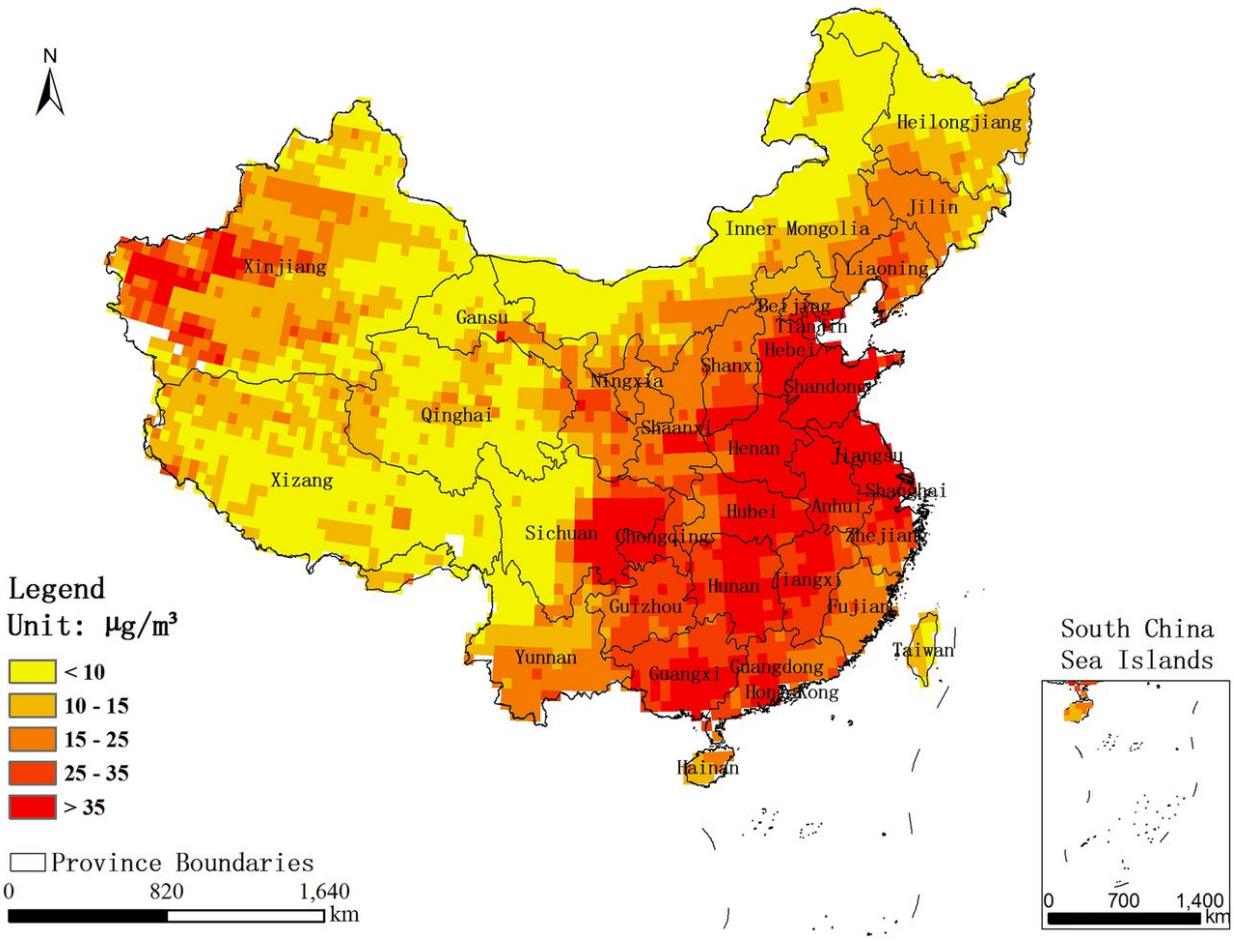
项目概述

董峦 新疆农业大学 2018

一、项目背景

人类社会大踏步发展带来的一个不容回避的问题是空气污染，这一点在大城市尤为突出，比如中国的北上广深。可吸入悬浮颗粒物是一类空气污染物，其中直径小于或等于2.5微米的颗粒物对人体损害最大，这类颗粒物即大家从媒体上听到的PM2.5，科学家用“PM2.5”来表示每立方米空气中这种颗粒的含量，其单位是$\mu g/m^3$这个值越高就代表空气污染越严重。

世界卫生组织(WHO)认为，PM2.5小于10是安全值，而中国的大部分地区高于50接近80，严重时甚至爆表。世卫组织为各国提出了非常严格的PM2.5标准，全球大部分城市都未能达到该标准。针对发展中国家，世卫组织也制订了三个不同阶段的准则值，其中第一阶段为最宽的限值，新标准的PM2.5与该限值统一，而 PM10此前的标准宽于第一阶段目标值，新标准也将其提高，和世卫组织的第一阶段限值一致。

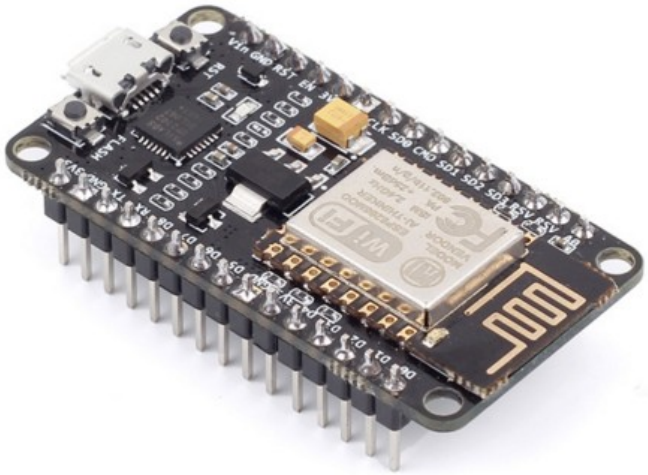


我国PM2.5标准采用世卫组织设定最宽限值，《标准》中PM2.5年和24小时平均浓度限值分别定为0.035毫克/立方米和0.075毫克/立方米，与世界卫生组织过渡期第一阶段目标值相同。

