

Python 编程硬件 教程

好好搭搭在线

目 录

第一部分 硬件入门

第一节 初识 Python 编程硬件.....	2
示例 1-1：点亮 LED	3
第二节 玩转点阵屏	6
示例 2-1：点阵屏绘制图案	6
示例 2-2：点阵屏显示文本	7
示例 2-3：点阵屏显示数字	7
第三节 数字量输出及其控制	9
示例 3-1：LED 的闪烁	9
示例 3-2：交通灯（红绿灯）	11
第四节 数字显示屏	13
示例 4-1：用数码管显示数字	13
第五节 模拟量输入与采集	15
示例 5-1：环境光检测仪	18
示例 5-2：超声波测距仪	20
示例 5-3：温湿度计	21
示例 5-4：DS18B20 温度计	23
第六节 模拟量的输出	25
示例 6-1：LED 调光	25
示例 6-2：旋钮调光	29
第七节 电机的驱动	32
示例 7-1：可调速电风扇	32
第八节 蜂鸣器	33
示例 8-1：外接蜂鸣器	35
示例 8-2：内置蜂鸣器播放两只老虎	36
示例 8-3：内置蜂鸣器演奏歌曲	36
第九节 RGB 七彩灯	42
示例 9-1：RGB 交通灯	43

第二部分 编程入门

第十节 编程基础知识介绍	47
第十一节 数字量输入与条件判断指令	49
示例 11-1：用点阵屏显示开关量传感器的状态，以按钮开关为例。	51
示例 11-2：门铃	55
第十二节 变量	58
第十三节 数学与逻辑运算	61
示例 13-1：华氏温度计	62
示例 13-2：光控灯（光敏开关版）	65
示例 13-3：夜间声控灯	67
第十四节 随机数	69
示例 14-1：摇号机（基本版）	69
第十五节 新建函数	72
示例 15-1：RGB 交通灯（功能块版）	72

第三部分 硬件进阶

第十六节 红外遥控接收	78
示例 16-1：红外遥控 LED	79
示例 16-2：用点阵屏显示红外遥控器的键值	80
第十七节 电子罗盘	83
第十八节 加速度计	85
示例 18-1：计步器制作	85
第十九节 MP3 音乐播放	88
第二十节 炫酷点阵屏	88
示例 20-1：按坐标点亮点阵模块	89
示例 20-2：音乐动感点阵屏——柱状图	89
第二十一节 蓝牙远程控制	93
示例 21-1：蓝牙控制 LED	99

第四部分 编程进阶

第二十二节 计时器	109
示例 22-1：长按点亮 LED，松开熄灭 LED	109
第二十三节 输入计数及其应用	111
示例 23-1：记录按键按下松开的次数，并显示到点阵屏上	111
示例 23-2：点控 LED	113
示例 23-3：按键控制切换 RGB 灯的颜色	116
示例 23-4：亮度传感器控制 LED 灯的亮灭（点控）	119

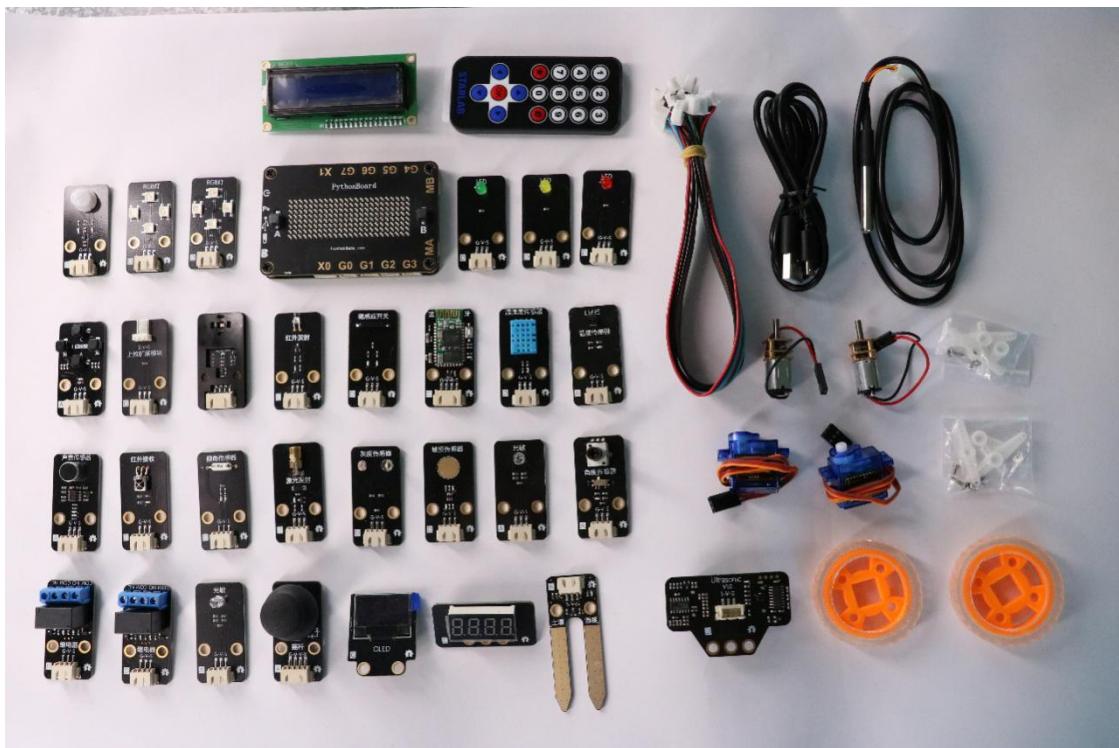
第一部分

硬件入门

第一节 初识 Python 编程硬件

一、认识 Python 编程硬件

Python 编程硬件是一款支持 Python 语言编程的硬件套件。



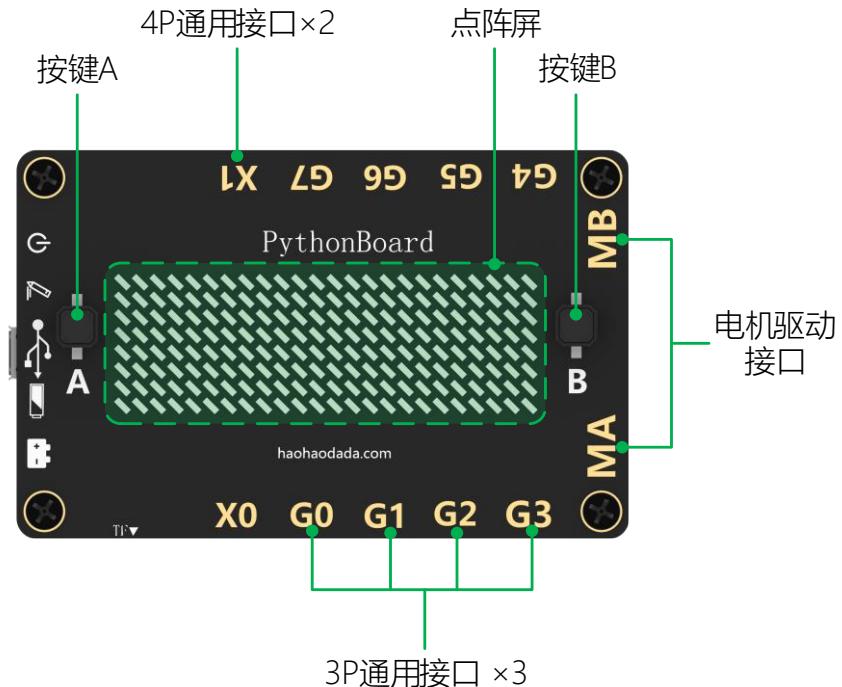
新一代图形化编程云平台，图形编程界面与字符编程界面之间方便切换、共存，可满足各种不同的学习场合。

The screenshot shows the PythonBoard graphical programming environment. On the left is a sidebar with categories: 基础功能, 控制, 数学与逻辑, 文本与数组, 变量, 函数, 点阵屏, 板载, 扩展. The main area has tabs for 点阵屏清除 (Dot Matrix Clear) and 重复执行 (Loop). A speech bubble says "点阵屏显示文本" with the code "“haohaodada”". The right side shows a code editor with the following Python script:

```
1 Screen1=Screen()
2
3
4 Screen1.clearScreen()
5 while 1:
6     Screen1.printText('haohaodada')
```

PythonBoard 是一款优秀的单机开发套件，内置电子罗盘、加速度计、按键、蜂鸣器、7×24 点阵 LED 等输入输出器件。

扩展接口提供了 2 个 4P 通用接口、8 个 3P 通用接口以及两个电机驱动接口。



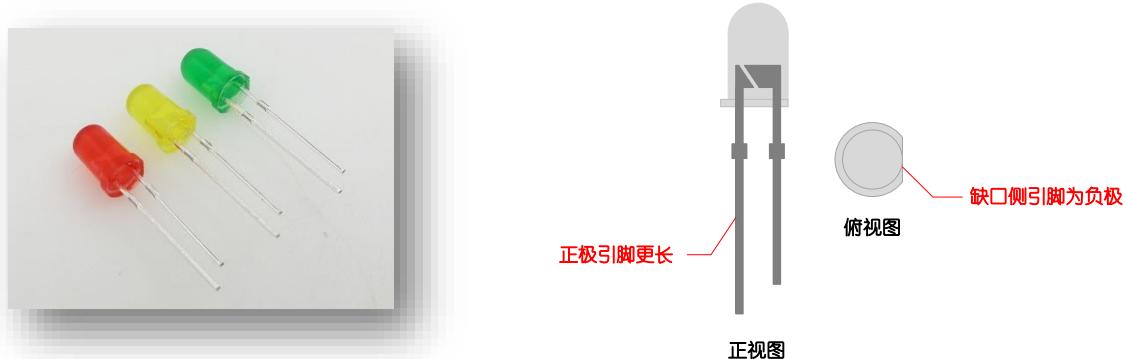
接下来用一个案例，亲身体验 Python 编程硬件套件的使用。

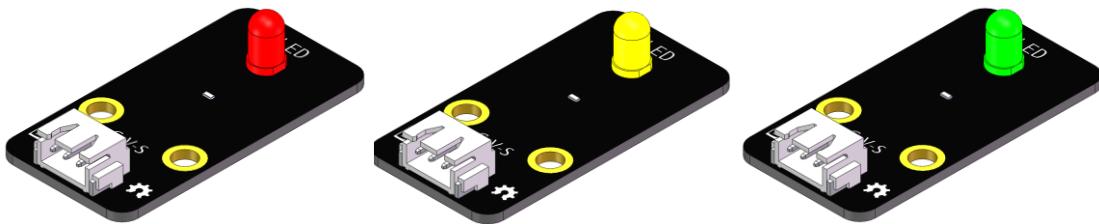
示例 1-1：点亮 LED

用 PythonBoard 点亮一盏 LED。

认识新模块：

LED 发光二极管：由含镓（Ga）、砷（As）、磷（P）、氮（N）等的化合物制成。具有单向导通性，即 LED 有电流正向流入能点亮，反向流入或无电流则不亮。辨别发光二极管的正负极方法如右图：





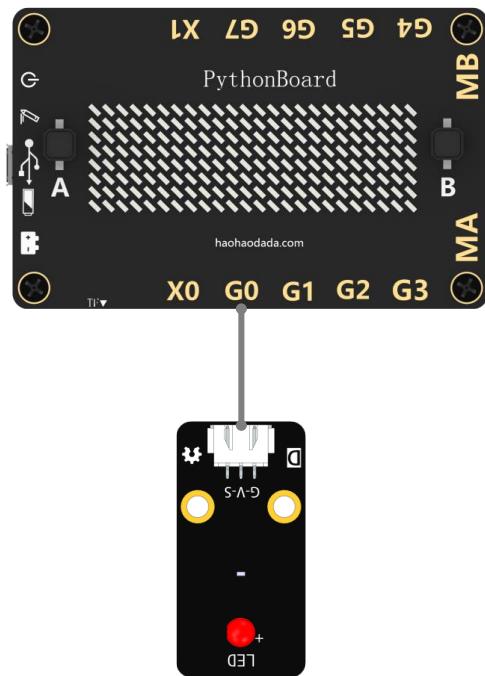
元器件列表:

PythonBoard 主控板 ×1

LED 模块（红） ×1

3Pin 连接线 ×1

电路连接:



进入编程界面:

登入网址：“<http://haohaodada.com/pythonboard>”

接下来开始编程吧！

编写第一个程序——点亮 LED 灯

认识新指令——设置数字量输出指令

所以需要使用数字量相关指令，对于主控板 PythonBoard 来说，点亮 LED 是由 PythonBoard 输出信号给 LED，因此使用“设置数字口...输出为...”指令：

设置数字口 S0 ▼ 输出为 低 ▼

设置端口

点击“G0”，弹出下拉菜单，选择数字口：



选择端口的原则是：**与硬件连接一致！**本例中 LED 模块是连接到 PythonBoard 的 G0 接口，所以指令选择“G0”。

输出可选择“低”或“高”，指的是输出为“低电平”或“高电平”，低电平关灭 LED，高电平点亮 LED，所以本例中选择“高”



那么本例中点亮 LED 的程序为：



```
1 G0=Port('G0')
2
3
4 G0.digitalWrite(1)
```

下载程序：

点击“下载程序”按钮，将程序下载到 PythonBoard 中。

运行程序：

PythonBoard 在下载程序之后，并不能自动运行程序，需等待电源开关旁的 RGB 灯熄灭之后，方可按压电源开关运行程序。

完成！可以看到 LED 发光二极管亮起。

第二节 玩转点阵屏

点阵屏在生活中随处可见，商业中心的外墙大屏幕、商店招牌、火车站候车信息等等。

PythonBoard 上集成有一个 7×24 的点阵屏，可以用来显示图案、文本和数字。

示例 2-1：点阵屏绘制图案

认识新指令——点选设置点阵屏显示



使用该指令可以设置 PythonBoard 板载 LED 点阵屏显示图案。用鼠标单击指令中间的显示区，显示黑色■的格子表示选中、相应位置的 LED 被点亮；再次单击黑色■消失、相应位置的 LED 熄灭。

指令上方是图案显示属性设置区域；主要有以下几项设置内容：

- “滚动方式”设置：默认为“无”，单击下拉列表可以选择“↑”、“↓”、“←”、“→”四种滚动方向；
- “滚动速度”设置：默认是“3”，可以在方框内输入其它数值，数值的取值范围是 $(0, 15)$ ，数值越小，滚动速度越快。
- “显示图案”设置：默认是“**I♥HD**”，可以单击下拉列表选择“雪花”、“全选”、“清零”。

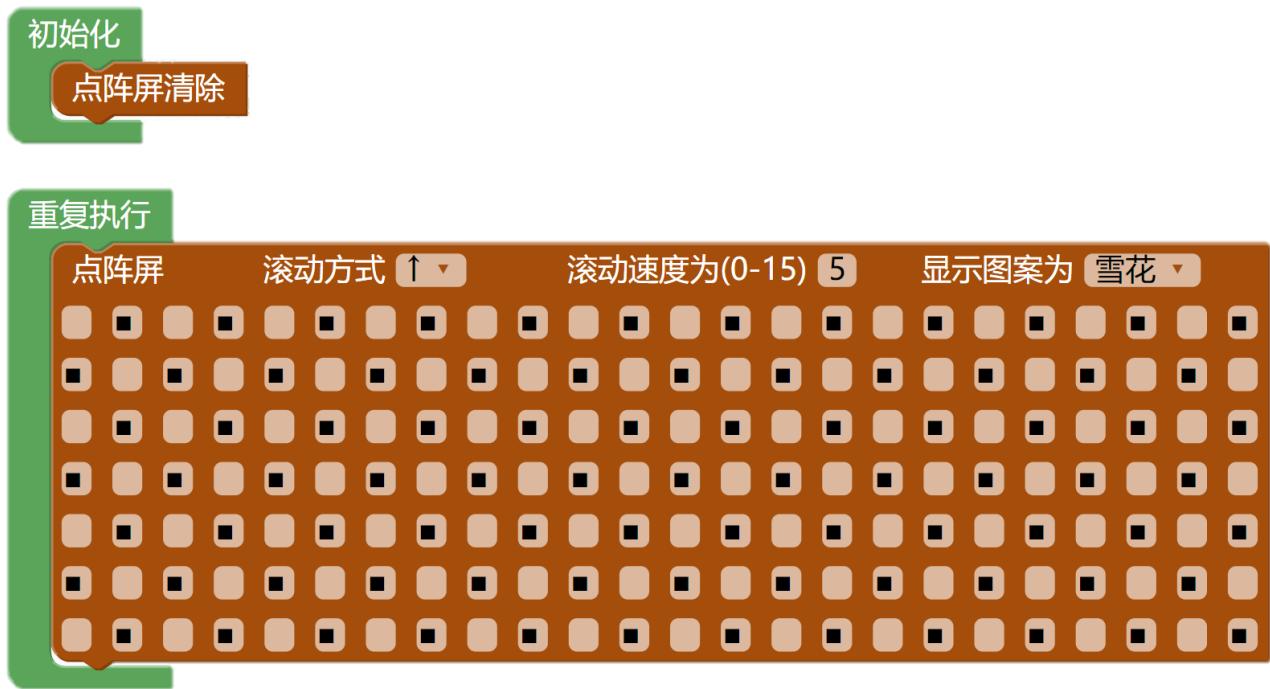
认识新指令——点阵屏清除指令

点阵屏清除

使用这个指令可以清除 PythonBoard 点阵屏原来的显示内容，为显示新的内容做准备。

当用到点阵屏时，通常会在初始化程序中加入一条点阵屏清除指令，以清除之前可能残留的显示信息。

程序编写：



示例 2-2：点阵屏显示文本

认识新指令——点阵屏文本显示指令

点阵屏显示文本 “YES”

该指令可以设置 PythonBoard 点阵屏显示指定的字母；默认显示“YES”，可以根据需要输入。由于 PythonBoard 一屏只能显示四个字母，当输入字母数超过四个，就会以滚动方式显示。

程序编写：

The screenshot shows the PythonBoard Scratch-like interface. On the left, there's a grey rounded rectangle labeled "点阵屏清除" (Clear Point Matrix Screen). Below it is a brown rounded rectangle labeled "点阵屏显示文本" (Print Text on Point Matrix) with a teal sub-box containing the text "Hello World!". On the right, there's a purple code editor window with the following code:

```
1 Screen1=Screen()  
2  
3  
4 Screen1.clearScreen()  
5 Screen1.printText('Hello World!')
```

点阵屏文本显示指令中的文本仅支持英文字母和数字，标点符号的输入也必须在英文输入法下进行，如上图程序中的感叹号。

示例 2-3：点阵屏显示数字

认识新指令——点阵屏数值显示指令

点阵屏显示数 123

该指令可以设置 PythonBoard 点阵屏显示指定的数；默认显示“123”，可以根据需要输入。由于 PythonBoard 一屏只能显示四个数字，当输入数字超过四个，会以滚动方式显示。

程序编写：



```
1 Screen1=Screen()
2
3
4 Screen1.clearScreen()
5 Screen1.printText(str(9527))
```

点阵屏数值显示指令可以用来读取各种传感器的数值，是一条使用频率非常高的指令。

第三节 数字量输出及其控制

在第一节课中大家已经掌握了 PythonBoard 的电路连接和图形编程器的基本操作，并实现了点亮 LED 的程序。本次课程用几个 LED 的案例让大家理解什么是数字量。

示例 3-1：LED 的闪烁

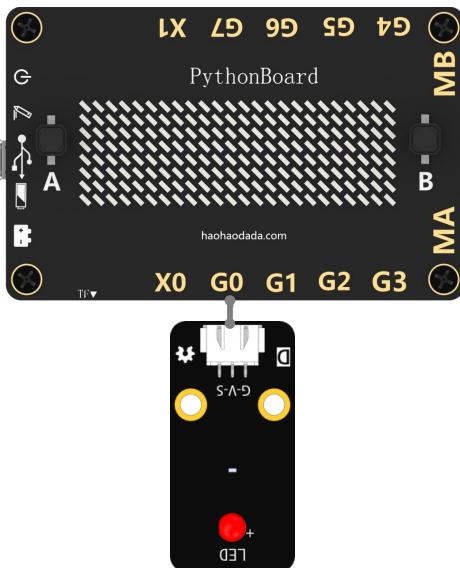
元器件列表：

PythonBoard 主控板 ×1

LED 模块（红） ×1

3Pin 2510 连接线 ×1

电路连接：



认识新指令——延时指令



延时指令的作用是让程序暂停执行一段时间，时间的长短可以设置。

认识新指令——重复执行指令



重复执行指令的作用是让其内部的程序不断重复的运行，永无止境。在好好搭搭硬件编程中，重复执行指令内部的程序被称为主程序。

程序分析:

我们来仔细思考下“闪烁”这个词，闪烁是灯亮一段时间，之后灭一段时间，如此循环往复。那么，“闪烁”应该被拆分成“亮”、“灭”、“持续一段时间”这些动作的组合。其中，“亮”和“灭”可以通过“数字输出赋值”指令实现：

数字输出 G0 赋值为 低

数字输出赋值为低指令的作用为熄灭 LED 灯。

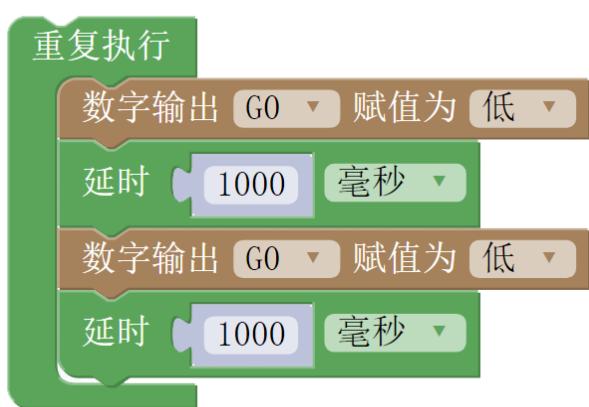
数字输出 G0 赋值为 高

相反的，数字输出赋值为高指令的作用为点亮 LED 灯。

“持续一段时间”需要使用延时指令。

不断的“闪烁”需要使用重复执行指令。

程序编写:



```
1 G0=Port('G0')
2
3 while 1:
4     G0.digitalWrite(0)
5     delay(1000)
6     G0.digitalWrite(1)
7     delay(1000)
```

拓展项目一：SOS 救援信号

背景知识：

S.O.S 是国际摩尔斯电码救难信号，并非任何单字的缩写。国际无线电报公约组织于 1908 年正式将它确定为国际通用海难求救信号。它的电码为“...---...”（三短三长三短）在电报中是发报方最容易发出，收报方最容易辨识的电码。

项目要求：

实现一个 LED 的闪烁步骤如下：

- (1) 连续短闪烁三次，单次闪烁时间间隔为 0.3 秒
- (2) 连续长闪烁三次，单次闪烁时间间隔为 1 秒
- (3) 连续短闪烁三次，单次闪烁时间间隔为 0.3 秒

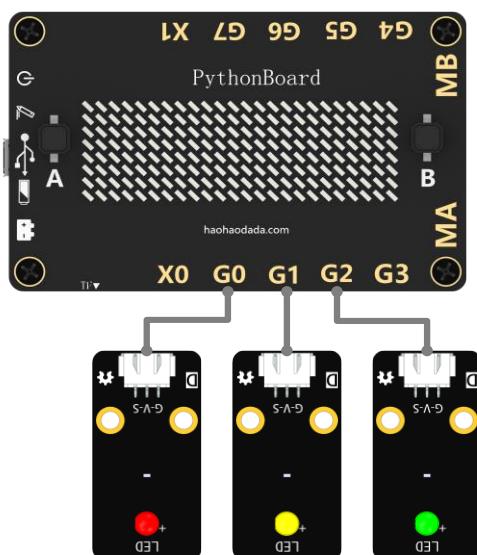
两个周期之间的时间间隔为 3 秒。

示例 3-2：交通灯（红绿灯）

元器件列表：

PythonBoard 主控板 ×1
 LED 模块（红） ×1
 LED 模块（黄） ×1
 LED 模块（绿） ×1
 3Pin 2510 连接线 ×3

电路连接：



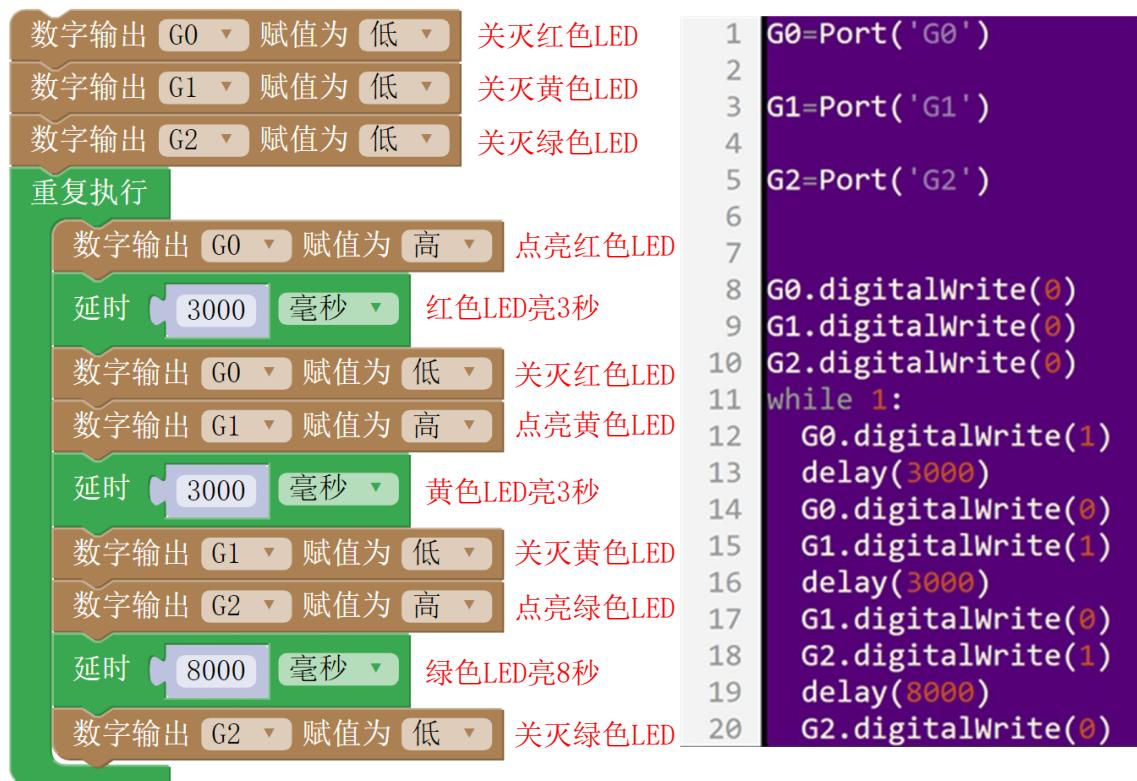
程序分析：

同学们可以去仔细观察或回想一下交通路口的红绿灯，它们是如何运作的：红黄绿三盏灯依次亮灭，各持续一段时间。程序的编写要点是用不同的端口控制不同的 LED 模块，实现依次亮灭的目的。

程序编写：

初始化程序：将三支 LED 灯全部关灭。

初始化程序是在运行开始时只执行一次的程序，位于重复执行指令之前。



第四节 数字显示屏

右图这种计算器相信大家都不陌生，这种液晶显示屏只能显示数字，简称为数码管。相对于电视或手机屏幕，这种显示屏使用上更加简单，当创客项目中仅需要显示数字而不需要显示文字、图片信息时，可以使用数码管来简单快速的实现。



