让不懂编程的人爱上iPhone开发(2017秋iOS11+Swift4+Xcode9版)-第8篇

又到了继续学习iPhone开发的时间了。
这一次我们先来点纯理论知识打打基础吧，千万别打瞌睡。

术语党科普：静态/动态类型语言、强/弱类型定义语言、脚本语言/通用语言，解释型语言/编译语言

相信大家经常被这些术语迷惑吧？这里又到了术语党科普的时间了。

这里的语言不是指英语，汉语，法语这种人类沟通语言，而是指计算机编程语言。

如果你英文好的话可以忽略掉下面的这段文字，直接看wiki的解释：

http://en.wikipedia.org/wiki/Type_system

当然知乎上也有这个话题的讨论：

http://www.zhihu.com/question/19918532

静态编程语言和动态编程语言主要是针对高级语言来说的。在编程语言的发展史上，经历了直接面向硬件的机器码，以及稍微人性化一点的汇编语言，再到所谓的高级语言（人类可以比较容易看懂的语言）。在高级语言中，则是从最初的静态类型语言发展到今天比较轻量级的动态类型语言。

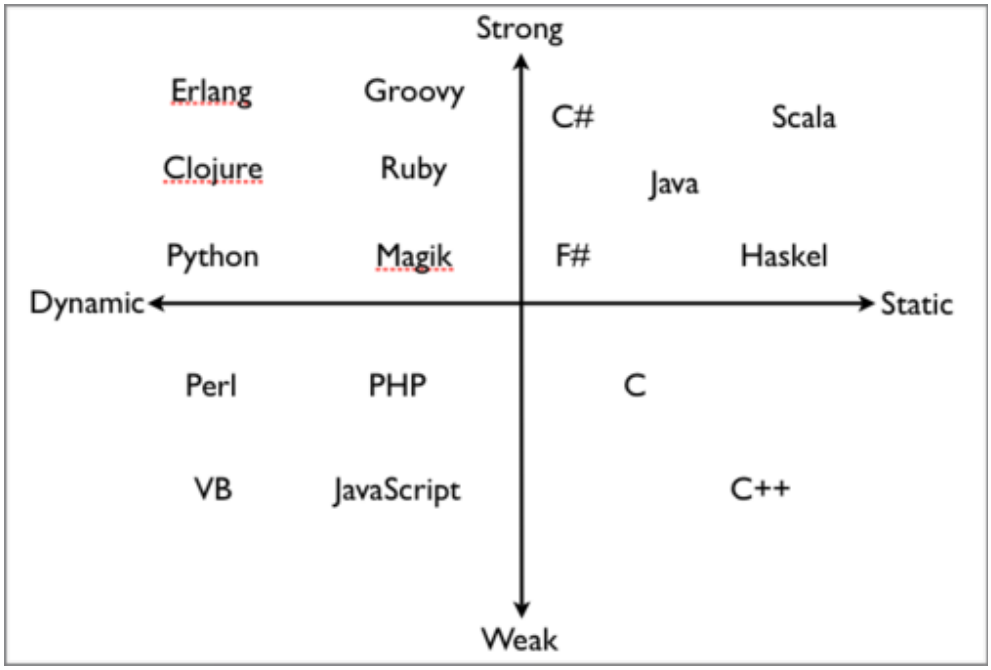
所谓的静态和动态是针对变量的数据类型而言的。使用静态类型语言编写的代码中，要声明变量的数据类型，而且不同数据类型的变量不允许直接赋值，它的数据类型是**编译期间**进行检查的。同时，在使用变量之前就需要为它们分配好内存控件。最典型的静态类型语言就是C,C++,C#,Objective-C,Java。这几种语言的共同特点可以说是功能非常强大，从操作系统到一般的应用，从网络通讯到客户端界面，都可以完全搞定。而且静态语言的结构非常规范，方便调试。但世上难有两全事，静态语言功能虽强大，但相对难以学习，而且灵活性也比较差。另外需要写更多的类型相关代码，让程序显得很臃肿。

静态语言的弱点恰恰是动态语言的强项。所谓的动态类型语言是指在**程序运行期间**才去进行数据类型检查的语言。也就是说，使用动态语言编程时，不需要给变量指定数据类型，它会在你第一次赋值给变量时，在内部记录数据类型。典型的动态类型语言如Python,Ruby,PHP,JavaScript,Erlang等。

