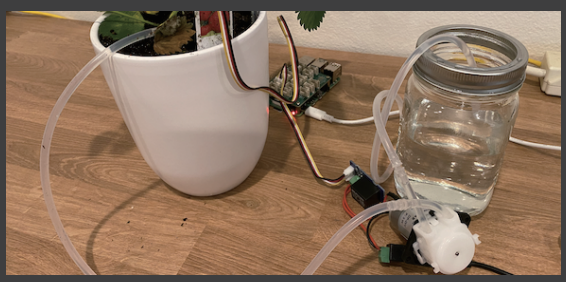
农场篇

随着人口的增长，人们对农业的需求也在增加。可用的土地数量不会改变，但气候会改变——这给农民带来更大的挑战，特别是 20 亿自给自足的农民，他们依靠自己种植的东西得以吃饭和养家。物联网可以帮助农民提高产量，在种植什么和何时收获方面做出更明智的决定，减少人工劳动量，并检测和处理虫害。

在这六节课中，你将学习如何应用物联网来改善农业并实现自动化。



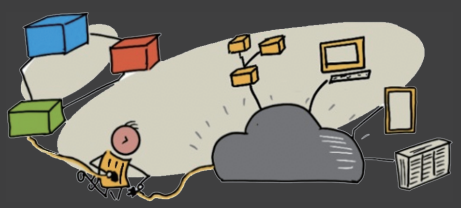
课程简介

本篇包含以下课程：

第 5 课　用物联网预测植物生长
第 6 课　检测土壤水分
第 7 课　自动浇灌植物
第 8 课　将你的种植数据迁移到云端
第 9 课　将你的应用逻辑迁移到云端
第 10 课　确保你的植物安全

感谢

本篇所有的课程都是由 Jim Bennett 用♥编写。



第 5 课
用物联网预测植物生长

课前准备

简介

植物需要满足某些条件才能正常生长——水分、养分、二氧化碳，以及光和热。在本课中，你将学习如何通过测量空气温度来得出植物的生长率和成熟率。

在本课中，我们将介绍：

5.1 数字农业
5.2 为什么温度对耕作很重要
5.3 测量环境温度
5.4 生长度日 (GDD)
5.5 使用温度传感器数据计算 GDD

课前测验

（1）IoT 设备可以用来支援农业。这个说法(　)。
A. 正确　　B. 错误

（2）植物需要的东西包括（　）（请选择最佳选项）。
A. 二氧化碳、水、养分
B. 二氧化碳、水、养分、光照
C. 二氧化碳、水、养分、光照、适宜的温度

（3）传感器对于发达国家的农民来说过于昂贵。这个说法（　）。
A. 错误　　B. 正确

5.1 数字农业

数字农业不同于我们传统的耕作方式，它使用工具来收集、存储和分析耕作数据。我们目前正处于被世界经济论坛描述为“第四次工业革命”的时期，而数字农业的崛起也被称为“第四次农业革命”或“农业 4.0”。

> 数字农业一词还包括整个“农业价值链”，即从农场到餐桌的整个过程。它包括在食品运输和加工过程中跟踪农产品质量，仓储和电子商务系统，甚至是拖拉机租赁应用程序！

这些变化使农民能够提高产量，减少化肥和农药用量，并且更有效地用水。虽然数字农业暂时主要被部署于较富裕的国家，但随着传感器和其他设备的价格慢慢降低，它们也变得更容易被发展中国家使用。

下面是促成数字农业的一些技术。

- **温度测量：**测量温度，使农民能够预测植物的生长和成熟度。
- **自动灌溉：**测量土壤湿度，在土壤过于干燥时打开灌溉系统，而不是定时浇水。定时浇水可能导致作物在高温干旱时无法及时获得水分，或在下雨时仍然进行灌溉。通过监测土壤湿度，可以只在土壤需要时才进行灌溉，农民可以使水资源被更充分地利用。
- **虫害控制：**农民可以使用自动机器人或无人机上的摄像头来检查虫害，然后只在需要的地方施用杀虫剂，减少杀虫剂的使用量，并减少杀虫剂流入当地水源。

> “精准农业”一词被用于定义在一块田地，甚至是在一块田地的部分区域进行独立的观察、测量和

响应。其中包括测量土壤水分、养分和害虫水平，并做出准确的响应，如根据各区域土壤湿度的不同而只对一小部分田地进行灌溉。

做点研究

还有哪些技术是用来提高农业产量的？

5.2 为什么温度对耕作很重要

在学习植物相关知识时，大多数学生被告知水、光、二氧化碳（CO_2）和养分的必要性。植物的生长也需要适宜的温度——这就是为什么植物在春天随着温度的升高而开花，为什么雪莲花或水仙能因短暂的暖流而提前发芽，以及为什么暖房和温室能使植物更好生长的原因。

暖房和温室的作用类似，但有一个重要区别：暖房是人工加热的，使农民能够更准确地控制温度，而温室依靠太阳取暖，通常控制温度的唯一手段是使用窗户或其他通风开口，控制热量的散发。

植物有一个基点温度（或最低温度）、最佳温度和最高温度，所有这些都是基于日平均温度而言的。

- **基点温度**：这是植物生长所需的最低日平均温度。
- **最佳温度**：这是获得最多生长的最佳日平均温度。
- **最高温度**：这是植物能够承受的最高温度。超过这个温度，植物就会停止自身生长以节省水分，维持自身存活。

这些是平均温度，是每日和每夜温度的平均值。植物在昼夜也需要不同的温度，以帮助它们在白天更有效地进行光合作用，并在夜间节省能量。

做点研究

看看你能否找到你在花园、学校或当地公园中见到的某种植物的基点温度。

每种植物的基点温度、最佳温度和最高温度都不同。这就是一些植物在炎热的国家茁壮成长，而另一些则在寒冷的国家能够更快生长的原因。

图 5-1 所示为一个生长速度与温度关系图的例子。在基点温度之前，植物不会生长。生长速度随着温度逐渐上升，直到到达最佳温度，然后在达到这个峰值后开始下降。在最高温度时，生长停止。