认识数码管



有小数点位，既可以显示整数也能够显示小数。

认识新指令——数码管显示指令



数码管显示指令可以方便的将四位以内的数字显示到数码管上。显示整数选择“整数”模式，显示带小数位的数字（浮点数）选择“小数”模式。所显示的数字可以通过 修改。也可以使用赋值变量指令 修改显示的数值。

认识新指令——数码管清屏指令



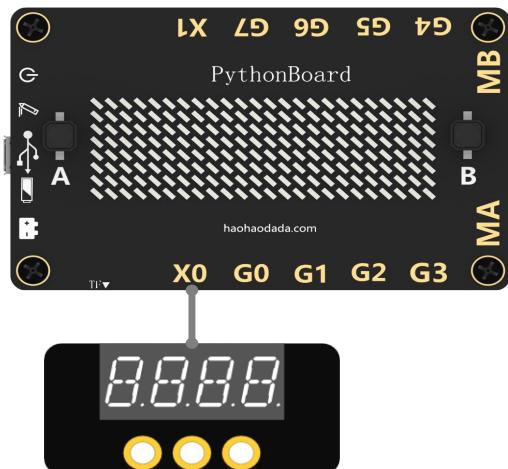
数码管清屏指令用于将数码管的显示全部清除，即将数码管上所有的 LED 关灭。

示例 4-1：用数码管显示数字

元器件列表：

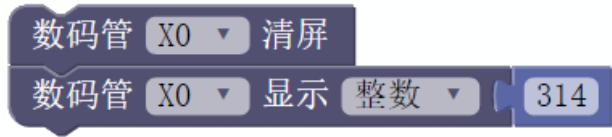
1. PythonBoard 主控板 ×1
2. 数码管模块 ×1
3. 4Pin 2510 连接线 ×1

电路连接:



程序编写:

显示一个整数:



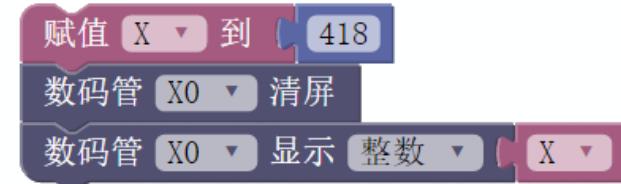
```
1 display_X0=DigitDisplay("X0")
2
3
4 display_X0.clearScreen()
5 display_X0.displayNum(314)
```

显示一个小数:



```
1 display_X0=DigitDisplay("X0")
2
3
4 display_X0.clearScreen()
5 display_X0.displayFloat(3.14)
```

显示一个变量:



```
1 display_X0=DigitDisplay("X0")
2
3
4 X = 418
5 display_X0.clearScreen()
6 display_X0.displayNum(X)
```

第五节 模拟量输入与采集

本次课将带大家认识传感器，传感器是一种检测装置，能感受到被测量的信息，并能将感受到的信息，按一定规律变换成为电信号或其他所需形式的信息输出，以满足信息的传输、处理、存储、显示、记录和控制等要求。

从传感器输入的数值看，一种是只有两个取值“0”和“1”的传感器，称为数字量传感器；另一种是有一个取值范围，如0到100、0到1023，称为模拟量传感器。

数字量传感器

详见第十一课《数字量输入与条件判断指令》

模拟量接口传感器

模拟量是指在一定范围连续变化的量，比如温度、亮度、距离、重量等。

模拟量传感器就是采集这类物理量，并将其转换为电压或电流信号的传感器。

认识新模块和新指令：

电位计模块：一种旋转变阻器，可以用作旋钮，可以感知角度的变化。



光敏传感器模块：用来检测当前的光照强度。



声音传感器模块：用来检测当前环境的声音强度。



以上三种模块都是模拟量传感器模块，输送到主控板中的是电压值，可以直接用“读模拟量”指令读取：

读模拟口 G0 ▾

主控板上能读取模拟量传感器的接口是 G0-G7。

另外还有一类传感器，它测量的是模拟量如距离、温度、湿度，但是输送给主控板的信号不是电压值，所以不能用读模拟口指令读取数值。好好搭搭为这类传感器设计了专用的指令。

超声波测距传感器：通过发射接收超声波来测量超声波模块与障碍物距离。图片和对应的指令如下：



读超声波传感器在 G0 ▾

温湿度传感器模块：用来检测当前环境的温度和湿度。下图中的蓝色塑料块即是温湿度传感器的核心器件——DHT11，所以对应的指令名称为“读 DHT11”，可以选择读温度和湿度（注意不能直接放入水中测量水温！）。当需要浸没在液态物质中测量时可选择温度传感器 DS18B20，所对应的指令名称为读 DH18B20 温度。



温湿度传感器模块 DHT11



温度传感器 DS18B20



用这些传感器配合数码管可以制作各式各样的测量仪表，如超声波测距仪、光照强度仪、噪音计

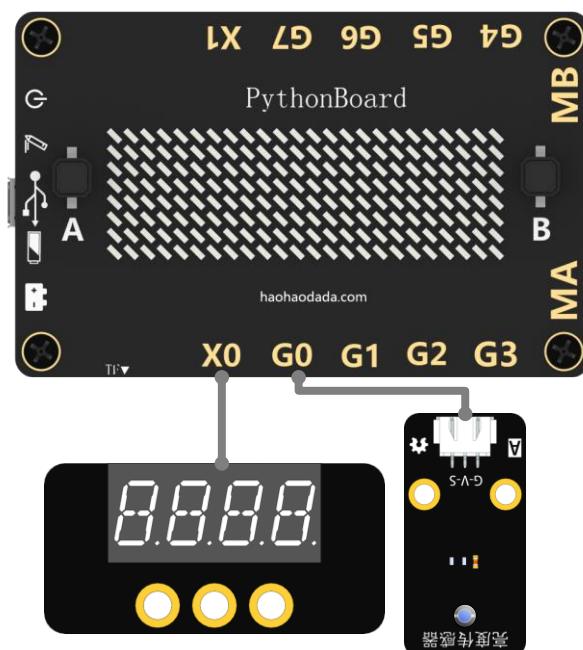
示例 5-1：环境光检测仪

用数码管实时的显示环境光的数值

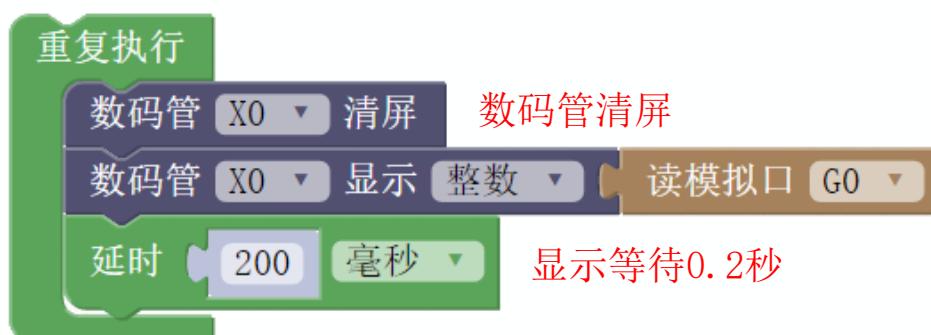
元器件列表：

1. PythonBoard 主控板 ×1
2. 亮度传感器模块 ×1
3. 数码管模块 ×1
4. 3Pin 2510 连接线 ×1
5. 4Pin 2510 连接线 ×1

电路连接：



程序编写：



将G0口的模拟值
显示在数码管上

```

1 display_X0=DigitDisplay("X0")
2
3 G0=Port('G0')
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayNum((G0.analogRead()))
9     delay(200)

```

以上程序即可实现环境光的检测，同学们试着测量各种情况下的亮度值。

环境情况	显示数值
白天房间里	
用阴影遮挡光敏	
直接盖在光敏上	
在灯光下	
在阳光下	

将传感器换成声音传感器，程序不变，即可制作环境声检测仪。同样，同学们也试着测量下各种环境中的声音响度值吧

环境情况	显示数值
比较安静的房间里	
2人正常对话	
大声喊叫	
教室里	
上体育课的操场上	

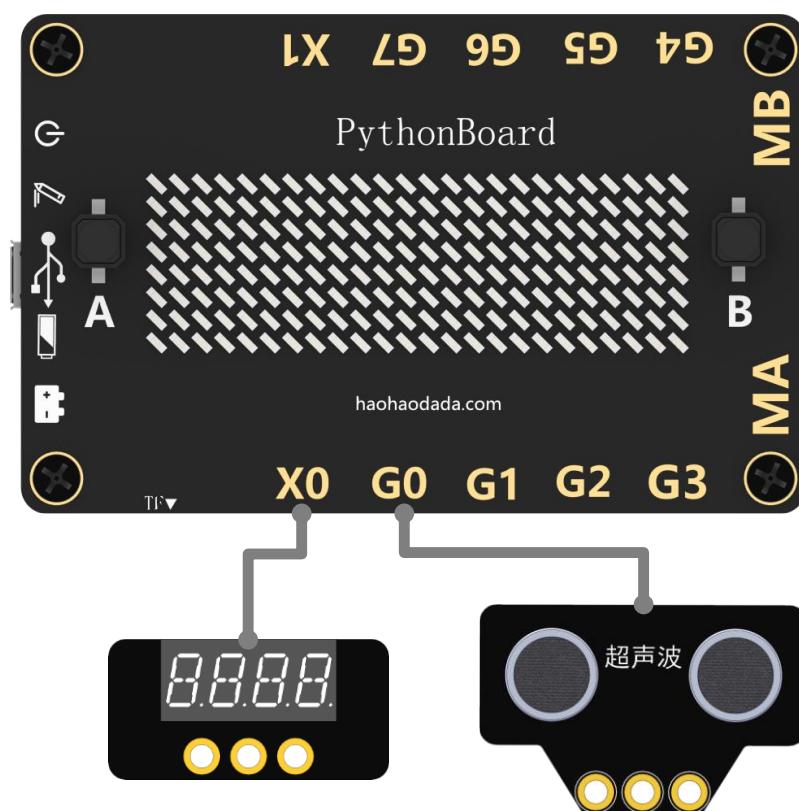
示例 5-2：超声波测距仪

用数码管实时的显示测量的距离值

元器件列表:

1. PythonBoard 主控板 ×1
 2. 超声波测距模块 ×1
 3. 数码管模块（计数） ×1
 4. 3Pin 2510 连接线 ×1
 5. 4Pin 2510 连接线 ×1

串路连接：



程序编写:

这里需要用超声波测距传感器模块的专用指令。



```
1 display_X0=DigitDisplay("X0")
2
3 Ultrasonic_0=Ultrasonic('G0')
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayNum((Ultrasonic_0.read()))
9     delay(200)
```

重复执行

```
1 display_X0=DigitDisplay("X0")
2
3 Ultrasonic_0=Ultrasonic('G0')
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayNum((Ultrasonic_0.read()))
9     delay(200)
```

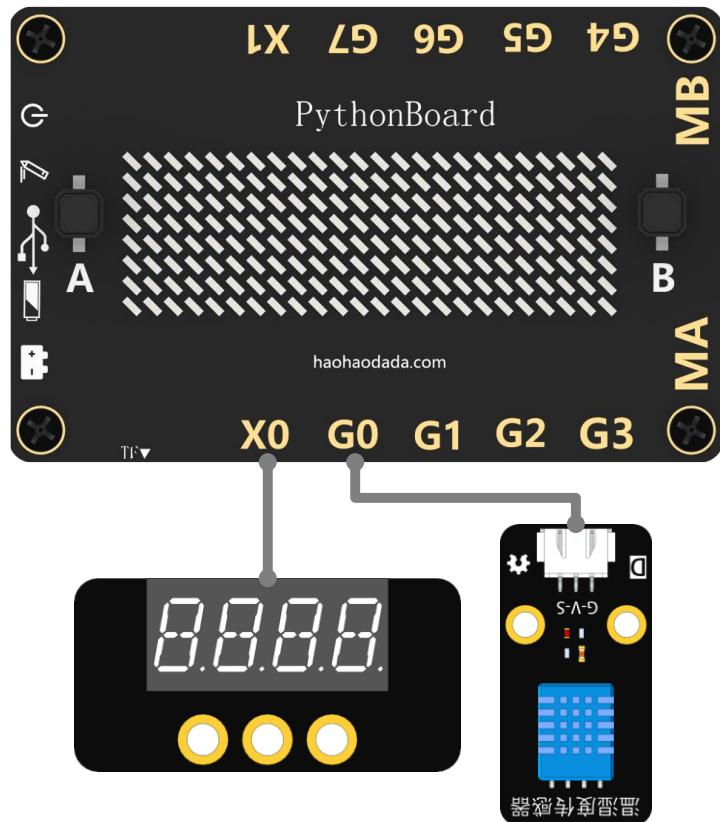
示例 5-3：温湿度计

用数码管实时的显示温度值/湿度值

元器件列表：

1. PythonBoard 主控板 × 1
2. 温湿度传感器 × 1
3. 数码管模块 × 1
4. 3Pin 2510 连接线 × 1
5. 4Pin 2510 连接线 × 1

电路连接：



程序编写:

这里需要用温湿度传感器的专用指令——读 DHT11。

```

repeat [ ]
    display_X0=DigitDisplay("X0")
    DHT11_0=DHT('G0','DHT11')
    while true:
        display_X0.clearScreen()
        display_X0.displayFloat((DHT11_0.readTemperature()))
        delay(200)
end

```



```
1 display_X0=DigitDisplay("X0")
2
3 DHT11_0=DHT('G0',"DHT11")
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayFloat((DHT11_0.readTemperature()))
9     delay(200)
```

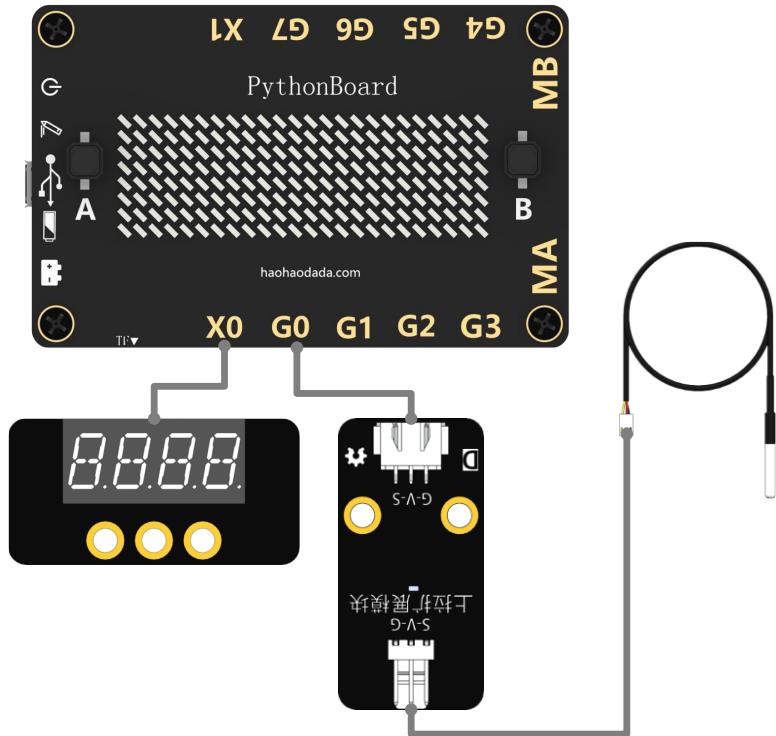
示例 5-4：DS18B20 温度计

用数码管实时的显示液体温度值

元器件列表：

1. PythonBoard 主控板 × 1
2. 温湿度传感器 × 1
3. 数码管模块 × 1
4. 3Pin 2510 连接线 × 1
5. 4Pin 2510 连接线 × 1

电路连接：



程序编写：

这里需要用液体温度传感器的专用指令——读 DS18B20 温度。



```

1 display_X0=DigitDisplay("X0")
2
3 DS_0=DS18B20('G0')
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayFloat((DS_0.read()))
9     delay(200)

```



```
1 display_X0=DigitDisplay("X0")
2
3 DS_0=DS18B20('G0')
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayFloat((DS_0.read()))
9     delay(200)
```

第六节 模拟量的输出

在第一、二课中同学们已经掌握控制 LED 亮灭的方法，那么 LED 除了开（完全点亮）和关（完全熄灭）两种状态之外，是不是能做到 LED 亮了，但是比完全点亮暗一些呢？

手机、平板可以说是大家使用最多的电子设备，大家会在晚上关灯后调低显示屏的亮度以保护眼睛，在白天强光下调高亮度以便看的更清楚。这些显示屏的背光光源都是 LED，LED 的调光是如何实现的呢？先从 LED 的闪烁说起。

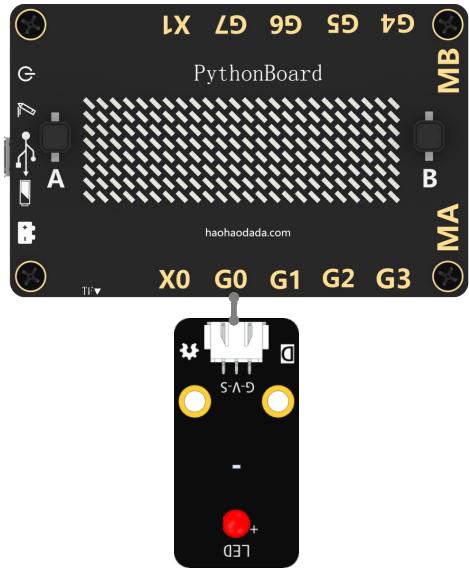
示例 6-1：LED 调光

在编程实现 LED 的调光之前，同学们可以先回顾下示例 2-1：LED 的闪烁。

元器件列表：

- 1.PythonBoard 主控板 × 1
- 2.LED 模块 × 1
- 3.3Pin 2510 连接线 × 1

电路连接：



LED 灯闪烁的程序已经实现，接下来做几个小实验：

小实验 1：LED 灯亮 100 毫秒，LED 灭 900 毫秒

程序：



```

1 G0=Port('G0')
2
3
4 while 1:
5     G0.digitalWrite(1)
6     delay(100)
7     G0.digitalWrite(0)
8     delay(900)

```

实验 1 直观结果：LED 闪烁，熄灭时间比亮起时间长。

小实验 2：LED 灯亮 10 毫秒，LED 灭 90 毫秒

程序：



```
1 G0=Port('G0')
2
3
4 while 1:
5   G0.digitalWrite(1)
6   delay(10)
7   G0.digitalWrite(0)
8   delay(90)
```

实验 2 直观结果：LED 快速闪烁，注意与实验 3 的直观结果做比较。

小实验 3：LED 灯亮 90 毫秒，LED 灭 10 毫秒

程序：



```
1 G0=Port('G0')
2
3
4 while 1:
5   G0.digitalWrite(1)
6   delay(90)
7   G0.digitalWrite(0)
8   delay(10)
```

实验 3 直观结果：LED 快速闪烁，看上去比实验 2 亮多了。

小实验 4：LED 灯亮 90 微秒，LED 灭 10 微秒

程序：



```
1 G0=Port('G0')
2
3
4 while 1:
5   G0.digitalWrite(1)
6   udelay(90)
7   G0.digitalWrite(0)
8   udelay(10)
```

实验 4 直观结果：LED 看上去完全不闪了，注意与实验 5 直观结果做比较。

小实验 5：LED 灯亮 10 微秒，LED 灭 90 微秒

程序：



```
1 G0=Port('G0')
2
3
4 while 1:
5     G0.digitalWrite(1)
6     udelay(10)
7     G0.digitalWrite(0)
8     udelay(90)
```

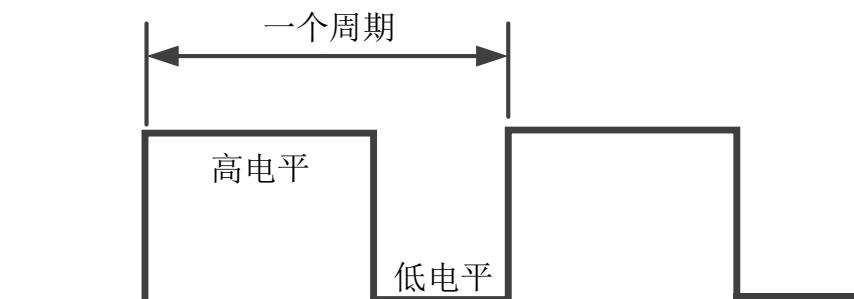
实验 5 直观结果：LED 看上去完全不闪，但是比实验 4 的亮度低很多。

大家可以再试试“0 100”“20 80”、“40 60”、“50 50”、“60 40”、“80 20”、“100 0”不同的组合，延时单位为微秒。可以看到这些参数组合的结果：亮度都不同。

结论：LED 的调光是通过在一定的时间间隔内，调节点亮和熄灭 LED 的相对时间来实现的。

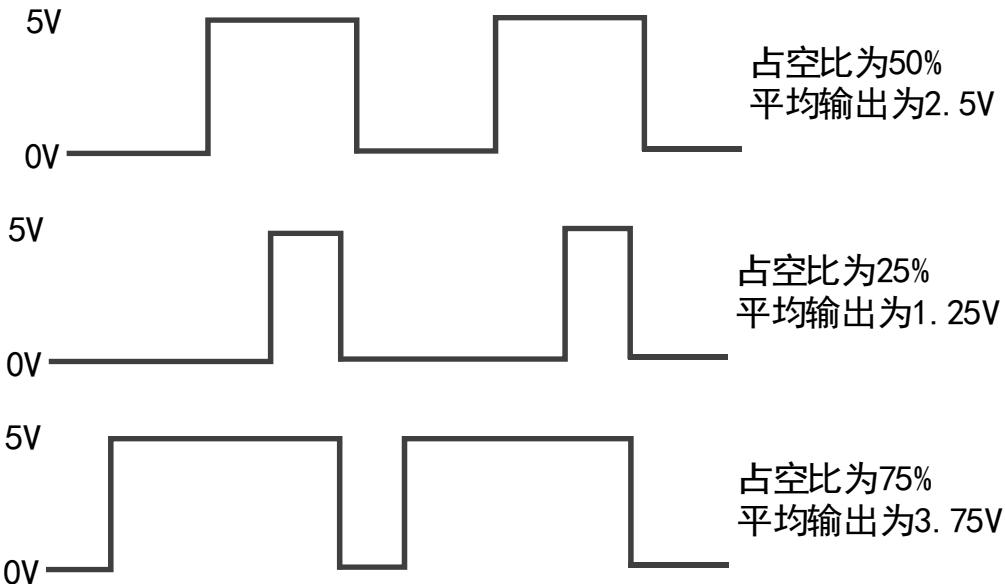
认识新概念——PWM

类似 LED 调光这种通过调节供电“通”和“断”的相对时间来改变平均输出大小方法，在工程上称为 PWM（脉冲宽度调制）。



其中，一个周期中高电平持续时间与低电平持续时间的比值，称为占空比。

如高电平电压为 5V，占空比 50% 的 PWM 信号的平均电压为 2.5V；占空比 25% 的 PWM 信号的平均电压为 1.25V；占空比 75% 的 PWM 信号的平均电压为 3.75V。



因为 PythonBoard 主控器运行程序是单线程的，如果在复杂程序里要调光，运行其他部分程序的时间也会被计算到点亮或熄灭 LED 的延时时间中去，导致亮度调节不准确。所以在复杂程序中需要实现调光功能时，需要调用 **PWM 输出指令**。

认识新指令——设置 PWM 输出指令

设置PWM口 G0 ▾ 输出为 (0-255) 10

PWM 输出指令能调用主控板的内部定时器，来实现平均输出的调节，无需使用等待或延时指令。PythonBoard 主控板上能输出 PWM 信号的端口是 G0-G7。

PWM 输出指令的数值范围为 0 到 255，输出数值为 0 时，输出电压为 0V，即一直为低电平，相当于数字口输出设置为 0；输出数值为 255 时，输出电压为 5V，即一直为高电平，相当于数字口输出设置为 1。

利用 PWM 输出指令来设置 LED 的亮度的程序为：

重复执行
设置PWM口 G0 ▾ 输出为 (0-255) 150

同学们试试不同的输出值，看看 LED 的亮度变化。

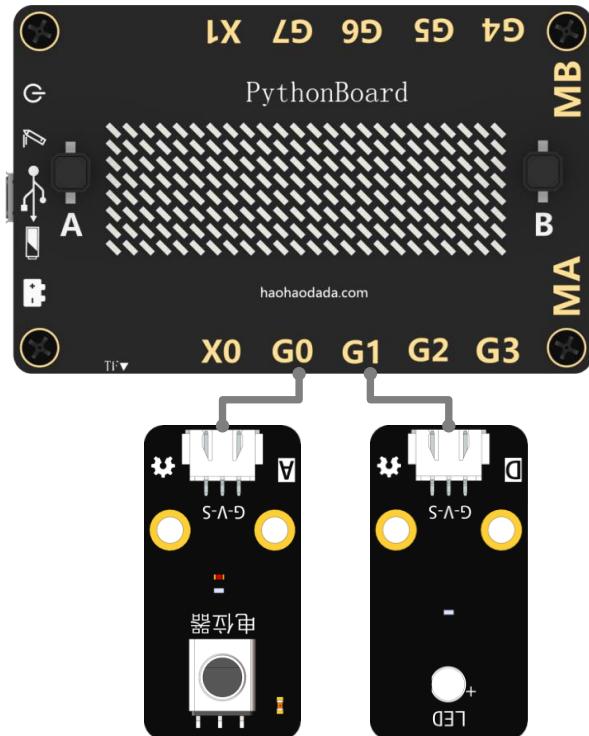
示例 6-2：旋钮调光

通过前面的内容，了解了模拟量输入和模拟量输出都是范围取值，那么如何用模拟量输入控制模拟量输出呢？

元器件列表:

1. PythonBoard 主控板 ×1
2. 电位器模块 ×1
3. LED 模块×1
4. 3Pin 2510 连接线 ×1
5. 3Pin 2510 连接线 ×1

电路连接:



程序编写:



```
1 import math
2
3 G1=Port('G1')
4
5 G0=Port('G0')
6
7
8 while 1:
9     G1.analogWrite((math.floor(G0.analogRead() / 16)))
```

其中电位器的取值范围为 0 到 4095，而 PWM 输出的取值范围为 0 到 255 的整数，所以从电位器模块读取到的模拟值需要经过换算并且为整数，让其最大值不超过 255，最小值不小于 0。