不过动态语言虽然快速灵活，但付出的代价却是，在代码运行前很难找到bug。所获得是

开发速度的提升，所失去的是程序的健壮性。

再来看看强类型定义语言和弱类型定义语言。



所谓的强类型定义语言是指强制数据类型定义的语言。也就是说，一旦某个变量被指定了某个数据类型，如果不经过强制转换，它就永远是这个数据类型。

弱类型定义语言当然是相对的，也就是说一个变量可以赋予不同数据类型的值。
比如，Python是动态语言，同时是强类型定义语言，Java是静态语言，也是强类型定义语言。Objective-C显然和Java类似。

然后来看什么是脚本语言?

脚本语言(scripting language)也是编程语言，它是相对通用性语言来定义的。它们在设计之初不是用来开发独立的应用程序，而是作为嵌入式的语言，用于和其它语言编写的程序组件之间建立通信。脚本语言都有相应的脚本引擎，需要解释器才能执行，而不能编译成可执行文件形式的二进制代码。
脚本语言通常语言简单，可以用文本形式保存，不需要编译成目标程序，在调用的时候直接解释。以JavaScript为例，只需要用记事本新建一个html文件，在里面添加相应的脚本就好了。当使用浏览器打开html文件时，会自动调用里面的JS脚本。
如今的很多脚本语言早就超越了初期的功能，成熟到可以编写独立而精巧的程序，但习惯上我们还是称之为基本语言。

常见的脚本语言有Python,vbscript,javascript,actionscript,Lua,Perl,PHP,Ruby等。

最后来看编译型语言和解释型语言的区别：

使用编译型语言写的程序在执行之前需要用编译器编译成二进制码，并生成独立的可执行文件，以后运行的时候不需要重新编译。比如C,C++,Objective-C都是如此。
解释型语言写的程序在运行的时候才翻译，通常需要一个解释器或者虚拟机对执行的语句进行解释。比如basic,Java等。不过Java语言有点特殊，是编译成字节码，然后用虚拟机来解释执行。
静态/动态，强类型/弱类型，脚本语言/通用语言，解释型/编译型语言都是独立的概念，彼此存在交集，而不是完全相同的概念。

具体到Swift语言，可以说Swift = 静态语言（带动态特性）+ 强类型+通用语言+ 编译型

从性能上来看Swift是个非常强大的语言，但是从很多语法习惯上又和Python和javascript有很多类似之处。

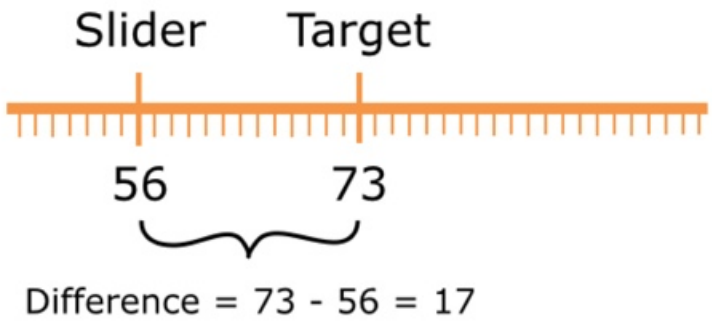
科普结束，相信作为非程序猿的你对上面的术语已经多少有点糊涂了。不过没关系，只要头脑中有个基本的概念就好。随着你编程经验的不断增长，以后就会逐渐明白的。为了安慰下你受到惊吓的心，还是先来点福利吧。

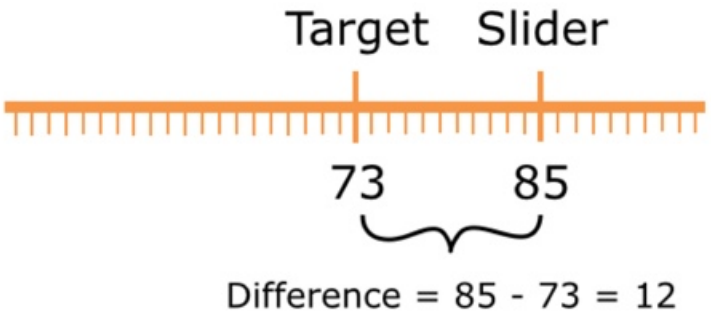
好了，现在你应该已经从术语的噩梦中脱身了，接着让我们继续前进。

计算玩家得分

在之前的学习里面，我们已经用随机数来设置了目标数值，同时也知道如何来读取滑动条结点对应的数值，现在就可以来计算玩家的得分了。滑动条上结点对应的数值和目标数值越接近，玩家的得分就越高。

