# 让不懂编程的人爱上iPhone开发(2017秋iOS11+Swift4+Xcode9版)-第9篇

欢迎回来，让我们继续iPhone开发之旅。

在上一篇的内容里面我们首次接触了算法，我也提到过还有其它方法来计算滑动条当前数值和目标数值之间的差值。之前所使用的算法虽然可以用，但它包含了八行代码。其实我们可以用更简单的方法来实现。

这里是我构想的新算法：

“首先用滑动条的当前数值减去目标数值。
如果这个结果小于0，就用它乘以-1。”
就这么简单，我们成功的把负数转换成正数，确保最后的差值总是正数或0.

小练习：
现在你可以尝试一下把上面的算法用代码来实现。

最后的结果是这样的：

```
var difference = currentValue - targetValue

if (difference < 0) {
    difference = difference  * -1
}
```

新的算法用代码体现出来就是这样的。

我们用currentValue变量的数值减去targetValue变量的数值，然后把结果保存在difference变量中。

这里我们需要注意的是，在创建新变量的时候，可以将创建变量和赋值放在一行代码中。

如果不嫌麻烦，也可以用两行代码的方式：

```
var difference: Int
difference = currentValue - targetValue
```

这两行代码的作用和刚才的一行代码是相同的:

```
var difference = currentValue - targetValue
```

我们也无需告诉编译器这里的difference类型是Int，因为currentValue和targetValue都是Int，而Swift足够聪明，它知道两者的差值仍然是Int类型的。

这个特性就是所谓的type inference（类型推断），这也是Swift相对Objective-C的一大卖点。

当我们获得了计算结果后，会使用if判断语句来判断它的值是否为负（小于0）。如果是，就让它乘以-1，然后保存这个新的结果到difference变量中。

当你用下面这行代码时：

```
difference = difference * -1
```

程序首先用difference的数值乘以-1，然后再把计算的结果重新保存到difference中去。实际的效果是，新计算的数值覆盖了difference之前的内容。

当然我们还有一种更简单的方法：

```
difference *= -1
```

*=操作符把*和=放到一个单独的操作里。最后的结果是相同的：变量中保存的旧数值被计算结果所替代。

同样，我们还可以用下面的代码来实现这个算法：

```
int difference = currentValue - targetValue
if  difference < 0 {
    difference = -difference
}
```

这里我们不是用-1乘以difference变量的数值，而是直接用取负操作符。这是因为负数的负数必然就是正数了。如果你不相信我，可以回小学请教下你的老师~

好了，让我们试试新的算法吧。

在Xcode中切换到ViewController.swift，找到showAlert()方法，用下面的代码替代：

```
    @IBAction func showAlert(){

        var difference = currentValue - targetValue
        if difference < 0 {
            difference = difference * -1
        }
```

```
    let mesage = "滑动条的当前数值是：\(currentValue)" +
                "\n目标数值是： \(targetValue)" +
                "\n两者的差值是： \(difference)"

    let alert = UIAlertController(title:"Hello Messi",
                                  message:mesage,
                                  preferredStyle: .alert)
    let action = UIAlertAction(title:"ok",style: .default,handler: nil)
    alert.addAction(action)

    present(alert, animated: true, completion: nil)

    startNewRound()
}
```

现在来点击Run运行游戏，你会发现结果和之前没有任何变化，只不过我们换了一种实现方式而已。

除了上面的方法，我们还有第三种方式，也就是用函数来处理这个问题。

之前我们已经接触了两个函数，分别是用来取整的lroundf(),以及用来生成随机数的arc4random_uniform()。

为了确保一个数字永远是正数，我们可以用abs()这个函数，它的作用是获取当前数字的绝对值。

只要你成功从小学毕业了，应该知道绝对值是什么意思吧，哥就不用介绍了吧。

使用这个函数，可以让我们的算法实现代码进一步缩减为一行代码：

```
let difference = abs(targetValue - currentValue)
```

