pinpong

发布 *0.1*

2021 年 08 月 26 日

pinpong 介绍

pinpong 库是一套控制开源硬件主控板的 Python 库，基于 Firmata 协议并兼容 MicroPython 语法，5 分钟即可让你上手使用 Python 控制开源硬件。

借助于 pinpong 库，直接用 Python 代码就能给各种常见的开源硬件编程。其原理是给开源硬件烧录一个特定的固件，使开源硬件可以通过串口与电脑通讯，执行各种命令。

pinpong 库的名称由 “Pin” 和 “Pong” 组成，“Pin” 指引脚，“PinPong” 为 “乒乓球” 的谐音，指信号的往复。

pinpong 库的设计，是为了让开发者在开发过程中不用被繁杂的硬件型号束缚，而将重点转移到软件的实现。哪怕程序编写初期用 Arduino 开发，部署时改成了掌控板，只要修改一下硬件的参数就能正常运行，实现了 “一次编写处处运行”。

注意： 当前 PinPong 库正在快速更新中，已支持 Arduino 系列 uno、leonardo、mega2560,ESP32 系列掌控板（handpy）以及 micro:bit 板，传感器支持 50+，其他主控板及更多扩展库将逐步支持。

本文档推荐阅读流程：

1. 查看安装教程进行准备
2. 查看示例快速上手测试
3. 查看教程进行系统学习

点击观看 pinpong 入门视频互动教程：https://www.bilibili.com/video/BV17K4y1T7MF


扫一扫 手机看
00:00 / 03:05
360P
进入bilibili,一起发弹幕吐槽!
去吐槽

CHAPTER 1

简介

PinPong 库是一套控制开源硬件主控板的 Pyhton 库，基于 Firmata 协议并兼容 MicroPython 语法，5 分钟即可让你上手使用 Python 控制开源硬件。

借助于 PinPong 库，直接用 Python 代码就能给各种常见的开源硬件编程。其原理是给开源硬件烧录一个特定的固件，使开源硬件可以通过串口与电脑通讯，执行各种命令。

PinPong 库的名称由“Pin”和“Pong”组成，“Pin”指引脚，“PinPong”为“乒乓球”的谐音，指信号的往复。

pinpong 库的设计，是为了让开发者在开发过程中不用被繁杂的硬件型号束缚，而将重点转移到软件的实现。哪怕程序编写初期用 Arduino 开发，部署时改成了掌控板，只要修改一下硬件的参数就能正常运行，实现了“一次编写处处运行”。

注意： 当前 PinPong 库正在快速更新中，已支持 Arduino 系列 uno、leonardo、mega2560,ESP32 系列掌控板（handpy）以及 micro:bit(microbit) 板，传感器支持 50+，其他主控板及更多扩展库将逐步支持。

本文档推荐阅读流程：

1. 查看安装教程进行准备
2. 查看示例快速上手测试
3. 查看教程进行系统学习

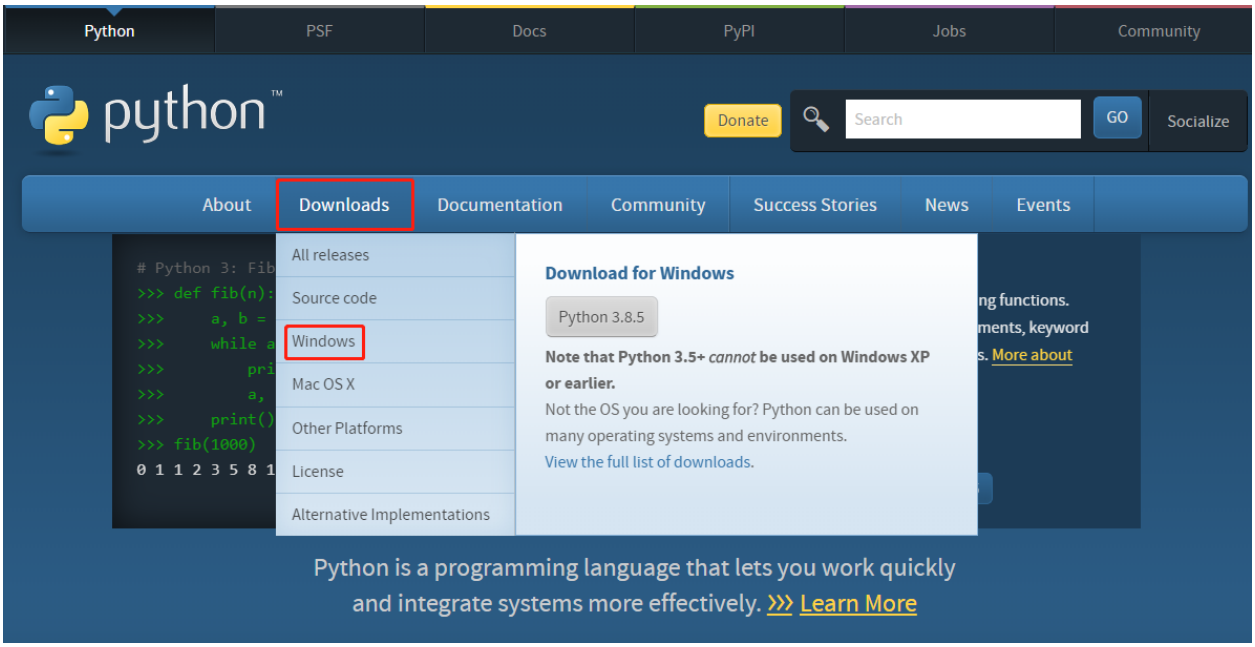
CHAPTER 2

安装教程

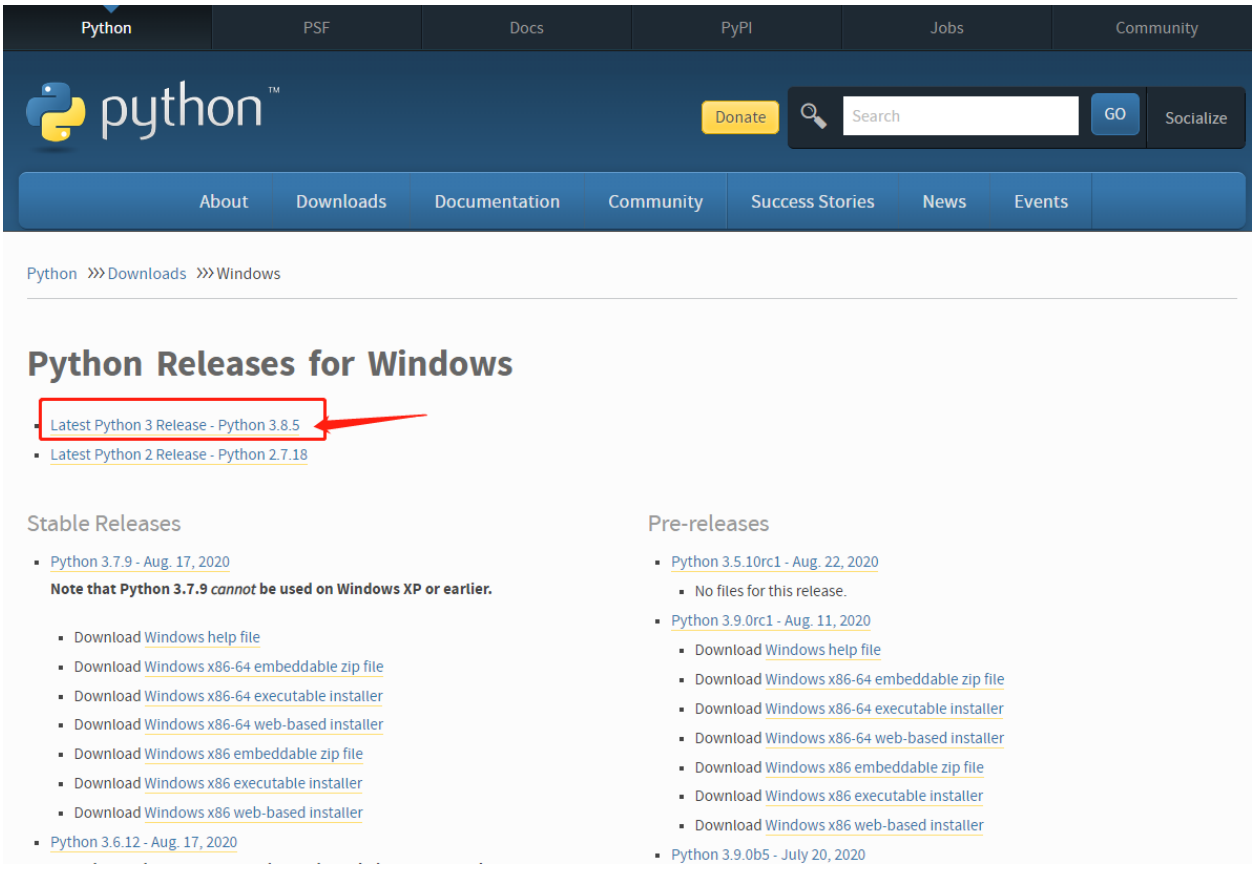
2.1 Windows 平台安装

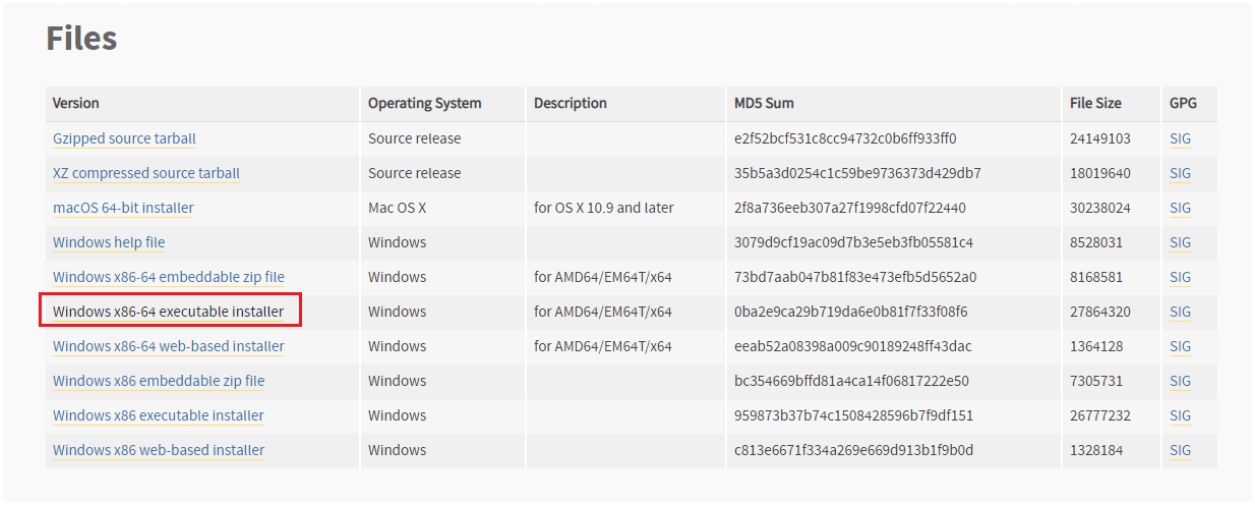
1. 安装 Python3。pinpong 库依赖 Python，因此请确保电脑安装了 Python3，如果已经安装，此步可以略过。

- 如果没有安装 Python3，则需要进行安装, Python 下载地址：点击打开



- Windows 平台对应点击 Windows，然后选择最新版本下载（也可选择其他版本）

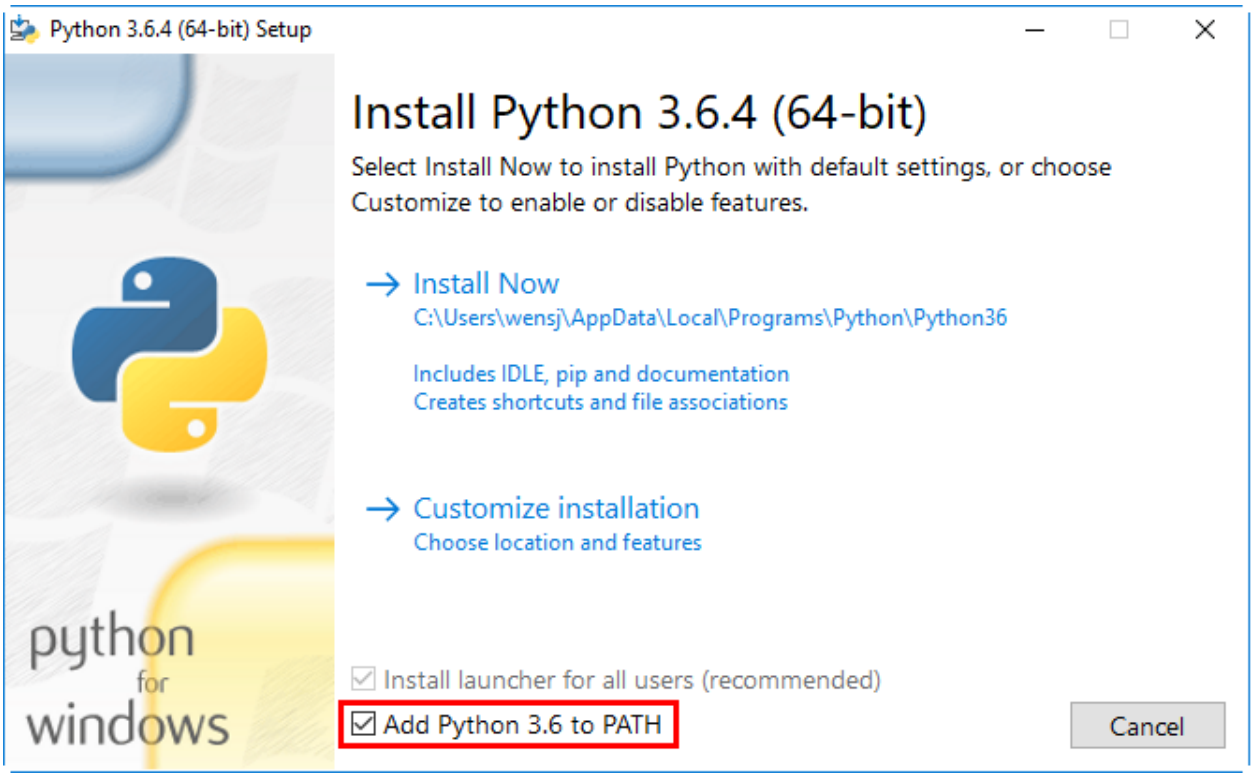




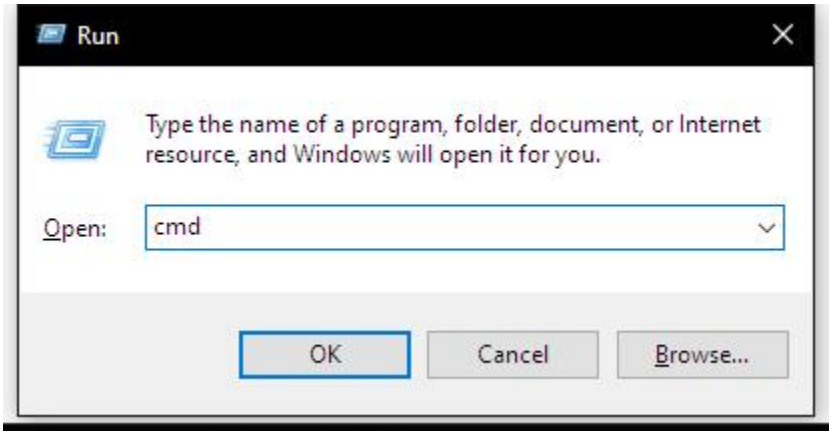
Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e2f52bcf531c8cc94732c0b6ff933ff0	24149103	SIG
XZ compressed source tarball	Source release		35b5a3d0254c1c59be9736373d429db7	18019640	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	2f8a736eeb307a27f1998cfd07f22440	30238024	SIG
Windows help file	Windows		3079d9cf19ac09d7b3e5eb3fb05581c4	8528031	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	73bd7aab047b81f83e473efb5d5652a0	8168581	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	0ba2e9ca29b719da6e0b81f7f33f08f6	27864320	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	eeab52a08398a009c90189248ff43dac	1364128	SIG
Windows x86 embeddable zip file	Windows		bc354669bffd81a4ca14f06817222e50	7305731	SIG
Windows x86 executable installer	Windows		959873b37b74c1508428596b7f9df151	26777232	SIG
Windows x86 web-based installer	Windows		c813e6671f334a269e669d913b1f9b0d	1328184	SIG

- 下载完成后根据引导进行安装，注意安装最后一步勾选 add to path，将 python 加入到环境变量。

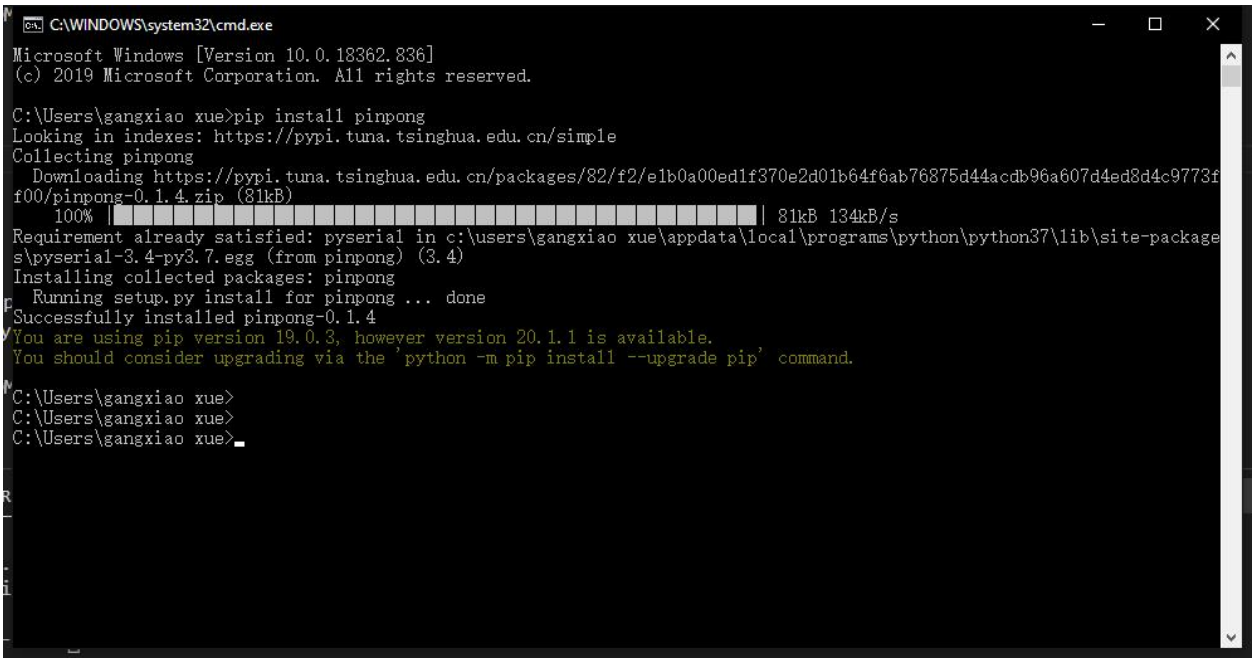


2. 打开命令提示窗。win+R 快捷键，输入 cmd

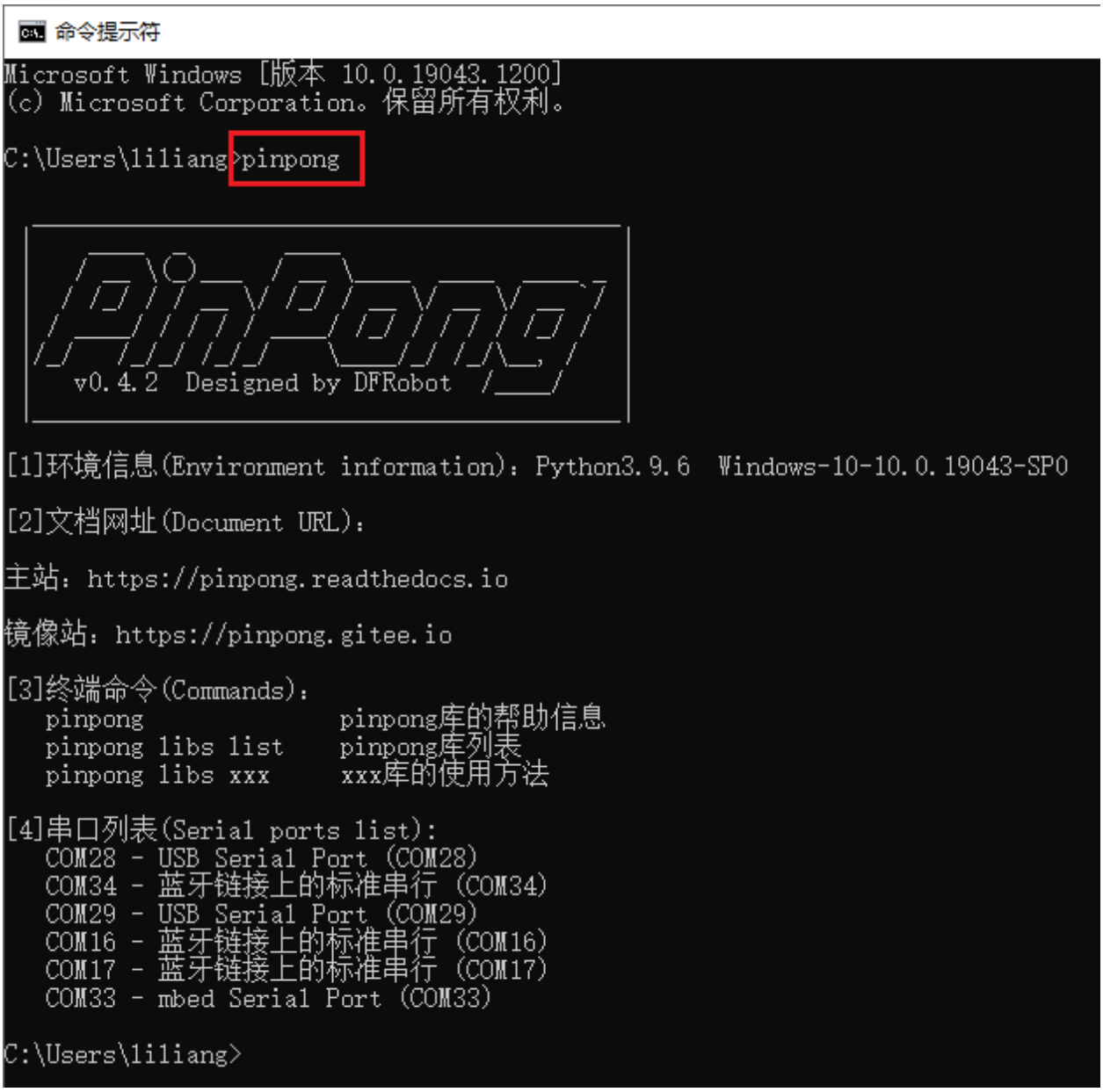


3. 安装 pinpong 库。小黑窗中输入 pip install pinpong ，等待片刻即可安装完成。

注意： 如果网络较慢安装不成功，可以指定国内源进行安装：pip install pinpong -i http://mirrors.aliyun.com/pypi/simple/

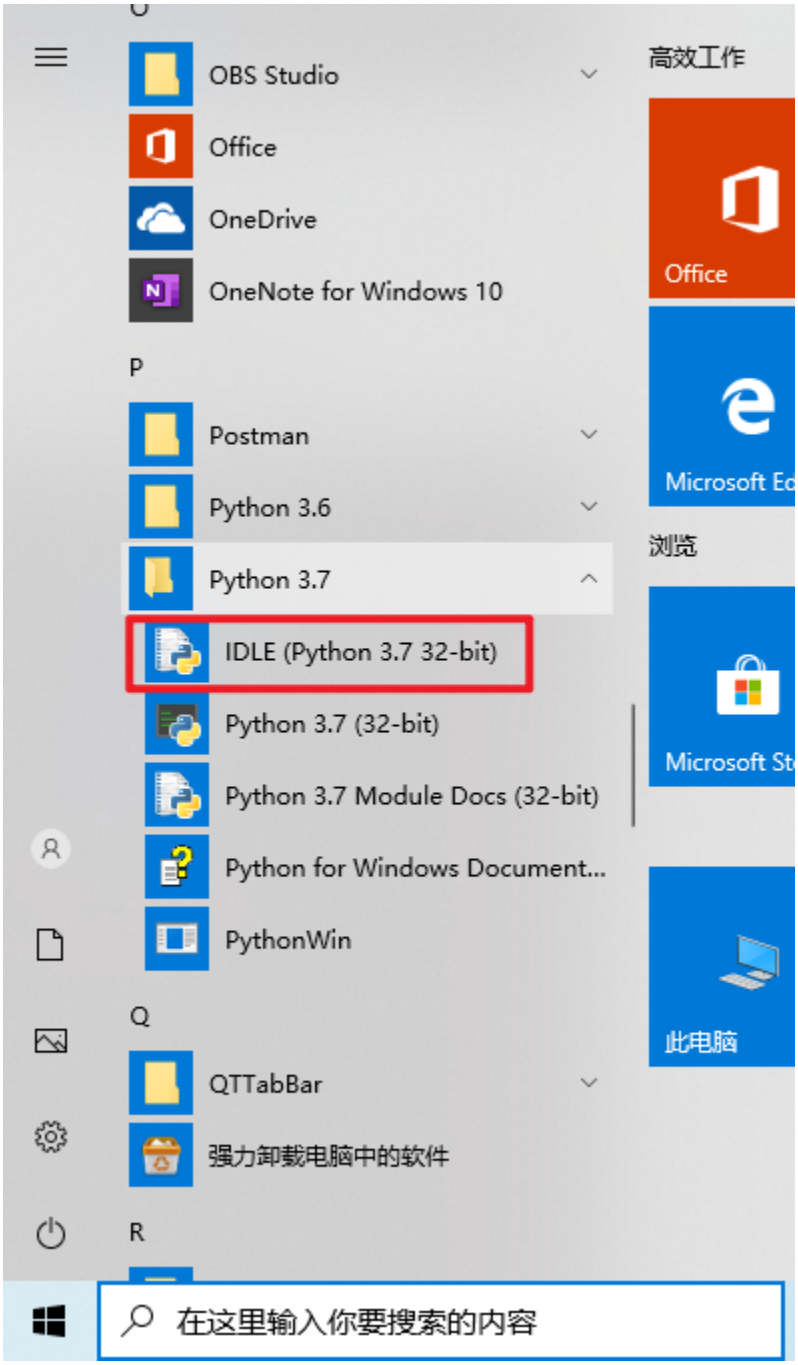


4. 帮助命令。在小黑窗中输入 pinpong ， 即可输出当前版本信息、官方文档网址、库列表查看、端口号。



2.1.1 开始第一个程序

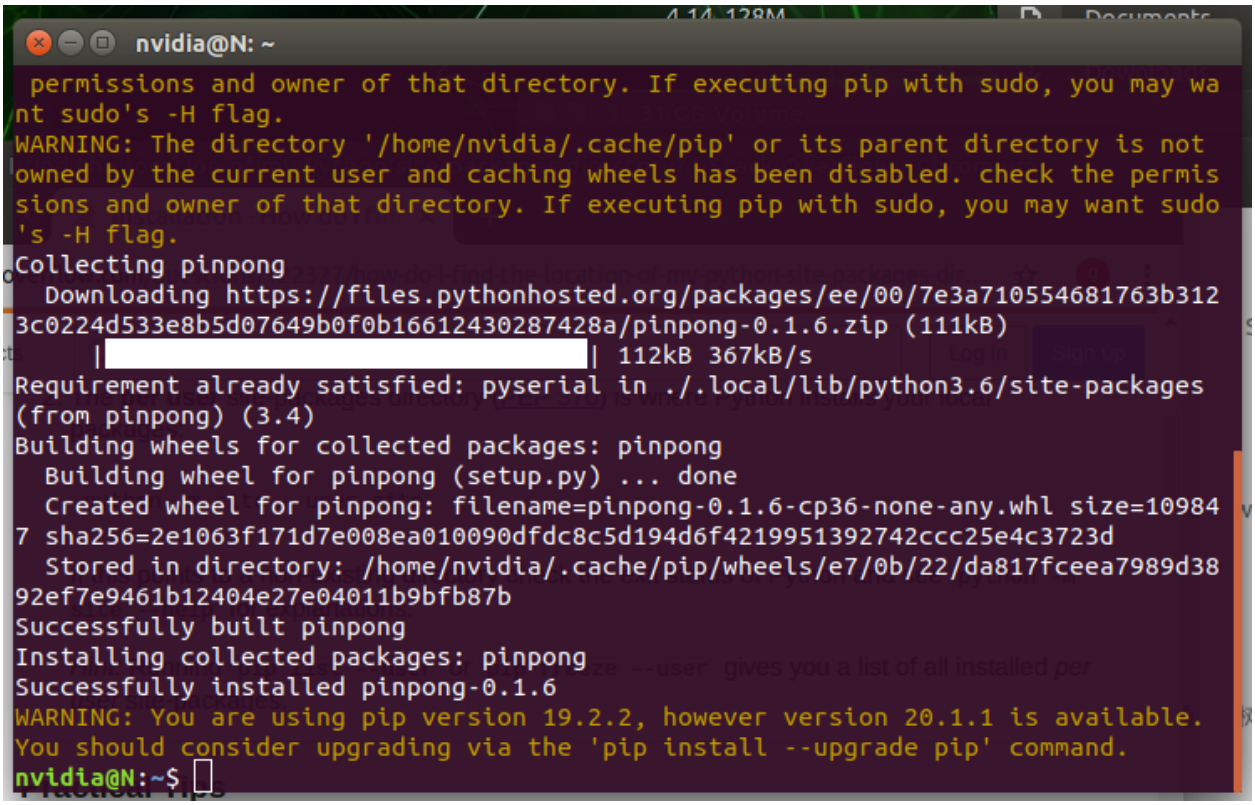
1. 连接 arudino uno 板至电脑；
2. 打开 IDLE 编辑器，新建文件 (New File)；
3. 从本文档 “PINPONG 示例” 复制 “blink” 示例程序的代码到 IDLE 中，点击 “运行 (RUN) > Run Module” 或按键盘上 F5 键即可运行代码；
4. Python Shell 窗口即会显示 PinPong 的 logo 和运行状况，Uno 板载 LED 开始闪烁，运行成功，按键盘上的 Ctrl+C 组合键即可停止代码运行。
5. 接下来请查看其他教程或运行其他示例程序进一步学习吧。


OBS Studio
Office
OneDrive
OneNote for Windows 10
P
Postman
Python 3.6
Python 3.7
IDLE (Python 3.7 32-bit)
Python 3.7 (32-bit)
Python 3.7 Module Docs (32-bit)
Python for Windows Document...
PythonWin
Q
QTTabBar
强力卸载电脑中的软件
R
高效工作
Office
浏览
此电脑
在这里输入你要搜索的内容