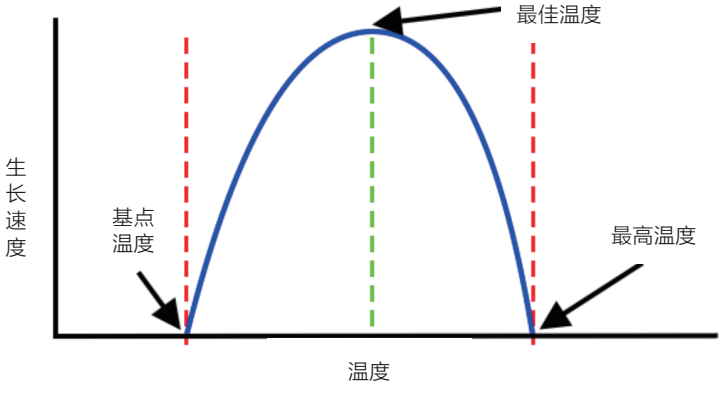


图 5-1　一张显示生长速度随着温度的升高而上升，然后随着温度继续升高而下降

该图的曲线形状因植物种类而异。有些植物在超过最佳温度后生长速度有较明显的下降，而有些植物从基点温度到最佳温度的增长较为缓慢。

农民要想获得植物最佳的生长速度，他们就需要知道植物的三个温度值，并了解他们所种植的植物的曲线形状。

如果农民能够控制温度，如在商业暖房中，那么他们就可以对植物的生长速度进行优化。例如，一个种植西红柿的商业暖房会在白天将温度设置为 25℃左右，晚上设置为 20℃，以获得最快的生长速度。

> 将这些温度与人工照明、肥料和受控的二氧化碳水平相结合，意味着商业种植者可以实现全年种植和收获。

5.3 测量环境温度

温度传感器可与物联网设备一起使用，测量环境温度。

任务：测量温度

按照相关教程做一遍，使用你的物联网设备监测温度。

- 用 Wio Terminal 测量温度。
- 用树莓派测量温度。
- 用虚拟物联网硬件测量温度。

连接设备：用传感器测量温度

用 Wio Terminal 测量温度

在这一部分，你将在 Wio Terminal 上外接一个温度传感器，并从中读取温度值。

硬件

Wio Terminal 需要一个温度传感器。

你将使用的传感器是一个 SHT40 型号的温湿度传感器 [L5-1]，如图 5-2 所示。此传感器模块合并了两个传感器，这在传感器中是比较常见的，有许多商业用的传感器整合了温度、湿度甚至还有气压等多种测量。SHT40 温湿度传感器是使用热电偶法来测量温度的。热电偶由两种不同材料的金属丝组成。两根导线的一端焊接在一起形成工作端，工作端接触待测量温度的环境；另一端称为自由端，与主控制器连接形成闭环。当工作端与自由端的温度不同时，回路中会出现一个热电动势，这个电压的变化将通过电路转换传送到单片机，并转换成机器能够识别的数字信号。

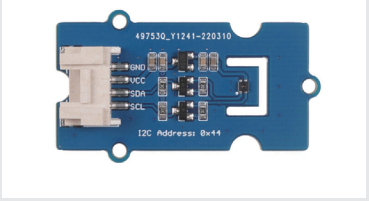


图 5-2　一个 Grove 的 SHT40 型温湿度传感器

SHT40 温湿度传感器使用的是 I^2C 的通信协议，所以需要使用板载的 I^2C，接收包含温度和湿度的数据信息。

连接温度传感器

Grove 温湿度传感器可以连接到 Wio Terminal 的数字端口。

任务：连接温度传感器

连接温湿度传感器。

（1）如图 5-3 所示，将 Grove 电缆的一端插入温湿度传感器的插


图 5-3　连接到 Wio Terminal 右边插座的 Grove 温湿度传感器

口，它只能从一个方向插入。

（2）在 Wio Terminal 与计算机或其他电源断开的情况下，当你看屏幕时，将 Grove 电缆的另一端连接到 Wio Terminal 右侧的 Grove 插座上。这是离电源按钮最远的一个插座。

对温湿度传感器进行编程

现在可以对 Wio Terminal 进行编程，以使用附加的温湿度传感器。

任务：对温湿度传感器进行编程

（1）使用 PlatformIO 创建一个全新的 Wio Terminal 项目。我们把这个项目称为 `temperature-sensor`。在 `setup` 函数中添加代码，配置串口。

⚠ 注意

如果需要的话，你可以参考第 1 课中创建 PlatformIO 项目的说明。

（2）在项目的 `platformio.ini` 文件中添加 Seeed Grove 温湿度传感器库的依赖库。

```
platform_packages = framework-arduino-samd-seeed@https://github.com/Seeed-Studio/ArduinoCore-samd.git
lib_deps =
    https://github.com/Sensirion/arduino-core
    https://github.com/Sensirion/arduino-i2c-sht4x
```

⚠ 注意

如果需要的话，你可以参考第 4 课中关于向 PlatformIO 项目添加库的说明。

（3）在文件的顶部，在现有的 `#include <Arduino.h>` 下面添加以下 `#include` 指令。

```
#include <SensirionI2CSht4x.h>
#include <Wire.h>
```

这样就导入了与传感器交互所需的文件，`SensirionI2CSht4x.h` 头文件包含了传感器本身的代码，而添加 `Wire.h` 头文件可以确保在编译应用程序时，与传感器对话所需的代码被链接进来。

（4）在 `setup` 函数之前，声明 SHT 传感器。

```
SensirionI2CSht4x sht4x;
```

这样声明了一个管理温湿度传感器的 `SensirionI2CSht4x` 类的实例。它被连接到板载的 I^2C 端口，即 Wio Terminal 左侧的 Grove 插座。

（5）在 `setup` 函数中，添加以下代码来设置串行连接。

```
void setup()
{
    Serial.begin(9600);

    while (!Serial)
```

```
        ; // Wait for Serial to be ready

    delay(1000);
}
```

（6）在 `setup` 函数最后的 `delay`（延迟）之后，添加下面的代码来检查传感器是否有异常，并启动 SHT 传感器。

```
    Wire.begin();

    uint16_t error;
    char errorMessage[256];

    sht4x.begin(Wire);

    uint32_t serialNumber;
    error = sht4x.serialNumber(serialNumber);
    if (error) {
        Serial.print("Error trying to execute serialNumber(): ");
        errorToString(error, errorMessage, 256);
        Serial.println(errorMessage);
    } else {
        Serial.print("Serial Number: ");
        Serial.println(serialNumber);
    }
```

