

# IoT for Fun!

## 2017 Maker Faire Workshop

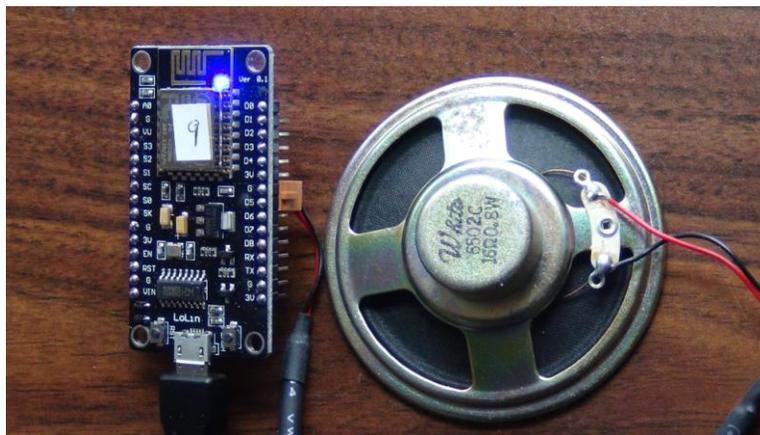
Dr. Chen-Hanson Ting

ESP8266 is the first commercial microcontroller with an integrated WiFi radio. With its 32-bit CPU, 128 KB of RAM, 4 MB of flash, and a MicroUSB connector, the NodeMCU board is easily the most capable microcontroller kit under \$5. NodeMCU thus opens the door for all people to explore IoT applications. This workshop exposes several different paths for you to try your hands on this kit, and encourages you to make the first step into the exciting IoT field.

WiFi is a very complicated subject involving many hardware and software issues. It is impossible to cover even a very small portion of it satisfactorily. The original developers of ESP8266 in Espressif Systems solved the hardware problem in silicon, and left a software development kit (SDK) for other software engineers to build programming tools known as IDE (Integrated Development Environment) for users to develop their own applications. In this workshop, we introduce three such IDE's for you to evaluate, hands on.

To help you taking the first step, we present you with a very simple challenge: to turn the LED on NodeMCU board on and off, remotely through WiFi. If you meet this challenge, you get a NodeMCU board for free!

Here is a picture of our project. A NodeMCU board with its LED turned on. A small speaker is also connected to a GND pin and GPIO-D5 pin. In our examples, the speak beeps when LED is turned on.



This task can be boiled down to two parts: making NodeMCU board a server and using PC as a client. The server and the client are connected to a WiFi network. You are sitting in front of the PC and typing in a command. The command is sent by the client to the server as a UDP packet. The server then turns the LED on and off, depending on the command you typed on PC.

You will use a UDP terminal emulator like Hercules as a client on PC. In a microPython experiment, you will use WebREPL as a client.

On NodeMCU board, you have the options to program it using espForth, Arduino, MicroPython, or Lua. In the Experiments Section, we detail steps to control the LED remotely:

espForth	Compile espForth_41.ino to NodeMCU under Arduino IDE; Send UDP packets to Forth server.
Arduino UDP Server	Compile UDPserver.ino to NodeMCU under Arduino IDE; Send UDP packets to server.
MicroPython/WebREPL	Flash NodeMCU with MicroPython; Enable WebREPL on MicroPython; Open WebREPL on PC to control LED.
MicroPython Server	Flash NodeMCU with MicroPython; Load UDPserver.py code on MicroPython; Send UDP packets to server.
Lua/ ESPlorer	Flash NodeMCU with Lua; Load UDPserver.lua on Lua; Send UDP packets to server.

To load programs and test them on NodeMCU board, you will need a few tools, which we provided on our PC workstations. They are:

TeraTerm	Serial Terminal Emulator to interact with NodeMCU
ESP8266Flasher	Write MicroPython or Lua into flash memory on NodeMCU
Arduino IDE	Program NodeMCU
Hercules SETUP utility	Send UDP packets to NodeMCU

In Tools section of this Manual, we detailed steps to set up these tools and to use them to meet the challenge.

Cheat sheets are available to guide you through the steps on the PC workstations, without reading the entire Manual. Following the cheat sheets, you can complete one of the experiments in about 30 minutes. If you encounter serious problems, then read the relevant sections in the Manual.

Since Espressif Systems took care of WiFi hardware, and the IDE developers took care much of the software, it is easy for you to meet this challenge. These experiments are simple because we use UDP protocols, the simplest way to get computers to talk to one another over WiFi.

For this workshop in the Maker Faire, we set up a WiFi router NETGEAR 150, with a SSID of “SVFIG” and a password of “12345678”. Our NodeMCU boards and PC’s send and receive UDP packets over this local area network, and avoid all the problems in using public WiFi networks to do experiments.

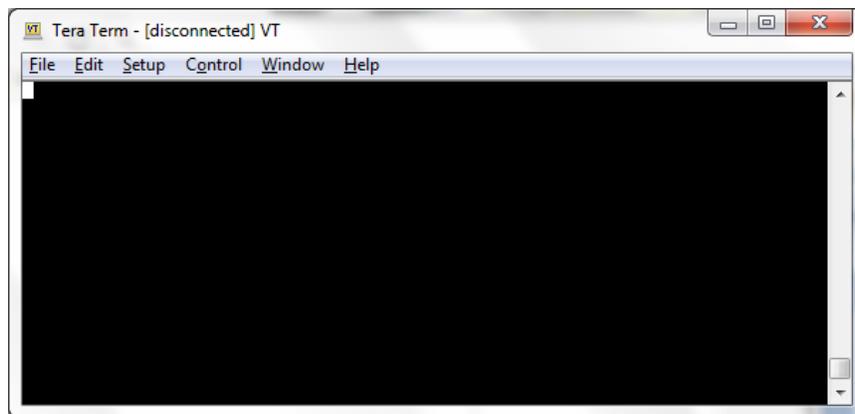
## Tool 1. TeraTerm Serial Terminal Emulator

ESP8266 has a number of serial communication ports to communicate with outside world. On NodeMCU board, one serial port is connected to a USB-Serial interface chip, which allows you to connect to a PC through a MicroUSB cable. When NodeMCU powers up, it configures the serial port to communicate at 11520 baud. Depending on the program loaded in its flash memory, NodeMCU can be controlled and also programmed through this USB-Serial cable.

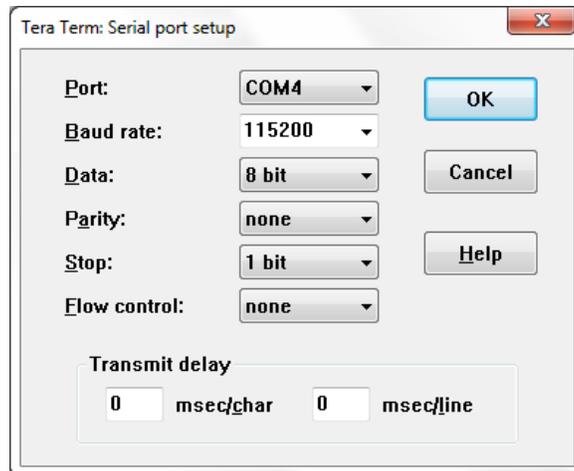
Different versions of NodeMCU used different USB-Serial interface chips. I have seen CP2101 chips and CH340g chips. They require specific USB drivers in PC to establish communication. All necessary drivers are included when Arduino IDE is installed. When you plug in a NodeMCU board, PC would recognize it and assign it a unique COM port. A terminal emulator on PC can then be opened to communicate with NodeMCU through this COM port.

There are many terminal emulators available on PC. The most convenient one is TeraTerm, which was installed on our PC for this workshop. It is generally the first thing you do in testing a fresh NodeMCU board, or any other microcontroller kit with USB-Serial interface.

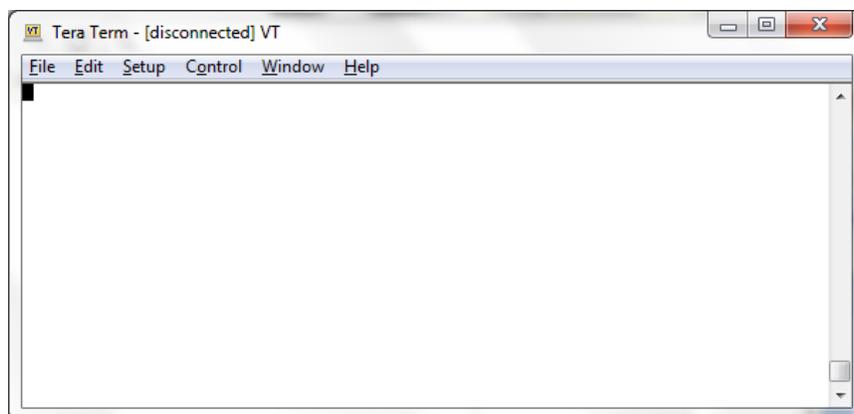
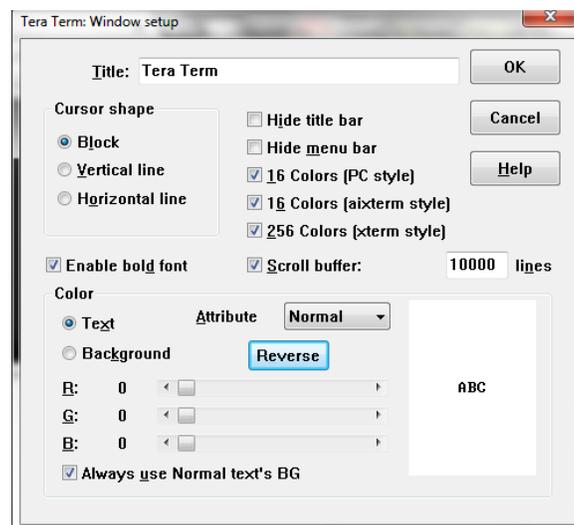
Open TeraTerm. If it is the first time you used it, you will see a black window:



Click Setup>Serial Port... and select the proper COM port. Set baud rate to 115200:

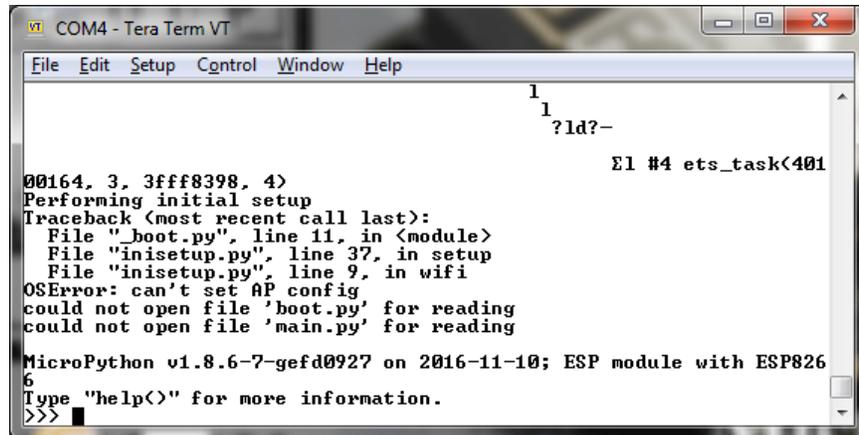


A black window was customary for the old PC-DOS operating system. In modern days, people expect black characters on a white screen. Click Setup>Window... and click the Reverse button to invert the screen color:



If you are happy with the screen, click Setup>Save Setup... and save your terminal settings in the file TERATERM.INI. Hereafter, when you open TeraTerm, you will have this screen to communicate with NodeMCU.

Plug in your NodeMCU board, and press the reset switch RST on NodeMCU, and you will see a screen full of characters. When I got MicroPython loaded in NodeMCU, the screen looks like this:



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
1
1
?ld?-
E1 #4 ets_task<401
00164, 3, 3fff8398, 4)
Performing initial setup
Traceback (most recent call last):
  File "_boot.py", line 11, in <module>
  File "inisetup.py", line 37, in setup
  File "inisetup.py", line 9, in wifi
OSError: can't set AP config
could not open file 'boot.py' for reading
could not open file 'main.py' for reading
MicroPython v1.8.6-7-gefd0927 on 2016-11-10; ESP module with ESP8266
6
Type "help(<)" for more information.
>>>
```

TeraTerm has many other functions and features. You only need it as a serial terminal emulator in this workshop.

Sometimes, when you reset NodeMCU board, you will get a screenful of garbage characters. It is because when ESP8266 boots up, it sends characters to PC and reads characters sent back from PC. Some control characters upset TeraTerm and caused it to use a different character set. To restore TeraTerm to its normal character set, click Control>Reset Terminal, and Edit>Clear Screen. You will get a blank screen to start over.

## Tool 2. Arduino IDE

Arduino was originally developed for lowly 8-bit AVR microcontrollers like ATmega328P from Atmel Corp. It greatly simplified the C programming language and made it very easy for you to write your own application on AVR chips. It gives you a very simple program template, which expects you to fill C code in two routines `setup()` and `loop()`. It captures the essences of firmware engineering and invites everybody to become a firmware engineer.

It is amazing that people in ESP8266 Community extended the Arduino IDE so that you can program this sophisticated 32-bit ESP8266 chip with ease. It even supports our ESP8266 NodeMCU 12E board!

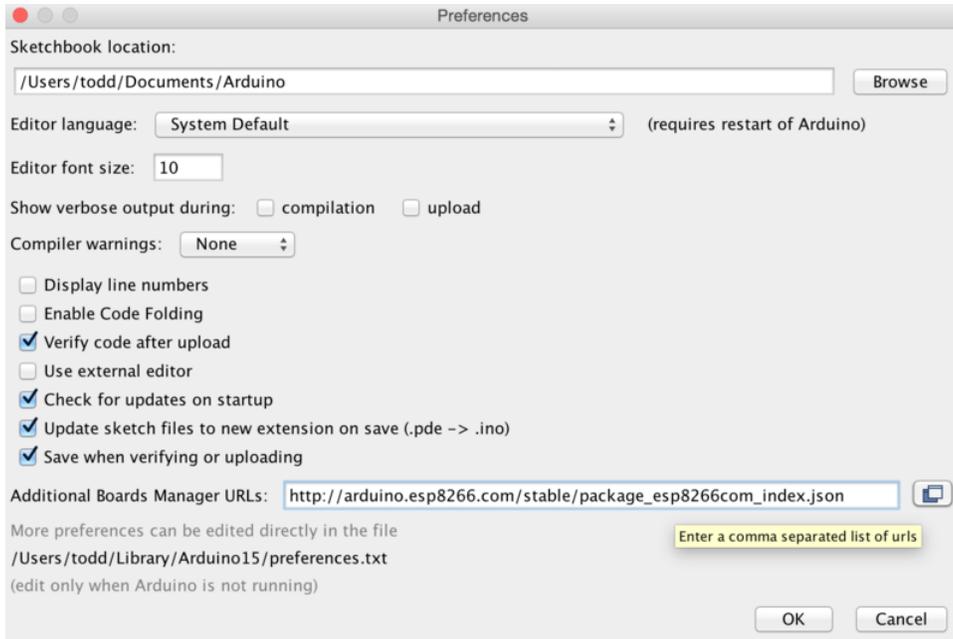
In the experiments with espForth and UDP Server, you will have to use Arduino IDE. If your computer does not have it, you have to install it first. Arduino IDE has to be extended so that it can compile programs for the ESP8266 chip, and to upload the compiled code to flash memory on NodeMCU board. After Arduino IDE is set up properly, it will be very easy to do experiments with NodeMCU board.

Download Arduino 1.8.2 IDE or the latest version from [www.arduino.cc](http://www.arduino.cc) and install it on your PC. Open Arduino, and you will see its title page:

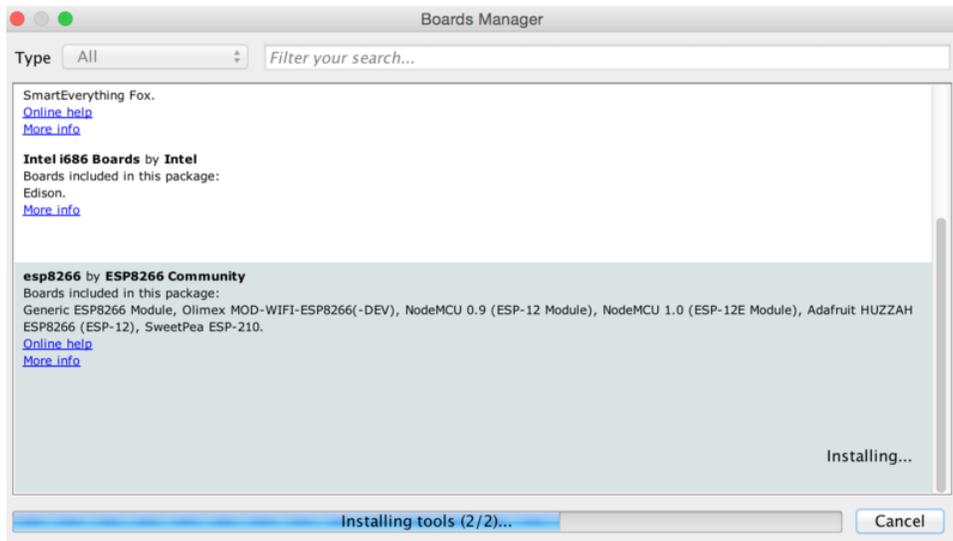


Click File>Preferences to open the Preferences window.

Enter [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json) into Additional Board Manager URLs field:



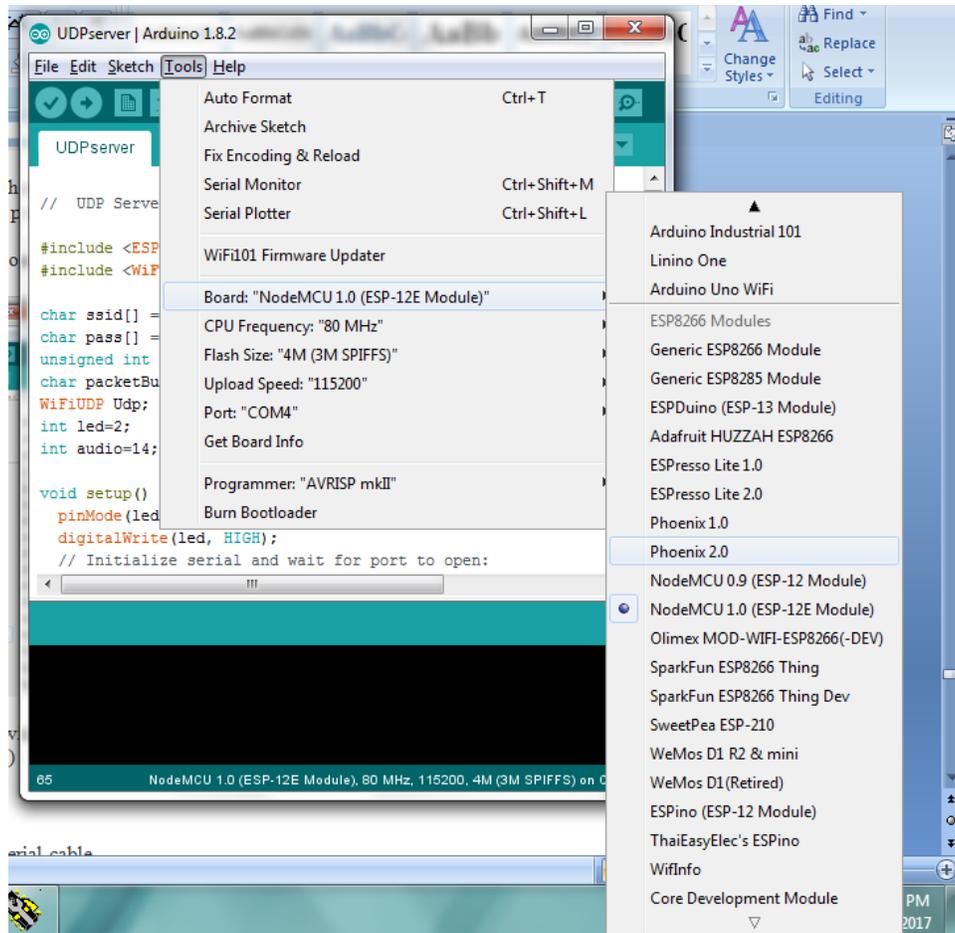
Next, Click Tools>Board:xxxxxxx>Boards Manager. Scroll to the bottom of the display, and click on the panel named esp8266 by ESP8266 Community to select it:



Click the Install button at bottom right to install the ESP8266 package.

After the install process, you should see that ESP8266 package is marked INSTALLED. Close the Boards Manager window once the install process has completed.

Select NodeMCU 1.0 from the Tools->Board dropdown menu:



In the Tools menu, you will see the following selections:

```
Board: NodeMCU 1.0 (ESP-12E Module)
CPU frequency: 80 MHz
Flash Size: 4M (3M SPIFFS)
Upload Speed: 115200 baud
Port: COM port for your FTDI or USB-Serial cable
```

Arduino IDE is now set up properly. You can now proceed to do espForth or UDP Server experiments.

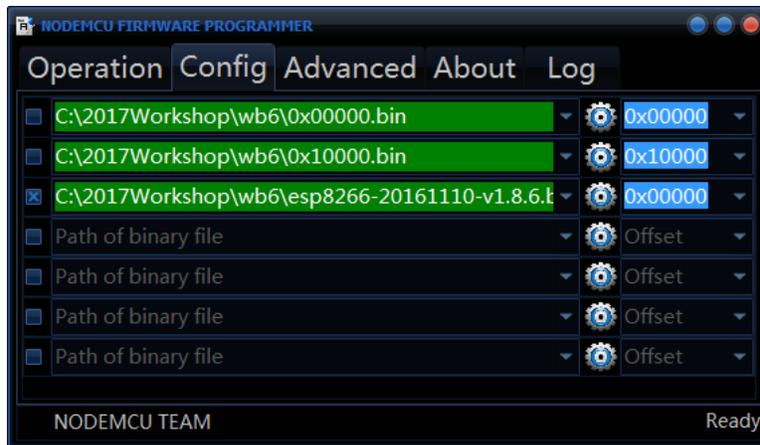
### Tool 3. ESP8266Flasher

If you will use Arduino IDE to do your experiment, skip this section, because Arduino IDE writes directly to flash memory when compiled code is uploaded to NodeMCU board.

In the experiments with MicroPython and Lua/ESPlorer, the language/programming operating system or the IDE (Integrated Development Environment) must be loaded into the flash memory on NodeMCU board. This is often called “Flashing”. The most convenient flashing tool is ESP8266Flasher on Windows PC.

Plug the MicroUSB end of the USB cable to NodeMCU, and the regular end of USB cable to your PC. If you have Arduino 1.8.2 installed properly, USB/Serial drivers are all loaded and NodeMCU should connect to PC automatically.

Open ESP8266flasher.exe. Click Config button. If you will use MicroPython, select binary file esp8266-20161110-v1.8.6.bin in the wb6 folder, and specify flash memory address 0x00000. If you will use Lua, first select binary file 0x0000.bin in the wb6 folder, and specify flash memory address 0x00000. Then select 0x10000.bin for memory address 0x10000, as shown here:

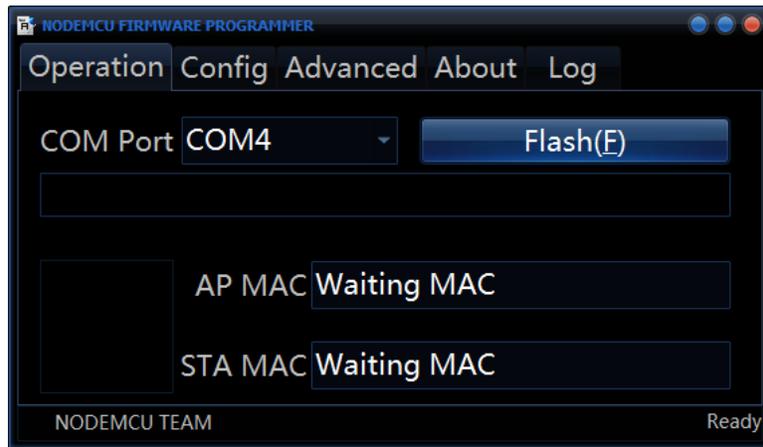


Make sure that the small square button to the left of a selected file is checked. Only a checked file will be written to the corresponding flash memory on NodeMCU board.

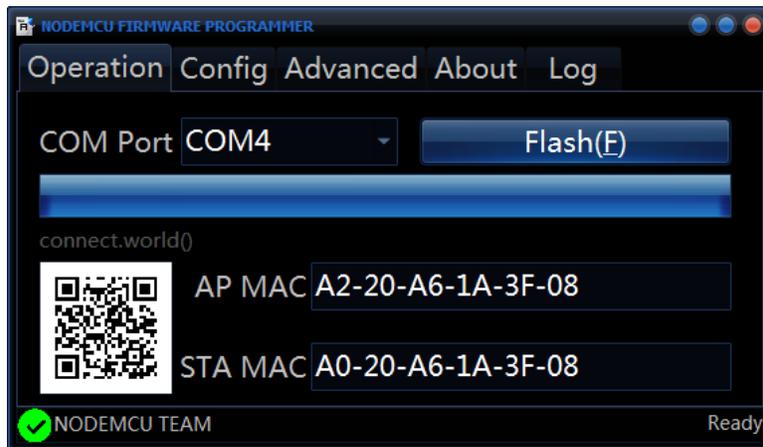
Click the Advance button, and select 115200 for baud rate:



Now click Operation button and return to the Operation main window. Click the big Flash(E) button to write the selected files into flash memory.



After binary files are written into flash memory, the Operation window will look like the following:



If you don't see the green button checked in the lower left corner, Flasher does not work correctly. Check Log window to see what is bothering it. Press the Reset (RST) button on NodeMCU board before flashing usually helps.

After flashing successfully, press the Reset button on NodeMCU board. Its LED flashes briefly, and NodeMCU should announce its presence to the terminal emulator on your PC. Be sure that the terminal emulator is set at 115200 baud, and the correct COM port is selected.

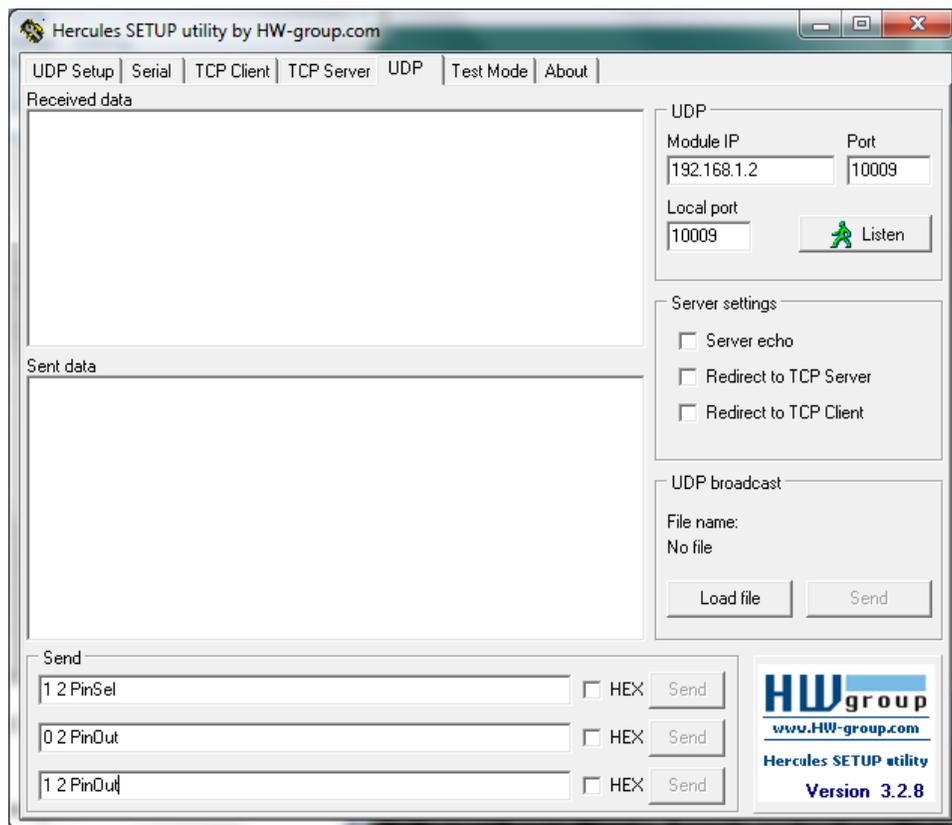
To run MicroPython, TeraTerm is a good terminal emulator. For Lua, use ESPlorer IDE. For Arduino, you can use its own Serial Monitor, or TeraTerm.

## Tool 4. Hercules SETUP Utility

To control the LED remotely over WiFi, the server on NodeMCU is set up to receive UDP packets, and act on the payloads. Here NodeMCU just turns its on-board LED on and off. To send packets you need a client on the same network. A client can be programmed to send many different packets, according to specific network protocols. Protocols can be very complicated, and for this workshop, you are required to learn the simplest protocol, UDP, the User Datagram Protocol. It does not make sense to use complicated protocols to do such a simple job: turning an LED on and off.

There are infinite ways to send UDP packets. We picked the Hercules SETUP Utility to do this job.

Open Hercules.exe, which is a network communication utility. Select UDP. Enter IP address and Port number of the server that you set up on NodeMCU board:



Press the Listen button to create a UDP socket which you can use to send UDP packets to NodeMCU.

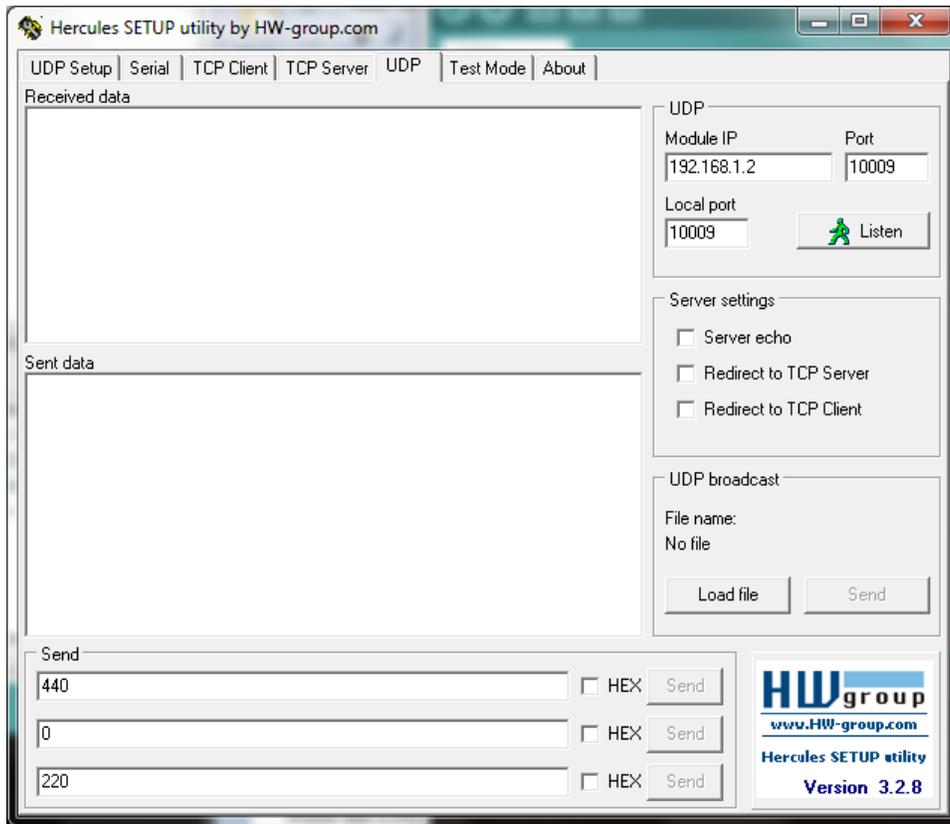
At the bottom of UDP window, there are three text boxes. If you are working on espForth Experiment, enter three messages,(as shown in the above picture):