2.2 Linux 平台安装

终端中输入 sudo pip install pinpong 即可安装。

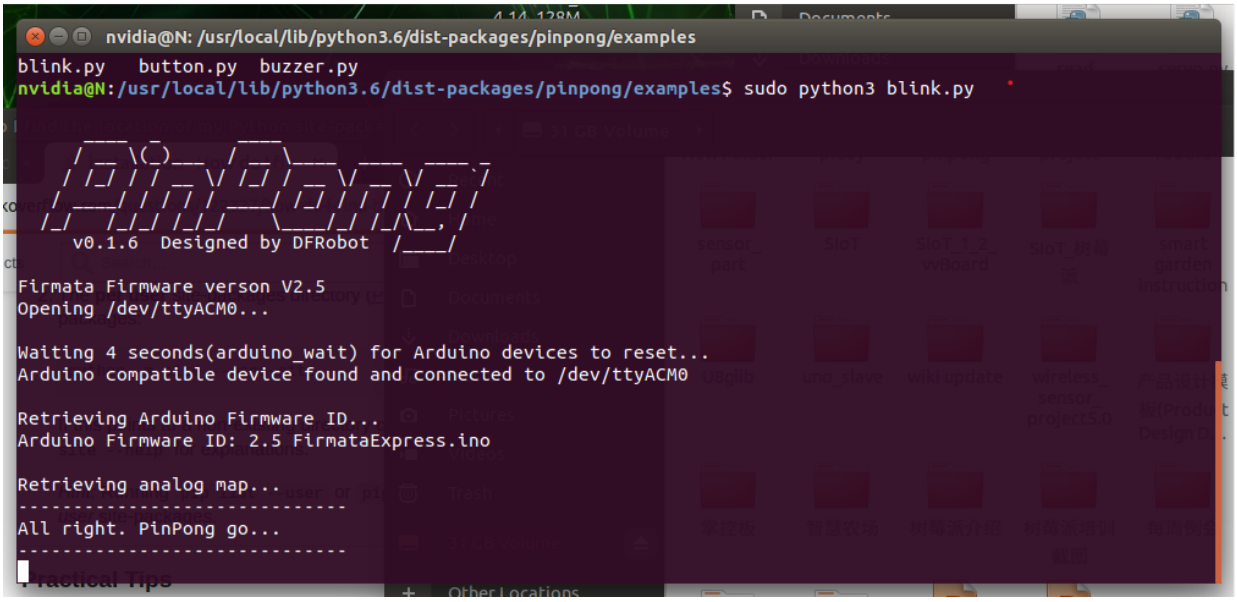
```
$ sudo pip install pinpong
```



帮助命令。在小黑窗中输入 pinpong ， 即可输出当前版本信息、官方文档网址、库列表查看、端口号。

2.2.1 开始第一个程序

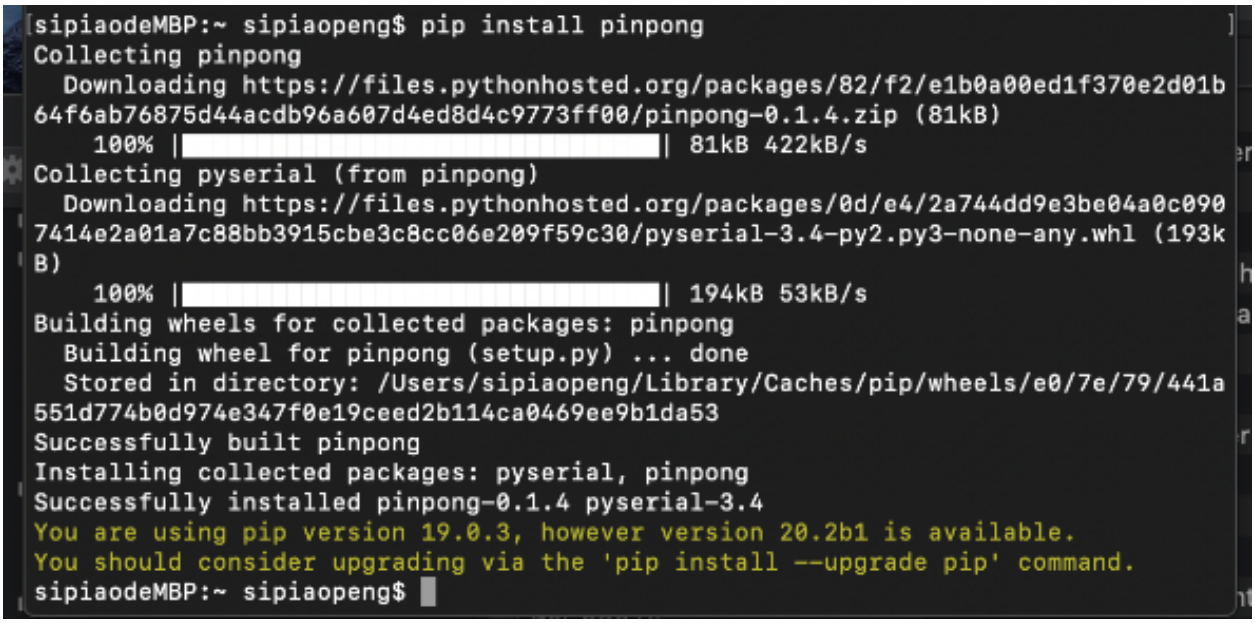
1. 连接 arudino uno 板至电脑;
2. 从本文档 “PINPONG 示例” 复制 “blink” 示例程序的代码到 Python 编辑器中，运行代码;
3. 接下来请查看其他教程或运行其他示例程序进一步学习吧。

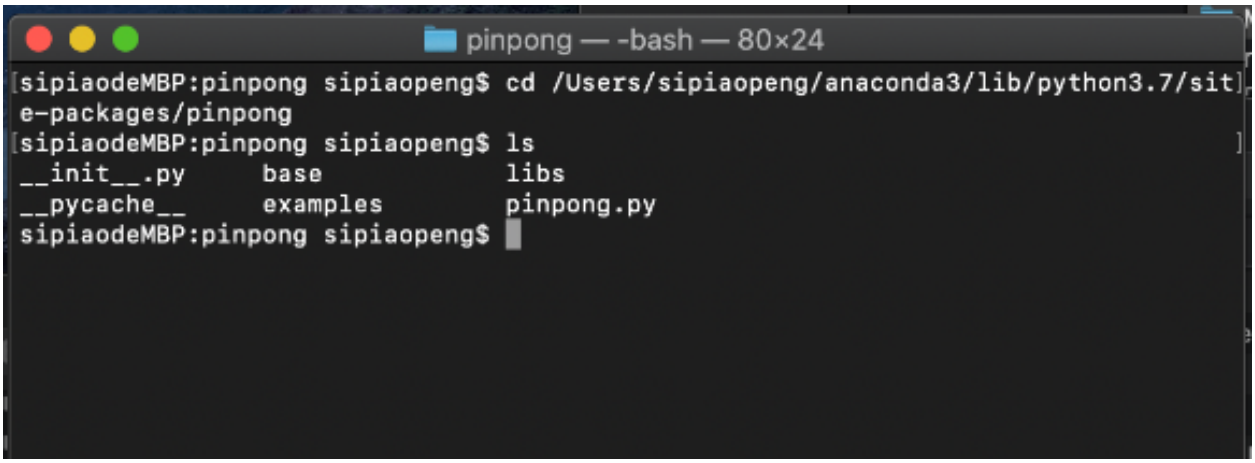


2.3 Mac OS X 平台安装

启动用命令行,(打开任意 finder 窗口，键入 Shift+Command+U)，双击 “终端”。输入命令行，安装 pinPong 库

```
$ pip install pinpong
```





帮助命令。在小黑窗中输入 pinpong ，即可输出当前版本信息、官方文档网址、库列表查看、端口号。

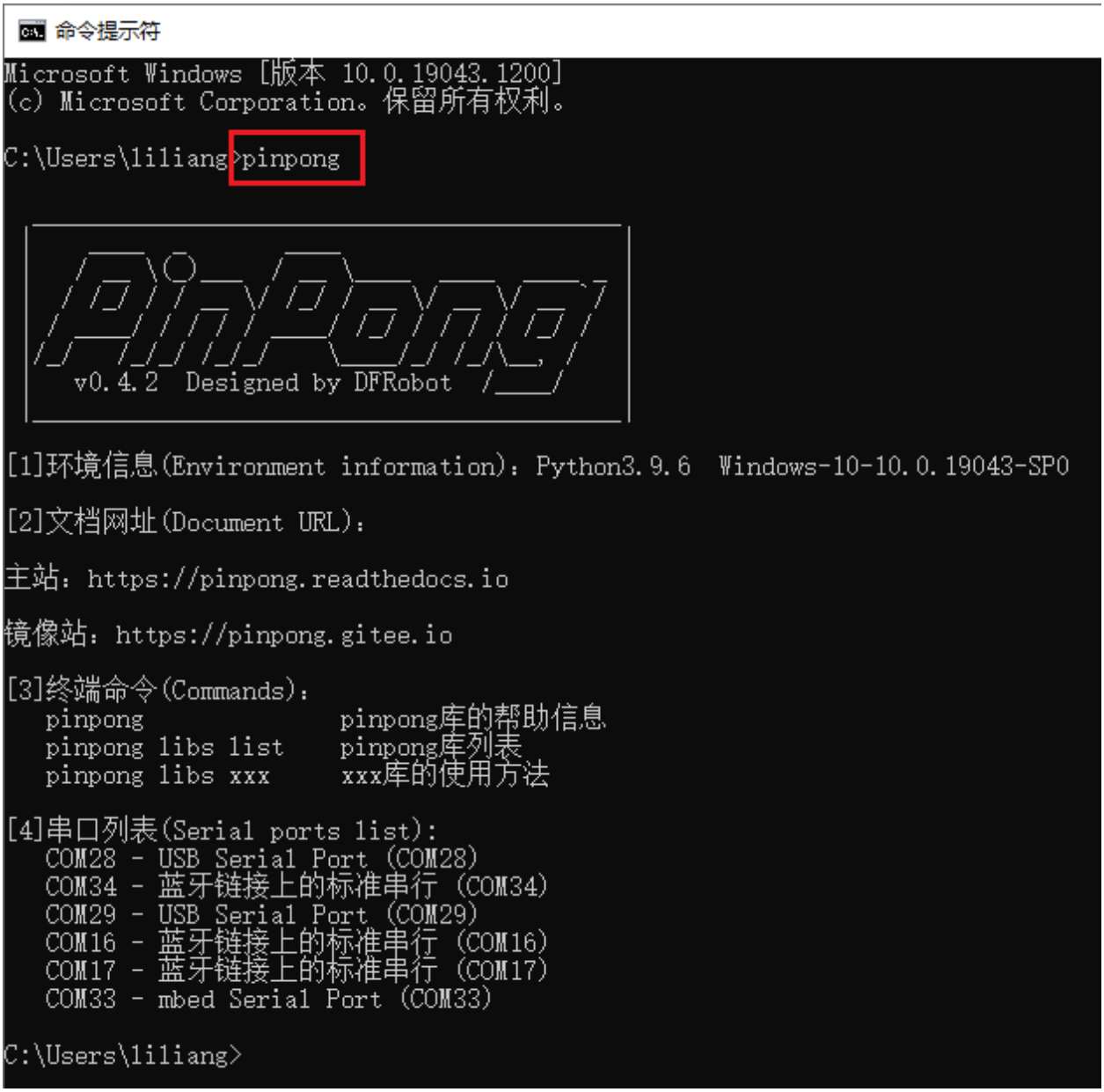
2.3.1 开始第一个程序

1. 连接 arudino uno 板至电脑；
2. 从本文档“PINPONG 示例”复制“blink”示例程序的代码到 Python 编辑器中，运行代码；
3. 接下来请查看其他教程或运行其他示例程序进一步学习吧。

```
$ python blink.py
```

2.4 查看 pinpong 版本

pinpong 安装成功后，在小黑窗中输入 pinpong ，即可输出当前版本信息、官方文档网址、库列表查看、端口号。



2.5 更新 pinpong 库

pinpong 目前处于快速更新迭代中，因此会不定期进行更新，通过如下命令可以进行版本更新：

```
$ pip install -U pinpong
```

2.6 卸载 pinpong 库

如果 pinpong 库出现异常情况时，可以尝试使用如下命令卸载 pinpong 库然后再使用 install 命令安装：

```
$ pip uninstall pinpong
```

CHAPTER 3

基础库示例

示例程序可帮助你快速验证模块的使用，复制粘贴代码到 python 编辑器中，并修改 Board 初始化版型为你使用的板子型号即可

- 常用库示例中的模块通过 board 库导入

3.1 1-01-blink: 数字输出

```
# -*- coding: UTF-8 -*-
# 实验效果：控制 arduino UNO 板载 LED 灯一秒闪烁一次
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板
import time
from pinpong.board import Board,Pin

Board("uno").begin()                # 初始化，选择板型（uno、microbit、RPi、handpy）和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化


led = Pin(Pin.D13, Pin.OUT) # 引脚初始化为电平输出
```

（下页继续）

(续上页)

```
while True:
  #led.value(1) # 输出高电平 方法 1
  led.write_digital(1) # 输出高电平 方法 2
  print("1") # 终端打印信息
  time.sleep(1) # 等待 1 秒 保持状态

  #led.value(0) # 输出低电平 方法 1
  led.write_digital(0) # 输出低电平 方法 2
  print("0") # 终端打印信息
  time.sleep(1) # 等待 1 秒 保持状态
```

3.2 1-02-button: 数字输入

```
# -*- coding: UTF-8 -*-
# 实验效果: 使用按钮控制 arduino UNO 板载亮灭
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, 主控板 D8 接一个按钮模块
import time
from pinpong.board import Board,Pin

Board("uno").begin()                # 初始化, 选择板型 (uno、leonardo、xugu) 和端口号, 不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

btn = Pin(Pin.D8, Pin.IN) # 引脚初始化为电平输入
led = Pin(Pin.D13, Pin.OUT)

while True:
  #v = btn.value()  # 读取引脚电平方法 1
  v = btn.read_digital()  # 读取引脚电平方法 2
  print(v)  # 终端打印读取的电平状态
  #led.value(v)  # 将按钮状态设置给 led 灯引脚  输出电平方法 1
  led.write_digital(v) # 将按钮状态设置给 led 灯引脚  输出电平方法 2
  time.sleep(0.1)
```

3.3 1-03-adc: 模拟输入

```
# -*- coding: UTF-8 -*-
# 实验效果: 打印 UNO 板 A0 口模拟值
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, 主控板 A0 接一个旋钮模块
import time
from pinpong.board import Board,Pin

Board("uno").begin()                # 初始化, 选择板型 (unomicrobit、RPi、handpy) 和端口号,
不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

#adc0 = ADC(Pin(Pin.A0)) # 将 Pin 传入 ADC 中实现模拟输入  模拟输入方法 1
adc0 = Pin(Pin.A0, Pin.ANALOG) # 引脚初始化为电平输出 模拟输入方法 2

while True:
  #v = adc0.read()  # 读取 A0 口模拟信号数值 模拟输入方法 1
  v = adc0.read_analog() # 读取 A0 口模拟信号数值 模拟输入方法 2
  print("A0=", v)
  time.sleep(0.5)
```

3.4 1-04-PWM: 模拟输出

```
# -*- coding: UTF-8 -*-
# 实验效果:  PWM 输出实验, 控制 LED 灯亮度变化
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主板, LED 灯接到 D6 引脚上
import time
from pinpong.board import Board,Pin

Board("uno").begin()                # 初始化, 选择板型 (uno、microbit、RPi、handpy) 和端口
号, 不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

#pwm0 = PWM(Pin(board,Pin.D6)) # 将引脚传入 PWM 初始化  模拟输出方法 1
```

(下页继续)

(续上页)

```
pwm0 = Pin(Pin.D6, Pin.PWM) # 初始化引脚为 PWM 模式 模拟输出方法 2

while True:
    for i in range(255):
        print(i)
        #pwm0.duty(i) #PWM 输出 方法 1
        pwm0.write_analog(i) #PWM 输出 方法 2
        time.sleep(0.05)
```

3.5 1-05-irq: 引脚中断

```
# -*- coding: UTF-8 -*-
# 实验效果: 引脚模拟中断功能测试
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, 主控板 D8 接一个按钮模块

import time
from pinpong.board import Board,Pin

Board("uno").begin()                # 初始化, 选择板型 (uno、leonardo、xugu) 和端口号, 不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

btn = Pin(Pin.D8, Pin.IN)

def btn_rising_handler(pin):# 中断事件回调函数
  print("\n--rising---")
  print("pin = ", pin)

def btn_falling_handler(pin):# 中断事件回调函数
  print("\n--falling---")
  print("pin = ", pin)

def btn_both_handler(pin):# 中断事件回调函数
  print("\n--both---")
  print("pin = ", pin)
```

(下页继续)

(续上页)

```
btn.irq(trigger=Pin.IRQ_FALLING, handler=btn_falling_handler) # 设置中断模式为下降沿触发
#btn.irq(trigger=Pin.IRQ_RISING, handler=btn_rising_handler) # 设置中断模式为上升沿触发，
及回调函数
#btn.irq(trigger=Pin.IRQ_RISING+Pin.IRQ_FALLING, handler=btn_both_handler) # 设置中断模式
为电平变化时触发

while True:
  time.sleep(1) # 保持程序持续运行
```

CHAPTER 4

常用库示例

- 常用库示例中的模块通过 board 库导入

4.1 2-02-servo: 舵机

```
# -*- coding: UTF-8 -*-
# 实验效果: 舵机控制
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, D4 连接一个舵机
import time
from pinpong.board import Board,Pin,Servo # 导入 Servo 库

Board("uno").begin()                # 初始化, 选择板型 (uno、microbit、RPi、handpy) 和端口
号, 不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

s1 = Servo(Pin(Pin.D4)) # 将 Pin 传入 Servo 中初始化舵机引脚

while True:
        #s1.angle(0) # 控制舵机转到 0 度位置 方法 1
        s1.write_angle(0)  # 控制舵机转到 0 度位置 方法 2
```

(下页继续)

(续上页)

```
        print("0")
        time.sleep(1)

        #s1.angle(90) # 控制舵机转到 90 度位置
        s1.write_angle(90)  # 控制舵机转到 90 度位置 方法 2
        print("90")
        time.sleep(1)

        #s1.angle(180) # 控制舵机转到 180 度位置
        s1.write_angle(180)  # 控制舵机转到 180 度位置 方法 2
        print("180")
        time.sleep(1)

        #s1.angle(90) # 控制舵机转到 90 度位置
        s1.write_angle(90)  # 控制舵机转到 90 度位置 方法 2
        print("90")
        time.sleep(1)
```

4.2 2-01-tone: 蜂鸣器

```
# -*- coding: UTF-8 -*-
# 实验效果: 控制蜂鸣器发声
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, 主控板 D8 接一个蜂鸣器模块
import time
from pinpong.board import Board,Pin,Tone # 导入 Tone 类实现控制蜂鸣器

Board("uno").begin()                # 初始化, 选择板型 (uno、leonardo、xugu) 和端口号, 不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

tone = Tone(Pin(Pin.D8)) # 将 Pin 传入 Tone 中实现模拟输出
tone.freq(200) # 按照设置的频率播放

'''
while True:
  tone.tone(200,500)  # 按照设置的频率和时间播放直到完成
```

(下页继续)

(续上页)

```
  time.sleep(1)
'''

while True:
  print("freq=",tone.freq()) # 读取频率并打印
  tone.on()  # 打开蜂鸣器
  time.sleep(1)
  tone.off() # 关闭蜂鸣器
  time.sleep(1)
  tone.freq(tone.freq()+100) # 按照设置的频率播放
```

4.3 2-03-sr04_urm10: 超声波传感器

```
# -*- coding: UTF-8 -*-
# 实验效果：读取超声波
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，使用 SR04 或 URM10 超声波，Trig
接 D7，Echo 接 D8
import time
from pinpong.board import Board,Pin,SR04_URM10 # 中导入 SR04_URM10

Board("uno").begin()               # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()      #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

TRIGER_PIN = Pin(Pin.D7)
ECHO_PIN = Pin(Pin.D8)

sonar = SR04_URM10(TRIGER_PIN,ECHO_PIN)

while True:
  dis = sonar.distance_cm() # 获取距离，单位厘米（cm）
  print("distance = %d cm"%dis)
  time.sleep(0.1)
```

4.4 2-04-dht: 温湿度传感器

```
# -*- coding: UTF-8 -*-
# 实验效果：读取 dht 温湿度传感器
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，dht11 连接 D6，dht22 连接 D7
import time
from pinpong.board import Board,Pin,DHT11,DHT22 # 导入 dht 库

Board("uno").begin()                # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

dht11 = DHT11(Pin(Pin.D6))
dht22 = DHT22(Pin(Pin.D7))

while True:
  temp = dht11.temp_c() # 读取摄氏温度
  humi = dht11.humidity() # 读取湿度
  print("dht11 temperature=",temp," humidity=",humi)

  temp = dht22.temp_c() # 读取摄氏温度
  humi = dht22.humidity() # 读取湿度
  print("dht22 temperature=",temp," humidity=",humi)
  time.sleep(1)
```

4.5 2-07-neopixel:WS2812 灯带

```
# -*- coding: UTF-8 -*-
# 实验效果：控制 WS2812 单线 RGB LED 灯
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，ws2812 灯接到 D9 口

import time
from pinpong.board import Board,Pin,NeoPixel # 导入 neopixel 类

Board("uno").begin()                # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输
入端口号则进行自动识别
```

(下页继续)

(续上页)

```
#Board("uno","COM36").begin()      #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

NEOPIXEL_PIN = Pin(Pin.D9)
PIXELS_NUM = 4 # 灯数

np = NeoPixel(NEOPIXEL_PIN,PIXELS_NUM)

while True:
    np[0] = (0, 255 ,0) # 设置第一个灯 RGB 亮度
    np[1] = (255, 0, 0) # 设置第二个灯 RGB 亮度
    np[2] = (0, 0, 255) # 设置第三个灯 RGB 亮度
    np[3] = (255, 0, 255) # 设置第四个灯 RGB 亮度
    print("color 1")
    time.sleep(1)
    np[1] = (0, 255, 0)
    np[2] = (255, 0, 0)
    np[3] = (255, 255, 0)
    np[0] = (0, 0, 255)
    print("color 2")
    time.sleep(1)
```

CHAPTER 5

扩展库示例

- 扩展库示例中的模块通过 libs 库导入，
- 可通过终端输入 pinpong 查询支持列表和使用方法
- 从安装目录下的 examples 文件夹中可以找到所有示例程序代码

5.1 2-05-lcd1602:1602 显示屏

```
# -*- coding: UTF-8 -*-
# 实验效果：I2C LCD1602 控制
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，LCD1602 显示屏接到 I2C 口 SCL 及
SDA
import time
from pinpong.board import Board
from pinpong.libs.lcd1602 import LCD1602_I2C # 从 libs 中导入 lcd1602_i2c 库

Board("uno").begin()                # 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化
```

(下页继续)

(续上页)

```
lcd = LCD1602_I2C(i2c_addr=0x20) # 初始化 LCD 的 I2C 地址
print("I2C LCD1602 TEST...")

lcd.backlight(True)  # 打开背光
lcd.clear()   # 清屏

lcd.set_cursor(0,0)  # 设置光标位置
lcd.print("Hello World") # 显示 "Hello World",1602 屏像素点少，不能显示汉字
lcd.set_cursor(1,1)  # 设置光标位置
lcd.print(1234) # 显示数字 1234

while True:
    time.sleep(1)
    lcd.scroll_left() # 滚动显示
```

5.2 2-06-oled12864:oled 显示屏

```
# -*- coding: UTF-8 -*-
# 实验效果：I2C OLED2864 屏控制
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，OLED2864 显示屏接到 I2C 口 SCL
及 SDA

import time
from pinpong.board import Board
from pinpong.libs.dfrobot_ssd1306 import SSD1306_I2C # 导入 ssd1306 库

Board("uno").begin()                # 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

oled=SSD1306_I2C(width=128, height=64) # 初始化屏幕，传入屏幕像素点数


while True:
    oled.fill(1) # 全部填充显示
    oled.show() # 显示生效
```

(下页继续)

(续上页)

```
    print("1")
    time.sleep(1)

    oled.fill(0) # 全部填充熄灭，清屏
    oled.show() # 显示生效
    print("0")
    time.sleep(1)

    oled.text("0") # 显示数字
    oled.text("Hello PinPong",8,8) # 指定位置显示文字
    oled.show()  # 显示生效
    time.sleep(2)
```

5.3 3-01-tcs34725: 颜色识别

```
# -*- coding: UTF-8 -*-
# 实验效果：读取 I2C TCS34725 颜色传感器数值
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，TCS34725 颜色传感器接到 I2C 口
SCL SDA

import time
from pinpong.board import Board
from pinpong.libs.dfrobot_tcs34725 import TCS34725 # 从 libs 导入 tcs34725 库

Board("uno").begin()                # 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

tcs = TCS34725() # 传感器初始化

print("Color View Test!");
while True:
  if tcs.begin(): # 查找传感器，读取到则返回 True
    print("Found sensor")
    break # 找到 跳出循环
  else:
```

(下页继续)

(续上页)

```
    print("No TCS34725 found ... check your connections")
    time.sleep(1)

while True:
  r,g,b,c = tcs.get_rgbc() # 获取 rgbc 数据
  print(r,g,b,c)
  print("C=%d\tR=%d\tG=%d\tB=%d\t"%(c,r,g,b))

  '''
  # 数据转换
  r /= c
  g /= c
  b /= c
  r *= 256
  g *= 256
  b *= 256;
  print("------C=%d\tR=%d\tG=%d\tB=%d\t"%(c,r,g,b))
  '''
  time.sleep(1)
```

5.4 3-02-urm09:I2C 超声波

```
# -*- coding: UTF-8 -*-
# 实验效果：读取 I2C 超声波传感器（URM09）数值
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，URM09 传感器接到 I2C 口 SCL SDA
import time
from pinpong.board import Board
from pinpong.libs.dfrobot_urm09 import URM09 # 从 libs 中导入 URM09 库

Board("uno").begin()                # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

urm = URM09(i2c_addr=0x11) # 初始化传感器，设置 I2C 地址
urm.set_mode_range(urm._MEASURE_MODE_AUTOMATIC ,urm._MEASURE_RANG_500) # 设置 URM09 模式为自动检测，最大测量距离 500cm
```

(下页继续)

(续上页)

```
while True:
  dist = urm.distance_cm() # 读取距离数据，单位厘米（cm）
  temp = urm.temp_c() # 读取传感器温度，单位摄氏度（℃）

  print("Distance is %d cm          "%dist)
  print("Temperature is %.2f .c     "%temp)
  time.sleep(0.5)
```

5.5 3-03-rgb1602: 彩色 1602 屏

```
# -*- coding: UTF-8 -*-
# 实验效果：控制 I2C RGB 彩色 LCD1602 液晶屏幕
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，LCD1602 接到 I2C 口 SCL SDA

import time
from pinpong.board import Board
from pinpong.libs.rgb1602 import RGB1602 # 从 libs 导入 rgb1602 库

Board("uno").begin()                # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

lcd = RGB1602()

print("I2C RGB1602 TEST...")

lcd.set_rgb(0,50,0);        # 设置 RGB 值
lcd.set_cursor(0,0)                  # 设置光标到原点
lcd.print("PinPong")        # 显示 "PinPong"

lcd.set_cursor(1,1)         # 设置光标位置
lcd.print(1234)             # 显示数字

while True:
    time.sleep(1)
```

(下页继续)

(续上页)

```
    lcd.scroll_left()           # 滚动显示
```

5.6 3-04-mlx90614: 红外测温

```
# -*- coding: UTF-8 -*-
# 实验效果: 读取 I2C MLX90614 远红外测温传感器
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, 红外测温传感器接到 I2C 口 SCL SDA

import time
from pinpong.board import Board
from pinpong.libs.dfrobot_mlx90614 import MLX90614 # 从 libs 导入 mlx90614 库

Board("uno").begin()                # 初始化, 选择板型 (uno、leonardo、xugu) 和端口号, 不输
入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

irt=MLX90614()

while True:
  print("Object  %s *C"% irt.obj_temp_c())      # 读取物体温度 摄氏度 (℃)
  print("Object  %s *F"% irt.obj_temp_f())      # 读取物体温度 华氏度 ( )
  print("Ambient %s *C"% irt.env_temp_c())      # 读取环境温度 摄氏度 (℃)
  print("Ambient %s *F"% irt.env_temp_f())      # 读取环境温度 华氏度 ( )
  print() # 空行
  time.sleep(1)
```

5.7 3-05-PN532:NFC 近场通讯模块

```
# -*- coding: utf-8 -*-
# 实验效果: NFC 近场通讯模块 IIC 读取卡片信息
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, NFC 近场通讯模块接到 I2C 口 SCL␣
↪SDA
import time
from pinpong.board import Board
```

(下页继续)

(续上页)

```
from pinpong.libs.dfrobot_pn532 import PN532

Board("uno").begin()  # 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()  #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

nfc = PN532()

while not nfc.begin():
  print("initial failure")
  time.sleep(1)
print("Please place the info card/tag on module..... ")

while True:
  if nfc.scan():
    info = nfc.get_information()
    if info != None:
      print("----------------NFC card/tag information----------------------")
      print("UID Lenght: %d"%info.lenght)
      print("UID: %x %x %x %x"%(info.uid[0],info.uid[1],info.uid[2],info.uid[3] ))
      print("AQTA: %x %x"%(info.AQTA[0], info.AQTA[0]))
      print("SAK: 0x%x"%(info.sak))
      print("Type: %s"%(info.types))
      print("Manu facturer: %s"%(info.manu))
      print("RF Technology: %s"%(info.RF))
      print("Memory Size: %d bytes(total)/%d bytes(available)"%(info.size_total, info.
↪size_available))
      print("Block/Page Size: %d bytes"%(info.block))
      print("Number of Blocks/pages: %d"%(info.num_block))
  time.sleep(1)
```

```
# -*- coding: utf-8 -*-
# 实验效果：NFC 近场通讯模块 IIC 读取卡片内信息
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，NFC 近场通讯模块接到 I2C 口 SCL␣
↪SDA
import time
from pinpong.board import Board
from pinpong.libs.dfrobot_pn532 import PN532
```

(下页继续)

(续上页)

```
Board("uno").begin()  # 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()  #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

nfc = PN532()

def print_data(block):
  value = nfc.read_data(block)
  if  value != None:
    for i in value:
      print("0x%x "%(i), end="")
    print("")
  else:
    print_data(block)

while not nfc.begin():
  print("initial failure")
  time.sleep(1)
print("Waiting for a card......")

while True:
  if nfc.scan():
    NFC = nfc.get_information()
    if NFC != None:
      if NFC.lenght == 0x02 or NFC.lenght == 0x04:
        print("----------------Here is the card information to read--------------------")
        for i in range(NFC.num_block):
          if i == 0:
            print("Block %d:UID0-UID3/MANUFACTURER--------->"%(i), end="")
            print_data(i);
          elif (i+1)%4==0 and i < 128:
            print("Block %d:KEYA/ACCESS/KEYB--------------->"%(i), end="")
            print_data(i)
          elif (i+1)%16==0 and i > 127:
            print("Block %d:KEYA/ACCESS/KEYB--------------->"%(i), end="")
            print_data(i)
          else:
            print("Block %d:DATA   ------------------------>"%(i), end="")
```

(下页继续)

