# Script——Studio的魂

编码是信息从一种形式或格式转换为另一种形式的过程，也称为计算机编程语言的代码简称编码。用预先规定的方法将文字、数字或其它对象编成数码，或将信息、数据转换成规定的电脉冲信号。编码在电子计算机、电视、遥控和通讯等方面广泛使用。编码是信息从一种形式或格式转换为另一种形式的过程。解码是编码的逆过程。

第一代编程语言是机器语言，相当于人类的原始阶段。机器语⾔是微处理器理解和使⽤的，⽤于控制它的操作⼆进制代码。8086到Pentium的机器语⾔指令长度可以从1字节到13字节。尽管机器语言非常复杂，但其任然是有迹可循的。存在着多⾄100000种机器语⾔的指令。这意味着不能把这些种类全部列出来。其优点是：直接执⾏，速度快，资源占⽤少；但其缺点也十分明显:可读性、可移植性差，编程繁杂

第二代编程语言为汇编语⾔（⾯向机器的程序设计语⾔），相当于人类的手工业阶段。相比第一代编程语言，汇编语言设计的程序最终被转换成机器指令，故能够保持机器语⾔的⼀致性，⾔设计的程序最终被转换成机器指令，故能够保持机器语⾔的⼀致性，直接、简捷，并能像机器置零一样访问、控制计算机的各种硬件资源如内存、磁盘、CPU等。目标代码也十分简短、占用内存少，执行速度快，是高效的编程语言。但其缺点任然十分明显，汇编语⾔是⾯向机器的，处于整个计算机语⾔层次结构的底层，故被视为⼀种低级语言。通常是为特定计算机或系列计算机设计的，不同的处理器有不同的汇编语⾔语法和编译器，编译的程序⽆法在不同的处理器上执⾏，缺乏可移植性；同时汇编语言的特性决定了其开发效率低，开发周期长。

第三代编程语言：⾼级语⾔，相当于人类的工业阶段。计算机语⾔具有⾼级语⾔和低级语⾔之分。⽽⾼级语⾔⼜主要是相对于汇编语⾔⽽⾔的，它是较接近⾃然语⾔和数学公式的编程，基本脱离了机器的硬件系统，⽤⼈们更易理解的⽅式编写程序。编写的程序称之为源程序。⾼级语⾔并不是语⾔，也并不是特指的某⼀种具体的语⾔，⽽是包括很多编程语⾔，如流⾏的Java，c，c++，C#，pascal，python，lisp，prolog，FoxPro，易语⾔，中⽂版的C语⾔习语⾔等等，这些语⾔的语法、命令格式都不相同。

Roblox Studio采用的编程语言为Lua。Lua 是一个小巧的脚本语言。它是巴西里约热内卢天主教大学（Pontifical Catholic University of Rio de Janeiro）里的一个由Roberto Ierusalimschy、Waldemar Celes 和 Luiz Henrique de Figueiredo三人所组成的研究小组于1993年开发的。 其设计目的是为了通过灵活嵌入应用程序中从而为应用程序提供灵活的扩展和定制功能。Lua由标准C编写而成，几乎在所有操作系统和平台上都可以编译，运行。Lua并没有提供强大的库，这是由它的定位决定的。所以Lua不适合作为开发独立应用程序的语言。Lua 有一个同时进行的JIT项目，提供在特定平台上的即时编译功能。

Lua脚本可以很容易的被C/C++ 代码调用，也可以反过来调用C/C++的函数，这使得Lua在应用程序中可以被广泛应用。不仅仅作为扩展脚本，也可以作为普通的配置文件，代替XML,ini等文件格式，并且更容易理解和维护。 [2] Lua由标准C编写而成，代码简洁优美，几乎在所有操作系统和平台上都可以编译，运行。

## 1 基础编码知识

在 Roblox Studio 中，Lua 代码行保存在**脚本**中。这些脚本提供的游戏指令集用于控制编辑器中的一切对象，包括玩家、场景、文件夹等等具体或者抽象的对象。

### 1.1创建新脚本

Studio的脚本分为Script和LocalScript两类，分别用于服务端和客户端。编辑器有严格的服务器和客户端界限，非常多的特殊文件夹无法跨服务器——客户端边界复制或者访问。本章节将侧重于介绍服务器脚本Script。

类似于在workspace中插入Part，脚本的插入也有数种方法。Studio编辑器中有一个名为ServerScriptService（服务器脚本服务）的特殊文件夹。该文件夹用于统一存放服务器脚本。当然你也可以将脚本放置于Part或者Model下方便区分。

#### 1.创建脚本

将鼠标移动至ServerScriptService上，点击弹出的加号按钮，选择Script，在ServerScriptService中插入一个脚本。如图1.1

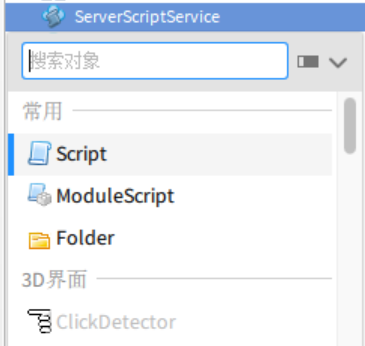


图1.1 脚本示例

#### 2.重命名脚本

右键单击脚本，在弹出的窗口上选择重命名，将脚本重命名为其他名称以示区分。这一步很重要，在开发的过程中，脚本名称是最明显的区分不同功能脚本的标识。如图1.2

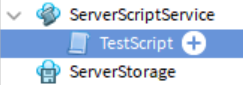


图1.2 脚本名称示例

#### 3.查看脚本编辑器

双击新建的脚本，脚本编辑器会自动弹出，这是编辑脚本内容的工作区域（如图1.3）。新建的脚本内会有一行所有程序员的第一行代码：“Hello world！”。点击工具栏**视图（View）**选项，在弹出的选项卡中选择**输出（OutPut），**将输出栏固定至你的编辑器下方（如图1.4）。点击上方测试，选择“开始游戏”按钮开始测试。（如图1.5）。观察下方输出框的输入。（如图1.5）

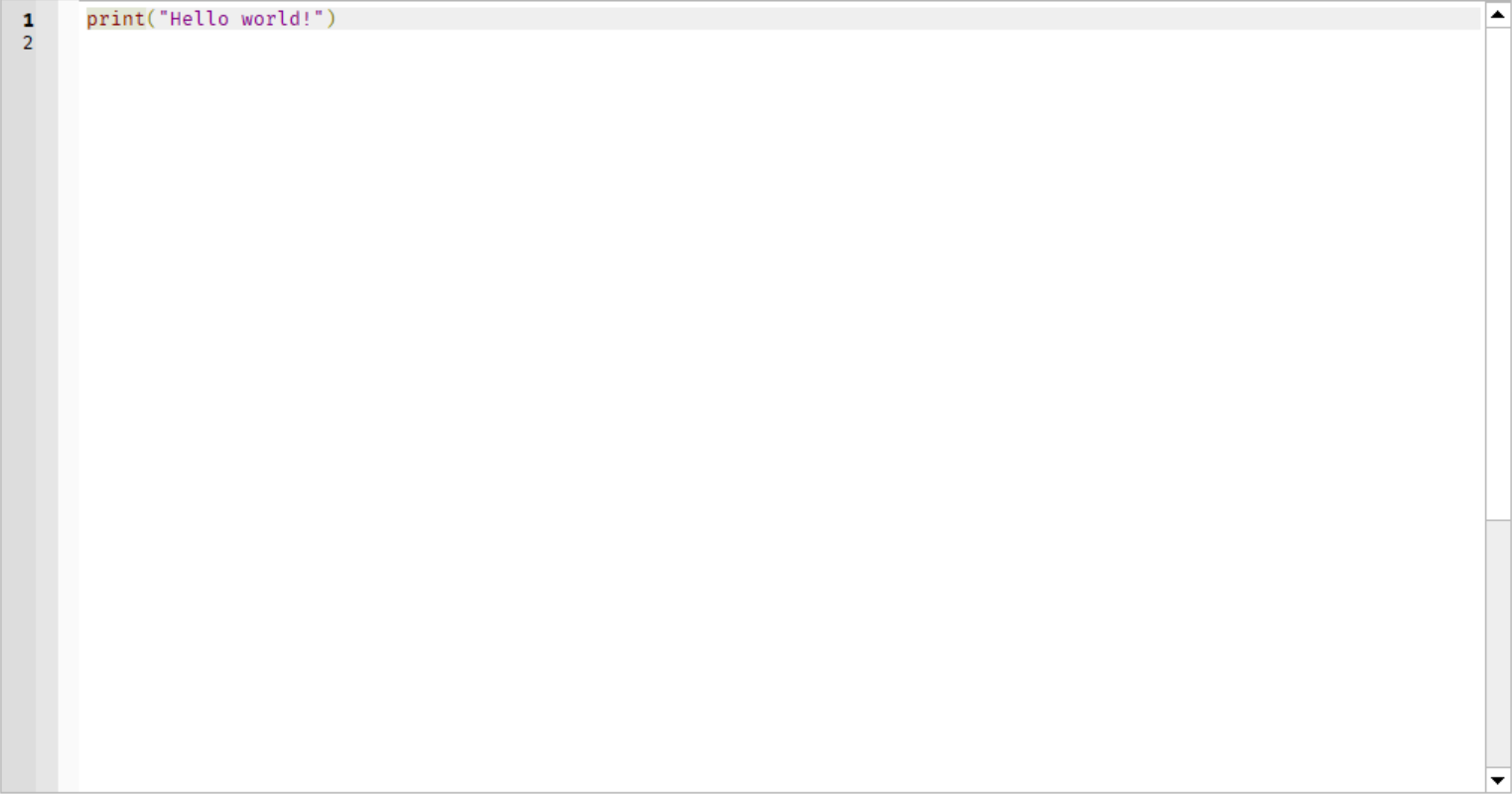


图1.3 脚本编辑器示例

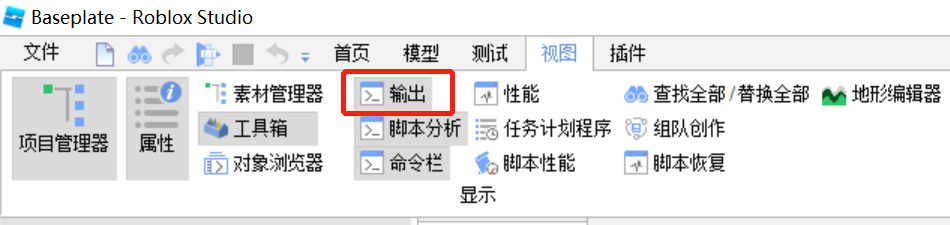


图1.4输出框窗口位置示例

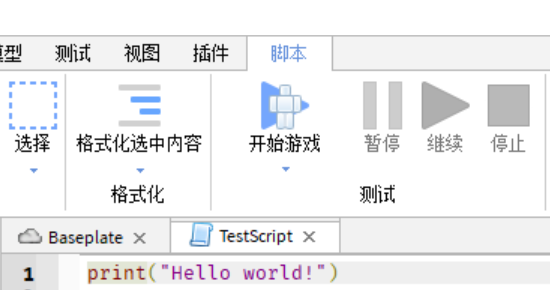


图1.5“开始游戏“按钮位置示例



图1.6 输出框示例

其中第一段数字为脚本运行时的本地时间，第二段为脚本运行的结果，第三段为脚本工作在哪个端口（客户端与服务器端），第四段为运行的脚本的名称和有效代码所在行数。读者可自行修改代码行数位置、脚本名称来测试与应证。测试完脚本后，按下 Stop（停止）。此时可以回到脚本选项卡对脚本进行编辑，或者对其它字符串进行测试。