```
1 2 pinSel
0 2 pinOut
1 2 pinOut
```

into the text boxes.

If you are working on all other experiments, enter these numbers in the text boxes:

```
440
0
220
```

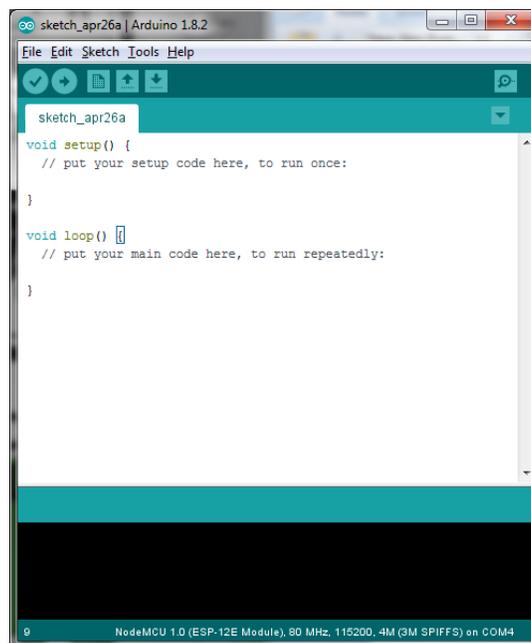


Pressing a Send button to the left of a text box, you send the corresponding UDP packet from Hercules to NodeMCU. You can see that the LED on NodeMCU is turned on and off, responding to payloads in packets.

## Experiment 1. espForth

Forth is the simplest programming language, and has been widely used for industrial, scientific, and embedded applications. eForth is the simplest Forth implementation for microcontrollers. I ported it to ESP8266 under Arduino IDE as espForth. Once espForth is loaded on NodeMCU, it allows you to explore this chip, and test its IO devices interactively. Since espForth accepts input from both the USB-serial COM port, and UDP packets simultaneously, you can turn its on-board LED on and off interactively through the Serial Monitor in Arduino, or through UDP packets through a UDP terminal like Hercules SETUP Utility.

Open Arduino IDE. If it is the first time you do anything with Arduino, you will probably have a default program template like this:

A screenshot of the Arduino IDE window titled "sketch\_apr26a | Arduino 1.8.2". The window has a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening, saving, and running. The main text area contains the following code:

```
sketch_apr26a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

The status bar at the bottom indicates "NodeMCU 1.0 (ESP-12E Module), 80 MHz, 115200, 4M (3M SPIFFS) on COM4".

Click File>Open..., and select C:/2017Workshop/espForth\_41/:espForth\_41.ino, supplied in the workshop project folder.

```
espForth_41 | Arduino 1.8.2
File Edit Sketch Tools Help
espForth_41 $
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
/*
const char* ssid = "ESPsoftAP";//type your ssid
const char* pass = "12345678";//type your password */
const char* ssid = "SVFIG";//type your ssid
const char* pass = "12345678";//type your password
unsigned int localPort = 10009;    // local port to listen on

WiFiUDP Udp;

typedef long cell;
typedef long instruction;

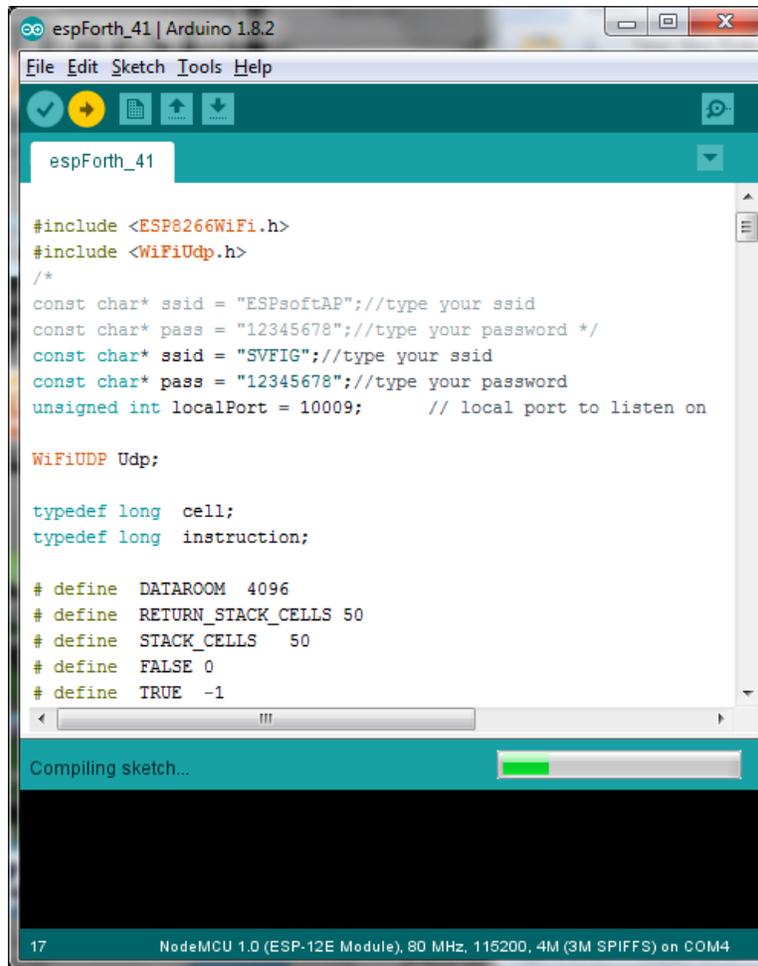
# define DATAROOM 4096
# define RETURN_STACK_CELLS 50
# define STACK_CELLS 50
# define FALSE 0
# define TRUE -1

Done Saving.
17 NodeMCU 1.0 (ESP-12E Module), 80 MHz, 115200, 4M (3M SPIFFS) on COM4
```

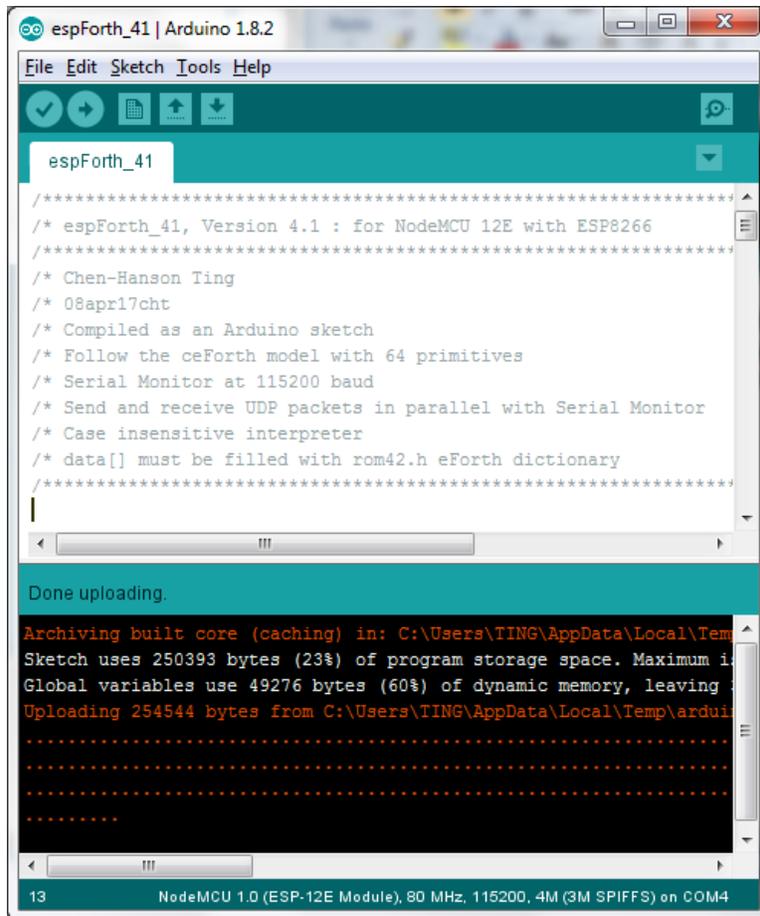
espForth is a file of 66 KB size, very small comparing to software of this age, but it is a complete interactive operating system with a high level Forth programming language. Here in this workshop, I will not bother you with its features and its usage. I just want to lead you to meet our challenge to turn a LED on and off.

Please note that espForth is connected to our local WiFi network, with a name of ‘SVFIG’ and a password of ‘12345678’. All NodeMCU boards and all PC’s used in this workshop have to be connected to this network, if they need to communicate with one another. Each NodeMCU are assigned a unique local port number from 10001 and up, to avoid conflicts.

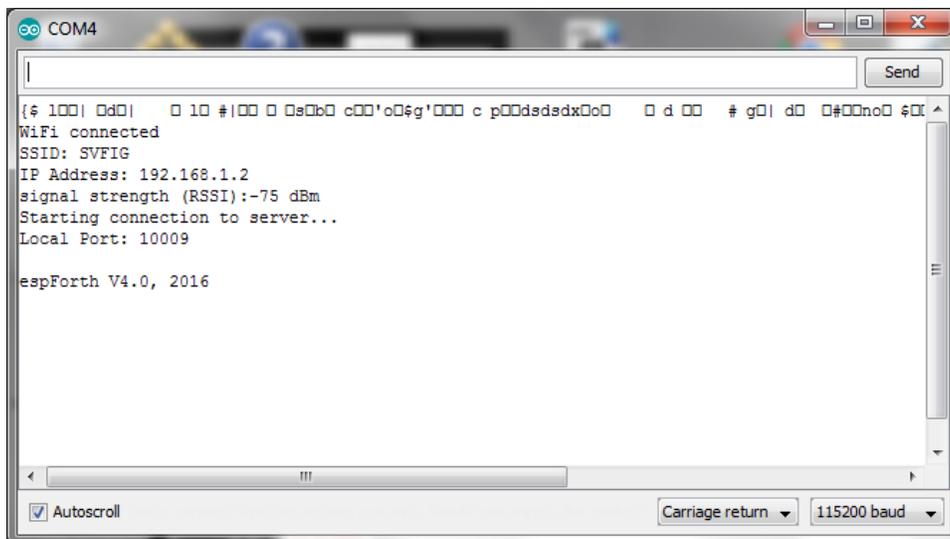
Click the Upload button(->), the one with an arrow pointing to right:



It takes a few minutes for Arduino IDE to compile the code and then upload the binary image into the flash memory on NodeMCU board. Eventually, it will report 'Done uploading', and reports to you what it accomplished:



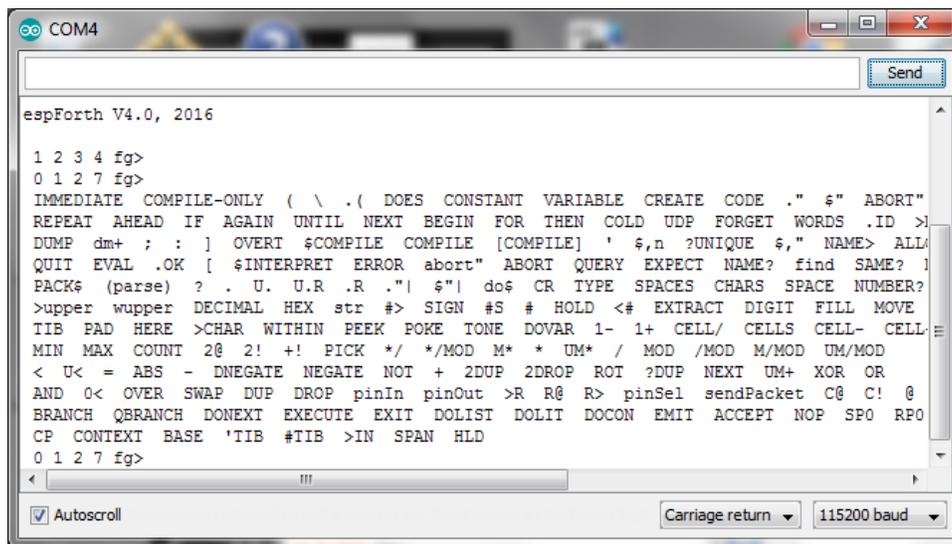
Click Tools>Serial Monitor. Be sure to select the correct COM port, and set baud rate to 115200. Press the RST (Reset) button on NodeMCU, and you will see espForth signing in:



Notice the IP Address and the Local Port number. You need these numbers to turn LED one and off remotely. Now you can type in the following Forth commands in the text box on the top of Serial Monitor window, (and click the Send button to the right), to exercise espForth system. After entering one line, press the Send button to the right of the text box.

```
1 2 3 4
+
WORDS
```

espForth is case insensitive. WORDS and Words are the same. After WORDS is entered, TeraTerm window looks like the following, showing all the Forth commands implemented in espForth:



Now, try to turn the on-board LED on and off with these lines of commands:

```
1 2 pinSel
0 2 pinOut
1 2 pinOut
```

The commands '1 2 pinSel' configure GPIO Pin 2 as an output pin. '0 2 pinOut' commands turn the on-board LED on. '1 2 pinOut' commands turns the LED off.