(续上页)

```
            print_data(i)
      else:
        print("The card type is not mifareclassic...")
  time.sleep(3)
```

```
# -*- coding: utf-8 -*-
# 实验效果: NFC 近场通讯模块 IIC 读写卡片信息
# 接线: 使用 windows 或 linux 电脑连接一块 arduino 主控板, NFC 近场通讯模块接到 I2C 口 SCL␣
↪SDA
import time
from pinpong.board import Board
from pinpong.libs.dfrobot_pn532 import PN532

Board("uno").begin()  # 初始化, 选择板型和端口号, 不输入端口号则进行自动识别
#Board("uno","COM36").begin()  #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

nfc = PN532()

write_data = "DFRobot NFC"
#write_data = [1, 2, 3, 4, 5, 6, 7, 8, 9]
#write_data = (10, 9, 8, 7, 6, 5, 4, 3, 2, 1)

block_num = 2

while not nfc.begin():
  print("initial failure")
  time.sleep(1)
print("Waiting for a card......")

def parse_data(read_data):
  if read_data != None:
    print("read success! data is ", end=" ")
    print(read_data)
  else:
    print("read failure!")

while True:
```

(下页继续)

(续上页)

```
if nfc.scan():
  info = nfc.get_information()
  if info != None:
    if info.lenght == 0x02 or info.lenght == 0x04:
      if not nfc.write_data(block_num, write_data):
        print("write failure!")
      else:
        print("write success! data is", end=" ")
        print(write_data)
      read_data= nfc.read_data(block_num)
      parse_data(read_data)
    else:
      print("The card type is not mifareclassic...")
time.sleep(2)
```

CHAPTER 6

掌控板示例

- 以下部分为掌控板板载特殊元件控制示例
- 非板载特殊器件（例如 IO、外接传感器等）操作方法同其他所有主控板基础及扩展示例

6.1 掌控板示例-屏幕控制

```
# -*- coding: UTF-8 -*-
# 实验效果：控制掌控板屏幕显示功能
# 接线：使用 windows 或 linux 电脑连接一块掌控板主控板

import time
from pinpong.board import Board
from pinpong.extension.handpy import *

Board("handpy").begin()# 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("handpy","COM36").begin()   #windows 下指定端口初始化
#Board("handpy","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("handpy","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

oled.DispChar('你好世界', 38, 0)          # 先写入缓存区，在 (38,0) 处显示 '你好世界'
oled.DispChar('hello,world', 32, 16)      # 先写入缓存区，在 (32,16) 处显示'hello,world'
oled.DispChar('🅵🅵🅵🅵🅵', 35, 32)          # 先写入缓存区，在 (35,32) 处显示 '🅵🅵🅵🅵🅵'
```

(下页继续)

(续上页)

```
oled.DispChar('こんにちは世界', 24, 48)  # 先写入缓存区，在 (24,48) 处显示 ' こんにちは世界 '
oled.show()                              # 显示画面


''' 其他屏幕控制的方法 '''
#oled.DispChar("PinPong 库",1)    # 屏幕显示"pinpong 库" 在第一行
#oled.DispChar("pinpong 库", 42, 22)   # 屏幕显示"pinpong 库" 在 x,y 坐标处，x:0-127,y:0-
↪63
#oled.Bitmap(0,0,50,50,"E:\\PinPong\\default.png")  # 依次是显示的坐标 X,Y，显示的宽和高，
图片路径
#oled.chear(1)   # 屏幕清除第一行的内容，参数 1,2,3,4
#oled.fill(0)    # 清屏黑色填 0，白色填 1
#oled.rotation(0)   # 屏幕旋转 0° 或者 180°
#oled.pixel(0,0)    # 在坐标 x,y 画点
#oled.set_line_width(1)  # 设置线宽范围 1 - 128
#oled.line(0,0,127,63)  # 划线，依次是起点坐标 x1,y1 和终点坐标 x2,y2
#oled.circle(63, 31, 20)    # 画圆，依次是坐标 x, y 和 半径，不填充
#oled.fill_circle(63, 31, 20)   # 画圆，依次是坐标 x, y 和 半径，填充
#oled.rect(0,0, 63, 31)   # 画矩形，依次是起点坐标 x, y, 宽，高，不填充
#oled.fill_rect(0,0, 63, 31)    # 画矩形，依次是起点坐标 x, y, 宽，高，填充
#oled.show()    # 显示生效，所有屏幕操作执行完成后调用 show 才会执行
```

6.2 掌控板示例-读取传感器

```
# -*- coding: UTF-8 -*-
# 实验效果：读取掌控板板载传感器功能，通过终端窗口查看数据
# 接线：使用 windows 或 linux 电脑连接一块掌控板主控板

import time
from pinpong.board import Board
from pinpong.extension.handpy import *

Board("handpy").begin()# 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("handpy","COM36").begin()   #windows 下指定端口初始化
#Board("handpy","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("handpy","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

while True:
```

(下页继续)

(续上页)

```
    print(button_a.value())                    # 按键 A 是否按下
    print(button_b.value())                    # 按键 B 是否按下
    print(button_ab.value())                   # 按键 AB 是否按下
    #  print(touchPad_P.is_touched())                              # 是否触摸 P
    #  print(touchPad_Y.is_touched())                              # 是否触摸 Y
    #  print(touchPad_T.is_touched())                              # 是否触摸 T
    #  print(touchPad_H.is_touched())                              # 是否触摸 H
    #  print(touchPad_O.is_touched())                              # 是否触摸 O
    #  print(touchPad_N.is_touched())                              # 是否触摸 N
    #  touch_threshold("all",60)                  # 设置按键 P/Y/T/H/O/N 的触摸阈值，all
代表全部
    #  print(touchPad_P.read())                       # 读取按键 P 的触摸值
    #  print(touchPad_Y.read())                       # 读取按键 Y 的触摸值
    #  print(touchPad_T.read())                       # 读取按键 T 的触摸值
    #  print(touchPad_H.read())                       # 读取按键 H 的触摸值
    #  print(touchPad_O.read())                       # 读取按键 O 的触摸值
    #  print(touchPad_N.read())                       # 读取按键 N 的触摸值
    print(sound.read())                        # 读取麦克风强度
    print(light.read())                        # 读取环境光强度
    #  print(accelerometer.get_x())                   # 读取加速度 X 的值
    #  print(accelerometer.get_y())                   # 读取加速度 Y 的值
    #  print(accelerometer.get_z())                   # 读取加速度 Z 的值
    #  print(accelerometer.get_strength())            # 读取加速度的强度
    print("--------------------")
    time.sleep(0.4)
```

6.3 掌控板示例-RGB 灯控制

```
# -*- coding: UTF-8 -*-
# 实验效果：控制掌控板板载 RGB 灯
# 接线：使用 windows 或 linux 电脑连接一块掌控板主控板

import time
from pinpong.board import Board
from pinpong.extension.handpy import *

Board("handpy").begin()# 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("handpy","COM36").begin()   #windows 下指定端口初始化
```

(下页继续)

(续上页)

```
#Board("handpy","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("handpy","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

rgb[0] = (255, 0, 0)  # 设置为红色，全亮度
rgb[1] = (0, 128, 0)  # 设定为绿色，一半亮度
rgb[2] = (0, 0, 64)   # 设置为蓝色，四分之一亮度
rgb.write()
```

6.4 掌控板示例-音乐播放

```
# -*- coding: UTF-8 -*-
# 实验效果：控制掌控板蜂鸣器播放音调
# 接线：使用 windows 或 linux 电脑连接一块掌控板主控板

import time
from pinpong.board import Board
from pinpong.extension.handpy import *

Board("handpy").begin()# 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("handpy","COM36").begin()   #windows 下指定端口初始化
#Board("handpy","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("handpy","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
        "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
music.play(tune)                # 播放自编乐谱
```

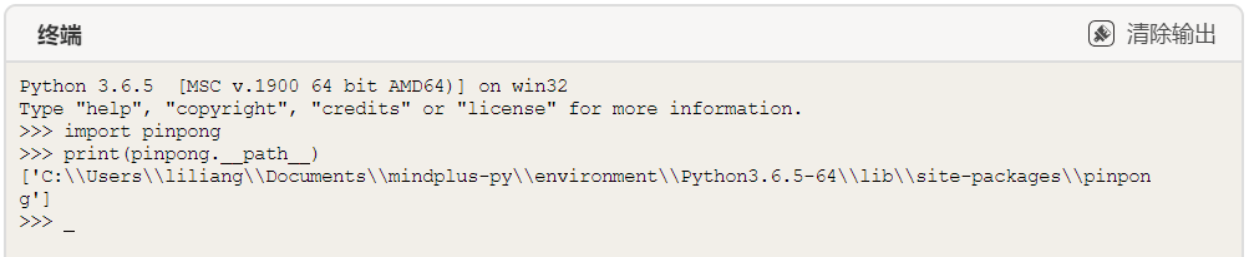
CHAPTER 7

更多示例

pinpong 库安装目录下有更多示例程序，可以通过如下方法找到.

1. 进入 Python 终端，然后依次输入如下代码即可查看 Python 库所在的目录（site-packages）

```
import pinpong
print(pinpong.__path__)
```



2. 打开对应路径的文件夹，其中 examples 文件夹下即为所有内置的示例程序

注意： 内置 examples 文件夹下的案例仅供参考使用，不排除后续会变更，最终以本文档网页中列出的示例为准。


0-此电脑 › 文档 › mindplus-py › environment › Python3.6.5-64 › Lib › site-packages › pinpong
名称
修改日期
类型
大小
__pycache__ 2021/4/7 10:40 文件夹
base 2021/4/7 10:38 文件夹
examples 2021/4/7 10:38 文件夹
extension 2021/4/7 10:38 文件夹
libs 2021/4/7 10:38 文件夹
__init__.py 2021/4/7 10:38 PY 文件 0 KB
board.py 2021/4/7 10:38 PY 文件 42 KB

CHAPTER 8

进阶教程

8.1 项目前置知识

8.1.1 概述

本系列教程是基于 win10 操作系统下，使用 Python3 通过 pinpong 库来控制 Arduino UNO 板来实现项目功能，在本节中我们将了解以下内容：

一、什么是 Arduino?

二、什么是 Python?

三、如何编辑 Python 代码?

四、什么是 pinpong 库?

8.1.2 一、什么是 Arduino？

Arduino 是一个开放源码电子原型平台，拥有灵活、易用的硬件和软件。Arduino 专为设计师，工艺美术人员，业余爱好者，以及对开发互动装置或互动式开发环境感兴趣的人而设的。

Arduino 可以接收来自各种传感器的输入信号从而检测出运行环境，并通过控制光源，电机以及其他驱动器来影响其周围环境。板上的微控制器编程使用 Arduino 编程语言（基于 Wiring）和 Arduino 开发环境（以 Processing 为基础）。Arduino 可以独立运行，也可以与计算机上运行的软件（例如，Flash，Processing，MaxMSP）进行通信。Arduino 开发 IDE 接口基于开放源代码，可以让您免费下载使用开发出更多令人惊艳的互动作品。

Arduino 是人们连接各种任务的粘合剂。要给 Arduino 下一个最准确的定义，最好用一些实例来描述。

您想当咖啡煮好时，咖啡壶就发出“吱吱”声提醒您吗?

您想当邮箱有新邮件时，电话会发出警报通知您吗?

想要一件闪闪发光的绒毛玩具吗?

想要一款具备语音和酒水配送功能的 X 教授蒸汽朋克风格轮椅吗?

想要一套按下快捷键就可以进行实验测试蜂音器吗?

想为您的儿子自制一个《银河战士》手臂炮吗?

想自制一个心率监测器，将每次骑脚踏车的记录存进存储卡吗?

想过自制一个能在地面上绘图，能在雪中驰骋的机器人吗?

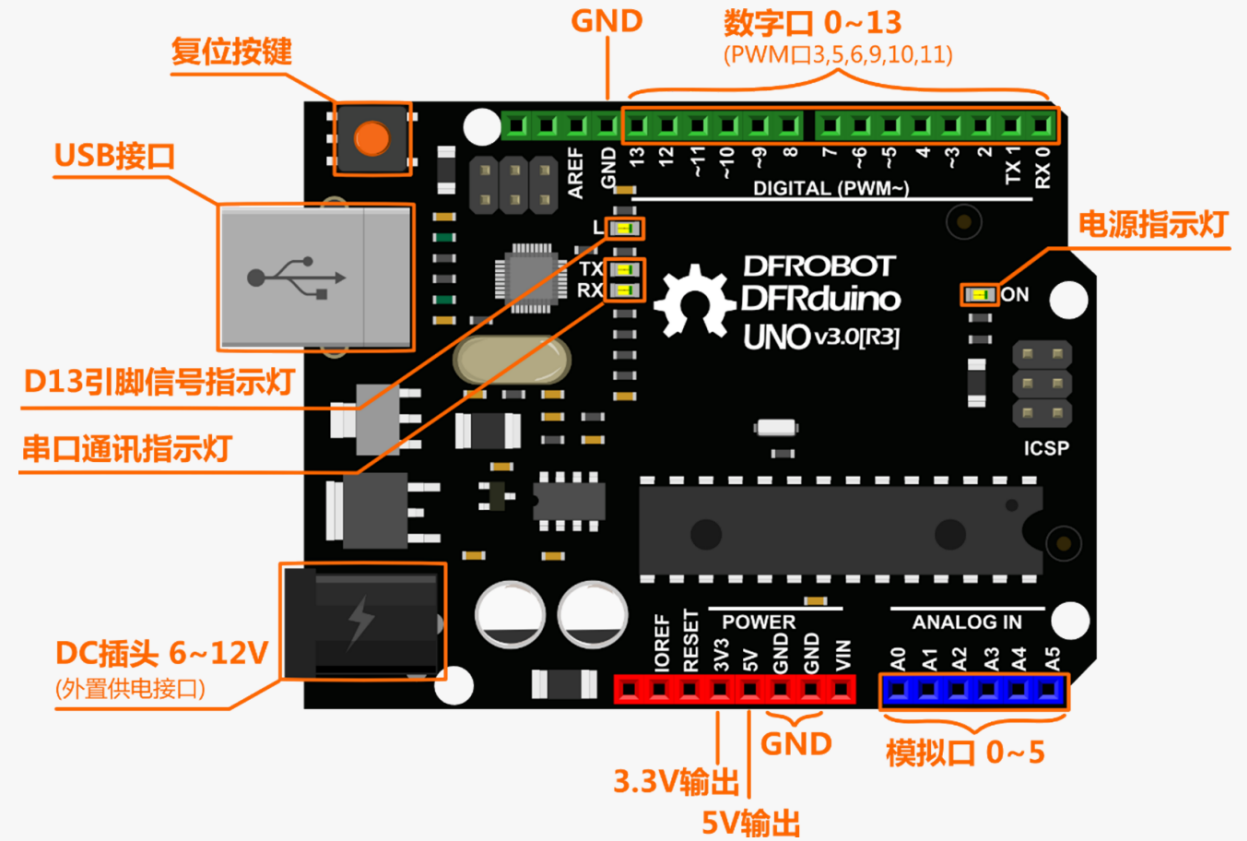
Arduino 都可以为您实现。

1.1 认识 Arduino UNO

Arduino 系列有众多的型号，其中最为经典的就是 Arduino UNO 了。在后续项目中我们也将以 Arduino UNO 为主。

先来简单的看下 Arduino UNO。下图中有标识的部分为常用部分。图中标出的数字口和模拟口，即为常说的 I/O。数字口有 0~13，模拟口有 0~5。

除了最重要的 I/O 口外，还有电源部分。UNO 可以通过两种方式供电方式，一种通过 USB 供电，另一种是通过外接 6~12V 的 DC 电源。除此之外，还有 4 个 LED 灯和复位按键，稍微说下 4 个 LED。ON 是电源指示灯，通电就会亮了。L 是接在数字口 13 上的一个 LED，在下节的项目中会进行教学的。TX、RX 是串口通讯指示灯，比如我们在下载程序的过程中，这两个灯就会不停闪烁。



8.1.3 二、什么是 Python？

Python 是一种跨平台的计算机程序设计语言。是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本 (shell)，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。

Python 是一种解释型脚本语言，可以应用于以下领域：

Web 和 Internet 开发

科学计算和统计

人工智能

桌面界面开发

软件开发

后端开发

网络爬虫

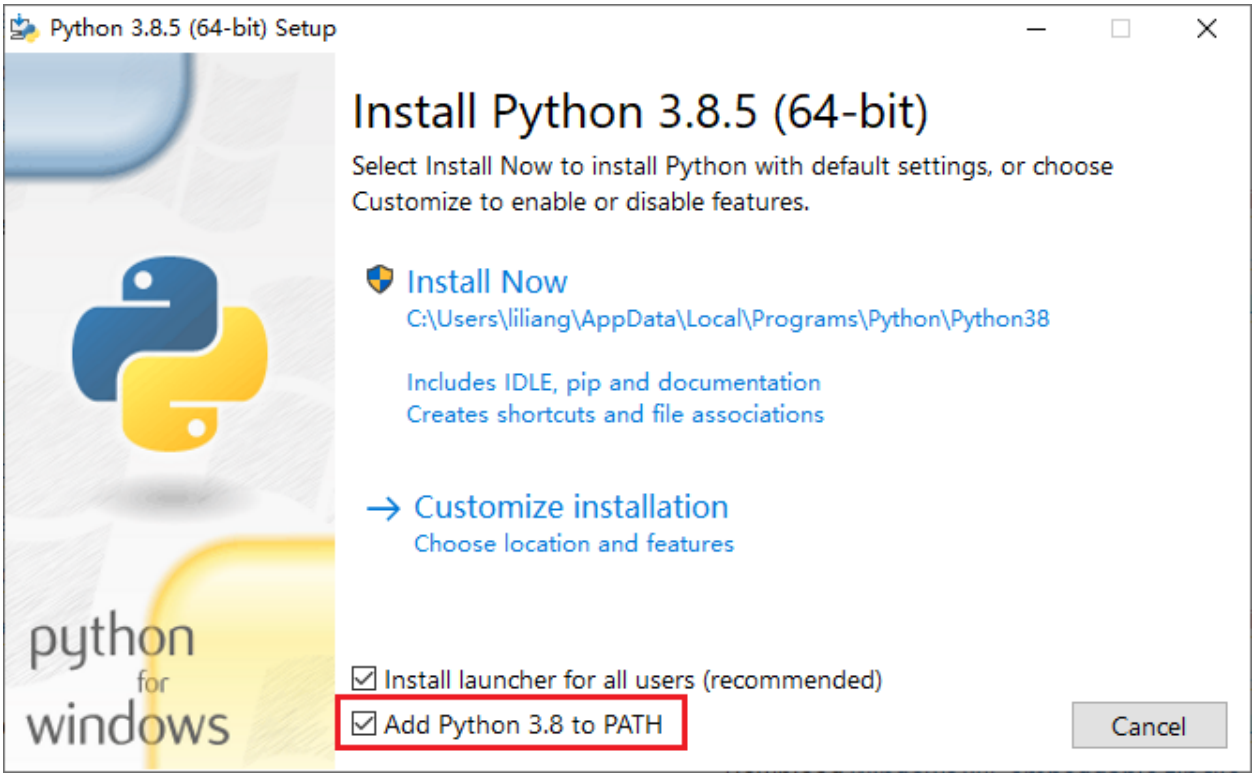
8.1.4 三、如何编辑 Python 代码？

可以用很多方法来编辑 Python 代码，甚至你可以使用文本编辑器编辑代码。在项目中我们将主要使用 Python 安装后自带的 IDLE 来编辑代码。

3.1 如何安装 Python

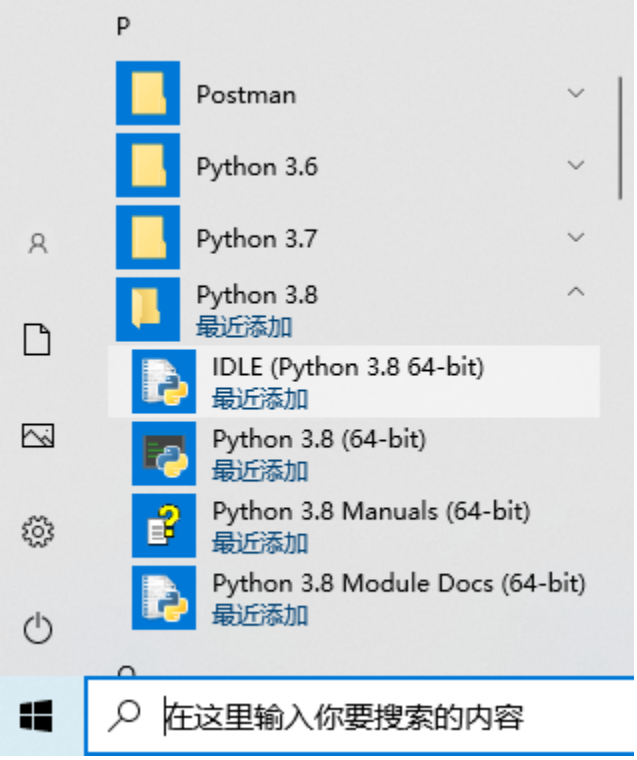
1、首先打开 Python 的下载链接（Python 下载 ）找到【Download Windows x86-64 executable installer】下载 Python 安装包，项目中使用到的 Python 版本为 3.8.5。选择适合的版本下载安装即可。

2、安装 Python，注意在安装时勾选 add Python x.x to path，然后点击 Install Now 进行安装。（如果安装时未勾选，可以重新安装时勾选）

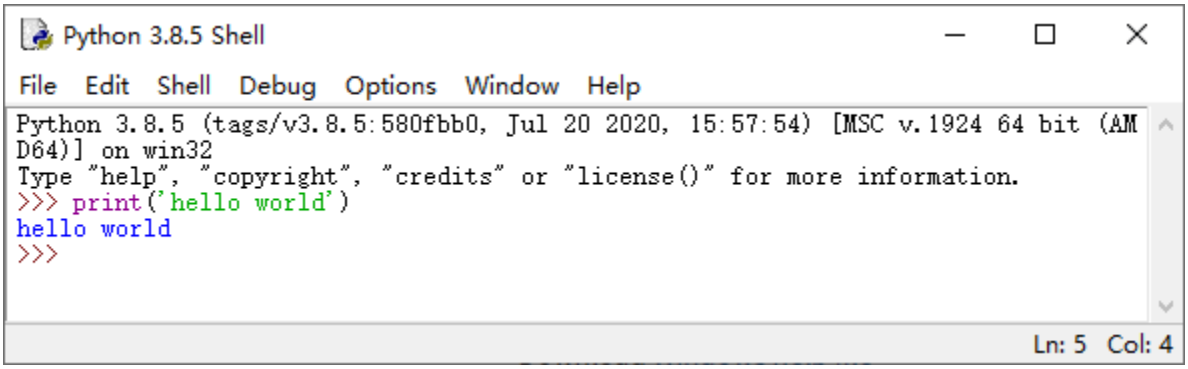


3.2 如何编辑 Python 代码？

安装好 Python 后，我们就可以在开始菜单中找到 Python 了，点击打开 IDLE。

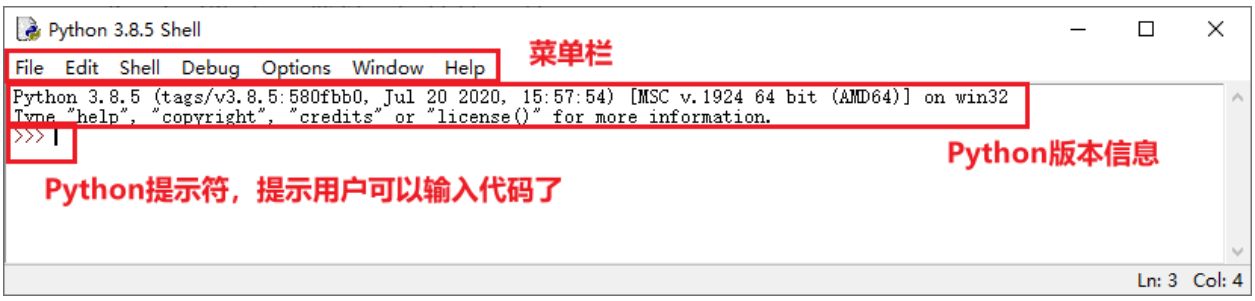


输入 print（‘hello world’）摁下回车，查看结果。



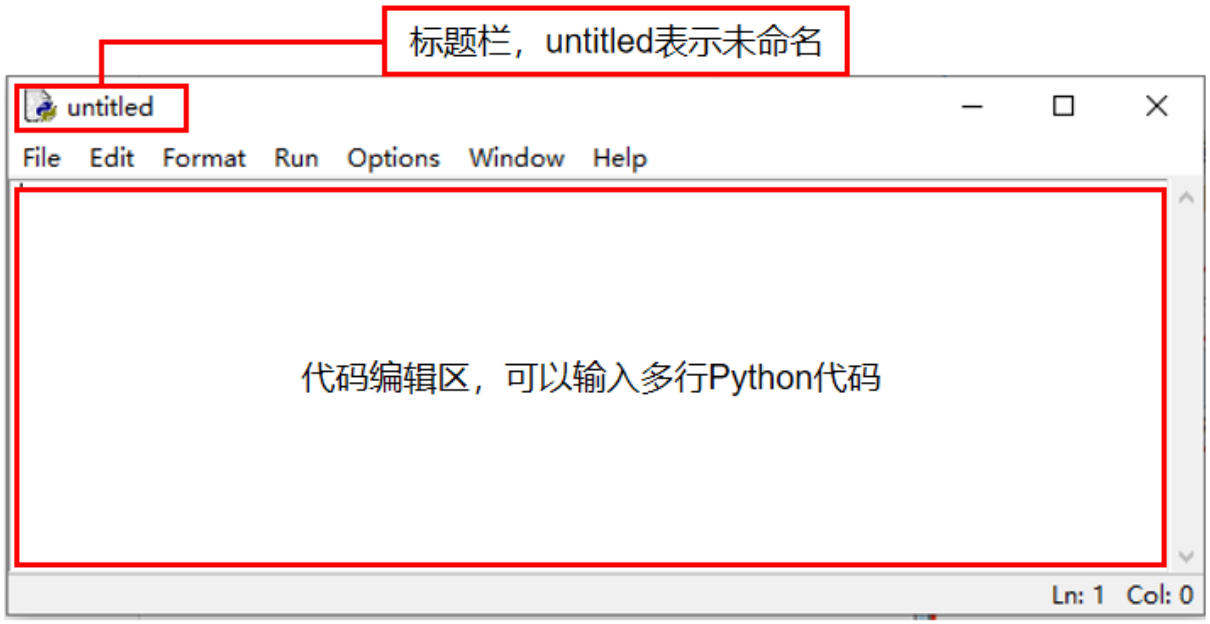
3.2 如何编辑 Python 代码？

1、打开 Python IDLE 窗口如图所示，我们可以看到窗口，这是 python shell 窗口。

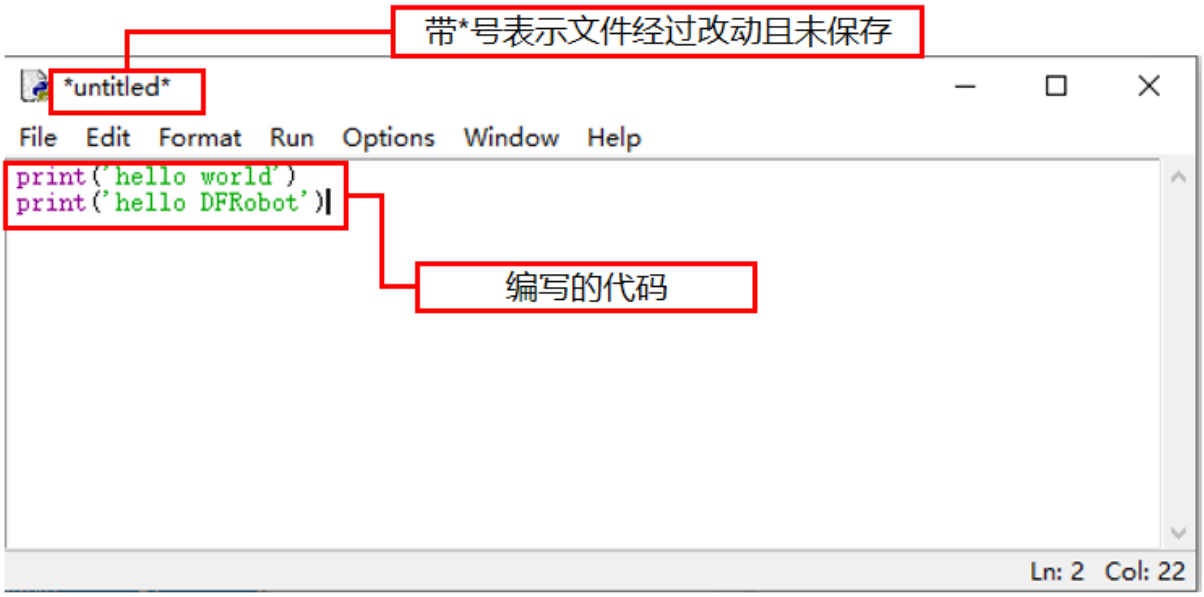


前面我们已经在 IDLE 中运行过简单语句了，但是在实际开发中，通常不能只包含一行代码，当需要编写多行代码时，就需要单独创建一个文件保存这些代码，在全部编写完成后一起执行，这里就需要新建一个文档

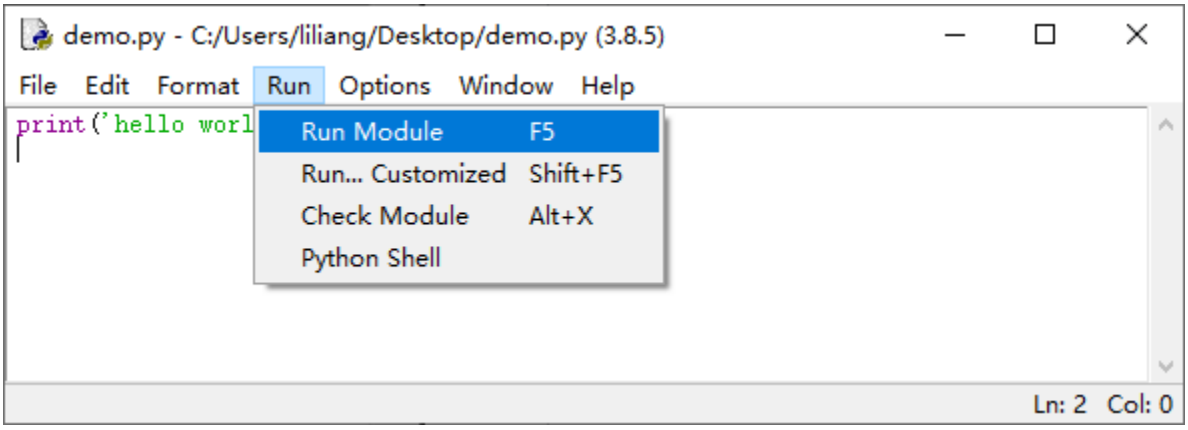
2、在 IDLE 主窗口的菜单栏上，选择“File -> New File”菜单项，将打开一个新窗口，在该窗口中，可以直接编写 Python 代码。在输入一行代码后再按下 <Enter> 键，将自动换到下一行，等待继续输入



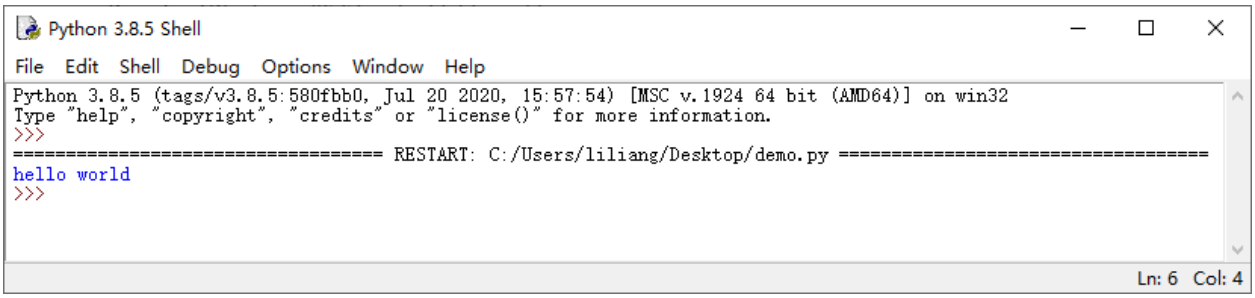
3、在代码区输入代码，我们输入 hello world 的指令



4、保存代码，点击菜单“Run Module”或按键盘上的“F5”键运行程序。如果是带 * 号的未保存状态执行运行功能的话，软件也会提示你保存后才可以运行。



5、运行成功，运行结果还是会返回 python shell 窗口反馈结果。



IDLE 的使用可以参考官方文档：官方文档

8.1.5 四、什么是 pinpong 库？

PinPong 库是一套控制开源硬件主控板的 Pyhton 库，基于 Firmata 协议并兼容 MicroPython 语法，5 分钟即可让你上手使用 Python 控制 Arduino。

借助于 PinPong 库，直接用 Python 代码就能给各种常见的开源硬件编程。其原理是给开源硬件烧录一个特定的固件，使开源硬件可以通过串口与电脑通讯，执行各种命令。

PinPong 库的名称由“Pin”和“Pong”组成，“Pin”指引脚，“PinPong”为“乒乓球”的谐音，指信号的往复。

pinpong 库的设计，是为了让开发者在开发过程中不用被繁杂的硬件型号束缚，而将重点转移到软件的实现。哪怕程序编写初期用 Arduino 开发，部署时改成了掌控板，只要修改一下硬件的参数就能正常运行，实现了“一次编写处处运行”。

使用 pinpong 库可以结合 Python 丰富的扩展库资源来驱动 Arduino。

附：

如何安装 pinpong 库?