在本课程里我将引导大家利用PM2.5传感器、嵌入式开发平台和必要外围器件建立一个PM2.5采集装置，采集到的空气质量数据将实时上传到物联网云平台上，供记录、分析之用。

二、嵌入式开发平台

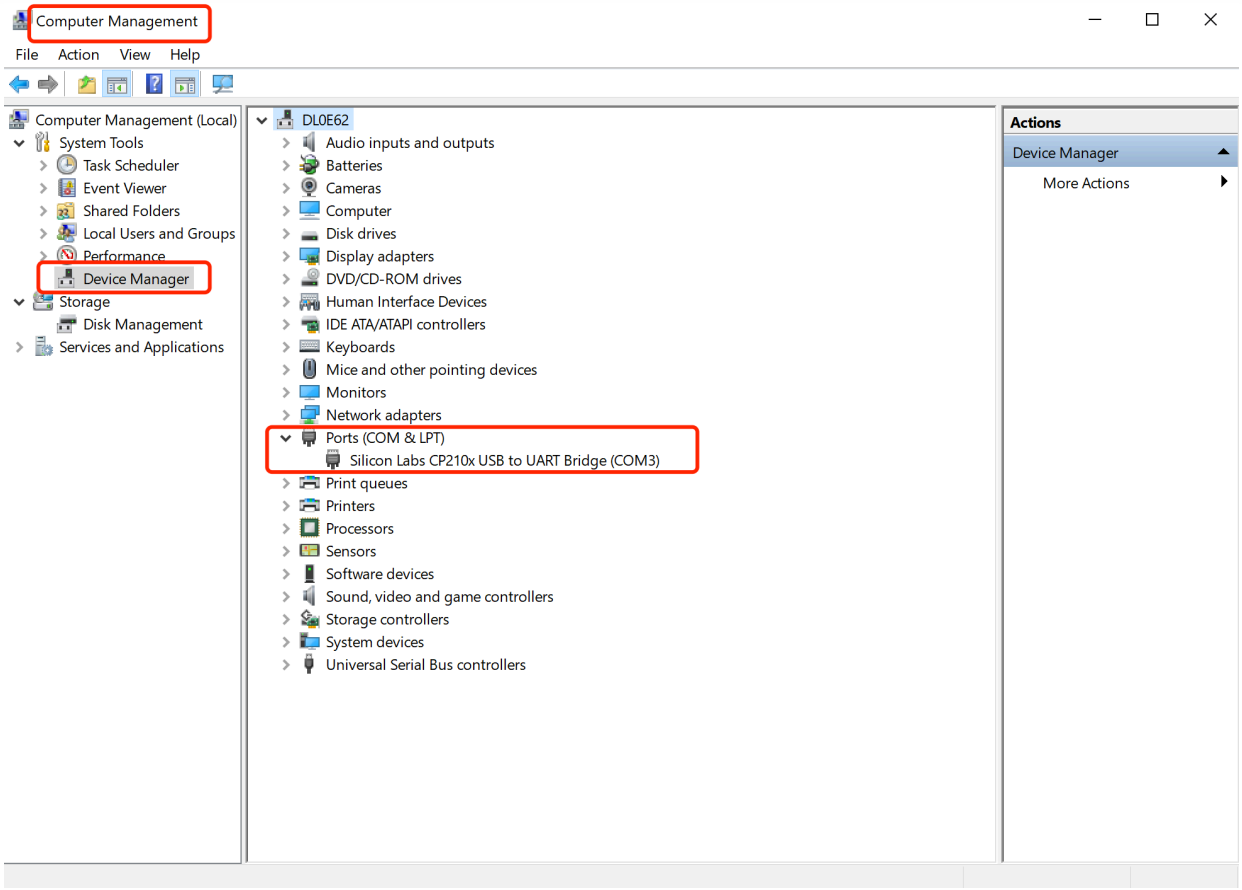
我们将采用的硬件平台叫NodeMCU，其官方网址是www.nodemcu.com。该开发板是基于Espressif（乐鑫）ESP8266芯片（内置WiFi功能的一款微控制器，无线联网功能与生俱来）的一款硬件原型开发平台。利用官方固件编程需使用Lua脚本语言，但由于其基于ESP8266芯片，所以很多人把其它固件移植到该平台，好用自己擅长的语言编程，比如Python。



MicroPython（micropython.org）由澳大利亚的物理学家Damien George开发，在2013年发布，是在微控制器上实现的部分兼容Python3语法的Python开发环境，包括C语言实现的Python编译器和运行时。目前MicroPython已可以运行在Arduino、ESP8266、ESP32等物联网平台上，人们终于可以用Python做硬件开发了！我们选购的该开发板已烧录了MicroPython固件，加之已有一定的Python开发经验，因此我们在这个平台上的开发过程将成为一个轻松愉快的经历。