OK. You can turn the LED on and off, interactively, through USB-Serial cable. Now, try to do it remotely.

### Hercules SETUP Utility

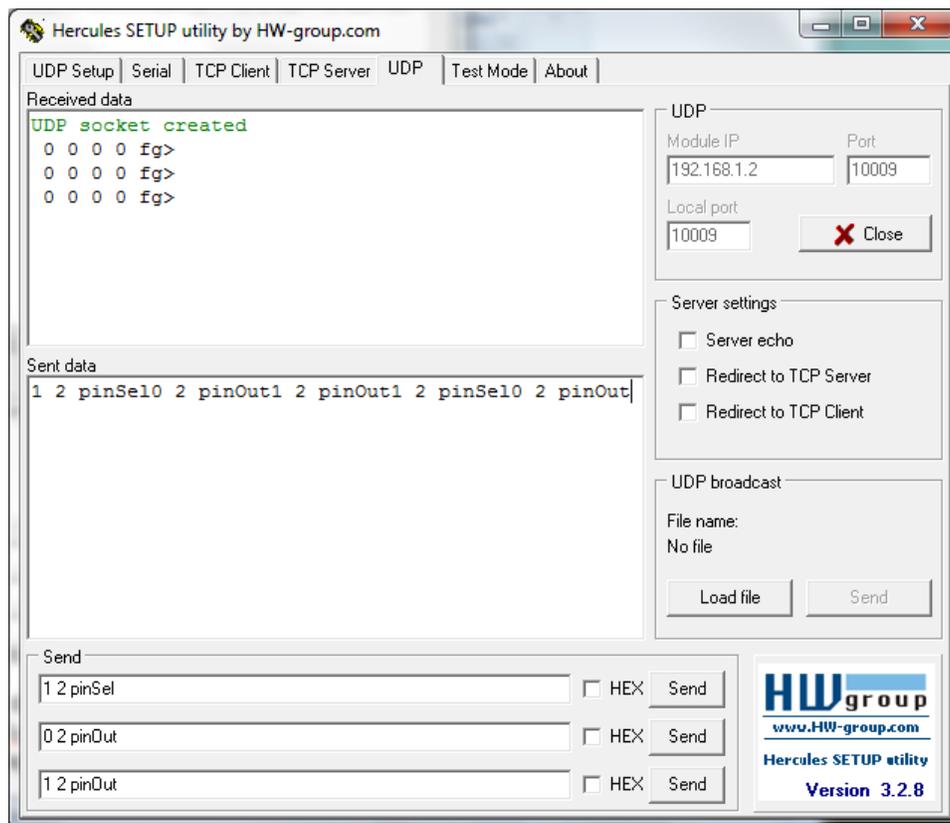
To control the LED remotely over WiFi, open Hercules.exe, which is a network communication utility. Follow instruction in Tool 4 Section to set up Hercules properly. Select UDP. Enter IP address and Port number.

Press the Listen button to create a UDP socket which you can use to send UDP packets to NodeMCU.

At the bottom of UDP window, there are three text boxes. Enter three messages:

```
1 2 pinSel  
0 2 pinOut  
1 2 pinOut
```

into the text boxes.



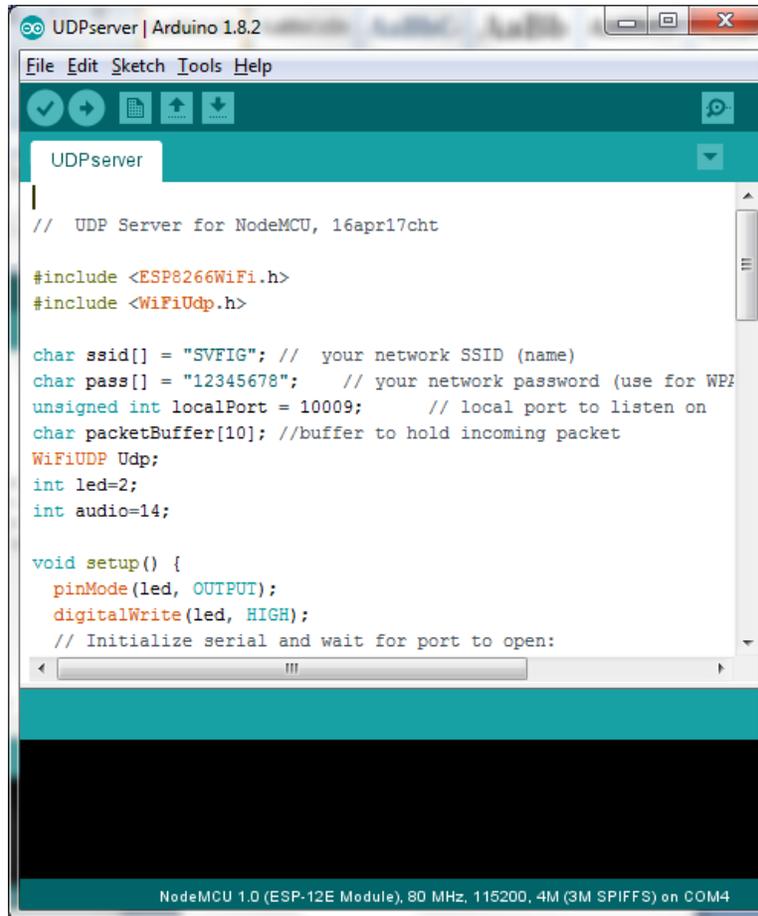
Pressing a Send button to the left of a text box, you send the corresponding UDP packet from Hercules to NodeMCU. You can see that the LED on NodeMCU is turned on and off, with the pinOut messages.

Congratulations! You have just proved that you can control the on-board LED on NodeMCU remotely. You've met our challenge. Take your NodeMCU board home, and have lots of fun with it.

## Experiment 2. Arduino UDP Server

In this experiment, you will build a server on NodeMCU board. This server waits for UDP packets. It will turn the on-board LED on and off, depending on the payload in the UDP packet. There are many different ways to send UDP packets. We will use a Hercules SETUP Utility by HW-Group to send UDP packets over our local WiFi network.

Open Auduino IDE, Click File>Open File... menu selection. Open UDPserver.ino, save it in a project file UDPserver when you are asked.



```
UDPserver | Arduino 1.8.2
File Edit Sketch Tools Help
UDPserver
// UDP Server for NodeMCU, 16apr17cht
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

char ssid[] = "SVFIG"; // your network SSID (name)
char pass[] = "12345678"; // your network password (use for WP
unsigned int localPort = 10009; // local port to listen on
char packetBuffer[10]; //buffer to hold incoming packet
WiFiUDP Udp;
int led=2;
int audio=14;

void setup() {
  pinMode(led, OUTPUT);
  digitalWrite(led, HIGH);
  // Initialize serial and wait for port to open:
```

UDPserver.ino implements a server on NodeMCU. The server waits for UDP packets sent to it on the WiFi network. It expects a number in the packet, and uses it to set the speaker frequency in Hz. It also turns on the on-board LED. If the number is 0, it silences the speaker and also turns off the LED.

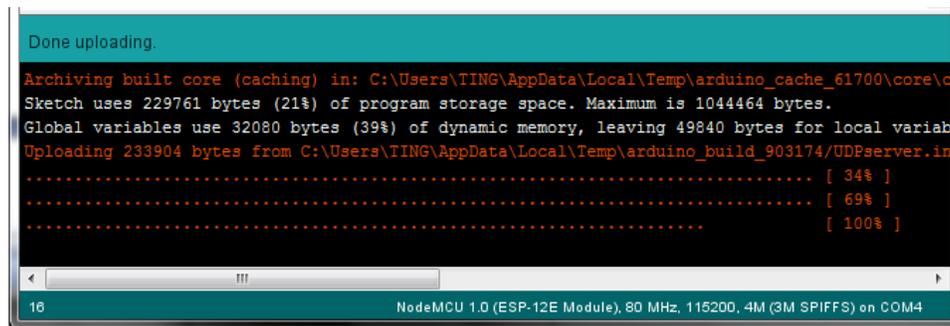
Source code of UDPserver.ino is actually very simple, if you ignore all the messages sent out to the Serial Monitor to help you understanding the booting procedure. The essential core of code is as follows:

```

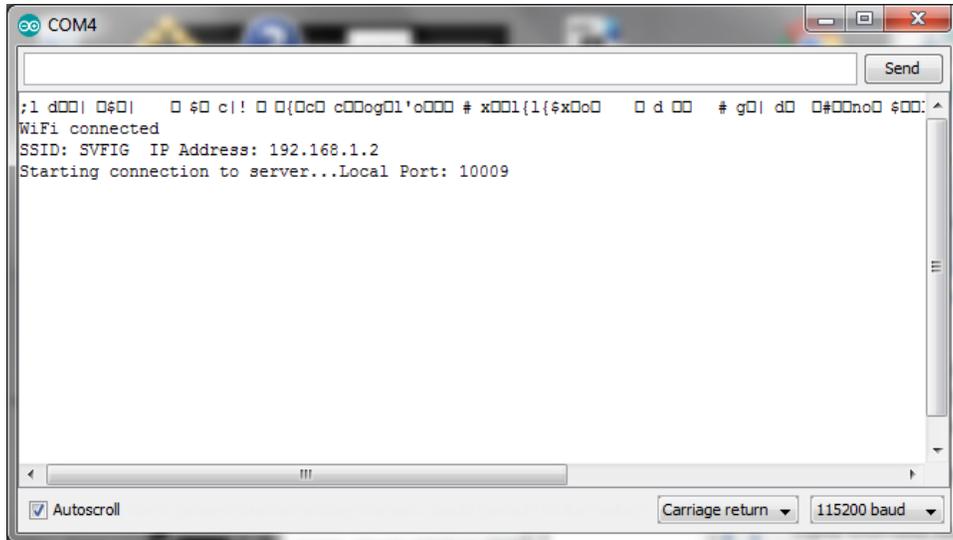
WiFiUDP Udp;
void setup() {
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  IPAddress ip = WiFi.localIP();
  Udp.begin(localPort);
}
void loop() {
  // if there's data available, read a packet
  int packetSize = Udp.parsePacket();
  int n;
  if (packetSize) {
    IPAddress remoteIp = Udp.remoteIP();
    int len = Udp.read(packetBuffer, 10);
    if (len > 0) { packetBuffer[len] = 0; }
    n=atoi(packetBuffer);
    if (n>0) { digitalWrite(led,LOW); tone(audio,n); }
    else { digitalWrite(led,HIGH); noTone(audio); }
  }
}

```

Click the Upload button. UDPserver is compiled and then uploaded to NodeMCU board. After uploading, you will see the compile/upload report at the bottom of the Arduino window:



Select Tools>Serial Monitor. Make sure that the baud rate is set at 115200 baud at the bottom of the Serial Monitor window. UDPsever will sign in:



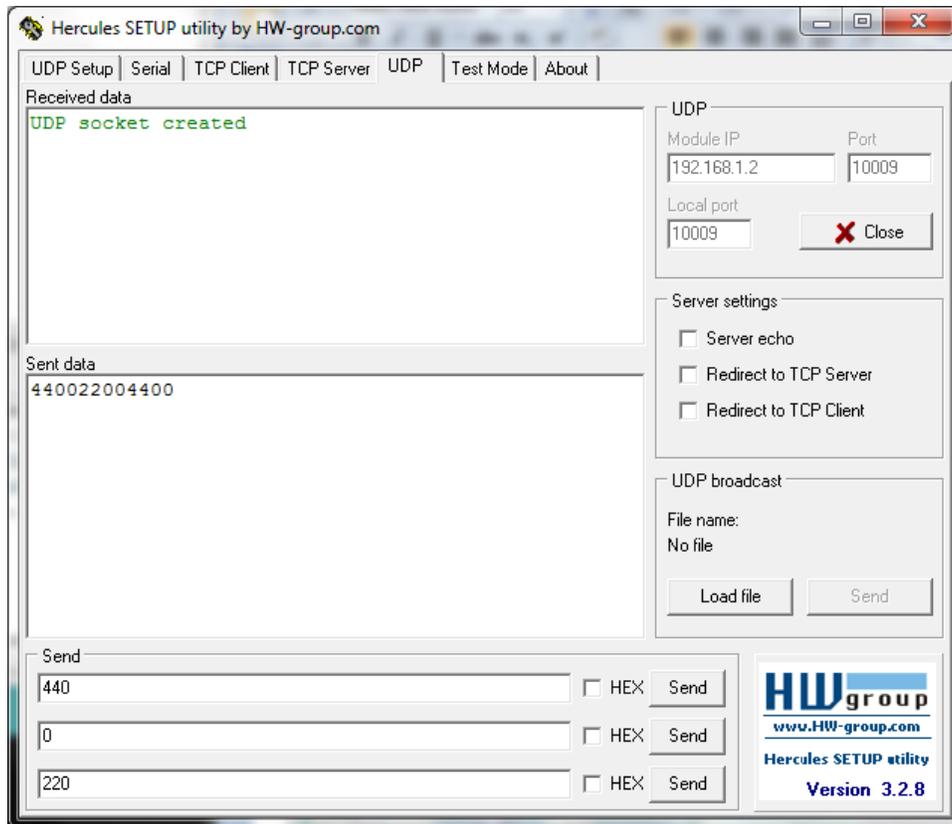
Please note the IP address shown. In this case, it is 192.168.1.2. The Port number assigned to UDP server is 10009. They will be used to communicate with NodeMCU remotely through a UDP terminal emulator Hercules.

### **Hercules SETUP Utility**

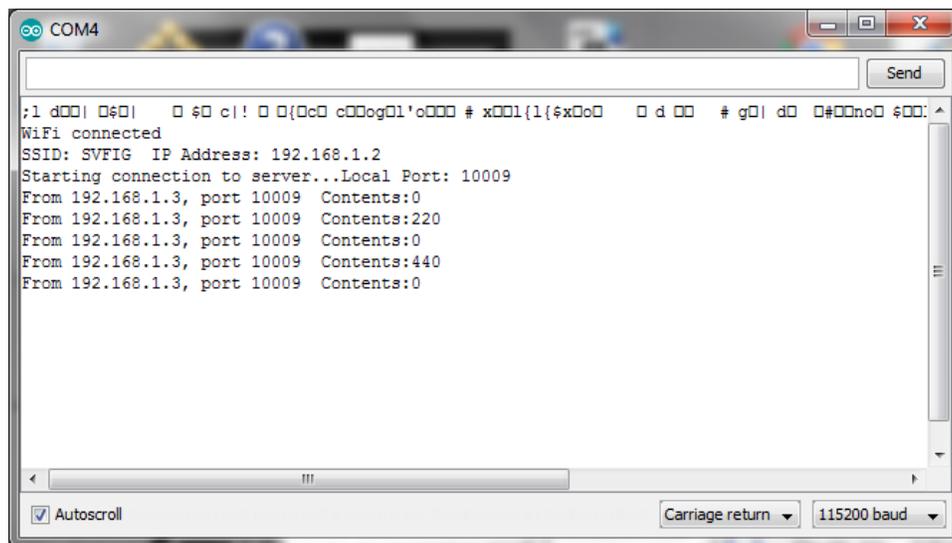
Open Hercules.exe, and select UDP menu. Enter IP address and Port number as you saw in Serial Monitor when NodeMCU started. See Tool 4 Section for detailed set up instructions.

