1.5 変化と対応

$y = 2x$ は比例の関係式である。x の値に対して 2 倍されたものが y の値になる。実際は x に数値を代入して y の値を求める。もちろん Swift でそれを試してもよい。

[Function.playground]

```
import UIKit

var x = 0
x = 4;   print("x = \(x) のとき、y = \(2*x)")
x = -3;  print("x = \(x) のとき、y = \(2*x)")
x = 5/3; print("x = \(x) のとき、y = \(2*x)")
```

playground で実行すると、`x = 5/3` の挙動がおかしいね。それは Swift が、最初の `var x = 0` によって変数が整数値であると理解したからだ。ここを `var x = 0.0` とすれば正しい値を示すけれど、残念なことに小数値としてだ。このことは繰り返し登場してきたから、十分理解できているだろう。

しかし、この調子で関数の値を調べるのはしんどい。このようなときは繰り返しの命令を与えて実行させよう。

[Function2.playground]

```
import UIKit

for x in -5...5 {
    print("x = \(x) のとき、y = \(2*x)")
}
```

たった、これだけだ。繰り返しの命令は `for` 文と呼ばれ、

`for` 変数 `in` 範囲 `{` 繰り返す処理 `}`

という書式である。変数には値を代入するための文字、範囲は `m...n` のように指定する。これは数学風には $m \leqq x \leqq n$（m, n は整数）を意味する。変数は `var x` でなく、`x` だけでよいことに注意しよう。ちなみに範囲指定は $m \leqq x < n$ の意味で `m..<n` という書き方は認められているが、$m < x \leqq n$ のつもりで `m<..n` と書くのはダメである。理由は次節ではっきり意識できるだろう。

また、範囲の小さい値 `m` から大きい値 `n` まで 1 ずつ増加するように繰り返すので、大きいほうから減少させたり、2 ずつ増加するようにしたいときは別の書式で対応することになる。

そういうときは `while` 文が便利で

var 変数の初期化
while 変数の範囲 { 繰り返す処理; 変数の増減式 }

という書式になっている。実際には

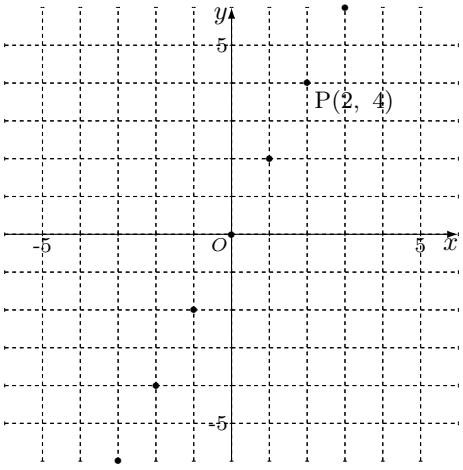
[Function3.playground]

```
1 import UIKit
2
3 var x = -5
4 while x <= 5 {
5     print("x = \(x) のとき、y = \(2*x)")
6     x = x+1
7 }
```

のように使う。いまの for 文とまったく同じことを while 文で書き換えたものだ。範囲の指定は、範囲の最初—小さいほうでなくてもよい—を var x = -5 で与えたあと、範囲の最後までの意味で while x <= 5 とすればよい。記号 <= の見栄えがイマイチだと思えば、x < 6 としよう。while 文は for と違って、変数を使う前に var で宣言しなくてはならないことも注意しておこう。

あとは{ }内に処理すべきことを記述するのだが、変数の増減は x = x+1 のように指定する。次の x はいまの x に 1 を加えるという意味だ。したがって、x = x-5 で 5 ずつ減る処理をすることができる。もし、var x = -5、while x <= 5 の範囲指定に対して、うっかり x = x-1 と書こうものなら、-5 から 5 に達することなく**無限ループ**に陥ってしまう。playground では即座に正しく訂正すれば処理は止まるが、アプリ作成では暴走の原因になりかねないので気をつけよう。

さて、比例の関係では、たとえば $x = 2$ のとき $y = 4$ というように x, y の値がセットになっている。このことから一組を x-y 座標に 1 点として取ることができる。大げさな言い方をしなくても、毎日の気温をグラフ用紙に記録することを想像すれば十分だね。$y = 2x$ の関係なら



な感じになることは知っているだろう。

では、これを playground でやってみようとなると大変だ。いずれグラフは描くことになるが、いまの時点では Swift 言語について学ぶのが先である。しばらく待ってもらいたい。ここでは Swift で扱える**タプル**について説明しよう。タプルとは、いろいろな型の値をまとめて一組で表せる記述方法である。具体的には

```
var P = (2, 4)
```

