1.3 方程式

方程式には解き方というものがある。解法の手順と言ってもよいだろう。でも、コンピュータを使う人なら**アルゴリズム**と呼ぶほうがしっくりくるのだろう。比較的簡単な方程式は

$$\text{a) } 6x-7=11、\qquad \text{b) } -2x+6=1、\qquad \text{c) } \frac{2}{3}x+8=4$$

のようなタイプで、もう解き方なんて完璧に理解してるのじゃないかな。

ひとつひとつ解いてもらってもよいのだけれど、これらはどれも $ax+b=c$ の形をしているので、すべて同じ手順で解くことができる。それは

$$\begin{array}{llll}
\text{a)} & 6x=11+7 & \text{b)} & -2x=1-6 & \text{c)} & \frac{2}{3}x=4-8 & \text{（移項する）} \\
 & x=\frac{11+7}{6} & & x=\frac{1-6}{-2} & & x=\frac{4-8}{2/3} & \text{（両辺を係数で割る）}
\end{array}$$

というアルゴリズムであるから、何も 3 通りを並べて書く必要もなく、$x=\dfrac{c-b}{a}$ ひとつで済んでしまう。だったら、1 次方程式を解くプログラムは簡単に書けてしまいそうだね。たとえば a) だったらこんな感じだ。このプログラムに関しては `import UIKit` はあってもなくてもよい。

[Equations.playground]

```
1 import UIKit
2
3 var a = 6
4 var b = -7
5 var c = 11
6 print("\(a)x + \(b) = \(c) の解は、x = \((c-b)/a)")
```

`print` 文が少し微妙な感じではあるが、書式は以前と同じである。`\( )` が変数の値を取り出すための書式だったね。こうしておけば 3–5 行目の `a`、`b`、`c` の値を書き換えるだけで 6 行目の出力がきちんとなされる。いまの場合の出力は

```
6 x + -7 = 11 の解は、x = 3
```

となる。

しかし、このプログラムでは方程式を解いた気がしないだろう。方程式を `var str = "6x-7=11"` のように与えたとき、`x = 3` を出力してくれれば完璧なんだけど。でも、そのようにするのはプログラムを書き始めたばかりの身には相当敷居が高い（おっと、敷居じゃなくてハードルだ！）。いまは、[Equations.playground] に手を加えてプログラムの大事な処理を学ぶことにしよう。

[Equations.playground] をプログラムと呼ぶのはだいぶ間抜けだが、たとえば次々と 1 次方程式とその解を表示するのであれば、ちょっとはカッコがつくというものだ。次のように手を加えてみた。今度は import UIKit がファイル最上部に書かれている必要がある。

[Equations2.playground]

```
1 import UIKit
2
3 var a = Int(arc4random()) % 100 - 50; if a == 0 { a = 1 }
4 var b = Int(arc4random()) % 100 - 50
5 var c = Int(arc4random()) % 100 - 50
6 print("\(a)x + \(b) = \(c) の解は、x = \(Float(c-b)/Float(a))")
```

プログラムを実行すると方程式とその解が Debug Area に表示されるだろう。画面下の ▽ ボタン右にある▷ボタン（プログラム実行直後なら □）—“Execute Playground” ボタンと呼ばれている—を押すたびに新しい方程式が解かれていくはずだ。実行ボタン ▶ を押しても同じことだ。

だいたい良さそうに見えるが、少し気にくわないところもある。たとえば、$a = -1$ のとき “-1x” と表示されたり、b が負の値のとき “+ -12” のように表示されてしまうことだ。ここは “-x” や “- 12” であってほしい。また、解 x が小数表示になってしまうのも数学っぽくない。しかし、これをきちんと**既約分数**で表すのはなかなかの手間なのだ。その手の処理方法については、いずれ学ぶことになるだろう。

さて、プログラムが何をしているか説明しよう。まず arc4random() は UIKit のコンポーネント、つまり UIKit という道具箱に入っている道具—Swift 風に言えば**関数**—のひとつで、UInt 型と呼ばれる**符号なし整数**の乱数を生成する。言い回しが難しければ、arc4random() は乱数を生成する命令と考えてもらえばよい。UIKit の関数を利用するから import UIKit が必要だったのだ。