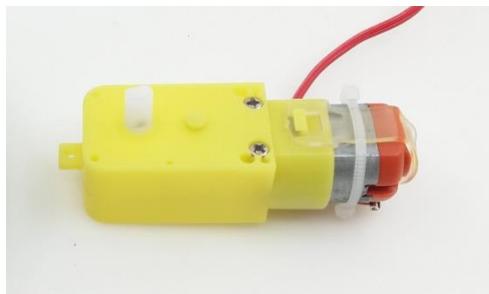
第七节 电机的驱动

电动机是将电能转换为机械能的一种执行器，通常简称为电机。让机构旋转起来，最简单直接的方法就是使用电机。

认识新模块——电机



130 电机（直流电机）



TT 电机（直流减速电机）

认识新指令——电机速度设置指令



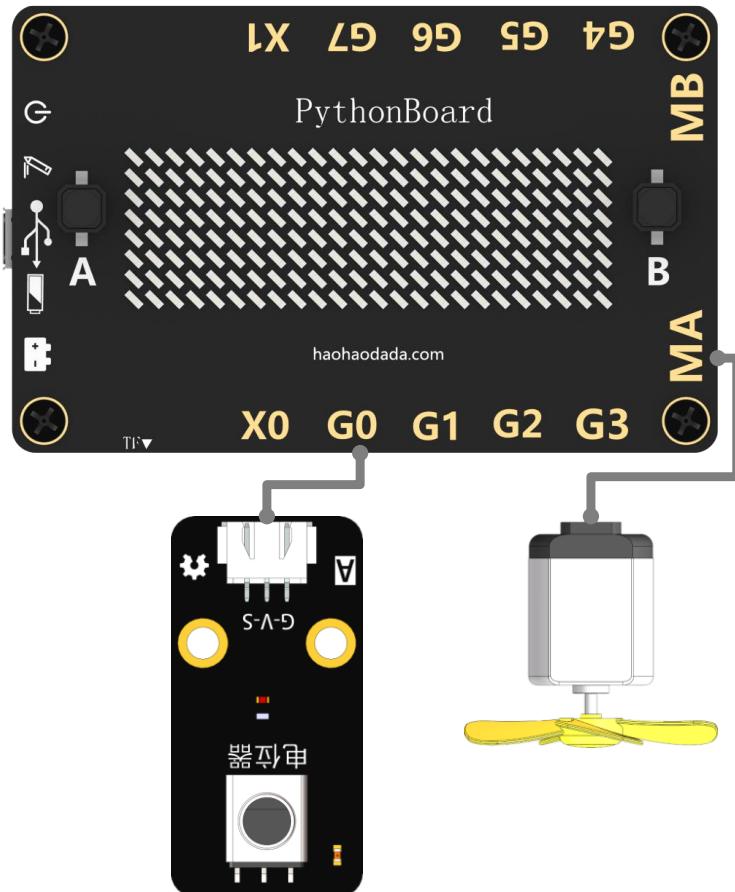
以上两种电机输出指令的数值输出范围均为-255 到 255。PythonBoard 能为电机提供驱动信号的端口为 MA、MB。

示例 7-1：可调速电风扇

元器件列表：

1. PythonBoard 主控板 × 1
2. 电位器模块 × 1
3. 130 电机 × 1
4. 风扇叶片 × 1
5. 4Pin 2510 连接线（红）× 1
6. 3Pin 2510 连接线（黑）× 1

电路连接：



程序编写

重复执行

电机 MA 速度 (-255~255) 向下舍入 读模拟口 G0 ÷ 16

```

1 import math
2
3 motor_MA=Motor("MA")
4
5 G0=Port('G0')
6
7
8 while 1:
9     motor_MA.run((math.floor(G0.analogRead() / 16)))

```

第八节 蜂鸣器

蜂鸣器模块用于发出一定频率的电子声。我们用它编制一首乐曲吧！

声音的三个主观属性分别是音量（响度）、音调和音色。音量指人耳感受到的声音强弱；音调指人的听觉能分辨一个声音的调子高低的程度；音色指声音的感觉特性，即根据不同的音色，即使在同一音高和同一声音强度的情况下，也能区分出是不同乐器或人发出的。

认识新模块——蜂鸣器



内置蜂鸣器

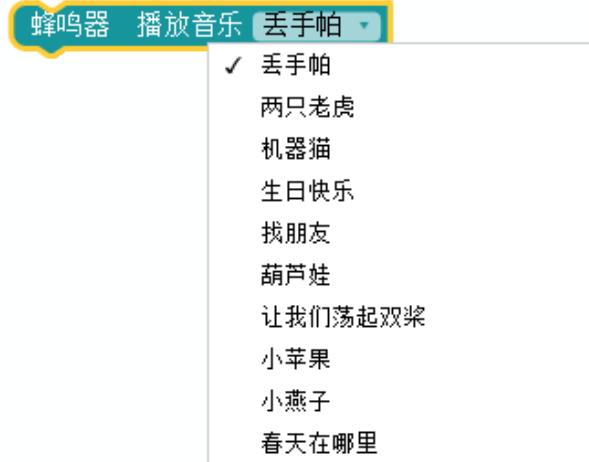
PythonBoard 内置一个蜂鸣器，并且平台图形编程命令中包含了部分音乐，可以直接调取播放。

认识新指令——蜂鸣器音调节拍设置指令



可以设置音乐的音符音调和节拍。

认识新指令——蜂鸣器播放音乐指令



可以播放这些音乐

认识新指令——蜂鸣器停止播放指令

蜂鸣器 停止播放

停止播放音乐

认识新指令——设置 PWM 口指令

设置PWM口 G0 ▾ 输出为 (0-255) 10

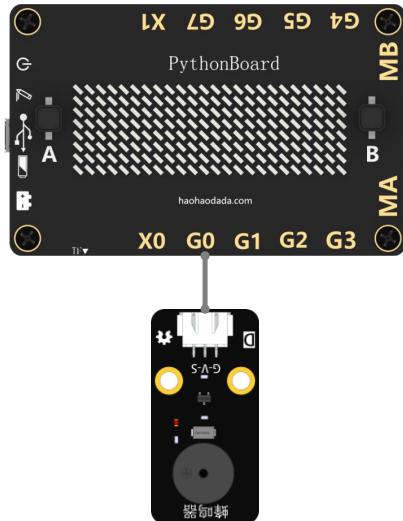
设置 PWM 频率指令是用于修改 PWM 周期的指令，从而实现修改蜂鸣器的音调。

示例 8-1：外接蜂鸣器

元器件列表：

1. PythonBoard 主控板 ×1

电路连接：



程序编写:



```

1 G0=Port('G0')
2
3
4 while 1:
5     G0.analogWrite(255)

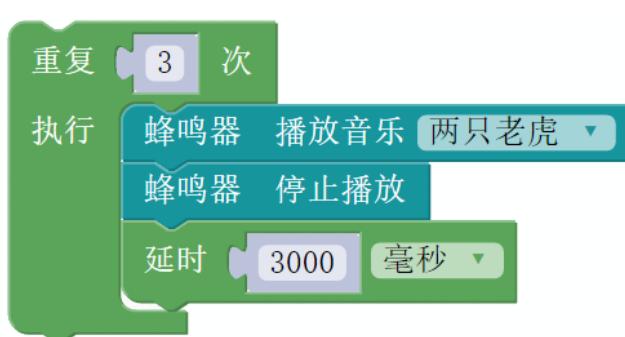
```

示例 8-2：内置蜂鸣器播放两只老虎

元器件列表:

2. PythonBoard 主控板 ×1

程序编写:



```

1 b=Buzzer()
2
3
4 for count in range(3):
5     b.music('Two tigers')
6     b.noTone()
7     delay(3000)

```

示例 8-3：内置蜂鸣器演奏歌曲

内置蜂鸣器鸣响程序范式:

重复执行

```
蜂鸣器 音调 Middle C 节拍 1/4  
蜂鸣器 音调 Middle D 节拍 1/4  
蜂鸣器 音调 Middle E 节拍 1/4  
蜂鸣器 音调 Middle C 节拍 1/4  
蜂鸣器 音调 Middle C 节拍 1/4  
蜂鸣器 音调 Middle D 节拍 1/4  
蜂鸣器 音调 Middle E 节拍 1/4  
蜂鸣器 音调 Middle C 节拍 1/4  
蜂鸣器 音调 Middle E 节拍 1/4  
蜂鸣器 音调 Middle F 节拍 1/4  
蜂鸣器 音调 Middle G 节拍 1/2
```

```
1 b=Buzzer()  
2  
3  
4 while 1:  
5     b.tone(262,250)  
6     b.tone(294,250)  
7     b.tone(330,250)  
8     b.tone(262,250)  
9     b.tone(262,250)  
10    b.tone(294,250)  
11    b.tone(330,250)  
12    b.tone(262,250)  
13    b.tone(330,250)  
14    b.tone(349,250)  
15    b.tone(392,500)
```

重复执行

- 蜂鸣器 音调 Middle C 节拍 1/4
- 蜂鸣器 音调 Middle D 节拍 1/4
- 蜂鸣器 音调 Middle E 节拍 1/4
- 蜂鸣器 音调 Middle C 节拍 1/4
- 蜂鸣器 音调 Middle C 节拍 1/4
- 蜂鸣器 音调 Middle D 节拍 1/4
- 蜂鸣器 音调 Middle E 节拍 1/4
- 蜂鸣器 音调 Middle C 节拍 1/4
- 蜂鸣器 音调 Middle E 节拍 1/4
- 蜂鸣器 音调 Middle F 节拍 1/4
- 蜂鸣器 音调 Middle G 节拍 1/2

重复执行

蜂鸣器 音调 Middle C	节拍 1/4	发出1/4节拍中音1
蜂鸣器 音调 Middle D	节拍 1/4	发出1/4节拍中音2
蜂鸣器 音调 Middle E	节拍 1/4	发出1/4节拍中音3
蜂鸣器 音调 Middle C	节拍 1/4	
蜂鸣器 音调 Middle C	节拍 1/4	
蜂鸣器 音调 Middle D	节拍 1/4	
蜂鸣器 音调 Middle E	节拍 1/4	
蜂鸣器 音调 Middle C	节拍 1/4	
蜂鸣器 音调 Middle E	节拍 1/4	发出1/4节拍中音3
蜂鸣器 音调 Middle F	节拍 1/4	发出1/4节拍中音4
蜂鸣器 音调 Middle G	节拍 1/2	发出1/2节拍中音5

```

1 b=Buzzer()
2
3
4 while 1:
5   b.tone(262,250)
6   b.tone(294,250)
7   b.tone(330,250)
8   b.tone(262,250)
9   b.tone(262,250)
10  b.tone(294,250)
11  b.tone(330,250)
12  b.tone(262,250)
13  b.tone(330,250)
14  b.tone(349,250)
15  b.tone(392,500)

```

发出音符

简 谱	1	2	3	4	5	6	7
音 名	C	D	E	F	G	A	B
唱 名	do	re	mi	fa	sol	la	si

音调（英文）	音调（简谱）	发音	频率	音调（英文）	音调（简谱）	发音	频率
C4	1	do	262	C5	1	do	523
D4	2	re	294	D5	2	re	587
E4	3	mi	330	E5	3	mi	659
F4	4	fa	349	F5	4	fa	698
G4	5	sol	392	G5	5	sol	784
A4	6	la	440	A5	6	la	880
B4	7	si	494	B5	7	si	988
C3	1	do	131				

D3	2	re	147				
E3	3	mi	165				
F3	4	fa	175				
G3	5	sol	196				
A3	6	la	220				
B3	7	si	247				

其中，C4-B4 为中音，是谱曲中最常用的基本音，C3-B3 为低音，C5-B5 为高音。

《两只老虎》的简谱如下：

两 只 老 虎

$1=\text{E} \frac{4}{4}$

中速

1 2 3 1 | 1 2 3 1 | 3 4 5 - | 3 4 5 - |

两只老虎，两只老虎，跑得快，跑得快，

5 6 5 4 3 1 | 5 6 5 4 3 1 | 1 5 1 - | 1 5 1 - ||

一只没有耳朵，一只没有尾巴，真奇怪，真奇怪。

要播放一首乐曲，除了要发出确定的音调，还需要有节拍的配合。如上图中



两拍的时间长度是一拍的 2 倍，半拍的时间长度是一拍的一半。

一拍的时间长度没有固定的限制，可以是 0.5 秒也可以是 2 秒，时间越短节奏越快。

那么在程序中通过设置蜂鸣器模块的“持续时间”来实现某个音的拍数

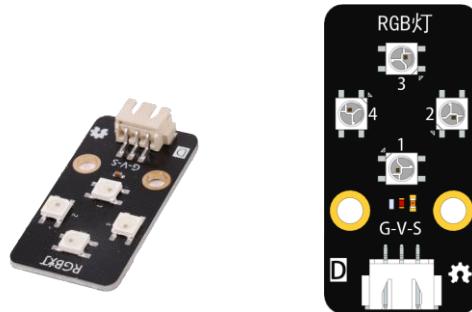
接下来开始编写曲子吧！

第九节 RGB 七彩灯

前面课程用到的 LED，只能发出固定颜色的灯光，当需要发出更多颜色的光时，可以使用 RGB 模块。很炫哦！

RGB 色彩模式是工业界的一种颜色标准，是通过对红(R)、绿(G)、蓝(B)三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色的，RGB 即是代表红、绿、蓝三个通道的颜色，这个标准几乎包括了人类视力所能感知的所有颜色，是目前运用最广的颜色系统之一。

认识新模块——RGB 模块



认识新指令——显示 RGB 彩灯指令



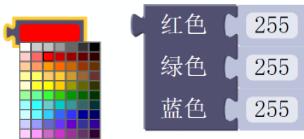
显示 RGB 彩灯指令是点亮 RGB 模块所必需的指令，即要点亮或修改 RGB 模块的颜色，必需使用显示 RGB 彩灯指令。

认识新指令——RGB 彩灯数量位置颜色设置指令



RGB 彩灯数量位置颜色设置指令是用于设置 RGB 彩灯个数位置和颜色的指令

认识新指令——RGB 彩灯颜色设置指令



在 PythonBoard 中 RGB 彩灯的颜色设置指令有两种，一种为直接选取标准色，另一种为设置为根据三基色原理，设置红绿蓝三基色参与混合的系数。

RGB 模块点亮程序：

重复执行

```

1  rgb_0=WS2812('G0',4)
2
3
4  while True:
5      rgb_0.colorSet(1,51,204,255)
6      rgb_0.colorSet(2,255,102,0)
7      rgb_0.colorSet(3,89,129,255)
8      rgb_0.colorSet(4,128,53,255)
9      rgb_0.show()

```

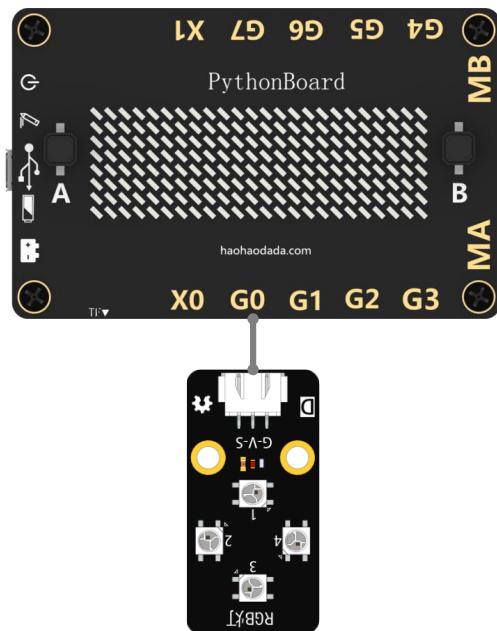
示例 9-1：RGB 交通灯

用 RGB 模块实现交通灯的红黄绿灯变换。

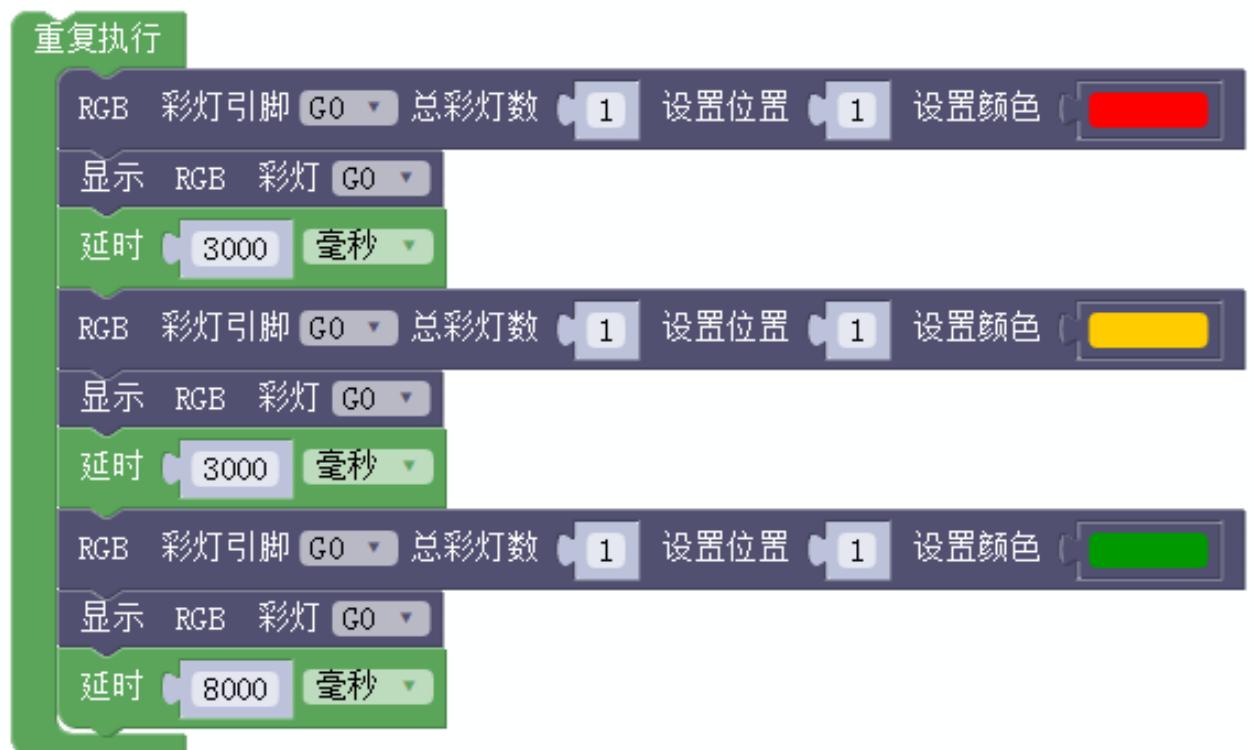
元器件列表：

1. PythonBoard 主控板 ×1
2. RGB 模块 ×1
3. 3Pin 2510 连接线 ×1

电路连接：



程序编写：



```
1  rgb_0=WS2812('G0',1)
2
3
4  while 1:
5      rgb_0.colorSet(1,255,0,0)
6      rgb_0.show()
7      delay(3000)
8      rgb_0.colorSet(1,255,204,0)
9      rgb_0.show()
10     delay(3000)
11     rgb_0.colorSet(1,0,153,0)
12     rgb_0.show()
13     delay(8000)
```

第二部分

编程入门

第十节 编程基础知识介绍

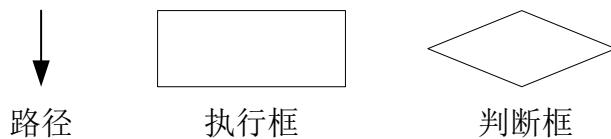
程序是一套人与计算机沟通的语言，既然是语言就有它一套独特的表达方式。

学习编程很像学习一门外语。然而，程序语言比任何一种人类语言都要古板，任何规范之外的表达方式都会被认为是错误的。这套规则来自一代又一代程序设计师的设定，随着时间的流逝，渐渐成为了计算机的行业规范。不了解程序语言的规范，是写不出正确程序的。

程序流程图

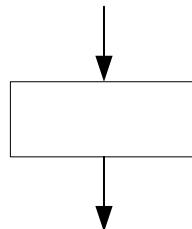
无论是代码程序还是图形化程序，当程序复杂到一定程度之后，都会变得难以读懂。工程上会利用各种图形来表达程序的结构和运行顺序，其中程序流程图是最简单最流行的一种。

程序流程图中最主要的三种符号：

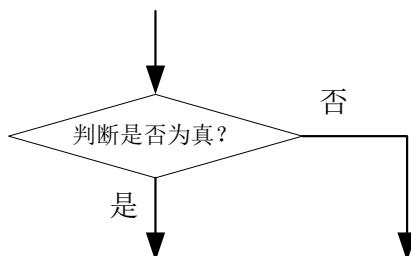


路径表示程序之间连接与流转关系，路径互相之间不能交叉。

执行框代表程序中的一个处理或者步骤，它只能接一条流入路径和一条流出路径，如：



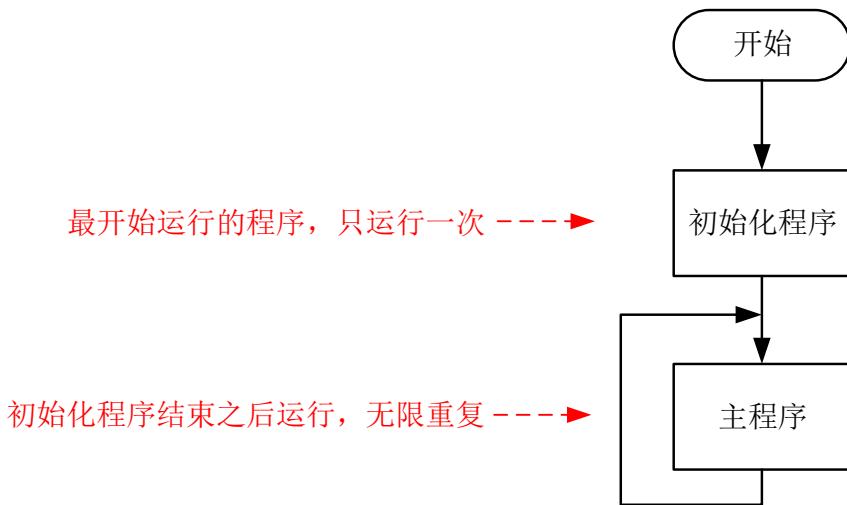
判断框是对一个条件进行判断抉择，它只能有一条流入路径，如果为“真”走一条流出路径，如果为“假”则走另一条流出路径，不可能同时往两条流出路径走，如：



Python 开源硬件的程序框架

Python 是目前全球主流编程语言之一, Python 代码简单可读性强易于学习等特点使 Python 深受编程学习者的欢迎, Python 跨平台开源且非常灵活的特点使得 Python 系列的开源硬件应用非常广泛, PythonBoard 系列的开源硬件程序框架与其他开源硬件程序框架相似。

PythonBoard 开源硬件的程序框架包括两部分: 初始化程序和主程序。程序运行的流程如下:



开始程序 PythonBoard 中开始程序预设置在编程平台内, 不需要进行命令的输入。

初始化程序 在程序中的作用一般是用于设定硬件启动时初始状态的, 位于“开始”之后, “主程序”之前, 在程序开始后率先运行一次, 之后不再运行。例如示例三交通灯程序中, 初始化程序的作用就是使接通电源时, 三个 LED 都处于熄灭状态。

主程序 是一直重复运行的, 位于“重复执行”指令之中。重复执行指令在程序中有且只允许有一个, 下图为 PythonBoard “重复执行指令”。



第十一节 数字量输入与条件判断指令

数字量传感器

数字量是只有“1”和“0”两种状态的量，比如：“是”与“否”、“开”与“关”、“真”与“假”，这些都是非此即彼的，都是数字量，也被称为开关量，只不过计算机系统只能处理数字信息，所以用“1”和“0”来指代。

认识数字量传感器

按钮开关：感知外界时候有否物体按压它。



磁感应开关：磁感应开关的敏感元件是干簧管，通常由两个软磁性材料做成的、无磁时断开的金属簧片触点。用于检测附近是否有磁性物体靠近。



震动开关：没有振动时，震动轴呈静止状态，导针 A 与导针 B 两端则为接通状态，当有震动时，震动轴会运动，导针 A 与导针 B 之间会有瞬间的断开，实现震动触发的作用。



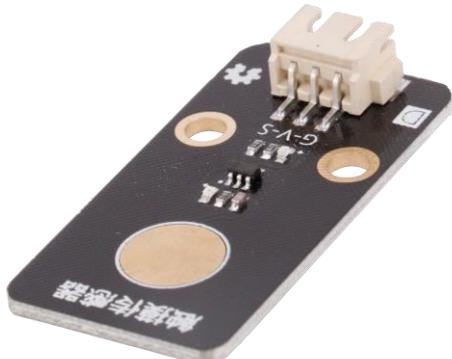
倾斜开关：垂直悬挂的倾斜开关探头在受到外力作用且偏离垂直位置 17 度以上时，倾斜开关内部的金属球触点动作，常闭触点断开。当外力撤消后，倾斜开关回复到垂直状态，金属球触点复又闭合。倾斜开关可以用于检测某个确定角度的状态。



人体红外感应开关：人体辐射的红外线中心波长为 9~10--μm，而探测元件的波长灵敏度在 0.2~20--μm 范围内几乎稳定不变。在传感器顶端开设了一个装有滤光镜片的窗口，这个滤光片可通过光的波长范围为 7~10--μm，正好适合于人体红外辐射的探测，而对其它波长的红外线由滤光片予以吸收，这样便形成了一种专门用作探测人体辐射的红外线传感器。



触摸开关：这是一个基于电容感应的触摸开关模块。人体或金属在传感器金属面上的直接接触碰会被感应到。除了与金属面的直接触摸，隔着一定厚度的塑料、玻璃等材料的接触也可以被感应到，感应灵敏度随接触面的大小和覆盖材料的厚度有关。

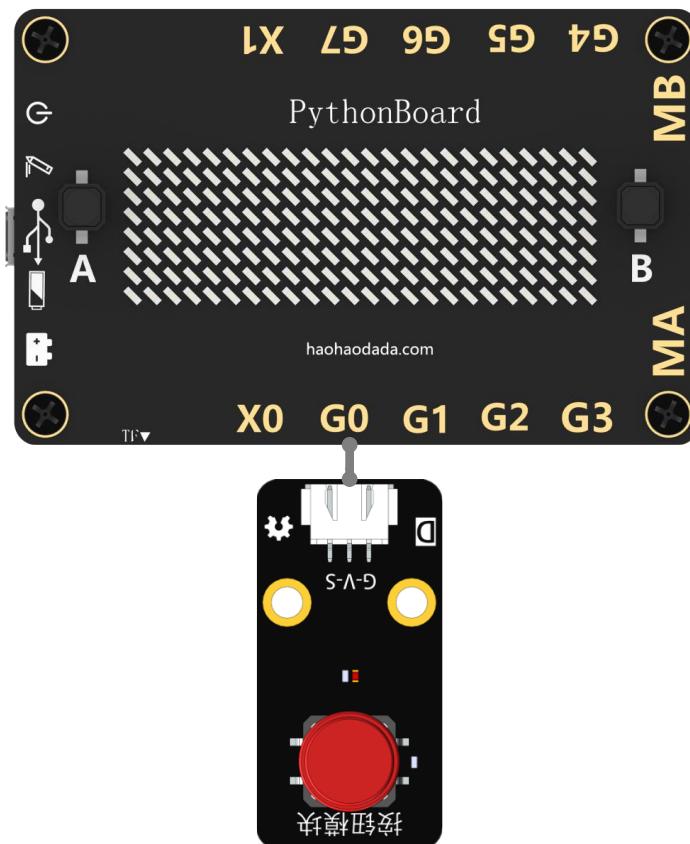


示例 11-1：用点阵屏显示开关量传感器的状态，以按钮开关为例。

元器件列表：

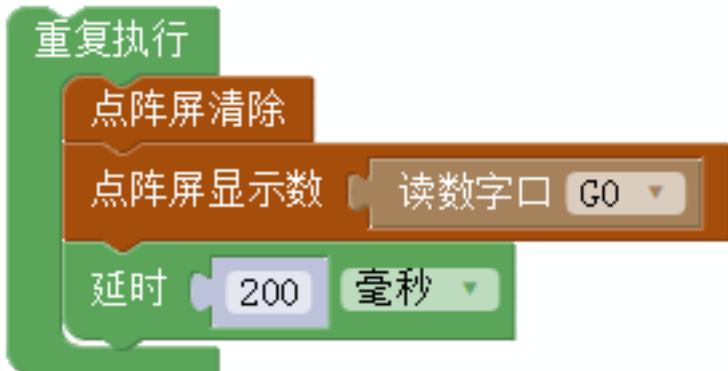
1. PythonBoard 主控板 ×1
2. 按钮开关模块 ×1
3. 3Pin 2510 连接线 ×1