这段代码声明了一个无符号 16 位的变量 `error`、一个能存放 256 个字符的数组 `errorMessage`，以及一个无符号 32 位的变量 `serialNumber`。变量 `error` 用于存放温湿度传感器的错误代码编号；数组 `errorMessage` 的作用是存放错误信息，输出给用户；变量 `serialNumber` 则表示当前所使用的串行端口编号。

（7）在 `loop` 函数中，添加调用传感器的代码，并将温度打印到串行端口。

```
void loop()
{
    uint16_t error;
    char errorMessage[256];

    delay(1000);

    float temperature;
    float humidity;
    error = sht4x.measureHighPrecision(temperature, humidity);
    if (error) {
        Serial.print("Error trying to execute measureHighPrecision(): ");
        errorToString(error, errorMessage, 256);
        Serial.println(errorMessage);
    } else {
        Serial.print("Temperature:");
        Serial.print(temperature);
    }
}
```

这段代码同样声明了一个变量 `error` 和数组 `errorMessage`。它们的作用与 `setup` 函数中使用它们时的作用一样，如果在传感器使用过程中发生了异常，能够让用户及时了解错误信息。

后面声明的两个变量 `temperature` 和 `humidity` 是浮点数。它们通过使用函数 `sht4x.measureHighPrecision()`，将获得的传感器温度和湿度值分别传递到这两个变量中。最后，将温度的值打印到串行端口。

（8）建立并上传代码到 Wio Terminal。

（9）一旦上传，你就可以用串行监视器监测温度了。

⚠ 注意

如有需要，你可以参考第 1 课中创建 PlatformIO 项目的说明。

```
> Executing task: platformio device monitor <

--- Available filters and text transformations: colorize, debug, default, direct, hexlify,
log2file, nocontrol, printable, send_on_enter, time
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/cu.usbmodem1201  9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Temperature: 25.00°C
Temperature: 25.00°C
Temperature: 25.00°C
Temperature: 24.00°C
```

你可以在 `code-temperature/wio-terminal` 文件夹 🔗 `[L5-2]` 中找到这段代码。

😀 恭喜，你的温度传感器程序很成功。

用树莓派测量温度

在这一部分，你将在树莓派上外接一个温度传感器。

硬件

你将使用的传感器是一个 DHT11 型号的温湿度传感器（合并了两个传感器🔗 `[5-3]`），如图 5-4 所示。这在传感器中是比较常见的，有许多商业用的传感器整合了温度、湿度甚至还有气压等多种测量。温度传感器组件是一个负温度系数（NTC）热敏电阻，这种热敏电阻的电阻值随着温度升高而降低。

这是一个数字传感器，所以有一个板载的 ADC，用来创建一个微控制器可以读取的包含了温度和湿度数据的数字信号。

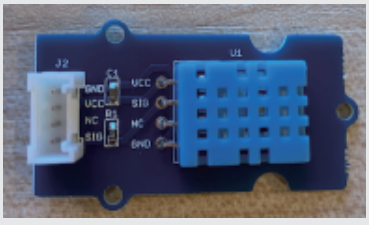


图 5-4　一个 Grove DHT11 型号温湿度传感器

连接温度传感器

将 Grove 温湿度传感器连接到树莓派上。

任务：连接温度传感器

（1）如图 5-5 所示，将 Grove 电缆的一端插入温湿度传感器的插口，它只能从一个方向插入。

（2）在树莓派电源关闭的情况下，将 Grove 电缆的另一端连接到树莓派 Grove Base HAT 上标有 D5 的数字插座上。这个插座在 GPIO 引脚旁边的那排插座上从左边数第二个接口。

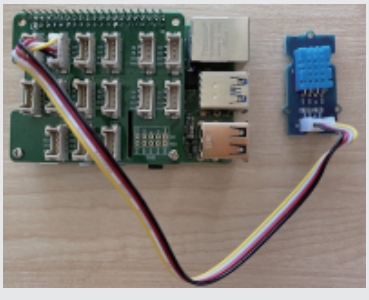


图 5-5　连接到 A0 插座的 Grove 温湿度传感器

对温湿度传感器进行编程

现在可以对设备进行编程以使用所连接的温湿度传感器。

任务：对温湿度传感器进行编程

（1）给树莓派上电并等待它启动。

（2）启动 VS Code，可以直接在树莓派上启动，也可以通过远程 SSH 扩展连接。

（3）在终端上，在树莓派用户的主目录下创建一个新的文件夹，名为 temperature-sensor。在这个文件夹中创建一个名为 app.py 的文件。

> **⚠ 注意**
>
> 如果需要，你可以参考第 1 课中关于设置和启动 VS Code 的说明。

```
mkdir temperature-sensor
cd temperature-sensor
touch app.py
```

（4）在 VS Code 中打开这个文件夹。

（5）为了使用温湿度传感器，需要安装一个额外的 Pip 包。在 VS Code 中的终端运行下面的命令，在树莓派上安装这个 Pip 包。

```
pip3 install seeed-python-dht
```

（6）在 app.py 文件中添加以下代码，导入所需的库。

```
import time
from seeed_dht import DHT
```

`from seeed_dht import DHT` 语句导入了 `DHT` 传感器类，以便与 `seed_dht` 模块中的 Grove 温湿度传感器进行交互。

（7）在上面的代码后面添加以下代码，创建一个管理温湿度传感器的类的实例。

```
sensor = DHT("11", 5)
```

这行代码声明了一个管理数字温湿度传感器的 DHT 类的实例。第一个参数告诉代码正在使用的传感器是 DHT11 传感器——你所使用的库支持该传感器的其他型号。第二个参数告诉代码，传感器被连接到 Grove 底座上的数字端口 `D5`。

> 注意：所有的插座都有唯一的引脚编号。其中，0、2、4、6 号引脚是模拟引脚；5、16、18、22、24、26 号引脚是数字引脚。

（8）在上面的代码后添加一个无限循环，轮询温度传感器的值，并将其打印到控制台。

```
while True:
    _, temp = sensor.read()
    print(f'Temperature {temp}°C')
```