のように書くのだが、`P = ("x", 2, 3.14)` のように要素数を 3 個以上にしたり異なる型を混在させたりできる。座標を表すのに便利そうだが、座標を意識して実装されているわけではないので、`(2, 4)` だからといって `x = 2`、`y = 4` ということではない。タプルは要素を順に 0 番目, 1 番目, 2 番目, ... と数えることになっている。コンピュータ言語の多くは "0 始まり" である。よって `p = (2, 4)` については `P.0 = 2`、`P.1 = 4` となる。もし、座標として扱うなら

```
var (x, y) = (2, 4)
```

のように代入するとよい。これなら、`x = 2`、`y = 4` と代入されている。

ここまでの例であることに気づいた人はいるかな？ それは、タプルを利用すればひとつの変数にふたつ以上の値を代入できるということだ。ただ、それは次回登場する "配列" を使ってもできるので、タプル特有の性質ではない。最大の違いは、タプルは異なる型の組を作れるが配列はそれができないことである。いまはこれ以上掘り下げるつもりはないので、今後のためにも言語の仕様には気を配っておくようにしよう。

コンピュータの動作や仕様というのは設計思想に影響されるとはいえ、ひとつの命令がひとつのアクションを起こすのが原則だろう。比例関係 $y = 2x$ にしても $x = 5$ の代入が $y = 10$ だけを返す。これを "返り値" と呼ぶと浴びたとき縁起でもないので**戻り値**と呼ぶが、数学でもひとつの値を返すものを扱うことが原則だ。

でも日常的にはそうでもない。数学っぽくない例で恐縮だが、スーパーマーケットでいろいろな生鮮食品について尋ねたとき、ひとつの品物について値段と消費期限の 2 値が返ることはあるだろう。こいう動作は数学やコンピュータには合わない。しかしタプルはそれを可能にしている。

ひとつの値に対してひとつのタプルを返せばよいからだ。いずれそのようなプログラムを書く日が来るかもしれないので、この性質はしっかり覚えておきたい。

タプルの真骨頂は別のところにある。反比例の関係を考えてみよう。反比例の関係とは $y = \dfrac{1}{x}$ のように逆数を求める関係である。$\dfrac{a}{b}$ の逆数は $\dfrac{b}{a}$ だから、逆数を求めることは分子と分母を交換することである。ところが Swift でこれをやろうとしても、分数は実数で扱われてしまうので無理である。しかし、タプルを擬似的に分数と考えれば (a, b) の逆数は (b, a) であるから

```
var (a, b) = (3, 5)
print("もとの数は、(\(a), \(b))")
a = b
b = a
print("その逆数は、(\(a), \(b))")
```

で良さそうに思えるのだが、実際には上手くいかない。なぜなら、最初に a = b の代入をしたとき a の値が b と同じ値になってしまったからである。それを防ぐには臨時変数 t を用意した上で、a の値を t へ退避させなくてはならない。つまり

[Reverse.playground]

```
1 import UIKit
2
3 var (a, b) = (3, 5)
4 print("もとの数は、(\(a), \(b))")
5 let t = a
6 a = b
7 b = t
8 print("その逆数は、(\(a), \(b))")
```

のようにして、上書きされる a の値を確保しておくのである。

タプルを用いればもっと直感的な代入ができる。

[Reverse2.playground]

```
1 import UIKit
2
3 var (a, b) = (3, 5)
4 print("もとの数は、(\(a), \(b))")
5 (a, b) = (b, a)
6 print("その逆数は、(\(a), \(b))")
```

でよいのだ。

さて、まだこの段階では気の利いたプログラムを書くのは厳しいので、覚えた知識だけで分数の計算をするプログラムでも書いてみよう。といっても一般の分数の四則演算をこなすには無理があ

る。エジプト分数を例にとろう。エジプト分数は、任意の分数を単位分数、すなわち分子が 1 である分数の和で表すもので、たとえば $\frac{5}{6}=\frac{1}{2}+\frac{1}{3}$ のようなものだ。ここではリンドパピルスに書かれている $\frac{2}{n}=\frac{1}{a}+\frac{1}{b}$ のタイプに限定しておく。それは分子が 2 で分母 n が奇数である分数とする。本来なら最小公倍数などの求め方が必要なのだろうが、ここでは $\frac{2}{n}-\frac{1}{a}=\frac{1}{b}$ になるかどうかを調べる方針をとることにする。

左辺は通分すると $\frac{2a-n}{na}$ だから、$na\div(2a-n)$ が割り切れれば b が求められたと判断するのだ。この理屈がすぐにわからなければ、例として $\frac{2}{9}$ がどうなるか調べた様子を見てほしい。まず、$\frac{2}{9}-\frac{1}{2}$、$\frac{2}{9}-\frac{1}{3}$、$\frac{2}{9}-\frac{1}{4}$ と、順に引く分数の分母を増やしていく。

$$\frac{2}{9}-\frac{1}{2}=\frac{2\times2-9}{9\times2}=\frac{-5}{18}、\quad \frac{2}{9}-\frac{1}{3}=\frac{2\times3-9}{9\times3}=\frac{-3}{27}、\quad \frac{2}{9}-\frac{1}{4}=\frac{2\times4-9}{9\times4}=\frac{-1}{36}$$

