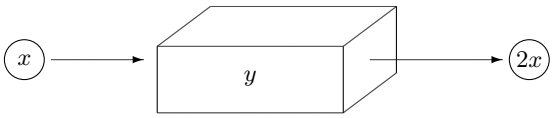
2.3 1 次関数

$y = 2x$ のような式は x と y の関係式で、もう少し丁寧に「y は x の**関数**」であるなどと表現する。関係を調べる式なのに関数と言うのは、変数のときと同様、関調と書くとお尻がムズムズするから... ではなく、関係を表す数式だからだろうか。英語では “function” と呼び、そのままの意味なら “機能” だ。それは「y は x の（値を 2 倍にする）機能」を持つからに他ならない。昔は関数ではなく、函数と書いていた。“函” は “はこ” のことであり、x を入れると $2x$ になって出てくる箱を意味する。



この函（はこ）y の機能を $y = \cdots$ で表すのである。関数には数値を代入して結果を求めることがある。以前書いたプログラム

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

を実行すると

```
x = -5 のとき、y = -10
x = -4 のとき、y = -8
        :
x = 5 のとき、y = 10
```

が出力されたことを覚えているだろう。なんら問題はないけれど、数式の計算結果に “... のとき、” などと言葉が入るのは煩わしい。数学は “1 足す 2 は 3” という表現を “$1 + 2 = 3$” として言葉を排除してきたので、“代入する” とか “のとき” などの言葉は見えないほうがよい。こういうときは関数の表現を $y = 2x$ ではなく、$f(x) = 2x$ とするとよい。f は関数の意味で “x の関数は $2x$” という言葉が “$f(x) = 2x$” になっていると思えばよい。こんなイメージである。

$$y \text{ は } x \text{ の関数} \quad \to \quad y = f(x) \quad \to \quad (f(x) = 2x \text{ だから} \ldots) \quad \to \quad y = 2x$$

$y = 2x$ と $f(x) = 2x$ は同じことだが、使い勝手がよいのは後者だ。

で、これが何の話につながるかと言えば、プログラミングにおける関数の使い方につながるのである。先の `for` 文で書いたプログラムを、関数 $y = -3x + 1$ に変えて利用したいときは

```
for x in -5...5 { print("x = \(x) のとき、y = \(-3*x+1)") }
```

とするだろう。“関数の変更” をするために “プログラムの変更” をしている。これはあまりに簡単なプログラム例なのでピンと来ないかもしれない。だが理想を言えば、“関数の変更” をするなら、プログラムは変更しないで “関数だけ変更” するのがよいのだ。

そのためには `print("x = \(x) のとき、y = \(f(x))")` となっているのが望ましい。だから `f(x)` が別の場所で定義されていれば、変更するならそっちの `f(x)` の中身を変えればよいことになる。実際には次のように書く。

[Function.playground]

```
1 import UIKit
2
3 func f(x: Int) -> Int { return -3*x+1 }
4
5 for n in -5...5 { print("f(\(n)) = \(f(x: n))") }
```

処理が簡単すぎるせいか、かえってわかりにくい例で申し訳ない。まず関数を定義したい。Swift では `func` が関数を定義するためのキーワードである。ここでは `f` という名前の関数を定義したのだが、数学のように `f(x) = -3x+1` とは書けない。Swift にとって `(` は単に文字の一種だから、この書き方で、`f(x)` という名前の変数に`-3x+1` を代入する、みたいな解釈になってしまう。したがって、基本的には `=` を用いないで `func f(x) { -3*x+1 }`と書く。関数を定義する書式は

`func` 関数名`(`変数`) {` 変数を用いた処理 `}`

のような形式をとる、というのは前に言ったと思う。処理が何行かあれば改行して書くのが普通だ。`func` で宣言することで、Swift は `(` を文字列でなく関数と変数の区切りと解釈する。

ただし Swift には数学と異なる事情が 2 点ある。数学の変数 x は実数が前提だが、Swift が使う変数には必ず型があるので、変数の後ろに型を明示しないといけない。ちなみに、関数に代入される変数は**引数（ひきすう）**という。もうひとつの事情は、数学の関数は計算結果が y または $f(x)$ の値として返ることがわかっているが、Swift では変数処理が何行にもなると何の値を返すべきかがはっきりしない。そのため値を返す場合は、返す値の型を `->` を用いて`{ }`の前に記述し、どこで値を返すかを`{ }`内に `return` を用いて書くのである。したがって関数のより正確な形式は

`func` 関数名`(`引数`:` 型`) ->` 型 `{ return` 引数処理の結果 `}`

である。

いま「関数が値を返す場合は」と表現したように、値を返さない関数というのもあるし、引数が不要の関数もある。そのような関数は単に「`func` 関数名 `() {` 処理 `}`」のように定義される。以前利用した `arc4random()` は、引数は取らないで値を返す関数である。