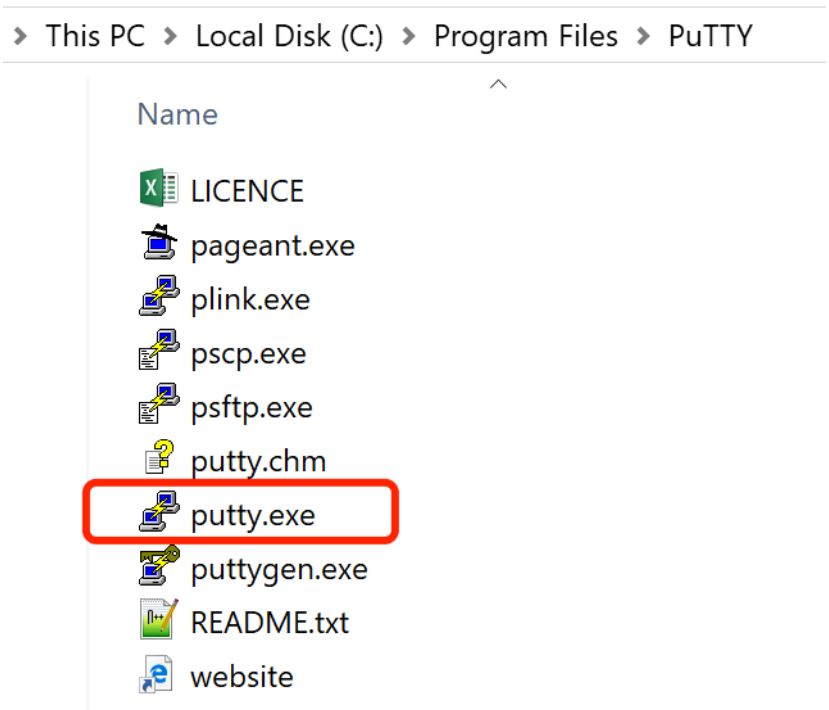
三、起步

首先我们要安装必要的驱动程序，在https://cn.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers 根据自己电脑的操作系统下载相应驱动程序。以Windows操作系统为例，安装完毕后能够在设备管理页面看到一个虚拟COM端口，在我的系统显示是COM3，如下图所示，记住该COM口编号。