arc4random() を Int 型の**符号付き整数**に直しているのは理由がある。まず、方程式の係数に大きな値は必要ないので、100 程度の大きさになるように arc4random() を 100 で割った余りを求めた。/ 100 では単に商を整数値で求めてしまうので % 100 としている。% は**剰余**を求める演算子だ。Swift には数学では用いない演算記号があるものだ。ただし、このままでは 0 から 99 の値になるので、負の数も出るように 50 を引いている。除算の余りを用いているため純粋な乱数ではなくなることにも注意しよう。しかし arc4random() % 100 - 50 のままだと、おそらく Swift の型推論によって、符号なし整数で (c-b)/a を計算することになり支障をきたす。それを回避するための Int() なのであった。

ex. 1 Int() を取り去って実行すると、どんなエラーを起こすか見るのも一興である。試してみよう。

これで a、b、c には -50 から 49 までの数があてがわれる。あとはそれらを print 文に渡せばよいのだが、割り算の際 0 で割ることがあっては困るので、除数の可能性がある a だけは調べている。このとき役立つのが if 文だ。if 文は

if 条件式 { 条件式が成立したときの処理 }

という書式である。いまの場合、条件式は "a が 0 かどうか" である。これが成立、つまり**条件が真のとき**に a に 1 が代入される。係数はどうせ乱数で数を選んでいるのだから、別に 1 でなくてもよいのだが。また、条件が成立しなければ if 文は無視される。

ただし、この処理はやや手抜きである。1 次方程式 $ax+b=c$ が $a=0$ では困るので強引に $a=1$ にしたのだが、本来なら乱数の計算をやり直すところだ。その場合は別の書き方をすることになる。いずれ目にすると思うので、今回は手抜き処理で我慢してもらいたい。

ところで、条件式に関して大変重要な注意がある。a が 0 に等しいことを表す書き方は a == 0 だ。= を重ねて書く。a = 0 と書くと a に 0 を代入することになってしまう。間違えやすいので気をつけよう。ちなみに、a が 0 に等しくないことを表す書き方は a != 0 である。

また、ここでは var a = ... 文の後ろに ; で区切って if 文を書いてみた。複数の文を 1 行に書くときは ; で区切るという見本だ。本来なら改行して書くべきだろうし、if 文自体も改行しながら書く習慣がある。こんな感じだ。

```
var a = Int(arc4random()) % 100 - 50
if a == 0 {
    a = 1
}
```

この書き方のポイントは、"if {" に対応する閉じ括弧 "}" が同じインデント（字下げ位置）で揃えられていることだ。このように書くと if の有効範囲がわかりやすい。いまは「それがどうした」と思うことかもしれないが、いずれ if の有効範囲を気にしなくてはならないときがくるだろう。

print 文の最後のほう、x = \(Float(c-b)/Float(a)) の部分が気になるかもしれないね。もしこれを x = \((c-b)/a) で済ませてしまうと、x の値は整数値で表示されてしまう。c-b、a が共に整数値だから、たとえ割り切れなくても Swift は整数値で表示する。そういう規則だったよね。そこで実数値で求めさせるために、分子・分母共に Float を付けてあるのだ。Double でもかまわないけれど、Float と Double が混ざってはだめだ。Swift はこういうところが気難しい。

ex. 2 前節で\(a) の代わりに String(a) を使って数値を文字列に直したことを覚えているだろうか。print 文の行を String() を用いて書き直してみよう。

これで方程式は間違いなく解けるのだが、先（せん）だって指摘したように解の表示がイマイチである。それの手直しはいくつかの if 文を追加すれば済むので、元のプログラムに手を加えて表示を完璧にしたのが次である。紙面幅の関係で 2 行に分けて書いてあるところがあるが、入力の際はコメント部分を削って 1 行で書いてもらいたい。

[Equations3.playground]

```
 1 import UIKit
 2
 3 var a = Int(arc4random()) % 100 - 50; if a == 0 { a = 1 }
 4 var stra = ""
 5 if a == 1 {
 6     stra = "x"
 7 } else if a == -1 {
 8     stra = "-x"
 9 } else {
10     stra = String(a) + "x"
11 }
12 var b = Int(arc4random()) % 100 - 50
13 var strb = ""
14 if b > 0 {
15     strb = " + " + String(b)
16 } else if b < 0 {
17     strb = " - " + String(-b)
18 }
19 var c = Int(arc4random()) % 100 - 50
20
21 print(stra + strb + " = " + String(c) + //__[続けて 1 行で書く]
                                " の解は、x = " + String(Float(c-b)/Float(a)))
```

急にプログラムの量が増えたように見えるが、そんなことはない。[Equations2.playground] と見くらべれば、それぞれの行間に if 文が挿入されただけであるのがわかると思う。行数を増やしているのは、最初の条件が真でないとき、つまり**条件が偽のとき**の処理が再度 if 文で、しかも丁寧に改行までして書かれているせいだ。それは

```
if 条件式 1 {
    条件 1 が真のときの処理
} else if 条件式 2 {
    条件 2 が真のときの処理
} else {
    条件 2 が偽のときの処理
}
```