这个时候showAlert方法的代码就会更加精悍：

```
@IBAction func showAlert(){

    let difference = abs(targetValue - currentValue)
    let message = …

}
```

abs()这个函数很有用，在实际的编码中也会经常碰到。

**变量和常量：**

仔细观察下刚才的代码，还有什么新的变化?
你会看到这里使用let关键词，而不是var。

let和var究竟有什么区别呢？在Swift编程中，变量(variable）和常量(constant)之间有很大的差异。和变量不同，常量的数值不会发生变化。

比如当我们给某个常量赋值后，就无法赋予它新的数值。

关键词var会创建一个变量，而let则创建一个常量。这里我们使用let来定义difference，就意味着它是一个常量。

在之前的算法中，difference的数值可能会改变。如果是负数，那么我们让它变成一个正数。在这种情况下，difference就必须是一个变量，因为只有变量才能赋予新的数值。

现在我们可以用一行代码来完成基本的计算。一旦我们给difference赋值，它的数值就不会发生变化。在这种情况下，使用let显然更加恰当。（为什么更好呢？这样做可以让我们的意图更加清晰，从而让Swift编译器更好的理解程序。）

同样，我们所定义的message,alert和action也是常量。现在你大概明白为什么我们使用let而非var来定义对象了。因为一旦它们被赋予数值后，就没有必要进行更改了。

常量在Swift中非常普遍。通常我们需要短暂保存某个数值。如果在这个时间段内该数值不需要发生变化，那么最好使用let将其定义为常量，而不是使用var定义成变量。

计算玩家得分

好了，既然我们已经有了差值，就可以根据它来计算玩家的得分。

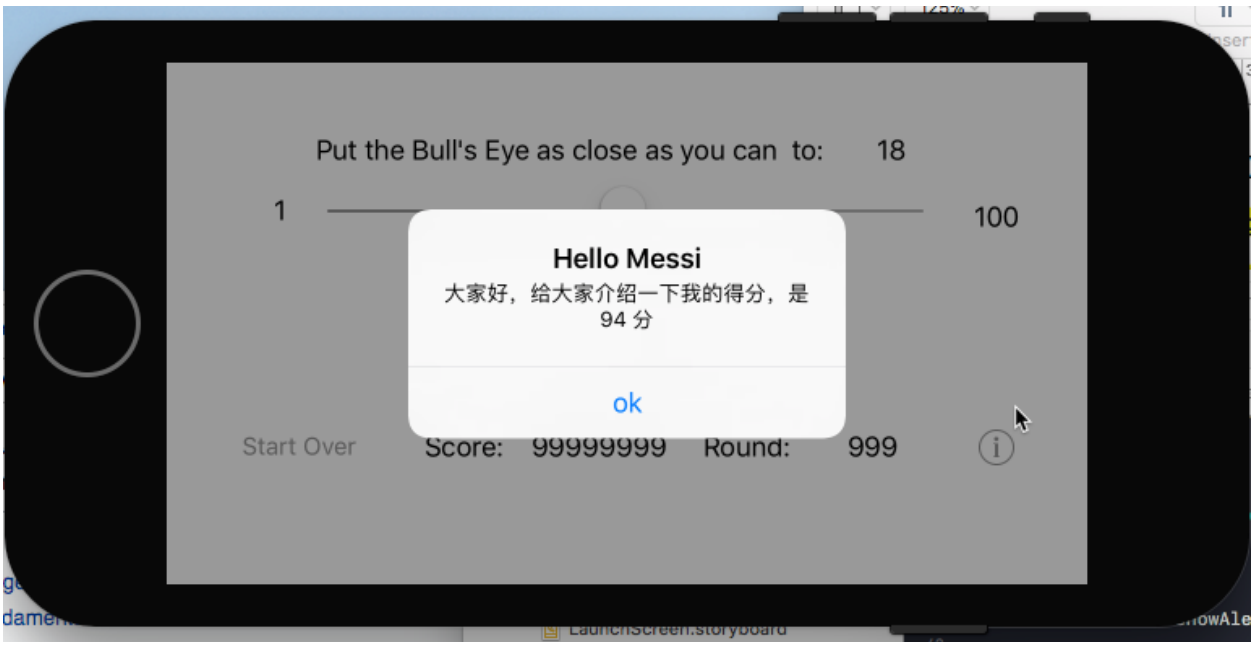
再次修改showAlert()方法的代码如下：

```
@IBAction func showAlert(){
    let difference = abs(targetValue - currentValue)
    let points = 100 - difference
```

```
        let message = “大家好，给大家介绍一下我的得分，是 \(points) 分"
        …
    }
```

