中2数学の砂場

2.1 式の計算

$3 \times 4 \times x \times y$ が $12xy$ であることは問題ないね。ところが Swift の立場では `3*4*x*y` はそう見えない。`x` と `y` に値—たとえば `x=5`、`y=6`—が代入されていると `3*4*x*y` は `360` だが、`" "`で囲まれているなら`"3*4*x*y"`なのである。器用に 3×4 だけ 12 にしたりしない。だったら Swift に手を差し伸べてあげなくちゃ。早速 Xcode の File メニューから New ▶ Playground... をたどって新しいファイルを作ろう。ここではファイル名を TextView にしてみた。

[TextView.playground]

```
 1 import UIKit
 2 import PlaygroundSupport // *
 3
 4 class ViewController: UIViewController {
 5
 6     override func viewDidLoad() {
 7         super.viewDidLoad()
 8
 9         let aText = UITextField(frame: //__[続けて 1 行で書く]
                                CGRect(x: 20, y: 20, width: 50, height: 25))
10         aText.borderStyle = .roundedRect
11         //（挿入予定地 1）
12         view.addSubview(aText)
13     }
14
15     //（挿入予定地 2）
16
17 }
18
19 let viewController = ViewController()                  // *
20 viewController.view.backgroundColor = UIColor.white    // *
21 PlaygroundPage.current.liveView = viewController       // *
22 PlaygroundPage.current.needsIndefiniteExecution = true // *
```

不明なことが多くても、とりあえず入力してみよう。これは Show Result ボタンや Quick Look ボタンだけでなく、Live View を見ることができるプログラムだ。実行ボタンを押したあとも従

来と同じ画面しか見えてないとしたら、Xcode の Editor メニューから Live View を選択してほしい。右側に Live View が表示されるはずだ。これは iPhone や iPad の画面を模していると思ってよい。

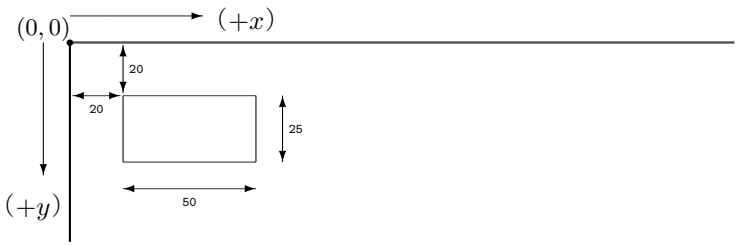
テキストフィールドがひとつ見えるだろう。つまり入力したプログラムは、テキストフィールドを iPhone や iPad に表示させるものだったのだ。ただし、いまはただの箱に過ぎないので何の処理もしないけど。なので、Live View を閉じたければ、右上の アイコンから “Show Editor Only” を選択しよう。ソースコード編集画面だけになる。

中 2 数学からは適宜テキストフィールドなどを配置するプログラムを書くことにしたい。そうすれば、Swift に任意の値を教えてあげられるからね。この時点では、行末に目印（`// *`）を付けた文があることやコード全体が `class ViewController { }`で囲まれていることが、playground で iPhone 的な画面（ビュー）を見るための処置であるとだけ言っておこう。テキストフィールドを表示させる核となるプログラムは、9–12 行目に書かれた 3 行のコード

```
let aText = UITextField(frame: CGRect(x: 20, y: 20, width: 50, height: 25))
aText.borderStyle = .roundedRect
//（挿入予定地 1）
view.addSubview(aText)
```

である。簡単に説明するので、命令に含まれる単語を見ながら「ああ、そんなもんね」程度に理解してもらえば十分だ。

先に 12 行目の `view.addSubview(aText)` を見よう。これは iPhone のビューに `aText` という名前のサブビューを配置するための命令である。`aText` は 9 行目で宣言されていて、かつ `UITextField()` が代入されている。つまり、`aText` は所定のテキストフィールドとして定義されたのである。その位置と大きさは下図のような関係になっている。画面左上が原点 (0, 0) で、y 軸は正・負の方向が数学で用いる座標軸とは逆である。