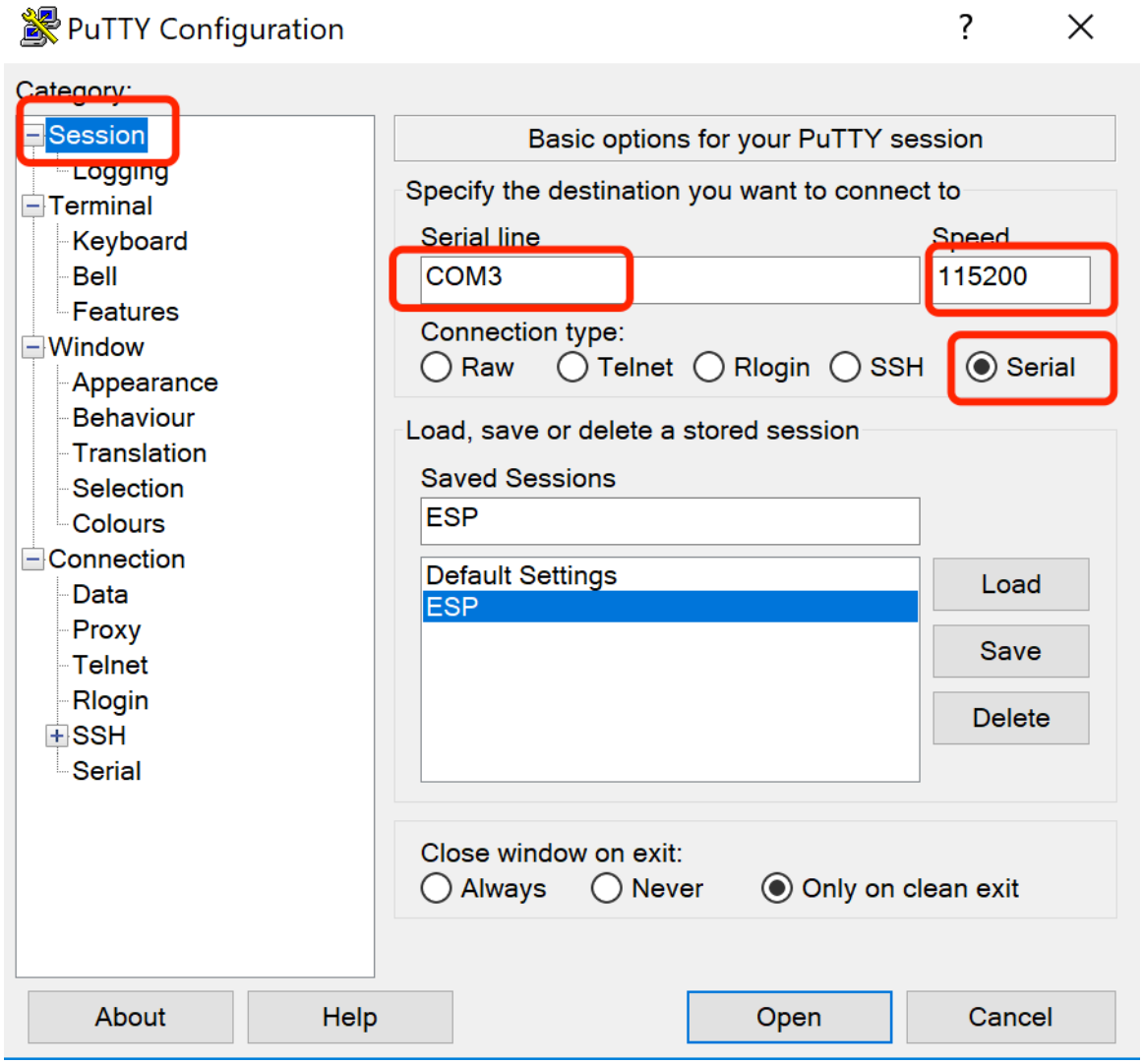


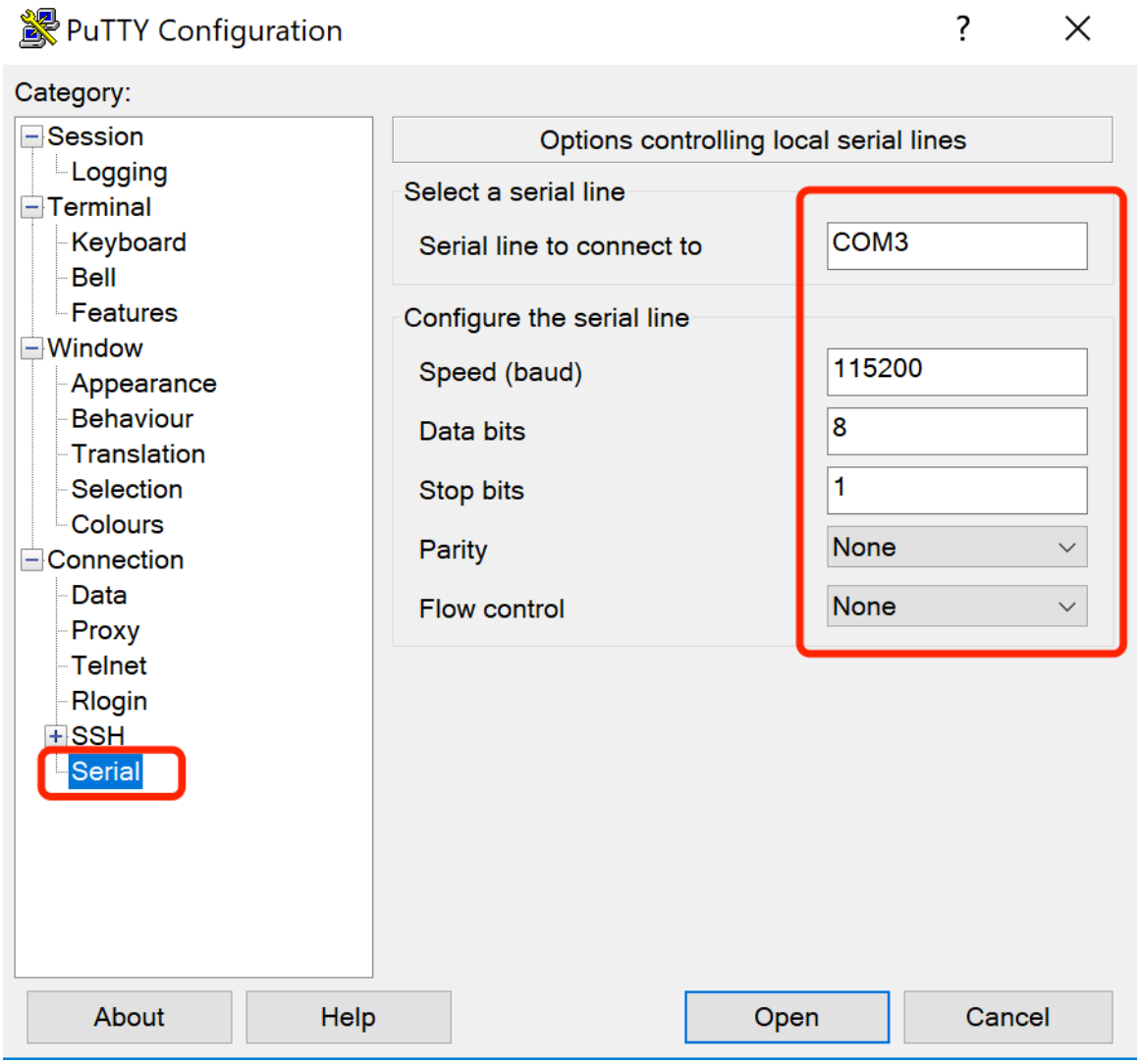
然后我们就可以与NodeMCU通信了。在这之前我们还需要一个串口通信工具，比如说PuTTY，从 https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html 根据你的操作系统下载安装文件，或使用随该文档提供给你的安装文件。

安装好PuTTY后运行putty.exe程序，



然后在‘Session’和‘Serial’页面按以下方式配置连接，配好后可将该配置保存，省的下次连接再次配置，下图中我已将配置保存为‘ESP’，所以直接载入就可以连接设备了。