对 `sensor.read()` 的调用返回一个湿度和温度的元组。因为你只需要温度值，所以湿度被忽略了。然后温度值会被打印到控制台。

（9）在循环结束时添加一个 10 秒的睡眠，这是因为温度值不需要连续检查。睡眠可以减少设备的功率消耗。

```
time.sleep(10)
```

在 VS Code 终端，通过运行以下程序来运行你的 Python 应用程序。

```
python3 app.py
```

你应该看到温度值被输出到了控制台。用一些东西来加热传感器，如用拇指按住它，或者用风扇吹一吹，同时观察数值的变化。

```
pi@raspberrypi:~/temperature-sensor $ python3 app.py
Temperature 26°C
Temperature 26°C
Temperature 28°C
```

你可以在 code-temperature/pi 文件夹 🔗 [L5-4] 中找到这个代码。

> 😀 恭喜，你的温度传感器程序运行成功了！

用虚拟设备测量温度

在这一部分，你将在你的虚拟物联网设备上添加一个温度传感器。

虚拟硬件

虚拟物联网设备将使用一个模拟的 Grove 数字湿度和温度传感器。这使得本实验与使用树莓派和物理的 Grove DHT11 传感器保持一致。

该传感器结合了温度传感器和湿度传感器，但在本实验室中，你只对温度传感器部分感兴趣。在物理物联网设备中，温度传感器将是一个热敏电阻，通过感应温度变化时的电阻变化来测量温度。温度传感器通常是数字传感器，在内部将测量的电阻转换为摄氏度（或开尔文、华氏度）的温度。

添加传感器到 CounterFit

要使用虚拟湿度和温度传感器，你需要将这两个传感器添加到 CounterFit 应用程序中。

⚠ 注意

如果需要，你可以参考第 1 课中创建和设置 CounterFit Python 项目的说明。

任务：将传感器添加到 CounterFit 中

将湿度和温度传感器添加到 CounterFit 应用程序中。

（1）在你的计算机上创建一个新的 Python 应用程序，文件夹名为 `temperature-sensor`，其中有一个名为 `app.py` 的文件和一个 Python 虚拟环境，并添加 CounterFit Pip 软件包。

（2）安装一个额外的 Pip 包，为 DHT11 传感器安装一个 CounterFit Shim，确保你是在激活了虚拟环境的终端上安装的。

```
pip install counterfit-shims-seeed-python-dht
```

（3）确保 CounterFit 的网络应用程序正在运行。

（4）创建一个湿度传感器，如图 5-6 所示。

①在“Sensor”（传感器）窗格的“Creat sensor”框中，下拉“Sensor Type”（传感器类型）框，选择“Humidity”（湿度）。

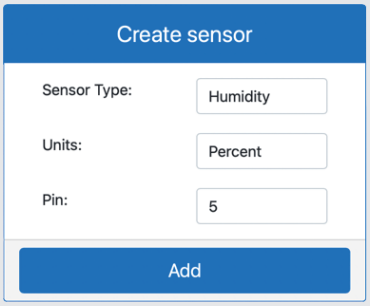


图 5-6　湿度传感器的设置

②将单位设置为“Percent”（百分比）。

③确保“Pin”（引脚）被设置为 5。

④单击“Add”（添加）按钮，在引脚 5 上创建湿度传感器。

湿度传感器将被创建并出现在传感器列表中，如图 5-7 所示。

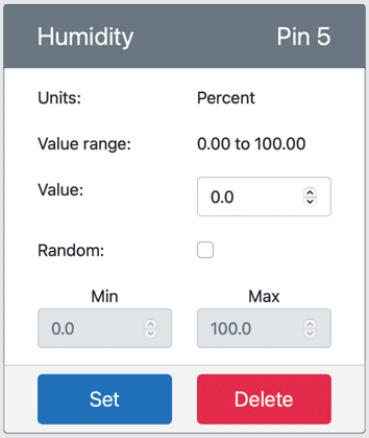


图 5-7　创建的湿度传感器

（5）创建一个温度传感器，如图 5-8 所示。

①在“Sensor”（传感器）窗格的“Creat sensor”框中，下拉“Sensor Type”（传感器类型）框，选择“Temperature”（温度）。

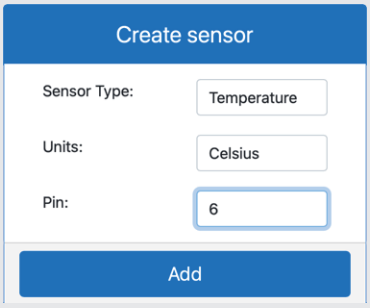


图 5-8　温度传感器的设置

②将单位设置为“Celsius”（摄氏度）。

③确保引脚被设置为 6。

④单击“Add”（添加）按钮，在引脚 6 上创建温度传感器。

温度传感器将被创建并出现在传感器列表中，如图 5-9 所示。

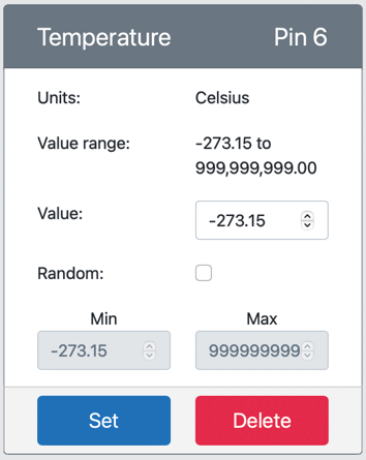


图 5-9　创建的温度传感器

对温度传感器应用程序进行编程

现在可以使用 CounterFit 传感器对温度传感器应用程序进行编程。

任务：为温度传感器应用程序编程

对温度传感器应用程序进行编程。

（1）确保在 VS Code 中打开 `temperature-sensor` 应用程序。

（2）打开 `app.py` 文件。

（3）在 `app.py` 的顶部添加以下代码，将应用程序连接到 CounterFit。

```
from counterfit_connection import CounterFitConnection
CounterFitConnection.init('127.0.0.1', 5000)
```

（4）在 `app.py` 文件中添加以下代码以导入所需的库。

```
import time
from counterfit_shims_seeed_python_dht import DHT
```

`from seeed_dht import DHT` 语句导入 `DHT` 传感器类，以便与使用 `counterfit_shims_seeed_python_dht` 模块 shim 的虚拟 Grove 温度传感器进行交互。