玩家的最高得分是100，也就说你猜的数值和目标数值完全相同。

运行应用看看你能得多少分




小练习：

因为滑动条的最大位置是100，最小位置是1，所以最大差异是100-1=99。这就意味着玩家最差的得分是1.
想一想为什么？（好吧，这个需要小学数学）

**继续前进-保存玩家的总得分**

在to-do 清单中，我们希望把玩家的总得分显示在屏幕上。在每一轮游戏结束后，我们都会把当前回合的得分添加到总得分中，然后更新得分标签。

因为我们需要长期保存总的得分，所以需要把它保存在实例变量中。

让我们回到Xcode，打开ViewController.swift，添加一个新的score实例变量的定义：

```
class ViewController: UIViewController {

    @IBOutlet weak var slider:UISlider!
    @IBOutlet weak var targetLabel: UILabel!

    var currentValue: Int = 50
    var targetValue: Int = 0
    var score = 0
    ……
```

且慢稍等？为毛这个score变量的定义和其它两个看起来不太一样啊。是的，我们没有指定变量类型Int。

这里其实又用到了类型推断(type inference)的概念。当我们没有特别指定某个数据类型时，Swift将会使用type inference来推断某个变量的类型。因为0是整数，Swift假定score应该是一个整数，因此自动赋予类型Int。

小技巧：
如果不能确定变量的推断类型，有一个简单的方法可以帮我们。按住键盘上的Alt键，然后将鼠标放到有疑问的变量上。此时该变量会被蓝色高亮，同时光标会编程一个问号。此时点击变量，就会看到一个弹出式的说明，其中显示了变量的类型，以及该变量是在哪个源文件中被声明的。大家可以试试看哦~

实际上之前的两个实例变量也无需特别指定变量类型Int，只需要改写成这样：

```
var currentValue = 0
var targetValue = 0
```

Swift同样会使用类型推断来确定这两个变量的类型。
只有当我们没有给变量赋予初值的时候，才需要特别指定变量类型。这就是Swift中使用类型推断的好处。

试着修改代码看看，真的管用！

现在我们可以来修正showAlert()方法，从而更新score得分标签。

更改showAlert()方法的代码如下：

```
  @IBAction func showAlert(){

        let difference = abs(targetValue - currentValue)
        let points = 100 - difference

        score += points

        …
    }
```

上面的代码没有什么好奇怪的，我们只是添加了黄色高亮部分的一行代码：

```
  score += points
```

它的作用是把玩家在本轮游戏的得分添加到总得分里面去。其实这行代码完全可以这样来写：

```
score = score + points
```

作用完全相同，不过我更喜欢精简一点的写法。

好了，继续前进。

**在屏幕上显示玩家得分**

现在该是在屏幕上显示玩家得分的时候了。

我们要做的事情其实很简单，和之前的目标数值标签一样，我们需要把得分标签关联到一个属性上，然后在标签的文本中显示得分。

小练习：试着看自己来操作一下。

首先在ViewController.swift中添加一行代码：

@IBOutlet weak var scoreLabel: UILabel!

接下来就是在用户界面元素和视图控制器刚定义的outlet变量之间创建关联了。
还记得该如何创建吗？其实有几种不同的方式，作用是完全一样的：
首先要打开Main.storyboard，然后：

- 按住Ctrl键，点击对象，会看到一个弹出菜单，然后从New Referencing Outlet拖一条线到View Controller（对付滑动条的时候我们就是这么干的）
- 或者是选择标签对象，然后在Xcode的右侧切换到Connections Inspector，从New Referencing Outlet拖一条线到View Controller（对targetLabel我们是这么干的）
- 或者是按住Ctrl键，从View Controller拖一条线到这个标签。但是不要反过来了（这次你可以这么试试看）。