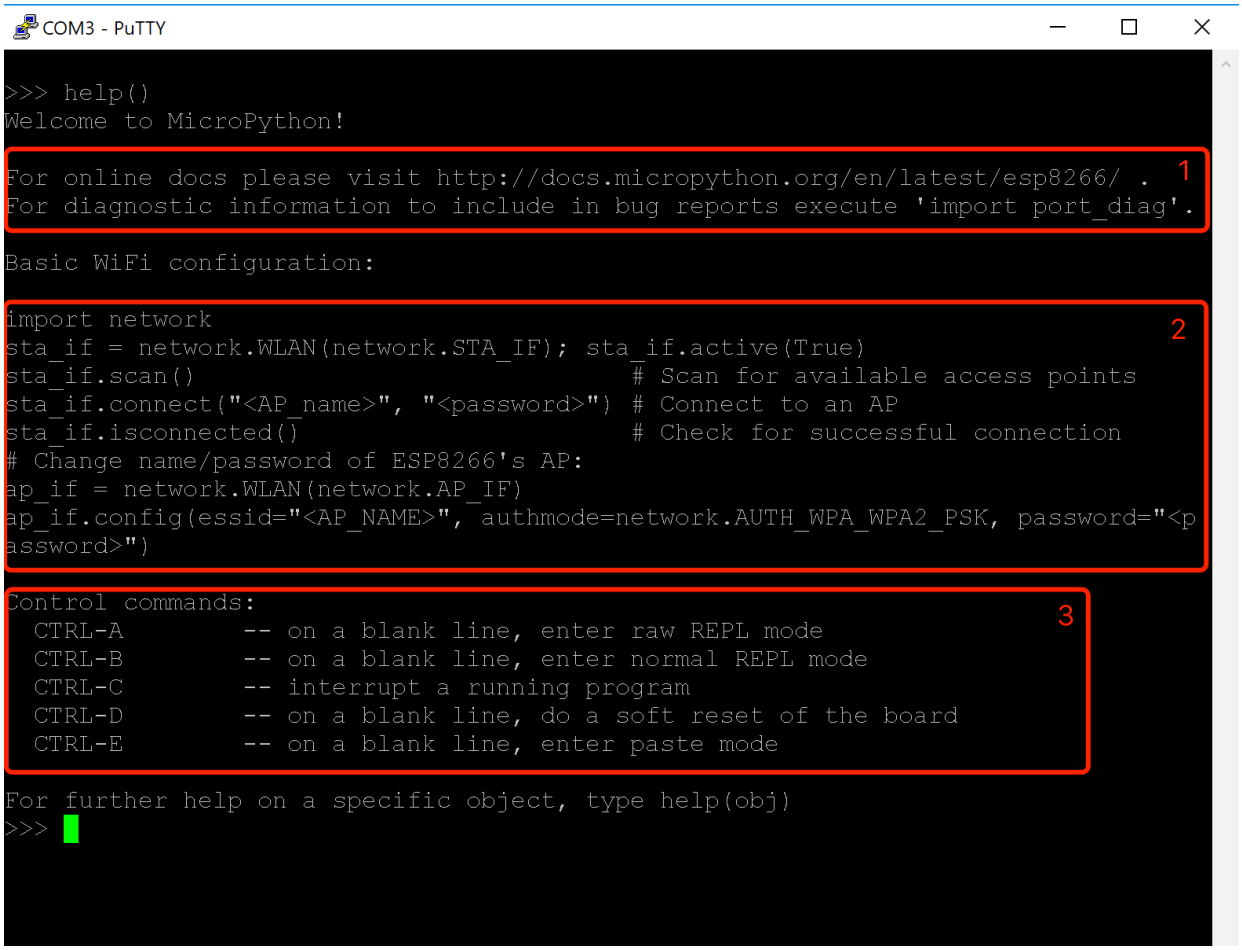
按图示配好参数后，点击‘Open’按钮，连接设备后可以Python解释器的提示符

如果看不到，可以按回车键或按NodeMCU上的RST键（reset键，重启设备）。看到提示符后，键入

```
print("hello world!")
```

看看是不是熟悉的Python。

键入 help() 可以显示操作该设备的基本方法，如下图所示



该帮助信息由三部分组成，十分重要。第一部分显示了MicroPython关于ESP8266的文档链接，该文档是我们使用Python对NodeMCU编程的权威参考，将频繁访问。

第二部分是配置无线网络的方法，ESP8266可工作在STA模式（Station模式，好比无线网卡，可理解为网络中的一个客户端）或AP模式（Access Point模式，好比无线路由器，提供无线接入服务，允许其它无线设备接入）或STA+AP的混合模式，既可以连接外部世界又可以对内部网络提供服务。利用该帮助提供的代码，我们可以配置手头的NodeMCU，比如我利用如下代码配置设备作为AP时的参数，即网络ID和密码均为micropython

```
import network
ap_if = network.WLAN(network.AP_IF)
ap_if.active(True)
ap_if.config(essid="micropython", authmode=network.AUTH_WPA_WPA2_PSK,
password="micropython")
```