（5）在上面的代码之后添加以下代码，以创建一个管理虚拟湿度和温度传感器的类的实例。

```
sensor = DHT("11", 5)
```

这行代码声明了一个管理虚拟数字湿度和温度传感器的 DHT 类的实例。第一个参数告诉代码正在使用的传感器是一个虚拟的 DHT11 传感器。第二个参数告诉代码传感器被连接到端口 `5`。

> CounterFit 通过连接两个传感器来模拟这个组合的湿度和温度传感器，湿度传感器在创建 DHT 类时给定的引脚上，而温度传感器则在下一个引脚运行。

（6）在上面的代码后添加一个无限循环，轮询温度传感器的值，并将其打印到控制台。

```
while True:
    _, temp = sensor.read()
    print(f'Temperature {temp}°C')
```

对 `sensor.read()` 的调用返回一个湿度和温度的元组。因为只需要温度值，所以湿度被忽略了。然后温度值会被打印到控制台。

（7）在 `loop` 结束时添加一个 10 秒的睡眠，这是因为温度不需要连续检查。睡眠可以减少设备的功率消耗。

```
time.sleep(10)
```

（8）在激活了虚拟环境的 VS Code 终端，通过运行以下程序来运行你的 Python 应用程序。

```
python app.py
```

（9）从 CounterFit 应用程序中，改变将被应用程序读取的温度传感器的值。你可以通过以下两种方式之一来完成这项工作。

①在温度传感器的值框中输入一个数字，然后单击 `Set`（设置）按钮。你输入的数字将成为传感器返回的值。

②勾选 `Random`（随机）复选框，并输入 Min（最小）和 Max（最大值），然后单击 `Set` 按钮。这样每次传感器会读到一个随机的数字。

你应该看到你设置的值出现在了控制台。改变数值或随机设置以看到数值的变化。

```
(.venv) ➜  temperature-sensor python app.py
Temperature 28.25°C
Temperature 30.71°C
Temperature 25.17°C
```

你可以在 `code-temperature/virtual-device` 文件夹 🔗 `[L5-5]` 中找到这段代码。

> 😀恭喜，你的温度传感器程序运行成功了！

5.4 生长度日（GDD）

生长度日（Growing Degree Days，GDD）也称为生长度单位，是一种根据温度衡量植物生长情况的方法。该方法假定植物有足够的水、养分和二氧化碳，只由温度作为变量来决定其生长速度。

生长度日，是以高于植物基点温度的一天的平均温度（摄氏度）来计算的。每种植物需要一定数量的生长度日来生长、开花或结果并使作物成熟。每天的 GDD 越高，植物成熟所需要的时间就越短。

GDD 的完整公式有点复杂，但有一个简化方程，经常被用于获得近似值，即

- **GDD**：生长度日的数量；
- T_{MAX}：这是每日最高温度，摄氏度；
- T_{MIN}：这是每日最低温度，摄氏度；
- T_{BASE}：这是植物的基点温度，摄氏度。

$$GDD = \frac{T_{MAX}+T_{MIN}}{2} - T_{BASE}$$

有一些变化涉及 T_{MAX} 高于 30℃ 或 T_{MIN} 低于 T_{BASE}，但我们现在不考虑这些。

示例：玉米

根据不同的品种，玉米（或苞谷）需要 800 至 2700 GDD 才能成熟，基点温度为 10℃。

在高于基点温度的第一天，如表 5-1 所示，测量以下温度。

表 5-1　高于基点温度第一天测量的最高和最低温度

测量	温度 /℃
最高	16
最低	12

将这些数字插入到我们的计算中，有

T_{MAX} = 16

T_{MIN} = 12

T_{BASE} = 10

$$GDD = \frac{16+12}{2} - 10 = 4$$

这就得出了一个计算结果：玉米在这一天获得了 4 个 GDD。假设一个玉米品种需要 800 GDD 才能成熟，那么它还需要 796 GDD 才能达到成熟。

做点研究

对于你在花园、学校或当地公园里见到的任何植物，看看你能否找到其达到成熟所需的 GDD 数量。

5.5 使用温度传感器数据计算 GDD

植物生长所花费的时间是不确定的。例如，你无法种下一粒种子并准确预测该植物将在 100 天后开花结果。相反地，作为一个农民，你对植物需要多长时间才能完成生长会有一个粗略的概念，然后你会每天检查，根据具体情况来推测作物何时成熟。这对大型农场来说会产生巨大的劳动力负担，而且农民有可能会错过意外提前成熟的作物。通过测量温度，农民可以计算出植物每日所接受的 GDD，让它们只在接近预期成熟度时检查。

通过使用物联网设备收集温度数据，农民可以在植物接近成熟期时自动得到通知。如图 5-10 所示，这方面的典型架构是让物联网设备测量温度，然后使用类似于 MQTT（消息队列遥测传输协议）的协议在互联网上发布这些遥测数据。然后服务器代码监听这些数据并将其保存在某个地方，如数据库中。这意味着以后可以对数据进行分析，如每晚计算当天的 GDD，计算到目前为止每种作物的 GDD，并在作物接近成熟时发出警报。

图 5-10　遥测数据被发送到服务器，然后保存到数据库中

服务器代码也可以通过添加额外的信息来增强数据。例如，物联网设备可以发布一个标识符，以表明它是哪个设备，而服务器代码可以利用这个标识符来查询设备的位置，以及它正在监测哪些作物。服务器代码还可以添加一些基本数据，如当前时间，因为一些物联网设备没有必要的硬件来跟踪准确的时间，或者需要额外的代码来通过互联网读取当前时间。

做点研究

为什么你认为不同的田地可能会有不同的温度？

5.5.1 任务：发布温度信息

通过相关指南，使用你的物联网设备通过 MQTT 发布温度数据，以便日后可以进行分析。

- 用 Wio Terminal 发布温度数据。
- 用虚拟物联网硬件和树莓派发布温度数据。

连接设备：用物联网设备发布温度数据

用 Wio Terminal 发布温度数据

在这一部分，你将通过 MQTT 协议发布 Wio Terminal 检测到的温度值，这样它们就可以在以后被用于计算 GDD。

发布温度

一旦温度被传感器读取，就可以通过 MQTT 协议发布给“服务器”一些代码，这些代码将读取这些值，并存储它们，以便用于 GDD 计算。微控制器不用开箱就能从互联网上读取时间并用实时时钟来维护时间信息，设备需要被编程才能做这件事，并假设它有必需的硬件。

