( 12, 23, 34, 45, 56 ) // tuple
print( tup.2 )  // index only
```

クロージャについて

- Swiftのクロージャでは、Javaの無名メソッドのようにクロージャ内に延々とプログラムが記述できるのは可読性を下げることになる。
- Javaも無名クラスや無名メソッドを高階関数の実引数として、延々と記述でき、どこまでが、その内容かわかりにくくしていた。複雑な処理をする関数は、無名ではなく、命名するべきであろう。
- Pythonの無名関数は、「**lambda** 仮引数: 返す値の式」の簡潔なラムダ式（関数プログラミング言語でいうところの）になっている。
- それ以上のことをするのであれば、通常の関数として定義するべきであり、いたずらに可読性を低くするのは良くない。

インタープリタについて

- 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。
- 例えば、簡単な「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る。C/C++/Javaにおけるプログラムの実行の前提となる公開クラスや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記述するべきである。

おわりに

- 近年は、Rust, Go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, JavaScript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、インタープリタを基本とすれば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。