观察无线网络，看看有没有名为‘micropython’的网络

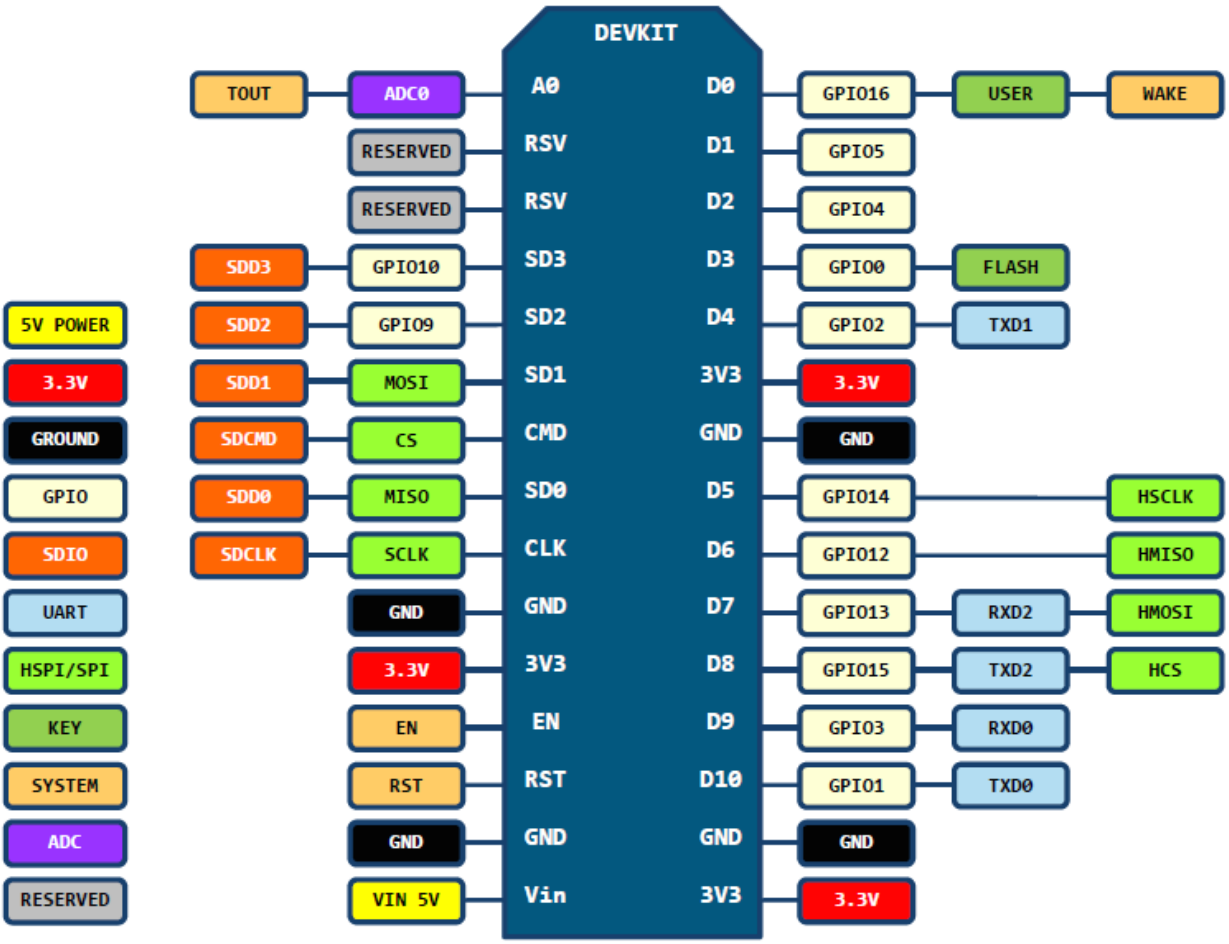
可以看到我们的NodeMCU正在提供无线接入服务（如果没看到该网络ID，试试重置设备）。在后续课程我们需要让设备连接互联网，即连接你的手机热点或宿舍中已连接互联网的无线路由器，则要输入如下代码

```
import network
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect("<AP_name>", "<password>")  # <AP_name>和<password>是你的热点或无线路由器的ID与密码
sta_if.isconnected()                       # 返回True说明连上了，否则没连上，需检查<AP_name>和<password>是否正确
```

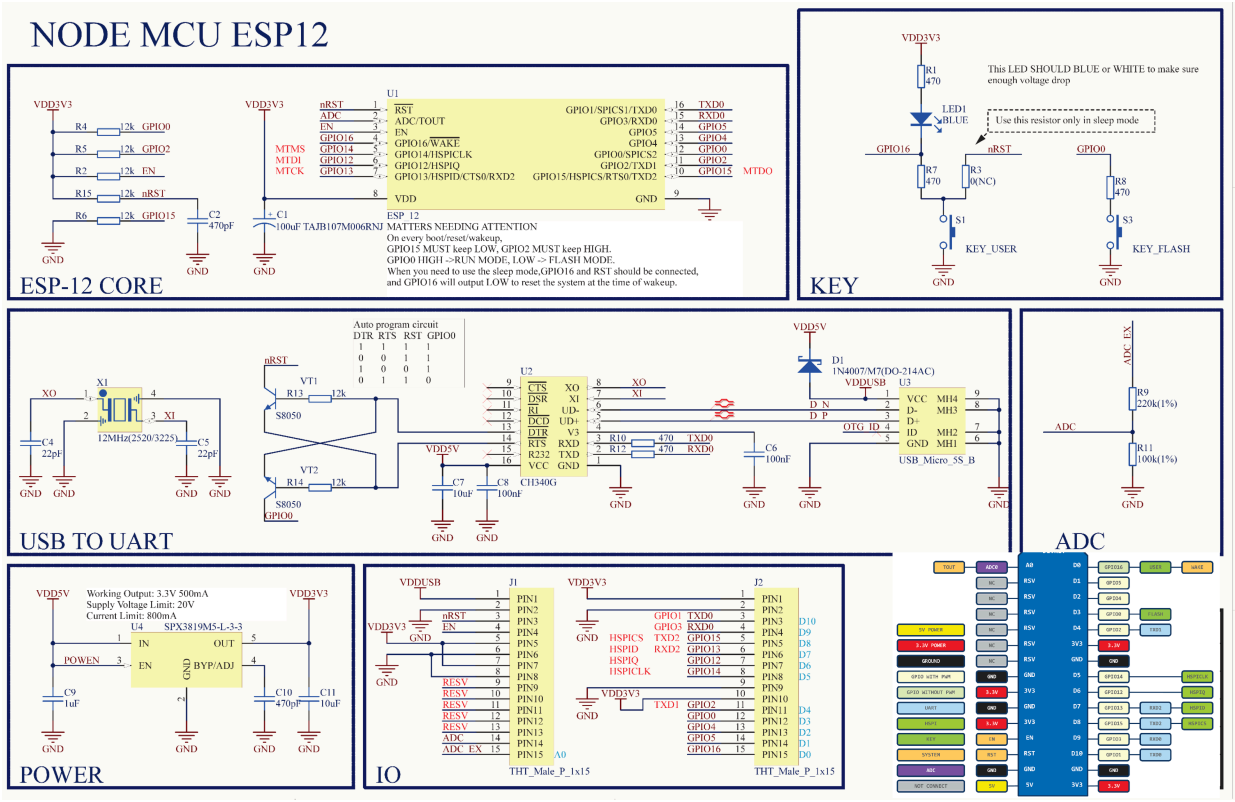
第三部分是我们使用MicroPython的REPL（Read-Eval-Print Loop，即交互式解释器）常用的命令键，比如按Ctrl+C组合键可以终止一个正在运行的程序。下面我们来点亮设备上的LED。