为了简化本课的内容，时间不会与传感器数据一起发送，而是由服务器代码在收到信息后添加。

任务：对设备进行编程以发布温度数据

（1）打开 `temperature-sensor` 的 Wio Terminal 项目。

（2）重复你在第 4 课中的步骤，连接到 MQTT 并发送遥测数据，你将使用相同的公共 Mosquitto 代理。操作的步骤如下。

- 在 `.ini` 文件中添加 Seeed Wi-Fi 和 MQTT 库。
- 添加配置文件和代码以连接到 Wi-Fi。
- 添加代码以连接到 MQTT 代理。
- 添加代码来发布遥测信息。

⚠ 注意

如果需要的话，请参考连接到 MQTT 的说明和第 4 课的发送遥测的说明。

（3）确保 `config.h` 头文件中的 `CLIENT_NAME` 体现了这个项目的主题。

```
const string CLIENT_NAME = ID + "temperature_sensor_client";
```

（4）对于遥测，不发送亮度值，而是通过改变 `main.cpp` 中的 `loop`（循环）函数，将从 SHT 传感器读取的温度值发送到 JSON 文档中的一个名为 `temperature`（温度）的属性中。

```
float temperature;
```

```
float humidity;
sht4x.measureHighPrecision(temperature, humidity);

DynamicJsonDocument doc(1024);
doc["temperature"] = temperature;
```

（5）温度值不需要经常被读取。它在短时间内不会有太大变化，所以在 `loop` 函数中设置 `delay`（延迟）为 10 分钟。

```
delay(10 * 60 * 1000);
```

> `delay`（延迟）函数取的是以毫秒为单位的时间，所以为了方便阅读，传递的是该表达式的计算结果。一秒钟为 1000 毫秒，一分钟为 60 秒，所以 10 ×（一分钟 60 秒）×（一秒钟 1000 毫秒）就得到了 10 分钟的延迟。

（6）将该程序上传到你的 Wio Terminal，并使用串口监视器来查看被发送到 MQTT 代理的温度。

```
> Executing task: platformio device monitor <

--- Terminal on COM5 | 9600 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, hexlify,
log2file, nocontrol, printable, send_on_enter, time
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Serial Number: 274563796
Connecting to Wi-Fi..
Connected!
Attempting MQTT connection...connected
Sending telemetry {"temperature":26.54879}
Sending telemetry {"temperature":26.49538}
```

你可以在 `code-publish-temperature/wio-terminal` 文件夹 🔗 `[L5-6]` 中找到这个代码。

> 恭喜，你已经将温度作为遥测数据从你的设备上成功发布了。

用虚拟设备和树莓派发布温度数据

在这部分课程中，你将通过 MQTT 发布由树莓派或虚拟物联网设备检测到的温度值，这样它们以后就可以用于计算 GDD。

发布温度

一旦温度被读取，就可以通过 MQTT 发布给“服务器”一些代码，这些代码将读取这些值，并存储它们，以便用于 GDD 计算。

任务：发布温度

对设备进行编程以发布温度数据。

（1）如果温度传感器应用项目还没有打开的话，应先打开温度传感器应用项目。

（2）重复你在第 4 课中的步骤，连接到 MQTT 并发送遥测数据，你将使用相同的公共 Mosquitto 代理。

操作步骤如下。

- 添加 MQTT pip 包。
- 添加连接到 MQTT 代理的代码。
- 添加代码来发布遥测信息。

⚠ 注意

如果需要的话，请参考连接到 MQTT 的说明和第 4 课的发送遥测的说明。

（3）确保 `client_name` 体现了这个项目的主题。

```
client_name = id + 'temperature_sensor_client'
```

（4）对于遥测，不发送亮度值，而是将从 DHT 传感器读取的温度值放到 JSON 记录中的一个名为 temperature（温度）的属性中。

```
_, temp = sensor.read()
telemetry = json.dumps({'temperature' : temp})
```

（5）温度值不需要经常被读取，它在短时间内不会有太大变化，所以将 time.sleep 设置为 10 分钟。

```
time.sleep(10 * 60);
```

`sleep`（睡眠）函数取的是以秒为单位的时间，所以为了方便阅读，该值是作为计算结果传递的。一分钟有 60 秒，所以 10 × 60 得到 10 分钟的延迟。

（6）以与你运行作业前一部分代码同样的方式运行代码。如果你使用的是虚拟物联网设备，那么请确保 CounterFit 应用程序正在运行，并且湿度和温度传感器已经在正确的引脚上被创建。

```
pi@raspberrypi:~/temperature-sensor $ python3 app.py
MQTT connected!
Sending telemetry  {"temperature": 25}
Sending telemetry  {"temperature": 25}
```

你可以在 `code-publish-temperature/virtual-device` 文件夹 🔗 `[L5-7]` 或 `code-publish-temperature/pi` 文件夹 🔗 `[L5-8]` 中找到这个代码。

😀 你已经将温度作为遥测数据从你的设备上成功发布了。

5.5.2 任务：捕获和存储温度信息

一旦物联网设备发布遥测数据，就可以编写服务器代码来订阅这些数据并存储它。服务器代码不是将其保存到数据库，而是将其保存到一个逗号分隔值格式（CSV）文件中。CSV 文件以文本形式存储数据，每个值用逗号隔开，每条记录在一个新行上。它们是一种方便的、人类可读的、受到良好支持的将数据保存为文件的方式。

CSV 文件将有两列，即日期和温度。日期列被设置为服务器收到信息的当前日期和时间，温度来自于遥测信息。

⚠ 注意

如果需要的话，你可以参考第 4 课的“接收来自 MQTT 代理的遥测数据”部分的内容。

（1）重复第 4 课的步骤，创建服务器代码以接收遥测信息。你不需要添加代码来发布命令。操作步骤如下。

- 配置并激活一个 Python 虚拟环境。

- 安装 paho-mqtt Pip 包。
- 编写代码来监听发布在遥测主题上的 MQTT 消息。

为这个项目的文件夹命名为 `temperature-sensor-server`。

（2）确保 `client_name` 体现了这个项目的主题。

```
client_name = id + 'temperature_sensor_server'
```

（3）在文件的顶部添加以下导入库的代码。

```
from os import path
import csv
from datetime import datetime
```