#### 代码错误排查

即使对于有经验的工程师来说，其所编写的代码有时也不会在第一次尝试时就正常运行。错误排查是浏览代码以查找并修复问题的过程。

如果你的代码未正常工作，请检查：

·确保 print 为小写。

·该字符串用引号引起来："就像这样"。

·该字符串用括号括起来：("就像这样")。

为了更好地理解错误是什么样子，请有目的地创建一个错误。

1.打开脚本。

2.从 print 语句中删除一个引号。

3.按照打开输出（OutPut）窗口的方式打开脚本分析栏，脚本分析栏中将会详细将错误地点写明。如图所示，ServerScriptService中的TestScript脚本中，第一行报错。系统给出了两种错误可能：字符串格式错误或Print函数未完成，缺少关闭Print函数的右括号。

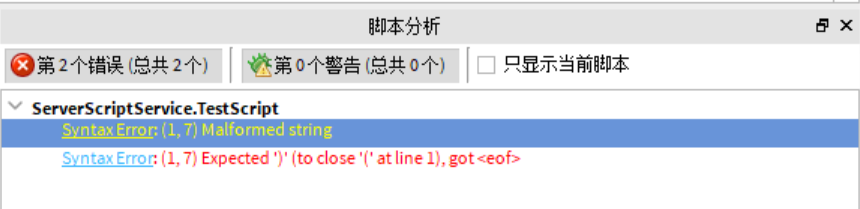


图1.7 报错示例

### 1.2 字符串变量

"Hello World" 是一个字符串变量类型。变量是一种程序可以利用和改变的信息容纳对象，比如玩家的名称、玩家的分数、玩家所拥有的金币数量等。

变量的类型有很多种。字符串变量中可以包含一系列字母和数字的组合。它们非常适合用来容纳名称或地址，但不适合用于计数。字符串类型的变量需要在前后加上双引号。

字符串示例：

·"Welcome joining the game!"

·"There are 10 players left"

·"衬衫的价格是九磅十五便士"

### 1.3利用BrickColor来理解属性

属性用于控制对象的外观和功能。一个对象的属性可能包括材质、颜色和形状等。你可以使用代码来更改这些属性并在游戏中实现相应的效果。例如，通过更改一个对象的颜色属性，你可以创建一个任意颜色的Part。

#### 属性窗口

Roblox Studio编辑器的对象属性都是显示在Properties（属性）窗口中的。在WorkSpace中新建一个Part，选择Part，在右下角使用滚轮浏览该对象的属性窗口，包括对象的名称（Name）、颜色（Color）、尺寸（Size）、位置（Position）等等。

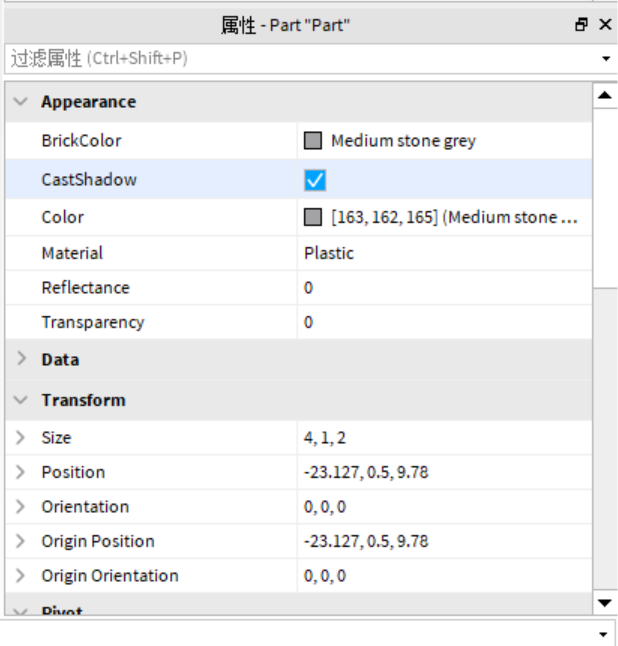


图1.8 Part对象属性示例

#### 设置部件和脚本

为了更好地理解对象属性和脚本之间的联系，我们现在试着完成一个这样的功能：在WorkSpace中创建一个在游戏开始时改变颜色的Part对象。

1.选择现有部件或创建新部件。

2.重命名部件。本例使用 TestPart。

3.将脚本重命名为 ChangeBrickColor。

4.删除或者注释掉 Hello World。

#### 编写脚本

1. 撰写注释：

你应该总是在开始编写新的脚本时对脚本的功能进行注释。注释是帮助编码人员记住程序的用途但实际上并不运行的特殊行。Lua语言中的注释命令为两个短横线“—“。现在我们新建一条注释用于解释即将编写的脚本的用途。



图1.9 注释示例

1. 获取部件路径：

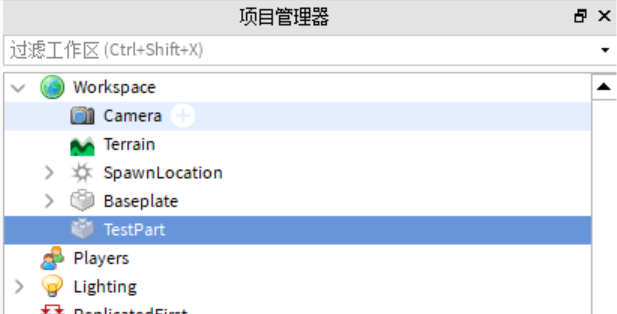
Lua语言是一门面向对象的编程语言。在改变对象的属性之前，我们需要首先获取到我们的目标对象Part。使用 Explorer（管理器）找出部件的位置。本次示例中的TestPart 在 Workspace（工作区）下。

图1.10 工作区与TestPart

1. 编写脚本：

在上一步中，我们寻找到了Part所在的位置。我们在脚本中输入如下代码：

game.Workspace.TestPart.BrickColor =

该段代码的含义是：游戏工程文件中的WorkSpace文件夹中的TestPart对象下面的Color属性。注意，在Roblox Studio中，会自动检索当前路径下的对象，使用Tab键可以选择相应对象自动补齐代码。灵活运用你的Tab键可以有效地减少你的工作量。

若要更改 BrickColor 属性，你将创建一个新的 BrickColor 来代替当前的 BrickColor。程序使用RGB值，即红色、绿色和蓝色的组合来创建屏幕上的所有颜色。

使用 RGB 值有以下几个规则：

1）使用 3 个小数数值，各自代表一种颜色。

2）数字之间以逗号（,）分隔。

3）使用 0 到 1 之间的数值。0 表示该颜色完全关闭。1 表示该颜色完全打开。

下面是一些 RGB 值的示例：



图1.11 RGB值示例

现在，在预留的括号内输入你想要的RGB值，点击开始游戏测试程序是否正常运行、Part是否更改颜色。

#### 代码错误排查

如果你的代码未正常工作，请检查：

·RGB的色域值是否合法地介于0和1之间

·程序内使用的逗号、括号、点号是否使用的是英文，请检查你的输入法

·程序内代码的大小写是否与例程一致，Studio是敏感大小写的

### 1.4 利用BrickColor来理解循环

在编码时不必局限于一种颜色。合理使用循环可以让你的部件颜色可以按照你所想的那样变化。

使用 while true do 循环重复代码。此循环会直到被停止时始终重复运行。由于我们没有编写让代码停止的部分，代码将一直循环运行。

#### 设置部件和脚本

1.选择现有部件或创建新部件。

2.重命名部件。本例使用 LoopingPart。

3在 ServerScriptService 下创建一个名为 LoopingScript 的新脚本。

#### 编写脚本

删除或者注释掉 Hello World并撰写注释：



1.12 注释示例

1. 使用变量储存对象：

变量不仅可以保存字符串，还可以保存对对象的指向。与其一遍又一遍地输入 game.Workspace.NameOfYourPart，不如将这一整行存储在一个局部变量中。局部变量是通过输入 local 加上变量的名称来创建的，例如 local nameOfVariable。局部变量只会在它被创建的脚本中起作用。你使用的大多数变量都是局部变量。在注释下方输入 local loopingPart，创建一个名为 loopingPart 的局部变量。

在同一行输入local colorofpart = game.Workspace.loopingPart 将变量设置为循环的部件。

1. local colorofpart=game.Workspace. LoopingPart

注意：

· 在计算机科学中，= 符号的含义与数学中的含义不同。它并不表示相等关系，而是表示更改变量的值。如果需要表示相等，Lua语言中需要使用==。

·Roblox 开发者在命名部件和变量时应当遵循一定的规则，以帮助了解哪些是变量，哪些是部件：

部件：PascalCase（首字母大写）

示例：LoopingPart、EndZone

变量：camelCase（首字母小写）

示例：loopingPart、playerHealth

1. 创建while循环

在下一行中输入 while true do，按下回车，单词end将会自动补齐。在while循环体中键入如下代码

1. colorofpart.BrickColor=BrickColor.new(0.1,0.1,0.1)
2. colorofpart.BrickColor=BrickColor.new(0.2,0.2,0.2)
3. colorofpart.BrickColor=BrickColor.Blue()）
4. colorofpart.BrickColor=BrickColor.Red()
5. colorofpart.BrickColor=BrickColor.Gray()

如果现在便执行程序的话，将会收到错误消息。这是因为脚本频繁更改颜色，导致 Studio 过载。变色函数反复快速执行而无间断，如果没有暂停或等待程序，砖块的颜色会被快速改变，让你甚至看不到第一个颜色的显示。为了使脚本在运行下一行代码之前等待，请灵活使用 wait() 函数。你可以自己设定自己想要的秒数。