四、第一个程序

下面我们来点亮NodeMCU开发板上的LED灯。要想点亮这个LED我们需要知道ESP8266哪个引脚连接着它。下图展示了NodeMCU的引脚定义，图例的含义在左下角处标出。需要注意的是在MicroPython上编程，是根据ESP8266芯片的引脚来的，而不是按照NodeMCU模块上的丝印来的，对应关系一定要搞清楚，比如开发板右上角的D0口对应ESP8266的GPIO16（GPIO，General Purpose Input/Output）。



而GPIO16与外围器件如何连接就要看原理图了（下图对应的原文件是与本文档一同提供的NODEMCU_原理图.pdf）。



从该原理图右上方的‘KEY’部分可以看到GPIO16与‘LED1’的阴极相连，如果我们将该引脚电平拉低，则该LED将点亮。参考文档页http://docs.micropython.org/en/latest/esp8266/esp8266/quickref.html ‘Pins and GPIO’部分的代码，我们利用如下代码将该蓝色LED点亮。

```
from machine import Pin
led = Pin(16, Pin.OUT)
# 该开发板可用的GPIO引脚编号是 0，1，2，3，4，5，12，13，14，15，16，对应ESP8266的实际GPIO引脚编号，共有11个
# 上述语句创建对应GPIO16的Pin对象，将其设为输出模式
led.off() # 调用该对象的off方法，将引脚电平拉低
print(led.value()) # 得到引脚的值，0对应低电平，1对应高电平
```

看到蓝色LED亮了吗？如果亮了那就太好了，我们终于可以用Python进行硬件编程了，不再需要编译的过程，利用REPL就能实时看到结果。下面我们让LED闪烁起来。

```
import time
while True:
    led.on()
    time.sleep_ms(500) # 延时500毫秒
    led.off()
    time.sleep_ms(500)
```

你能够根据蓝色LED的表现理解上述代码吗？好了，让我们用前面介绍的组合键 Ctrl+C 让这个无限循环结束。下面我们复习一下Python中函数的定义

```
def blink(target, width, period, duration):
    import time
    start = time.ticks_ms() # 记录当前时间戳
    while time.ticks_diff(time.ticks_ms(), start) < duration:
        target.off()
        time.sleep_ms(width)          # 引脚拉低的时间由width参数决定
        target.on()
        time.sleep_ms(period - width) # 引脚拉高的时间由period - width决定

# 下面调用该函数
for i in range(10):
    blink(led, i, 10, 1000)
```

你能根据观察到的现象理解上述代码吗？试着阅读 time 模块的文档http://docs.micropython.org/en/latest/esp8266/library/utime.html 理解 ticks_ms，ticks_diff 方法的含义。思考题：请改写上述代码使得LED从亮到暗逐渐变化。

以上代码是手工实现了脉宽调制（PWM，Pulse Width Modulation），但根据文档ESP8266除了GPIO16，其它口都具有原生PWM功能，当我们需要PWM时，请使用其它GPIO口。

五、NodeMCU文件系统与启动过程

当芯片的FlashROM容量超过1M byte时（NodeMCU的ESP8266有4M byte FlashROM），MicroPython将在其固件之后创建一个FAT格式的文件系统，输入以下代码检查我们所处的目录

```
import os
os.getcwd()
```

结果显示我们在根目录‘/’，显示根目录下文件

```
os.listdir('/')
```

可以看到结果为 ['boot.py'] 即只有一个名为 boot.py 的文件，下面我们简要介绍MicroPython的启动过程。

上电后，MicroPython首先执行其内部的 _boot.py 脚本，挂载文件系统后执行 boot.py 文件，你可以查看一下该脚本内容

```
with open('boot.py', 'r') as fh:
    print(fh.read())
```