Press the Listen button to get Hercules connected to NodeMCU. In the Receive Data panel, you will see that Hercules announces “UDP socket created”.

At the bottom of UDP window, there are three text boxes. Enter three messages: 440, 220 and 0, into the text boxes. Pressing a Send button to the left of a text box, you send the corresponding UDP packet from Hercules to NodeMCU. You can see that the LED on NodeMCU is turned on and off, and the speaker beeps accordingly.



On the Serial Monitor opened by Arduino, you can see the packets NodeMCU received:



Here you are. NodeMCU was programmed as a server, and receives packets over our WiFi network to control its LED and speaker. You've met our challenge. You have just made the first step into the wonderful IoT world.

## Experiment 3. MicroPython/WebREPL

In this experiment, you will first set up MicroPython on a NodeMCU board. When powered up, MicroPython communicate with you through REPL interpreter, across the USB-Serial cable. Its WiFi communication is generally not enabled. You have to set WiFi up correctly so that NodeMCU can receive commands remotely from a web page called WebREPL. Once WiFi is enabled and WebREPL is set up correctly, you can issue commands on WebREPL to turn the LED on NodeMCU on and off. You do not have to write any code to meet our challenge.

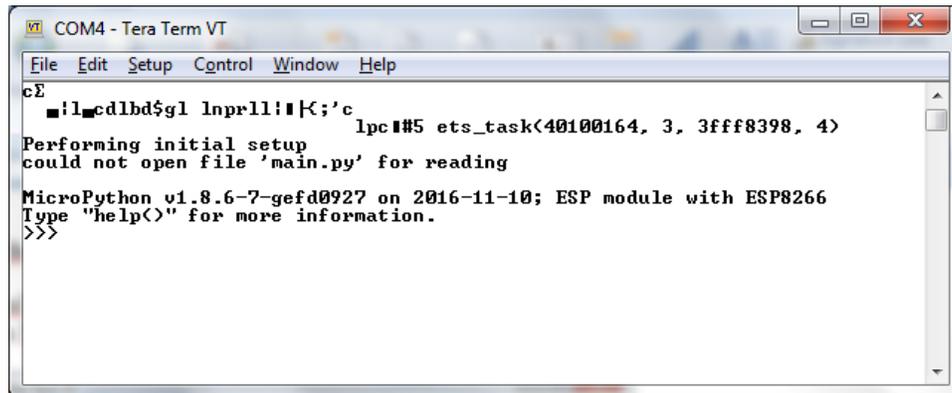
First, you have to load MicroPython binary into the flash memory of NodeMCU. Follow the directions in the Tool 3 ESP8266Flasher section earlier.

### REPL

After you flash MicroPython on NodeMCU, you can talk to it in REPL(Read Evaluate Process Loop). Open TeraTerm or any other serial terminal emulator to talk to it through the USB-Serial cable. TeraTerm is my preferred terminal emulator. If it is not set up properly on your PC, follow instructions in Tool 1 section.

Open TeraTerm. Click Setup>Serial Port... and select the proper COM port. Set baud rate to 115200.

Press the RST (Reset) switch on NodeMCU board. MicroPython you just flashed in NodeMCU will sign in:

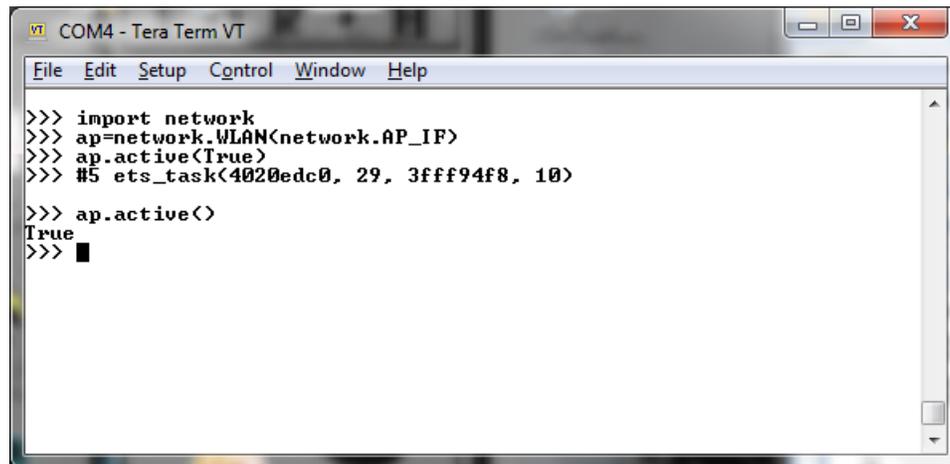


```
COM4 - Tera Term VT
File Edit Setup Control Window Help
cE
■!l_cdlbd$g1 lnprll:|K;c
lpc|#5 ets_task<40100164, 3, 3fff8398, 4>
Performing initial setup
could not open file 'main.py' for reading
MicroPython v1.8.6-7-gef40927 on 2016-11-10; ESP module with ESP8266
Type "help<>" for more information.
>>>
```

When NodeMCU is reset, you may see lots of garbage characters on screen, and characters you typed might be garbled. Click Setup>Serial Port, and select the right COM port and baud rate. Click Control>Reset Terminal to display ASCII characters properly. Click Edit>Clear Screen to clear the terminal window.

Sometimes, you may get these messages, with an 'OSError: can't set AP config'. It is because MicroPython expects an AP (access point) WiFi configuration on booting. Just type in the following commands to enable AP:

```
>>>import network
>>>ap=network.WLAN(network.AP_IF)
>>>ap.active(True)
```

A screenshot of a Tera Term window titled 'COM4 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The terminal area shows the following text:

```
>>> import network
>>> ap=network.WLAN(network.AP_IF)
>>> ap.active(True)
>>> #5 ets_task(4020edc0, 29, 3fff94f8, 10)

>>> ap.active()
True
>>> █
```

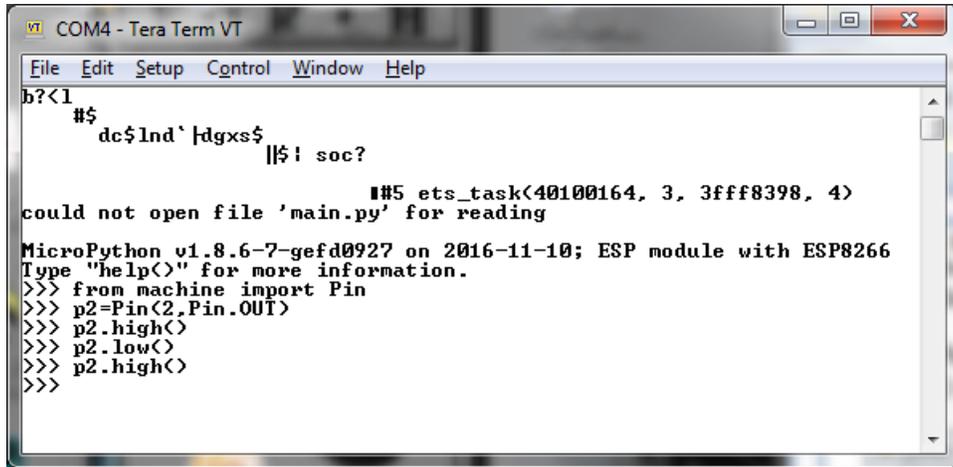
Now, reset NodeMCU by pressing its RST switch, and you will get the sign in window as shown earlier.

You are now in REPL(Read Evaluate Process Loop), which is the interactive interpreter of MicroPython. With REPL, you can turn the LED on and off easily by typing the follow commands:

```
>>>from machine import Pin
>>>p2=Pin(2,Pin.OUT)
>>>p2.low()
>>>p2.high()
```

>>> are prompt characters issued by REPL. You just type in characters after the prompt. Remember, REPL is case sensitive, and you have to type in commands exactly as shown. Misspelling and wrong cases will be rejected with error messages.

p2 is a Pin object, which initializes GPIO Pin 2 as a output pin. After p2 is initialized, issuing p2.low() command turns the on-board LED on. Issuing p2.high() command turns it off.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
b7<1
  # $
    dc $ lnd ` | dgxs $
      || $ ! soc?

      ##5 ets_task(40100164, 3, 3fff8398, 4)
could not open file 'main.py' for reading
MicroPython v1.8.6-7-gef0927 on 2016-11-10; ESP module with ESP8266
Type "help(<)" for more information.
>>> from machine import Pin
>>> p2=Pin(2,Pin.OUT)
>>> p2.high()
>>> p2.low()
>>> p2.high()
>>>
```

You can now turn the on-board LED on and off now manually through TeraTerm and the USB-Serial cable. However, the challenge of this workshop is to switch the LED remotely through WiFi. Then you need WebREPL.

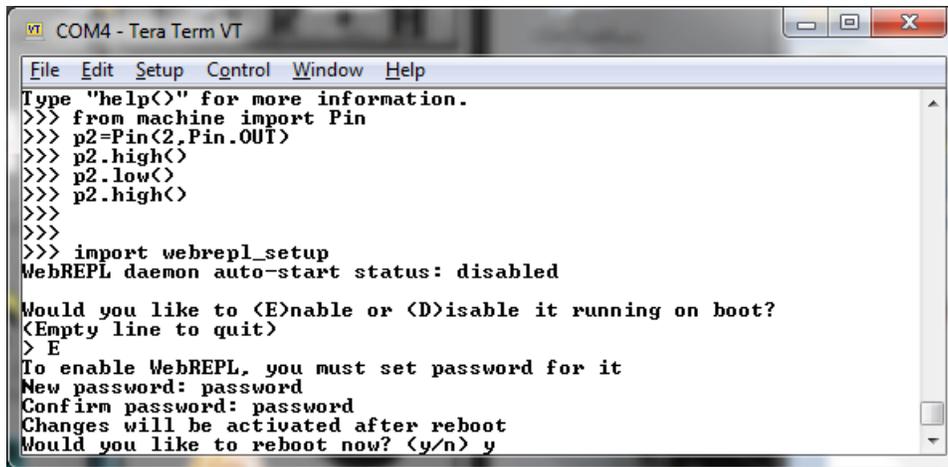
## WebREPL

WebREPL is a special terminal emulator on PC to control a MicroPython microcontroller through WiFi. It is in the form of an html file C:/2017Workshop/webrepl-master/webrepl.html, which you can open in Windows, preferably with Google Chrome.

Normally, MicroPython disables its WebREPL on power-up. To enable WebREPL over WiFi, bring up REPL on TeraTerm, and type:

```
>>>import webrepl_setup
```

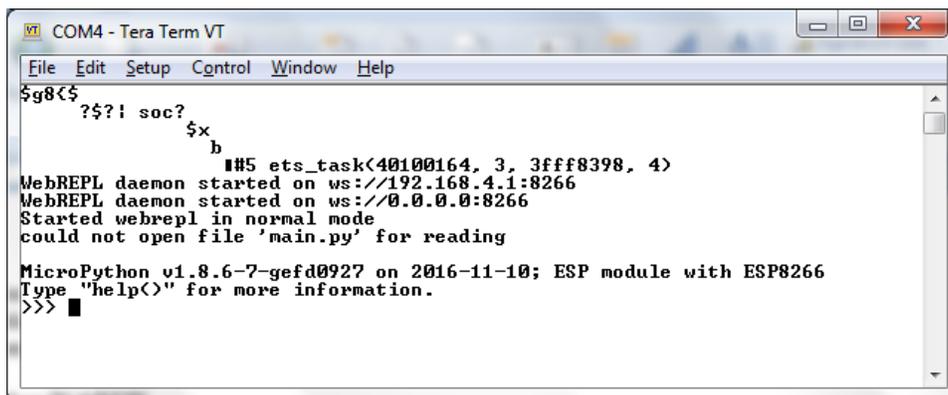
You will be prompted to enable or disable WebREPL, and to select a password. Follow the instructions displayed on REPL. Enter your choice for both password prompts, and then press **y** when prompted to reboot.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Type "help(<)" for more information.
>>> from machine import Pin
>>> p2=Pin(2,Pin.OUT)
>>> p2.high(<)
>>> p2.low(<)
>>> p2.high(<)
>>>
>>>
>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
<Empty line to quit>
> E
To enable WebREPL, you must set password for it
New password: password
Confirm password: password
Changes will be activated after reboot
Would you like to reboot now? (y/n) y
```

After you type ‘y’, and hit Enter key, MicroPython reboots and show you the following window. webREPL is now enabled.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
$g8<$
?${! soc?
      $x
      b
      ##5 ets_task<40100164, 3, 3fff8398, 4>
WebREPL daemon started on ws://192.168.4.1:8266
WebREPL daemon started on ws://0.0.0.0:8266
Started webrepl in normal mode
could not open file 'main.py' for reading