Wait函数用于暂停程序的执行，暂停的时长决定于其中的参数。例如：

1. wait（1）--等待一秒

现在，自行插入wait函数，让你的代码在运行过程中有间隙，让你得以观察到循环颜色的改变

#### 代码示例

1. *– Changing the color of Part every few seconds*
2. *-- Create a variable to store the part*
3. local colorofpart = game.Workspace. LoopingPart
4. *-- Looping Code*
5. while true do
6. *-- Changes loopingPart's color*
7. colorofpart.BrickColor=BrickColor.new(0.1,0.1,0.1)
8. *-- Wait 3 seconds before next instruction*
9. wait(3)
10. colorofpart.BrickColor=BrickColor.new(0.2,0.2,0.2)
11. wait(3)
12. colorofpart.BrickColor=BrickColor.Blue()）
13. wait(3)
14. end

#### 代码错误排查

·如果代码设定的颜色被跳过，请检查是否在该行代码前后都设置了wait函数，并确保你的变色代码位于while true do 和 end之间

·如果未添加wait函数便运行脚本，将会收到错误消息。这是因为脚本频繁更改颜色，导致 Studio 过载。

### 1.5 利用Script.Parent重复利用代码

如果程序中包含具有相同名称的多个部件，那么你目前所使用的代码将无法运行。也就是说，如果你想让多个部件改变颜色，就需要不断创建新部件并分别为它们创建新脚本。这需要花费大量的时间。除此之外，Studio提供了一个简便方法，让你能够将脚本随意复制粘贴并让其生效。

#### 将脚本放置于部件下

不用在ServerScriptService下放置脚本，而是将其作为子项在对象下放置。

1. 创建一个新部件并为其重命名。本课程中将使用 *LoopingPart*。
2. 右键单击该部件，然后选择 **Insert Object**（插入对象） > **New Script**（新建脚本）。
3. 重命名脚本。本课程中将使用 *ColorChangeScript*。
4. 在脚本编辑器中，删除  print("Hello world!")。
5. 将上一节的变色代码复制粘贴到 *ColorChangeScript* 中，创建一个循环颜色的部件，每隔 3 秒颜色改变一次。

现在回顾上一节中编写的代码。原来的代码是通过根目录从game开始索引目标对象，现在我们进行一点小小的修改。使用local colorofpart = script.Parent替换掉local colorofpart = game.Workspace..LoopingPart。运行游戏，观察到程序仍然正常运行。



图1.13 脚本父项设置示例

#### 父项和子项

父项是一类对象的统称，你可以将其理解为文件夹，子项就是文件夹内包含的对象。在本例中，父项就是LoopingPart，它的子项就是ColorChangeScript。在刚刚创建的示例中，部件是父项，脚本是子项（通常为了方便我们在实际开发中也会这样设置）。而这一段特殊的代码 Script.Parent则会告诉脚本如何寻找其父项也就是目标部件。这样做的好处是可以避免从根目录开始索引对象而节省大量时间，你甚至不需要知道对象的名称（Name）就可以索引到它。

#### 测试脚本

在WorkSpace下新建多个任意Part，将修改过后的代码复制粘贴至每个Part下作为其子项。由于脚本中的变量是参考的每个脚本各自的父项，因此这些脚本将会对它们各自的父项起作用。开始游戏进行测试，观察现象。

## if/then与循环

### 2.1．通过Print（）来理解函数

#### 封装的概念

封装，即隐藏对象的属性和实现细节，仅对外公开接口，控制在程序中属性的读和修改的访问级别；将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成“类”，其中数据和函数都是类的成员。函数是指令的集合，是封装概念的具体实现之一，可在一个脚本中多次使用。有些属于预制函数（例如 print和 wait），这些函数构建在大多数编程语言中。程序员也可以为想要多次使用的代码自行创建自定义函数。在 Lua 中，通过输入 local function NameOfTheFunction() 可实现此操作。

举个简单的例子，英文字母从A到Z一共有26个，我每次和别人交流的时候，我都需要从A到Z说一遍，这样太麻烦了，所以我给它们起了个名字叫做字母表。这样我需要告诉别人的时候，直接说字母表就可以了。这里的字母表就是A到Z 26个字母的封装。

再举一个例子，降龙十八掌一共有十八个招式，神龙摆尾、亢龙有悔、飞龙在天等等。我每次要告诉别人这套拳法的时候，都需要和报菜名一样念十八个招式，这样太麻烦了。我只需要用降龙十八掌这个词来代替十八个招式就可以了。这里的降龙十八掌就是对十八个招式的封装。

函数采用驼峰式拼写法，即首字母小写，紧跟的单词首字母大写。

* 示例：addPoints()、restartGame()、pointAdded()

#### 设置脚本

1. 在ServerScriptService下新建一个脚本，重命名为funcScript
2. 在脚本编辑器中，删除  print("Hello world!")。
3. 键入 local function printSth（），按下回车，函数体将会自动补齐end。End是一个函数结束的标志，如果没有end，代码将会无法运行
4. 在local function printSth（）和end之间输入如下代码：**print**("This is a function for test!")
5. 在end后方，输入 printSth（），使你的函数运行。
6. 点击开始游戏，观察程序输出窗口。

#### 设置函数体外的print

1. 在上一步完成的代码的基础上，进行部分修改。在printSth（）函数体外创建一个print语句。
2. 在新建的print后方，对你的printSth（）函数调用四次
3. 开始游戏，观察程序输出窗口。

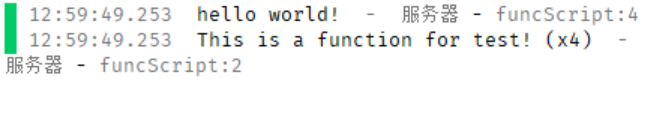


图 2.1 函数调用输出测试

可以观察到，“This is a function for test”输出了四次，而“hello world”只输出了一次。这有助于你理解函数的概念。而先输出“hello world”后输出“This is a function for test”的原因是，程序是按顺序进行的，在代码中我们首先使用了print函数，然后才调用了自定义函数。

#### 代码示例

1. local function printSth()
2. print("This is a function for test!")
3. end
4. print("hello world!")
5. printSth()
6. printSth()
7. printSth()
8. printSth()

#### 代码错误排查

·必须在 local function printSth（） 和 End 之间输入函数的所有代码。函数运行时，任何不在两点之间的代码都无法运行。

·检查脚本，查看是否在函数体结束后运行你的自定义函数。

·检查大小写，Studio是对大小写敏感的编辑器。

### 通过实例来练习函数

Instance（实例） 是 Roblox 类层次结构中所有类的基类。Roblox 引擎定义的每个其他类都继承 Instance 的所有成员。无法直接创建 Instance 对象。

Instance 具有称为 Instance.new（） 的特殊函数，用于通过代码创建对象。此函数采用类的名称作为参数并返回创建的对象。无法使用 Instance.new 函数创建抽象类和服务。

#### 创建粒子发射器

1. 在workspace中新建一个Part对象，用于为粒子发射器提供父项。在part的下方新建一个子项脚本，重命名为particleScript。再新建一个ParticleEmitter作为part的子项。现在你的项目管理器应该像这样。

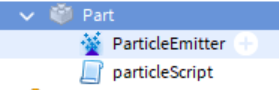


图2.2 对象父子项层级关系示例

1. 在脚本编辑器内，输入如下代码
2. local firePart= script.Parent.ParticleEmitter

该代码的含义是，新建一个firePart变量，用其容纳粒子发射器。由于我们设置了特殊的子父项层级，我们可以通过朔源和遍历索引到我们想要的粒子发射器对象。这样我们后续调用该实例就可以直接使用firePart进行调用了，这也是一种封装的思想。

粒子发射器是一个Studio中特殊的效果对象，向世界中发射可定制的 2D粒子。在 Roblox 中，粒子是一种在世界中渲染的 2D 图像，因此它总是面向镜头。你可以理解为，粒子发射器是一个不停向外发射图片的喷射器。通过修改其材质属性可以修改喷射出来的效果。ParticleEmitter 必须设置为一个 BasePart (如 Part) 或 Attachment 的子代，这样它才会使能。如果不对其进行父项的设置，它将无法工作。

#### 设置脚本

1. 在particleScript内，新建一个自定义函数stopFunc（）。
2. 在函数体内部，输入如下代码
3. firePart:Destroy()
4. 该代码的含义是，摧毁firePart对象。在此例中即为摧毁新建的粒子发射器。
5. 在函数体外部、程序结尾处，调用你的自定义函数。
6. 在游戏开始之前，观察workspace内的情况，观察粒子发射器是否正常使能。点击开始游戏进行测试，观察现象

#### 代码示例

1. local firePart= script.Parent.ParticleEmitter
2. local function stopFunc()
3. firePart:Destroy()
4. end
5. stopFunc()

#### 代码错误排查

·如果你的粒子发射器在开始测试前并未使能，观察你的子父项层级设置是否有误。

·如果开始测试，粒子发射器仍然工作，观察程序尾部是否进行了自定义函数的调用。

### 通过函数来了解参数和事件

除了属性和函数之外，每个对象还具有**事件**，用于建立因果关系系统。当游戏中发生特定事件时，事件会发出**信号**，例如玩家触摸了某个对象或某个玩家正在连接到游戏。**触发**事件就是让它发出这样的信号。

如果给定函数需要在**每次**触发事件时都进行运行，则可以使用 Connect() 函数。此函数将立即返回一个连接对象。 在下面的示例中，我们将函数 onTouched() 连接到 Workspace 中的给定 Part。当有部件与 Part 碰撞时，脚本将输出碰撞中涉及的另一个部件的名称。

#### 设置部件和脚本

1. 在workspace下新建一个part，在其下新建一个脚本。
2. 对脚本进行编辑，键入如下代码：
3. local myPart = script.parent
4. local function onTouched(otherPart)
5. print("Part was touched )
6. end
7. myPart.Touched:Connect(onTouched)

该段代码的含义是：当part被触碰时，输出触碰到该part的部位的名称。

其中：

* 1. 自定义函数后方的参数otherPart用于接收触发函数的对象，我们之后可以很方便地对其进行进一步操作。
  2. Touched是一个由Studio封装好的事件，这个事件将会在被接触时触发。这句代码的含义是：部件触发了touched事件的话，就链接到自定义函数onTouched。也就是说，一旦有物体触碰了part对象，就调用onTouched函数。“:” 是一个特殊符号，用于将对象连接到函数。在本例中，Destroy() 将作用于存储在 otherPart参数中的任何对象。

运行游戏，通过wsad控制你的角色触碰part，观察程序输出窗口的输出结果。



图2.3 输出结果测试

#### 利用传参对代码进行改动

在该段程序中，otherPart就是作为传参起作用，它用于容纳触发touched函数的对象。在该段程序中，其作用域只包含了该函数体（从local function开始，到end为止）。我们可以直接对其进行操作而不需要重新索引。

1. 在上一步的代码中进行一些增添。在函数体中加入如下代码：
2. otherPart:Destroy()
3. 在workspace中将part对象移动至浮空状态，在其属性栏中寻找Anchor属性并将其勾选（或者在工具栏中选择锚固），以免误触其他对象。