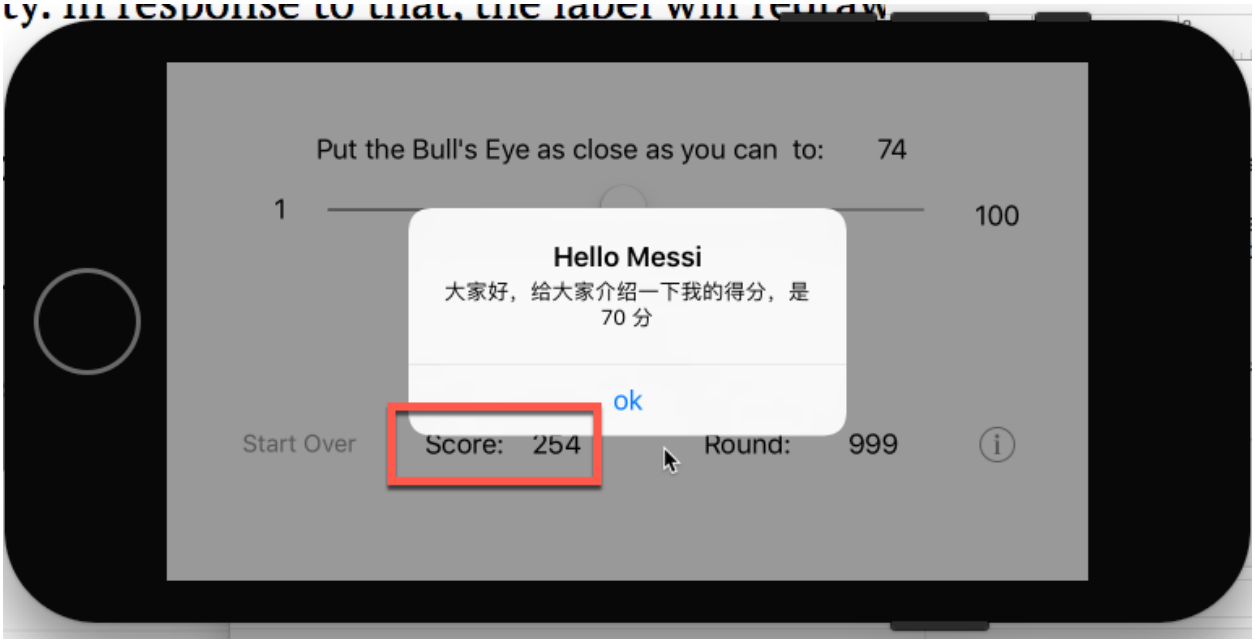
通过上面的操作，我们就有了一个绑定到界面元素的scoreLabel变量，可以往里面放一些文字。
好了，那么在代码里面我们该做点什么呢?

回到ViewController.swift，然后更改updateLabels()方法如下：

```
func updateLabels(){
    targetLabel.text = String(targetValue)
    scoreLabel.text = String(score)
}
```

这个也不算新东西了。我们将Int类型的分数强制转换成String字符串，然后把它赋予标签的text属性。实际执行的时候，标签会使用新的得分更新自己的文本内容。

编译运行游戏，试试看总的得分是不是对的。




**One more round…（再来一回）**

乔布斯时代，每次苹果发布会最期待的就是One more thing…
因为在这之后，我们就有希望看到一款令人‼️的新产品上市

进入Cook时代，我记忆中只有发布Apple Watch和今年发布iPhone X的时候才又有机会听到这熟悉而又陌生的One more thing…
只不过，Cook时代的One more thing含金量比起帮主时代还是颇有逊色。

好了，不瞎扯了，回到我们的游戏开发。

对于游戏开发者来说最大的慰藉就是，玩家希望一次次的再来重新体验。

而说到再来一回，实际上我们还得在每次开启新一轮游戏的时候更新回合数。

小练习：试着自己来尝试一下，看看之前学的东西管不管用。

提示：

先不急着写代码，想在纸面上或者头脑中想想要做哪些事情。

- 首先要把当前的回合数（起始数是1）保存起来
- 然后每个新的回合开始时让它加1.
- 然后要把相应的回合数显示在对应的标签上。

聪明如你应该已经想到我们该怎么写代码了。

你可能已经想到了我们需要定义一个新的实例变量。

恭喜你猜对了。首先我们需要在定义其它实例变量的代码后面添加一行新代码：

```
var round = 0
```

当然，如果你是个处女座（别拍，求轻虐。。。），你很可能用下面的写法：

```
var round: Int = 0
```

好吧，处女座不要来寻仇，开个玩笑而已。

接下来要添加一个新的outlet标签对象。

```
@IBOutlet weak var roundLabel: UILabel!
```

**提醒：别忘了创建关联**

很多新手都会忘了在Interface Builder里面创建关联，这是非常常见的错误。很多时候我创建了一个按钮属性，然后也写了当玩家触碰按钮时应该如何处理的代码。结果当游戏跑起来的时候，发现按钮没用。通常这要浪费哥几分钟时间来冥思苦想，到底哪里出了错。实在是浪费生命啊。