为了计算当前回合的玩家得分，我们需要比较滑动条的数值和目标数值之间的差距：





最简单的方法当然就是直接用currentValue（滑动条的当前数值）减去targetValue（目标数值）。

不过， 当滑动条的结点位置的数值小于目标数值时，得到的差值就是负数。

这个时候我们就得把负数转换成整数。那么如果用targetValue减去currentValue可以吗？一样的，如果这样来计算，当滑动条的结点位置的数值大于目标数值时，得到的差值就是负数。

小练习：想想看，如果我们用人类的语言来描述这个问题，该如何接近呢？先不要想怎么用代码实现，先用普通话来描述一下怎么来解决这个问题。

好吧，哥又要唠叨一回了。在写代码之前，先想想在现实生活中会怎么解决这个问题，先用自己的语言来描述一下问题的解决方式，不管是四川话，普通话，广东话还是英语。如果你都没想清楚，那么写下来的代码肯定也是逻辑混乱一团糟。

我是这样来想的：

“如果滑动条的当前数值大于目标数值，那么它们之间的差值应该是滑动条的当前数值减去目标数值。
反过来，如果目标数值大于滑动条的当前数值，那么它们之间的差值应该是目标数值减去滑动条的当前数值。
最后，如果两个数值是相同的，那么它们之间的差值就等于零。”

用上面的方式来计算，无论两个数值的关系如何，最终的结果总是正数或0，因为我们始终用较大的数值减去较小的数值。

好吧，现在可以来点小学数学计算了。如果滑动条结点的位置在60，目标数值是40，那么差值就是60-40= 20。如果滑动条结点的位置在10，而目标值为30，那么差值就是30-10 = 20。

看，就这么简单。

术语党科普：算法

看到这两个字，很多人都要心惊胆战。莫说是产品和设计人员，哪怕是个资深的程序猿，听到算法这两个字都要肃然起敬，敬之如鬼神。传说中很多程序猿在面试的时候都因为算法题目不过关而被拒之门外。那么，作为非程序猿的你，是否看到这两个字就要打道回府，从此不再学编程了呢？

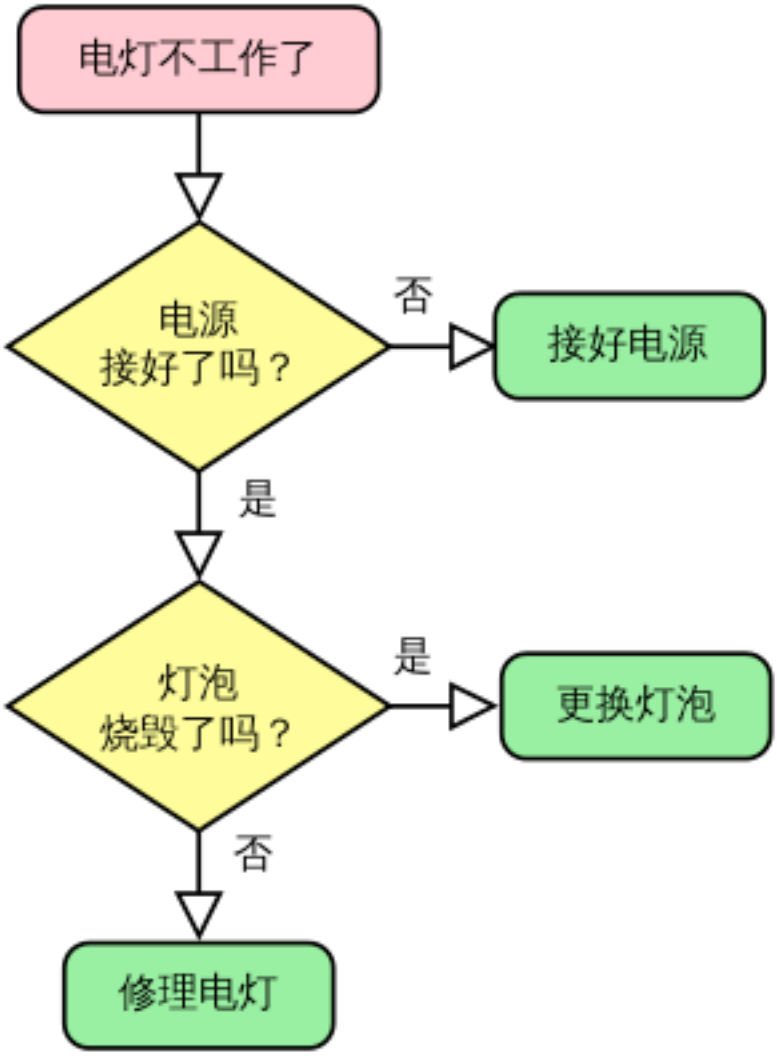
肿么办？不要忘了，在《苦逼程序猿的生存指南》这本书的第1页和最后1页都有同样的一句话。Don’t Panic~不要恐慌！