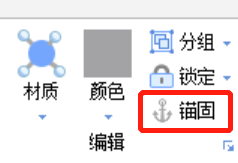


图2.4 锚固功能

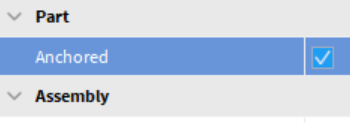


图2.5 锚固功能

3．开始测试。控制你的角色触碰part并观察现象。

#### 代码示例

1. local myPart = game.Workspace.Part
2. local function onTouched(otherPart)
3. print("Part was touched  " )
4. otherPart:Destroy()
5. end
6. myPart.Touched:Connect(onTouched)

#### 代码错误排查

·检查脚本和Part对象的父子项层级是否正确

·检查Destroy（）是否拼写正确

·检查Part是否已经锚固至半空

·检查Touched的首字母是否为大写

### 函数进阶用法——如何建造一座开关桥

通过前面数节的学习，我们应该已经对脚本有了一定的认知。现在我们来尝试一下如下任务：

创建一个红色的按钮和红色半透明的桥，此按钮将用于激活该座桥。当按下该按钮时，按钮会变成绿色，桥会也变成绿色，同时玩家可以使用桥。如果玩家没有按下按钮，那么当他们尝试从桥上走过时，会从桥上跌落。对象的半透明效果可以通过调节transparency（透明度）来实现，其范围为0（完全可视）到1（完全不可视）。桥的可用与否可以更改Cancollide（）属性来完成，CanCollide 属性用于确定某个部件是否与其他部件进行物理交互。除此之外，你也可以自行添加其他代码，修改对象的一切属性如角度、位置等等。

#### 设置部件

1. 在workspace中创建一个part，将其重命名为*Bridge*。对其进行缩放等变化，让其看起来像一座桥的桥面
2. 选择桥，在其属性窗口中进行修改：
   * 将 Transparency（透明度）更改为 0.5。
   * 选中 Anchor（锚固）。
   * 取消选中 CanCollide。
3. 测试游戏，桥看起来应该比较模糊，并且无法在桥上行走。
4. 在桥下新建一个子项part对象。放置于桥的附近。修改其颜色并在半空锚固它，作为开启桥的按钮

#### 设置脚本

1. 在上一步新建的按钮下新建一个子项脚本。修改脚本名称为“bridgeScript”
2. 使用script.parent进行索引，你需要分别使用两个变量索引桥和按钮以免混淆
3. 自定义一个函数，在函数体内编写修改桥、按钮颜色、透明度和cancollide属性的代码。结合你已经学过的知识，自行思考如何编程。
4. 新建链接函数，将按钮的Touched事件与你的自定义函数联系起来。检查一下程序是否有逻辑错误。
5. 点击开始游戏，进行测试。

#### 代码示例

1. *--设置局部变量button*
2. local button = script.Parent
3. local bridge = script.Parent.Parent
4. *--定义函数，实现打印、按钮变色、桥体透明度更改和桥体触碰true*
5. local function Bridgeon()
6. print("somebody touched the button")
7. button.BrickColor=BrickColor.Green()
8. bridge.Transparency=0
9. bridge.CanCollide=true
10. bridge.BrickColor = BrickColor.Green()
11. end
12. *--当按钮被触碰后，将按钮触碰的事件与Bridgeon函数连接起来*
13. button.Touched:Connect(Bridgeon)

#### 代码错误排查

* 确保该部件已正确锚固且未接触任何其他对象。一定要避免接触地形或其他部件而导致意外触发 Bridgeon() 函数。

### if/then 条件语句

制作游戏时，你会发现自己想要确定如下因果关系：

·如果玩家得到 10 分，则在游戏中获胜。

·如果玩家获取了某个道具，则大幅提高奔跑速度。

·如果玩家在聊天中说“生日快乐”，则放出烟花。

所有这些关系都需要使用条件语句，即在特定条件为 true 时才会运行的代码行。一种条件语句是 if/then 语句，它会在运行指令之前检查某个条件是否为 true。

在接下来的示例中，我们将要完成这样的功能：如果语句条件为true，则 更改部件的透明度。例如1+1=2，由于此语句结果为true，因此将会进行下一阶段的代码执行工作。

#### 设置部件和脚本

1. 在ServerScriptService中，创建一个脚本，重命名为ifthenScript
2. 在workspace中创建一个新的Part部件，重命名为ifthenPart。
3. 进行脚本编辑，输入如下代码：
4. local part = workspace.ifthenPart
5. if 1+1==2 then
6. part.Transparency = 0.5
7. end
8. 开始游戏进行测试，观察Part的性质。

可以显而易见地观察到部件的透明度发生了更改。但是需要注意以下问题：

如果你更改透明度的代码放置到了if 和end语句外面，代码仍然会生效，但是严格角度来说，并不是if then 语句发生了作用。

现在，请有意地更改判断条件，将条件更改为false。开始游戏测试，观察现象。

#### Lua语言所用的数学运算符和逻辑运算符

Lua语言有独特的数学和逻辑运算符，你也可以进行相乘、相加、相减或其他大小比较，而不是检查两个数字是否相等。根据你制作的游戏，其他的比较可能更有用。



图2.6 Luau语言的数学运算符



图2.7 Lua语言的逻辑运算符

### 2.6If/then条件语句制作陷阱

使用 if/then 语句，你可以创建更改玩家生命力的陷阱，而不仅仅是销毁与陷阱接触的所有部件。Roblox 虚拟角色包含一个名为 Humanoid 的特殊对象，它能让玩家四处移动并控制玩家的生命力。if/then 语句可用于检查接触陷阱的玩家是否具有 Humanoid 部件。如果有，则将玩家的生命力设置为 0。下面我们通过一个例程来帮助理解if/then的用法。

#### 设置部件和脚本

* 1. 插入并命名Trap部件，将其放置于地面上并锚固，更改它的颜色为红色、材质为霓虹，让它看起来更醒目。
  2. 在Trap部件下方新建一个子项脚本。在其中根据前面所学的知识，自行定义所需要用到的对象。
  3. 新建一个自定义函数，将其传参命名为otherpart。
  4. 定义一个新的变量character，用于储存otherpart的父级对象。

注意，Humanoid 对象就像现实生活中的人类，由许多部位组成，如手臂和腿。在游戏中，只有 Humanoid 的某些特定部位（如他们的脚或者手）会更容易接触陷阱。在此实例中，会接触到Trap的对象只有玩家角色的腿部。

要更改 Humanoid 的生命力，脚本需要获得腿的所有者或父级对象。腿是没有生命值的，只有腿的所有者或者父级对象才会拥有生命值。otherPart.Parent 是一种存储部件的父级对象以便稍后使用的方法。

* 1. 将函数链接至Trap的Touched事件上，以便让Trap被触碰后会触发函数。
  2. 查询character是否为人类。由于触碰到Trap的不一定全部都是玩家，例如Baseplate（地面）可能触碰到Trap，其父级对象为workspace，由于workspace没有生命值属性，程序就会报错。为了解决这个问题，我们需要使用索引函数Find FirstChildWhichIsA（）函数查询所有的character对象。如果查询到character对象是Humanoid，则才会进行下一步，否则将会一直等待。

请记住，参数 otherPart 记录触碰 Trap 部件的任何对象，但发生接触的对象可能是玩家的某个部件，也可能是底板。为了使陷阱只会销毁 Humanoid，可以使用 if/then 语句检查 otherPart 中的部件是否包含 Humanoid 对象。函数 FindFirstChildWhichIsA() 可用于查找特定类型的对象，这是一个Studio封装好的函数，因为我们要查找的是 Humanoid 类型的对象。你可以使用变量存储找到的所有 Humanoid 类型的对象。

现在你的代码应该是这样的：

1. local part  = script.Parent
2. local function onTouch(otherpart)
3. local character = otherpart.Parent
4. local human = character:FindFirstChildWhichIsA("Humanoid")
5. end
6. part.Touched:Connect(onTouch)
   1. 使用if/then函数判断，如果寻找到了Humanoid对象（即true），则进行下一步操作：将玩家的Health属性设置为0，玩家将会死亡
   2. 运行游戏，测试Trap功能，观察现象

#### 代码示例

1. local part  = script.Parent
2. local function onTouch(otherpart)
3. local character = otherpart.Parent
4. local human = character:FindFirstChildWhichIsA("Humanoid")
5. if human then
6. human.Health = 0
7. end
8. end
9. part.Touched:Connect(onTouch)

#### 代码错误排查

·销毁 humanoid或 character 不会实际销毁玩家，但可能会导致游戏中断，因为该玩家无法正常重生，所以不要对humanoid或者character对象使用destroy函数。如果你希望他们重新开始游戏，请始终将玩家的生命力设置为 0。

·检查所有变量是否相匹配，检查大小写和拼写是否正确。

·程序应该是有两个end，一个用于关闭if/then语句，一个用于关闭自定义函数onTouch。

·另外，NPC对象也是拥有Humanoid的，这也就意味着你的NPC角色也可能受到Trap的影响。

### 2.7 If/then条件语句进阶使用

通过if/then条件语句以及上一节中介绍的humanoid的相关知识，我们可以尝试给予玩家特殊能力，比如加速模块、隐身能力等等。本例将介绍加速模块的设置。

在开始实例之前，我们需要了解Humanoid的相关信息：

humanoid（人形对象）是一种特殊的对象，可使模型具有角色的功能。它赋予模型在 Roblox 关卡内四处走动并与各种部件交互的能力。

Humanoid 总是 Model （模型）内部的子项，而模型应为 BasePart 和 Motor6D 的集合。集合的根部件应命名为 HumanoidRootPart。一个名为 Head （头部）的部件也应连接至角色的躯干部件，可以是直接也可以是非直接连接。值得注意的是Head （头部）部件必须连接至一个名为 Torso （躯干）的部件，否则 Humanoid 会立即死亡。详细属性读者可以自行查阅Roblox Studio 的Api接口文档——[Humanoid (roblox.com)](https://developer.roblox.com/zh-cn/api-reference/class/Humanoid)

在默认的Studio中，玩家的奔跑速度WalkSpeed默认为16格/秒。在此基础上我们可以使用if/then语句进行下一步的操作。

#### 设置部件和脚本

1. 在workspace中新建一个部件，将其重命名为speedPart，在其下新建一个名为SpeedScript的脚本。
2. 结合上一章所学的知识，自行编写代码。在触碰Part后不再设置玩家Health值，而是设置玩家的WalkSpeed。另外，在判断是否为玩家接触part的时候，应使用逻辑运算符一并判断玩家的walkspeed是否为默认值16.
3. 在玩家高速运动尽兴后，应该将他们的速度重新设定为正常值，以免制造出一群闪电侠。使用wait（）函数，在调高玩家的速度后几秒，将玩家的速度重新调至16.