はいずれも分子が負の数になるのでダメだ。その次は

$$\frac{2}{9}-\frac{1}{5}=\frac{2\times5-9}{9\times5}=\frac{1}{45}$$

でうまくいった。$45\div1$ は割り切れるしね。だから、移項して $\frac{2}{9}=\frac{1}{5}+\frac{1}{45}$ が正解だ。

ex. **1** エジプト分数の計算練習をしてみよう。$\frac{2}{3}$、$\frac{2}{7}$、$\frac{2}{13}$ などを単位分数の和で表してみよう。

手順がわかればプログラムを書くのはそう難しいことではないと思われる。プログラムも練習問題にしたいところだが、さっさと例を示してしまおう。

[EgyptFraction.playground]

```
1 import UIKit
2
3 var n = (2, 5)
4 for a in 2..<n.1 {
5     if (n.1*a) % (2*a-n.1) == 0 {
6         if 2*a-n.1>0 {
7             print("2/\(n.1) = 1/\(a) + 1/\((n.1*a)/(2*a-n.1))")
8         }
9     }
10 }
```

プログラムからは何をやっているか見当がつかないのではないだろうか。1 行目の `n = (2, 5)` が $\frac{2}{5}$ を表している。$\frac{2}{n}$ は間違いなく $\frac{1}{n}+\frac{1}{n}$ にできるので、探す単位分数はこれ以外の $\frac{1}{a}+\frac{1}{b}$ $(a\neq b)$ である。このとき $a<n<b$ または $b<n<a$ であることに注意しよう。どちらも同じことなの

で、結局 $\frac{1}{a}$ として探すのは、$\frac{1}{2}$ 以上 $\frac{1}{n}$ 未満でよいことになる。だから for 文の範囲が 2..<n.1 なのだ。こういう考え方が数学的な考え方って言うんだね。

そして計画どおり、$\frac{2a-n}{na}$ が割り切れるかどうかで結果を出力すればよい。$\frac{1}{b}$ は正の値のはずだから分子が負の値のものは対象外だ。

とくに最小公倍数を求めて通分などしなくとも、案外目的は果たせるものである。わかりやすいプログラムではないものの、工夫すればなんとかなることを理解してもらいたいのである。

ex. 2 [EgyptFraction.playground] で他の分数を調べるには n = (2, 5) を書き換えて実行しなくてはならない。たとえば $\frac{2}{3}$ から $\frac{2}{99}$ までのエジプト分数の一覧は、for 文をもうひとつ使えばできるのでやってみよう。

エジプト分数を純粋に数学の目で眺めると何か発見できないだろうか。たくさん計算練習をして、引く分数を $\frac{1}{2}$、$\frac{1}{3}$、$\frac{1}{4}$、...としていくと、初め分子は必ず負の数になって途中から正の数になることに気づくはずだ。そして分子は必ず...、-3、-1、1、...と変化することにも気づくだろう。だから、分子が 1 になったときに引いた数が、求めるものである。じゃあ、$\frac{2}{13}$ は何を引けばいい？そう、$\frac{1}{7}$ だ。なぜなら $\frac{2}{13}-\frac{1}{\triangle}=\frac{2\times\triangle-13}{13\times\triangle}$ で、$2\times\triangle-13=1$ を満たす△は 7 だからだ。そしてすぐ $\frac{2}{13}=\frac{1}{7}+\frac{1}{13\times 7}$ がわかってしまう。△を求める仕組みは $2\times\triangle=13+1$ を解くことだ。

$$\frac{2}{13} = \frac{1}{7} + \frac{1}{13\times 7}\,; \qquad \frac{2}{n} = \frac{1}{(n+1)/2} + \frac{1}{n\times(n+1)/2}$$

← (2 数の積)　　← (2 数の積)

求める数は、2 倍したとき 14(=13+1) になる数　...　逆に言えば、$(n+1)$ の半分の数

なんだ、この 1 行↓

```
print("2/\(n.1) = 1/\((n.1+1)/2) + 1/\(n.1*(n.1+1)/2)")
```

だけ書けば答え一発じゃん、と思ったら大間違い。たしかに、これでエジプト分数は単位分数の和にできる。で、もう一度 $\frac{2}{9}$ に戻ってみよう。$\frac{2}{9}-\frac{1}{5}$ までで答えが得られたんだよね。でも飽きずにその続きをやってみよう。

$$\frac{2}{9}-\frac{1}{6}=\frac{2\times 6-9}{9\times 6}=\frac{3}{54}=\frac{1}{18}$$

なんと、もうひとつ解があった。最初のひとつの解で満足するならプログラムは要らない。でも、すべての解を求めるならコードを書かなくちゃ。