ただし `let` 文 1 行で終わりにすると、テキストフィールドは作られてもそれの場所が見えない（でも入力はできる）。そこで 10 行目において、`aText.borderStyle = .roundedRect` で `aText`

のプロパティに外枠の属性を与えている。テキストフィールドに限らず、さまざまな部品は「定義して → 属性を与えて → 配置する」が基本である。

さらに、テキストフィールドに外枠以外の属性を追加していくことにする。まず、aText に文字が入力されても活用できなくては意味がないので.addTarget() メソッドを追加しよう。これは「何（対象）に対して/何（イベント）が起きたら/何のアクション（処理）を」行うかを指示するもので

```
.addTarget(対象, action: #selector(処理), for: イベント)
```

のような書式になる。「aText.add...」とキーボードを打ち始めると Xcode が予測補助をしてくれて、可能性がある命令群の一覧がポップアップされる。この機能のおかげで入力の負担はだいぶ軽減されるはずだ。

ここでは、「aText 自身に対して/編集変更があったら/getText の実行を」行いたいので

```
aText.addTarget(self, action: #selector(getText), for: .editingChanged)
```

とする。これを（挿入予定地 1）に記述する。#selector() に指定した getText は Swift に備わった命令や関数ではなく、自分で作成するもので、とりあえず

```
func getText(sender: UITextField) {
    let a = (sender.text! as NSString).integerValue
    print(a)
}
```

として（挿入予定地 2）に記述しておこう。これはあとで少し修正を加える。なぜ挿入場所を 2 ヶ所に分けたかというと、action: #selector() の位置に処理したいコードを長々と書くわけにいかないからだ。そのため関数名だけ書いて、関数の仕事内容は別途記述するのである。

以前、arc4ramdom() が UIKit のコンポーネントで乱数を生成する機能を持った関数である、と言ったことを覚えているだろうか。関数 getText を書くことは、テキストフィールド aText が編集されたときに機能する関数を作ることに他ならない。Swift にない機能は自分で作るものなのだ。関数は基本的に

```
func 関数名 () { この関数が行う処理 }
```

という書式で定義する。関数名の直後にある () は必須である。() は関数であることを示す符丁ではなく、関数が利用する値などを受け取るために必要なのだ。第一、関数であることは func が示しているでしょ。たとえば arc4random() は関数が利用する値を与えずとも勝手に乱数を生成する。

関数の詳細はあとの節で述べることにするが、（挿入予定地 2）に記述した関数 `getText` が `getText(sender: UITextField)` と書かれているのは、この関数が値を受け取る必要があるからだ。`getText()` の機能は、与えられたテキストフィールドを丸ごと受け取り、そのうちの文字列だけを表示することである。`sender: UITextField` はテキストフィールドを `sender` という名前で受け取る、といった意味でとらえてもらってかまわない。いまテキストフィールドは `aText` の名前で作ったのだから `getText(aText: UITextField)` と書いてもよいのだが、Swift は送り手の意味である `sender` を用いるのが好みのようだ。

もし、単にテキストフィールドに入力された文字だけを表示させるなら、関数 `getText` 内に書く文は `print(sender.text)` の 1 行で十分である。しかし今回の潮干狩りでは、いずれ数値を計算等に使いたい。実はテキストフィールドは名前のとおり、たとえば数字 `1` は文字列の`"1"`であって数値 1 として見ない。そのため数値に変換する必要があるのだ。さらにテキストフィールドに文字が入力されても無効にしたいが、ここでは簡単に `0` に変換することにしておく。ただし、変換の仕方はちょっとややこしい。

a =	sender.text		aText
文字列 ab	"ab"	←	ab
文字列 12	"12"	←	12
エラー	nil	←	

たとえば `let a = Int(sender.text)` ではテキストフィールドに値がないときエラーになる。詳細は省くが**オプショナル**にすれば問題ないし、そのことは Xcode が注意喚起してくれる。いまは素直に従えばよい。ただ、`let a = Int(sender.text!)` にするだけでは結局のところ不十分で、文字が入力されるとうまくない。この場合、整数化には `Int()` ではなく `.integerValue` を用いる。ただし `.integerValue` は NSString 型のメンバーであるため、`sender.text!` は `as NSString` を付けて NSString 型へ変換しておく。これらが合わさった結果

```
a = (sender.text! as NSString).integerValue
```