#### 代码示例

1. *-- Gives temporary speed boost*
2. local part = script.Parent
3. local function onTouch(otherPart)
4. *-- Looks for a humanoid and stores it*
5. local character = otherPart.Parent
6. local humanoid = character:FindFirstChildWhichIsA ("Humanoid")
7. *-- Print statement for debugging*
8. print("Something touched speedBoost")
9. *-- Checks for Humanoid without speed boost*
10. if humanoid and humanoid.WalkSpeed <= 16 then
11. print("A humanoid touched speedBoost")
12. *-- Change to faster speed, wait, then change to normal*
13. humanoid.WalkSpeed = 50
14. print("Player is moving super fast")
15. wait(2)
16. humanoid.WalkSpeed = 16
17. print("Player is back at normal speed")
18. end
19. end
20. part.Touched:Connect(onTouch)

#### 代码错误排查

·超能力也是需要限制的，玩家的速度应该设置在10到100之间，以免发生意外错误

·确保在改变速度的代码之间添加了wait函数。否则玩家速度将会在极短的时间内变化回来，看起来效果就是什么都没有发生

·请确保使用 = 设置 WalkSpeed 的值。使用 == 对比 if/then 语句中两个值。在程序语言中，赋值和恒等是两个完全不同的概念。

### 使用 For 循环重复执行任务

让代码重复运行的方法很多。如果你希望代码只运行特定次数，则可以使用 for 循环。一个好例子就是创建倒计时定时器。倒计时定时器的时钟每次减少 1 秒。如果你不确定代码应重复执行多长时间，或者如果你希望永久重复，则可以使用 while 循环，就如 BrickColor一节中所述。

#### For循环的工作原理

For 循环使用三个值来控制运行次数：一个控制变量，一个结束值，一个增量值。For 循环从控制变量的值开始运行，循环中的代码每运行一次，就增加或减少一个增量的变量值，直到变量值超过结束值。正增量值表示相加，而负增量值表示相减。Studio的默认增量为1，也就是说当你的增量为1的时候，你可以略过第三个值。

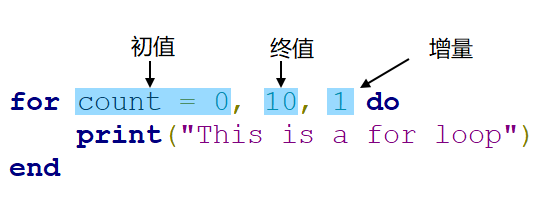


图 2.8 For循环参数

程序会自动对比当前count值与终值，如果count值小于终值，则会运行一次内部代码并让count加上增量值；如果count值大于增量值，则会终止循环。其中count为自定义变量，可以使用除关键字以外的任何名称。

#### For循环编码练习

1. 在ServerScriptService中新建一个脚本，在脚本编辑器里进行编程。
2. 输入关键字for开始循环。关键字是编程语言中有特定用途的字词，除了规定用途外，不能将其用于任何其他目的。例如，输入 while就会开始创建一个 while 循环。关键字不能用作变量名。
3. 设置for循环的参数，自行设置初值、终值和增量，在for循环体内加入print函数用于debug。在此例中可以设置 print（count），程序将会在每次循环中输出当前循环轮时count的值。
4. 在循环体内部设置wait函数，模拟每秒运行一次代码的环境。
5. 运行游戏，观察输出窗口

#### 代码错误排查

·检查for循环参数设置中，每两个参数之间是否使用了英文逗号进行分隔。其他符号或者中文逗号都会导致报错。请确保你的输入法为英文输入法。

·如果 for 循环一次性打印出所有结果，请确认是否有一个 wait 函数且它至少使用 1 秒的参数值。

### 使用 For 循环制作一台呼吸灯

For循环可以使用的场景非常非常多，比如倒计时，比如重复制作一堆五颜六色的砖块等等。在此例中我们将尝试利用for循环制作一台呼吸灯。

#### 设置部件和脚本

1. 在workspace中新建一个part，将其重命名为BreathLight。在其下创建一个名为breathScript的脚本和一个名为pointlight的示例。读者自行根据所学定义以方便后续调用。
2. Pointlight是一个特殊实例，它的作用是为其父项提供一个点光源的照明效果，能够模拟灯光等光照效果。它最重要的属性包括Brightness（光照强度）、color（光源颜色）、Enable（是否使能）、Range（照明半径）等。此例中我们将修改其Brightness（光照强度）以达到呼吸灯效果
3. 将游戏世界变成夜晚时间。在 Explorer（管理器） > Lighting（照明） 中，把 ClockTime 更改为 0。这样会让我们的灯光更为显眼。
4. 在脚本中创建两个独立的变量，分别命名为brightChange和timeChange并赋值。后续我们将使用这两个变量进行循环。执行额外的步骤来创建亮度变量和时间变量，这样更易于在稍后需要的时候调整值。程序员这样操作就可以在一处方便地更改变量而无需浏览可能会很长的脚本更改更多地方。这是一种很重要的编程思维。
5. local brightnessChange = 1
6. local timeChange = 1
7. 根据2.8中介绍的for循环知识，将brightChange和timeChange作为参数进行for循环的设置。
8. for currentBrightness = 0, 5, brightnessChange do
9. light.Brightness = currentBrightness
10. end
11. 由于没有设置wait函数，会导致循环瞬间执行了很多次，为了呼吸灯亮度变化更为自然，我们可以设置wait函数，将timeChange作为参数设置进去。
12. 开始游戏进行测试，观察灯光效果。自行更改brightChange和timeChange以及for循环的参数，让你的呼吸灯看起来更为自然。
13. 以上代码只会让呼吸灯从暗变亮，读者自行思考如何将呼吸灯的亮度反向变化直到熄灭。参考代码如下：
14. for currentBrightness = 0, 5, brightnessChange do
15. light.Brightness = currentBrightness
16. wait(timeChange)
17. end
19. *-- 第二个循环会使灯变暗*
20. for currentBrightness = 5, 0, -brightnessChange do
21. light.Brightness = currentBrightness
22. wait(timeChange)
23. end
24. 运行代码，可以观察到呼吸灯由暗变亮，又由亮变暗。但是只会执行一次。这是由于我们的代码没有外套循环函数，导致只会执行一次。为了解决这个问题，我们可以使用一个while true do 函数将呼吸灯的两个循环套起来，让它们一直循环下去。

#### 代码示例

1. while true do
2. for currentBrightness = 0, 5, brightnessChange do
3. light.Brightness = currentBrightness
4. wait(timeChange)
5. end
6. for currentBrightness  = 5, 0, -brightnessChange do
7. light.Brightness = currentBrightness
8. wait(timeChange)
9. end
10. end

#### 代码错误排查

·确保 light.Brightness = currentBrightness 位于 for 循环的 do 和 end 之间。

·查看 timeChange 是否至少为 1 或更高的数值。数字越小，亮度变化速度越快，但也更难看到亮度随时间的变化。

·查看 for 循环的第一行是否总共有两个分隔初值、终值和增量的逗号。

### 利用for循环对你的开关桥进行改造

回顾2.4中我们制作的开关桥。我们将使用for循环为其添加一个定时器，定时器读秒时桥可以行走，读秒结束后桥将变回不可触碰的状态。

#### 创建表面GUI

1.要观察到计时效果，我们需要使用到GUI功能。在桥的另一边创建一个足够大的Part对象，重命名为timePart；在其上添加一个名为SurfaceGui的子项。该子项将提供一个附着在父项上的能够容纳UI的场所。

2.在SurfaceGui上添加一个TextLable子项，该对象的功能是提供一个可视的文字容纳框，与之类似的还有ImageLable，用以提供可视的图片容纳框。但是现在，你可能甚至无法看到 TextLabel，它非常小，并且可能位于计时器显示的错误一侧。为了玩家可以更好地查看计时器，你可以更改SurfaceGui的“Face”（面向）属性，直到让其正确地显示在正确的方向上。调整SurfaceGui的CanvasSize大小，将其更改为（100，100）。

3.此时SurfaceGui已覆盖部件，但 TextLabel 还是过大。在项目管理器中选择TextLable，展开其Size（尺寸）区域，，扩展X和Y，将它们的Scale（缩放）设置为1，Offset（偏移）设置为0，这样将使TextLable和part部件的面尺寸相同。其中Scale属性取值范围为0~1，0为消失，1为全覆盖；Offset属性单位为像素，代表在相应的X或Y方向上、Scale的基础上额外移动多少像素。勾选TextScaled属性。这将自动缩放文本以适应表面 GUI。

4.修改TextLable的Text属性，该属性内容将会显示于TextLable上。删除其中的内容，让它不显示任何内容。我们将使用脚本来控制它的显示

#### 设置脚本

1. 在 bridgeScript脚本中，创建一个变量用于定义TextLable的Text属性，此时你的代码应该如下所示
2. local button = script.Parent
3. local bridge = script.Parent.Parent
4. local text = workspace.timePart.SurfaceGui.TextLabel
5. local function Bridgeon()
6. print("somebody touched the button")
7. button.BrickColor=BrickColor.Green()
8. bridge.Transparency=0
9. bridge.CanCollide=true
10. bridge.BrickColor = BrickColor.Green()
11. end
12. button.Touched:Connect(Bridgeon)
13. 在自定义函数的内部进行程序的增添，这样会让定时功能与桥的开关绑定到一起，触碰到开始按钮后才会开始工作。我们需要完成定时功能，使其显示于TextLable上。读者可自行思考如何编写合适的代码。

代码示例：

1. local button = script.Parent
2. local bridge = script.Parent.Parent
3. local text = workspace.timePart.SurfaceGui.TextLabel
4. local function Bridgeon()
5. print("somebody touched the button")
6. button.BrickColor=BrickColor.Green()
7. bridge.Transparency=0
8. bridge.CanCollide=true
9. bridge.BrickColor = BrickColor.Green()
10. for i =5,0,-1 do
11. text = i
12. wait(1)
13. end
14. bridge.Transparency = 0.8
15. bridge.CanCollide = false
16. timerText.Text = ""
17. end
18. button.Touched:Connect(Bridgeon)
19. 开始游戏测试，功能正常。但是你会发现如果你持续在按钮附近与其接触，定时器会反复重启。这是因为每次触摸按钮时都会调用 for 循环并从头开始启动 for 循环。为了解决此问题，我们将引入debounce（防抖动）的算法。

#### Debounce（防抖动）算法的思维和实现方法

Debounce（防抖动）系统是一个用来防止函数运行次数过多的代码集合。这个概念来自于电子元件开关的接点弹跳现象，也就是开关开启时接点在多次弹跳后才能稳定下来，导致多个信号产生。该问题在 Roblox 中主要体现于 BasePart.Touched 事件：在两个部件短时间内多次触碰时会产生类似问题。不过，防抖动系统在其他情况下也能够派上用场。