这样便导入了一个读取文件的库，一个与 CSV 文件交互的库，以及一个帮助处理日期和时间的库。

（4）在 `handle_telemetry` 函数前添加以下代码。

```
temperature_file_name = 'temperature.csv'
fieldnames = ['date', 'temperature']

if not path.exists(temperature_file_name):
    with open(temperature_file_name, mode='w') as csv_file:
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
```

这段代码为要写入的文件名和 CSV 文件的表头声明了一些常量。通常 CSV 文件的第一行包含由逗号分隔的表头。

然后，代码检查 CSV 文件是否已经存在。如果它不存在，就用第一行的表头来创建它。

（5）在 `handle_telemetry` 函数的末尾添加以下代码。

```
with open(temperature_file_name, mode='a') as temperature_file:
   temperature_writer = csv.DictWriter(temperature_file, fieldnames=fieldnames)
   temperature_writer.writerow({'date' : datetime.now().astimezone().replace(microsecond=0).
isoformat(), 'temperature' : payload['temperature']})
```

这段代码打开了 CSV 文件，然后在最后添加了新的一行。该行有当前的数据和时间，并被格式化为人类可读的格式，随后是从物联网设备上接收的温度。该数据以 ISO 8601 格式 🔗 `[L5-9]` 存储，带有时区，但没有微秒。

（6）以与之前相同的方式运行这段代码，确保你的物联网设备正在发送数据。一个名为 `temperature.csv` 的 CSV 文件将在同一文件夹中被创建。如果你查看它，你会看到日期 / 时间和测量温度。

```
date,temperature
2021-04-19T17:21:36-07:00,25
2021-04-19T17:31:36-07:00,24
2021-04-19T17:41:36-07:00,25
```

（7）运行这段代码一段时间以捕获数据。理想情况下，你应该运行一整天，以收集足够多的数据用于 GDD 计算。

> 如果使用的是虚拟物联网设备（见图 5–11），请选择随机复选框并设置一个范围，以避免每次返回温度值时得到相同的温度。
>
> 如果想运行一整天，那么需要确保运行服务器代码所用的计算机不会进入休眠状态，可以改变电源设置，或者运行像链接中这样保持系统活跃的 Python 脚本 [L5–10]。

你可以在 `code-server/temperature-sensor-server` 文件夹 [L5–11] 中找到完整的程序代码。

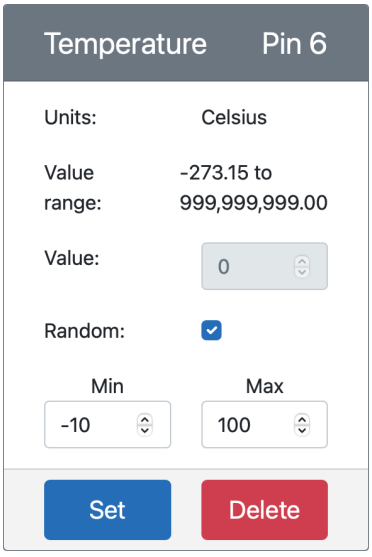


图 5–11　在虚拟物联网设备中可以设置返回的温度范围

5.5.3　任务：使用存储的数据计算 GDD

一旦服务器采集了温度数据，就可以计算出植物的 GDD。

手动操作的步骤如下。

（1）找到该植物的基点温度。例如，草莓的基点温度是 10℃。

（2）从 `temperature.csv` 中找到当天的最高温度和最低温度。

（3）使用前面给出的 GDD 计算方法来计算 GDD。

例如，如果当天的最高温度为 25℃，最低温度为 12℃，则有

25 + 12 = 37

37 / 2 = 18.5

18.5 – 10 = 8.5

$$\mathrm{GDD} = \frac{25+12}{2} - 10 = 8.5$$

因此，草莓已经收到了 8.5 个 GDD。草莓需要大约 250 GDD 才能开花结果，所以还需要一段时间。

课后练习

挑战

植物生长需要的不仅仅是热量。除了热量之外还需要哪些必要条件?

对于这些条件，寻找是否有可以测量它们的传感器，以及控制这些条件水平的执行器呢？你将如何组合一个或多个物联网设备来优化植物生长?

复习和自学

在数字农业百度百科 [L5–12] 页面上阅读更多关于数字农业的内容。还可以阅读更多关于精准农业 [L5–13] 的内容。

完整的生长度日数 GDD 计算比这里给出的简化计算更为复杂。在生长度日百度百科 GDD 页面 [L5–14] 上阅读更多关于生长度目的更复杂的方程式以及如何处理低于基线的温度的内容。

即使我们使用如此高效的耕作方法，未来我们也可能会面临食物短缺的危机。在 bilibili 上的《未来的高科技农场》视频 [L5–15] 中了解更多关于高科技耕作的技术。

作业

使用 Jupyter Notebook 工具，将 GDD 数据可视化。

说明

在本课中，你使用物联网传感器收集了 GDD 数据。为了获得良好的 GDD 数据，你需要收集多天的数据。为了帮助可视化温度数据和计算 GDD，你可以使用 Jupyter Notebook 🔗 `[L5-16]` 等工具来分析数据。

从收集几天的数据开始，你将需要确保你的服务器代码在你的物联网设备运行期间持续运行，你可以通过调整你的电源管理设置或运行像链接中这样保持系统活跃的 Python 脚本 🔗 `[L5-10]`。

一旦你有了温度数据，就可以使用 Jupyter Notebook 将其可视化并计算出 GDD。Jupyter Notebook 通常是将 Python 的代码和指令混合在称为单元的块中。你可以阅读指令，然后逐块运行每个代码块；也可以编辑代码。例如，在这个代码笔记本中，你可以编辑用于计算你植物对应 GDD 的基础温度。

（1）创建一个名为 `gdd-Calculation` 的文件夹。

（2）下载 `gdd.ipynb` 文件 🔗 `[L5-17]`，并将其复制到 `gdd-calculation` 文件夹中。

（3）复制由 MQTT 服务器创建的 `temperature.csv` 文件。

（4）在 `gdd-calculation` 文件夹下创建一个新的 Python 虚拟环境。

（5）安装一些用于 Jupyter Notebooks 的 Pip 包，以及管理和绘制数据所需的库。

```
pip install --upgrade pip
pip install pandas
pip install matplotlib
pip install jupyter
```