MicroPython v1.8.6-7-gefd0927 on 2016-11-10; ESP module with ESP8266
Type "help(<)" for more information.
>>> █
```

Anytime afterwards, you can reset NodeMCU by pressing its RST switch, and MicroPython will get WebREPL enabled.

Next, you have to connect your PC to NodeMCU board over WiFi.

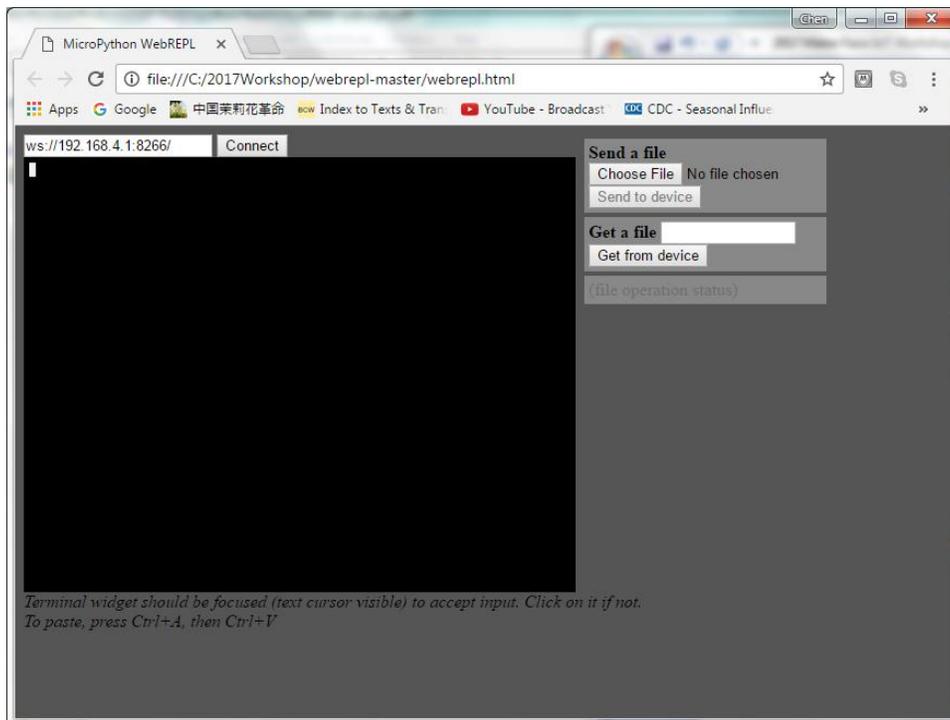
As I discussed a while ago, MicroPython configured itself as an Access Point (AP) on booting. To communicate with it, our PC must be connected it as an AP. As an AP, the ESP8266 chip on NodeMCU board has its own unique SSID name and password.

On PC, go to your Wireless Network Connection panel, which shows all the WiFi stations in its receiving range.



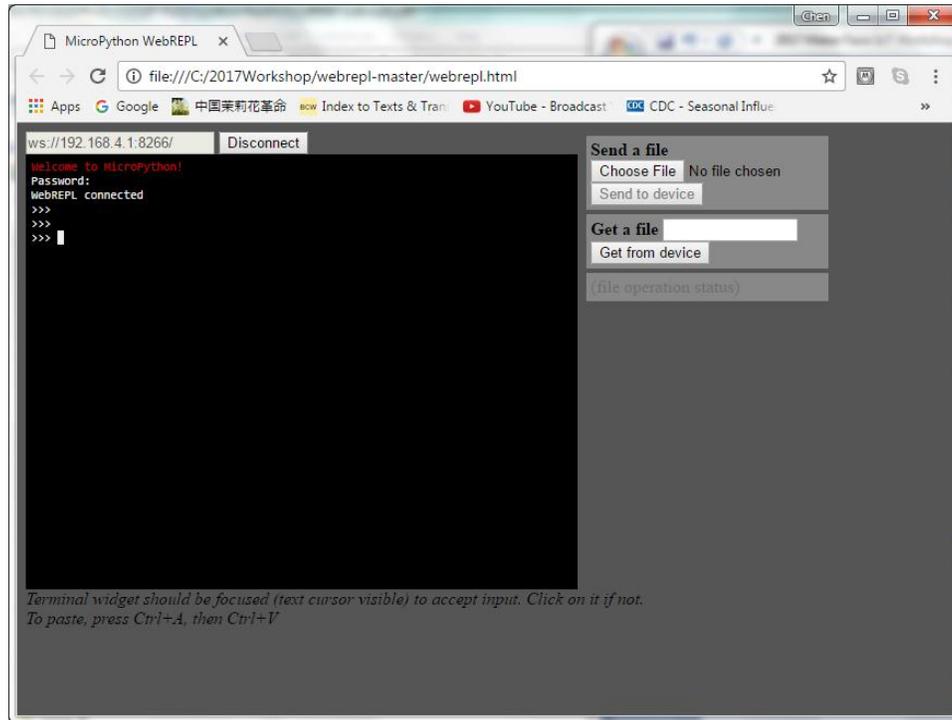
Connect to MicroPython-xxxxxx station, where xxxxxx is the ID of the ESP8266 chip. You will be asked to enter a password to connect. Type in the universal password for NodeMCU: 'micropythonN', ending with a capitalized N.

Now, find the webrepl.html file in the C:/2017Workshop/webrepl-master folder, and double click it. WebREPL web page will be opened:



It shows the IP address 192.168.4.1, and the Port number 8266. Click the Connect button to the right of IP address, and you are asked to enter a password. Type in the password you selected

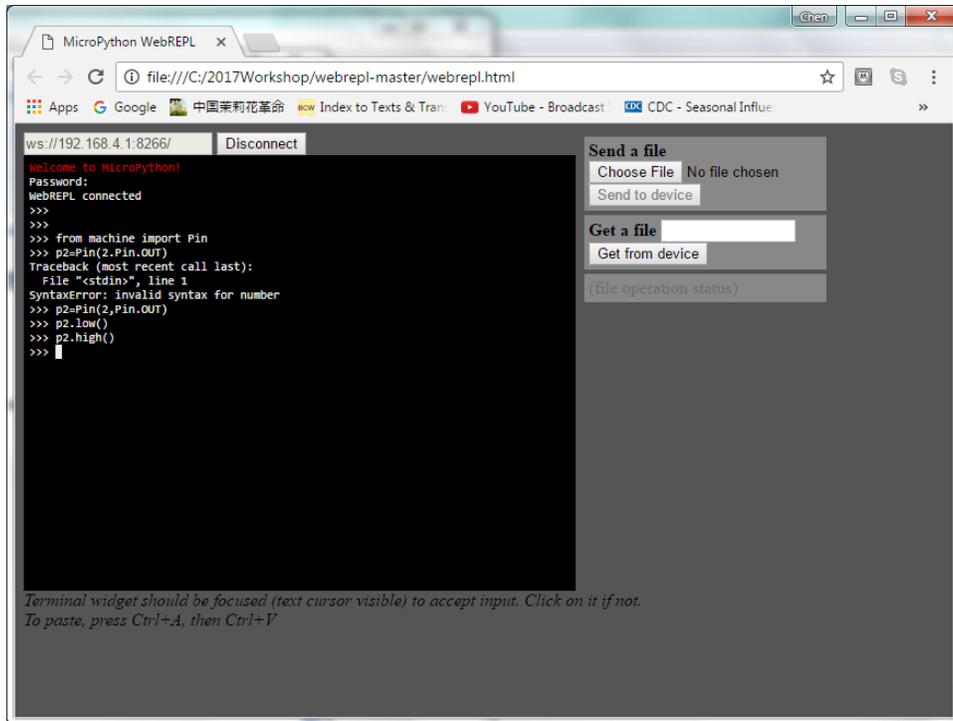
earlier in REPL, and WebREPL is activated. You get the prompts as shown in follow screen, and you can type in MicroPython commands, just like in REPL.



WebREPL acts exactly like REPL, and you can type these commands to turn the LED on and off:

```
>>>from machine import Pin
>>>p2=Pin(2,Pin.OUT)
>>>p2.low()
>>>p2.high()
```

The WebREPL window looks like:



`p2.low()` turns the LED on, and `p2.high()` turns it off.

Congratulations!!! You succeeded to meet our challenge. Take the NodeMCU board home and have lots of fun with it.

In this experiment, you were not asked to do any programming, as MicroPython has library routines controlling the on-board LED. You were using the interpreters REPL and WebREPL to issue the canned commands to do your work. In the other experiments, you have to learn how to program NodeMCU to meet the challenge. It is more fun to get NodeMCU to do what you like it to do, than just following directions to do something other people allow you to do.

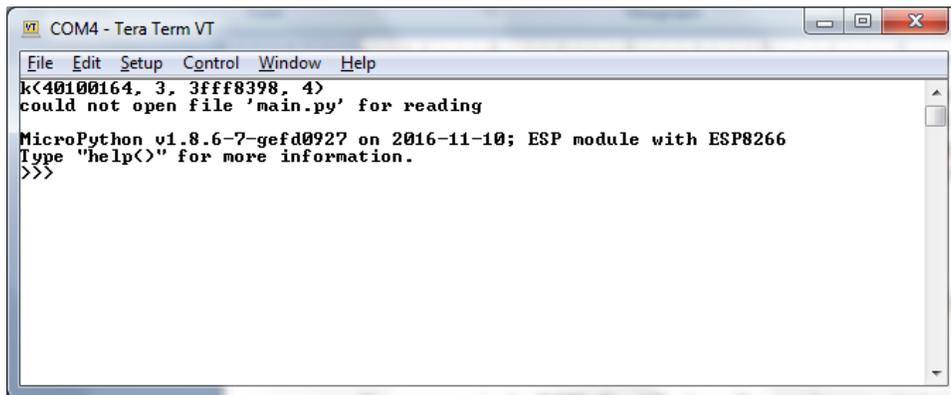
## Experiment 4. MicroPython UDP Server

In this experiment, you will set up a MicroPython UDP server on NodeMCU board. The server waits for UDP packets from a client, directing it to turn the on-board LED on and off. It can also drive a speaker to beep at different frequencies, if you had a small speaker connected to NodeMCU. The client is Hercules SETUP Utility on PC, which sends out UDP packets to the MicroPython server to turn the LED on and off.

First, you have to load MicroPython binary in the flash memory of NodeMCU. Follow the directions in the Tool 3 ESP8266Flasher section earlier.

### MicroPython REPL

After MicroPython is loaded into the flash memory of NodeMCU. Open TeraTerm on PC. Press the reset switch RST on NodeMCU board to get MicroPython REPL running:

A screenshot of a TeraTerm window titled "COM4 - Tera Term VT". The window shows the MicroPython REPL interface. The output text is: "k<40100164, 3, 3fff8398, 4> could not open file 'main.py' for reading", "MicroPython v1.8.6-7-gef0927 on 2016-11-10; ESP module with ESP8266", "Type 'help(<)' for more information.", and the prompt ">>>".

```
COM4 - Tera Term VT
File Edit Setup Control Window Help
k<40100164, 3, 3fff8398, 4>
could not open file 'main.py' for reading
MicroPython v1.8.6-7-gef0927 on 2016-11-10; ESP module with ESP8266
Type "help(<)" for more information.
>>>
```

With REPL, you can turn the LED on and off easily by typing the follow commands:

```
>>>from machine import Pin
>>>p2=Pin(2,Pin.OUT)
>>>p2.low()
>>>p2.high()
```

### MicroPython UDP Server

The server on MicroPython is very simple. The code is as follows:

```
import machine,time
p14=machine.PWM(machine.Pin(14, 1))
p2=machine.Pin(2,machine.Pin.OUT)
p2.low()
def beep(n):
    if n:
        p14.duty(512)
        p14.freq(n)
```

```

        p2.low()
    else:
        p14.duty(0)
        p2.high()

import network
sta=network.WLAN(network.STA_IF)
sta.connect('SVFIG','12345678')
#static IP
#sta.ifconfig(('192.168.1.10','255.255.255.0','192.168.1.1','192.168.1.1'))
time.sleep(1)
newconfig=sta.ifconfig()
print(newconfig)

import socket
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
addr=(newconfig[0],8266)
print(addr)
s.bind(addr)

def listen():
    while True:
        data,address=s.recvfrom(10)
        beep(int(data))
listen()

```

The first section of the code defines a routine beep() which produces an audio beep and turns the LED on, given a frequency parameter. If the frequency is 0, stop the audio and also turns the LED off.

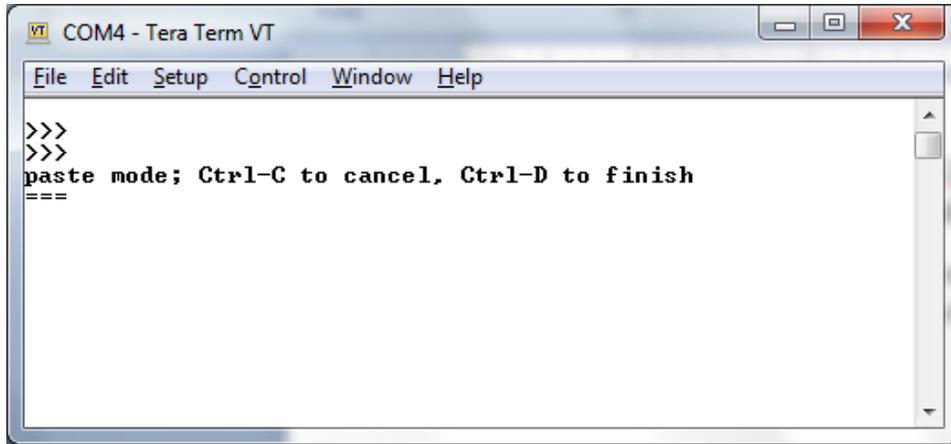
The second section establishes NodeMCU as a server station, connection to the WiFi network. The IP address can be assigned dynamically or statically.

The third section sets up a socket to receive UDP packets. Finally, the routine listen() receives UDP packets and drives the audio and the LED.

This piece of code must be stored in the flash memory of NodeMCU as main.py file, which is executed when NodeMCU boots up, and performs server services.

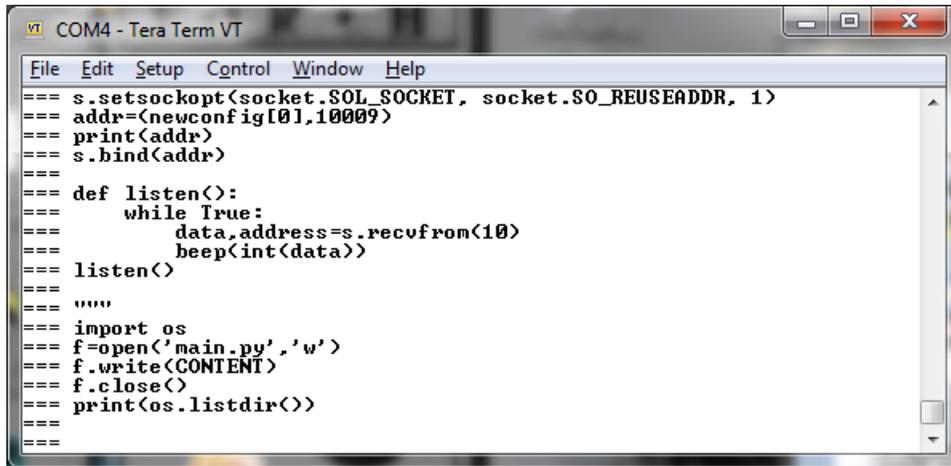
This code is embedded in a loader file WriteFile.py, which is executed by REPL to generate main.py. Following these steps:

In REPL, press Ctrl-E to enter into the 'Paste Mode':



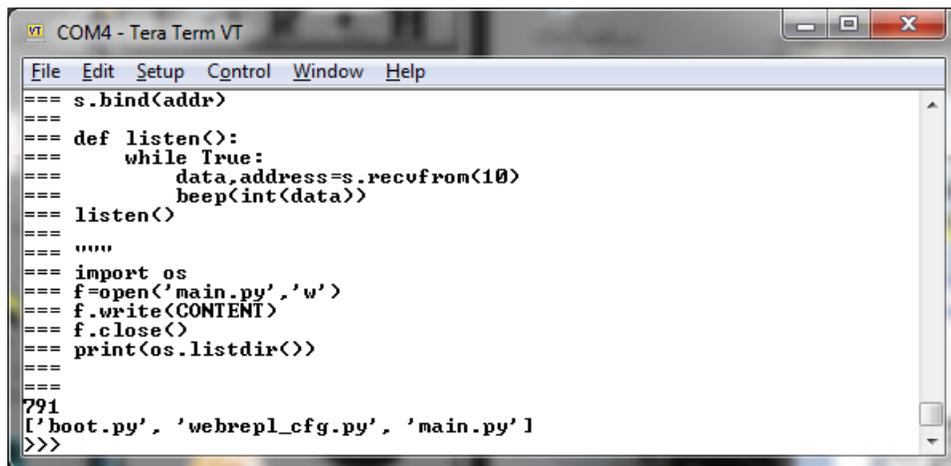
```
COM4 - Tera Term VT
File Edit Setup Control Window Help
>>>
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
===
```

Click File>Send File... option, and then chose WriteFile.py. It is pasted into REPL:



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
=== s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
=== addr=(newconfig[0],10009)
=== print(addr)
=== s.bind(addr)
===
=== def listen():
===     while True:
===         data,address=s.recvfrom(10)
===         beep(int(data))
=== listen()
===
=== """
=== import os
=== f=open('main.py','w')
=== f.write(CONTENT)
=== f.close()
=== print(os.listdir())
===
===
```