例如：当当希望制作一个触碰后有伤害的part的时候，抖动很可能导致一次性接触非常多次从而使得玩家瞬间死亡。防抖动的基本工作原理是：当特定动作发生时（例如当玩家跳上按钮上或者接触某个BasePart时），脚本会封锁所有对函数的再次调用，直到指定时长后或者动作完成后为止。

例程如下：

1. local buttonPressed = false
2. *--将按钮是否被按下的状态储存在局部变量中*
3. local function hit()
4. if not buttonPressed then
5. *-- 状态是否为“未按下”？*
6. buttonPressed = true
7. *-- 若是，则将其标记为“已按下”，防止函数多次运行*
8. print("按钮被按下")
9. wait(1)
10. print("Hi :D")
11. *-- 执行操作*
12. buttonPressed = false
13. *-- 执行操作后将其标记为“未按下”，以便再次执行函数*
14. end
15. end)
16. Workspace.Button.Touched:Connect(function

使用一个布尔值储存当前状态，对当前状态进行一次if then的判断，在指定时间内的其他碰撞将会被无视，从而达到目标效果。更高级的防抖动算法可以在官方的API文档中查阅，详见[防抖动：使用时机及原因说明 (roblox.com)](https://developer.roblox.com/zh-cn/articles/Debounce)

在此例中，我们也可以进行相关防抖动的设计。

1. 为防止计时器持续重新启动，你需要添加其他可控制是否允许再次调用 startTimer() 的变量。此变量将使用称为布尔值的新类型的值。布尔值的编写方式与其他变量相同，但并不使用数字或字符串，而只能将其设置为 true 或 false。在这种情况下，脚本在启动计时器之前将使用布尔值来检查计时器当前是否正在运行。因为当前还没有人按下按钮，请在脚本顶部创建名为 timerActive 的变量，并将其设置为 false。用于表示计时未开始
2. 为确保函数只会在timerActive为false时被调用，我们需要向if添加 timerActive==false的条件。在函数开始调用时，将其值改为true进行封锁。在函数执行完成后重新将timerActive置false复位，从而达到debounce的效果。

#### 代码示例

1. local button = script.Parent
2. local bridge = script.Parent.Parent
3. local text = workspace.timePart.SurfaceGui.TextLabel
4. local timerActive  = false
5. local function Bridgeon()
6. if timerActive ==false then
7. timerActive=true
8. print("somebody touched the button")
9. button.BrickColor=BrickColor.Green()
10. bridge.Transparency=0
11. bridge.CanCollide=true
12. bridge.BrickColor = BrickColor.Green()
13. for i =5,0,-1 do
14. text = i
15. wait(1)
16. end
17. bridge.Transparency = 0.8
18. bridge.CanCollide = false
19. timerText.Text = ""
20. timerActive = false
21. end
22. end
23. button.Touched:Connect(Bridgeon)

#### 代码错误排查

·检查是否在函数Bridgeon（）的 if 语句中添加了等待秒条件。

·确保在函数开始之前将 timerActive 设置为 true，在函数结束之前将timerActive置false。

#### Debounce（防抖动）实战

制作一个被玩家触碰时使玩家生命值减少 5 的部件。通过防抖动功能确保此效果每 3 秒只能触发一次。

例程：

1. *--将按钮是否被按下的状态储存在局部变量中*
2. local buttonPressed = false
3. script.Parent.Touched:connect(function(hit)
4. *-- 状态是否为“未按下”？*
5. if not buttonPressed then
6. *-- 若是，则将其标记为“已按下”，防止函数多次运行*
7. buttonPressed = true
8. if hit.Parent then
9. hum = hit.Parent:FindFirstChild("Humanoid")
10. hum.Health = hum.Health - 5
11. end
12. wait(3)
13. *-- 执行操作后将其标记为“未按下”，以便再次执行函数*
14. buttonPressed = false
15. end
16. end)

## 使用 Elseif 和 Else 的多条件判断

编程语言的基本思维包括循环、判断、封装等等，If、else和elseif的判断逻辑是编程中的重要一环。

在开发过程中，有时需要使用 if 语句处理多个可能的结果。比如在比赛中，你可能要根据玩家完成赛程的速度颁发不同的奖牌。你可以分别为授予玩家的每一块奖牌编写一条 if 语句，但是这样很耗时。一种更快捷的方法是只编写一条 if/then 语句，然后使用关键字 elseif 提供选择条件来测试第一个条件是否为“true”。

当 if/then 语句运行时，它会从顶部开始顺序往下判断，并仅在找到第一个为“true”的条件后立即运行代码。下面是使用 elseif 和不使用 elseif 语句的代码对比。请注意，当使用 elseif 时，只需要输入关键字 end 一次，即可让代码变得更加简洁。



图3.1 elseif和if

### 3.1 通过一款实战小游戏来理解多条件判断

在本节的项目中，你将制作一款单人赛跑游戏，玩家可以根据完成比赛的速度而获得不同的奖牌。首先设置比赛的起点和终点线，然后编写为玩家计时和授予不同奖牌的脚本。

#### 创建部件和脚本

1. 创建一个名为 FinishLine 的锚定部件并附加一段脚本。为了更快速地进行测试，请将终点放在离起点很近的位置。你可以在完成脚本后移动终点线。
2. 在 ParkourScript 中，创建两个变量以储存完成赛程的耗时和终点线part
3. local timePassed = 0
4. local finishLine = script.Parent
5. 创建一个名为 finish() 的新函数，并编写一条随后对代码进行测试的 print 语句。
6. local timePassed = 0
7. local finishLine = script.Parent
9. local function finish()
10. print("touched the finish line")
11. end
12. 自行编写一个名为 finished() 的函数作为完成赛程后的判断函数，当玩家碰到终点线时，运行该函数。将 finishline 的 Touched 事件连接到 finished函数。
13. 测试finished函数是否正常被调用
14. 用同样的方法为开始按钮添加函数，并将其与Touched事件连接起来
15. 我们的游戏逻辑是：玩家触碰游戏开始按钮，游戏开始，计时器开始计时。玩家触碰finishline，游戏结束，计时器停止计时，同时判断timePassed的大小并分发奖励。在脚本开头添加一个防抖动的布尔值raceActive并置false。

我们首先进行赛程开始的函数编写。在游戏开始时，将用于监听游戏赛程状态的布尔值置true，并开始使用while循环对timePass计数。

1. local function begin(otherpart)
2. local human = otherpart.parent:FindFirstChildWhichIsA("Humanoid")
3. if human and not raceActive  then
4. raceActive = true*--此时raceactive置true了，race开始*
5. print ("the race begins")
6. while raceActive == true do*--只当raceactive为true时，才会开始计数*
7. wait(1)
8. timePassed = timePassed +1
9. print  (timePassed)
10. end
11. end
12. end

8． 测试begin函数是否正常运作

9． 编写finish函数，在触碰到finishline时，调用函数。判断此时timePass的值处于哪个区间，根据区间分配不同奖励。

1. local function race(otherpart2)
2. local human2 = otherpart2.parent:FindFirstChildWhichIsA("Humanoid")
3. if human2 and raceActive then
4. print("You finished in " .. timePassed.." second")
5. if timePassed <= 10 then
6. print("You get a gold medal in "..timePassed)
7. elseif timePassed > 10 and timePassed <= 20 then
8. print("You get a silver medal in "..timePassed)
9. elseif  timePassed > 20 and timePassed <= 30 then
10. print("You get a bronze medal in "..timePassed)
11. else
12. print("Try again please")
13. end
14. end
15. raceActive = false*--raceactive置false防抖动，同时结束计数器计数。*
16. end
17. end
18. end
19. 在脚本末端，分别将两个函数绑定至对应的part。在脚本内部适当的地方添加部分代码更改finishline的透明度和cancollide属性以及cantouch属性以完善游戏显示，开始游戏测试功能

#### 代码示例

1. *--------------------初始化------------------------*
2. local finishPart = script.Parent.Parent.finishPart
3. local beginPart = script.Parent
4. local timePassed = 0
5. local raceActive = false*--防抖动初值为fales*
6. *--------------------终点线初始化------------------*
7. finishPart.CanTouch = false
8. finishPart.Transparency = 1
9. *--按下开始按钮之后的函数，添加了一个计数器，每秒加一并自我打印。全局使用了一个防抖动raceactive，可以兼容计时器的开始结束监听功能*
10. local function begin(otherpart)
11. local human = otherpart.parent:FindFirstChildWhichIsA("Humanoid")
12. if human and not raceActive  then
13. raceActive = true*---此时raceactiv置truerace开始*
14. print ("the race begins")
15. finishPart.CanTouch = true
16. finishPart.Transparency = 0*--开始之后把终点线置正常*
17. while raceActive == true do*--只当raceactive为true时，才会开始计数，触碰到终点线即停止计数*
18. wait(1)
19. timePassed = timePassed +1
20. print  (timePassed)
21. end
22. end
23. end
24. local function finish(otherpart2)
25. local human2 = otherpart2.parent:FindFirstChildWhichIsA("Humanoid")
26. if human2 and raceActive then
27. if  raceActive then
28. print("You finished in " .. timePassed.." second")
29. if timePassed <= 10 then
30. print("You get a gold medal in "..timePassed)
31. elseif timePassed > 10 and timePassed <= 20 then
32. print("You get a silver medal in "..timePassed)
33. elseif  timePassed > 20 and timePassed <= 30 then
34. print("You get a bronze medal in "..timePassed)
35. else
36. print("Try again!")
37. end
38. finishPart.CanTouch = false
39. finishPart.Transparency = 1
40. end
41. raceActive = false*--raceactive置false防抖动，同时结束计数器计数。当然这句代码也能加入到finish函数里面*
42. end
43. end
44. beginPart.Touched:Connect(begin)
45. finishPart.Touched:Connect(race)*--将两个触碰事件和两个函数连接起来*

#### 代码错误排查

·检查 else 语句是否未包含条件，例如后面漏掉了 timePassed <= 20 或 then 语句。

·在 if 语句中，最后应始终检查是否有 else 语句。另外，请检查 else 语句后是否包含任何其他条件，例如 if、elseif 或 else，还要检查是否漏掉了起始 if 语句的 end。

·每一个 elseif 的条件后都应有一个 then。示例：elseif timePassed > 10 then。

### 3.2 通过一款实战小游戏来理解排行榜和随机函数

#### 项目目标

设置一个会随机变色的Part部件，在接触到该部件后，部件会被摧毁，并向玩家提供相应的分数。为完成此目标，我们需要用到排行榜服务和一个会变色的part对象。

#### 游戏内排行榜系统

Roblox 为开发人员提供了内置的排行榜系统，可以用来显示如得分、游戏货币或比赛最快完成时间等玩家信息。每当有玩家进入游戏，就应该将其添加至排行榜中。

1. 在ServerScriptService中新建一个脚本。在脚本中编写一个函数，将玩家加入的事件连接至此函数
2. local Players = game:GetService("Players")
3. local function leaderboardSetup(player)
4. end
5. Players.PlayerAdded:Connect(leaderboardSetup)