という風に書かれる。処理は何行かにわたることも多いので、このような書き方が主流となってい

る。else は “条件に合わなければ” という意味であるから、else の後にもう一度 if で分岐させるかは状況による。ここでは a == 1 でないとき、再度 a == -1 の真・偽を問うている。

今回のプログラムでやりたいことは、係数 a の値を x の前に書いて ax と表示させたいのである。それは "\(a)x" か String(a)+"x" で実現できるのだが、数学の約束は $1x \to x$、$-1x \to -x$ とすることなので、a == 1 と a == -1 の場合は都合が悪い。それで場合分けが必要になった。しかし、それ以外では単に String(a)+"x" で済むので、それを else 処理に書けば終わりだ。もし、とりあえず String(a)+"x" を作ってしまい、あとから不都合の修正をするという考えなら、4–11 行目を

```
var stra = String(a) + "x"
if a == 1 {
    stra = "x"
} else if a == -1 {
    stra = "-x"
}
```

のように書けばプログラムをより短くできる。プログラムの順次処理を利用したものだ。ただ、プログラムの見やすさとしては、先に stra に値が代入されているので、if stra == "1x"としたほうが良かったかもしれない。

次は b の処理だ。数学では $ax+b$ と書いても、$a=3$、$b=-2$ なら $3x+-2$ ではなく $3x-2$ と書く。$+$ はどこかへ飛んでしまうのだ。一歩譲ったとしても $3x+(-2)$ である。その場合は ( ) が必要だ。[Equations2.playground] では 3x+-2 を表示してしまうので b の正負で場合分けが必要になる。本来ならどちらにも合致しないとき、すなわち else if b == 0 も加えるべきところだが、すでに var strb = ""で初期化されているので else 後の if b == 0 は省略できたのだ。

ex. 3 [Equations3.playground] を、$b<0$ のときは ( ) を付けて表示するとの考えで書き換えてみよう。

最後の print 文は直感的にわかりにくい書き方になっているだろう。方程式とその解を出力するなら以前のように、係数や解を\( ) で囲んで文字列に埋め込むのが見やすい書き方なのかもしれない。print("\(a)x + ...") と書くほうが print(stra + ...) と書くより、何を出力するか推測しやすいだろう。しかしそうすると、1x などと表示させないために a の値を ""（これは**空文字**という文字列—**ナルストリング**とも言う—である）にする必要が生じるが、いまのところ a は整数型の変数なのだ。もちろん整数型を文字列に変換したり、その逆の変換を行ったりできるのだが、それはいささか煩（わずら）わしい。そこで出力を見越した場合分けをしたのだが、他にもっとよい方法があったかもしれない。研究するとよいだろう。

ここで print 文について少しばかり話しておきたい。[Equation3.playground] において解の表示をする際、print 文がだいぶ煩雑になっている。原因は 1 行の中に + による結合が多すぎるのだ。できることなら

```
print(stra + strb + " = " + String(c) + " の解は、")
print("x = " + String(Float(c-b)/Float(a)))
```

のように 2 行に分けて書くことができれば少しは見やすくなるのだが、print() は文字を表示したら改行する仕様である。これまでに print 文の結果がサイドバー領域に示されたとき、"18.8\n" みたいな表示が気にならなかっただろうか。\n ってなんだ？

\n は**エスケープシーケンス**と呼ばれる記号で、\（バックスラッシュ）と組み合わせて特別な意味を持たせている。で、\n は改行の意味になる。これは大事な仕組みでたとえば

```
print("これがエスケープシーケンスです")
```

で「これがエスケープシーケンスです」が出力されるが、強調のため「これが ”エスケープシーケンス”です」と出力したいときどうするのだろう。

```
print("これが"エスケープシーケンス"です")
```

はエラーになる。なぜなら、どこが " " で囲まれているかわからないからだ。そこで " を出力したいときのために \" が用意されている。この場合は

```
print("これが\"エスケープシーケンス\"です")
```

と書けば目的どおりの出力になる。細々したことかもしれないが、少しずつ身につけよう。

でも、改行しないで print() を使う方法もある。以下のように書けば、改行されず文字列がつながる。

```
print(stra + strb + " = " + String(c) + " の解は、", terminator: "")
print("x = " + String(Float(c-b)/Float(a)))
```