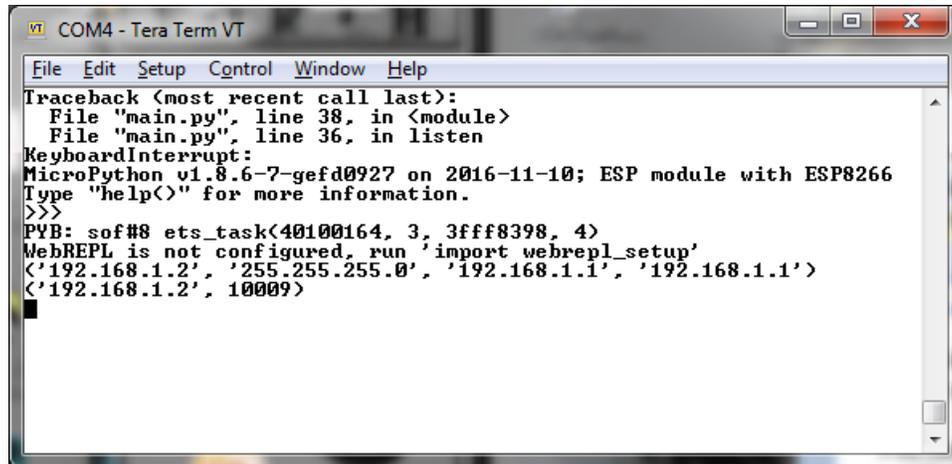
Press Ctrl-D to exit paste mode, and also compile the code pasted.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
=== s.bind(addr)
===
=== def listen():
===     while True:
===         data,address=s.recvfrom(10)
===         beep(int(data))
=== listen()
===
=== """
=== import os
=== f=open('main.py','w')
=== f.write(CONTENT)
=== f.close()
=== print(os.listdir())
===
===
791
['boot.py', 'webrepl_cfg.py', 'main.py']
>>>
```

791 bytes were received in paste mode. Code is compiled and a main.py file is written to the flash memory. When MicroPython re-boots, it automatically compiles main.py. You put your application code in main.py, and NodeMCU will execute your code when you power it up. This is what I called firmware engineering.

You can type in 'import main', to exercise the code. Or, you can press Ctrl-D to do a software reset in MicroPython to start the server. Or, you can press the RST switch on NodeMCU board for a hardware reset to start the server. After reset, you will have this screen in TeraTerm:



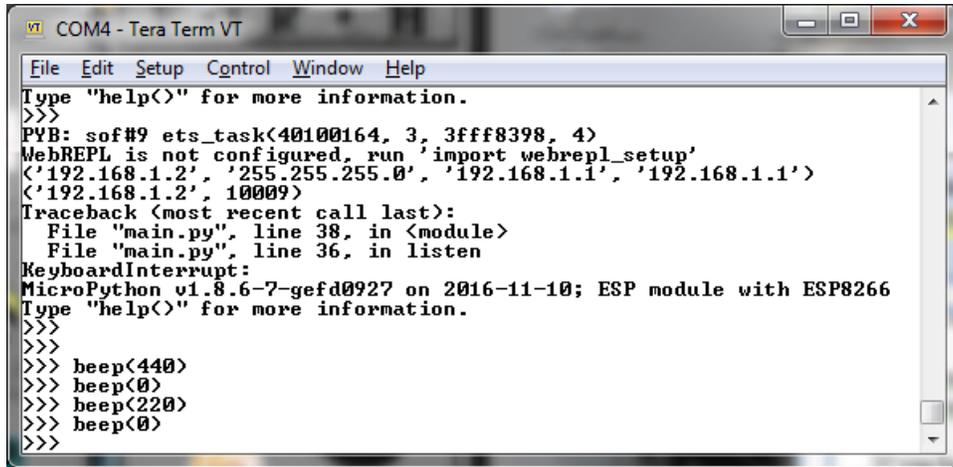
```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Traceback (most recent call last):
  File "main.py", line 38, in <module>
    File "main.py", line 36, in listen
KeyboardInterrupt:
MicroPython v1.8.6-7-gefd0927 on 2016-11-10; ESP module with ESP8266
Type "help(<)" for more information.
>>>
PYB: sof#8 ets_task(40100164, 3, 3fff8398, 4)
WebREPL is not configured, run 'import webrepl_setup'
<'192.168.1.2', '255.255.255.0', '192.168.1.1', '192.168.1.1'>
```

The server is now running. Note that its IP address is 192.168.1.2, or whatever is shown on TeraTerm screen. Its Port number is 10009. These numbers will be used in Hercules Client to send UDP packets to this server.

Press Ctrl-C to return to REPL. You can type the following commands to turn the LED on and off:

```
>>>beep (440)
>>>beep (0)
```

If you have a small speaker connected to GND and D5, you can hear the audio beep at the specified frequency. beep (440) makes the speaker beep at 440 Hz, and also turns the LED on. beep (0) silences the speaker, and also turns the LED off.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
Type "help(<)" for more information.
>>>
PYB: sof#9 ets_task(40100164, 3, 3fff8398, 4)
WebREPL is not configured, run 'import webrepl_setup'
<'192.168.1.2', '255.255.255.0', '192.168.1.1', '192.168.1.1'>
<'192.168.1.2', 10009>
Traceback (most recent call last):
  File "main.py", line 38, in <module>
  File "main.py", line 36, in listen
KeyboardInterrupt:
MicroPython v1.8.6-7-gefd0927 on 2016-11-10; ESP module with ESP8266
Type "help(<)" for more information.
>>>
>>>
>>> beep(440)
>>> beep(0)
>>> beep(220)
>>> beep(0)
>>>
```

Here you demonstrated that you can turn the LED on and off remotely. You've met our challenge. Take a NodeMCU board home and do more challenging experiments.

## Experiment 5. Lua/ESPlorer

When the Chinese engineers in Espressif Systems released ESP8266, no high level programming tools were available. Then a group of Russian engineers ported the Brazilian programming language Lua to ESP8266, and released an IDE ESPlorer with it. It was the first high level language allowing users to program ESP8266 for IoT applications. Then we had Arduino IDE and MicroPython with different approaches to program ESP8266.

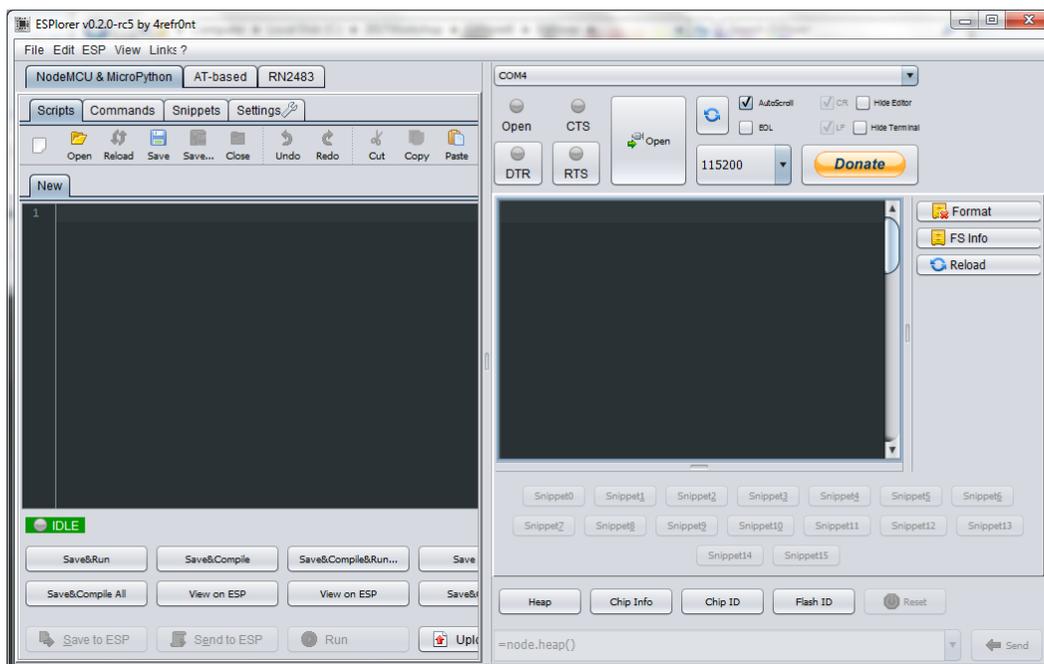
In this experiment, you will control the LED on NodeMCU board, first with ESPlorer interactively, and then remotely with an UDP client Hercules.

First you have to flash ESP8266 with Lua. Hook your NodeMCU board with your PC with the MicroUSB cable. Follow the instructions in the Tool 3 ESP8266Flasher section of this workshop.

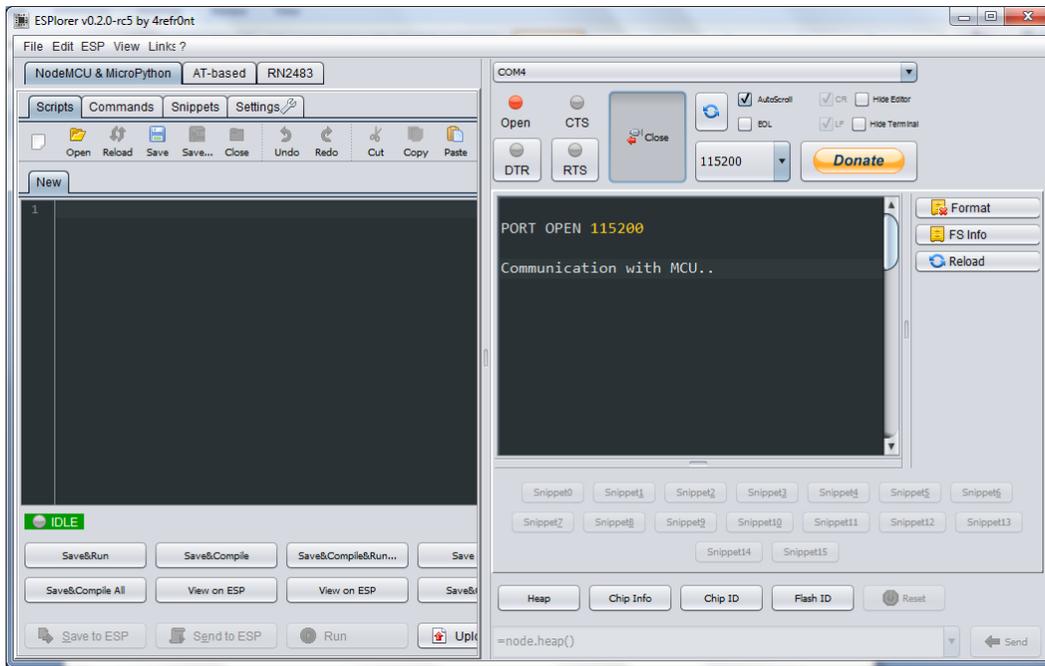
Lua language system came in the form of two binary images 0x00000.bin and 0x10000.bin in the C:/2017Workshop/wb6 folder. They are to be flashed into the flash memory of NodeMCU board, using ESP8266Flasher.exe.

### ESPlorer IDE

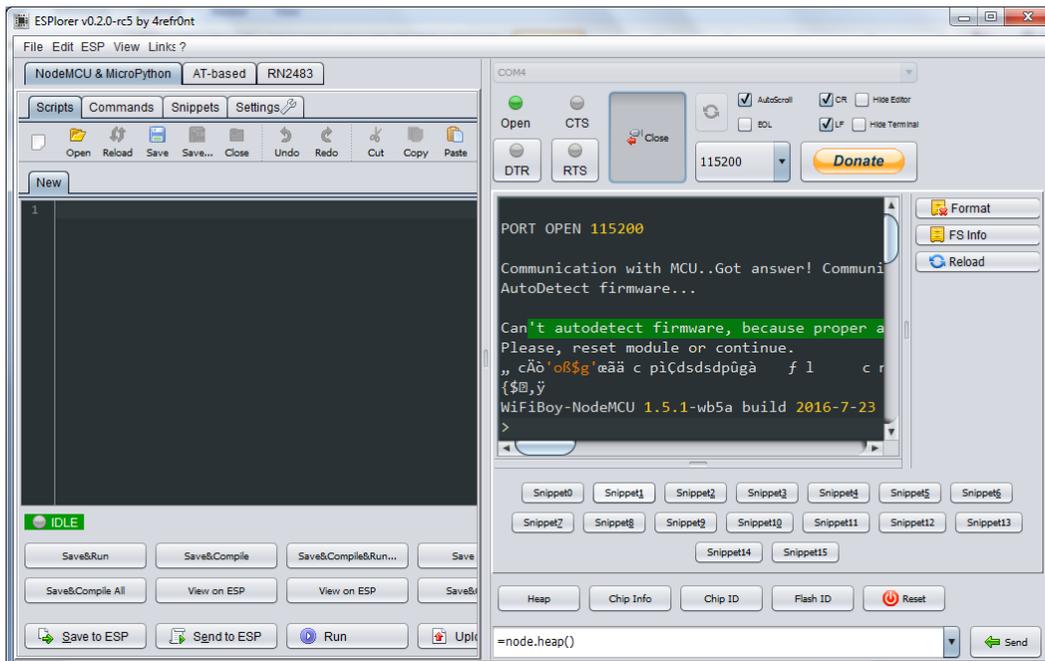
To bring up ESPlorer IDE, double click c:/2017Workshop/Esplorer6/Esplorer/ESPlorer.jar, and you get the ESPlorer windows:



On the right is the Editor panel, and on the left is the Terminal panel. On the top of Terminal panel, make sure that the right COM port is selected, and that the baud rate is set to 115200. Then click the big Open button to start communicating with NodeMCU board:



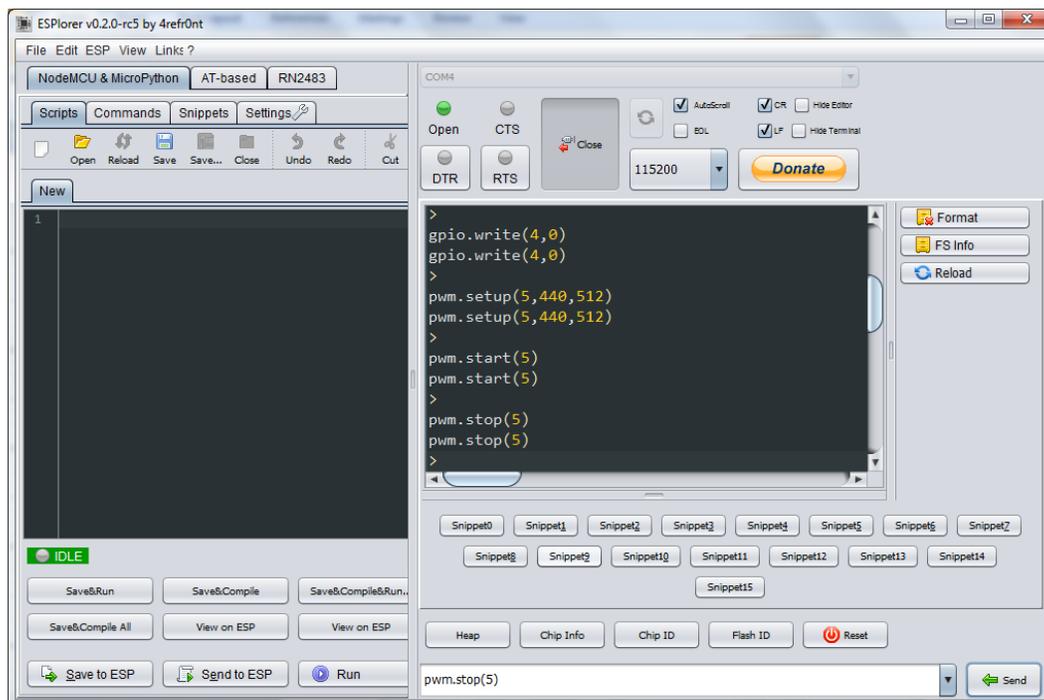
On the Terminal panel, it shows “Communication with MCU..” Now, press the RST reset switch on NodeMCU board and Lua signs in:



At the bottom of the Terminal Panel, there is a text box for you to enter Lua commands. Type in these commands to turn the LED on and off, and to generate an audio beep at 440 Hz:

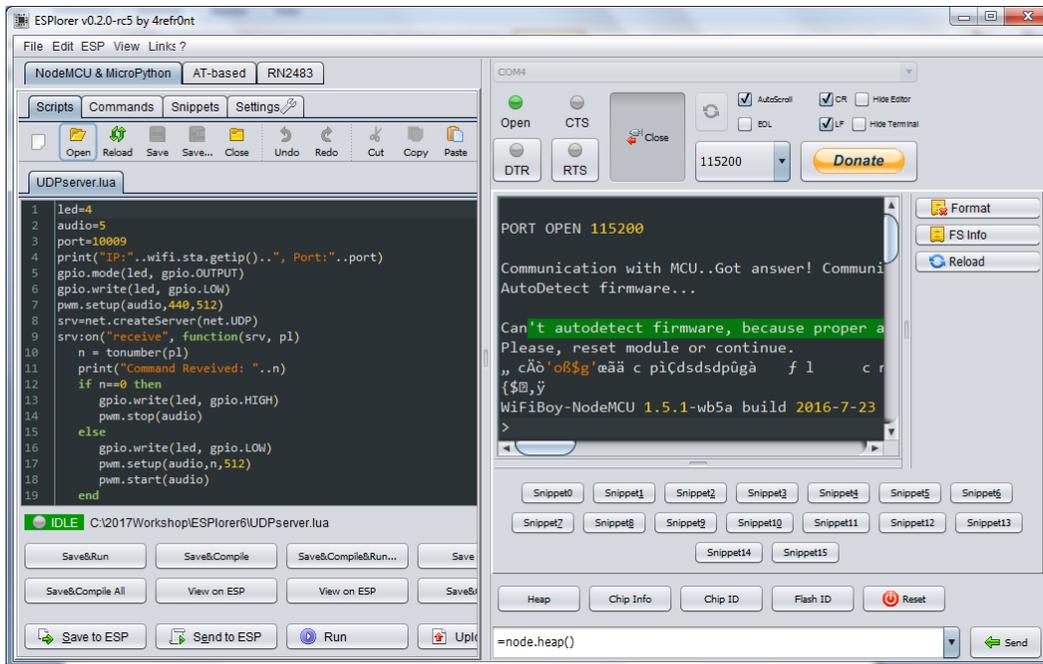
```
gpio.write(4,0)
gpio.write(4,1)
pwm.setup(5,440,512)
pwm.start(5)
pwm.stop(5)
```

Type one line of code at a time, and press the Send button to the right of the text box.



## UDP Server

The code to turn LED on and off remotely is also easy. The server code receiving UDP packets is in a Lua file C:/2017Workshop/UDPserver.lua. In the Editor panel, open this file:



Source code in UDPserver.lua is very simple, as shown here:

```

led=4
audio=5
port=10009
print("IP: "..wifi.sta.getip()..", Port: "..port)
gpio.mode(led, gpio.OUTPUT)
gpio.write(led, gpio.LOW)
pwm.setup(audio, 440, 512)
srv=net.createServer(net.UDP)
srv:on("receive", function(srv, pl)
  n = tonumber(pl)
  print("Command Reveived: "..n)
  if n==0 then
    gpio.write(led, gpio.HIGH)
    pwm.stop(audio)
  else
    gpio.write(led, gpio.LOW)
    pwm.setup(audio, n, 512)
    pwm.start(audio)
  end
end)
srv:listen(port)

```

The Lua commands accomplish these functions:

```

wifi.sta.getip()
gpio.mode(led, gpio.OUTPUT)
gpio.write(led, gpio.LOW)
pwm.setup(audio, 440, 512)
srv=net.createServer(net.UDP)

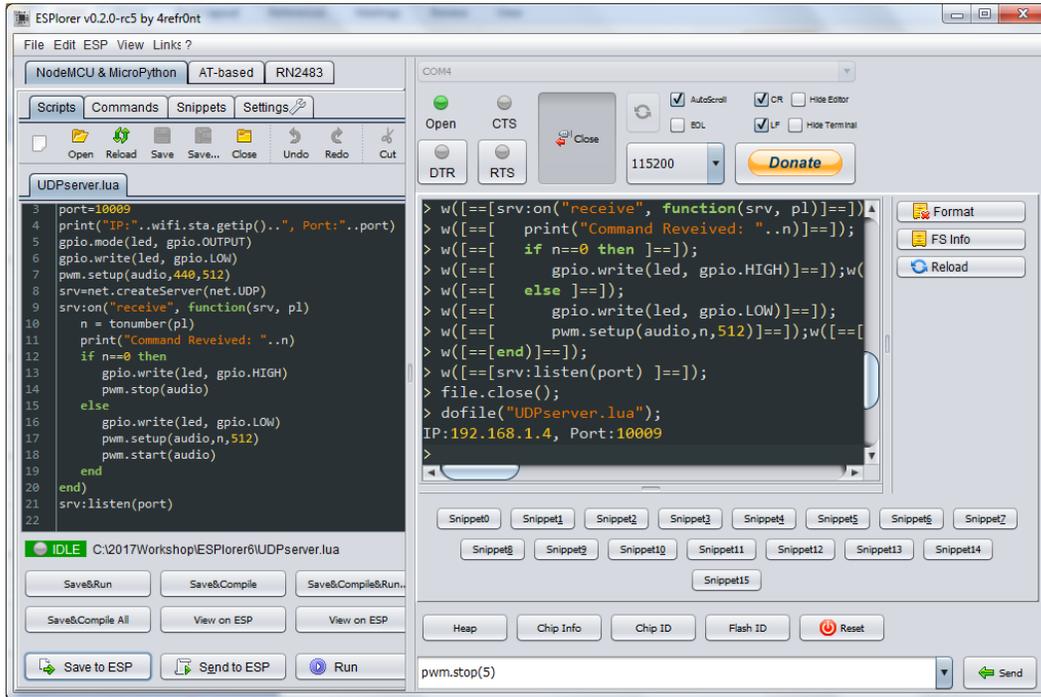
```

- obtains the IP address on the current WiFi network
- initializes Pin 4 as output to LED
- turns on LED
- sets up Pin 5 for PWM output
- sets up NodeMCU as an UDP server

srv:on("receive", function(srv, pl) receives payload pl in an UDP packet

If the payload in a UDP packet is a number 0, turn off LED and silence beeper. If the payload is a number other than 0, use it as the frequency for the beeper, and also turn the LED on.

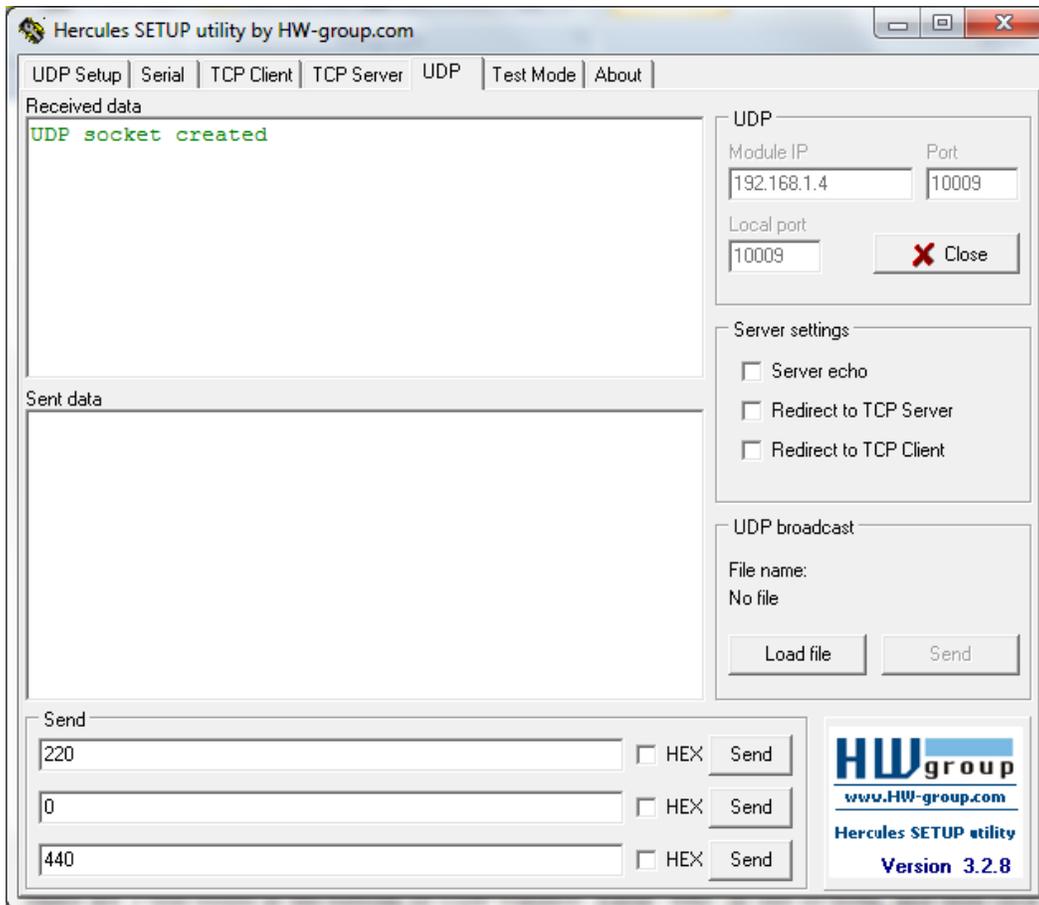
Now, press the Save to ESP button at the bottom left corner of Editor panel. This UDPserver.lua file is sent to NodeMCU and compiled.



Notice that in the Terminal panel, Lua reports that the IP address is 192.168.1.4, and the Port number is 10009. These numbers characterizes the UDP server socket on NodeMCU.

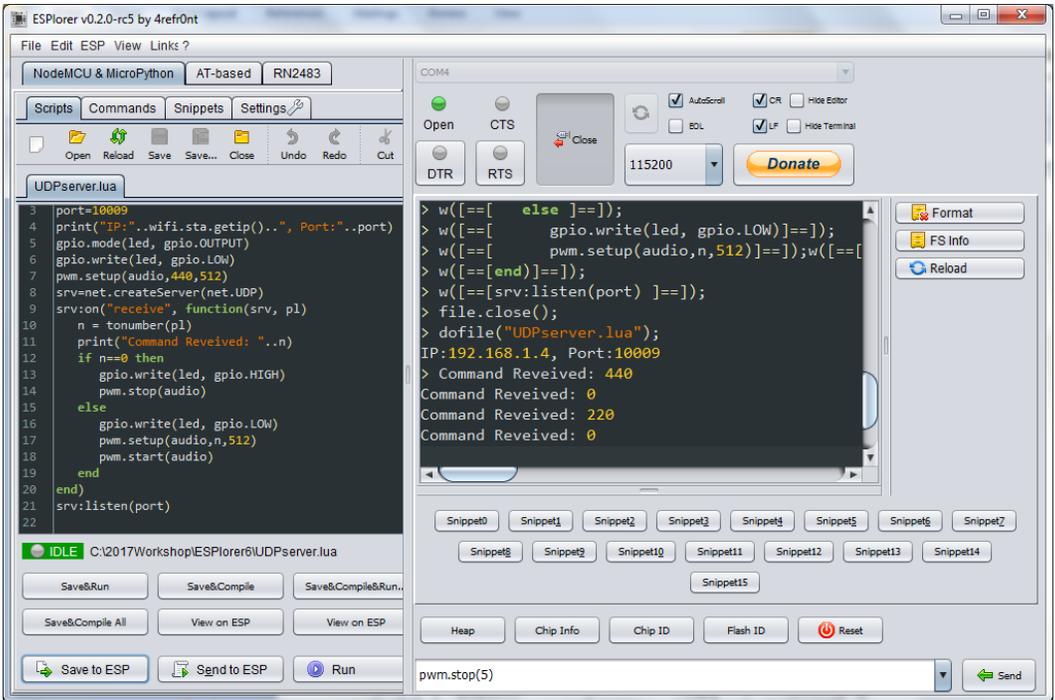
## Hercules SETUP Utility

Now, open Hercules SETUP Utility. Refer to Tool 4 Section for details. Select UDP menu, enter the IP/Port addresses, and then click the Listen button. The Listen button changes to a Close button:



There are 3 text boxes at the bottom of UDP window. Enter “440” in one of them, and then click the Send button to its left. Hercules send a “440” packet to NodeMCU. The on-board LED will turn on, and the speaker beeps at 440 Hz. Enter a “0” in another text box, and click the corresponding Send button, the LED is turned off, and the speaker is silenced. This way you can control NodeMCU board remotely.

ESPlorer Terminal panel will show the commands received from Hercules.



According to ESPlorer manual, you can save UDPserver.lua file as an init.lua file in the flash memory of NodeMCU. After that, if you press RST switch on NodeMCU, the init.lua file will be loaded after ESP8266 resets, and you can send UDP packets to it at will. However, on my NodeMCU board, init.lua is not loaded automatically, and I have to load it manually for UDP communication.

## Postlude

With the advent of personal computers and cell phones, the entire humanity is turned into monkeys trained to push buttons. WiFi become so ubiquitous that we don't even think about it. We just push buttons, and are fully occupied. Are we happy? I am not sure.

Here you are challenged to do a very simple task: turn an LED on and off remotely over WiFi. It is really quite complicated. It is almost impossible without ESP8266 on a NodeMCU board. ESP8266 solved the hardware and software problems so that you can meet this challenge in a \$3.18 kit. But, it requires more than a monkey to do it.

Moreover, as we firmware engineers always claim: "If you can turn a LED on and off, you can do anything!"

Having proven that you are more than a button pushing monkey, go ahead and do something useful in the IoT world.