2． 在用来连接的函数中创建新的 Folder（文件夹）实例并将其命名为 leaderstats（排行榜数据）。此实例的父项应为新加入的玩家，且此实例的名称必须与文中严格一致，使用任何其它名称的文件夹都无法让 Roblox 将其父项玩家添加至游戏内排行榜中。Roblox 排行榜使用值类对象（Value Type Objects）存储及显示玩家数据。下列脚本将会使用 IntValue（整数值的占位符）来显示玩家的游戏内金币数量。

1. local Players = game:GetService("Players")
2. local function leaderboardSetup(player)
3. local leaderstats = Instance.new("Folder")
4. leaderstats.Name = "leaderstats"
5. leaderstats.Parent = player
7. local gold = Instance.new("IntValue")
8. gold.Name = "Gold" *--排行榜的分数名称*
9. gold.Value = 0
10. gold.Parent = leaderstats
11. end
13. Players.PlayerAdded:Connect(leaderboardSetup)

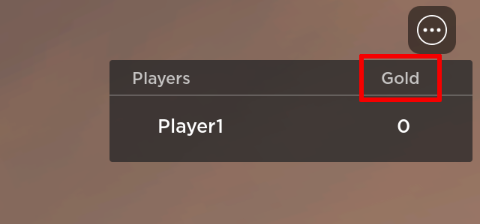


图3.2 排行榜示例

该段代码的含义是：

1.获取player服务

2.新建一个文件夹，严格命名为“leaderstats”，将其设置为player服务的子项

3.新建一个intvalue对象，将其命名为Gold，赋值为0，并将其设置为文件夹leaderstats的子项

4.每次有新玩家加入游戏时，就调用该函数，为每一个玩家建立排行榜

#### 设置部件和脚本

与其他循环部件教程不同，此脚本会将每种颜色存储在其自己的变量中以确定稍后在脚本中向玩家奖励的分数。每种颜色的分数值将存储在单独的变量中。



图3.3 分数与颜色示意

1. 创建名为 PointPart 的锚定部件以及名为 PointScript 的附加脚本。在 PointScript 中，将父部件存储在变量中。
2. 设置颜色的 RGB 值，蓝色会向玩家奖励少量分数。创建名为 blue 的新变量并将其设置为 Color3.fromRGB()。要创建颜色，请在Color3.fromRGB（）的括号中输入 0, 0, 255（即蓝色的 RGB 值）。接着将绿色和红色都设置完成
3. 设置三种颜色代表的三种不同分数。使用三个不同变量，分别设置为10，50，-100
4. 若要获得或者失去分数，我们需要访问之前创建的玩家的Gold信息，该信息存储在管理器中的 Players（玩家）服务下，并且与角色对象分开。在这里你可以找到排行榜统计信息。

你可以通过在脚本中添加 Players service（玩家服务）来实现该目标。Services（服务）是 Roblox为节省你的时间而额外制作的一系列预构建函数。通过输入以下内容来获取玩家服务：local Players = game:GetService("Players")。至于要对服务变量重命名的原因是，虽然其他变量以小写字母开头，但诸如 Players 、SoundService服务之类的一切服务会以大写字母开头，告诉你该变量是一个服务。

1. 根据我们的项目目的，我们将会需要用到两个函数，一个用于判断接触并销毁给分部件，一个作为给分函数。

创建一个名为 givePoints() 的新函数以及一个名为 player 的参数。在函数内部，添加一个 print 语句以用于测试。  
创建名为 partTouched() 的第二个函数，其中包含名为 otherPart 的参数。使用GetPlayerFromCharacter函数来查找接触部件的玩家。如果玩家接触了部件，该玩家将被存储在 player 变量中。如果没有接触部件，变量将保留为空。

1. local function givePoints(player)
2. print("Giving player points")
3. end
4. local function partTouched(otherPart)
5. local player = Players:GetPlayerFromCharacter(otherPart.Parent)
6. end

在其中输入print函数进行debug。

1. 在函数内部，检查 player 是否具有值。如果有，则调用 givePoints(player)。在函数下面，将 partTouched() 连接到 pointPart 的 Touched 事件。进行测试并检查输出窗口是否符合预期。
2. 为了遍历颜色，脚本将使用 while true do 循环，每隔几秒种更改一次部件的颜色。在脚本底部使用while true do 循环，循环改变part的颜色。如果 while true do 循环不在脚本的底部，则它下面的任何代码将永不运行。由于 while 循环没有停止，它将永远继续运行循环，而不是运行在其后面的任何代码。
3. 进行游戏测试并检查所有三个颜色是否循环且没有停止。
4. 现在，该部件将循环遍历颜色，它需要向玩家奖励分数。因为每种颜色奖励不同数量的分数，脚本将使用 if 语句来检查触摸时激活的颜色，并根据该颜色给出分数。在玩家获得适当数量的分数奖励之前，你需要设置变量以捕捉该部件在玩家触摸它时显示的颜色以及玩家已经拥有的分数。

使用

1. local function givePoints(player)
2. local currentColor = pointPart.Color
3. end

用于捕捉玩家触摸part时，part的颜色。然后创建变量以获得玩家的排行榜和Gold值（排行榜的子级）

1. local function givePoints(player)
2. local currentColor = pointPart.Color
4. local playerStats = player:WaitForChild("leaderstats")
5. local playerPoints= playerStats:WaitForChild("Points")
6. end
7. 判断当前的currentColor，并给玩家给分。在完成给分后将目标对象销毁，以便脚本只运行一次，并且不会继续奖励分数。。
8. if currentColor == blue then
9. playerPoints.Value = playerPoints.Value + smallPoints
10. elseif currentColor == green then
11. playerPoints.Value = playerPoints.Value + largePoints
12. else
13. playerPoints.Value = playerPoints.Value - losePoints
14. end
16. pointPart:Destroy()

#### 代码示例

1. local pointPart = script.Parent
3. local blue = Color3.fromRGB(0, 0, 255)
4. local green = Color3.fromRGB(0, 255, 0)
5. local red = Color3.fromRGB(255 ,0, 0)
7. *-- 奖励给玩家的分数*
8. local smallPoints = 10
9. local largePoints = 50
10. local losePoints = 100
12. local Players = game:GetService("Players")
13. local function givePoints(player)
14. local currentColor = pointPart.Color
15. local playerStats = player:WaitForChild("leaderstats")
16. local playerPoints= playerStats:WaitForChild("Gold")
18. *-- 奖励给玩家的分数基于部件颜色*
19. if currentColor == blue then
20. playerPoints.Value = playerPoints.Value + smallPoints
21. elseif currentColor == green then
22. playerPoints.Value = playerPoints.Value + largePoints
23. else
24. playerPoints.Value = playerPoints.Value - losePoints
25. end
26. pointPart:Destroy()
27. end
29. local function partTouched(otherPart)
30. local player = Players:GetPlayerFromCharacter(otherPart.Parent)
31. if player then
32. givePoints(player)
33. end
34. end
36. pointPart.Touched:Connect(partTouched)
38. *-- 改变基于变量上的部件颜色。需在脚本底部进行。*
39. while true do
40. pointPart.Color = blue
41. wait(3)
42. pointPart.Color = green
43. wait(2)
44. pointPart.Color = red
45. wait(1)
46. end

#### 额外项目：向玩家提供反馈效果

玩家在接触到part并给分后，向玩家提供一些视觉效果的反馈，让他们明白刚刚发生了什么或是接触到的part是什么颜色。为显示其工作原理，可以在销毁 PointPart 后添加跟随玩家的彩色粒子。

在玩家使用某个部件（如声音、摇动或粒子）时添加反馈，可以让玩家获得更满意的对象交互。、

1. 创建粒子效果。粒子效果的颜色将与部件被触摸时显示的颜色相同。销毁部件后，在 givePoints() 中创建新的 ParticleEmitter 实例，并将其附着于玩家角色的头部部件下作为子项。请确保实例名称的拼写与引号中所示完全一致。
2. pointPart:Destroy()
3. local particle = Instance.new("ParticleEmitter")
4. particle.Color = ColorSequence.new(currentColor)
5. particle.Parent = player.Character:WaitForChild("Head")
6. 等待几秒钟后，销毁这个粒子发射器，让特效只持续几秒
7. local particle = Instance.new("ParticleEmitter")
8. particle.Color = ColorSequence.new(currentColor)
9. particle.Parent = player.Character:WaitForChild("Head")
11. wait(1)
12. particle:Destroy()Head")

#### 代码错误排查

·检查 while 循环是否位于脚本底部、Touched 事件的下面。如果该循环不在底部，它会使脚本的其他部分无法正常运行。

·检查 Color3.fromRGB() 中的每种颜色是否输入正确。它们必须是三个介于 0 到 255 之间且用逗号分隔的数字，例如 (255, 50, 0)。

·确保创建新实例时，ParticleEmitter 的拼写与引号中所示完全一致。

·将粒子用作父级时，确保在 playerCharacter 和 WaitForChild() 之间使用 : 且中间没有空格。

## 4. 表、数组和字典

表（Table）是一种将多个值组合到一起的方法。与仅存储单个数字或字符串的变量相比，表格不具有固定大小，并且可混合不同的值类型。表(Table)是Lua语言中最主要(事实上也是唯一的)和强大的数据结构。使用表，可以以一种简单、统一且高效的方式表示数组、集合、记录和其他很多数据结构。通过使用表格，你可以将游戏内物品的统计数据组合到一起，或者创建一个包含数千个玩家名称的列表。Lua语言中的表本质上是一种辅助数组( associative array ),这种数组不仅可以使用数值作为索引，也可以使用字符串或其他任意类型的值作为索引( nil除外)。

### 4.1 通过一个对话NPC理解数组

表格有不同类型。一种是数组，即以特定顺序存储值的列表。要更好地理解表格，我们现在来完成这样的项目：创造一个点击按钮会说话的机器人：

#### 获取部件和创建脚本

1. 打开工具箱，在类型框中选择模型。键入NPC，挑选一个满意的NPC模型并拖入workspace中。在NPC模型下新建一个clickDetector对象作为模型的子项。另外新建一个名为DialogueScript的脚本，复制或输入如下代码：
2. local Chat = game:GetService("Chat")
4. local npc = script.Parent
5. local head = npc.Head
6. local clickDetector = npc.ClickDetector
7. local dialogueArray= {}
9. local function speak()
10. local dialogue = "This is dialogue!"
11. Chat:Chat(head, dialogue)
12. end

代码的第1行是获取游戏中的Chat服务，在Studio中，你需要的任何服务都需要提前获取。代码的3~5行相信读者已经很熟悉了，这里不作过多赘述。代码的第7行就是创建空数组的一种常用方法。clickDetector是一个特殊的对象，它必须附着于一个父项上才能正常工作。它可以检测基本的鼠标事件：进入、离开、左键单击和右键单击