在 windows 系统上使用快捷键 win+R 输入 cmd，在弹出的小黑窗中输入 pip install pinpong 即可完成安装。

在 pinpong 官方文档 点击打开 中可以查看详细安装教程。

8.2 项目 1 LED 闪烁

8.2.1 一、概述

Hello World 是所有编程语言学习的第一课，但是在 Arduino 学习中，我们的 Hello World 叫做 Blink。Blink 是什么意思呢？闪烁。其实 Blink 就是点亮 Arduino 机上的板载 LED 灯，并让它闪烁的一个程序。

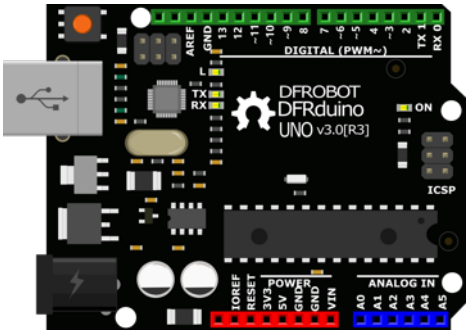
8.2.2 二、项目实施

（1）点亮板载 LED 灯

硬件准备：

主控：Arduino UNO

连接线：TypeAtoB 方口 USB 连接线



软件准备：

本教程项目使用操作系统为 Win10，Python 版本为 3.8.5，编译器使用 Python 自带 IDLE 编辑器，Arduino 板使用 pinpong 库驱动。

1、打开 pinpong 官方文档，找到基础库示例中的“blink”，并用 IDLE 打开。

2、blink.py 示例程序，右键选择 Edit with IDLE→Edit with IDLE3.8.5 即可使用 IDLE 打开 Python 程序编辑内容。

程序编写：

将 Arduino Uno 主板通过 USB 连接到电脑后，无需选择端口，在运行程序时会自动识别端口，如识别失败则可以采用“指定端口初始化”的形式手动指定 COM 口。

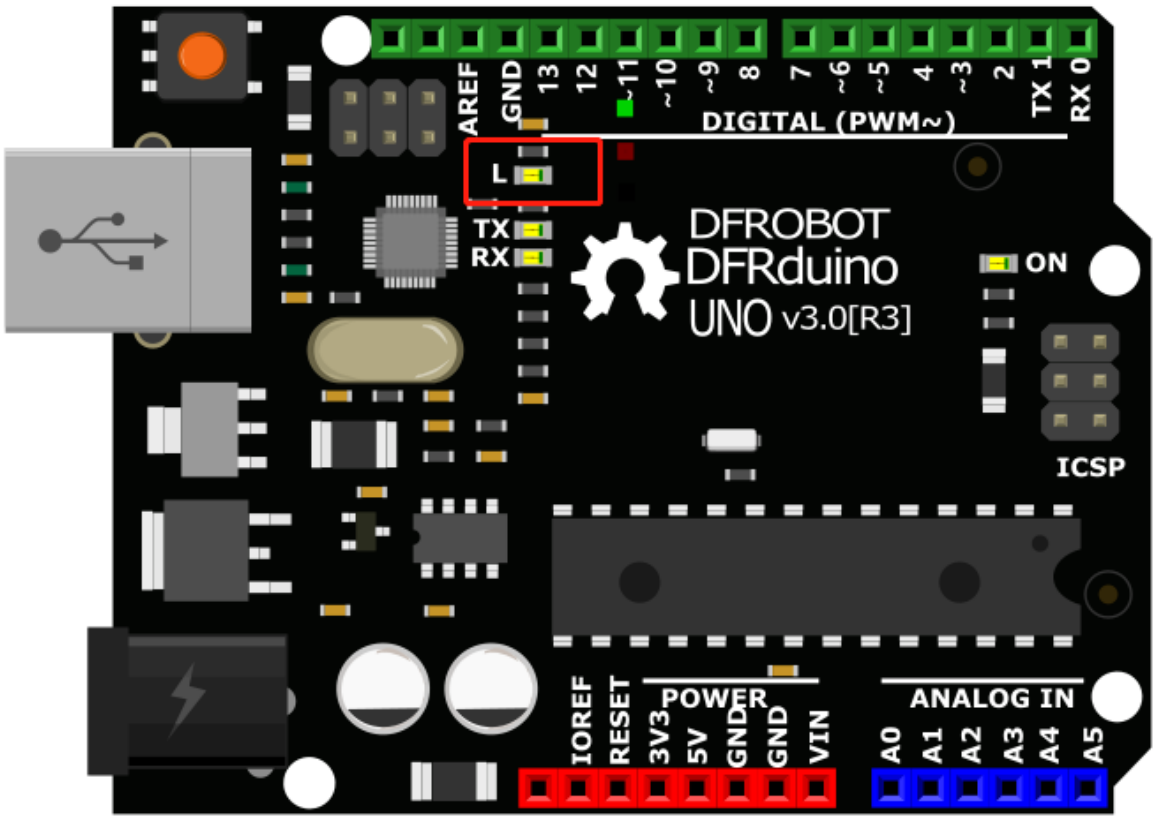
注：第一次使用 pinpong 库会自动给 Arduino 主板烧录 Firmata 固件。所以不需要专门给 Arduino 上传固件。

按下 F5 运行程序，会提示保存程序，点击确定保存即可。运行成功后会弹出一个新的窗口，等待程序运行如图所示即为成功。

查看效果。Arduino Uno 上的板载 LED 会按照亮 1 秒，熄灭 1 秒的规律闪烁。（如果此步执行没有成功可以查看官方文档中的常见问题）

注意：在程序运行时不可以拔掉与 Arduino 连接的 USB 线，且不能关闭新弹出的运行窗口，如果拔线或者关闭运行窗口，程序功能就会停止执行。

运行效果



红圈中的 LED 灯会按照亮 1 秒，熄灭 1 秒的规律闪烁。

1、如果我们想要修改小灯闪烁的频率，应该怎么修改程序？只需要修改代码中延时部分的时长即可。如图所示修改后的效果为小灯点亮 2 秒，熄灭 1 秒。

```
# 实验效果：控制 arduino UNO 板载 LED 灯一秒闪烁一次
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板
import time
from pinpong.board import Board,Pin
Board("uno").begin()                  # 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()         #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化
led = Pin(Pin.D13, Pin.OUT) # 引脚初始化为电平输出
while True:
  led.write_digital(1) # 输出高电平
  time.sleep(2) # 等待 2 秒 保持状态
  led.write_digital(0) # 输出低电平
  time.sleep(1) # 等待 1 秒 保持状态
```

（2）点亮外接 LED 灯

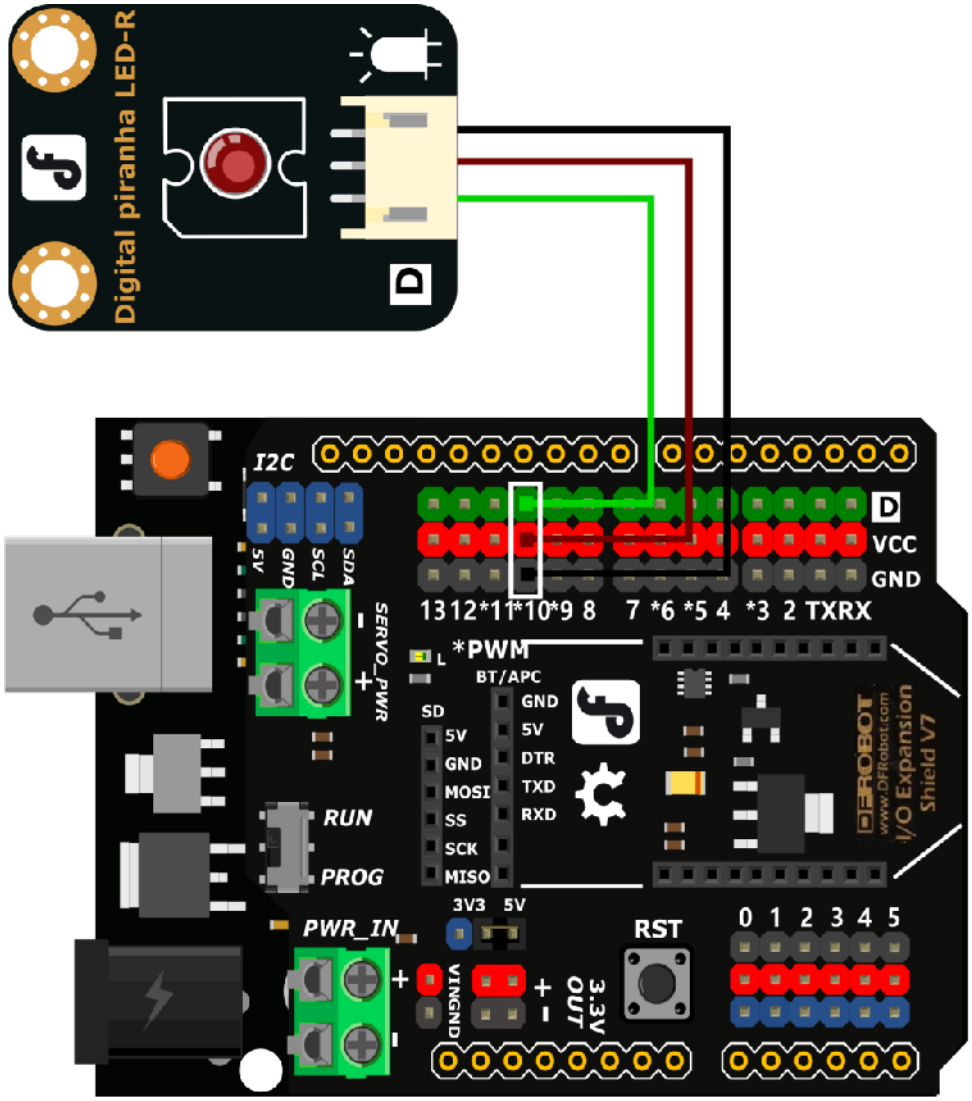
硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块

连接线：TypeAtoB 方口 USB 连接线

硬件连接图：



- 将扩展板接入 Arduino Uno 主控板上，组装时注意对准后在压入，以防压弯插针。
- 将 LED 模块接入 10 号数字引脚。

程序编写：

1、我们在接线的时候将外接 LED 接到了 10 号引脚，所以我们需要修改程序才可以点亮这颗外接的小灯，将 led 对应的引脚修改为 10。

```
# -*- coding: utf-8 -*-

#实验效果：控制arduino UNO让外接的LED灯一秒闪烁一次
#接线：使用windows或linux电脑连接一块arduino主控板
import time
from pinpong.board import Board,Pin

Board("uno").begin()                #初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()       #windows下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()    #mac下指定端口初始化

led = Pin(Pin.D10, Pin.OUT) #引脚初始化为电平输出

while True:
  led.write_digital(1) #输出高电平
  time.sleep(1) #等待1秒 保持状态

  led.write_digital(0) #输出低电平
  time.sleep(1) #等待1秒 保持状态
```

2、如果想要小灯有一个呼吸灯的效果应该如何执行？找到示例程序中的 PWM 模拟输出功能，并用 Python IDLE 运行。

3、修改引脚号与端口号，然后运行程序，小灯就会循环执行 LED 灯渐渐变亮，到最亮时熄灭的命令。

示例程序

```
# -*- coding: utf-8 -*-
# 实验效果：小灯会循环执行渐渐变亮，到最亮时熄灭的命令
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，主控板 D6 接一个 LED 灯模块
import time
from pinpong.board import Board,Pin,PWM # 导入 PWM 类实现模拟输出

Board("uno").begin()  # 初始化，选择板型和端口号，不输入端口号则进行自动识别
#Board("uno","COM36").begin()  #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

pwm0 = PWM(Pin(Pin.D6)) # 将 Pin 传入 PWM 中实现模拟输出

while True:
for i in range(255): # 从 0 到 255 循环
    pwm0.duty(i)  # 设置模拟输出值
    print(i)
    time.sleep(0.05)
```

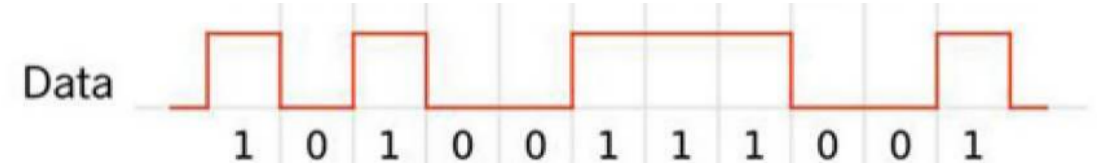
8.2.3 三、代码分析

数字信号与模拟信号

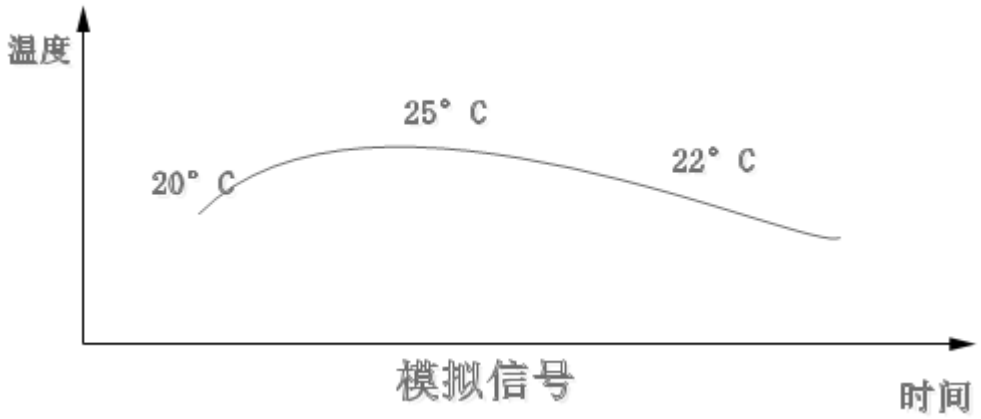
你知道什么是数字信号什么是模拟信号吗？让我们先看看数字信号与模拟信号的概念吧。

资料阅读：

> 数字信号：数字信号指自变量是离散的、因变量也是离散的信号，这种信号的自变量用整数表示，因变量用有限数字中的一个数字来表示。在计算机中，数字信号的大小常用有限位的二进制数表示。



> 模拟信号：模拟信号是指用连续变化的物理量表示的信息，其信号的幅度，或频率，或相位随时间作连续变化，或在一段连续的时间间隔内，其代表信息的特征量可以在任意瞬间呈现为任意数值的信号。



光看概念感觉特别抽象，那么我们用生活中的实例来理解，比如我们平时用到的用于开关灯的开关，开和关是两个状态，非开即关。那么对灯来说开和关就是数字信号。再想想家中如果有一个温度计的话，温度变化是一个连续变化的的数值，并不能用某个特殊的状态来表示，温度的变化就是模拟信号。

在理解了数字信号与模拟信号之后，思考一下，在本项目中控制 LED 灯亮灭的信号是数字信号还是模拟信号?

8.2.4 四、硬件分析

不知道同学们有没有注意到，我们在第二个呼吸灯案例的时候，LED 灯是接在 10 号数字引脚上的，但是按照数字信号和模拟信号的概念来看，应该是模拟信号才能实现呼吸灯的效果。这里就需要我们了解一个新的知识点，PWM 信号。

观察我们手中的 Arduino UNO 主控板的数字引脚上，是不是有些引脚号旁边有 * 标记（有些板子是波浪号~）这些引脚就是支持 PWM 信号输出的引脚。

PWM（Pulse width modulation，中文名脉冲宽度调制），脉冲宽度调制是一种模拟控制方式。在 Arduino 中 pwm 是不断的做高低电平切换模拟出一种近似模拟量的输出的效果来实现变化的。但是这里仅仅得到了近似模拟值输出的效果，如果要输出真正的模拟值，还需要在模拟引脚上执行。

8.3 项目 2 神奇的按键

8.3.1 一、概述

按钮开关，也称作按键开关，早期也称作敏感型开关，广泛应用在灯，插座总开关，门铃，汽车中控台等。按键开关的出现给用电安全增加了一层保护膜，方便控制电器的同时也进一步保护了元器件。

在 Arduino 的学习过程中，我们会接触各种各样的输入设备，其中，按键开关是最简单也是应用最广泛的一种。这里我们将用 Arduino 控制 LED 灯，实现按键按下开，再按下关的效果。

8.3.2 二、项目实施

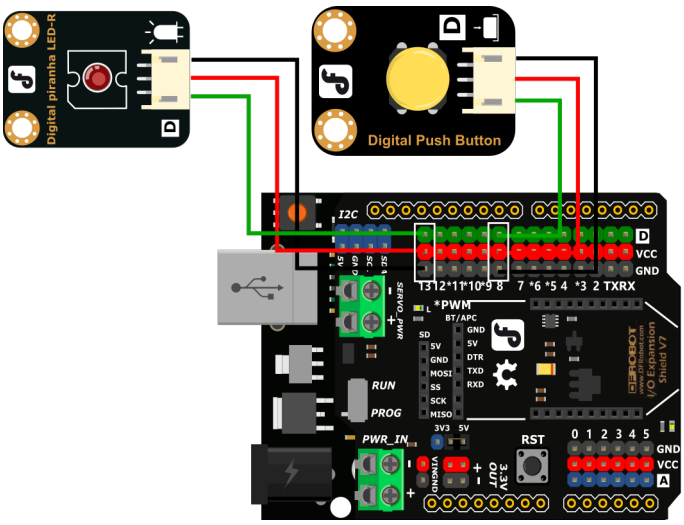
（1）使用按钮点亮小灯

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块、按钮模块

连接线：TypeAtoB 方口 USB 连接线



- 将 LED 发光模块接入 13 号数字引脚，将按钮模块接入 8 号数字引脚

程序编写：

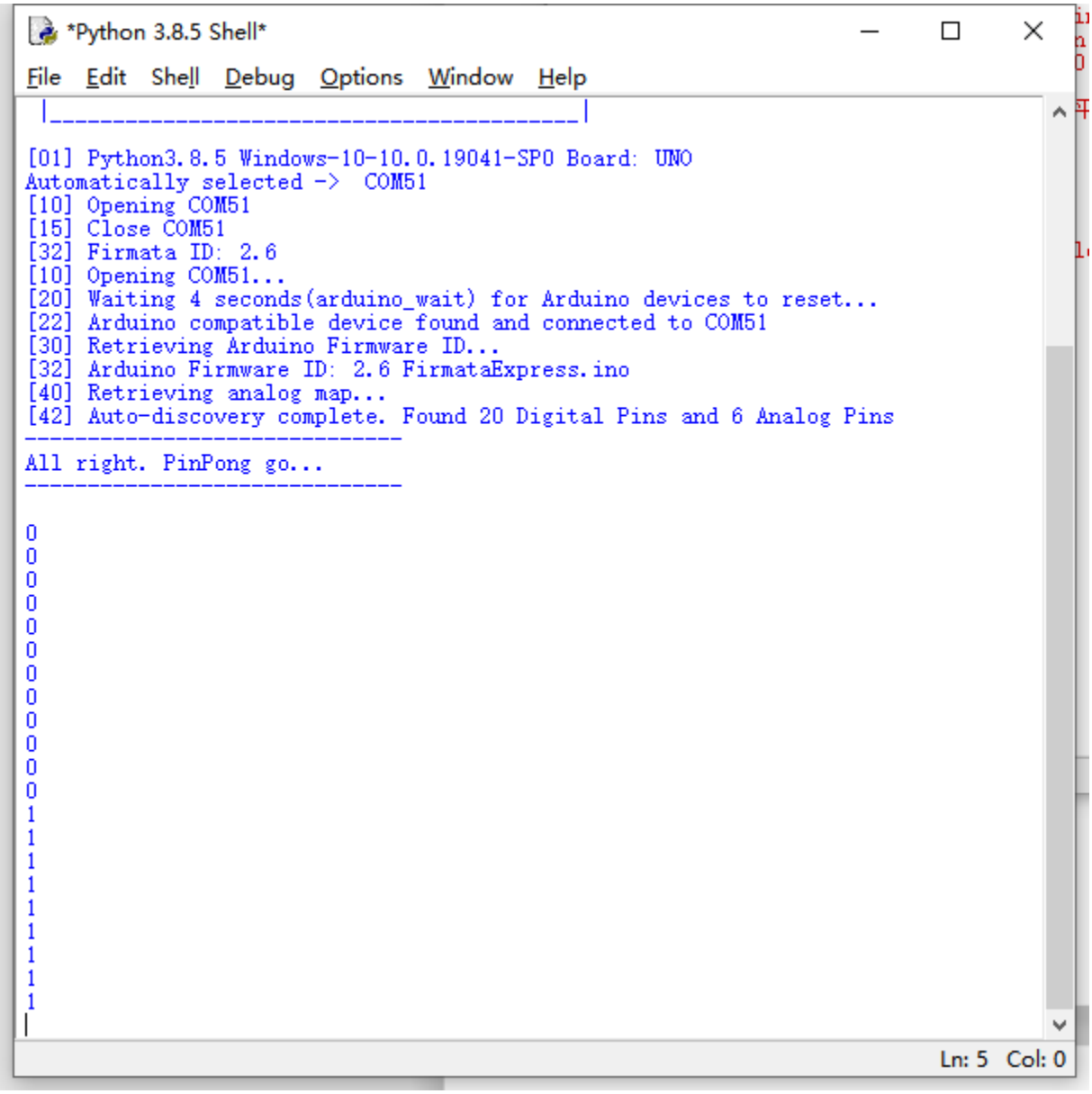
1、打开 pingpong 库的官方文档，找到基础库示例中的“数字输入”，并用 IDLE 打开。

1button.py - C:/Users/咪嗷喵/Desktop/1button.py (3.8.5)

File Edit Format Run Options Window Help

```
# -*- coding: utf-8 -*-

#实验效果：使用按钮控制arduino UNO控制外接的led灯
#接线：使用windows或linux电脑连接一块arduino主控板，主控板D8接一个按钮模块
import time
from pinpong.board import Board,Pin

board = Board("uno").begin()  #初始化，选择板型和端口号，不输入端口号则进行自动识别
#board = Board("uno","COM36").begin()  #windows下指定端口初始化
#board = Board("uno","/dev/ttyACM0").begin()   #linux下指定端口初始化
#board = Board("uno","/dev/cu.usbmodem14101").begin()   #mac下指定端口初始化

btn = Pin(Pin.D8, Pin.IN) #引脚初始化为电平输入
led = Pin(Pin.D13, Pin.OUT)

while True:
  v = btn.read_digital()  #读取引脚电平
  print(v)  #终端打印读取的电平状态
  led.write_digital(v)  #将按钮状态设置给led灯引脚
  time.sleep(0.1)
```

2、按下 F5 运行程序，查看效果。当按下按钮时板载的 LED 灯会亮起（并打印 1），松开会熄灭（打印 0）。



注意：在程序运行时不可以拔掉与 Arduino 连接的 USB 线，且不能关闭新弹出的 Python shell 运行窗口，如果拔线或者关闭运行窗口，程序功能就会停止执行。

（2）按钮开关灯

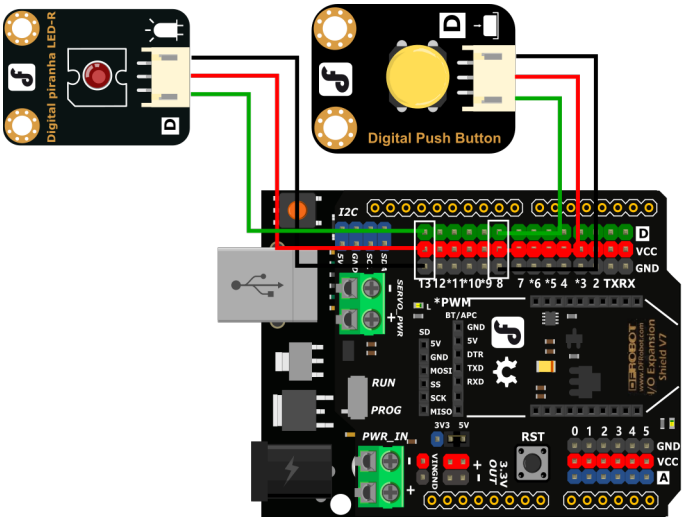
在上步完成的功能中，我们通过按键可以实现“按下按键-LED 亮”，“松开按键-LED 灭”，但实际运用的开关却是“首次按下打开，再次按下关闭”，我们将在本步中实现这个功能。

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块、按钮模块

连接线：TypeAtoB 方口 USB 连接线



- 将 LED 发光模块接入 13 号数字引脚，将按钮模块接入 8 号数字引脚

程序编写：

```
import time
from pinpong.board import Board,Pin

board = Board("uno").begin()  # 初始化，选择板型和端口号，不输入端口号则进行自动识别
#board = Board("uno","COM36").begin()  #windows 下指定端口初始化
#board = Board("uno","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#board = Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

btn = Pin(Pin.D8, Pin.IN) # 引脚初始化为电平输入
led = Pin(Pin.D13, Pin.OUT)
i=0  # 设置变量 i=0

while True:
v = btn.read_digital()  # 读取引脚电平
#print(v)  # 终端打印读取的电平状态
if (v == 1):
    if (i == 1):
    i=0
    led.write_digital(0)  # 将按钮状态设置给 led 灯引脚
    print("LED off")
    else:
    i=1
    led.write_digital(1)  # 将按钮状态设置给 led 灯引脚
```

(下页继续)

(续上页)

```
    print("LED on")


    time.sleep(0.5)
```

运行代码，摁下按钮可以切换 LED 灯的亮灭。

8.3.3 三、代码分析

1、导入必要的包和初始化设置。

```
import time
from pinpong.board import Board,Pin
```

(下页继续)

(续上页)

```
board = Board("uno").begin()  # 初始化，选择板型和端口号，不输入端口号则进行自动识别
#board = Board("uno","COM36").begin()  #windows 下指定端口初始化
#board = Board("uno","/dev/ttyACM0").begin()   #linux 下指定端口初始化
#board = Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

btn = Pin(Pin.D8, Pin.IN) # 引脚初始化为电平输入
led = Pin(Pin.D13, Pin.OUT)# 引脚初始化为电平输出
```

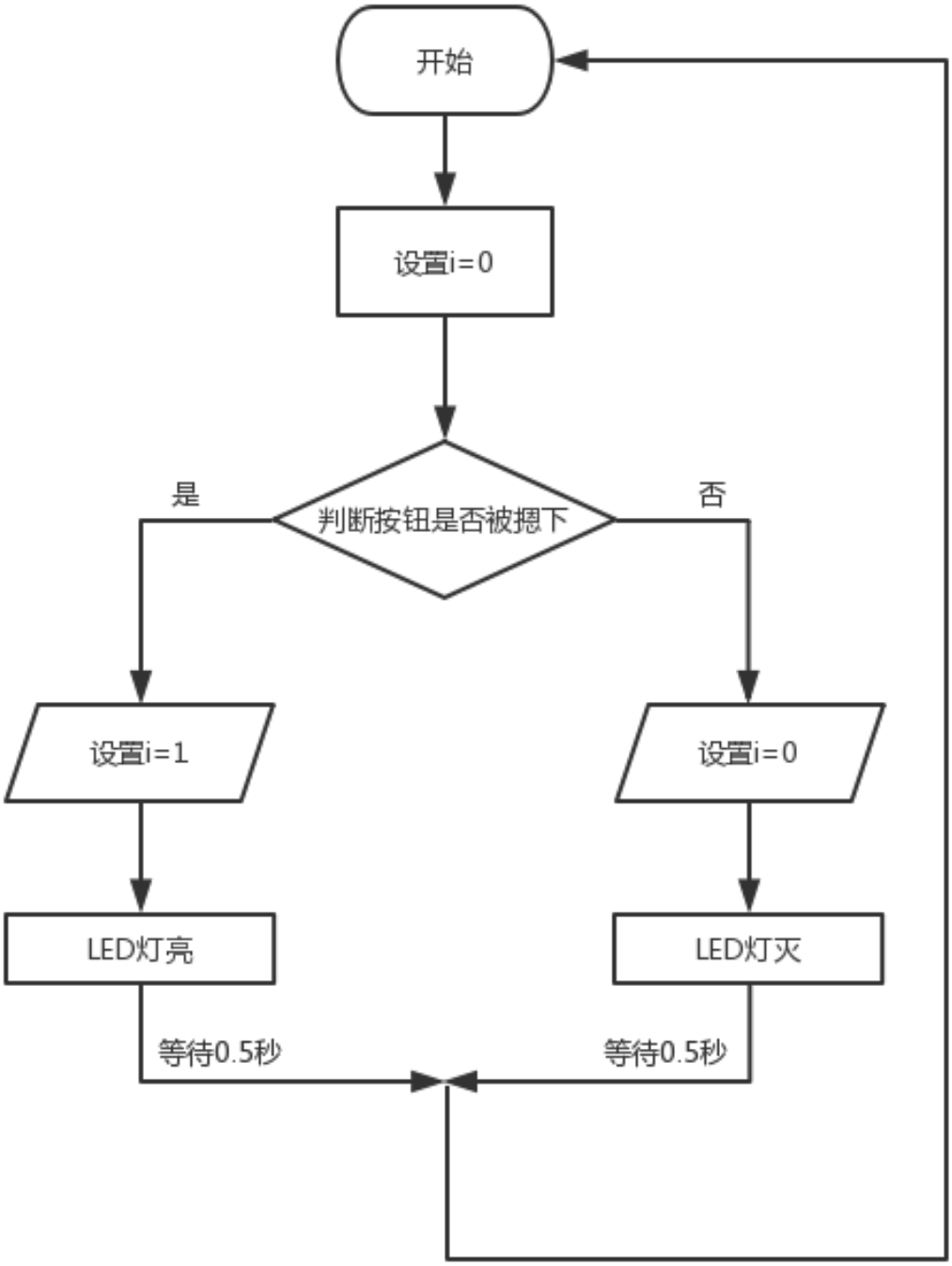
2、我们需要一个按键能够得到两个不同的结果，那么就需要设置一个中间值来实现切换状态的作用。所以首先我们要设置一个变量 i=0。

```
i = 0 # 设置变量 i=0
```