という代入文になったのだ。これでようやく数字は整数で、文字も `0` で `a` に代入されることになった。やれやれ。

これでプログラムは、キーボードから 1 文字入力するたび、Debug Area にテキストフィールドに入力された文字が表示される仕様となった。`.addTarget()` の `for:` に対して `.editingChanged` を指定したのでそういう動作になる。テキストフィールドの文字を数値に変換しているため、数字以外は `0` 扱いになる。

ところが、このように **playground** でビュー表示を利用するプログラムを書くと、**Xcode** のバージョンによっては、**PC** の物理キーボードからの入力ができないとか、不明なエラーが起きるとかの不具合が生じることがある。そうなってしまったら悲しいけれど、新たな挑戦を試すチャンスだ！　格好の練習課題が与えられたと思えばいい。

ex. **1** playground で十分な動作が得られなければ、同じコードを Project ファイルとして作成すればよい。次節で簡単に説明しているので、実際にやってみよう。

しかし、テキストフィールドに入力した値を Debug Area に出力してもアプリにはならない。ビューに表示できるように部品を追加しよう。追加するのはラベルで、それに文字列を表示することにする。結局、次のようなプログラムとなった。

[TextView.playground]

```
 1 import UIKit
 2 import PlaygroundSupport // *
 3
 4 class ViewController: UIViewController {
 5
 6     override func viewDidLoad() {
 7         super.viewDidLoad()
 8
 9         let aText = UITextField(frame: //__[続けて 1 行で書く]
                                CGRect(x: 20, y: 20, width: 50, height: 25))
10         aText.borderStyle = .roundedRect
11         aText.addTarget(self, action: //__
                                 #selector(getText), for: .editingChanged)
12         view.addSubview(aText)
13     }
14
15     let labelA = UILabel(frame: //__
                               CGRect(x: 200, y: 20, width: 50, height: 25))
16     @objc func getText(sender: UITextField) {
17         let a = (sender.text! as NSString).integerValue
18         labelA.text = String(a)
19         view.addSubview(labelA)
20     }
21 }
22
23 let viewController = ViewController()                  // *
24 viewController.view.backgroundColor = UIColor.white    // *
25 PlaygroundPage.current.liveView = viewController       // *
26 PlaygroundPage.current.needsIndefiniteExecution = true // *
```

さっき書いたものと違うところは `func getText()` 内の 2-3 行目の文と、`func getText()` の前の行で `labelA` を定義したこと、そして `func` に `@objc` を付け加えたことである。

labelA を func getText() の直前、つまり関数の外にはみ出して書いたのは間違いではない。getText() はイベント発生のたびに処理が行われるので、中に書いてしまうと、その都度ラベルが定義されてしまうからである。どのみちラベルの定義は関数の外でするとの理由で、もっと前、つまり viewDidLoad() の前の行で定義しても同じだ。どちらかというと、その位置に書くほうがわかりやすかったかもしれない。

func の前に@objc を付け加えたのは、Swift に叱られたからである。@objc を書かないとエラーとなるが、その理由は Swift が英語で教えてくれて、修正も半自動でやってくれる。少々煩（わずら）わしいけど、なんと頼もしいことか。

さあ、これで自由に入力ができ、入力した値を処理できる環境が整った。今後はこれをひな型にして、機能を増やしていくつもりだ。十分とは言えないものの、アプリもどきで学習が進められるのは大変良いことだ。ああ、良き日である。