电路连接：



程序编写：

```
1 Screen1=Screen()
2
3 G0=Port('G0')
4
5 while 1:
6     Screen1.clearScreen()
7     Screen1.printText(str((G0.digitalRead())))
8     delay(200)
```



```
1 Screen1=Screen()
2
3 G0=Port('G0')
4
5
6 while 1:
7     Screen1.clearScreen()
8     Screen1.printText(str((G0.digitalRead())))
9     delay(200)
```

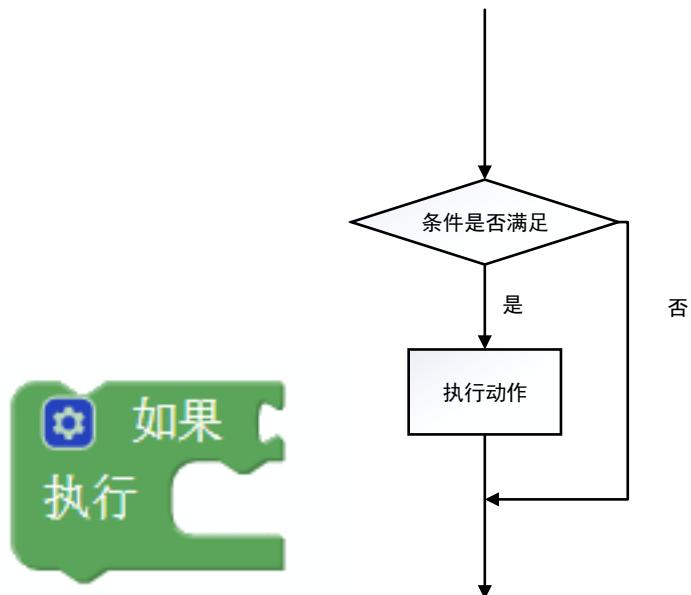
上传程序后，可以看到当按钮按下时，数码管显示 1，当按钮松开时，数码管显示 0。大家可以把按钮开关换成其他开关量传感器模块，试试看。

条件判断指令

数字量传感器的使用，通常都要配合条件判断指令来使用，即需要判断数字量传感器输入的数值是“1”还是“0”：如果是“1”，则认为是“真”；如果是“0”，则认为是“假”。PythonBoard 中条件判断命令可以根据具体问题需要进行设置，如图所示：

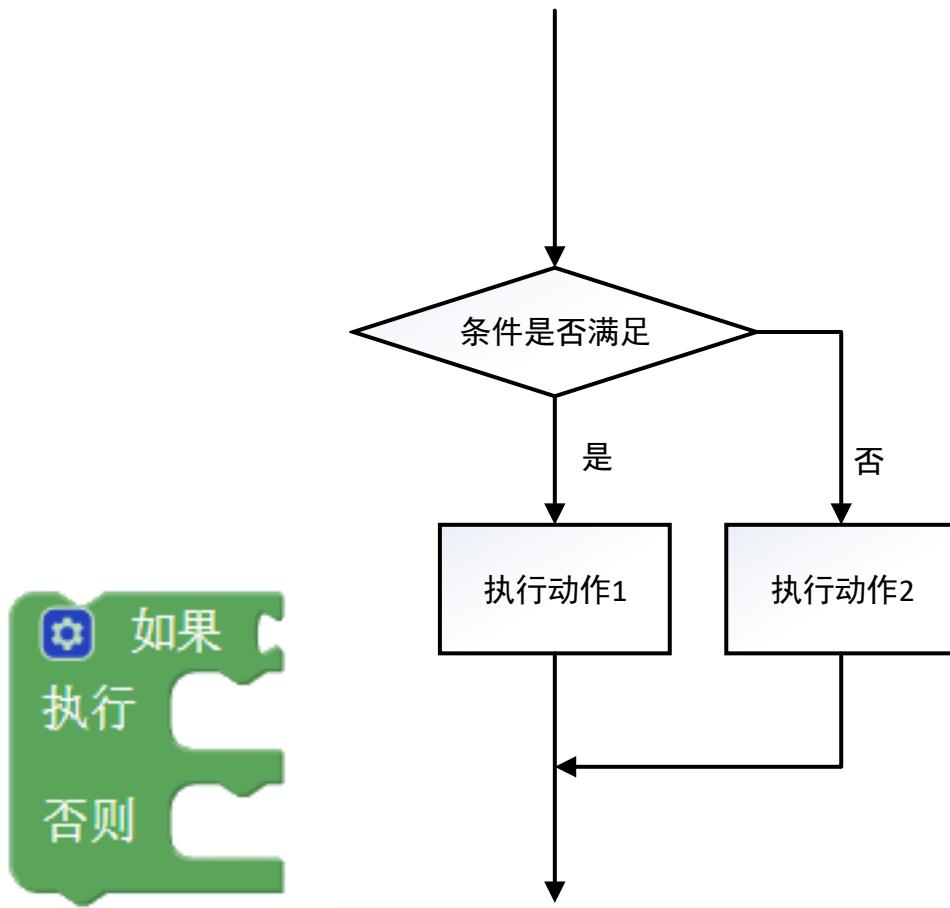


认识新指令——如果指令



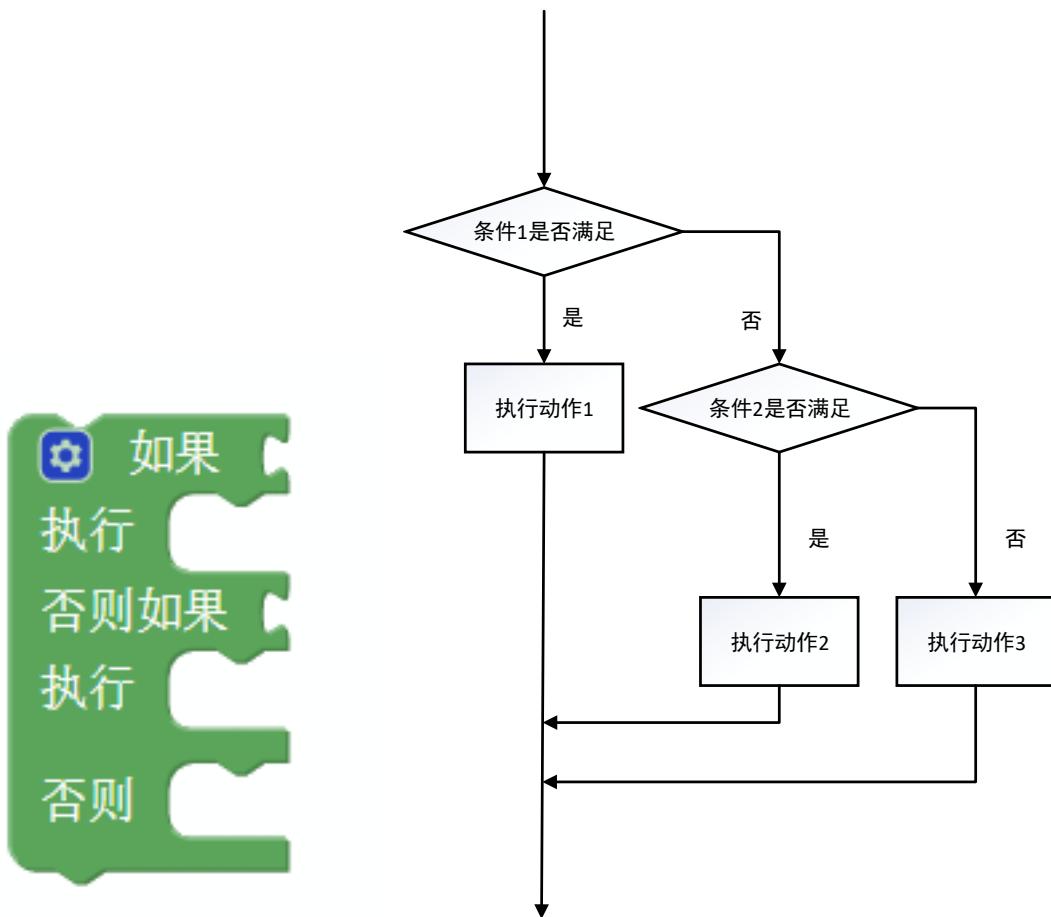
功能说明：如果条件成立，则执行动作，否则不执行动作。

认识新指令——如果-那么-否则指令



功能说明：如果条件成立，则执行动作 1 不执行动作 2，否则不执行动作 1 执行动作 2。

认识新指令——如果-否则如果-否则指令



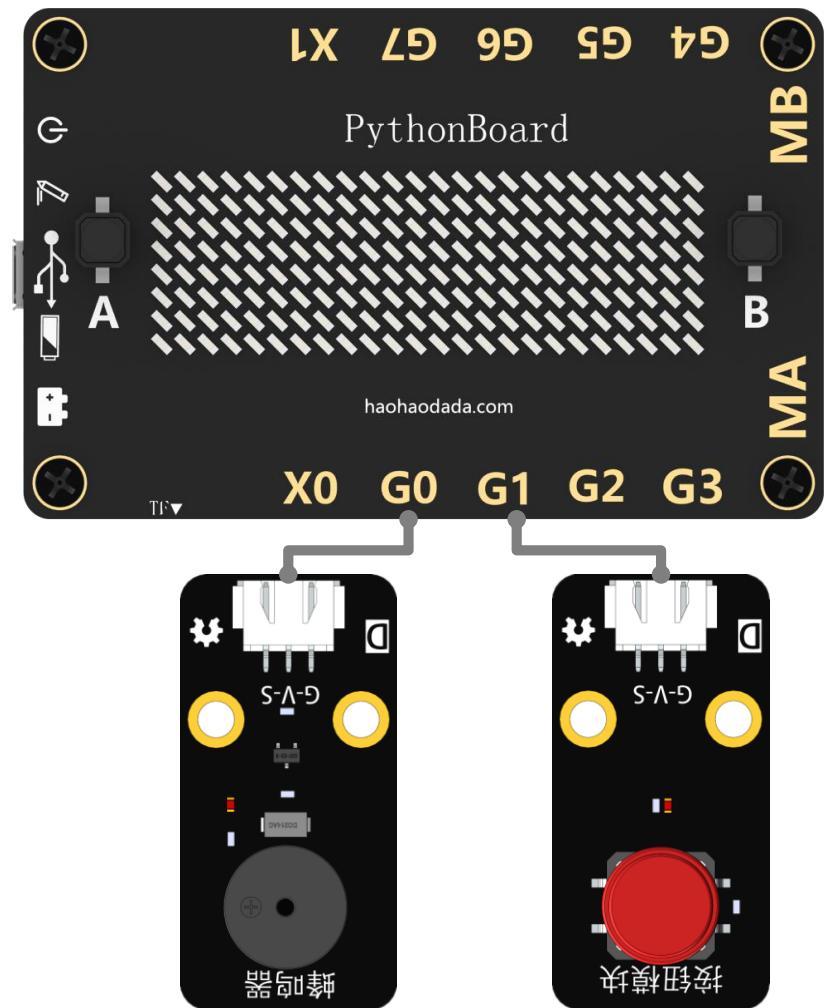
功能说明：如果条件 1 成立，则执行动作 1，否则判断条件 2 是否成立，如果成立执行动作 2，否则执行动作 3。

示例 11-2：门铃

元器件列表：

1. PythonBoard 主控板 ×1
2. 按键模块 ×1
3. 蜂鸣器模块 ×1
4. 3Pin 2510 连接线 ×2

电路连接：



程序编写:



```

1 G1=Port('G1')
2
3 G0=Port('G0')
4
5
6 while 1:
7     if G1.digitalRead() == 1:
8         G0.analogWrite(50)
9     if G1.digitalRead() == 0:
10        G0.analogWrite(0)

```



```

1 G1=Port('G1')
2
3 G0=Port('G0')
4
5
6 while 1:
7     if G1.digitalRead() == 1:
8         G0.analogWrite(50)
9     else:
10        G0.analogWrite(0)

```

条件判断指令是硬件编程中最重要的指令。能否灵活巧妙的使用它，可以表现出一个程序员的逻辑思维能力。

后续的课程会不断的使用到条件判断指令，会有更多精妙的应用！

第十二节 变量

概念：

变量可以保存程序运行时用户输入的数据和特定的运算结果。

在程序中使用变量几个主要作用是：

- (1) 提升程序可读性；
- (2) 方便批量修改程序数据；
- (3) 存放程序中间运算结果。

变量的运用是编程最重要的能力之一，能否灵活恰当的使用变量能直接体现对编程理解的深度。

创建变量：



新建完成后，所有已建的变量，都会出现在“变量”脚本类中。



注意，变量名不能含有中文！

认识新指令——赋值指令

用于将用户要输入的数据或中间运算结果写入到变量中。



认识新指令——增值指令

用于实现变量值的变化，每运行一次，变量值增加一定数值。



认识新指令——变量调用指令

当需要处理变量时使用。



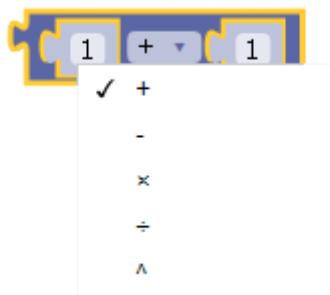
变量的作用要在运算、逻辑判断、循环、函数等构成复杂程序逻辑的场合才能真正的体现，

所以变量的相关案例和解释将在后续的章节中详细说明。

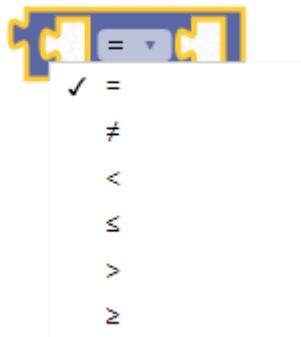
第十三节 数学与逻辑运算

在程序编写中经常需要对变量或数值进行换算或比较，这时可以调用数学与逻辑运算类指令。

认识数学运算指令：



认识逻辑运算指令：



比较指令的运行结果为“真”或“假”，即正确的为真，错误的为假。例如：

比较指令常配合“如果...那么...”指令使用。

认识布尔运算指令：



布尔运算指令是对真值（真或假）进行操作的指令，达到多个条件的协同判断的目的。

布尔运算真值表		
 真	 假	 假
 真	 真	 假
 假	 真	

示例 13-1：华氏温度计

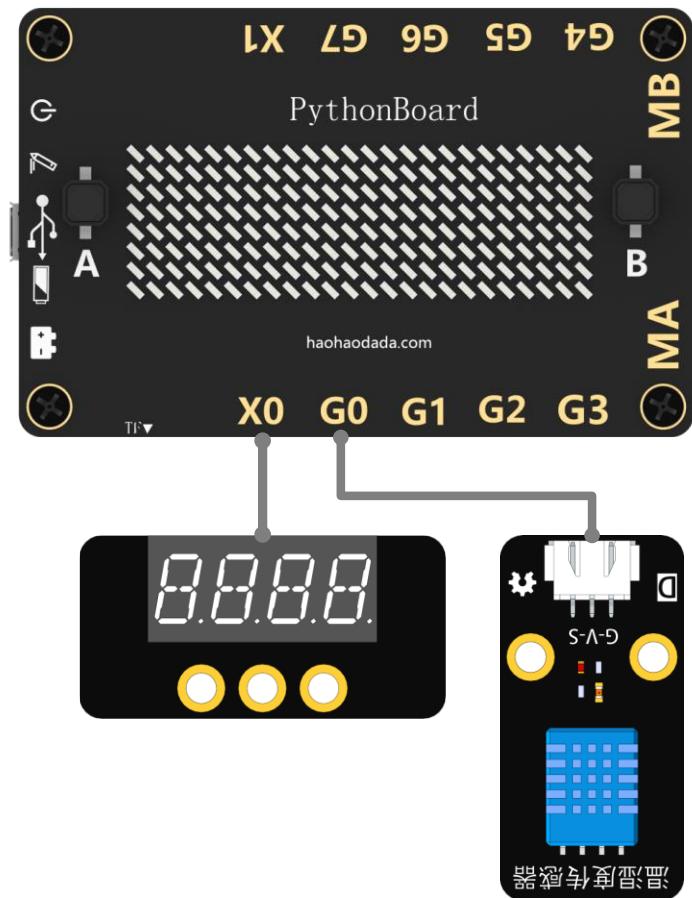
在日常生活中，常见的温度单位又两种：摄氏度和华氏度，前者是我国使用的温度单位（°C），后者是美国使用的温度单位（°F），两者的关系是：华氏度=摄氏度×1.8+32

通过程序实现：数码管显示摄氏度和华氏度，并用按键做为输入实现切换。

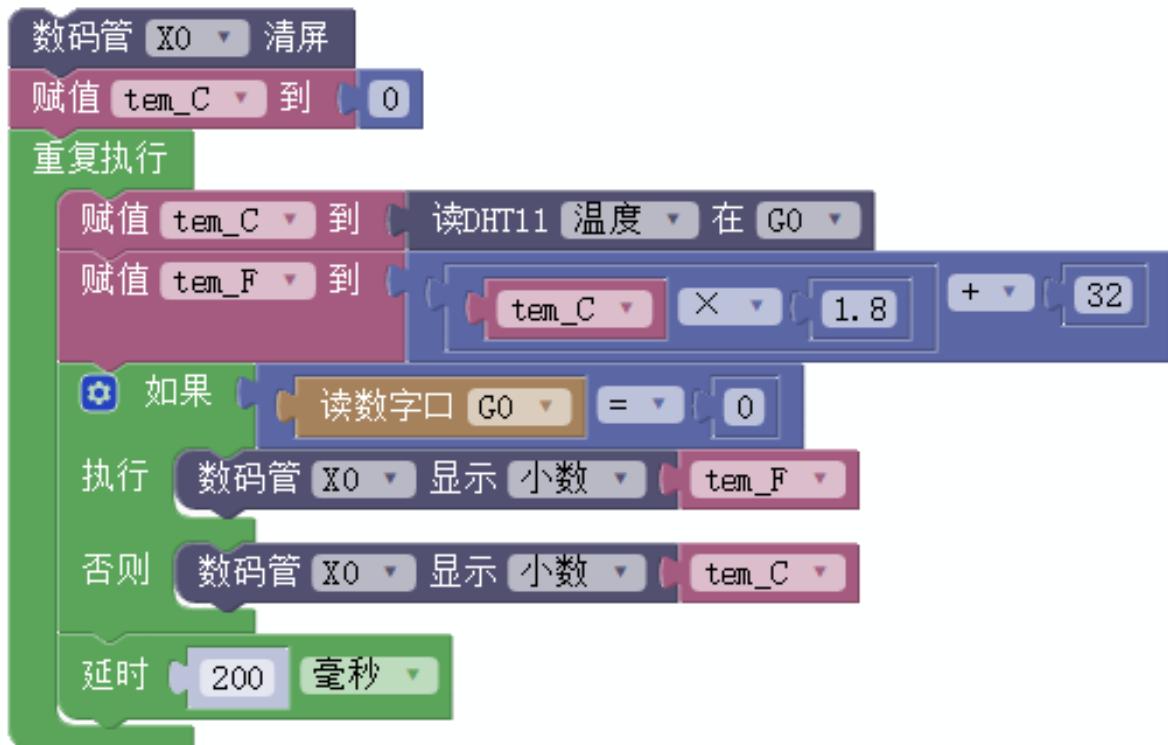
元器件列表：

1. PythonBoard 主控板 × 1
2. 温湿度传感器模块 × 1
3. 数码管模块 × 1
4. 3Pin 2510 连接线 × 2
5. 4Pin 2510 连接线 × 1

电路连接：



程序编写:



```

1 display_X0=DigitDisplay("X0")
2
3 DHT11_0=DHT('G0','DHT11')
4
5 G0=Port('G0')
6
7
8 display_X0.clearScreen()
9 tem_C = 0
10 while 1:
11     tem_C = DHT11_0.readTemperature()
12     tem_F = tem_C * 1.8 + 32
13     if G0.digitalRead() == 0:
14         display_X0.displayFloat(tem_F)
15     else:
16         display_X0.displayFloat(tem_C)
17     delay(200)

```

其中，变量 tem_C 和 tem_F 分别指代摄氏度和华氏度。这里变量的作用是提升程序的可

读性和存放中间计算结果。

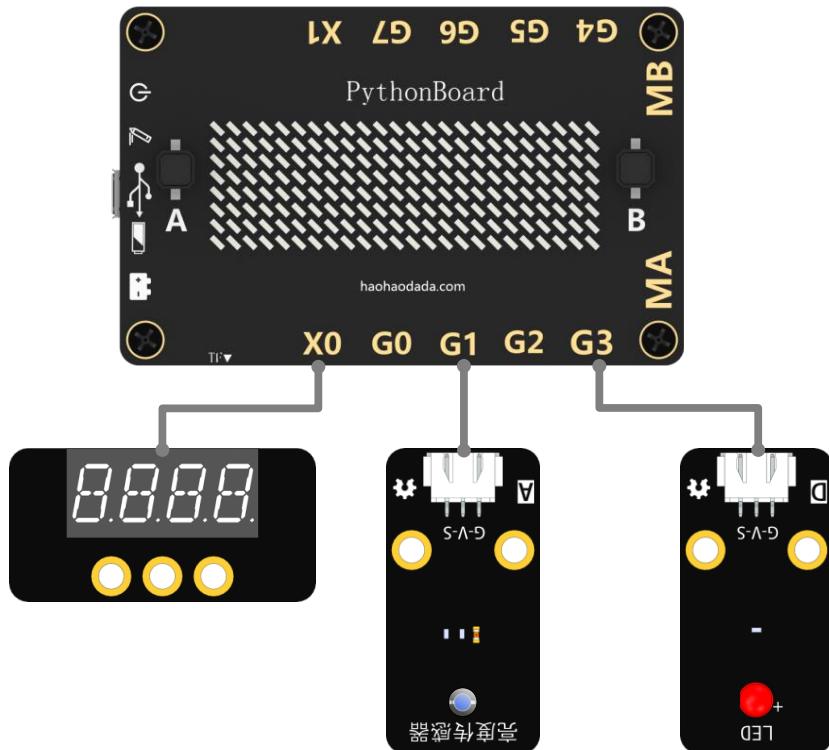
示例 13-2：光控灯（光敏开关版）

白天（光照强度高），LED 自动熄灭；夜晚（光照强度低），LED 亮起，LED 只需开关两种状态。

元器件列表：

1. PythonBoard 主控板 ×1
2. LED 模块 ×1
3. 光敏传感器模块 ×1
4. 数码管模块（计数） ×1
5. 3Pin 2510 连接线 ×1
6. 3Pin 2510 连接线 ×1
7. 4Pin 2510 连接线 ×1

电路连接：



程序编写：

第 1 步

测量夜晚需要开灯时的光照强度，观察数码管的显示值并记录下来，这个值就是用于比较的阈值。

阈值（Critical Value）是指两个相邻定义域的边界。比如常说的白天和黑夜的边界，春天和夏天的边界，白色和灰色的边界。有的阈值是约定俗成的，如春天和夏天边界是立夏这一天；有的阈值则是可以自定义的，如冷和热，亮和暗，每个人的标准都不一样。

模拟量输入与阈值进行比较大小，即可将模拟量输入传感器视为一个开关。

```
1 display_X0=DigitDisplay("X0")
2
3 G1=Port('G1')
4
5
6 while 1:
7     display_X0.clearScreen()
8     display_X0.displayFloat((G1.analogRead()))
9     delay(200)
```

第 2 步

根据的第 1 步测量到的阈值，本例中使用的阈值为 50，编写光控开关的程序：

```
赋值 [CV] 到 [50]
重复执行
  如果 [读模拟口 [G1] > [CV]]
    执行 [数字输出 [G3] 赋值为 [低]]
  否则 [数字输出 [G3] 赋值为 [高]]
```

```

1 G1=Port('G1')
2
3 G3=Port('G3')
4
5
6 CV = 50
7 while 1:
8     if G1.analogRead() > CV:
9         G3.digitalWrite(0)
10    else:
11        G3.digitalWrite(1)

```

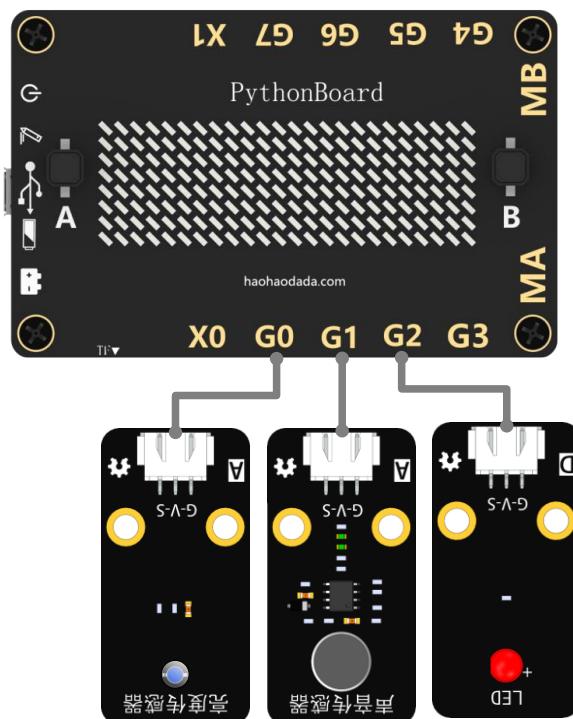
示例 13-3：夜间声控灯

只有在夜间才能通过声音点亮的灯

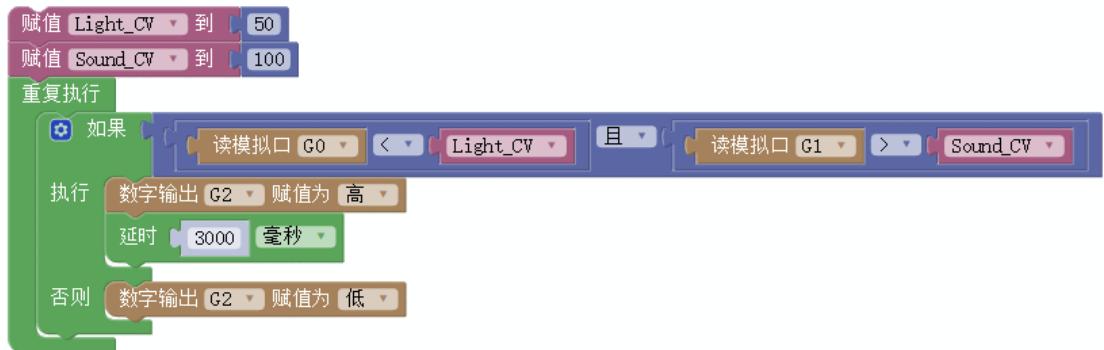
元器件列表：

1. PythonBoard 主控板 ×1
2. 光敏传感器模块 ×1
3. 声音传感器模块 ×1
4. LED 模块 ×1
5. 3Pin 2510 连接线 ×3

电路连接：



程序编写：



```

1 G0=Port('G0')
2
3 G1=Port('G1')
4
5 G2=Port('G2')
6
7
8 Light_CV = 50
9 Sound_CV = 100
10 while 1:
11     if G0.analogRead() < Light_CV and G1.analogRead() > Sound_CV:
12         G2.digitalWrite(1)
13         delay(3000)
14     else:
15         G2.digitalWrite(0)