3、接着加入判断，判断按钮摁下的状态。为了方便设置，定义变量 v 为按钮的状态。

```
while True:
    v = btn.read_digital()  # 读取引脚电平
    #print(v)  # 终端打印读取的电平状态
    if (v == 1):
```

4、然后加入判断，借助 i 来区别状态，如逻辑图所示，每次摁下按钮时 i 的值会在 0 和 1 之间切换，借助 i 值的变化来确定 LED 灯的亮灭状态即可，根据逻辑完成代码。




```
if (i == 1):
    i=0
```

(下页继续)

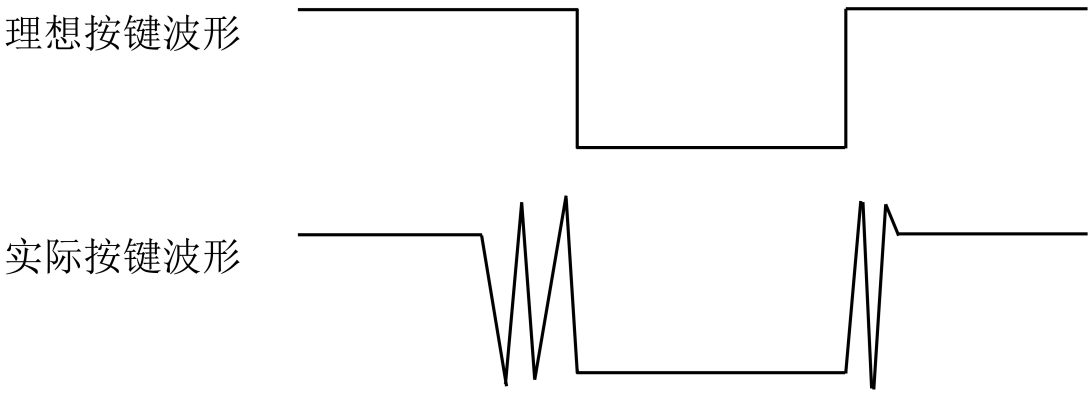
(续上页)

```
    led.write_digital(0)  # 将按钮状态设置给 led 灯引脚
    print("LED off")
    else:
    i=1
    led.write_digital(1)  # 将按钮状态设置给 led 灯引脚
    print("LED on")

    time.sleep(0.5)
```

什么是按键抖动？

我们想象的开关电路是“按下按键-立刻导通”“再次按下-立刻断开”，而实际上并非如此。

按键通常采用机械弹性开关，而机械弹性开关在机械触点断开闭合的瞬间（通常 10ms 左右），会由于弹性作用产生一系列的抖动，造成按键开关在闭合时不会立刻稳定的接通电路，在断开时也不会瞬时彻底断开。



那又如何消除按键抖动呢？

常用除抖动方法有两种：软件方法和硬件方法。这里重点讲讲方便简单的软件方法。

我们已经知道弹性惯性产生的抖动时间为 10ms 左右，用延时命令推迟命令执行的时间就可以达到除抖动的效果。

所以我们在代码中加入了 0.5 秒的延时以实现按键防抖的功能。

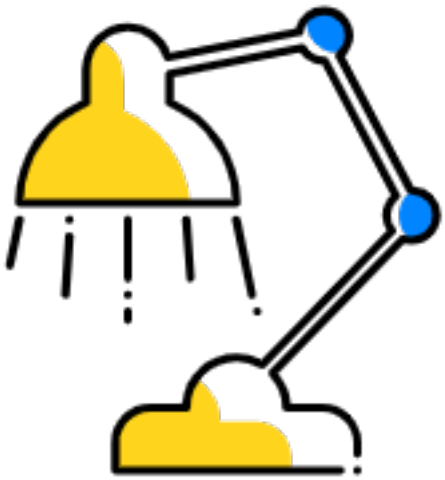
除抖动后效果

8.4 项目 3 调光台灯

8.4.1 一、概述

在前面的课程中我们学会了如何点亮并控制 LED 灯，那 LED 灯的亮度我们可以调节吗？如果输出的值不是数字量，而是模拟量，是不是就可以实现通过数值来精确控制灯光亮度了。

在上节课的设计中，我们使用的是按钮控制灯的亮灭，按钮只有两个状态，是没办法输出模拟值的，所以本项目我们就需要使用到旋钮来作为灯光亮度的输入。



8.4.2 二、项目实施

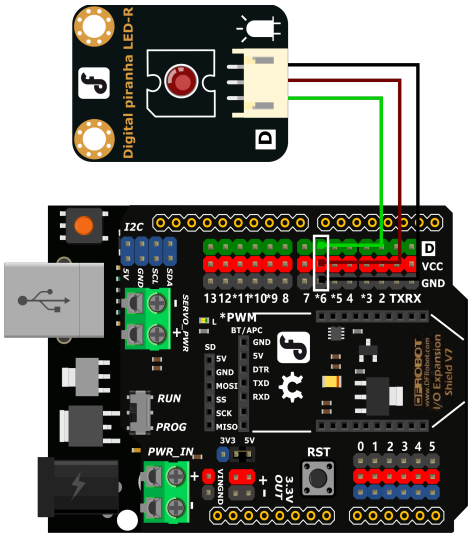
（1）渐变灯光

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块

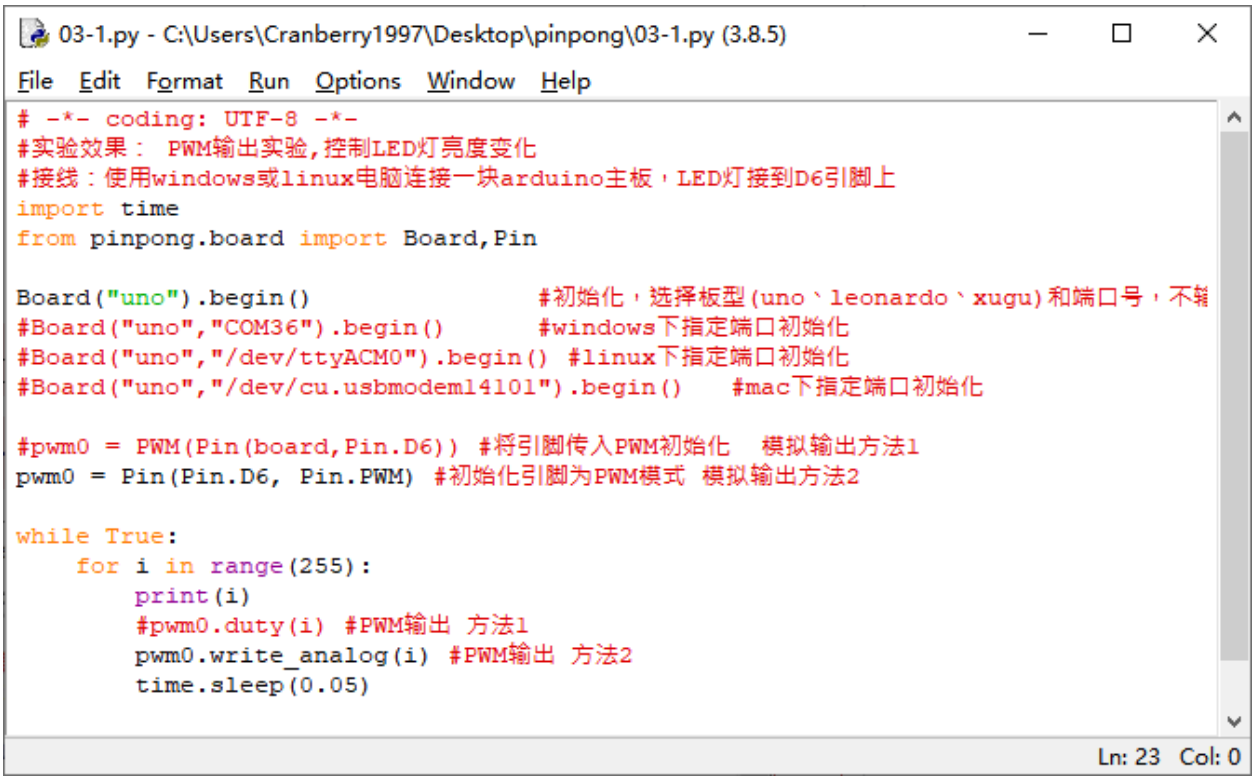
连接线：TypeAtoB 方口 USB 连接线



- 将 LED 发光模块接入 6 号数字引脚

程序编写：

1. 打开 pingpong 库的官方文档，找到基础库示例中的“模拟输出”，并用 IDLE 打开。



2. 摁下 F5 运行程序，查看效果。LED 灯会逐渐变亮，当到最亮时熄灭再慢慢变亮，循环变化。

（2）旋钮调光

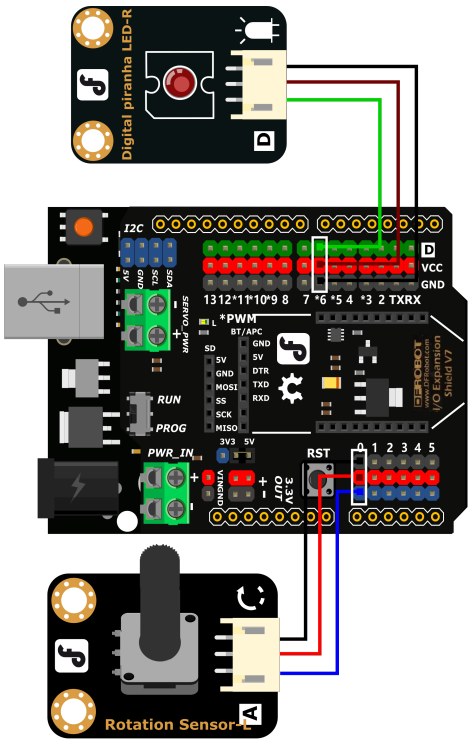
在上步完成的功能中, 我们旋钮来精确的控制 LED 灯的亮度。

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块、旋钮模块

连接线：TypeAtoB 方口 USB 连接线



- 将 LED 发光模块接入 6 号数字引脚，将旋钮模块接入 A0 模拟引脚

程序编写：

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()                    # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别

pwm0 = Pin(Pin.D6, Pin.PWM) # 初始化引脚为 PWM 模式 模拟输出方法 2
adc0 = Pin(Pin.A0, Pin.ANALOG) # 引脚初始化为电平输出

while True:
    v = adc0.read_analog() # 读取 A0 口模拟信号数值
    L = int(v*255/1024)
    pwm0.write_analog(L) #PWM 输出
    print("A0=",L)
```

运行代码，旋转旋钮 LED 灯会慢慢亮起和熄灭。

8.4.3 三、代码分析

1. 导入必要的包和初始化设置。

```
import time
from pinpong.board import Board,Pin
Board("uno").begin() # 初始化，选择板型和端口号，不输入则留空进行自动识别
```

(下页继续)

(续上页)

```
pwm0 = Pin(Pin.D6, Pin.PWM)
adc0 = Pin(Pin.A0, Pin.ANALOG) # 将 Pin 传入 ADC 中实现模拟输入
```

2. 我们需要将旋钮输出的模拟值转换为 LED 灯亮度的 PWM 值，为了得到整数这里用了整型功能 int。

```
v = adc0.read_analog() # 读取 A0 口模拟信号数值
L = int(v*255/1024)
```

3. 接着让 LED 灯根据我们转换过的数值亮起。

```
pwm0.write_analog(L) #PWM 输出
print("A0=",L)
```

数据类型-整数类型

在我们做运算的时候，传感器读取到的数值经过 *180/1024 运算的结果往往不会是一个整数，常常会带着长长的小数尾数。我们试着不加 int 整形得到的结果如下：

```
1.93359375
3.33984375
1.23046875
0.52734375
28.65234375
4.04296875
1.23046875
9.4921875
6.50390625
0.0
11.42578125
11.77734375
19.3359375
5.44921875
2.98828125
13.18359375
0.0
10.546875
```

这样的数值我们是无法直接使用的，所以就需要使用到整形 int 将数据取整。

8.5 项目 4 智能节能灯

8.5.1 一、概述

在前面的章节我们学习了如何点亮小灯，如何调节亮度等等功能，那像楼道灯那样，只会在黑夜且发出声音时才会亮起的灯是如何实现的呢？我们将在本项目中学习。

8.5.2 二、项目实施

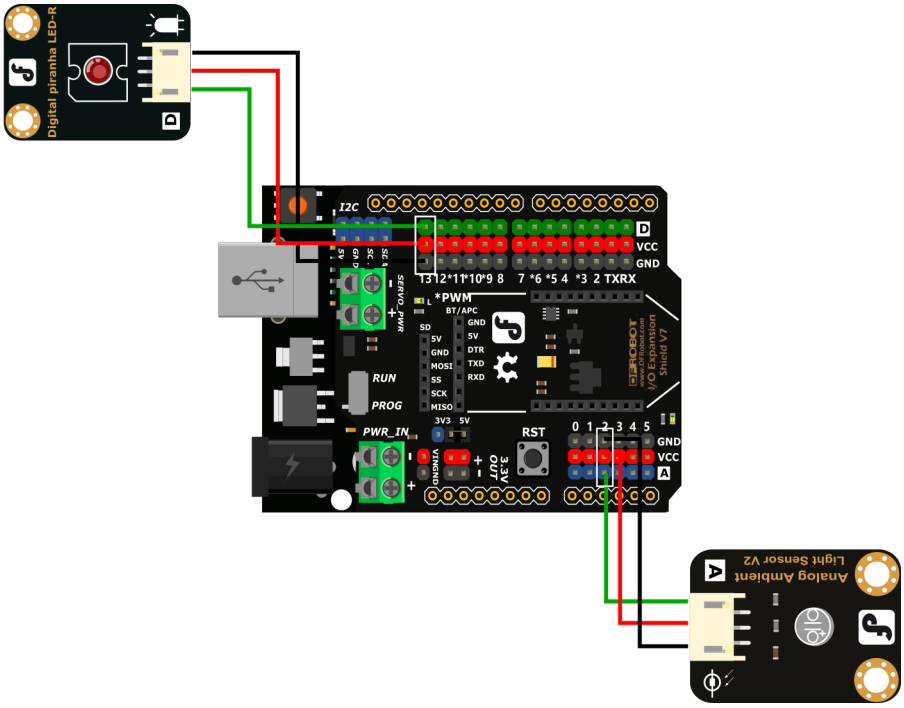
（1）通过光线亮度控制小灯

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块、光线传感器

连接线：TypeAtoB 方口 USB 连接线



- 将 LED 模块接入 D13 引脚, 光线传感器接入 A2 引脚

程序编写：

1. 打开 pingpong 库的官方文档，找到基础库示例中的“模拟输入”，并用 IDLE 打开。

2. 修改代码，当亮度低于 100 的时候，点亮 LED 灯。

```
import time
from pinpong.board import Board,Pin
Board("uno").begin()
Light = Pin(Pin.A2,Pin.ANALOG)
LED = Pin(Pin.D13, Pin.OUT)
while True:
    vl=Light.read_analog()
    print("Light=",vl)
    if vl < 100:
        LED.write_digital(1)
    else:
        LED.write_digital(0)
```

3. 摁下 F5 运行程序，查看效果。当亮度低于 100（遮住光线传感器）的时候 LED 灯会亮起，恢复后 LED 灯会熄灭。

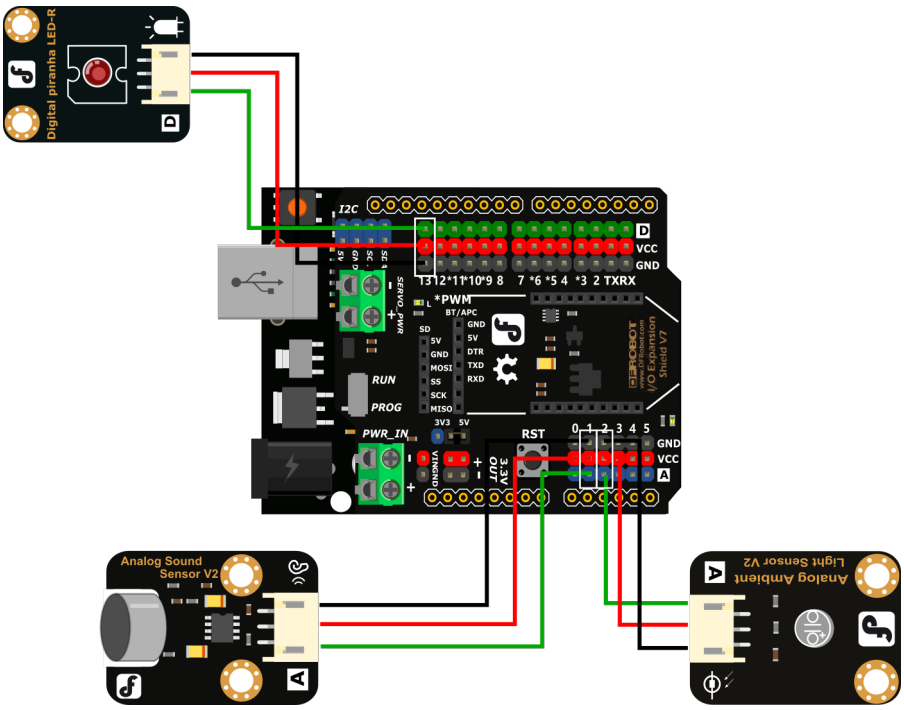
（2）加入声音传感器检测

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LED 发光模块、光线传感器、声音传感器

连接线：TypeAtoB 方口 USB 连接线



- 将 LED 模块接入 D13 引脚，光线传感器接到 A2 引脚，声音传感器接到 A1 引脚。

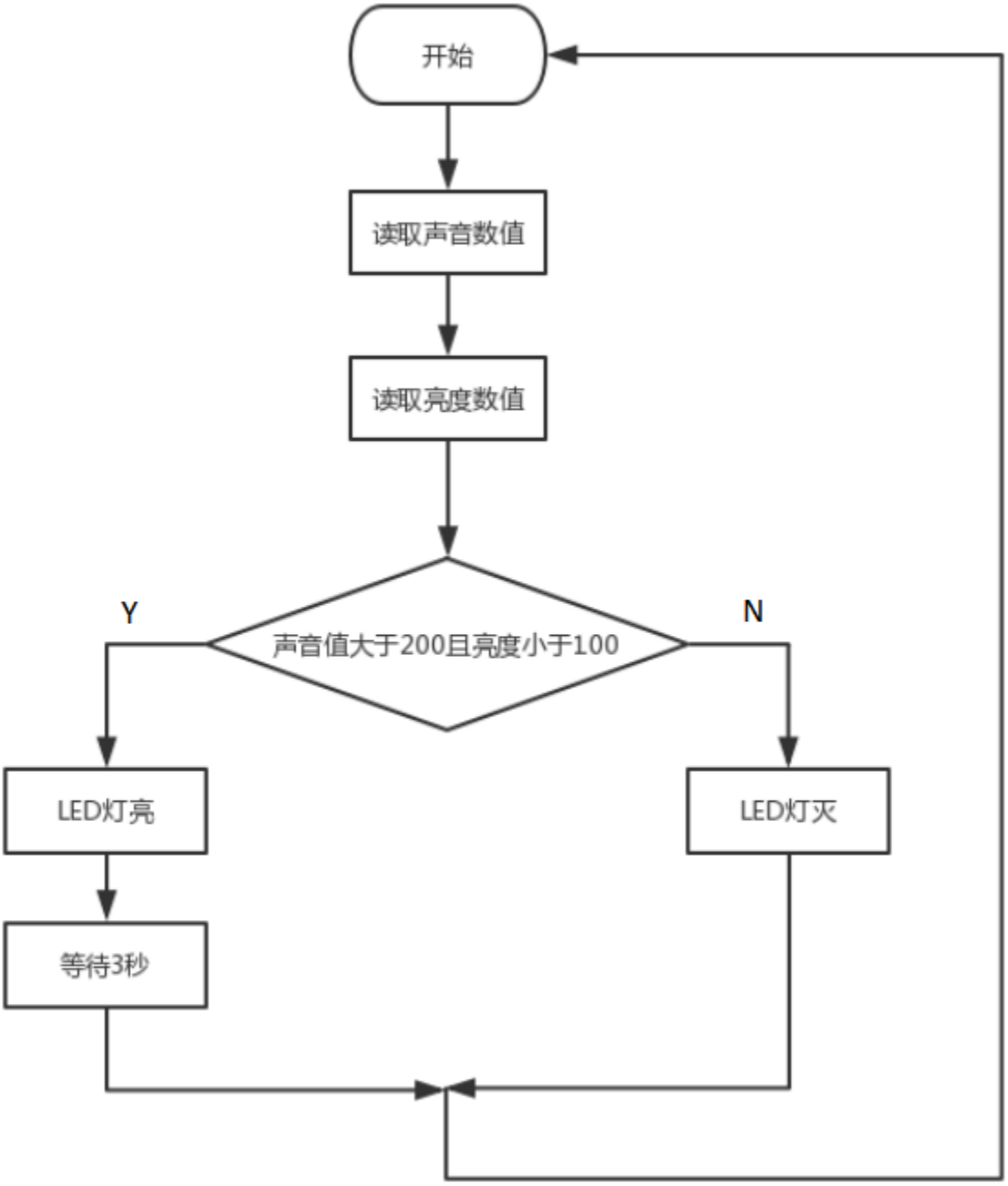
程序编写

```
import time
from pinpong.board import Board,Pin # 导入必要的库函数

Board("uno").begin()# 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输入端口号则进行自动识别

Sound = Pin(Pin.A1,Pin.ANALOG) # 初始化声音传感器引脚为 A1，检测声音大小
Light = Pin(Pin.A2,Pin.ANALOG)# 初始化光线传感器引脚为 A2，检测光线强度
LED = Pin(Pin.D13, Pin.OUT) # 初始化 LED 引脚为 D13
while True:
    vs=Sound.read_analog()# 读取模拟声音信号数值
    vl=Light.read_analog()# 读取模拟灯光信号数值
    print("Sound=",vs,"Light=",vl)# 打印声音和光线数据
    if vs>200 and vl < 100:# 判断光线和声音大小
        LED.write_digital(1)
        time.sleep(3)
    else:
        LED.write_digital(0)
```

运行代码，当亮度值低于 100 且声音值大于 200 时（遮住光线传感器且发出声音时），LED 灯会点亮，当没有声音或光线传感器没有被遮住的状态维持 3 秒以上时，LED 灯会熄灭。

8.5.3 三、代码分析

1. 设计程序逻辑图



2. 导入必要的包和初始化设置。

```
import time
from pinpong.board import Board,Pin # 导入必要的库函数

Board("uno").begin()# 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别
```

(下页继续)

(续上页)

```
Sound = Pin(Pin.A1,Pin.ANALOG) # 初始化声音传感器引脚为 A1，检测声音大小
Light = Pin(Pin.A2,Pin.ANALOG)# 初始化光线传感器引脚为 A2，检测光线强度
LED = Pin(Pin.D13, Pin.OUT) # 初始化 LED 引脚为 D13
```

3. 加入判断

```
while True:
    vs=Sound.read_analog()# 读取模拟声音信号数值
    vl=Light.read_analog()# 读取模拟灯光信号数值
    print("Sound=",vs,"Light=",vl)# 打印声音和光线数据
    if vs>200 and vl < 100:# 判断光线和声音大小
        LED.write_digital(1)
        time.sleep(3)
    else:
        LED.write_digital(0)
```

如何进行多条件判断

在本项目中我们需要判断两个条件，在我们做逻辑判断的时候经常会需要进行多条件判断，有些时候是需要两个条件都要满足的，有些时候是两个条件满足任意一条就可以了。在这种情况下，我们应该如何编写程序呢?

两条条件都需要满足的情况下，我们可以这样写

```
if 条件 A and 条件 B
```

两条条件满足任一即可的情况下，我们可以这样写

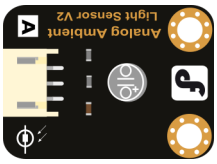
```
if 条件 A or 条件 B
```

想要得到反向的结果，即当 x 为 true，得到的结果为 false，当 x 为 false 时，得到的结果为 true，我们可以这样写 not 条件。

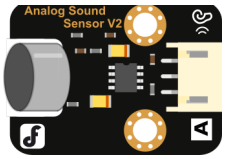
8.5.4 四、硬件分析

在项目中我们用到了两种传感器，光线传感器和声音传感器。

光线传感器是将光信号变成电信号的特殊电子元件，在光线传感器中起到主要作用的就是就是光敏二极管。光敏二极管是光敏电阻中的一种。光明电阻在黑暗环境中，具有非常高阻值的电阻。光线越强，电阻值反而越低。随着两端电阻值的减小，电压也就相应减小，所以从模拟口独到的值也就变小。我们读取的数据也是由此而来的。



声音传感器的作用相当于一个麦克风。它用来接收声波，反馈声音的振动图像。在声音传感器上起到主要作用的就是麦克风，声波使麦克风内的薄膜震动，导致内部电容的变化，而产生与之对应的电压变化，经过转换为可监测的电压值反馈而来。



8.6 项目 5 近视警示器

8.6.1 一、概述

近视眼越来越多，抛开遗传因素，主要还是不良用眼习惯导致的。特别是弓腰驼背，埋头看书，但是知道归知道，真的坐下本来挺直的腰杆又不知不觉地趴在桌子上。

为了健康，为了保护视力，我们用蜂鸣器和超声波传感器做一个简易近视警示器。

8.6.2 二、项目实施

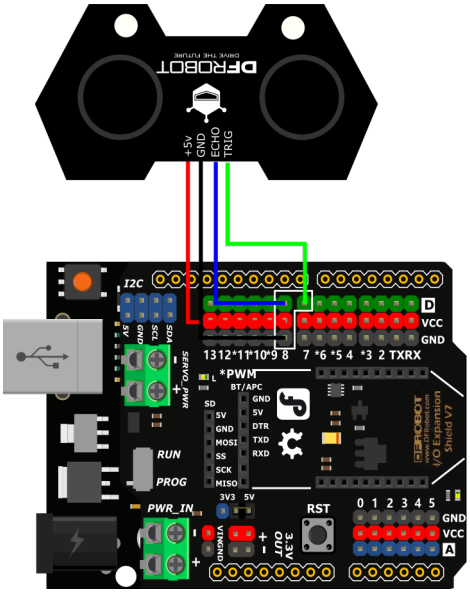
（1）使用超声波测距

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：超声波传感器

连接线：TypeAtoB 方口 USB 连接线



- 接线引脚 TRIG–D7 、ECHO–D8、GND–GND 、+5v–VCC

程序编写：

1. 打开 pingpong 库的官方文档，找到常用库示例中的“超声波传感器”，并用 IDLE 打开。

2. 摁下 F5 运行程序，查看效果。在新弹出的窗口中会显示超声波传感器测得的距离。

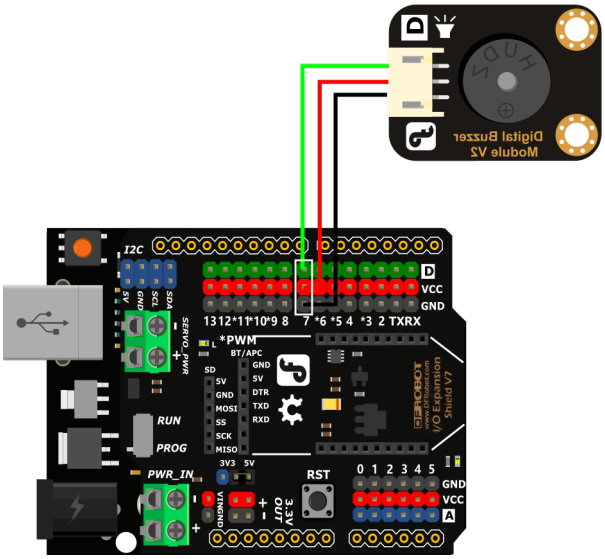
（2）使用蜂鸣器模块

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：蜂鸣器模块

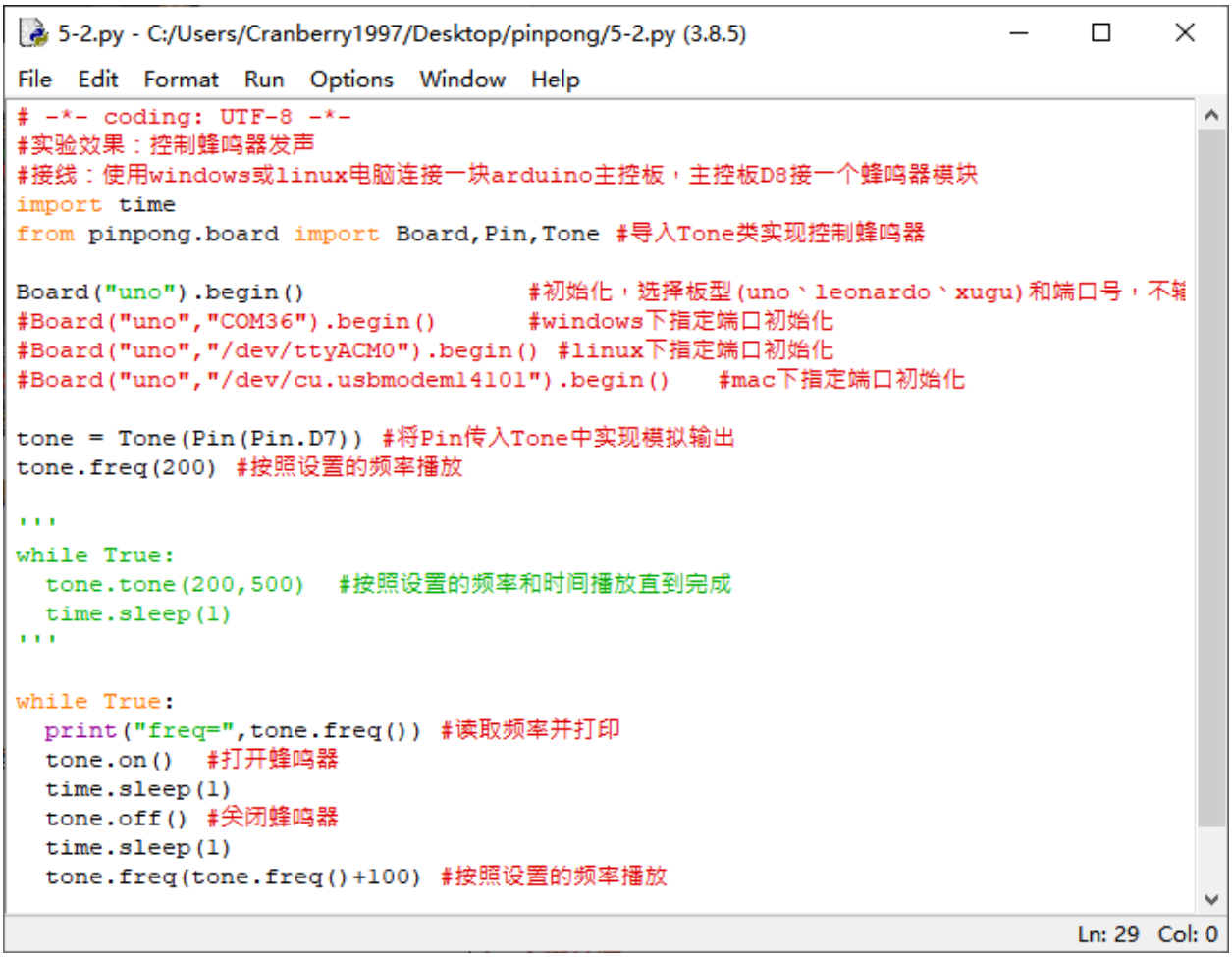
连接线：TypeAtoB 方口 USB 连接线



- 蜂鸣器模块连接到 D7 引脚

程序编写：

1. 打开 pingpong 库的官方文档，找到常用库示例中的“蜂鸣器”，并用 IDLE 打开，将程序中的 D8 改为 D7。



2. 摁下 F5 运行程序，查看效果。在新弹出的窗口中会显示蜂鸣器的声音频率，蜂鸣器会按音阶递进响起。

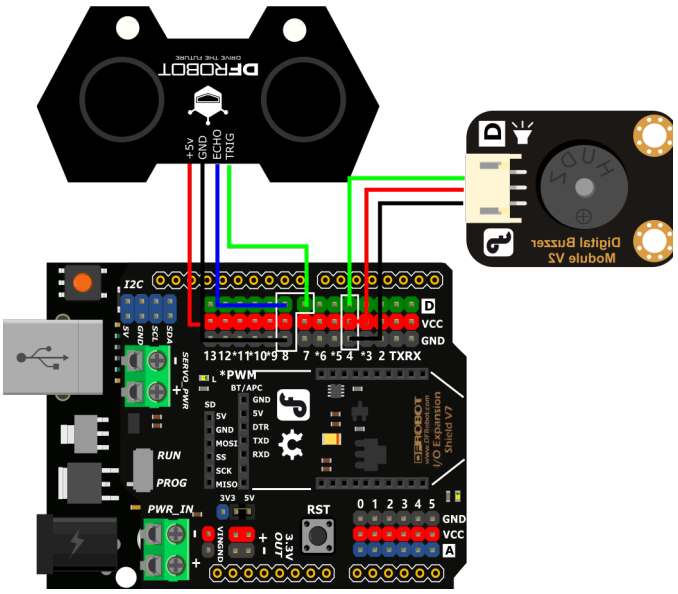
（3）完成近视警示器

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：蜂鸣器模块、超声波传感器

连接线：TypeAtoB 方口 USB 连接线



- 接线引脚 TRIG–D7 、ECHO–D8、GND–GND 、+5v–VCC
- 将蜂鸣器模块接入 D4 引脚。

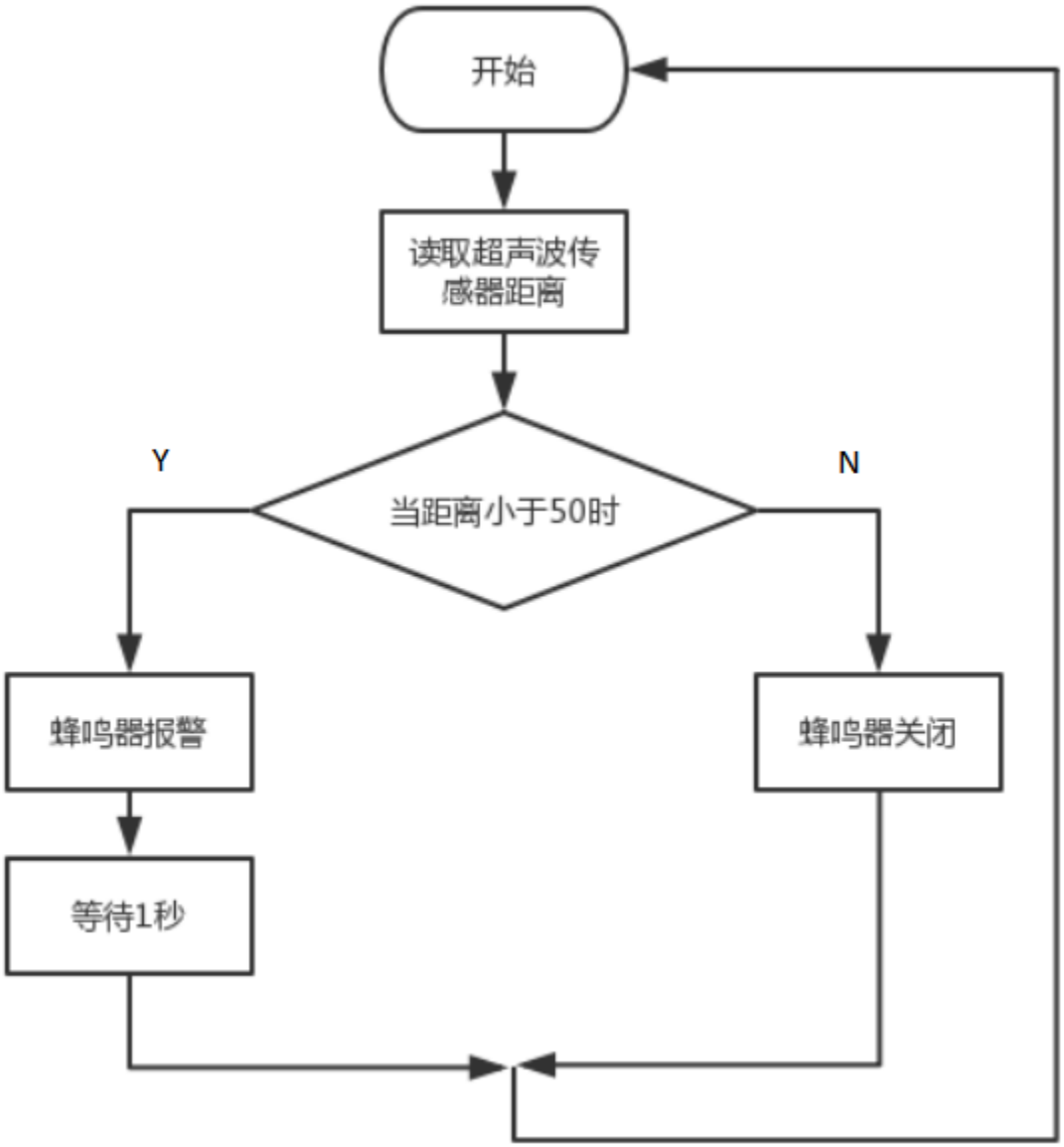
程序编写