（6）在 Jupyter Notebook 中运行代码笔记本。

```
jupyter notebook gdd.ipynb
```

Jupyter Notebook 将启动并在你的浏览器中打开代码笔记本。通过代码笔记本中的说明，将测量的温度可视化，并计算出生长度日，如图 5-12 所示。

课后测验

（1）植物的生长取决于温度。这个说法（　）。

A. 正确

B. 错误

（2）植物生长需要考虑的温度包括（　）。

A. 最低温度、最高温度

B. 植物发育基点温度、最适温度、最高温度

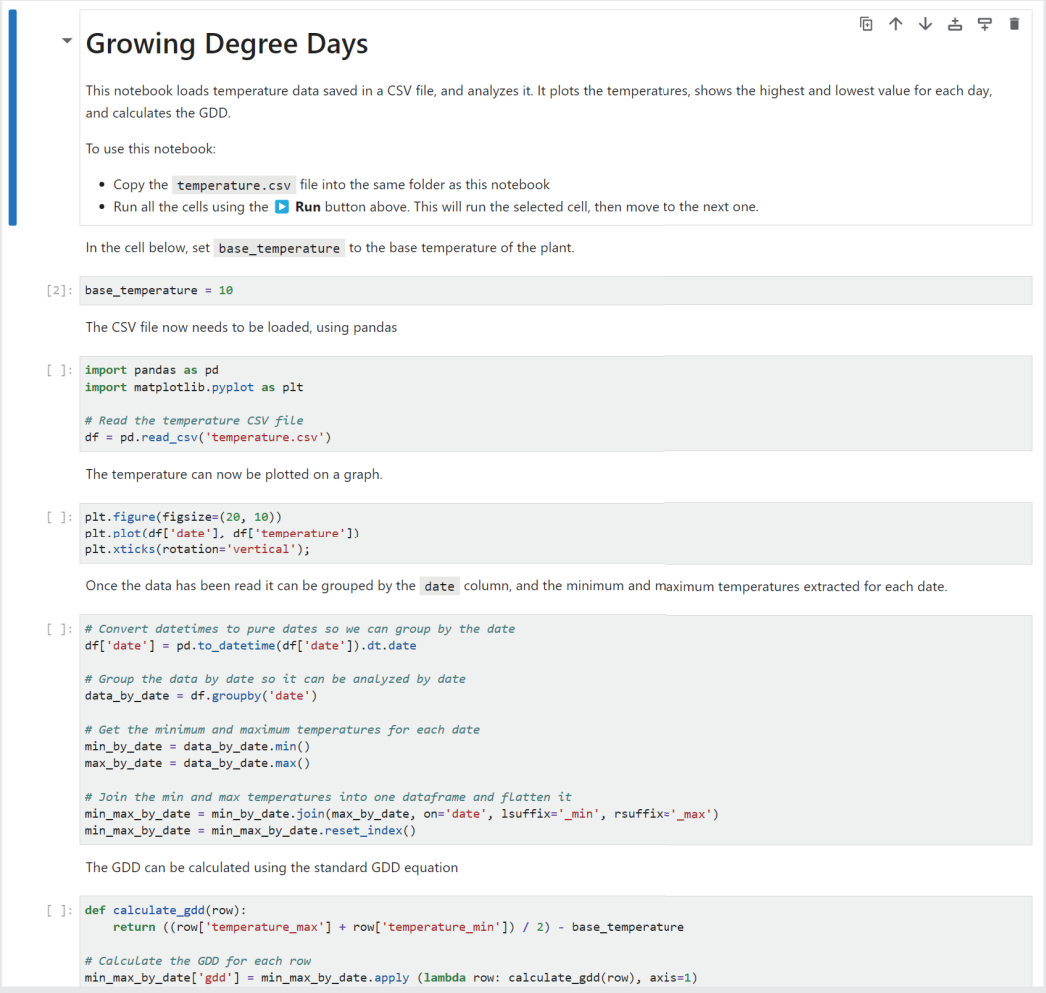
C. 仅最高温度

（3）下列哪个公式可以计算生长度日？（　）

A.（日内最高温度 + 日内最低温度）- 植物发育基点温度

B.（（日内最高温度 + 日内最低温度）/ 2）- 植物发育基点温度

C.（日内最低温度 + 植物发育基点温度）/ 2



Growing Degree Days

This notebook loads temperature data saved in a CSV file, and analyzes it. It plots the temperatures, shows the highest and lowest value for each day, and calculates the GDD.

To use this notebook:

- Copy the `temperature.csv` file into the same folder as this notebook
- Run all the cells using the **Run** button above. This will run the selected cell, then move to the next one.

In the cell below, set `base_temperature` to the base temperature of the plant.

[2]:
```
base_temperature = 10
```

The CSV file now needs to be loaded, using pandas

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Read the temperature CSV file
df = pd.read_csv('temperature.csv')
```

The temperature can now be plotted on a graph.

[ ]:
```
plt.figure(figsize=(20, 10))
plt.plot(df['date'], df['temperature'])
plt.xticks(rotation='vertical');
```

Once the data has been read it can be grouped by the `date` column, and the minimum and maximum temperatures extracted for each date.

[ ]:
```
# Convert datetimes to pure dates so we can group by the date
df['date'] = pd.to_datetime(df['date']).dt.date

# Group the data by date so it can be analyzed by date
data_by_date = df.groupby('date')

# Get the minimum and maximum temperatures for each date
min_by_date = data_by_date.min()
max_by_date = data_by_date.max()

# Join the min and max temperatures into one dataframe and flatten it
min_max_by_date = min_by_date.join(max_by_date, on='date', lsuffix='_min', rsuffix='_max')
min_max_by_date = min_max_by_date.reset_index()
```

The GDD can be calculated using the standard GDD equation

[ ]:
```
def calculate_gdd(row):
    return ((row['temperature_max'] + row['temperature_min']) / 2) - base_temperature

# Calculate the GDD for each row
min_max_by_date['gdd'] = min_max_by_date.apply (lambda row: calculate_gdd(row), axis=1)
```

图 5-12　在 Jupyter Notebook 中运行将测量温度可视化的程序

评分标准

标准	优秀	合格	需要改进
采集数据	采集至少两个完整天数的数据	采集至少一个完整天数的数据	采集一些数据
计算 GDD	成功地运行代码笔记本并计算 GDD	成功地运行代码笔记本	不能运行代码笔记本