```

第十四节 随机数

在编写程序时，尤其是设计游戏和模拟实验时，你有可能需要生成随机数让程序富有变化。

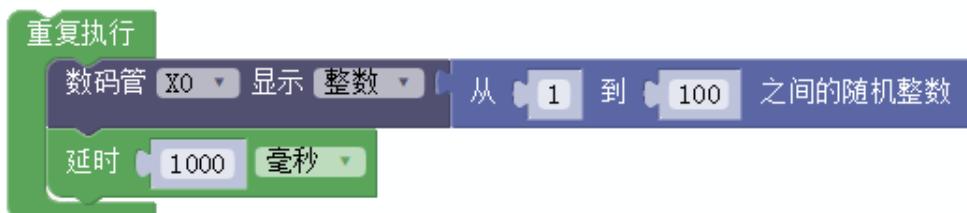
为此，PythonBoard 专门提供了在…到…间随机选一个数的指令。

认识随机数指令

用于随机从某数字范围中抽取一个数（整数）。



指令中的两个参数指定随机数的取值范围。随机数指令每运行一次，产生一个随机数，产生的这个数即为该指令的运行结果。试运行以下程序并分析：



```

1 display_X0=DigitDisplay("X0")
2
3 random = Random()
4
5
6 while 1:
7     display_X0.displayNum((random.read(1, 100)))
8     delay(1000)

```

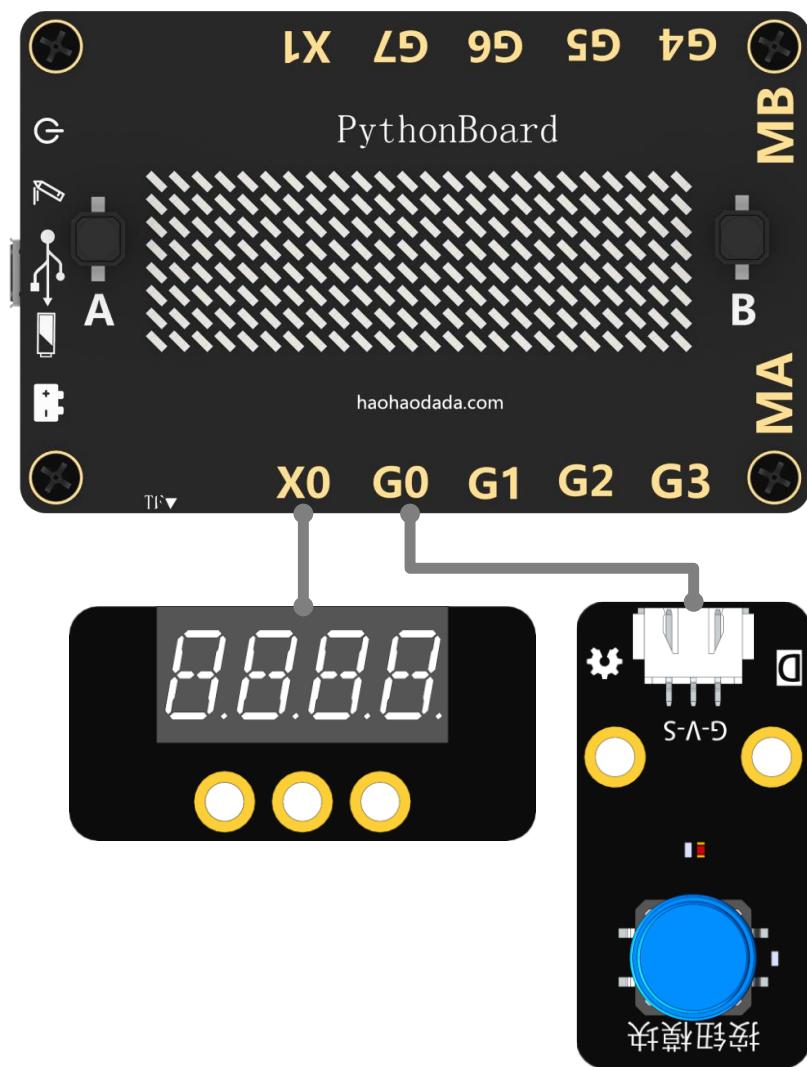
示例 14-1：摇号机（基本版）

从学号尾号 01 到 50 之间抽取一个号码，多次抽取，号码可能出现重复。

元器件列表：

1. PythonBoard 主控板 × 1
2. 数码管模块 × 1
3. 按键模块 × 1
4. 3Pin 2510 连接线 × 1
5. 4Pin 2510 连接线 × 1

电路连接:



程序编写:

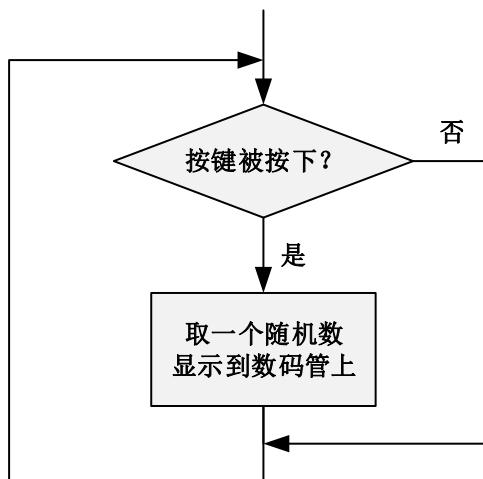


```

1 G0=Port('G0')
2
3 display_X0=DigitDisplay("X0")
4
5 random = Random()
6
7
8 while 1:
9     if G0.digitalRead() == 1:
10        display_X0.clearScreen()
11        display_X0.displayNum((random.read(1, 100)))

```

程序说明:



- (1) 当按键被按下，不断运行随机数指令，显示到数码管上，看到不断滚动的数字；
- (2) 当按键被松开，不运行随机数指令，数码管数字不更新，看到一个确定的数字。

第十五节 新建函数

当程序复杂到一定程度之后，一部分功能相对独立或会被多次使用的程序段通常会采用新建功能块的方式简化程序，提升程序的可读性。

认识新建功能块



将指令拖到脚本区按钮，修改函数名称可以创建一个新的函数：

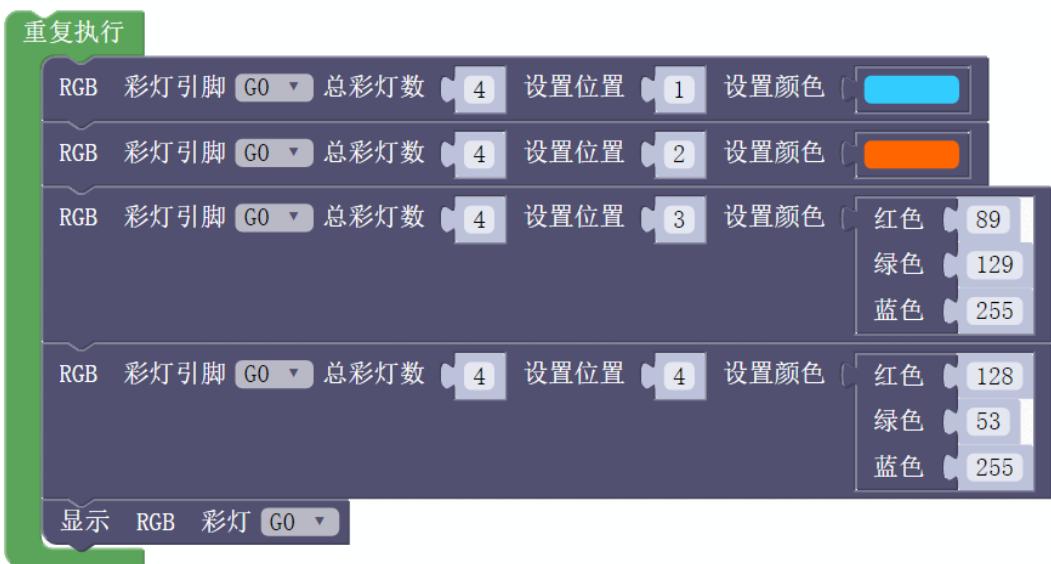


注意：功能块名称和参数都不能出现中文！

示例 15-1：RGB 交通灯（功能块版）

在示例 8-1 中，用 RGB 模块实现了交通灯的红黄绿灯变换。其中，RGB 模块的相关指令

在使用时有固定的范式：

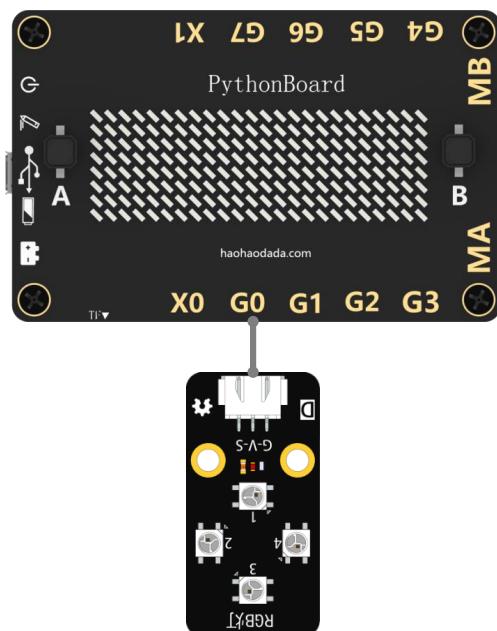


像这样的程序段，就可以采功能块的方式提升程序的可读性。

元器件列表：

1. PythonBoard 主控板 ×1
2. RGB 模块 ×1
3. 3Pin 2510 连接线 ×1

电路连接：

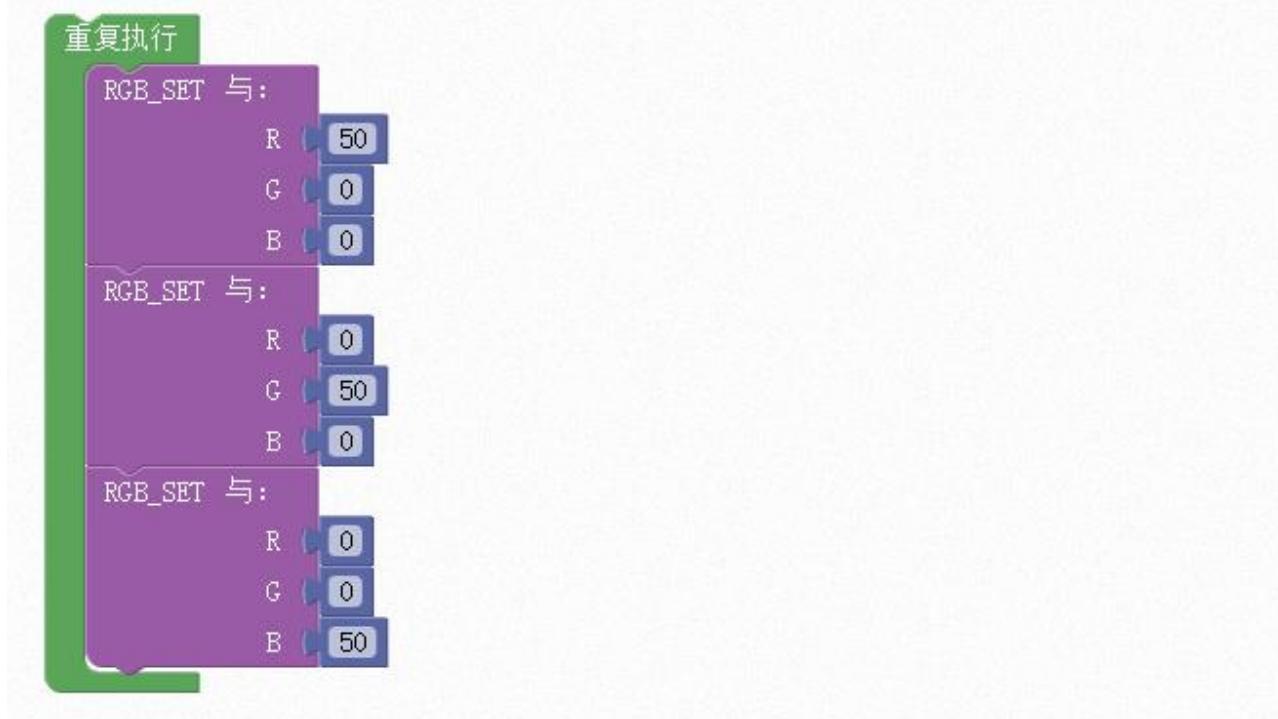


程序编写：

新建带参数的功能块：



程序为：



```
1  rgb_0=WS2812('G0',1)
2
3  """描述该功能...
4  """
5  def RGB_SET(R, G, B):
6      global x
7      rgb_0.colorSet(1,R,G,B)
8      rgb_0.show()
9      delay(1000)
10
11
12 while 1:
13     RGB_SET(50, 0, 0)
14     RGB_SET(0, 50, 0)
15     RGB_SET(0, 0, 50)
```

第三部分

硬件进阶

第十六节 红外遥控接收

红外遥控是广泛应用与生活电器的远程控制方式，需要一个红外信号发射器，一个红外信号接收器。

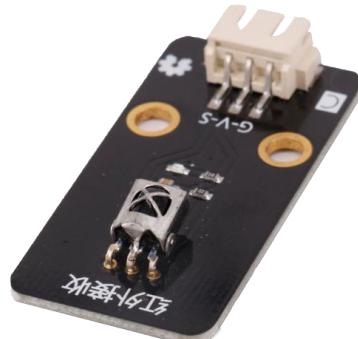
认识红外遥控器

红外信号发射器，简称“遥控器”，不同的按键按下，会发送不同信息，称为键值。



认识红外接收模块

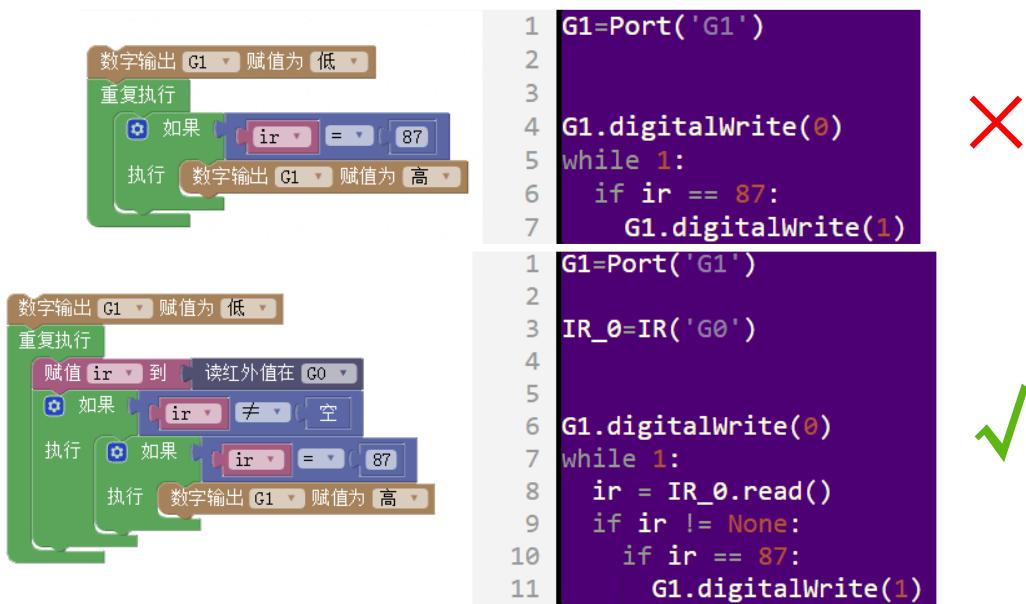
利用红外接收模块，可以制作一个红外信号接收器，用于接收遥控器发出的信号。



认识读取红外值指令

读取红外值在 G0

读取红外值指令相对于其他输入指令，不同之处在于，它不能直接放入诸如逻辑比较指令、数码管显示指令或串口输出指令中，必须先用变量将其保存起来。



认识红外按键值指令



红外按键值指令是为了方便用户使用制作的图形化指令，可以通过下拉菜单来选择对应的按键，使用方法详见示例 19-1。

注意：该指令只支持官方配套遥控器，其它遥控器的使用方法详见 19-2。

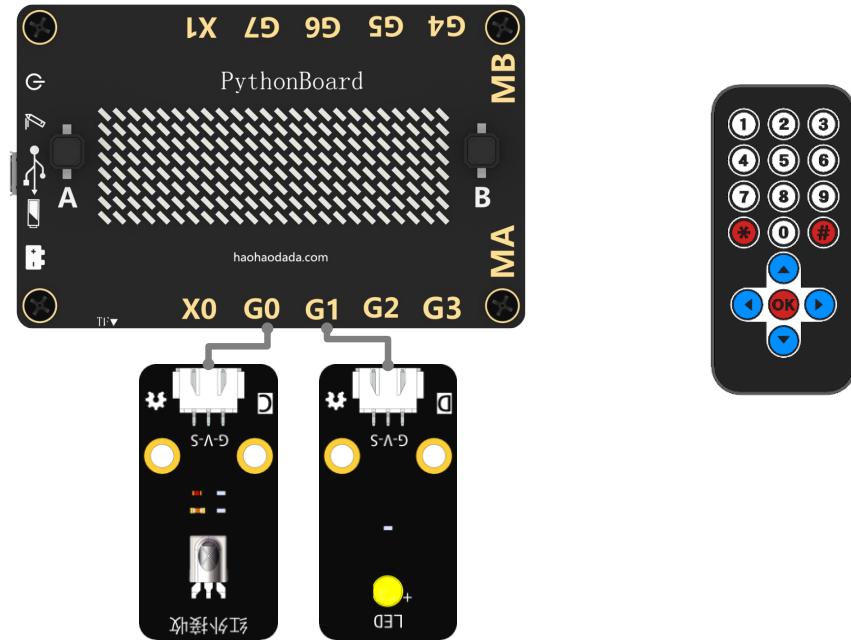
示例 16-1：红外遥控 LED

用红外遥控方式控制 LED 的亮灭

元器件列表：

1. PythonBoard 主控板 × 1
2. 红外接收模块 × 1
3. LED 模块 × 1
4. 红外遥控器 × 1
5. 3Pin 2510 连接线 × 2

电路连接：



程序编写：

```

1 G1=Port('G1')
2
3 IR_0=IR('G0')
4
5
6 G1.digitalWrite(0)
7 while 1:
8     ir = IR_0.read()
9     if ir != None:
10        if ir == 87:
11            G1.digitalWrite(1)
12        if ir == 79:
13            G1.digitalWrite(0)

```

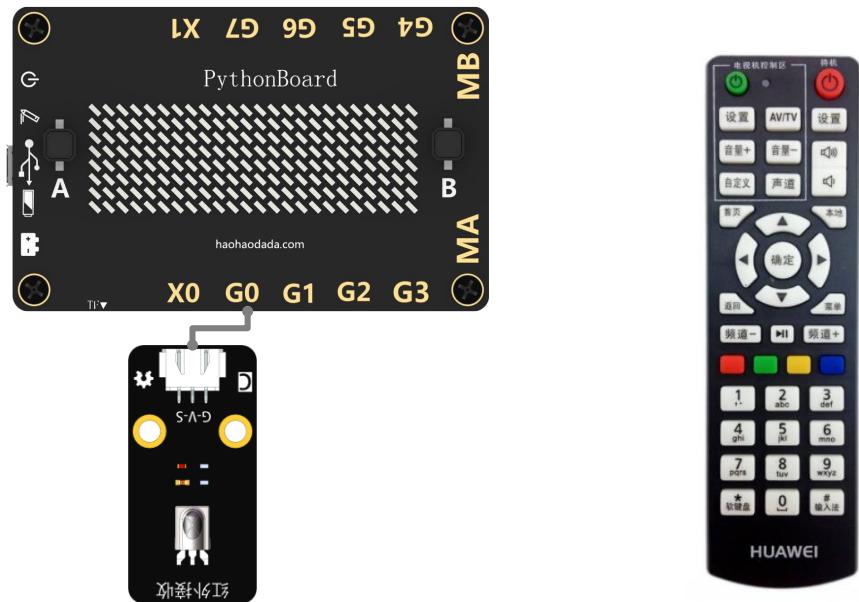
示例 16-2：用点阵屏显示红外遥控器的键值

红外接收模块除了支持官方配套遥控器外，也支持大多数生活中的遥控器，如电视遥控器、空调遥控器、投影机遥控器。

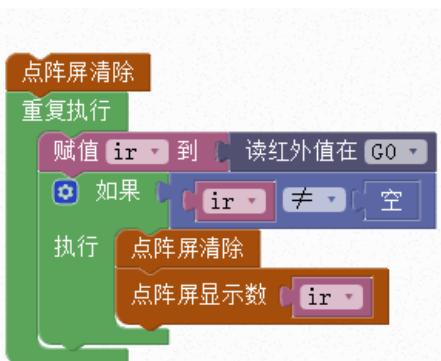
元器件列表：

6. PythonBoard 主控板 ×1
7. 红外接收模块 ×1
8. 红外遥控器 ×1
9. 3Pin 2510 连接线 ×1

电路连接：



程序编写:



```

1 Screen1=Screen()
2
3 IR_0=IR('G0')
4
5
6 Screen1.clearScreen()
7 while 1:
8     ir = IR_0.read()
9     if ir != None:
10        Screen1.clearScreen()
11        Screen1.printText(str(ir))

```

当没有操作红外遥控器时，红外接收模块返回的一直是 0；而当按下红外遥控器上的按键，红外接收模块会接收到有且只有一个值，之后就算一直按住按键不放，红外接收模块也不会继续接收到新的值，只会返回 0，所以按键值就一闪而过，数码管上只能看到数字 0。

所以用 a 不等于空的条件将按键值留下显示，其它的全部过滤掉。

通过上述方法，把遥控器的各个按键的值记录下来，如本例中用的某款电视遥控器中的“确定”键的值为 157，“返回”键的值为 295，利用这两个按键控制 LED 的亮灭，示例 19-1 的程序变为：



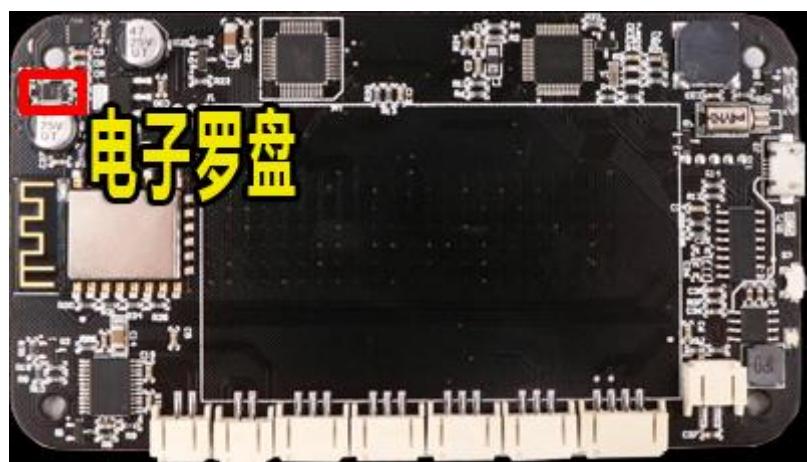
```
1 G1=Port('G1')
2
3 IR_0=IR('G0')
4
5
6 G1.digitalWrite(0)
7 while 1:
8     ir = IR_0.read()
9     if ir != None:
10         if ir == 157:
11             G1.digitalWrite(1)
12         if ir == 295:
13             G1.digitalWrite(0)
```

第十七节 电子罗盘

罗盘也叫做指南针、司南，是中国古代四大发明之一，它是利用地球磁场来指示方向的一种工具。罗盘在航海中的广泛应用，使得哥伦布到达美洲大陆、麦哲伦环球航行成功。

传统的罗盘相对都比较大。随着电子技术的发展，使用磁阻传感器可以生产出非常小巧的电子罗盘，PythonBoard 内部的电路板上就集成了一块电子罗盘芯片（如下图所示）。

认识电子罗盘



认识校准电子罗盘指令

校准板载指南针

由于地球磁场非常弱，因此电子罗盘容易受到身边各种电子产品的干扰，为了提高数据准确性，电子罗盘都需要校准。

认识校准电子罗盘指令

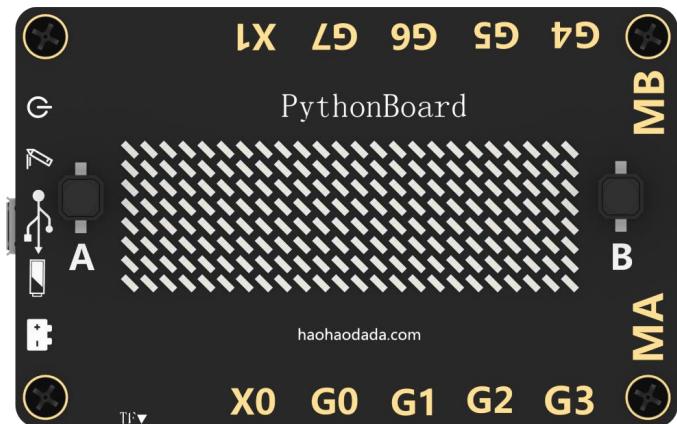
读板载指南针（°）

用于读取板载电子罗盘的角度值。取值范围是 0-360，其中 0 度表示正北方，90 度表示正东方，180 度表示正南方，270 度表示正西方。

元器件列表：

1. PythonBoard 主控板 × 1

电路连接：



程序编写：



```

1 Screen1=Screen()
2
3 G0=Port('G0')
4
5 compass=Compass()
6
7
8 Screen1.clearScreen()
9 Screen1.printText('wait')
10 G0.analogWrite(10)
11 compass.calibrate()
12 Screen1.clearScreen()
13 Screen1.printText('ok')
14 G0.analogWrite(0)
15 while 1:
16     Screen1.clearScreen()
17     Screen1.printText(str((compass.getHeading())))
18     delay(100)

```

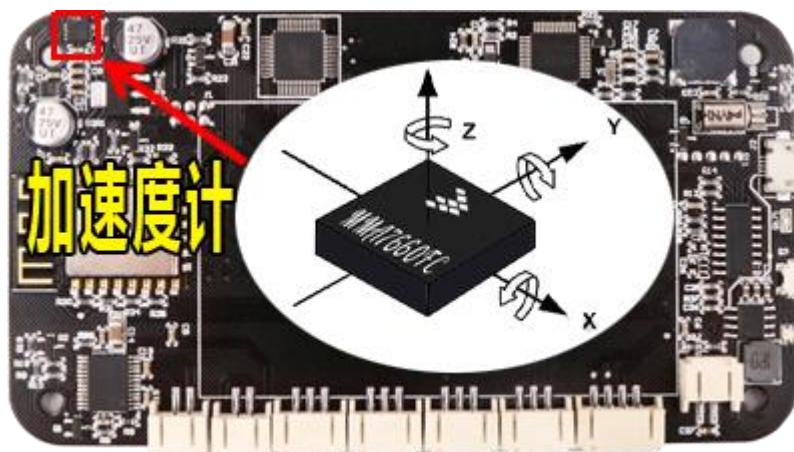
完整程序参见链接：<http://www.haohaodada.com/>。

第十八节 加速度计

加速度计就是能够检测到这种变化并转化、输出相应数值的传感器。在汽车安全系统、地震检测、游戏控制、手机计步器等很多方面都有广泛的应用。

PythonBoard 在主板上集成的加速度计型号是“MMA7660FC(如下图所示)。它的大小是 $3 \times 3 \times 0.9$ (毫米)，可以检测 X、Y、Z 三个方向所受到的加速度大小，检测精度是 ± 1.5 个重力加速度。