```
import time
from pinpong.board import Board,Pin,SR04_URM10,Tone# 导入必要的库函数

Board("uno").begin()# 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别

tone = Tone(Pin(Pin.D4))# 初始化蜂鸣器模块引脚为 D4
TRIGER_PIN = Pin(Pin.D7)# 初始化超声波传感器 TRIG 引脚为 D7
ECHO_PIN = Pin(Pin.D8)# 初始化超声波传感器 ECHO 引脚为 D8

tone.freq(200) # 初始化蜂鸣器频率
sonar = SR04_URM10(TRIGER_PIN,ECHO_PIN)# 初始化超声波传感器

while True:
    dis = sonar.distance_cm() # 读取超声波传感器距离
    print("distance = %d cm"%dis)
    if dis < 50:
        tone.on()
        time.sleep(1)
    else:
        tone.off()
```

运行代码，当超声波传感器监测到距离小于 50 时，蜂鸣器会报警提示距离桌面太近。

8.6.3 三、代码分析

1. 设计程序逻辑图



2. 导入必要的包和初始化设置。

```
import time
from pinpong.board import Board,Pin,SR04_URM10,Tone# 导入必要的库函数

Board("uno").begin()# 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别

tone = Tone(Pin(Pin.D4))# 初始化蜂鸣器模块引脚为 D4
TRIGER_PIN = Pin(Pin.D7)# 初始化超声波传感器 TRIG 引脚为 D7
ECHO_PIN = Pin(Pin.D8)# 初始化超声波传感器 ECHO 引脚为 D8
```

(下页继续)

(续上页)

```
tone.freq(200) # 初始化蜂鸣器频率
sonar = SR04_URM10(TRIGER_PIN,ECHO_PIN)# 初始化超声波传感器
```

3. 加入判断

```
while True:
    dis = sonar.distance_cm()# 读取超声波传感器距离
    print("distance = %d cm"%dis)
    if dis < 50:
        tone.on()
        time.sleep(1)
    else:
        tone.off()
```

8.6.4 四、硬件分析

1. 认识超声波传感器

目前主流的测距传感器有超声波测距传感器，红外线测距传感器，激光测距传感器和雷达传感器。其中，超声波传感器适用于大幅平面静止测距。普通超声波传感器测距范围约 2cm~450cm。

我们可以很清楚的看到实物超声波传感器上有 4 个角：VCC—5V 电源脚, Trig—出发控制端，Echo—接收端，GND—地段。图片中双探头传感器中，一个用来发送超声波，一个用来接收超声波。中间的单头超声波传感器即可以发送也可以接收超声波。这个传感器是我们接触的第一个四个引脚的传感器，使用的接线也比较特殊，后面的硬件连接需要做特殊处理。

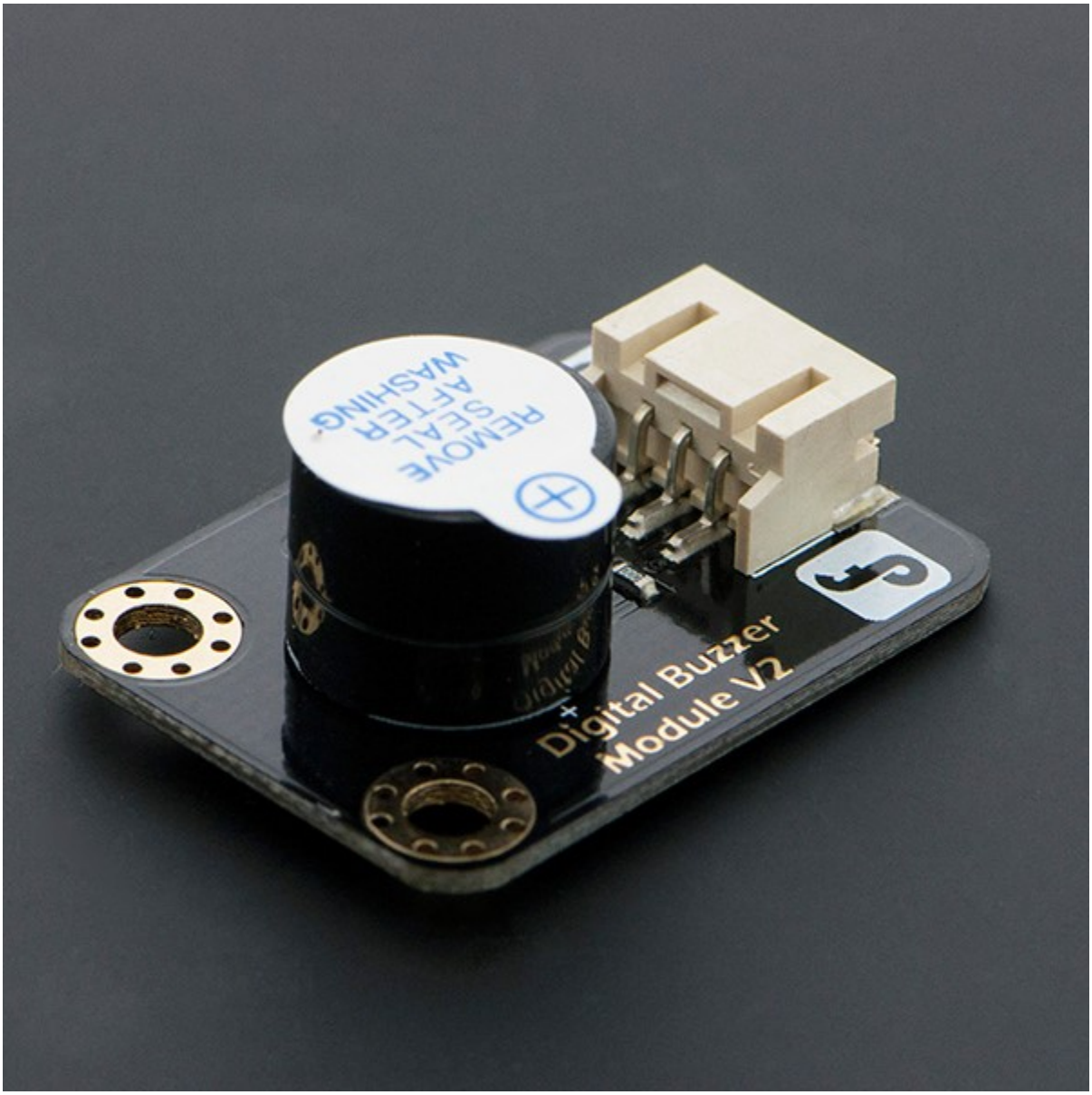
超声波传感器测量距离的过程，超声波发射器向某一方向发射超声波，同时开始计时；超声波在空气中传播，一旦碰到障碍物立即折返；超声波接收器接收到反射波，同时停止计时。那么距离就可以通过时间差来计算出来。

2. 认识蜂鸣器

首先，我们对喇叭应该非常熟悉，常见的耳机就是两个小喇叭，还有收音机，MP3，PM4 播放器，电视机音响的发生原件都是喇叭。喇叭也叫做扬声器，是电声转换期间，它可以把模拟电信号转换为声音信号，属于宽频率发声器件。

而蜂鸣器是一体化的电子讯响器，可以在不同驱动波形下发出单调的声音，属于窄频率发声器件。我们可以通过改变频率设置蜂鸣器发声音高。

从外在表现看，喇叭和蜂鸣器最大区别是喇叭可以发出各种声音，而蜂鸣器只能发出几种单调的声音。从内在发声原理来看，蜂鸣器是利用压电陶瓷将电信号转化为机械振动信号；扬声器是利用电磁铁将电信号转化为机械振动信号。



8.7 项目 6 噪声检测仪

8.7.1 一、概述

在日常生活会有很多噪声的来源，当噪声大到一定程度，就会影响我们的身体健康。那你有想过制作一个噪声检测装置吗？在之前的项目中我们有使用过声音传感器，那如何将检测结果更直观的显示出来？请跟着本文完成项目吧。

8.7.2 二、项目实施

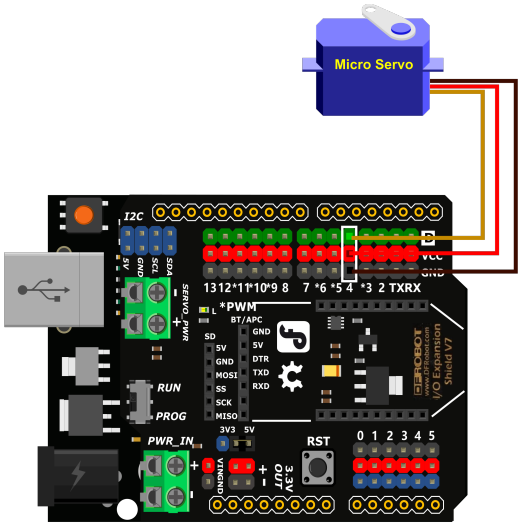
（1）驱动舵机

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：舵机

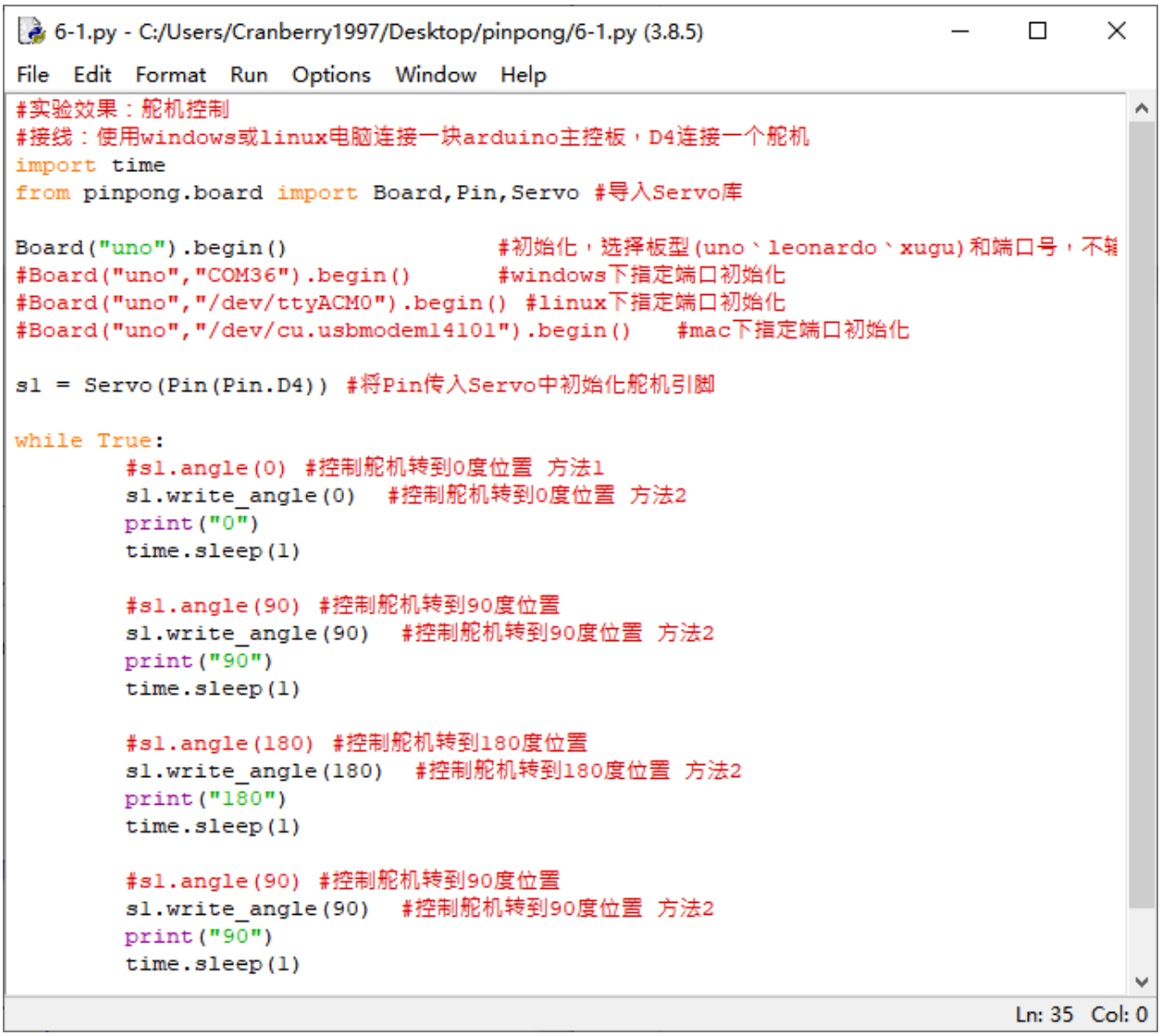
连接线：TypeAtoB 方口 USB 连接线



- 将舵机接入 4 号数字引脚

程序编写：

1. 打开 pingpong 库的官方文档，找到常用库示例中的“舵机”，并用 IDLE 打开。



2. 摁下 F5 运行程序，察看效果。

运行效果

舵机会以 0 度转到 90 度转到 180 度，再转回 90 度转到 0 度循环。

注意：在程序运行时不可以拔掉与 Arduino 连接的 USB 线，且不能关闭新弹出的 Python shell 运行窗口，如果拔线或者关闭运行窗口，程序功能就会停止执行。

3. 如果我们想要修改舵机旋转的角度或让舵机按一定速度慢慢旋转，那我们只需要修改角度即可。示例程序为舵机从 0 度慢慢旋转至 120 度。

```
import time
from pinpong.board import Board,Pin,Servo

Board("uno").begin()

s1 = Servo(Pin(Pin.D4))

while True:
    for i in range(120):
        s1.write_angle(i)
        time.sleep(0.1)
```

（2）用舵机反馈声音数值

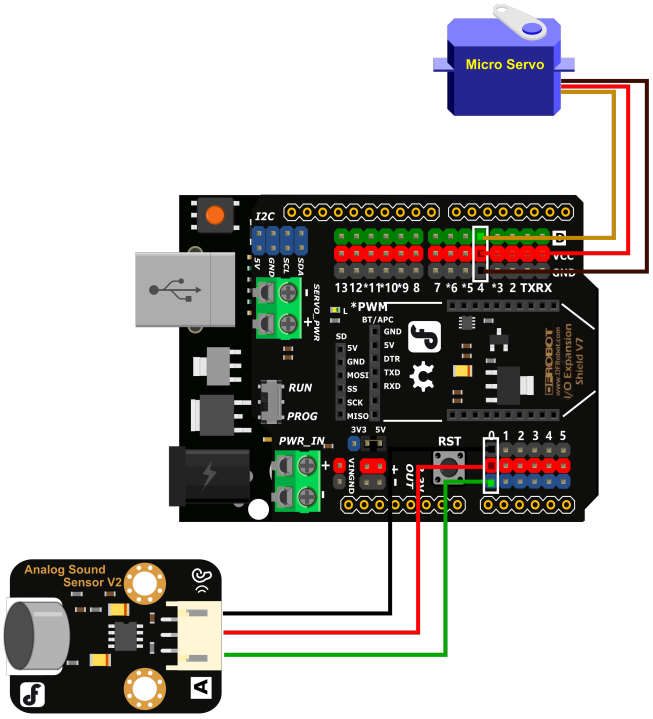
大家可能会好奇，舵机是如何实现反馈声音数值的功能。如图所示，我们可以将读取到的数值转换成舵机的角度值，然后用舵柄当作指针来反馈数值。

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：舵机、声音传感器

连接线：TypeAtoB 方口 USB 连接线



- 将舵机接入 4 号数字引脚，声音传感器接入 A0 模拟引脚。

程序编写

1. 我们在之前的项目中使用过声音传感器，知道如何读取它的数值，但是声音传感器读出的数据是 0~1023 而我们使用的舵机的旋转角度是 0~180 度，所以我们需要在程序中对这个数值进行转换。

```
vs = Sound.read_analog()
servoTurn = int(vs*180/1024)
```

在这里加入 int，将数值整形，转化为舵机旋转的角度。

2. 在了解了如何将声音传感器读取的数值转换为舵机可旋转的角度值之后，修改程序，示例程序如下。

```
import time
from pinpong.board import Board,Pin,Servo

Board("uno").begin()

s1 = Servo(Pin(Pin.D4))
Sound = Pin(Pin.A0,Pin.ANALOG)

while True:
    vs = Sound.read_analog()
    servoTurn = int(vs*180/1024)
```

(下页继续)

(续上页)

```
    print(servoTurn)
    s1.write_angle(servoTurn)
    time.sleep(0.5)
```

8.7.3 三、代码分析

```
import time
from pinpong.board import Board,Pin,Servo

Board("uno").begin()

s1 = Servo(Pin(Pin.D4)) # 初始化舵机在 D4 引脚
Sound = Pin(Pin.A0,Pin.ANALOG)# 初始化声音传感器在 A0 引脚

while True:
    vs = Sound.read_analog()
    servoTurn = int(vs*180/1024)# 转换声音数值为舵机角度
    print(servoTurn)
    s1.write_angle(servoTurn)
    time.sleep(0.5)
```

思考

在实际使用时我们会发现舵机的转动角度基本不会超过 90 度，这里我们可以试着将读取到的声音值打印出来观察一下，看看究竟为什么舵机只会在一个较小的范围内转动。

8.7.4 四、硬件分析

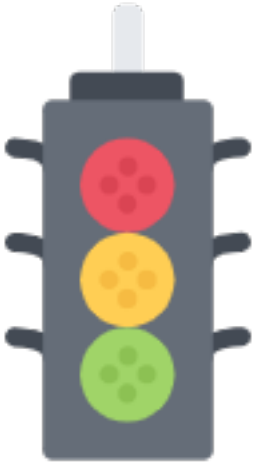
什么是舵机？

舵机是一种可以指定控制位置（角度）的电机，可以通过程序来指定控制舵机旋转的角度。我们最常用的舵机大多最大旋转角度是 0°~180°，也有 90° 或者其他角度的。也有比较特殊的 360° 舵机，但是 360° 舵机不能够控制其旋转到指定的角度。本项目中我们使用的是 180° 舵机。

8.8 项目 7 模拟交通灯

8.8.1 一、概述

交通灯我们经常能在路口上见到，已经习以为常，但是你知道它是如何工作的，如何实现交通灯的功能呢？接下来跟着我们来学习一下吧。



8.8.2 二、项目实施

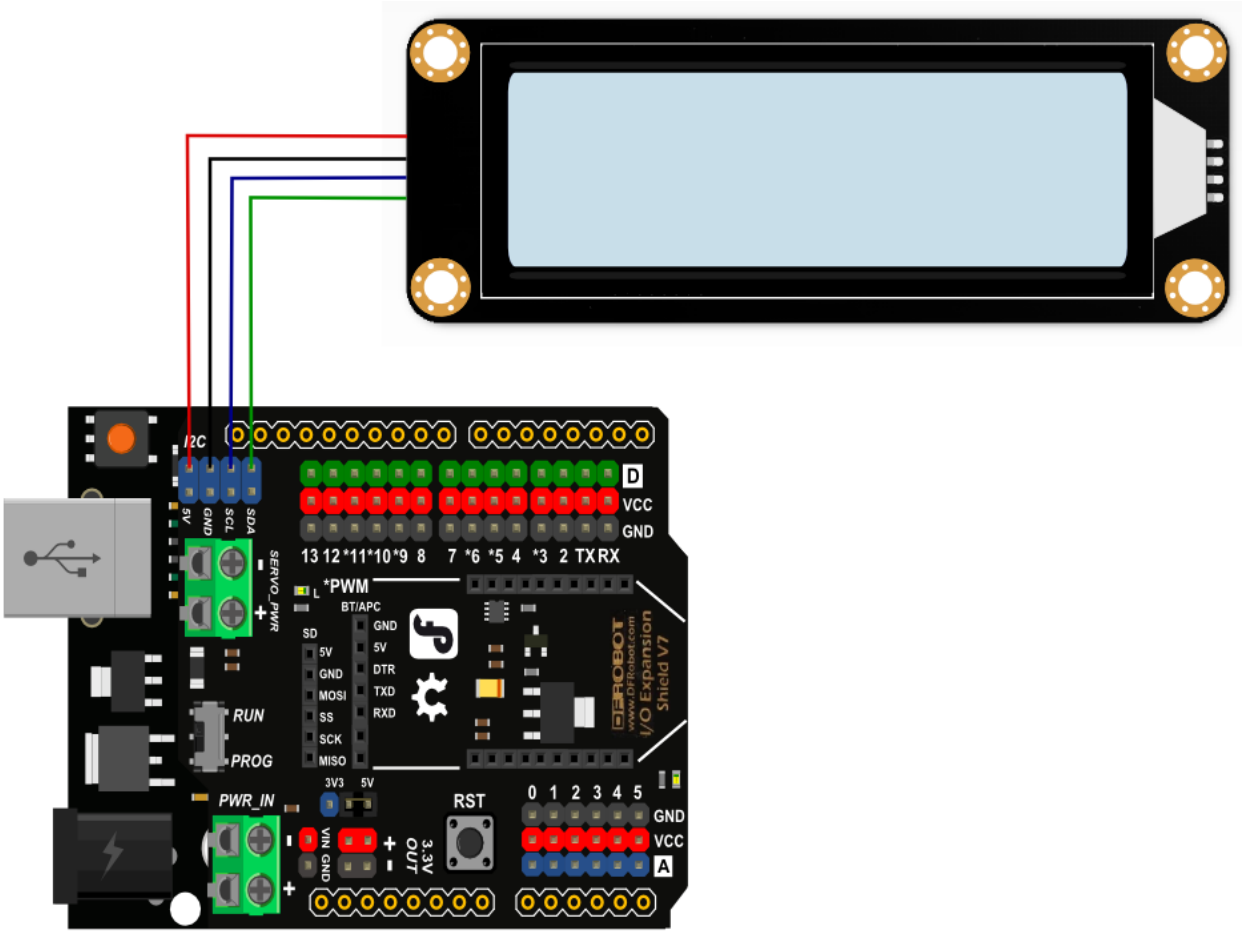
（1）驱动 LCD 显示屏

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：1602LCD 显示屏

连接线：TypeAtoB 方口 USB 连接线



- 将 1602LCD 显示屏接入 IIC 接口

程序编写：

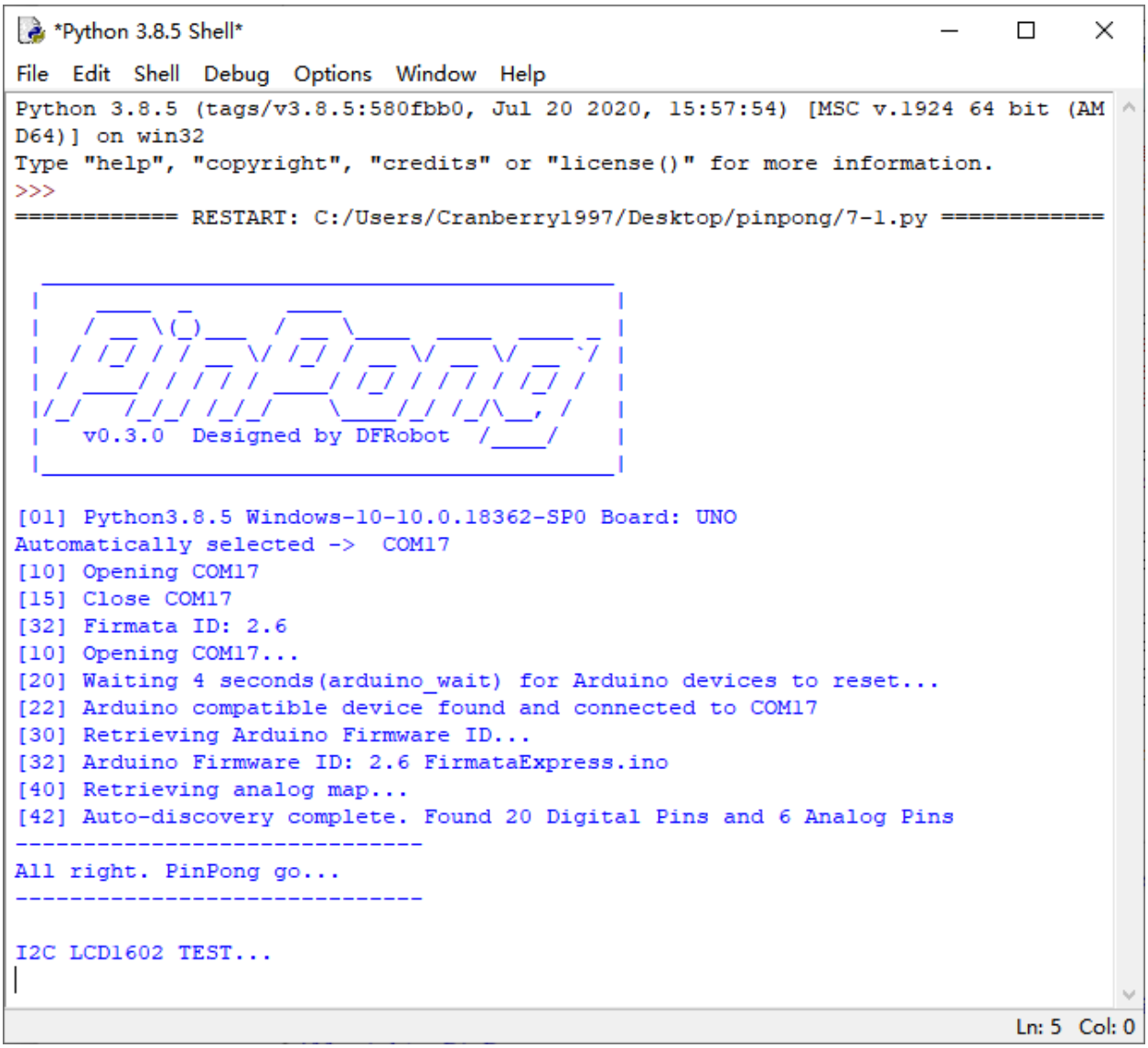
1. 打开示例程序 lcd1602.py，运行程序。（关于 1602 的示例程序有两个，一个是 lcd1602.py 一个是 rgb1602.py, 本项目使用的是单色背光的 1062 模块，所以使用 lcd1602.py。打开 pingpong 库的官方文档，找到扩展库示例中的 “1602 显示屏”，并用 IDLE 打开。

7-1.py - C:/Users/Cranberry1997/Desktop/pinpong/7-1.py (3.8.5)

File Edit Format Run Options Window Help