1. 在数组dialogueArray的括号 {} 之间，至少输入三个用逗号分隔的对话字符串。本例中使用
2. local dialogueArray= {"welcome~","This is a dialogue","see u next time"}
3. 现在数组已创建，你需要采取一种方式来显示其中的值。输入 print(dialogueArray)并测试输出，可以观察到输出窗口不会打印出数组，而只会输出一系列数字。这不符合我们的预期。要从数组中获取其中的内容，脚本需要知道该内容在数组中的位置。只要将值添加到数组中，就会为其分配一个数字。此数字称为索引。索引从 1 开始，每添加一个值就会增加一个数。例如，我们输入的字符串中，"welcome~" 位于索引 1，而 "see u next time" 位于 3。

注意：有编程基础的同学可能发现了，Lua语言与其他语言不太相同，Lua语言的表和数组的索引值是从1开始而不是从0开始的，而C语言等大部分语言都是从0开始的，这是Lua语言比较重要的特征之一。

现在我们需要在特定索引处使用值，请在数组名称后面添加索引，不包含任何空格，例如 dialogueArray[1]，这样将会索引到数组dialogueArray中索引号为1的值，即“welcome~“。

1. 测试数组。输入 = dialogueArray[2] 将变量 dialogue 设置为第二个索引。
2. local function speak()
3. local dialogue = dialogueArray[2]
4. Chat:Chat(head, dialogue)
5. end

注意：如果使用不存在的索引，你将收到 nil 报错。例如，上述数组长度仅为 3，因此，尝试获取 dialogueArray[4] 将导致错误。  
开始游戏测试对话，点击NPC，NPC头部将会冒出聊天气泡，气泡内容应为“This is a dialogue”

#### 代码错误排查

·如果收到类似is not a valid member of 的报错，请检查你的项目管理器中的结构层次是否正确。

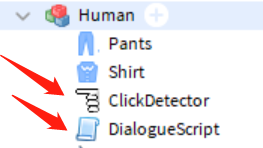


图4.1 脚本父子项结构层级关系

·检查你的符号是否为英文符号，包括引号、括号和方括号、花括号以及冒号和逗号等。

·检查你的索引值是否超出了设定数组的长度。

#### 进阶用法

自行设置索引值确实是个好方法，但是大多数情况下这样太笨拙了。可以使用变量代替索引号，以便使代码更灵活。例如，每次调用函数以获取数组中的下一个值时，都可使用变量来获取随机索引号或增加索引号。

##### 使用循环思维和if/else来遍历数组

首先明确我们的目的：每当玩家单击 NPC 时，作为索引值的变量将增加 1 以显示下一行对话。

1. 要跟踪当前索引，请创建名为 Index的新变量。如果要从数组的开头开始，请将变量设置为等于 1。
2. local dialogueArray= {"welcome~","This is a dialogue","see u next time"}
3. local Index =1
4. 在speak（）函数内部，使用刚创建的变量替换dialogueArray[2]中的索引值2，
5. local function speak()
6. local dialogue = dialogueArray[Index]
7. Chat:Chat(head, dialogue)
8. end
9. 在函数内部、调用数组完成之后、函数结束之前的任意处，输入如下代码：
10. Index = Index +1

该代码的作用是：完成函数调用时，我们将索引值加一，然后将其投入到第二次点击事件中作为下一轮的索引值。从效果上来说，就好像每次点击都导致了索引值自增了1，同时根据此索引值输出了数组。

我们还需要考虑数组长度的问题，本例中我们的数组长度只有3，那么当Index自增到4的时候，将会引起报错。此时我们可以添加一段if/else函数来让Index自动复位：

1. if Index == #dialogueArray then
2. Index = 1
3. else
4. Index = Index + 1
5. end

其中，#dialogueArray的含义是数组dialogueArray的长度，通过在数组的变量名称前输入不含空格的 #（例如 #dialogueArray），可以找到数组的大小。一旦数组中有许多元素，你可以根据变量的当前值对其进行检查，以了解何时从头开始。在本例中即为3

1. 开始测试，点击npc，观察对话框现象。

##### 代码示例

1. local Chat = game:GetService("Chat")
3. local npc = script.Parent
4. local head = npc.Head
5. local clickDetector = npc.ClickDetector
7. *-- 存储要说的对话列表*
8. local dialogueArray= {"welcome~","This is a dialogue","see u next time"}
9. local Index = 1
11. *-- 点击 NPC 时显示新对话框*
12. local function speak()
13. local dialogue = dialogueArray[Index]
14. Chat:Chat(head, dialogue)
15. *-- 如果增量位于数组的末尾，请将其重新设置为开始*
16. if Index == #dialogueArray then
17. Index = 1
18. *-- 否则，只需将1加到增量即可继续在数组中向上移动*
19. else
20. Index = Index + 1
21. end
22. end
24. clickDetector.MouseClick:Connect(speak)

##### 代码错误排查

·检查 if 语句，确保 dialogueIndex 重新设置为 1。在 else 语句中，检查 dialogueIndex 是否已向自身添加了 1。

·获取数组的大小时，确保 #dialogueArray的#号后没有空格。

##### 自主挑战

1. 让你的数组倒序循环而不是顺序循环
2. 使用math.Random()函数进行随机取值让你的数组随机取值而不是按照顺序或者倒叙。

### 4.2数组的增删改查

数组的内容经常会发生更改，如从玩家物品栏中移除某个物品，或者将玩家姓名添加到列表中。Lua 提供了预先构建的函数，可轻松与表一起使用。详细信息请参阅API文档：[table (roblox.com)](https://developer.roblox.com/zh-cn/api-reference/lua-docs/table)，这里只对常用功能进行介绍。

#### 数组的增添

要将新值添加到数组，请使用 table.insert(array, value)第一个参数是你想要进行修改的表的名字。第二个参数可以为任意类型的值，包括字符串、数字、甚至一个新的table。

1. 新建一个空脚本，在其中创建一个空的数组。使用table.insert()函数为其中添加你想要的东西。
2. 复制下列代码并粘贴至你的数组后方，尝试阅读和理解这段代码，review代码会对你的代码能力有很大帮助
3. for index = 1, *#Array do*
4. local somevalue = Array[index]
5. print("Index " .. index .. ": " .. itemString)
6. end

值得一提的是：要将字符串与变量或其他字符串合并在一起，你可以使用一种称为串联的处理方法，即在字符串与变量名之间输入 ..

1. 检查你的代码，开始测试。观察输出窗口的输出

#### 数组的删除

要删除数组中的最后一个值，请输入 table.remove(arrayName)。

1. 在数组的增添一段中，修改代码。在循环打印之前、数组增添之后输入如下代码：
2. table.remove(Array)
3. 开始测试，观察输出窗口

要按照索引删除数组的值，请将索引作为第二个参数添加到table.remove(arrayName, Index)

1. 在上一步的代码的基础上做少量修改，删除固定索引的某个值而不是数组的最后一个值。
2. 开始测试，观察输出窗口。

需要注意的是，所有的索引值也已经出现了改变。从数组中移除值时，将会更改数组的大小并将移除项后的索引全部向前移动一位。

#### 数组的更改

数组的更改本质上是修改对应索引值的value，在此基础上我们可以通过直接给Array[Index]赋值来完成。读者可以自行尝试，此处不作过多赘述

#### 数组的查找

如果希望在数组中找到特定值，则需要使用 for 循环 和 if 语句自行编写搜寻函数。Roblox 内目前没有此类内置功能。

在数组中寻找单一值时，需要创建名为 findValue() 的函数，使其遍历数组，直到第一次找到符合值为止。找到该值后，函数将会使用关键字 return 传回该值的索引。开发者可以利用索引对该值进行删除等操作。

1. 复制下方名为 playerItems 的数组
2. local playerItems = {
3. "Water",
4. "Bread",
5. "Bread",
6. "Gun"
7. }

2. 编写一个名为 findValue() 且包含两个参数的新函数：Array和itemName。这将告诉脚本，需要在哪个数组中搜寻哪个对象。

1. local function findValue(Array, itemName)
3. end

3. 在 findValue() 中，使用一个 for 循环检查数组中是否有任何值匹配 itemName。如果有匹配，则使用关键字 return 返回 Index （即索引）的值。

1. local function findValue(Array, itemName)
2. for Index = 1, #Array do
3. if Array[Index] == itemName then
4. return Index
5. end
6. end
7. end

4. 调用写好的函数，为函数提供数组名称和搜寻的对象的名称两个参数，使用一个新的变量将函数返回的结果容纳起来方便输出（当然你也可以直接输出而不用将其容纳）。

1. local valueFound = findValue(playerItems, "Bread")
3. print("The value is at index " .. valueFound)

获取了目标对象的索引值之后，我们可以用前面学到的方法对数组进行增删改查的处理。此处不作过多赘述，读者请自行进行尝试。

### 4.3 字典

字典是使用键而非编号索引来识别值的表。值存储与该键相关的数据。需要标记值时使用字典，而不仅仅像数组那样按顺序列出值。数组和字典都是表的一类，区别在于它们的索引值一个是数字一个是键。

和数组一样，字典使用大括号 {} 创建。键值对存储在单独的行中，后跟逗号。键和值可以是任何数据类型 - 包括字符串、数字和变量名。键和值统称键值对。

#### pairs 和 iPairs

pairs() 和 ipairs() 是可以与 for 循环一起使用的函数，可遍历数组或字典中的每个元素，而无需设置起点或终点。pairs() 与字典一起使用，ipairs() 与数组一起使用。ipairs() 中的“i”代表“index（索引）”。

1. 复制以下代码，使用脚本进行测试，同时阅读并尝试理解其用法：

1. local Dictionary = {
2. ["A"] = "character",
3. ["1"] = "Num",
4. ["dog"] = "Animal",
5. }
7. for key, value in pairs(Dictionary) do
8. print(key .. " is " .. value)
9. end

其中，for key value in pairs可以用for k，v in pairs代替。在这个遍历循环中，k即为字典的键，v即为字典的值。

2. 复制以下代码，使用脚本进行测试，同时阅读并尝试理解其用法：

1. local Array = {"A", "B", "C"}
2. for index, value in ipairs Array) do
3. print(value .." is " .. Index)
4. end

其中，for index， value in ipairs可以用for i，v in ipairs代替。在这个遍历循环中，i即为数组的索引，v即为对应的值。

#### 自主挑战

根据本节所学知识，完成以下项目：

1.有两个篮子，分别为A和B。篮子A中装有一个Apple，一个Orange，一块Bread，一根Banana；篮子B中什么也没有。

2使用字典来表示篮子A中物品，键使用物品名称，值使用布尔值。使用数组来表示篮子B中的物品。

3使用我们用过的数组机器人来完成操作，点击机器人，随机从篮子A中取出一个物品放入篮子B中，同时更改字典A和数组B的内容。