认识加速度计模块



认识加速度计相关指令

检测到摇晃

用于检测 PythonBoard 有没有被晃动。如果被晃动，返回值为“1”；否则，返回值为“0”。

读板载加速度(g) X

可以读取 PythonBoard 在左右(X)、垂直(Y)、前后(Z)三个方向上的运动变化数值，从而更为精确的了解 PythonBoard 的运动状态。

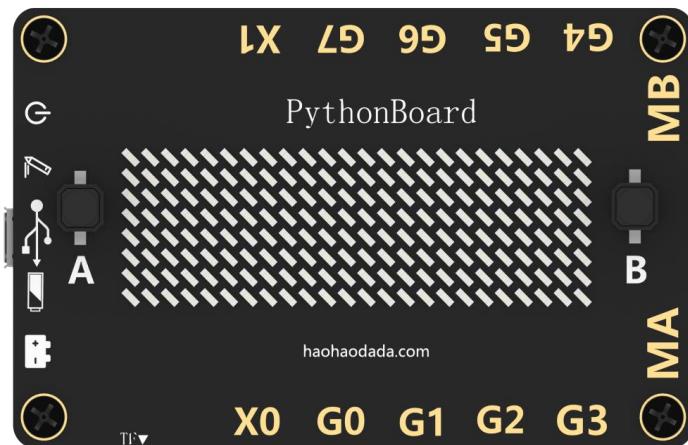
指令有一个下拉列表参数，单击可以选择“X”、“Y”、“Z”三个选项；分别返回 PythonBoard 在左右、垂直、前后这三个方向的运动变化数值。

示例 18-1：计步器制作

元器件列表：

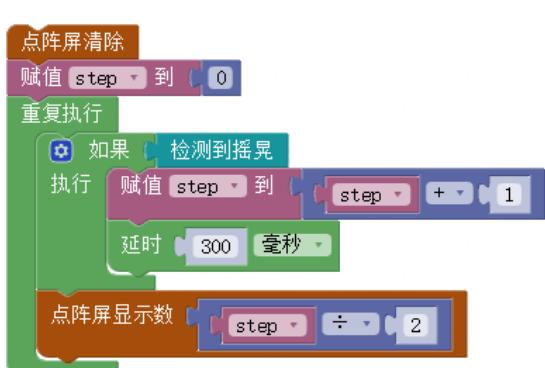
1. PythonBoard 主控板 × 1

电路连接：



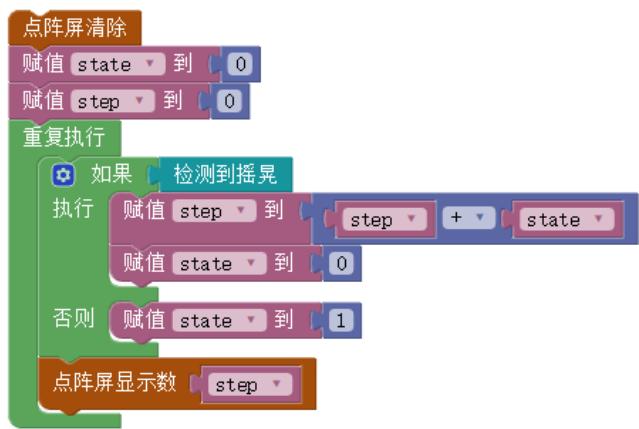
程序编写：

要在计步器上显示步数，首先需要创建变量用于累计加速度计检测到摇晃的次数，但是人走路时一般会自然地前后摆动，走一步，手臂会自然摆动两次，所以人所走的步数是加速度计检测到摇晃次数的一半，让点阵屏显示记录的步数。具体的程序可如下：



```
1 Screen1=Screen()
2
3 accel=Accel()
4
5
6 Screen1.clearScreen()
7 step = 0
8 while 1:
9     if accel.isShake():
10         step = step + 1
11         delay(300)
12     Screen1.printText(str((step / 2)))
```

以上的程序是预测每个人摆一次手的时间是 0.3 秒，但是每个人走路的速度和摆手的频率都是不一样的。如何能够提高计步器的准确性呢？此时，我们可以再增加一个变量 (state) 来记录加速度计检测到的摇晃状态。当检测到摇晃时，步数值增加 1，并将记录摇晃状态的变量 (state) 赋值为 0；当未检测到摇晃状态时（即摆手到达最高点时），将记录摇晃状态的变量 (state) 赋值为 1。具体程序代码可如下：



```
1 Screen1=Screen()
2
3 accel=Accel()
4
5
6 Screen1.clearScreen()
7 state = 0
8 step = 0
9 while 1:
10     if accel.isShake():
11         step = step + state
12         state = 0
13     else:
14         state = 1
15     Screen1.printText(str(step))
```

完整程序参见链接: <http://www.haohaodada.com/>

第十九节 MP3 音乐播放

第二十节 炫酷点阵屏

点阵屏在生活中随处可见，商业中心的外墙大屏幕、商店招牌、火车站候车信息等等。创意作品中恰当的使用点阵屏同样能极大的提升作品的表现力。

认识点阵模块

点阵模块实际上集成了很多个 LED 的模块，每个 LED 都有一个唯一的位置坐标。



认识点阵选点指令

点阵屏写入点在第 **1** 行第 **1** 列

通过该指令中输入的坐标值可以选取要点亮的点。

认识点阵显示指令

点阵屏显示点

单独运行选点指令不能直接点亮点阵模块，必须运行点阵显示指令之后才能点亮。

认识点阵清除指令

点阵屏清除

运行该指令清除点阵模块中的所有数据，同样需要配合点阵显示指令使用。

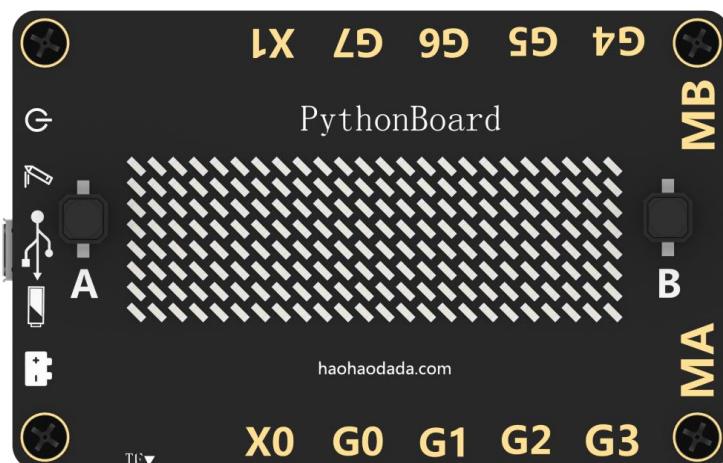
示例 20-1：按坐标点亮点阵模块

一行一行的把点阵点亮

元器件列表：

1. PythonBoard 主控板 × 1

电路连接：



程序编写：

```
1 Screen1=Screen()
2
3
4 Screen1.clearScreen()
5 x = 0
6 y = 0
7 while 1:
8     for count2 in range(8):
9         for count in range(24):
10             Screen1.setPixel(x,y)
11             Screen1.writeScreen()
12             y = y + 1
13             delay(100)
14             x = x + 1
15             y = 0
16
```

示例 20-2：音乐动感点阵屏——柱状图

利用声音传感器采集声音信号，将音乐的律动显示到点阵屏上。

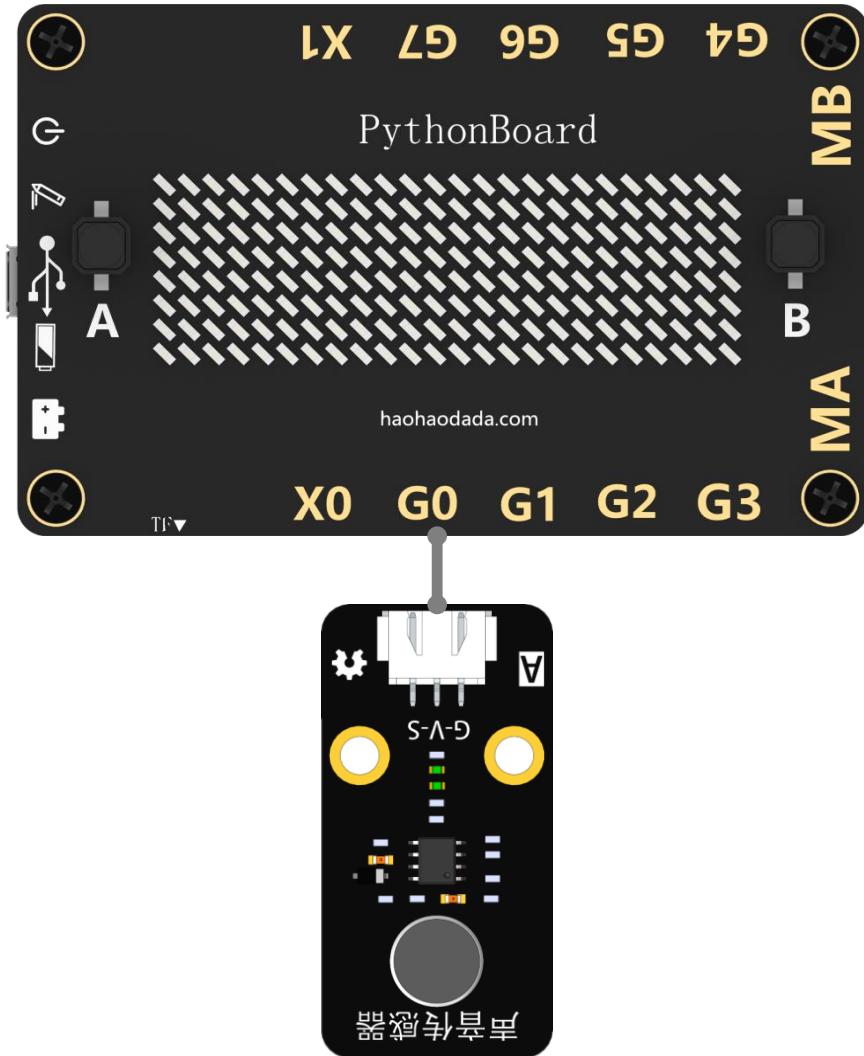
元器件列表：

1. PythonBoard 主控板 × 1

2. 声音传感器模块 × 1

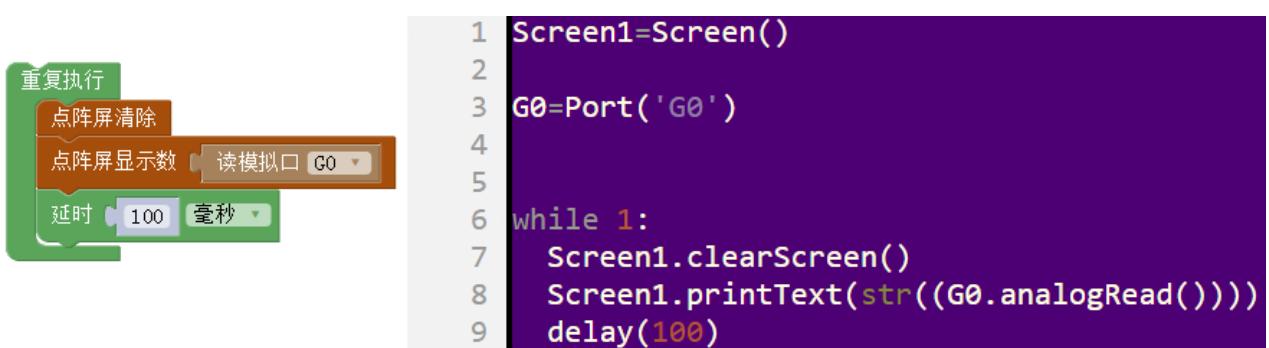
3. 3Pin 2510 连接线 × 1

电路连接:

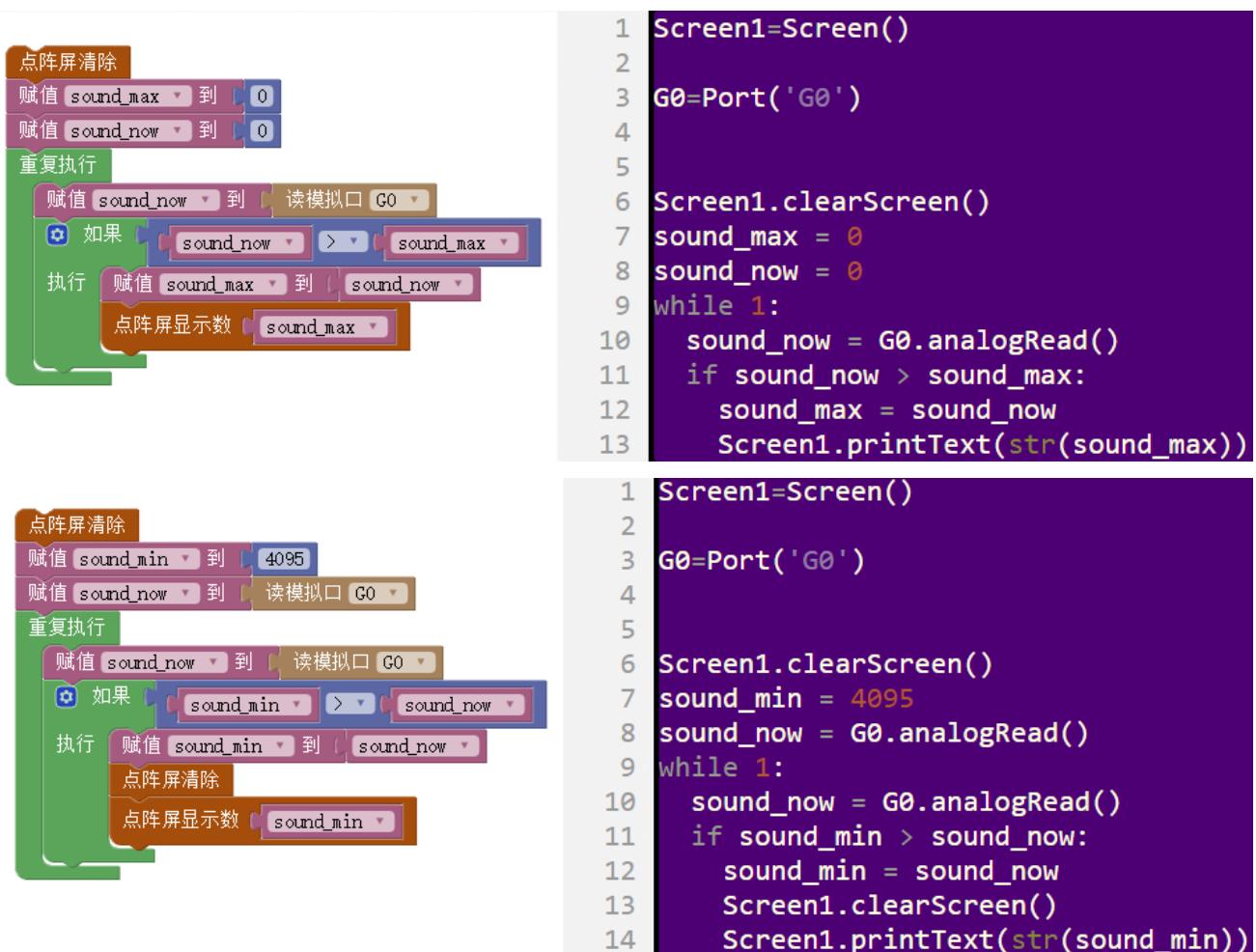


程序编写:

打开音频播放器，当然也可以把 MP3 模块集成进去用于发声。可用点阵屏来显示声音传感器的数值。



由于声音传感器数值不断根据环境声音进行变化，所以此处可以设置两个程序，分别得出声音传感器能够检测到的最大值和最小值。

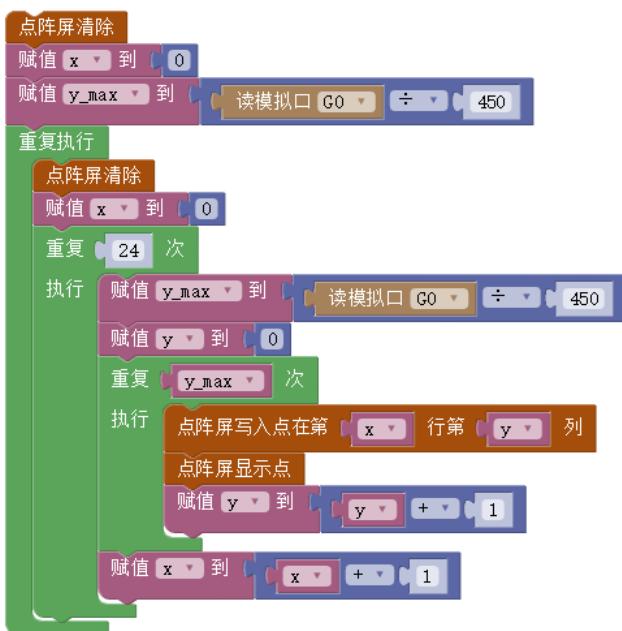


通过以上两个程序，可以得到最大值约为 3600，最小值约为 0。正好分为 8 个区间，与点阵模块 Y 坐标的 8 个点对应。如声音值=535，则对应点亮 Y=0 和 Y=1 的点；若声音值=300，则对应点亮 Y=0~8 的点。

(1) 点阵点亮的顺序是从左到右按列的点亮；

(2) 每列都从 Y 坐标为 0 开始，一直点亮到峰值为止。

所以音乐动感点阵屏的程序如下：



```
1 Screen1=Screen()
2
3 G0=Port('G0')
4
5
6 Screen1.clearScreen()
7 x = 0
8 y_max = G0.analogRead() / 450
9 while 1:
10   Screen1.clearScreen()
11   x = 0
12   for count2 in range(24):
13     y_max = G0.analogRead() / 450
14     y = 0
15     for count in range(int(y_max)):
16       Screen1.setPixel(x,y)
17       Screen1.writeScreen()
18     y = y + 1
19   x = x + 1
```

第二十一节 蓝牙远程控制

如今，人们已经习惯了手机应用带来的各种便利。做了那么多作品，大家对通过手机 APP 控制自己制作的智能硬件作品都有着不小的冲动吧！这节课，将带大家实现一个简单的手机蓝牙 APP 远程控制 LED 亮灭的应用。

认识蓝牙模块

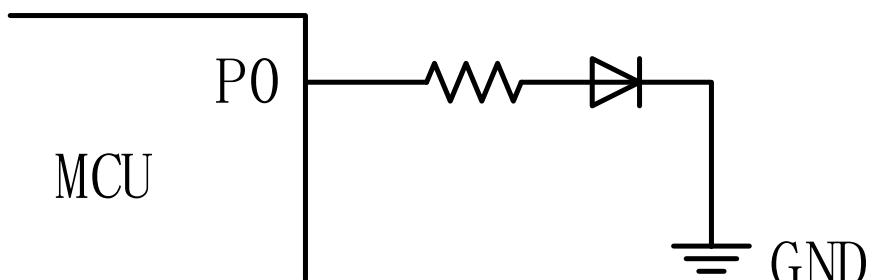


蓝牙模块的安装

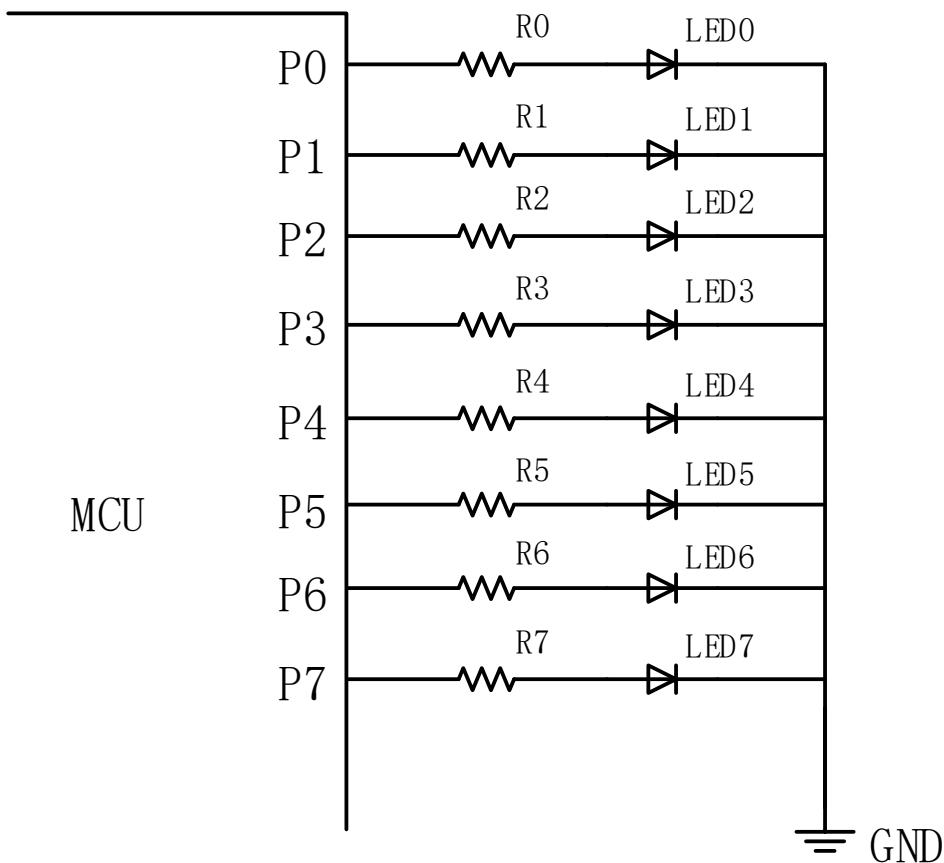
蓝牙模块的安装与 RTC 一样，是可以直接插到主控板上的。

认识蓝牙通讯的原理

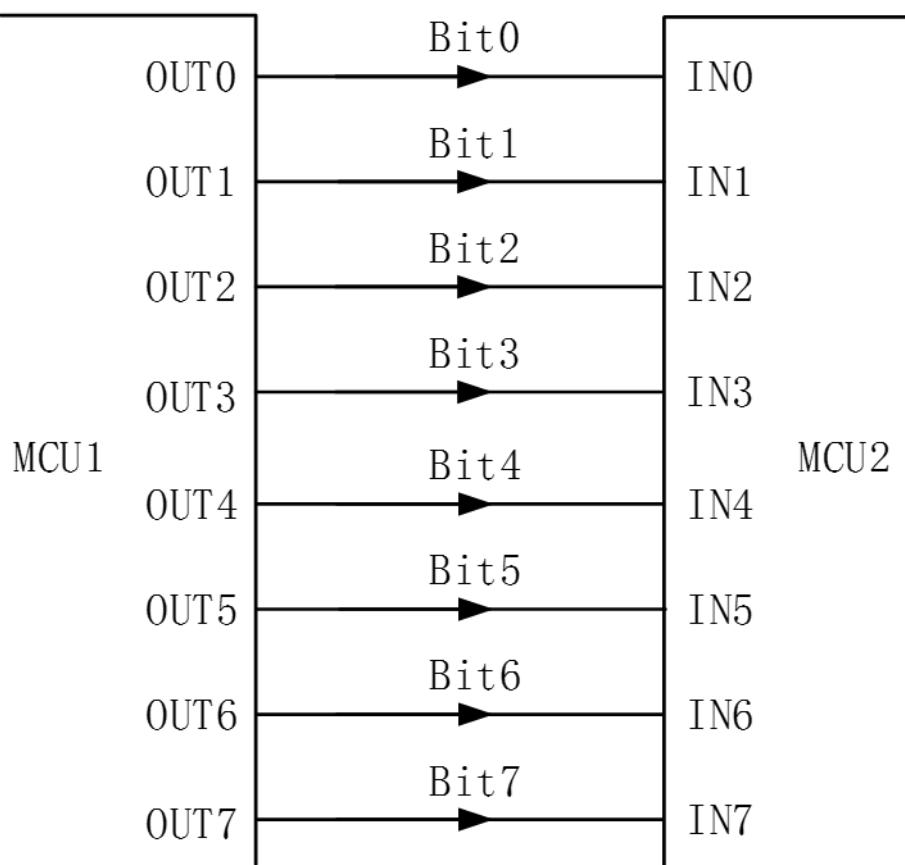
在“第十四节 串口打印”中介绍了串口的相关应用。这里对串口通讯做更详细的介绍。



上图是 MCU（微控制单元）驱动单个 LED 的电路，那么驱动多个 LED 的电路呢？玩过 Arduino 朋友都不会陌生。如下图：

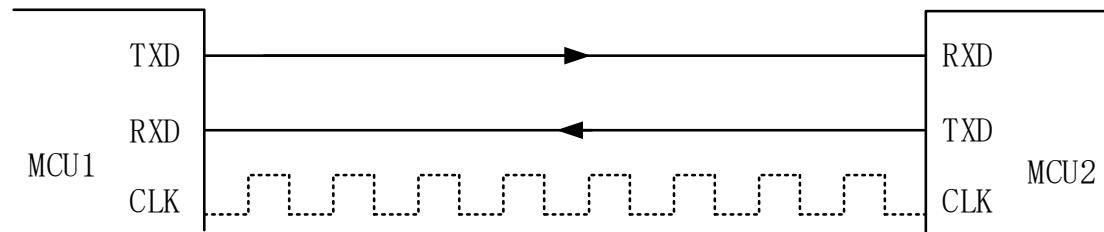


像上图这样由多个 IO 口同时传输数据的通讯方式，称为“并口通讯”。



形如上图的通讯方式均为“并口通讯”。不难想到并口通讯方式的优点是传输速度快，而缺点是线路复杂，一次要传输多少位的数据，就得连接多少根导线。

而“串口通讯”，通常只有“TXD（发送）”和“RXD（接收）”两根信号线，不论有多少数据需要传输，都通过这两根线实现，接线简单。

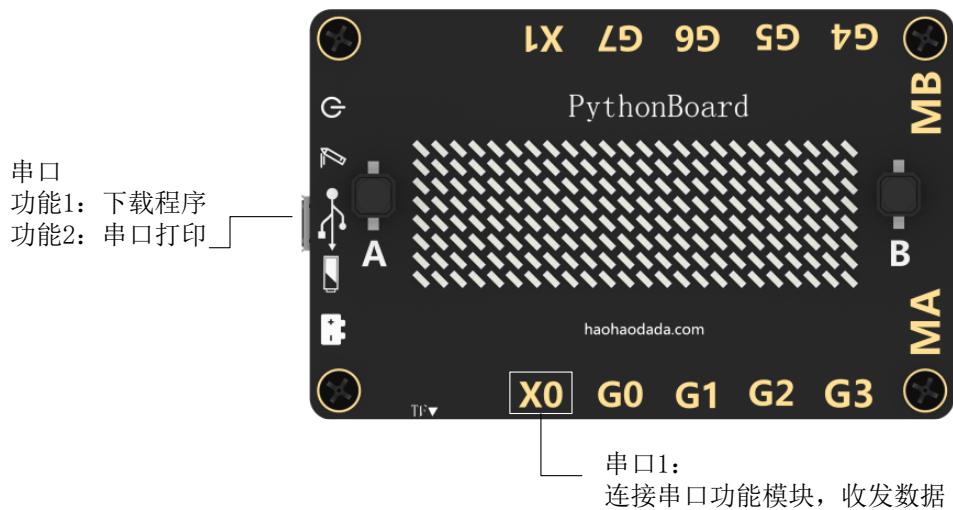


其中 CLK 不是一根导线，它是“波特率”，指通过串口发送数据的速度，即单位时间内发送数据的个数。互相传输数据的两端的波特率设置必须完全相同，才能保证通讯的正常。