```
# -*- coding: UTF-8 -*-
#实验效果：I2C LCD1602控制
#接线：使用windows或linux电脑连接一块arduino主控板，LCD1602显示屏接到I2C口SCL及SDA
import time
from pinpong.board import Board
from pinpong.libs.lcd1602 import LCD1602_I2C #从libs中导入lcd1602_i2c库

Board("uno").begin()                #初始化，选择板型(uno、leonardo、xugu)和端口号，不输
#Board("uno","COM36").begin()       #windows下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()    #mac下指定端口初始化

lcd = LCD1602_I2C(i2c_addr=0x20) #初始化LCD的I2C地址
print("I2C LCD1602 TEST...")

lcd.backlight(True)  #打开背光
lcd.clear()    #清屏

lcd.set_cursor(0,0)  #设置光标位置
lcd.print("Hello World") #显示 "Hello World",1602屏像素点少，不能显示汉字
lcd.set_cursor(1,1)  #设置光标位置
lcd.print(1234) #显示数字1234

while True:
    time.sleep(1)
    lcd.scroll_left() #滚动显示
```

Ln: 29 Col: 0



运行效果

显示屏上第一行显示 hello world，第二行显示 1234，在屏幕上滚动播放。

2. 如果我们想要修改屏幕上的内容，还有文字显示的位置，我们可以根据内容进行调整。

```
# -*- coding: UTF-8 -*-
# 实验效果：I2C LCD1602 控制
# 接线：使用 windows 或 linux 电脑连接一块 arduino 主控板，LCD1602 显示屏接到 I2C 口 SCL 及 SDA
import time
from pinpong.board import Board
from pinpong.libs.lcd1602 import LCD1602_I2C # 从 libs 中导入 lcd1602_i2c 库
```

(下页继续)

(续上页)

```
Board("uno").begin()                # 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输
入端口号则进行自动识别
#Board("uno","COM36").begin()      #windows 下指定端口初始化
#Board("uno","/dev/ttyACM0").begin() #linux 下指定端口初始化
#Board("uno","/dev/cu.usbmodem14101").begin()   #mac 下指定端口初始化

lcd = LCD1602_I2C(i2c_addr=0x20) # 初始化 LCD 的 I2C 地址
print("I2C LCD1602 TEST...")

lcd.backlight(True)  # 打开背光
lcd.clear()   # 清屏

lcd.set_cursor(2,0)  # 设置光标位置
lcd.print("Hello PinPong") # 显示 "Hello PinPong",1602 屏像素点少，不能显示汉字
lcd.set_cursor(6,1)  # 设置光标位置
lcd.print(6666) # 显示数字 1234

while True:
    time.sleep(1)
    lcd.scroll_left() # 滚动显示
```

（2）加入 LED 和倒计时

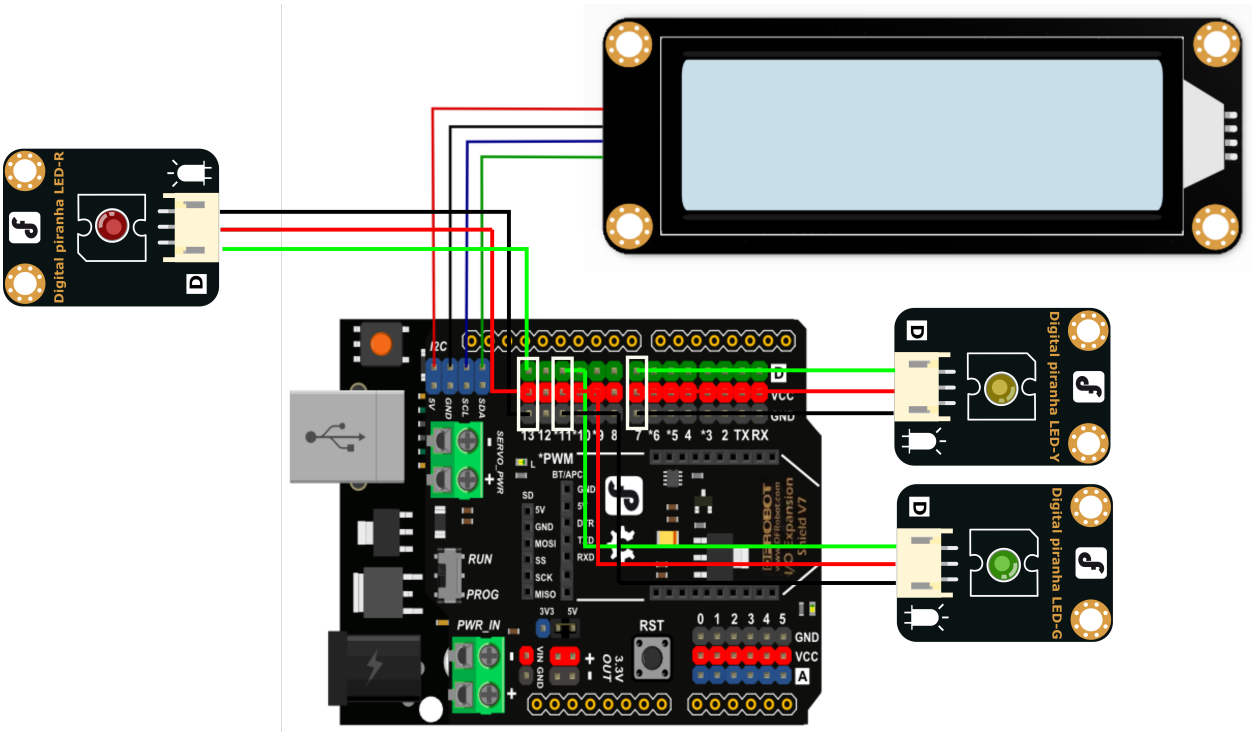
仔细回顾一下交通灯的灯光切换方式，在红灯时，倒计时结束会变为绿灯，当绿灯倒计时结束时会先切到黄灯几秒后再切换到红灯亮起，依据这个规律，我们让 LCD 显示倒计时秒数。

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：1602LCD 显示屏、红色黄色绿色 LED 模块

连接线：TypeAtoB 方口 USB 连接线



- 将 1602LCD 显示屏接入 IIC 接口
- 将红色 LED 灯模块接 D13 数字引脚、绿色 LED 灯模块接 D11 数字引脚、黄色 LED 灯模块接 D7 数字引脚

程序编写

```
import time
from pinpong.board import Board,Pin
from pinpong.libs.lcd1602 import LCD1602_I2C

Board("uno").begin()
ledR = Pin(Pin.D13,Pin.OUT)# 初始化红灯引脚在 D13
ledG = Pin(Pin.D11,Pin.OUT)# 初始化绿灯引脚在 D11
ledY = Pin(Pin.D7,Pin.OUT)# 初始化黄灯引脚在 D7
lcd = LCD1602_I2C(i2c_addr=0x20) # 初始化 LCD 地址为 0x20
lcd.backlight(True) # 打开背光
lcd.clear()  # 清屏

while True:
    for G in range(30,-1,-1):  # 设置倒计时数值
        ledR.write_digital(0)
        ledG.write_digital(1)
        ledY.write_digital(0)
```

(下页继续)

(续上页)

```
        lcd.set_cursor(7,1)  # 设置光标位置
        lcd.print(G)  # 让 LCD 显示倒计时数值
        time.sleep(1)
        lcd.clear()

    for Y in range(5,-1,-1):
        ledR.write_digital(0)
        ledG.write_digital(0)
        ledY.write_digital(1)
        lcd.set_cursor(7,1)
        lcd.print(Y)
        time.sleep(1)
        lcd.clear()

    for R in range(30,-1,-1):
        ledR.write_digital(1)
        ledG.write_digital(0)
        ledY.write_digital(0)
        lcd.set_cursor(7,1)
        lcd.print(R)
        time.sleep(1)
        lcd.clear()
```

8.8.3 三、代码分析

1. 导入必要库和模块，参考之前使用过 LED 灯的案例加上本次使用的 LCD 屏所需的部分。

```
import time
from pinpong.board import Board,Pin
from pinpong.libs.lcd1602 import LCD1602_I2C
```

2. 然后对需要用到的功能进行初始化设置。

```
Board("uno").begin()
ledR = Pin(Pin.D13,Pin.OUT)
ledG = Pin(Pin.D11,Pin.OUT)
ledY = Pin(Pin.D7,Pin.OUT)
lcd = LCD1602_I2C(i2c_addr=0x20)
lcd.backlight(True)
lcd.clear()
```

3. 设置一段红灯的倒计时功能，另外两种灯用相同的方法设置。

```
for G in range(30,-1,-1):
    ledR.write_digital(0)
    ledG.write_digital(1)
    ledY.write_digital(0)
    lcd.set_cursor(7,1)
    lcd.print(G)
    time.sleep(1)
    lcd.clear()
```

for i in range () 的作用：

range () 是一个函数，for i in range () 就是给 i 赋值，比如：

for i in range（30）：

就是把 0~30 依次赋值给 i，在程序中加入的

for i in range（30，-1，-1）：

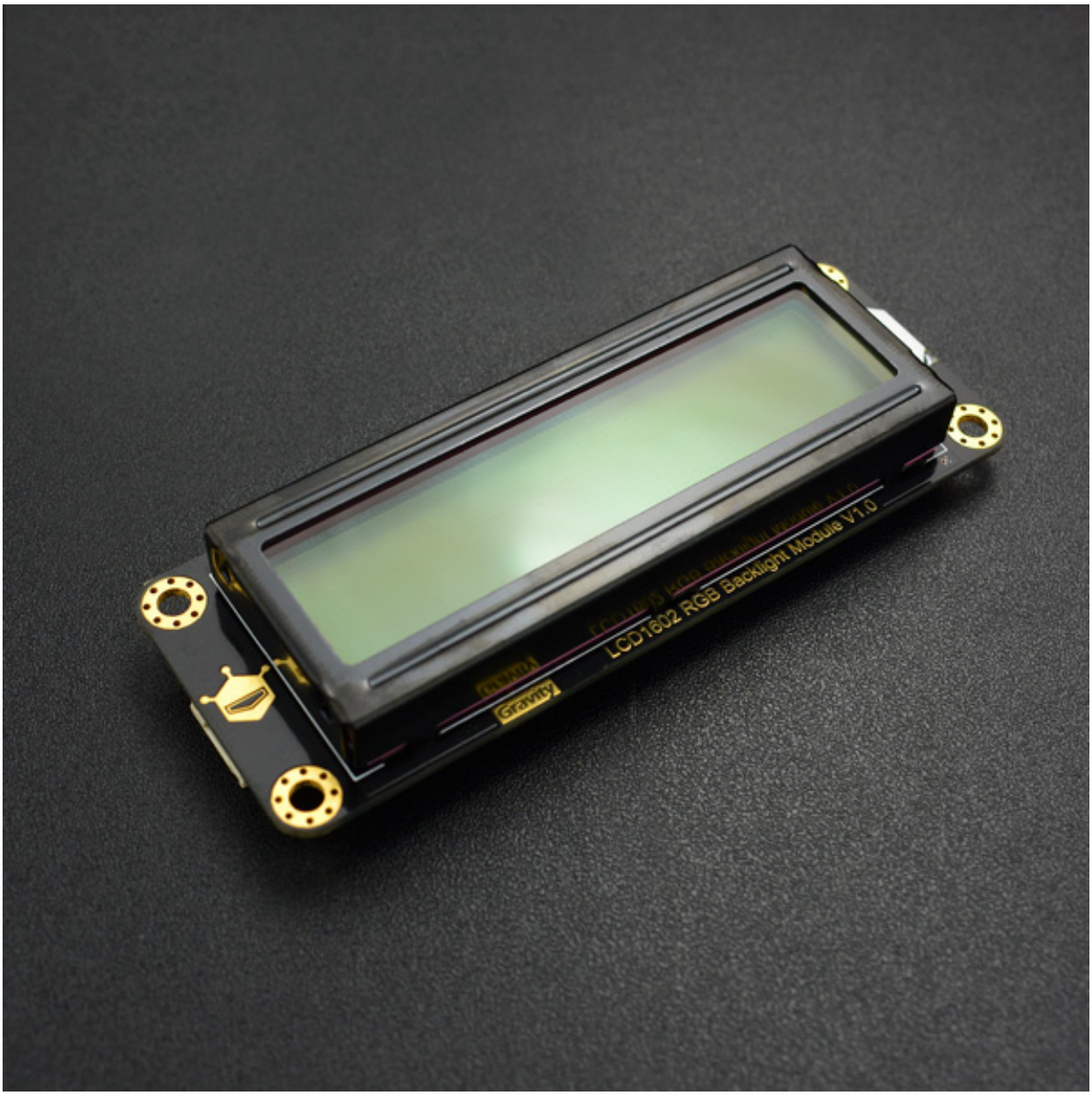
是将 0~30 的数值顺序倒序，按 30~0 的顺序赋值给 i。

8.8.4 四、硬件分析

什么 LCD1602？

LCD1602 液晶显示器是广泛使用的一种字符型液晶显示模块。1602 的意思是显示屏一行最多可以显示 16 个字符，一共有 2 行。

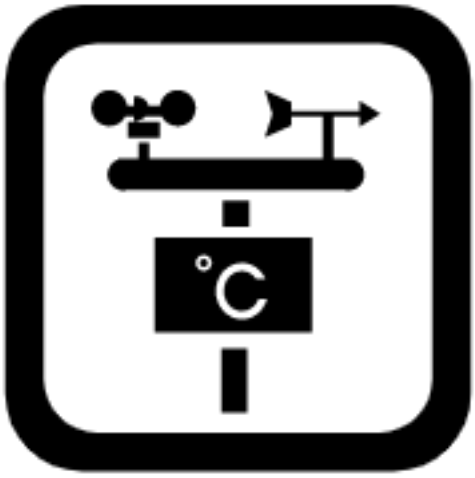
我们使用的是已经集成了控制驱动电路的成品模，通过 iic 接口与主控板连接。



8.9 项目 8 桌面气象站

8.9.1 一、概述

在日常生活中，你是否会关注自己身处环境的温度？自己身处的环境是否过热 or 过冷？什么时候需要开空调了？如果你有这些疑问，就请跟着我们来完成一个桌面气象站吧。



8.9.2 二、项目实施

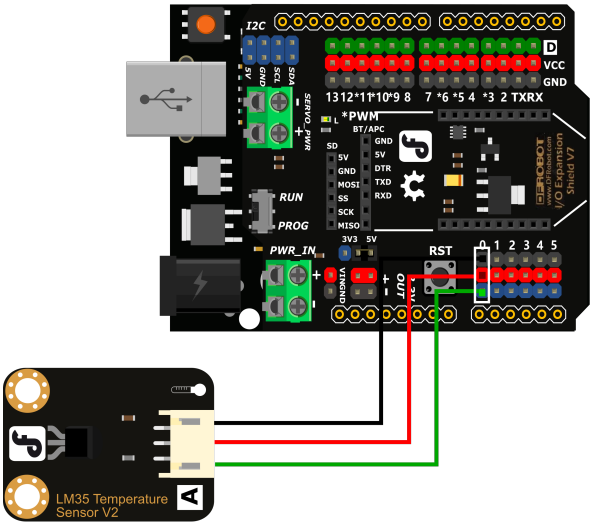
（1）读取温度传感器数据

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：LM35 线性温度传感器

连接线：TypeAtoB 方口 USB 连接线



- 将 LM35 线性温度传感器接入 A0 模拟接口

程序编写：

1. LM35 温度传感器需要通过读取模拟值经过一定的换算得到温度数据，那么首先就需要读取对应引脚的模拟值，回顾之前案例，打开 pingpong 库的官方文档，找到基础库示例中的“模拟输入”，并用 IDLE 打开。

2. 修改程序，添加转换公式：温度 = 读取到模拟值 * (5/10.24); 这样就能顺利读取温度数据了。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()

adc0 = Pin(Pin.A0, Pin.ANALOG)

while True:
v = adc0.read_analog()
tem = round(v*(5/10.24),2)
print("temperature:", tem)
time.sleep(0.5)
```

（2）让屏幕显示温度数据

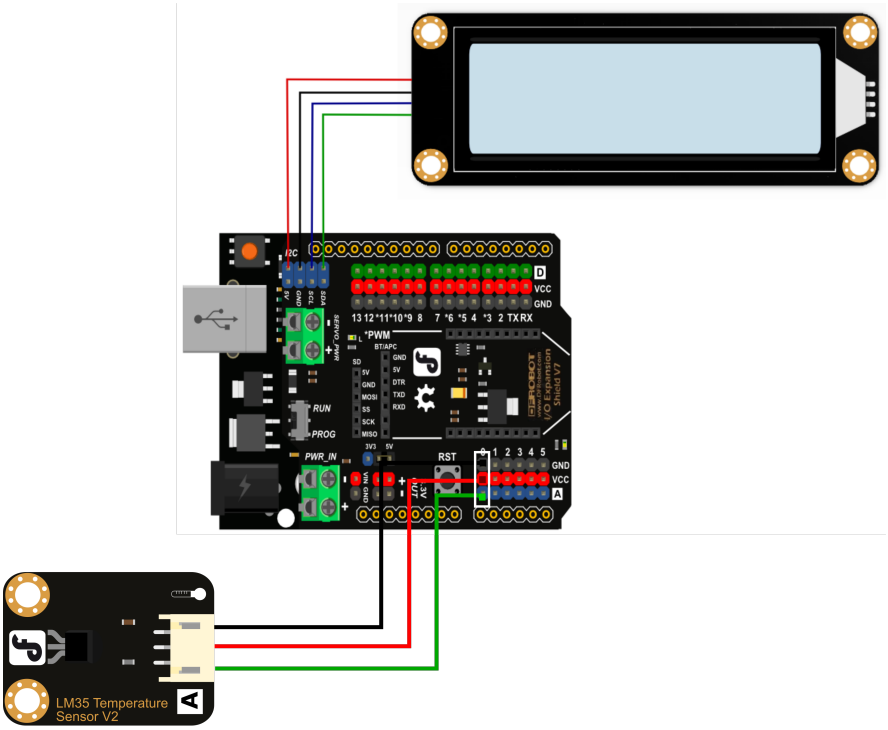
在上个项目中我们使用了 LCD 显示屏，为了方便查看温度的实时数据，我们将数据在屏幕上显示出来。

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：1602LCD 显示屏、LM35 线性温度传感器

连接线：TypeAtoB 方口 USB 连接线



- 将 1602LCD 显示屏接入 IIC 接口
- 将 LM35 线性温度传感器接入 A0 模拟接口

程序编写

```
import time
from pinpong.board import Board,Pin
from pinpong.libs.lcd1602 import LCD1602_I2C

Board("uno").begin()# 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别
adc0 = Pin(Pin.A0, Pin.ANALOG) # 初始化温度读取引脚在 A0
lcd = LCD1602_I2C(i2c_addr=0x20)# 初始化 LCD 的 I2C 地址
lcd.backlight(True) # 打开背光
lcd.clear()# 清屏
lcd.set_cursor(2,0)# 设置光标位置
lcd.print('temperature')# 显示 “temperature”

while True:
```

（下页继续）

(续上页)

```
    v = adc0.read_analog()# 读取模拟量的值
    tem = round(v*(5/10.24),2)# 将读到的数值转化为温度数据
    lcd.set_cursor(5,1)
    lcd.print(str(tem))
    lcd.print('C')
    time.sleep(1)
```

8.9.3 三、代码分析

1. 导入必要库和模块，参考之前使用过的 LCD 屏所需的部分和 adc 部分。

```
import time
from pinpong.board import Board,Pin
from pinpong.libs.lcd1602 import LCD1602_I2C
```

2. 然后对需要用到的功能进行初始化设置。

```
Board("uno").begin()# 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输入端口号则进行自
动识别
adc0 = Pin(Pin.A0, Pin.ANALOG) # 初始化温度读取引脚在 A0
lcd = LCD1602_I2C(i2c_addr=0x20)# 初始化 LCD 的 I2C 地址
lcd.backlight(True) # 打开背光
lcd.clear()# 清屏
lcd.set_cursor(2,0)# 设置光标位置
lcd.print('temperature')# 显示 “temperature”
```

3. 让屏幕显示温度，每隔一秒钟刷新。

```
while True:
    v = adc0.read_analog()# 读取模拟量的值
    tem = round(v*(5/10.24),2)# 将读到的数值转化为温度数据
    lcd.set_cursor(5,1)
    lcd.print(str(tem))
    lcd.print('C')
    time.sleep(1)
```

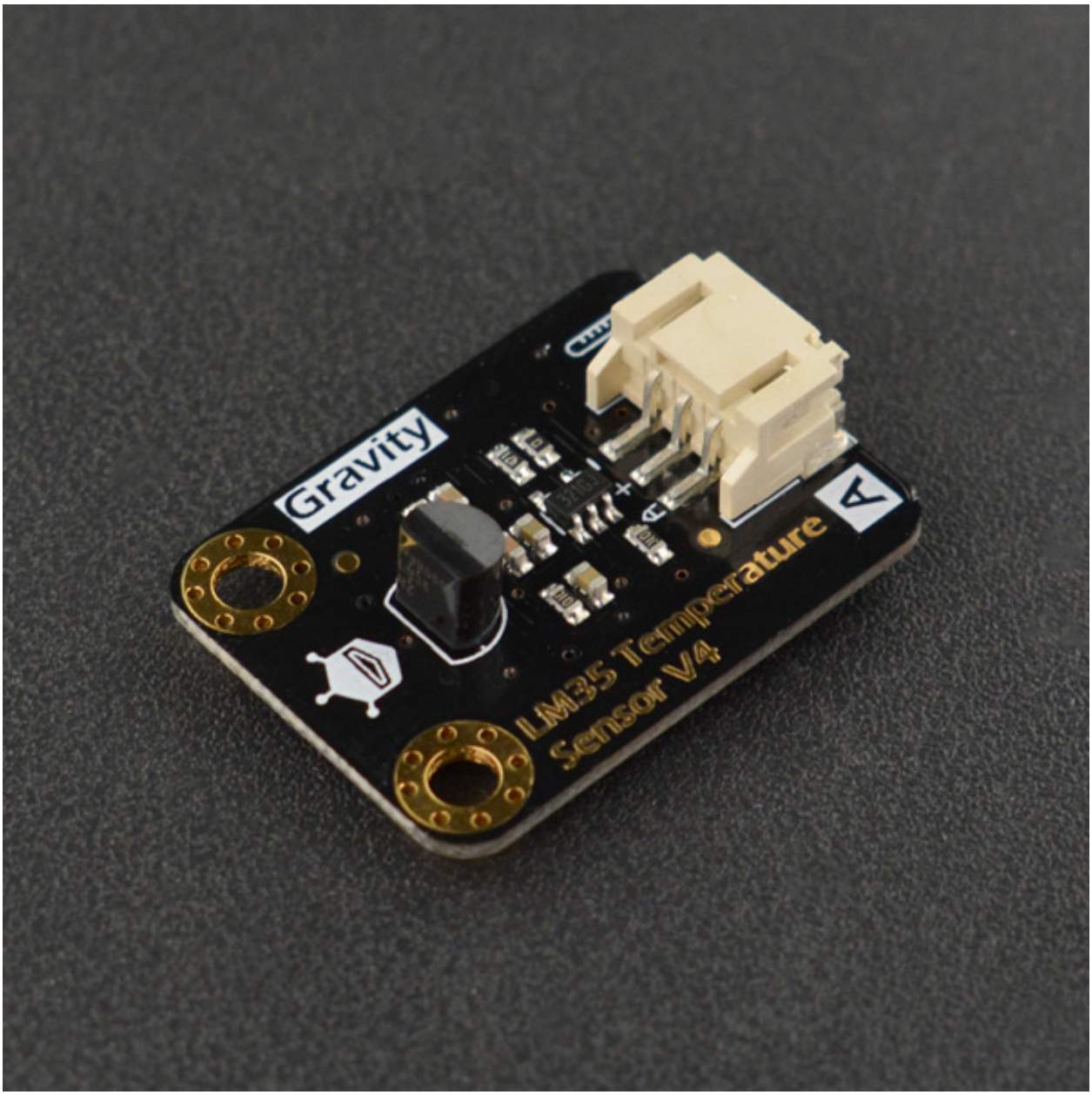
round() 的作用：

round() 函数用来返回浮点数四舍五入后的值，例如 round(35.543,2) 返回的结果就是 35.54，(35.543,2) 中的 2 代表保留两位小数。

8.9.4 四、硬件分析

LM35 线性温度传感器

基于 LM35 半导体的温度传感器，可以用来对环境温度进行定性的检测。LM35 半导体温度传感器是美国国家半导体公司生产的线性温度传感器。其测温范围是-40℃ 到 150℃，灵敏度为 10mV/℃，输出电压与温度成正比。



8.10 项目 9 游园人数统计

8.10.1 一、概述

在去景区游玩的时候我们常能看到门口会有个计数器，上面写着“今日游客数量 XXXX”用于告知大家今日在园内游玩的人数，那这个计数功能是如何实现的呢？接下来跟我们一起来学习吧。

8.10.2 二、项目实施

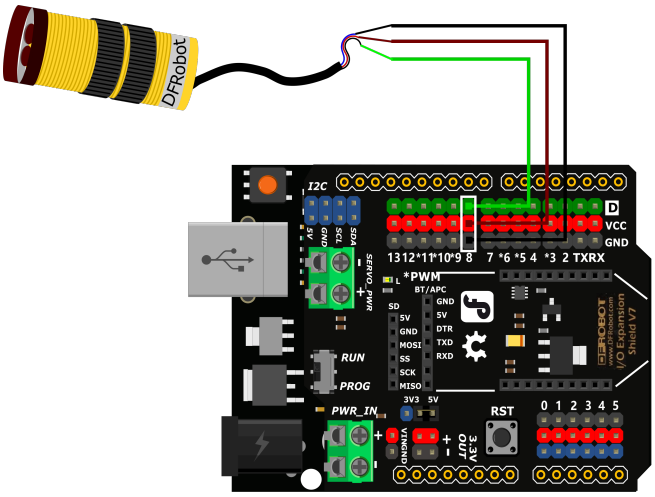
（1）使用红外光电开关检测是否有人经过

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：红外光电开关

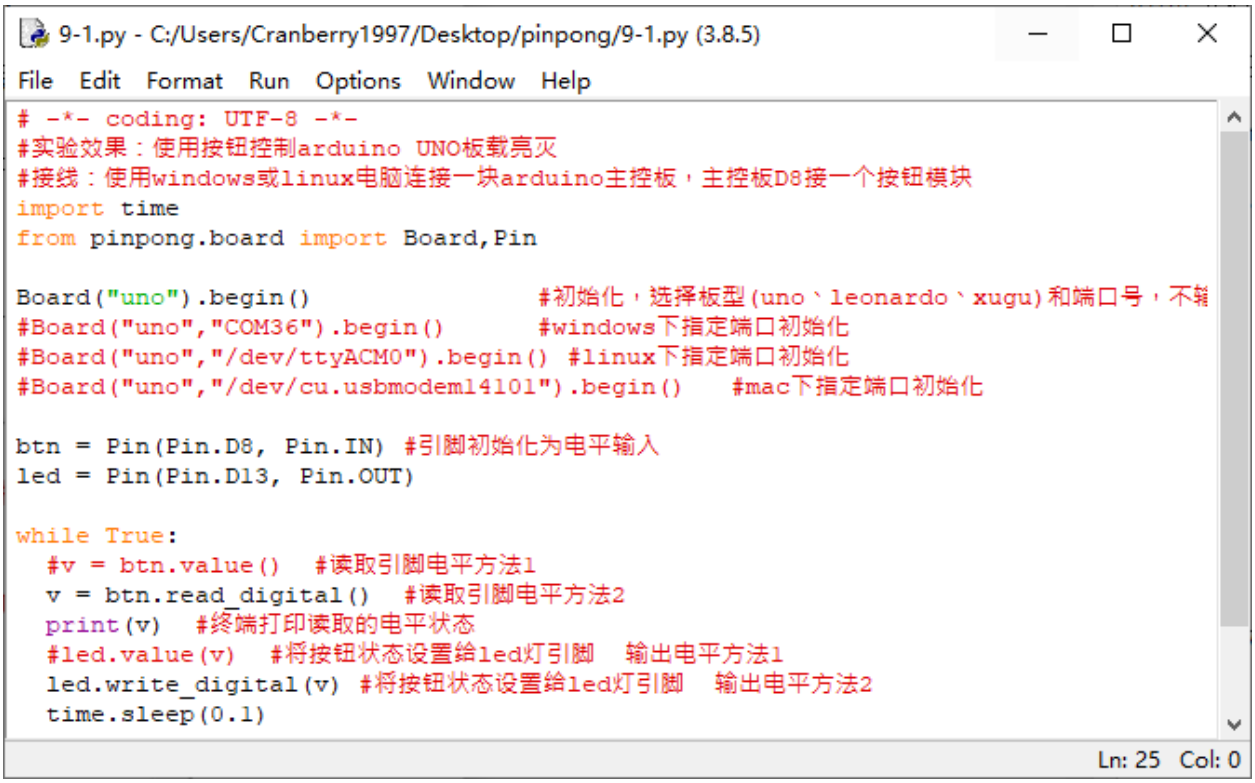
连接线：TypeAtoB 方口 USB 连接线



- 将红外光电开关接入 8 号数字接口

程序编写：

1. 红外光电开关是通过检测指定距离内（3~80cm，可以调节）是否有物体经过，当检测到有人通过的话，会输出低电平，当没人经过的时候会输出高电平。看起来功能和我们之前用过的按钮的功能类似，是数字输入，所以我们以官方文档基础库示例中的“数字输入”为基础进行修改。