さて、ここでは関数の様子をグラフに描くプログラムを提示しておこう。Xcode の File メニューから、New ▶ Playground... をたどって FunctionView という名前のファイルを作ってみよう。`//__` は続けて 1 行で書くこと。ただし Xcode のバージョンによっては、おそらく `CAShapeLayer()` を使用しているせいでエラーとなるようだ。その場合は、New ▶ Project... をたどって Project ファイルを作ろう。その際、Project ファイルに不要な `// *` の行は削除してほしい。

[FunctionView.playground]

```
 1 import UIKit
 2 import PlaygroundSupport // *
 3
 4 class ViewController: UIViewController {
 5
 6     var iTF = [UITextField()]
 7     let W: CGFloat = 400 // 画面サイズ（横）
 8     let H: CGFloat = 400 // 画面サイズ（縦）
 9     let E: CGFloat =  20 // 1 目盛サイズ
10
11     override func viewDidLoad() {
12         super.viewDidLoad()
13         // x-y 座標
14         let gline = UIBezierPath()
15         gline.move(to:    CGPoint(x: 0, y: H/2))
16         gline.addLine(to: CGPoint(x: W, y: H/2))
17         gline.move(to:    CGPoint(x: W/2, y: H))
18         gline.addLine(to: CGPoint(x: W/2, y: 0))
19
20         let axis = CAShapeLayer()
21         axis.strokeColor = UIColor.black.cgColor
22         axis.path = gline.cgPath
23         view.layer.addSublayer(axis)
24
25         let OLabel = UILabel(frame: //__
                                CGRect(x: W/2, y: H/2, width: 12, height: 14))
26         OLabel.text = "O"
27         view.addSubview(OLabel)
28         // テキストフィールド
29         let labelA = UILabel(frame: //__
                                CGRect(x: 10, y: 25, width: 100, height: 25))
30         labelA.text = "y = -----x +      "
31         view.addSubview(labelA)
32         iTF = []
33         iTF += [UITextField(frame: //__
                               CGRect(x: 40, y: 10, width: 35, height: 25))]
34         iTF += [UITextField(frame: //__
                               CGRect(x: 40, y: 40, width: 35, height: 25))]
35         iTF += [UITextField(frame: //__
                               CGRect(x:105, y: 25, width: 35, height: 25))]
```

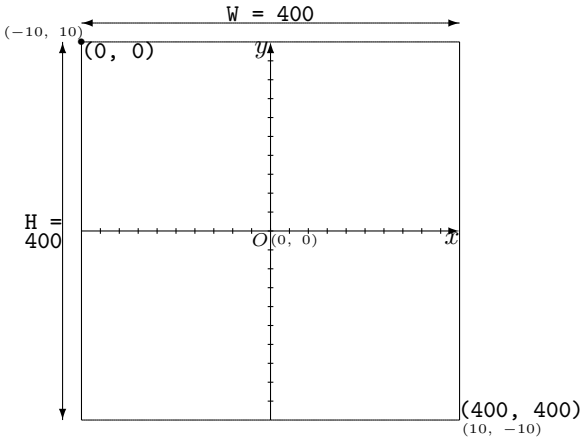
```
36          for tf in iTF {
37              tf.borderStyle = .roundedRect
38              tf.addTarget(self, action: //__
                                  #selector(drawLine), for: .editingChanged)
39              view.addSubview(tf)
40          }
41      }
42
43      let graph = CAShapeLayer()
44      @objc func drawLine() {
45          let PL = makePointOf(x:-10.0)
46          let PR = makePointOf(x: 10.0)
47
48          let line = UIBezierPath()
49          line.move(to: PL); line.addLine(to: PR)
50          graph.strokeColor = UIColor.blue.cgColor
51          graph.lineWidth = 1.0
52          graph.path = line.cgPath
53          view.layer.addSublayer(graph)
54      }
55
56      func makePointOf(x: Float) -> CGPoint {
57          let m = (iTF[0].text! as NSString).integerValue
58          let n = (iTF[1].text! as NSString).integerValue
59          let b = (iTF[2].text! as NSString).integerValue
60          if n != 0 {
61              let y = Float(m)/Float(n) * x + Float(b)
62              return CGPoint(x: W/2 + CGFloat(x) * E, //__
                               y: H/2 - CGFloat(y) * E)
63          } else {
64              return CGPoint(x: 0, y: 0)
65          }
66      }
67  }
68
69  let viewController = ViewController()                    // *
70  viewController.view.backgroundColor = UIColor.white      // *
71  PlaygroundPage.current.liveView = viewController         // *
72  PlaygroundPage.current.needsIndefiniteExecution = true // *
```

この頃はコードの量が多くなってきて大変である。関数の話をする前に、ビューにグラフを表示する処理について述べておこう。ビューにはテキストフィールドも配置して、$y = \frac{m}{n}x + b$ の係数などを入力するようにしているが、これは前節とまったく同様なので説明は要らないね。