至于具体怎么做我就不想多废话了，如果你不知道该怎么做，建议回头看看前面的操作。

最后让我们更改updateLabels()方法的代码如下：

```
func updateLabels(){
    targetLabel.text = String(targetValue)
    scoreLabel.text = String(score)
    roundLabel.text = String(round)
}
```

好吧，还有一件事别忘了，我们需要让round变量的数值加1。在哪里加呢?

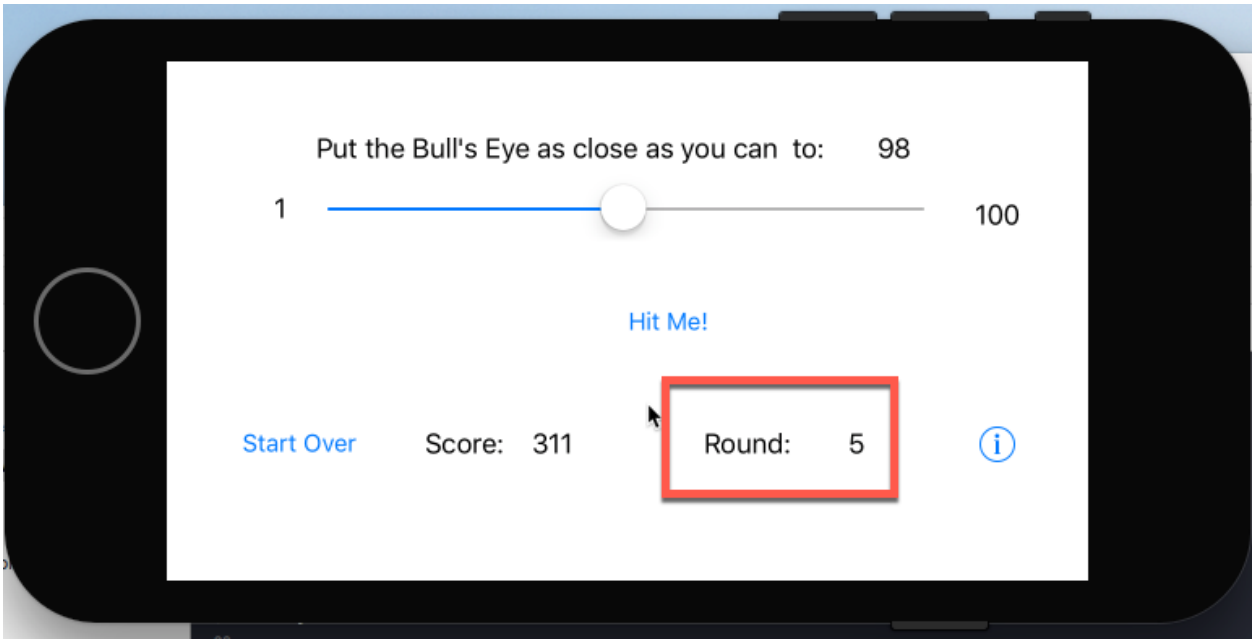
我认为startNewRound()方法是个不错的地方。毕竟每次开启新一轮游戏的时候我们都会调用这个方法。

在ViewController.swift中找到startNewRound()方法，更改其中的代码如下：

```
func startNewRound() {
    round += 1
    targetValue = 1 + Int (arc4random_uniform(100))
    currentValue = 50
    slider.value = Float(currentValue)
    updateLabels()
}
```

有一件事哥忘了告诉你。实例变量的默认数值是0。也就是说，当应用开始跑起来的时候，回合数round变量的初始值是0.而当我们调用startNewRound()方法的时候，都会让回合数变量的值加1，所以第一回合就会显示1。

点击Run按钮来运行游戏，现在就会看到游戏回合数每次都会自动增加。




到了这一步，基本的游戏逻辑都已经实现了。我们可以先小小的庆祝一下顺利，休息一下。

老规矩，上图！
这次献给大家的是AKB48，宅男最爱的秋叶原超级组合。


AKB48