2. 原程序是摁下按键时 L 灯亮起，但是红外光电开关是触发时输出低电平，未触发时输出高电平，所以将条件置反，就可以当传感器触发时灯亮，未触发时灯灭的功能。

```
import time
from pinpong.board import Board,Pin

Board("uno").begin()

btn = Pin(Pin.D8, Pin.IN)
led = Pin(Pin.D13, Pin.OUT)

while True:
    v = btn.read_digital()
    print(v)
    led.write_digital(not v)
    time.sleep(0.1)
```

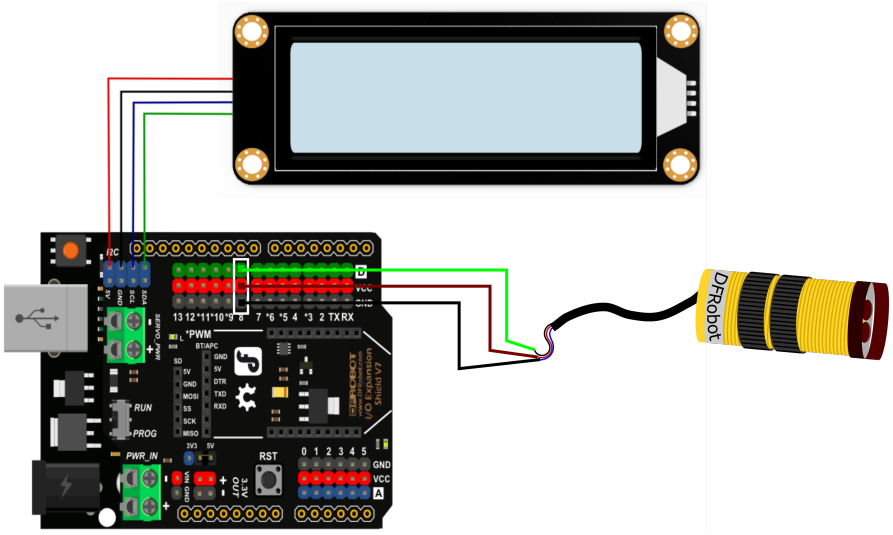
(2) 让屏幕显示人数

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：1602LCD 显示屏、红外光电开关

连接线：TypeAtoB 方口 USB 连接线



- 将 1602LCD 显示屏接入 IIC 接口
- 将红外光电开关接入 8 号数字接口

程序编写

```
import time
from pinpong.board import Board,Pin
from pinpong.libs.lcd1602 import LCD1602_I2C

Board("uno").begin()# 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别
sw = Pin(Pin.D8, Pin.IN) # 初始化红外开关引脚在 D8
lcd = LCD1602_I2C(i2c_addr=0x20)# 初始化 LCD 的 I2C 地址
lcd.backlight(True) # 打开背光
lcd.clear()# 清屏
v=0
lcd.set_cursor(2,0)# 设置光标位置
lcd.print('Visitors')# 显示标题

while True:
    if sw.read_digital() == 0:
        v = v+1
        led.write_digital(1)
        time.sleep(0.5)
    print(v)
```

(下页继续)

(续上页)

```
    lcd.set_cursor(7,1)# 设置光标位置
    lcd.print(v)# 显示人数
```

8.10.3 三、代码分析

1. 导入必要库和模块，参考之前使用过的 LCD 屏所需的部分和 pin 部分。

```
import time
from pinpong.board import Board,Pin
from pinpong.libs.lcd1602 import LCD1602_I2C
```

2. 然后对需要用到的功能进行初始化设置。

```
Board("uno").begin()# 初始化，选择板型 (uno、leonardo、xugu) 和端口号，不输入端口号则进行自动识别
sw = Pin(Pin.D8, Pin.IN) # 初始化红外开关引脚在 D8
lcd = LCD1602_I2C(i2c_addr=0x20)# 初始化 LCD 的 I2C 地址
lcd.backlight(True) # 打开背光
lcd.clear()# 清屏
v=0
lcd.set_cursor(2,0)# 设置光标位置
lcd.print('Visitors')# 显示标题
```

3. 每当有人经过就将变量 v 增加 1 并用显示屏显示人数。

```
while True:
    if sw.read_digital() == 0:
        v = v+1
        led.write_digital(1)
        time.sleep(0.5)
    print(v)
    lcd.set_cursor(7,1)# 设置光标位置
    lcd.print(v)# 显示人数
```

8.10.4 四、硬件分析

红外光电开关

红外接近开关是一种集发射与接收于一体的光电开关传感器。数字信号的输出伴随传感器后侧指示灯亮的亮灭，检测距离可以根据要求进行调节，可调范围 3-80cm。该传感器具有探测距离远、受可见光干扰小、价格便宜、易于装配、使用方便等特点，可以广泛应用于机器人避障、互动媒体、工业自动化流水线等众多场合。

8.11 项目 10 定时浇花装置

8.11.1 一、概述

相信大家的家中应该都会养有一些植物吧，想要植物生长的旺盛我们就需要定期给他们浇水，但是我们可能常常会忘记给家中的植物浇水，可能就会导致植株枯萎，为了避免出现这类问题，我们可以自己设计一个定时浇花装置。

8.11.2 二、项目实施

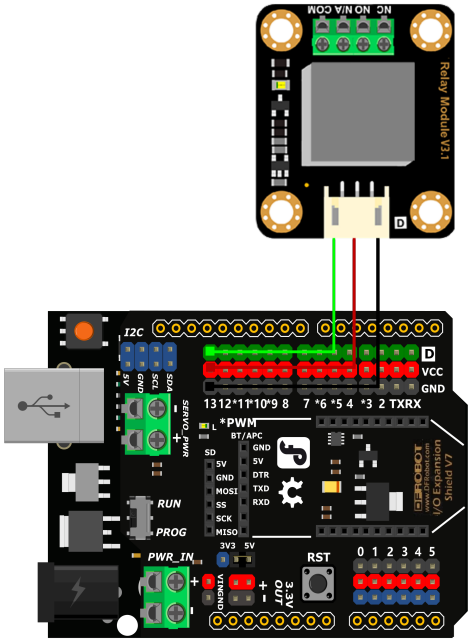
（1）驱动继电器

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：继电器模块

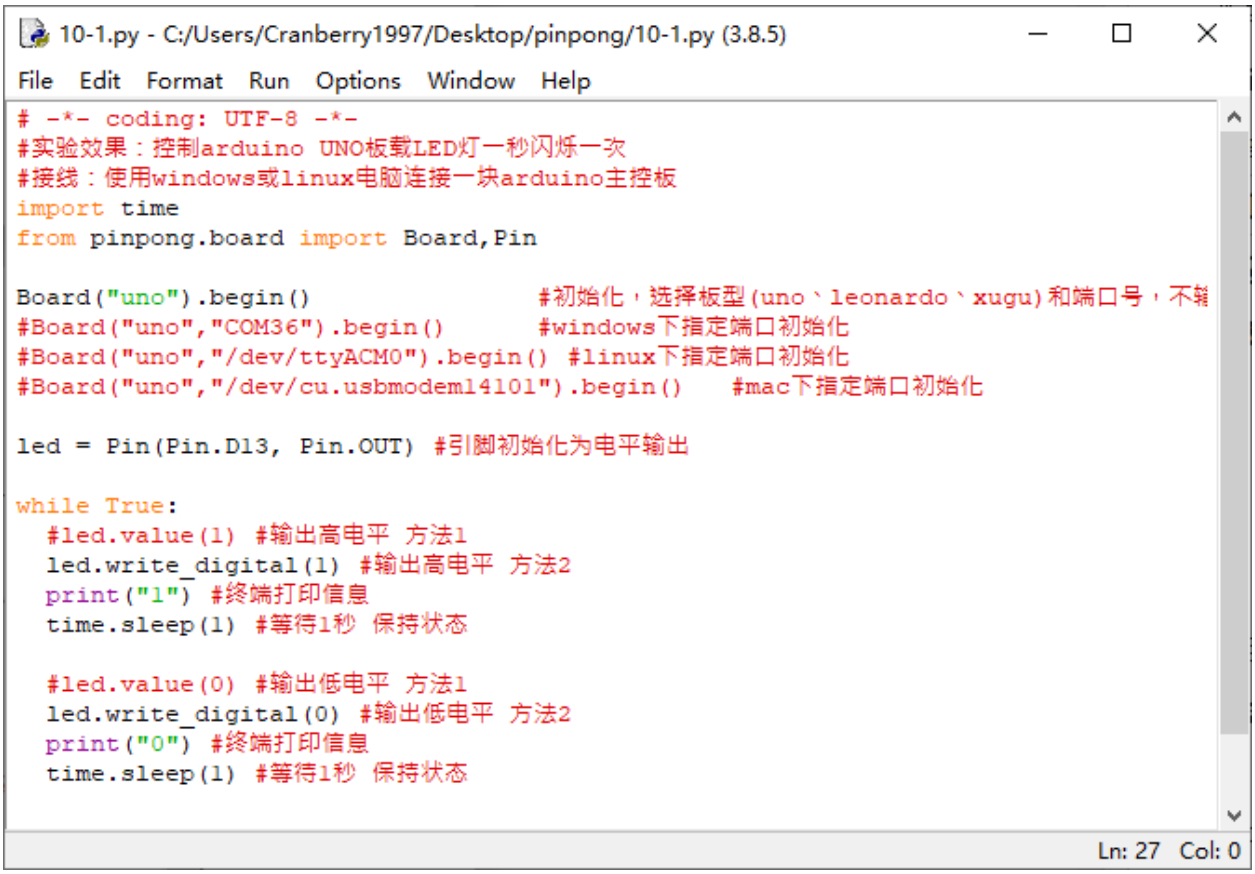
连接线：TypeAtoB 方口 USB 连接线



- 将继电器接入 13 号数字接口

程序编写：

1. 继电器是通过高低电平来控制开关的，所以我们可以参考官方文档基础库示例中的“数字输出”。运行这段程序会出现的效果是继电器随着 L 灯的闪烁会有“啪嗒”声，这就是继电器内的电磁开关在切换时发出的声音。



（2）用继电器控制水泵

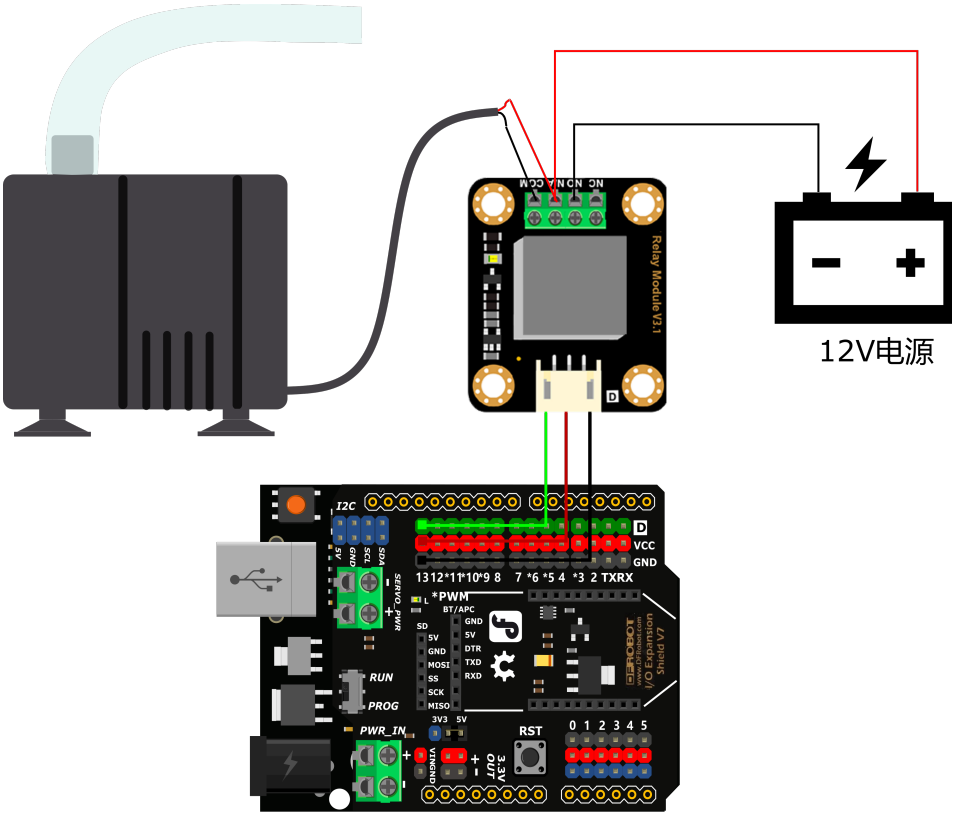
要实现浇花的功能那么就会需要使用到水泵来帮我们实现，但是水泵的工作电压大多是 12V 的，但是我们使用的 arduino uno 的输出电压是 5v，无法直接驱动水泵，这时我们就需要借助继电器来驱动水泵了。

硬件准备：

主控：Arduino UNO、IO 传感器扩展板 V7.1

模块：继电器模块、水泵、12V 电源

连接线：TypeAtoB 方口 USB 连接线



- 将继电器接入 13 号数字接口
- 水泵与继电器连接

程序编写

1. 浇花需要每隔一定的时间再执行，那如何控制间隔时间呢？用之前项目中最常用的 time.sleep () 行不行？用延时功能来控制是可以实现功能的，但是无法精确的定位在指定的时间，限制非常多，每次重启程序延时的时间就会重新计算，影响浇水的效率。
2. 这里我们将使用 time 函数的 time.strftime() 功能来定位时间，如示例中，在每天下午 15 点 30 分 10 秒的时候打印“浇花”。在实际使用中将继电器执行功能加进来就可以了。

```
import time
while True:
time_now = time.strftime("%H:%M:%S", time.localtime())
if time_now == "15:30:10":
print("浇花")
time.sleep(1)
```

3. 加入继电器实现每天定时浇水功能。

```
import time
```

(下页继续)

(续上页)

```
from pinpong.board import Board,Pin

Board("uno").begin()
led = Pin(Pin.D13, Pin.OUT)
while True:
    time_set = time.strftime("%H:%M:%S",time.localtime())
    print(time_set)
    if time_set == "15:30:10":
        led.write_digital(1)
        print("浇花")
        time.sleep(5)
    else:
        led.write_digital(0)
        time.sleep(1)
```

8.11.3 三、代码分析

```
import time
from pinpong.board import Board,Pin

Board("uno").begin() # 初始化，选择板型（uno、leonardo、xugu）和端口号，不输入端口号则进行自动识别
led = Pin(Pin.D13, Pin.OUT) # 初始化继电器引脚在 D13

while True:
    time_set = time.strftime("%H:%M:%S",time.localtime())# 刷新时间
    print(time_set)
    if time_set == "15:30:10": # 设定每天 15:30:10 浇水
        led.write_digital(1)# 输出高电平，继电器吸合
        print("浇花")# 终端打印信息
        time.sleep(5)# 等待 5 秒 保持状态
    else:
        led.write_digital(0)# 输出低电平
        time.sleep(1)# 等待 1 秒 保持状态
```

time 库函数

time 库在我们之前的案例中一直有使用到的一个库，但是我们大多数情况只用到了其中的延时的功能，其实还有很多功能，让我们来了解一下吧。

time 库是 python 中处理时间的标准库

1. time 库的使用

```
时间获取-------time() ctime()  localtime()
时间格式化-------strftime()  strptime()
程序计时-------sleep()  perf_counter()
```

2. 时间获取函数

```
time()-------获取当前时间戳，浮点数形式
ctime()-------以可读的方式返回字符串时间
localtime()-------计算机可以处理的时间格式
```

3. 时间格式化

```
strftime()--------将时间进行合理输出
strptime()-------自定义时间
```

4. 程序计时

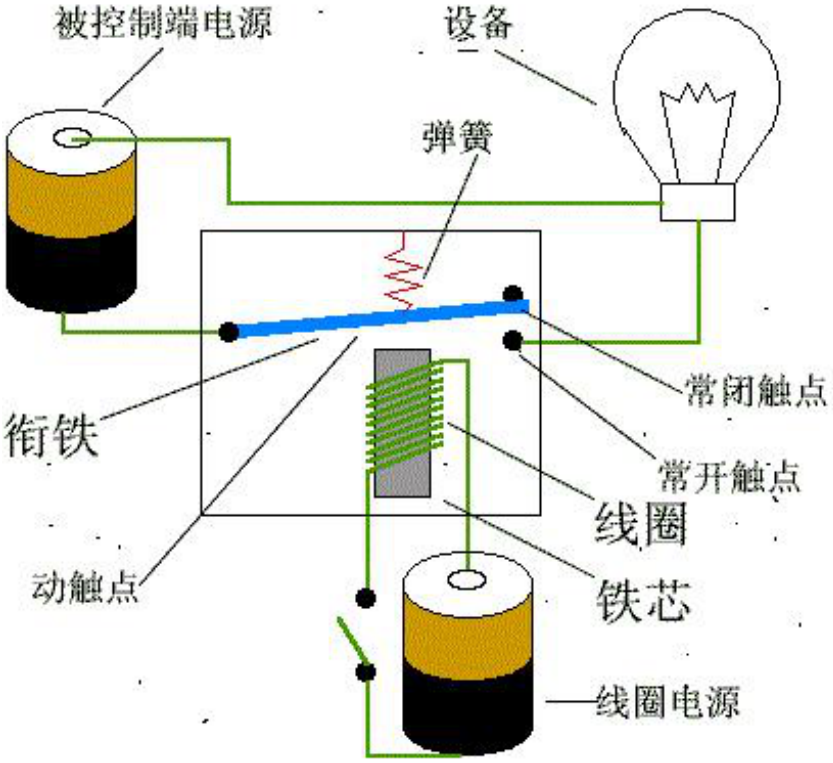
```
perf_counter()-------测量时间函数
sleep()-------产生时间函数，模拟休眠的时间，单位是秒，可以是浮点数
```

8.11.4 四、硬件分析

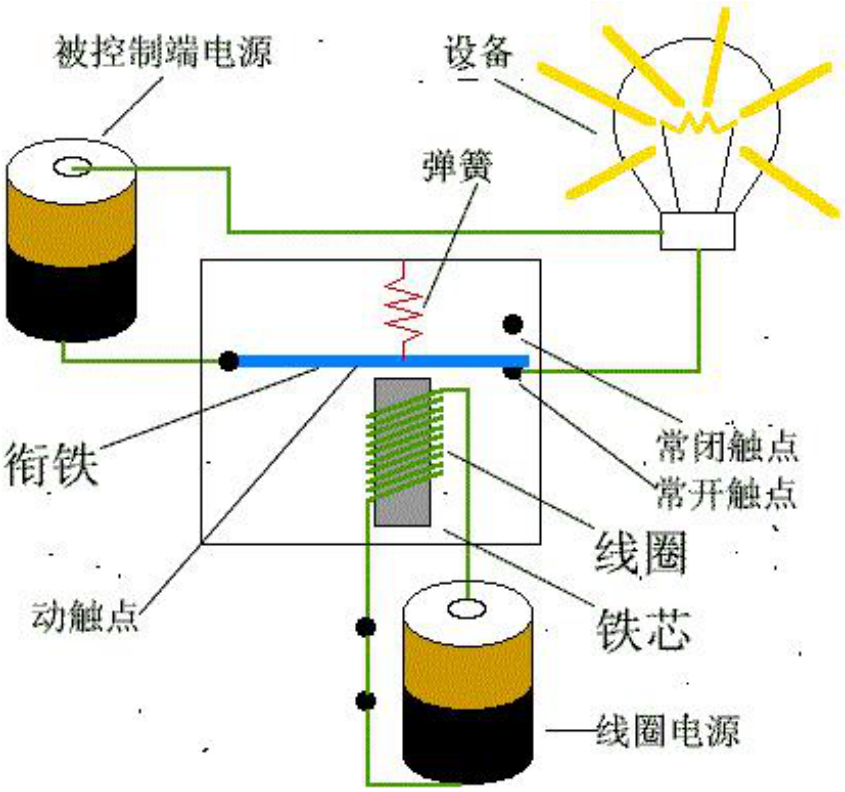
继电器

继电器（英文名称：relay）是一种电控制器件，是当输入量（激励量）的变化达到规定要求时，在电气输出电路中使被控量发生预定的阶跃变化的一种电器。它具有控制系统（又称输入回路）和被控制系统（又称输出回路）之间的互动关系。通常应用于自动化的控制电路中，它实际上是用小电流去控制大电流运作的一种“自动开关”。故在电路中起着自动调节、安全保护、转换电路等作用。

光看文字描述有点难以理解，下面看看图示帮助理解继电器的作用。继电器工作原理图：



这里有两个电源回路，一个是线圈部分的回路另一个是设备（灯珠）端的回路。我们把线圈铁芯这端理解成掌控板这端，设备（灯珠）端我们理解成水泵；线圈电源就是掌控板的输出电压 5V，被控制端电源电压就是我们驱动水泵的 12V 电源。我们通过 5V 的电压控制线圈铁芯这个电磁铁，当继电器高电平时，线圈通电，将衔铁吸引，这样设备端的回路就接通了。



继电器的作用

继电器是具有隔离功能的自动开关元件，广泛应用于遥控、遥测、通讯、自动控制、机电一体化及电力电子设备中，是最重要的控制元件之一。

继电器一般都有能反映一定输入变量（如电流、电压、功率、阻抗、频率、温度、压力、速度、光等）的感应机构（输入部分）；有能对被控电路实现“通”、“断”控制的执行机构（输出部分）；在继电器的输入部分和输出部分之间，还有对输入量进行耦合隔离，功能处理和对输出部分进行驱动的中间机构（驱动部分）。

作为控制元件，概括起来，继电器有如下几种作用：

1）扩大控制范围：例如，多触点继电器控制信号达到某一定值时，可以按触点组的不同形式，同时换接、开断、接通多路电路。

2）放大：例如，灵敏型继电器、中间继电器等，用一个很微小的控制量，可以控制很大功率的电路。

3）综合信号：例如，当多个控制信号按规定的形式输入多绕组继电器时，经过比较综合，达到预定的控制效果。

4）自动、遥控、监测：例如，自动装置上的继电器与其他电器一起，可以组成程序控制线路，从而实现自动化运行。

CHAPTER 9

高级教程

9.1 高级教程

教程编写中，如果你想参与，请联系我们。

CHAPTER 10

经典案例

10.1 虚谷号案例

- 虚谷号入门使用教程：厚物扩展板
- 虚谷号案例：百科相机

CHAPTER 11

PinPong 类 Board class

11.1 构造器 constructor

创建和初始化一块板子。

board = PinPong(board_name,port)

- boardname: 板子的类型。
- port: 设置对应的端口号，如省略参数则自动识别选择一个端口。如何查询端口号，见安装及快速开始教程。对于基于 linux 的开发板, 例如树莓派，哪吒板，不需要这个参数
- 说明：board_name 命名列表
- 规则：

1. 简单短小好记忆
2. 不区分大小写，样例中推荐全小写
3. 长单词或需要区分版本的单词间使用下划线区分

板子名称	board_name	备注
Arduino uno r3	uno	
Arduino leonardo	leonardo	
Arduino nano	nano	
Arduino mega1280	mega1280	暂未支持
Arduino mega2560	mega2560	暂未支持
micro:bit	microbit	
掌控板	handpy	
虚谷号	xugu	等效于 uno
树莓派	rpi	
Firebeetle ESP32	firebeetle_esp32	暂未支持
Firebeetle ESP8266	firebeetle_esp32	暂未支持
LattePanda	lp	等效于 leonardo
ESP32	esp32	暂未支持
ESP8266	esp8266	暂未支持
哪吒开发板	nezha	

11.2 方法 method

```
board.begin()
```

连接 arduino 板，检测固件版本，若没有烧录固件或版本有误，会烧录最新固件。

CHAPTER 12

Pin 类 Pin class

12.1 常量 constants

12.2 构造器 constructor

创建和初始化一个引脚。

pin = Pin(board, vpin, mode)

- board: 通过 pinpong 类创建的板子的对象, 只有一块板子时可以省略此参数。
- vpin: 板子上所用到的引脚号。(数字引脚 1-Pin.D1, 模拟引脚 1-Pin.A1)
- mode: 定义引脚的输入、输出。Pin.IN，Pin.OUT (定义模拟量引脚时省略，默认输入)

定义一个数字输入，比如按键，

button_pin = (Pin.D8, Pin.IN)

定义一个模拟量传感器，

Analog_pin = (Pin.A0)

12.3 方法 method

```
pin.value()

调用 value(), 没有提供 args 时, 为数字读取, 返回 0 或 1。

v = button_pin.value()//获取引脚 button_pin 的引脚状态

pin.value(x)

调用 value(), 提供 args 时, 为数字写入。

pin.value(1) //引脚 pin 输出高电平
```

```
pin.on()
引脚 pin 设置为高电平, 同 pin.value(1)
```

```
pin.off()
引脚 pin 设置为低电平, 同 pin.value(0)
```

```
pin.irq(trigger,handler)
设置中断,
- trigger: 中断模式, rising - 上升沿, falling - 下降沿, low - 低电平, high - 高电平...
- handler:
```

CHAPTER 13

ADC 类 ADC class

13.1 构造器 constructor

创建和初始化一个 ADC, 获取模拟量数据。

adc = ADC(board,Pin(board, Pin.A0))

- board: 通过 pinpong 类创建的板子对象, 只有一块板子时可以省略此参数。
- Pin: 通过 pin 类创建的引脚对象。

13.2 方法 method

```
adc.read()
读取 adc 的模拟量。比如, brightness = adc.read()
```

CHAPTER 14

PWM 类 PWM class

14.1 构造器 constructor

创建和初始化一个 PWM, 来输出 PWM 信号。

pwm0 = PWM(board, Pin(board, Pin.D6))

- board: 通过 pinpong 类创建的板子对象, 只有一块板子时可以省略此参数。
- Pin: 通过 pin 类创建的引脚对象。

14.2 方法 method

```
pwm0.freq()
freq(), 没有提供 args 时, 返回频率。
f = pwm0.freq()//获取 PWM 频率。
pwm0.freq(x)
freq(x), 提供 args 时, 设置 PWM 频率为 x。
pwm0.freq(1000)//设置 PWM 频率为 1000。
```

```
pwm0.duty()
duty(), 没有提供 args 时, 返回占空比。
f = pwm0.duty()//获取 PWM 占空比。
```

(下页继续)

(续上页)

```
pwm0.duty(x)
duty(x), 提供 args 时, 设置 PWM 占空比, 范围 0-255。
pwm0.duty(127)//设置 PWM 占空比为 50%。
```

```
pwm0.deinit()
```

CHAPTER 15

开发计划

- 这里将记录一些正在开发或计划开发的内容，如果你有什么建议可至本文档的 github 留言

1. micro:bit 板载元件 api 调整为与 microPython 相同。
2. arduino nano 的支持。
3. 其他更多扩展模块的支持。

CHAPTER 16

更新记录

16.1 V0.4.2 公测版 20210820

1. 重构 handpy 板载传感器的调用接口为 micropython 语法
2. 增加 handpy、micro:bit 对外接传感器模块的支持
3. 增加哪吒板的支持
4. 解决发现的 bug

16.2 V0.3.5 公测版 20210407

1. 支持传感器 12 位 DA 转换模块
2. 支持传感器 bmi160 6 轴传感器
3. 支持 RGB LED 点阵表情包
4. 支持实时时钟 SD2405
5. 支持 TM1650 四位数码管
6. 支持 SHT31 数字温度传感器
7. 支持臭氧传感器
8. 电容式指纹识别传感器

9. 解决发现的 bug

16.3 V0.3.4 公测版 20201231

1. 解决 uno、leonardo pwm 问题
2. 新增 microbit 板载资源支持
3. 新增 handpy 板载资源支持
4. 新增 bme680 支持（环境传感器）
5. 新增 max17043 支持（3.7V 锂电池电量计）

16.4 V0.3.3 公测版 20201116

1. 支持 DS18B20（温度传感器）传感器
2. 支持音频分析模块
3. 支持 HX711（重量传感器）传感器
4. 支持 TDS（测量水的 TDS）传感器
5. 支持心率传感器
6. 支持 BME280（环境传感器）传感器
7. 支持 VL53L0（激光测距）传感器
8. 支持 LIS2DH（三轴加速计）传感器
9. 支持 BMP388（气压传感器）传感器
10. 支持 INA219（数字功率计）传感器
11. 支持 CCS811（空气质量）传感器
12. 支持 ADS1115（16 位 AD 转换模块）传感器
13. 支持 huskylens（哈士奇）

16.5 V0.3.2 公测版 20200929

1. 增加树莓派、掌控板 (handpy) 以及 micro:bit 板的支持（当前仅支持数字模拟读写及舵机控制功能）。
2. 新增扩展库：BNO05510DOF 绝对定向传感器、BMP280 气压传感器、NFC 模块、PAJ7620U2 手势识别模块、DS1307 实时时钟模块、RPI TLC_10bit_adc 树莓派 10 位 adc 模块、PRI pca_9685 树莓派 16 路 pwm 模块。

16.6 V0.3.0 公测版 20200727

注意: 重要版本：API 重做最终版本，后续 API 将不再进行修改，可用于教程编写。

1. PinPong 类名替换为 Board 类名便于理解
2. 初始化方式修改，使用 Board 及 begin() 函数
3. 增加默认板，无需给所有函数输入 board 参数
4. 将常用库放入 Board 中，导入更方便

16.7 V0.2.2 公测版 20200715

1. Pin 类增加基础函数:
2. read_digital()
3. write_digital(v)
4. read_analog()
5. write_analog(v)
6. 详细用法见 “PinPong 示例”。

16.8 V0.2.0 公测版 20200625

1. 语法及库名称规范完成
2. 增加常见扩展硬件库的支持
3. 增加自动识别串口功能

16.9 V0.1.x 内测版

:: 功能开发及内测版本，仅供测试，代码与 V0.2.0 可能存在不兼容.

CHAPTER 17

常见问题

17.1 虚谷号上出现错误：ImportError: cannot import name ‘Pin’

1. 尝试先重启虚谷号
2. 重启后如果依然出现，则先尝试更新 pinpong 库，终端运行: pip install -U pinpong
3. 如依然出现，则尝试卸载重装 pinpong 库, 终端运行: pip uninstall pinpong ，然后重启虚谷号后重新安装: pip install pinpong

17.2 使用 pinpong 库控制的硬件可以脱离电脑运行吗？

1. pinpong 是一个 Python 库，主要实现硬件与 Python 的交互，因此只有能运行 Python（注意: Python 和 microPython 不一样）的设备才可以使用，因此如果要脱离 PC，可以使用树莓派、LattePanda 等便携式单板计算机。
2. 使用实时通信的方式与 Python 交互运行，后续将考虑开发通过网络的方式进行通信以实现在电脑上运行 pinpong（Python），通过网络实时控制带网络功能的硬件设备。

17.3 运行程序时出现 SerialExcedption: could not open port ‘COM’: PermissionError(13,’拒绝访问。’,None,5 怎么办？

1. 此错误说明有设备占用了串口，或上一次程序没有正常关闭，或当前识别的串口错误。

2. 如果是以前可以正常使用而本地运行无法使用则一般为串口被占用了，则可以尝试关闭上一次的程序及其他占用串口的软件，或重新插拔 USB 口及重启电脑。
3. 如果是第一次使用这个主控板，则可能为串口没有安装驱动，需要安装一次驱动（推荐使用 Mind+ 内置的一键安装驱动功能），或打开设备管理器查看是否有端口（COM）。

CHAPTER 18

索引

- genindex
- modindex
- search