可以看到该脚本很简短，建议只有设置系统参数或添加服务时再编辑该文件。

启动的最后一步是执行 main.py 文件，顾名思义该脚本是用户程序的起点，下面我们就来创建该文件

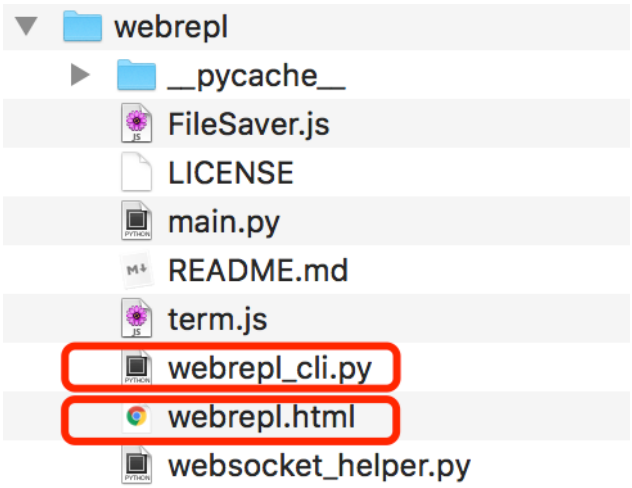
```
with open('main.py', 'w') as fh:
    fh.write('print("hello world!")')
```

按Ctrl+D组合键，软件重启该设备，可以看到每次重启都会输出‘hello world!’，即验证了 main.py 每次都会执行。

```
PYB: soft reboot
#9 ets_task(40100130, 3, 3fff838c, 4)
WebREPL is not configured, run 'import webrepl_setup'
hello world!
MicroPython v1.9.3-238-g42c4dd09 on 2017-12-30; ESP module with ESP8266
Type "help()" for more information.
>>>
```

但用这种方式编写大段代码显然不方便，下面我们用WebREPL这个工具从NodeMCU下载或上传文件。

我已将WebREPL与本文档一同提供，其包含如下内容



其中 webrepl.html 可以让你通过浏览器访问NodeMCU，要启动该功能，需要先在NodeMCU上输入下列语句，然后根据提示设置连接密码。

```
import webrepl_setup
```

配置好NodeMCU后，双击 webrepl.html 在浏览器中看到如下界面

默认IP为 192.168.4.1，但我已经将NodeMCU连接到了无线路由器上，其IP如图所示，所以你要将地址改为NodeMCU实际IP。如果你连接到了NodeMCU自己的网络中，则不需要修改默认IP。如果你之前设置了NodeMCU与手机热点或本地无线路由器的连接则每次重启设备时会看到其分配到的IP地址。

点击‘connect’，输入刚才设置的密码，就与设备连接了。注意浏览器窗口右侧可以‘send a file’也可以‘get a file’，在‘get a file’文本框里输出 main.py ，下载该文件，然后输入你自己的代码，比如让LED闪烁的代码，再通过‘send a file’上传该文件，观察运行效果。

WebREPL还处在开发之中，根据其文档描述，其工作模式是‘Single connection/channel’，因此在传输文件时会有与NodeMCU断开连接的情况，如果遇到这种情况，可以试着重新连接或重启设备。

如果你电脑上有Python开发环境，并熟悉命令行工具，也可以使用 webrepl_cli.py 工具传递文件，其语法是：

```
python webrepl_cli.py -p <password> 192.168.4.1:/main.py .     # 把远端文件 main.py 下载到本地，<password>处填实际连接密码
python webrepl_cli.py -p <password> main.py 192.168.4.1:/     # 把 main.py 上传到远端设备的根目录下
```

在上一节中你可能注意到了，当代码执行的时候，你是无法使用终端访问NodeMCU的，也就是说它的CPU被完全占用了。下面我们换一种实现方式，借此体会一下文件传递功能，复习一下Python面向对象编程的基础知识，也了解两个概念，分别是定时器和中断（interrupt）。

创建 main.py 文件，写入如下代码，然后用上述文件传输功能把 main.py 传到NodeMCU里。

```
import micropython
micropython.alloc_emergency_exception_buf(100)
# 当使用中断时，程序务必包含这两句，以便程序出错时显示具体的错误信息

import time
from machine import Pin
from machine import Timer # 定时器模块

class SquareWave(object):
    def __init__(self, pin_num, period, width):
         # pin_num:引脚编号，period:方波周期（单位毫秒），width:高电平持续时间
        self.period = period
        self.width = width
        self.pin = Pin(pin_num, Pin.OUT)
        self.timer = Timer(-1) # 创建虚拟定时器，请勿用硬件定时器
        self.timer.init(period=self.period, mode=Timer.PERIODIC, callback=self.pullup)
         # 该定时器定时时间是period，工作方式是周而复始的，回调函数是pullup方法。
         # 即每隔一个period调用一次pullup方法

    def pullup(self, t): # 在该方法中，pin_num引脚被拉高，并创建了一个一次性定时器
        self.pin.on()
        temp_timer = Timer(-1)
        temp_timer.init(period=self.width, mode=Timer.ONE_SHOT, callback=self.pulldown)
         # 该定时器在self.width指定的时间后执行一次pulldown方法

    def pulldown(self, t):
        self.pin.off() # 拉低pin_num引脚

wave = SquareWave(16, 1000, 500) # 创建对象时其__init__方法被执行，从而定时器被创建
```

该段代码的注释已经解释了很多问题，关于定时器请阅读文档 http://docs.micropython.org/en/latest/esp8266/esp8266/quickref.html#timers 和 http://docs.micropython.org/en/latest/esp8266/library/machine.Timer.html ，而中断的内容在 http://docs.micropython.org/en/latest/esp8266/reference/isr_rules.html 。那么一个简单的让LED闪烁的代码如此改进后带来什么好处了呢？首先，最根本的是程序的运行方式变了，从顺序方式变成了事件驱动方式，因此CPU被解放了；其次定时更准确了。

至此你已经具备了操纵NodeMCU的必备手段。接下来我们将了解PM2.5传感器的工作原理并用NodeMCU读取数值。