正如很多绝症患者是看到医院的诊断书后自己被自己吓死的一样，很多准程序猿也是被代

码世界里面的各种看似高深无比的术语给活活吓死的，从此和代码世界无缘。

那么，究竟神马是算法？如果哥告诉你刚才我们讨论的东西就是算法你会信我吗？不管你信不信，我反正是相信。通俗点说，算法就是用来计算某个计算问题的数学步骤。从这个定义出发，我们刚才说的加减法就是算法，只不过是比较简单的算法而已。

为了让你不再恐惧，下面再给个简单是算法流程示意图：



这也叫算法？是的，不要把算法看成洪水猛兽，它只是为了帮助我们解决某个特定的问题。

事实上，今天的苦逼程序猿要比上世纪的程序猿幸福的多。原因在于，在计算机科学的发展过程中，已经有很多NB人物为我们研究了各种各样的或高深或浅显的算法。实际上，

大学里面搞计算机研究的大致分两派，一派专门研究计算机的硬件设计与开发，一派美其名曰计算机软件工程，实际上都是在研究新的算法。

虽然在大学计算机系的课程里面，算法和数据结构算得上是真正的挂科屠夫了，但如果你不是想搞研究，或是进行一些大型的科学计算，大可以直接运用现成的很多算法。而且这些算法都有对应主流编程语言的源代码可供参考。

比较著名的算法有冒泡法，贪心算法，递归法，迭代法，分治法，动态规划法，分支限界法，回溯法，A*寻路算法等等。

在实际使用的过程中，通常我们要做的不是发明一套自己的新算法，而是根据所遇到的问题选择使用已有的算法。当然，一些比较简单的问题，比如我们这里的加减运算，我们可以直接绘制一个流程图来理顺自己的思路。

当然，我的意思不是说算法不重要，而是说要循序渐进，即便是研究算法也是。目前学校教育的错误方式是，先塞给你一些东西，然后让你学习怎么用，然后就没有然后了。

个人觉得真正重要的是了解这些算法是怎么产生的，如果你不清楚这个思维的过程，就是懂1万种算法也没有用。因为实际生活中会遇到无数种在课本上碰不到的问题，面对小毛贼砍过来的一刀，你是该用落英缤纷掌还是大日如来掌？真正有效的做法是根据实际情况见招拆招，把现有的算法结合起来，甚至需要创造出新的算法。

那么，当我们遇到某个需要通过计算才能解决的问题时，该如何下手呢？还是那句话，在开始写代码之前，先动脑筋思考一下。先试着用自己的语言把这个问题描述清楚（当然是在大脑里面，不是让你对着mm说出来），然后在纸上把解决问题的思路用流程图的方式画下来。如果一时想不出来，可以去看看NB人物已经设计好的算法。客观的讲，太阳底下没有新鲜事，以我们目前水平所遇到的问题，前人百分99.99999都已经遇到过，大可以去直接借鉴。当然有一个前提是，即便是借鉴别人的算法和解决思路，我们自己也需要完全理解，起码要能看懂。