最初にグラフを表示する画面のサイズ等を決めている。7–9 行目の `W`, `H`, `E` がそれだが、`CGFloat` が新登場の型だ。でも、これは前の節からこっそり使っていた。テキストフィールドやラベルを配置するのに用いた座標がそうだ。[FunctionView.playground] でもあちこちに使われているのが

見えるね。Swift では Float でも CGFloat でも実数値という点で違いはないのに、数値計算用とグラフィック用で異なる型にしている。このほうが安全なんだろうか。

さて、とにかく私たちがグラフを描こうとしている座標系はこんな感じだ。



関数のグラフを描くときは一般に数学でおなじみの x-y 座標を使う。しかし Swift の座標系はビューの左上角が (0, 0) である。そのため同じものを異なる座標で見る目が必要だ。

座標軸は直接ビューに描くので Swift の座標系を用いる。描き方はテキストフィールドを配置したときと似ている。つまり「定義して → 属性を与えて → 配置する」だ。定義というのは座標軸がどのようなものかを決めることで、14 行目で線を引くための変数 gline を UIBezierPath() で初期化した。線の引き方は「位置を指定して/線を付け足す」だ。15–18 行目の gline.move() と gline.addLine() を見ればわかるだろう。ただし、gline は単に “線の引き方” を示しただけで、実際に線を引いているわけではないことに注意しよう。座標軸を描くためには「何色で/どんな太さで/どんな引き方で/実線か破線か/...」等々を指定してから配置しなければならない。20–23 行目がそれだ。

まず座標系を描く対象に変数 axis を CAShapeLayer() で初期化した上で、axis に属性を与えてから配置している。axis は変数というより “台紙” のイメージだろうか。台紙に線の色や太さを決めて gline を載せるみたいな。原点 O を表示したければ、これまた「定義して → 属性を与えて → 配置する」のだ。ちょっと面倒だね。29–40 行目にかけてテキストフィールドも配置したら、いよいよグラフを描くプログラムに取り掛かろう。グラフはテキストフィールドが編集されたときのアクションとして描くつもりなので、テキストフィールドの属性から関数 drawLine を呼ぶことにしてある。43 行目以降を見てもらいたい。

直線のグラフは座標軸と異なる線であるから、グラフを描く対象に変数 graph を CAShapeLayer()

で初期化した上で、座標軸を描いたときと同じく属性を与えている。これまた台紙 graph に line を載せるイメージかな。このとき変数 line には、線の引き方として何を示せばよいだろうか。直線は 2 点の座標で決まるから、画面の両端に当たる $f(-10)$、$f(10)$ の位置を示せば事足りる。だから感覚的には

```
gline.move(to:    CGPoint(x:-10, y: f(-10)))
gline.addLine(to: CGPoint(x: 10, y: f( 10)))
```

としたいのだが、これでは数学の座標軸の値で線を引くことになってしまい、Swift のビュー上ではとんでもない場所に線が引かれることになってしまう。そこで数学座標を Swift 座標に変換する機能が要るのだ。それが `func makePointOf(x: Float) -> CGPoint` である。

私たちがすべきことは、$x = -10$ のときの y の値を計算した点 $(-10,\ y)$ を Swift 座標に直した点 `(X, Y)` に変換することである。そのために目的の関数は、実数値 x を受け取って Swift 座標 `(X, Y)` を返す仕様にする必要がある。`makePointOf(x: Float) -> CGPoint` は、たしかにそうなっているね。関数内の処理は、以前、連立方程式を解いたときの処理に似ているかもしれない。数学座標の x の値から Swift 座標の `(X, Y)` を作れば解決だ。それには次のようにする。

まず x の値から y の値を求めるのは、$y = \dfrac{m}{n}x + c$ に代入すればよい。テキストフィールドから得た値 m, n, b は整数値なので、y を実数値で求めるには `Float(m)/Float(n) * x + Float(b)` としよう。次に $(x,\ y)$ を `(X, Y)` に変換しないといけない。Swift 座標は数学座標に対して、1 目盛サイズは E 倍大きく、中心 `(W/2, H/2)` の長さだけずれている。また Y 座標は数学座標とは正負が逆である。このことから

```
(X, Y) = (W/2 + CGFloat(x) * E, H/2 - CGFloat(y) * E)
```

となることはよいだろうか。

数学でもそうだが、プログラミングをするとき常に気にかけておくことがある。

0 除算を避けること

である。とくに今回はテキストフィールドを編集するたびに `makePointOf()` が呼ばれるので `n = 0` になる可能性は非常に高い。そのための `if` 文だが、そのときの処理をどうするかは考えどころだ。ここでは関数が座標を返す仕様なので `(0, 0)` を返すことにした。つまり分母のテキストフィールドに入力がない限り、$x = -10$ のときの座標も $x = 10$ のときの座標も $(0,\ 0)$ になるということだ。結局、直線は引かれないことになる。

ex. 1 数学では、なぜ 0 除算がダメなのか正しく理解しているだろうか。あやふやなら、確認しておくとよいだろう。