“一汽车倒车，一路人很热心——“倒……倒……倒……倒不得了！”可车子一只轮胎已滑进路边水沟。车夫怒气冲冲下来，旁观者说，那人是一结巴。”

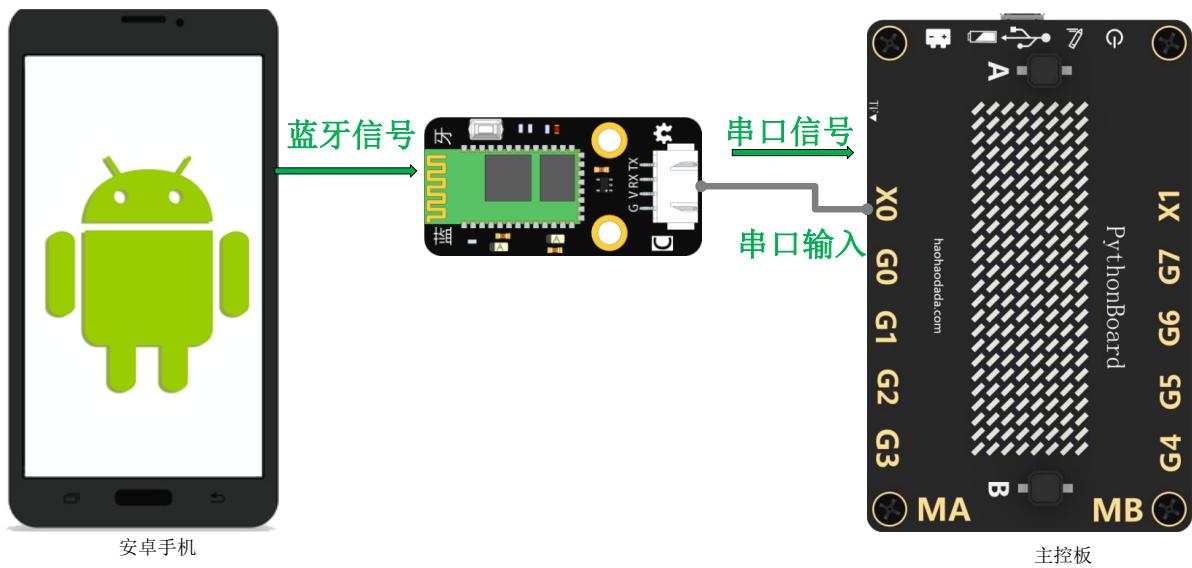
通讯两端波特率不同就会闹出像上面笑话里那样的误会。

PythonBoard 主控板上的串口有两个，分别为“X0”和“X1”，可用来连接蓝牙模块。

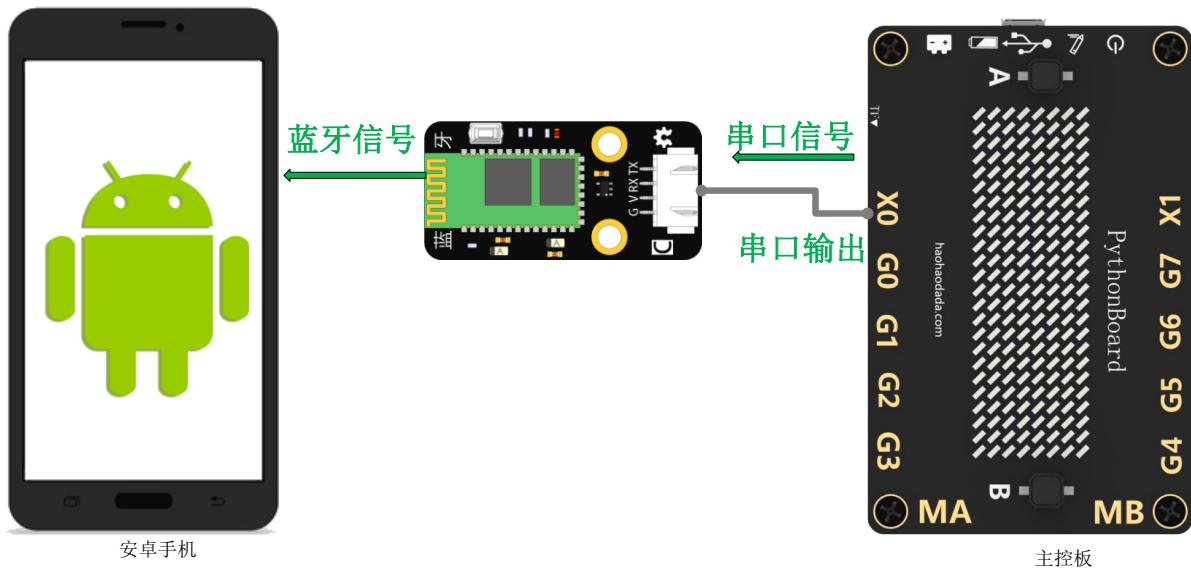


这里有的同学可能会问，为什么不叫“串口 1”和“串口 2”？是因为其命名规则沿用 Arduino 体系下的“Serial”和“Serial1”。

手机 APP、蓝牙模块、NOVA HD 主控板三者的连接和数据流向是怎样的呢？



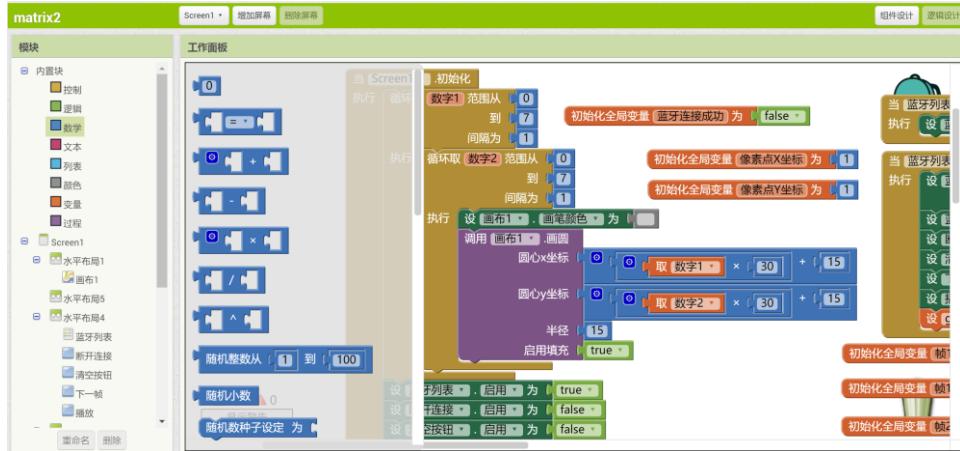
手机 APP 向 NOVA HD 主控板发送数据



NOVA HD 主控板向手机 APP 发送数据

认识 App Inventor

App Inventor 是一款谷歌公司开发的手机 APP 编程软件，与是类似于 Scratch、Mixly 的图形化积木式编程软件。



APP Inventor 也有在线编程平台，这里笔者推荐广州电教网的：<http://app.gzjkw.net>

APP Inventor 的相关资料网站推荐：<http://www.17coding.net/>。

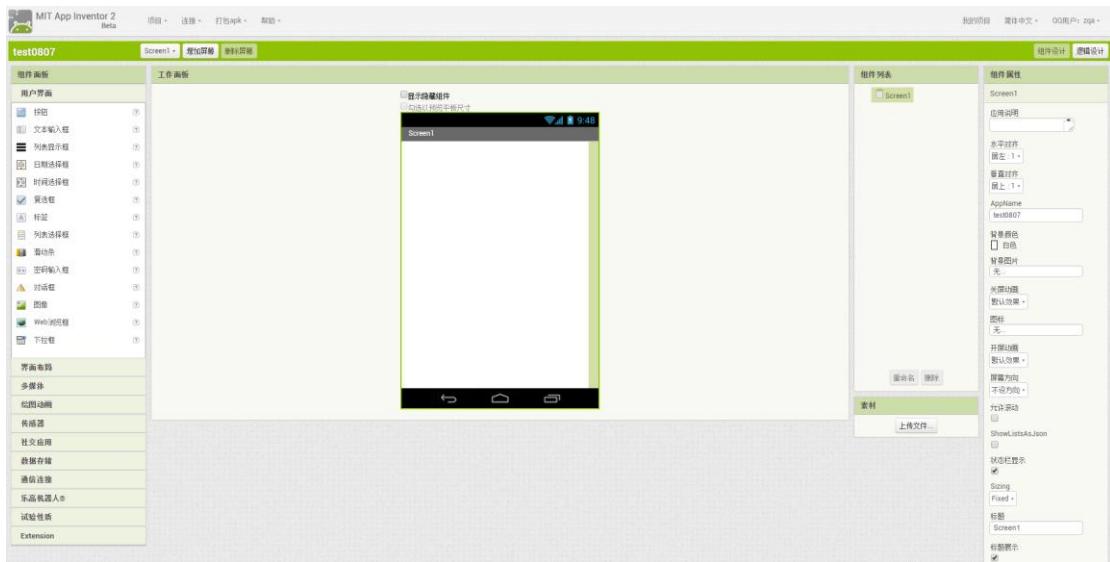
APP Inventor 因为没有官方中文版，所以各家中文版平台的翻译略有不同，查阅资料时发现命名上的冲突，最好是查找对应的英文说明。

登录 APP Inventor 编程平台



可以直接用 QQ 账号登录。

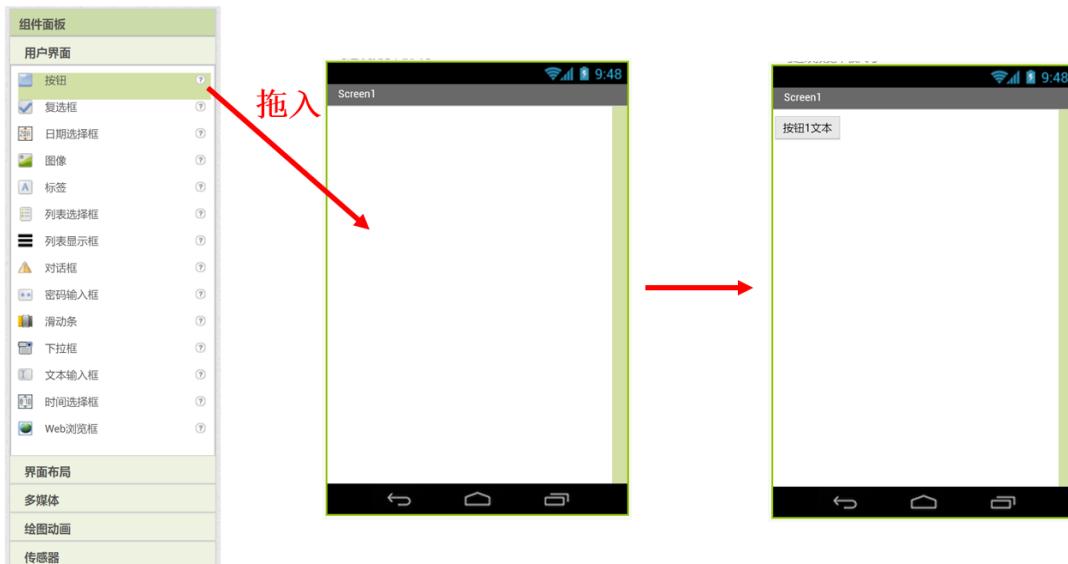
新建项目：



APP Inventor 的项目分为“组件设计”和“逻辑设计”两部分，上图是组件设计界面。

其中，“组件面板”是组件库，里面有按键、文本框、滑动条、画布、蓝牙客户端等一系列组件，可以通过鼠标拖拽的方式添加到 APP 中。

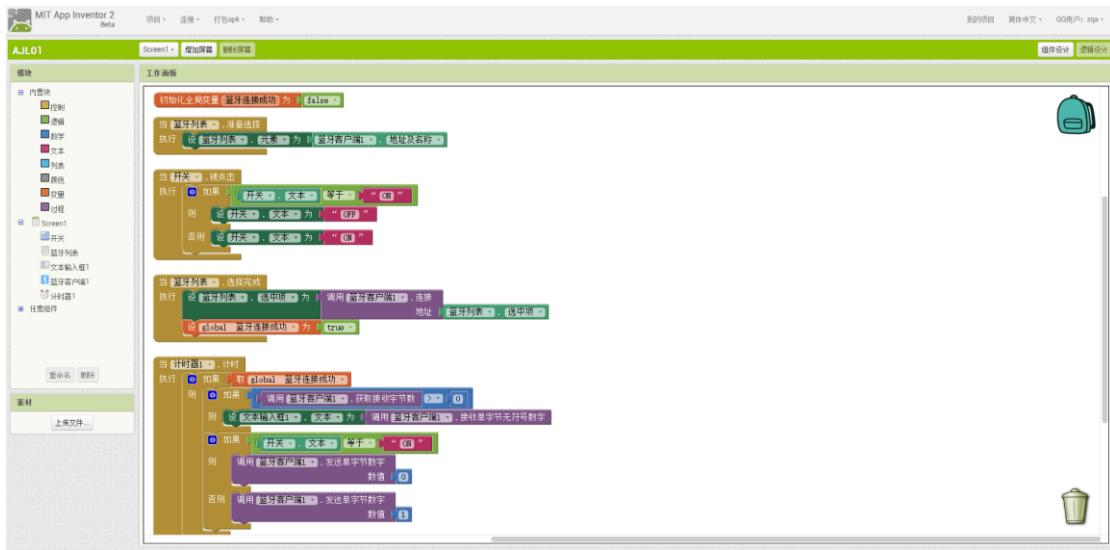
“工作面板”的内容和最后生成的 APP 是完全一致的，即可以通过工作面板看到你最后做出 APP 是什么样子的。



“组件列表”是 APP 中所有组件的关系树，即屏幕 1 中有几个组件、屏幕 2 中有几个组件，名称各是什么。

“组件属性”是各个组件的具体参数设置，大部分参数可以在逻辑程序运行过程中修改。

逻辑设计是用于设计 APP 各组件对应的程序，即设计当用户操作各组件时，APP 做出什么样的反应。



示例 21-1：蓝牙控制 LED

本次教程将通过 APP Inventor 编写一个控制 NOVA 端 LED 灯亮灭的程序。

在编写完整的 APP 之前，先实现按钮的一个小程序，即按一下按钮，按钮的文本由“ON”变为“OFF”；再按一下按钮，按钮的文本由“OFF”变为“ON”；



这些模块的颜色与 Scratch 相似，按照不同的功能种类，有不同的颜色，大家可以根据示例程序中各模块的颜色去模块库中寻找，这里不多赘述。

在做完上述小程序 APP 后，可以让它在手机上运行。可以用 AI 伴侣快速的在手机上运行写好的 APP。



MITAI2Companion.apk

点击“帮助”菜单下的“AI 同伴信息”：



点击链接下载：



浏览器可能会弹出“不安全”提醒，选择继续访问。该网站是安全可靠的，由广州市电教馆开发维护。

手机上打开 AI 伴侣 APP，扫描网页上 AI 伴侣生成的二维码，即可快速的运行编写好的 APP，具体操作流程如下：



蓝牙程序的编写

蓝牙相关程序的编写，需要添加一个“蓝牙客户端”组件，它是非可视组件，即在 APP 中看不见它。

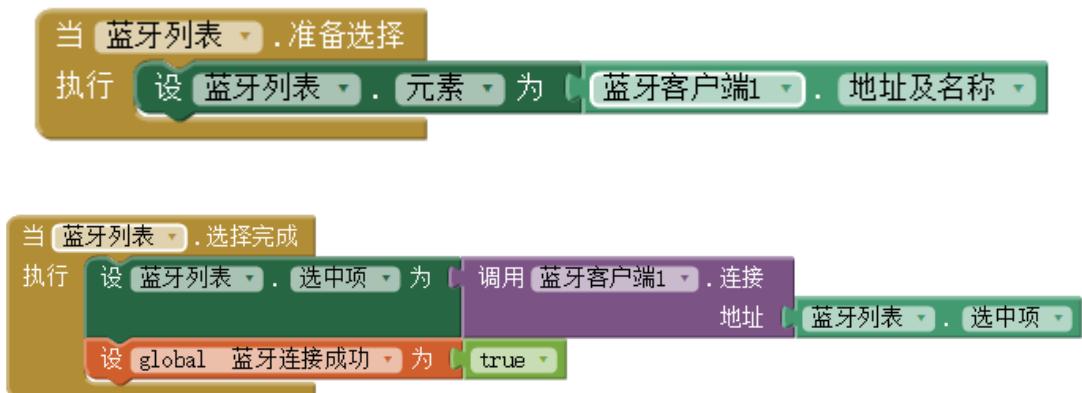


蓝牙模块需要选择之后才能连接，所以这里需要添加一个“列表选择框”组件。



将“列表选择框”组件名和文本名都改为“蓝牙列表”，提升程序可读性。

在逻辑设计界面里编写蓝牙模块选择程序。

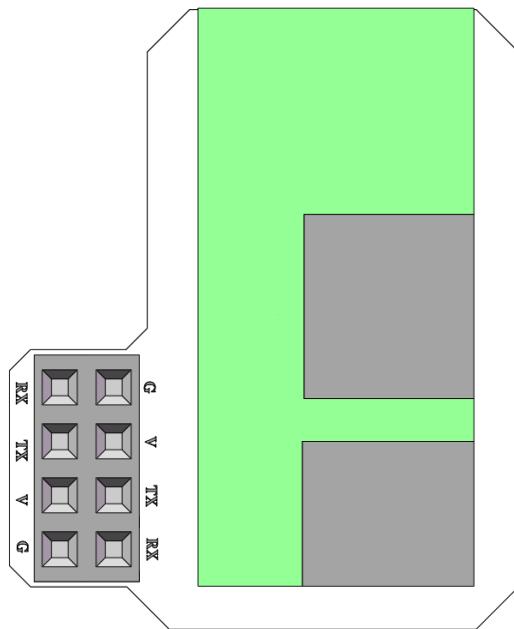


程序运行效果：点击“蓝牙列表”组件，弹出蓝牙地址的选择框；再点击选择要连接的蓝牙地址，之后蓝牙连接成功。

接下来配合 NOVA HD 主控板和蓝牙模块，来实现手机 APP 和蓝牙模块的连接。

NOVA HD 主控板与蓝牙模块的连接

NOVA 的蓝牙模块上有一个 8 针的接口



这里的设计是为了实现防反接，两列的 4 个接口是轴对称的，只要做到与 C0 4 对 4 的接插，就一定不会有问题是。

供电之后，蓝牙模块上的 LED 灯会闪烁，表示未连接，一旦连接成功，LED 将变为长亮。

手机与蓝牙模块的配对

这里用的蓝牙模块为蓝牙 2.0 模块，在连接之前，需要在系统设置中完成配对。

配对示例：（示例系统版本为 Android 5.0）



用 AI 伴侣，将写好的 APP 在手机上运行。按照下图的操作顺序，实现蓝牙模块的连接。



LED 长亮之后，代表蓝牙连接成功。接下去继续编写蓝牙程序，让按钮处于“ON”和“OFF”时发送不同的数值。



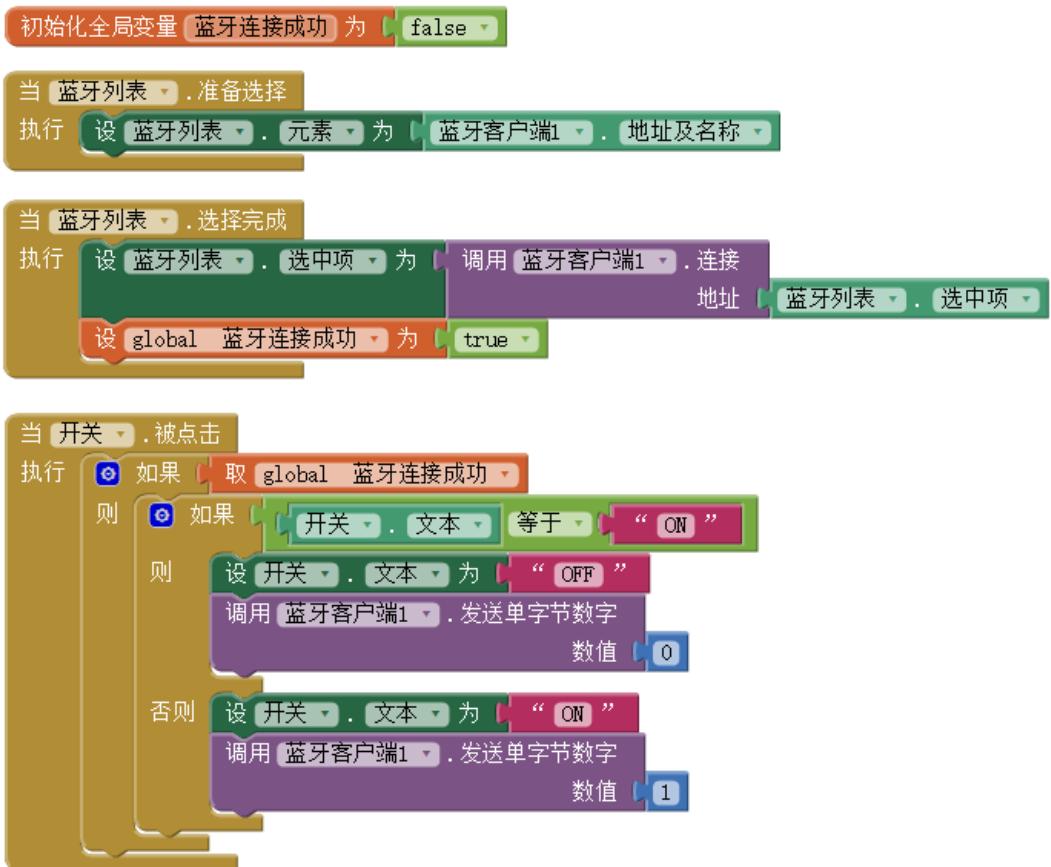
在按钮文本切换的程序中添加蓝牙客户端发送数字的程序模块。

这时运行 APP，如果未连接蓝牙，就电机“ON/OFF”按钮，会弹出一个错误提示，是因为蓝牙无连接情况下发送数字的报错。

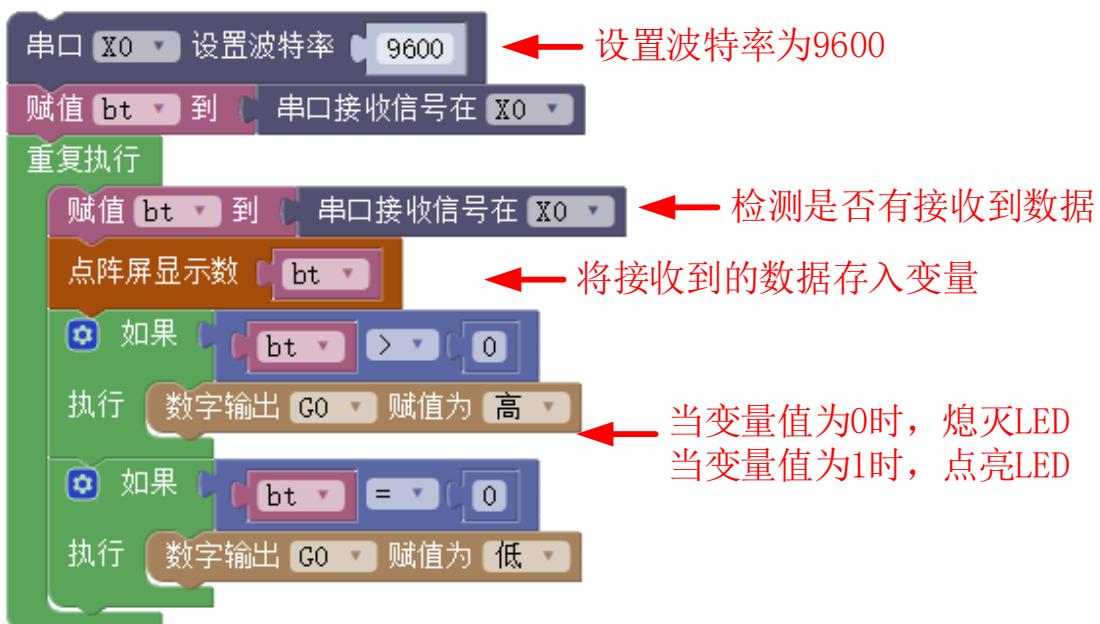


所以这里应该添加一个判断条件。即新建一个全局变量“蓝牙连接成功”，它是一个布尔量，只有“`true`（真）”和“`false`（假）”两个值可取。

初始化时，变量“蓝牙连接成功”为“`false`（假）”，当蓝牙地址选择完成之后，将其变为“`true`（真）”。



手机 APP 端的程序编写完成，接下来编写 NOVA HD 端的程序。



```
1 UART_1=UART(1)
2
3 Screen1=Screen()
4
5 G0=Port('G0')
6
7
8 UART_1.init(9600)
9 bt = UART_1.read()
10 while 1:
11     bt = UART_1.read()
12     Screen1.printText(str(bt))
13     if bt > 0:
14         G0.digitalWrite(1)
15     if bt == 0:
16         G0.digitalWrite(0)
```

全部的编程工作完成，可以用手机控制 LED 的亮灭了，同学们试试看吧

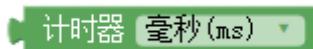
第四部分

编程进阶

第二十二节 计时器

之前的项目案例多为单一功能，不存在多个功能之间产生冲突的问题。如一个复杂应用中需要有 LED 闪烁这个小功能，如果使用第二节中介绍 LED 闪烁程序的方法实现，等待指令将导致整个程序的响应能力变的极差。这节的内容将介绍解决复杂程序中的冲突问题。

认识计时器指令



计时器的值由主控芯片自动产生，单位为毫秒，按时间自动增加，完全不占用程序运行的时间等待指令的运行会使整个程序停止，如果这时输入有改变是读取不到的。这是两者最大的区别。

计时器指令需要配合两个变量一起使用，取名为 t1 和 t2。其中 t1 负责读取系统运行时间，t2 用于与 t1 进行比较。

如希望某事件的时间间隔为 1 秒（1000 毫秒）：

- (1) 让 t1 的值与系统运行时间始终保持一致；
- (2) 当 $t1-t2$ 的值大于 3000（毫秒）时，则让 t2 的值更新为 t1 的值，否则不更新。

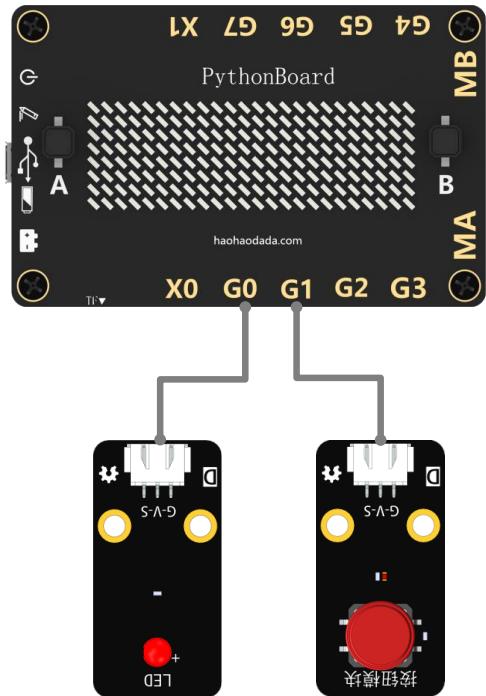
示例 22-1：长按点亮 LED，松开熄灭 LED

长按按键 3 秒以上才能点亮 LED，短按按键则马上熄灭 LED。

元器件列表：

1. Python 主控板 × 1
2. 按键模块 × 1
3. LED 模块 × 1
4. 3Pin 连接线 × 2

电路连接：



程序编写:

Scratch Script:

```

数字输出 G0 赋值为 低
赋值 t1 到 计时器 毫秒(ms)
赋值 t2 到 计时器 毫秒(ms)

重复执行
  赋值 t1 到 计时器 毫秒(ms)
  如果 读数字口 G1 = 1
    执行 数字输出 G0 赋值为 高
    否则 数字输出 G0 赋值为 低
  否则 赋值 t2 到 t1

```

Python Code:

```

1 G0=Port('G0')
2
3 G1=Port('G1')
4
5
6 G0.digitalWrite(0)
7 t1 = millis()
8 t2 = millis()
9 while 1:
10   t1 = millis()
11   if G1.digitalRead() == 1:
12     if t1 - t2 > 3000:
13       G0.digitalWrite(1)
14     else:
15       G0.digitalWrite(0)
16   else:
17     t2 = t1

```

第二十三节 输入计数及其应用

要实现诸如记录按键按下次数，或通过按下松开按键一次切换输出器件状态，需要巧用变量来实现。

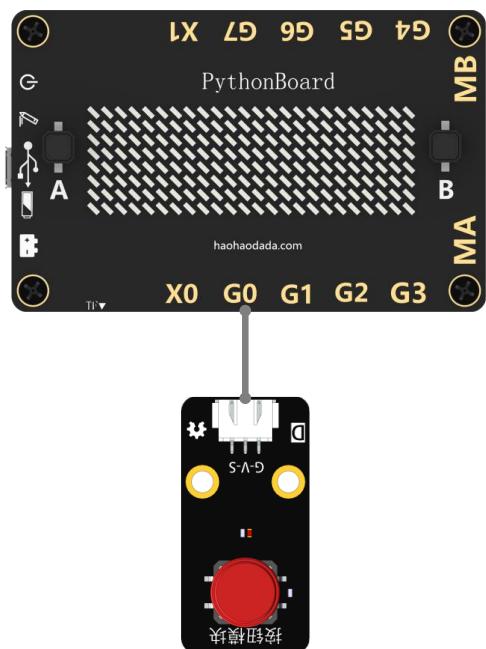
示例 23-1：记录按键按下松开的次数，并显示到点阵屏上

按下按键一次，不论按下多长时间，只要不松开，点阵屏上的数字只增加 1。

元器件列表：

1. Python 主控板 × 1
2. 按键模块 × 1
3. 3Pin 连接线 × 1

电路连接：



程序编写：

大家首先想到应该是下面这样的程序：

The Scratch script shows a repeating loop that initializes `num` to 0, checks if port G0 is 1, and then increments `num` by 1. It also clears the screen and displays `num`. The Python code on the right translates this into a more efficient version using a while loop and a flag variable.

```

1 Screen1=Screen()
2
3 G0=Port('G0')
4
5
6 Screen1.clearScreen()
7 num = 0
8 while 1:
9     if G0.digitalRead() == 1:
10         num = num + 1
11     Screen1.clearScreen()
12     Screen1.printText(str(num))

```

这个程序的运行结果是按下一次，数码管显示的数字增加的非常快。即使手速再快，每按一次，数码管显示的数值都会至少增加几十。

大家应该都能分析出程序错误的原因：按键被按下的时间里，程序运行了很多次，变量 `num` 自增了很多次。

所以，问题的关键在于按键被按下时，如何让计数程序只运行一次。

这需要声明一个变量，用于记录按键的松开和按下状态，工程上这样的变量被称为“标志位”。

The annotated Scratch script shows a fix using a flag variable. It initializes `num` and `flag` to 0. In the loop, it checks if both port G0 is 1 and `flag` is 0. If true, it increments `num` and sets `flag` to 1. When port G0 is 1 and `flag` is 1, it only clears the screen. When port G0 is 0 and `flag` is 1, it sets `flag` back to 0. Finally, it displays `num`.

Annotations:

- 记录次数的变量 (Variable for counting)
- 按键状态标志位 (Flag for key state)
- 记录次数的程序由按键状态和标志位共同控制。 (The counting program is controlled by the key state and the flag.)
- 按键按下后第一次运行时flag=0, times自增1；之后因为flag变为1, times自增的程序则不再运行。 (When the button is pressed for the first time, flag=0, times increases by 1; after that, because flag becomes 1, the increasing program no longer runs.)
- 按键松开后，标志位flag重新改为0，为下一次按键按下时做准备。 (When the button is released, the flag is reset to 0, preparing for the next press.)
- 点阵屏显示次数 (Displaying the count on the dot matrix screen)

```

1 Screen1=Screen()
2
3 G0=Port('G0')
4
5
6 Screen1.clearScreen()
7 num = 0
8 flag = 0
9 while 1:
10     if G0.digitalRead() == 1 and flag == 0:
11         num = num + 1
12         flag = 1
13     if G0.digitalRead() == 1:
14         flag = 0
15     Screen1.clearScreen()
16     Screen1.printText(str(num))

```

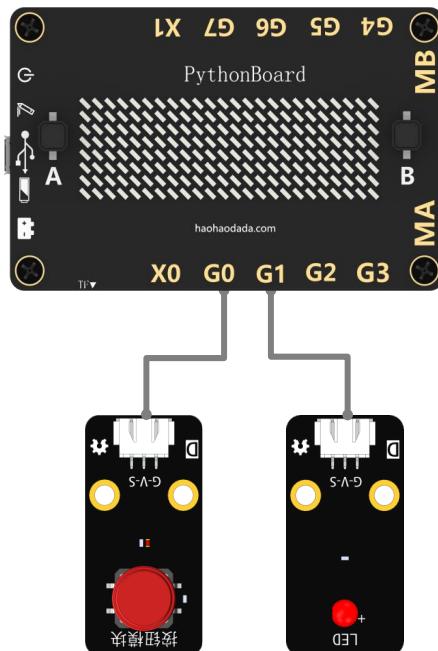
示例 23-2：点控 LED

按键按下松开一次，LED 亮；再按下松开一次，LED 灭；如此往复。使用重复程序块和使用延时程序块一样，会阻断程序的运行。接下来介绍的才是真正正确的办法。

元器件列表：

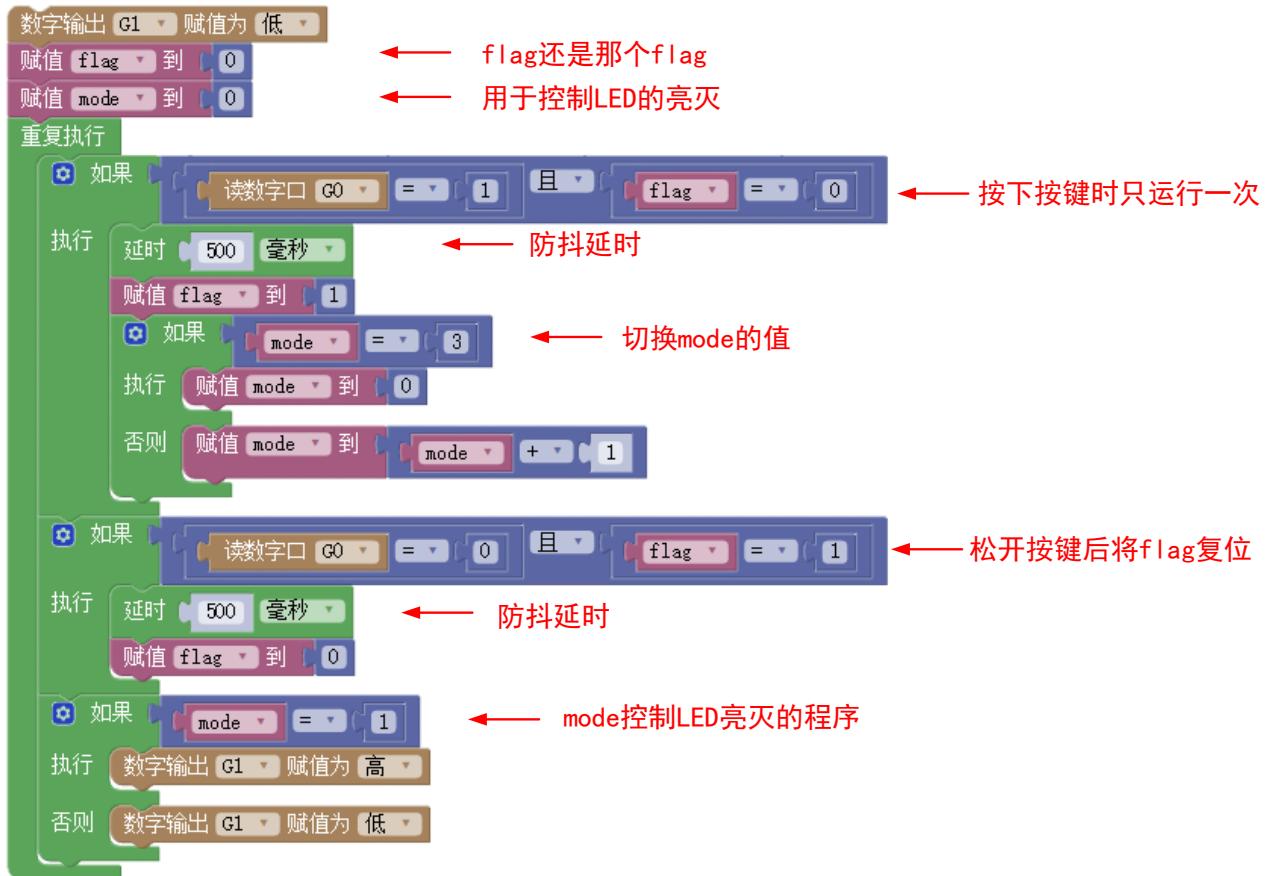
1. Python 主控板 × 1
2. 按键模块 × 1
3. LED 模块 × 1
4. 3Pin 连接线 × 2

电路连接：



程序编写：

根据示例 23-1，可以轻松的理解下面的程序：



```

1 G0=Port('G0')
2
3 G1=Port('G1')
4
5
6 G0.digitalWrite(0)
7 flag = 0
8 mode = 0
9 while 1:
10    if G1.digitalRead() == 1 and flag == 0:
11        delay(500)
12        flag = 1
13        if mode == 3:
14            mode = 0
15        else:
16            mode = mode + 1
17    if G1.digitalRead() == 0 and flag == 1:
18        delay(500)
19        flag = 0
20        if mode == 1:
21            G0.digitalWrite(0)
22        else:
23            G0.digitalWrite(0)

```

其中，按键按下部分的程序如下：

```

1 G0=Port('G0')
2 G1=Port('G1')
3
4
5
6 if mode == 1:
7     mode = 0
8 else:
9     mode = mode + 1

```

这是更通用的方式，当按键控制的模式不只有亮和灭两种状态时，依然可用。具体应用请看示例 23-3。

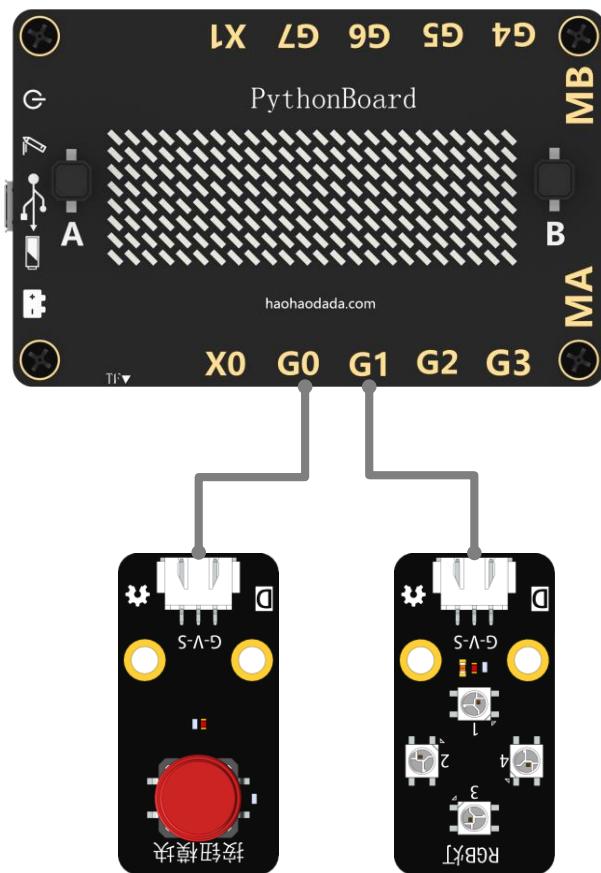
示例 23-3：按键控制切换 RGB 灯的颜色

按键按下松开一次，RGB 灯的颜色切换一次。这种应用在家里的吸顶灯中非常常见。

元器件列表：

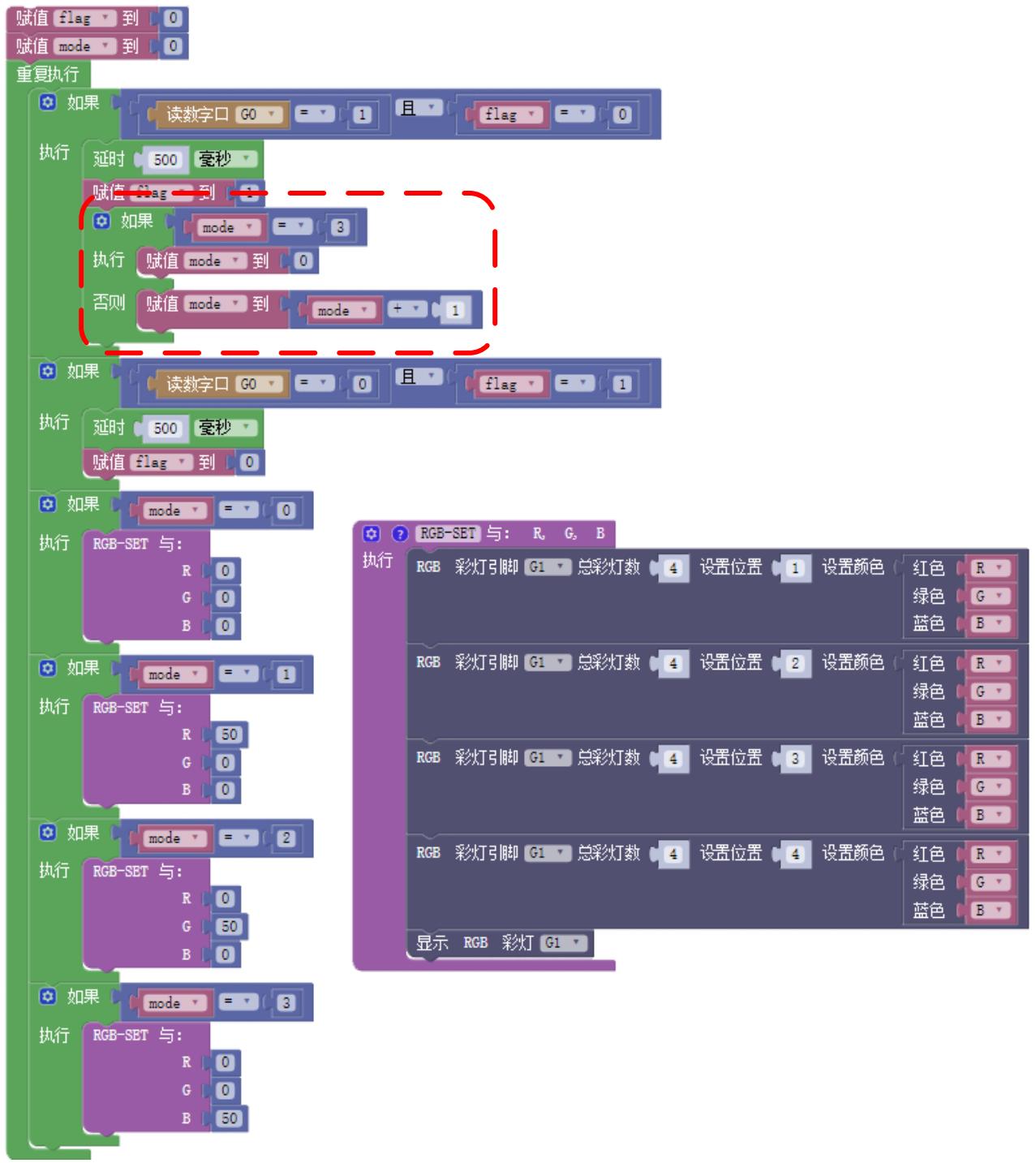
1. Python 主控板 × 1
2. 按键模块 × 1
3. RGB 模块 × 1
4. 3Pin 连接线 × 2

电路连接：



程序编写：

只需在示例 23-2 程序的基础上稍作修改即可：



```
1 G0=Port('G0')
2
3 rgb_1=WS2812('G1',4)
4
5 """描述该功能...
6 """
7 def RGB_SET(R, G, B):
8     global mode, flag
9     rgb_1.colorSet(1,NaN,G,B)
10    rgb_1.colorSet(2,NaN,G,B)
11    rgb_1.colorSet(3,NaN,G,B)
12    rgb_1.colorSet(4,NaN,G,B)
13    rgb_1.show()
14
15
16 flag = 0
17 mode = 0
18 while 1:
19     if G0.digitalRead() == 1 and flag == 0:
20         delay(500)
21         flag = 1
22         if mode == 3:
23             mode = 0
24         else:
25             mode = mode + 1
26     if G0.digitalRead() == 0 and flag == 1:
27         delay(500)
28         flag = 0
29     if mode == 0:
30         RGB_SET(0, 0, 0)
31     if mode == 1:
32         RGB_SET(50, 0, 0)
33     if mode == 2:
34         RGB_SET(0, 50, 0)
35     if mode == 3:
36         RGB_SET(0, 0, 50)
```

注意虚线红框的程序，与示例 23-2 的程序基本相同，只是 mode 的取值范围变为了 0、1、2、3。

以上程序中的输入计数用的都是按键这种数字量输入信号，如果换成模拟量信号呢？聪明的同学肯定已经想到办法了。

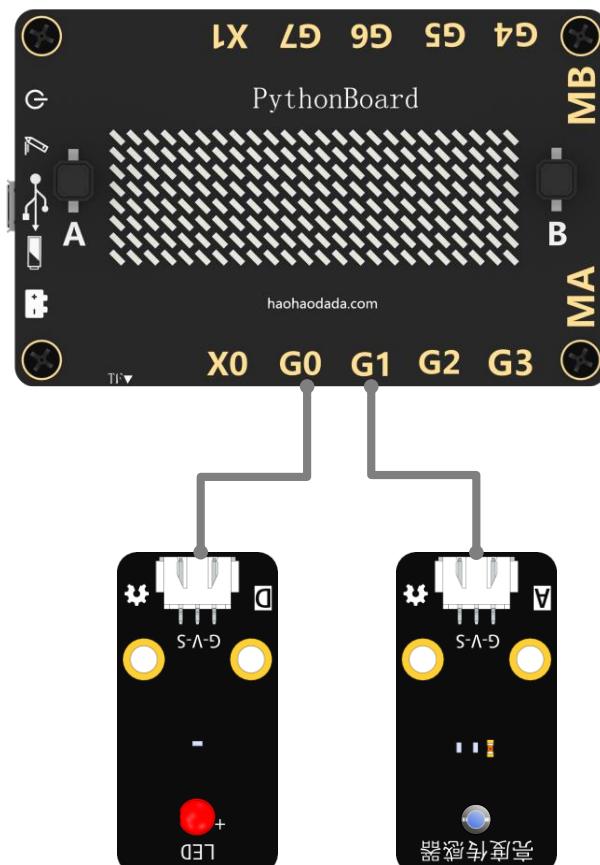
示例 23-4：亮度传感器控制 LED 灯的亮灭（点控）

摸一次亮度传感器，LED 亮，再摸一次，LED 灭，如此往复。

元器件列表：

1. Python 主控板 × 1
2. 亮度传感器模块 × 1
3. LED 模块 × 1
4. 3Pin 连接线 × 2

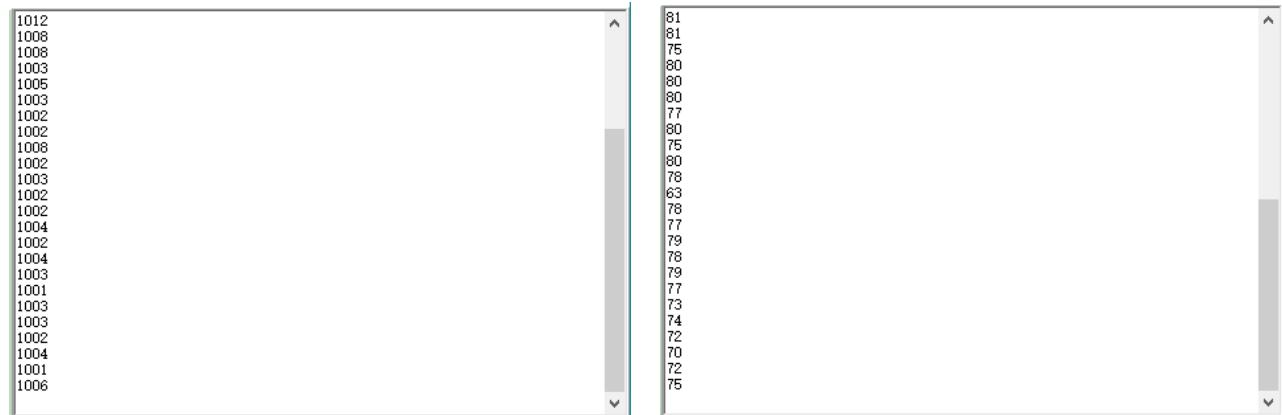
电路连接：



程序编写：

只需在示例 23-2 程序的基础上稍作修改，即将按下按钮替换为亮度值小于阈值；将松开按钮替换为亮度值大于阈值即可。所以需要先测定这个阈值。

定义一个变量 CV 为认定亮度传感器是否被触摸的阈值（Critical Value），这个阈值应该设为多少，需要做一次测定。通过串口助手打印测试亮度传感器在有无触摸时的数值。



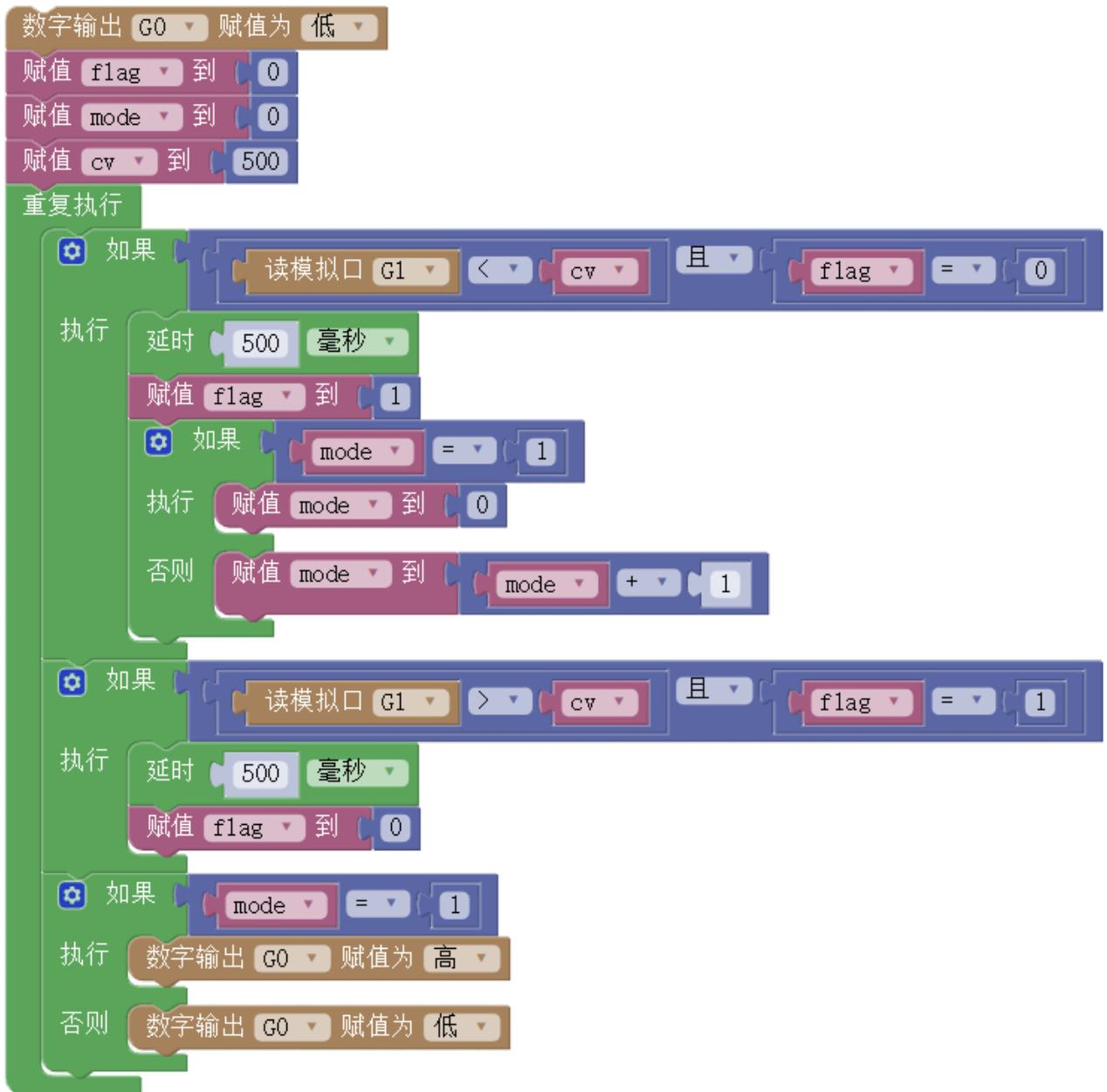
未触摸传感器时

触摸传感器时

未触摸传感器时	触摸传感器时
1012	81
1008	81
1008	75
1003	80
1005	80
1003	80
1002	77
1002	80
1008	75
1002	80
1002	78
1003	63
1002	78
1002	77
1004	77
1002	79
1004	78
1003	79
1001	77
1003	73
1003	74
1002	72
1004	70
1001	72
1006	75

阈值的选取可以从 70 到 1000 之间选取一个，本例中选取的阈值为 500。

那么程序为：



```
1 G0=Port('G0')
2
3 G1=Port('G1')
4
5
6 G0.digitalWrite(0)
7 flag = 0
8 mode = 0
9 cv = 500
10 while 1:
11     if G1.analogRead() < cv and flag == 0:
12         delay(500)
13         flag = 1
14         if mode == 1:
15             mode = 0
16         else:
17             mode = mode + 1
18     if G1.analogRead() > cv and flag == 1:
19         delay(500)
20         flag = 0
21         if mode == 1:
22             G0.digitalWrite(1)
23         else:
24             G0.digitalWrite(0)
```

通过以上四个案例可以总结出：利用“标志位”的方法，可以让持续型信号触发一次跳变信号，避开重复运行程序导致信号多次跳变的问题。