所以，写代码和写长篇小说有点类似，如果你在哪一块卡住了，不要对着电脑发呆崩溃，而应该转过身来看看窗外的风景，或者开个小窗口看看mm。同时在你的大脑里面想想应该怎样一步步解决这个问题，最好是拿出一张纸，把自己想到的思路用流程图的方式画出来。虽然程序猿的工作价值表面看起来似乎是体现在一行行的代码上，其实真正高效的程序猿更需要思考和在纸上写写画画。当你头脑中和纸上有了解决问题的方法之后，转换成代码就是轻而易举的事情了。

记住，**程序猿的工作不是体力活，而是脑力活，除非你想一辈子当码农。**一个高效的程序猿可以在很短的时间内通过思考，通过写写画画，通过借鉴前人解决问题的思路来处理所

遇到的问题，效率堪比苦逼程序猿的10倍甚至更高。如果恰恰相反，你不幸就真成了所谓的码农。如果你认为自己是码农，那是因为你在偷懒，不是在键盘上敲打偷懒，而是自己在思考上偷懒了。
记得有人说过一句话，创业者不要用战术上的勤奋替代战略上的懒惰，这一点对程序猿同样适用~

打住打住，话说还能跟得上哥的思维吗?

现在我们要回到正事了。回到当前这个游戏，我们需要计算玩家的得分。刚才哥已经用说话的方式解决了这个问题，现在该用代码来体现出来了。

对这个问题，可以用两种方式来具体实现，让我一一道来。

第一个：
```
var difference: Int
if (currentValue > targetValue) {
    difference = currentValue - targetValue
} else if (targetValue > currentValue) {
    difference = targetValue - currentValue
} else {
    difference = 0
}
```

这里我们遇到了一个和if有关的新代码逻辑。if就是如果的意思，else if 就是如果不是这样，而是如果... 其实你也猜到了，这种类似英语的描述方式告诉我们如何来进行判断。

通常来说它的结构是这样的：

```
if (something is true) {
    //then do this
} else if (something else is true) {
    //then do that instead
} else {
    //do something when neither of the above are true
}
```

我们在if后面的()括号里面放上了一个条件判断语句。

如果判断的结果是true，也就是说currentValue的数值大于targetValue，那么就会执行花括号里面紧跟的代码。如果判断的结果不是true，程序就会查看else if条件，然后再次进行判断。很可能我们会使用多个else if，而程序就会从头到尾逐一进行判断，直到遇到某个判断结果是true的情况。如果没有一个结果是true，就会执行最后的else里面的语句。

在刚才的这段代码里面，首先我们创建了一个本地变量difference，用它来保存计算结果。我们希望这个数值是整数或者0，所以int类型的变量是符合要求的：

```
var difference: Int
```

接着我们判断currentValue和targetValue之间的关系。
首先判断currentValue是否大于targetValue:

```
if (currentValue > targetValue) {
```

这里的>就是大于号的意思，江湖人称大于操作符。当currentValue里面所保存的数值大于targetValue里面保存的数值时，currentValue > targetValue的判断结果就是true。
此时就会执行下面的语句：

```
difference = currentValue - targetValue
```

我想这句代码基本上人人都能看懂吧，我们用currentValue的数值减去targetValue的数值，然后把结果保存在difference里面。和小学数学的表达一模一样，只是两边不是具体的数字，而是变量。
还要废话一句，你会看到这里哥用的变量名称其实是有选择的。

很多程序猿写的代码会是这样的：

```
  a = b - c
```

然后下一个接手的程序猿就会感谢你八辈子祖宗，或者几个月后的某个深夜，你需要自己对着电脑屏幕发呆，顾不上和mm的温情约会了。

这个涉及到所谓的编码规范和编码习惯的问题。如果我们把代码世界里面的一切都看做生命体，那么起码你要给人家起个好听和有意义的名字吧。你爸妈给你取名字会随便用小三，小二，小十九这样简单但却毫无意义的名字吗？当然，不排除部分人的名字是这样的，比如张三很可能在家里排行老三。

无论是本地变量，实例变量，还是属性，包括方法名称，函数名称，我们都尽可能用一眼看起来就大概知道什么意思的名称。当然本地变量的命名可以稍微放松一点，但这样做也是没有坏处的。如果你把语言和程序中的所有东西看做有生命的个体，相信你会从中找到更多的乐趣。

继续回到我们的if条件语句。如果currentValue和targetValue的数值相同或是略小，那么currentValue > targetValue 的结果就是false（Objective-C里面的术语，表示不正确），

这个时候我们会忽略花括号紧跟的语句，而是直接切换到下一个条件语句：

```
} else if (targetValue > currentValue) {
```

同样的事情会再次发生，只不过这次targetValue和currentValue的角色颠倒过来了。只有当targetValue的数值更大的时候，才会执行下面的语句：

```
 difference = targetValue - currentValue
```

用targetValue里面保存的数值减去currentValue里面保存的数值，然后把结果保存在difference里面。

最后还有一种情况，就是两个变量里面的数值相等。也就是说我们的玩家太NB了，一发命中。这个时候的差值就是0.

```
} else {
    difference = 0
}
```

好了，到目前为止，我们已经搞定了人生中的第一个算法。

现在可以把这个算法放到动作方法里面了。

在Xcode里面切换到ViewController.swift，然后修改showAlert()方法的代码：

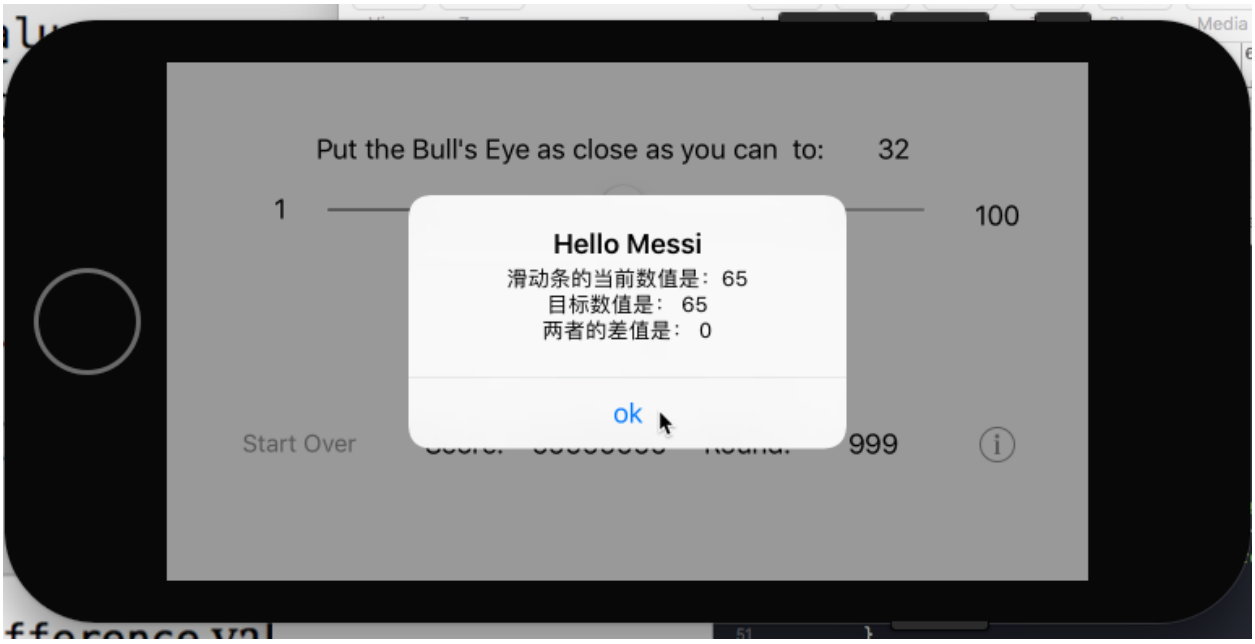
```
    @IBAction func showAlert(){

        var difference: Int
        if currentValue > targetValue{

            difference = currentValue - targetValue
        }else if targetValue > currentValue{
            difference = targetValue - currentValue
        }else{
            difference = 0
        }

        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()
    }
```

我们把差值的信息也放到了提示对话框里面。

需要注意的是：
Swift语言对空格拼写的要求非常严格，所以在变量和操作符（比如+）之间必须添加空格，否则你的代码里面会大量提示出错~

现在点击Run运行一下看看。



看着这个小小的应用一步步接近自己预设的目标，是不是有那么一点小激动呢~

辛苦了一天，又到发放福利的时间了。明天再见。


PHOTOGRAPHY