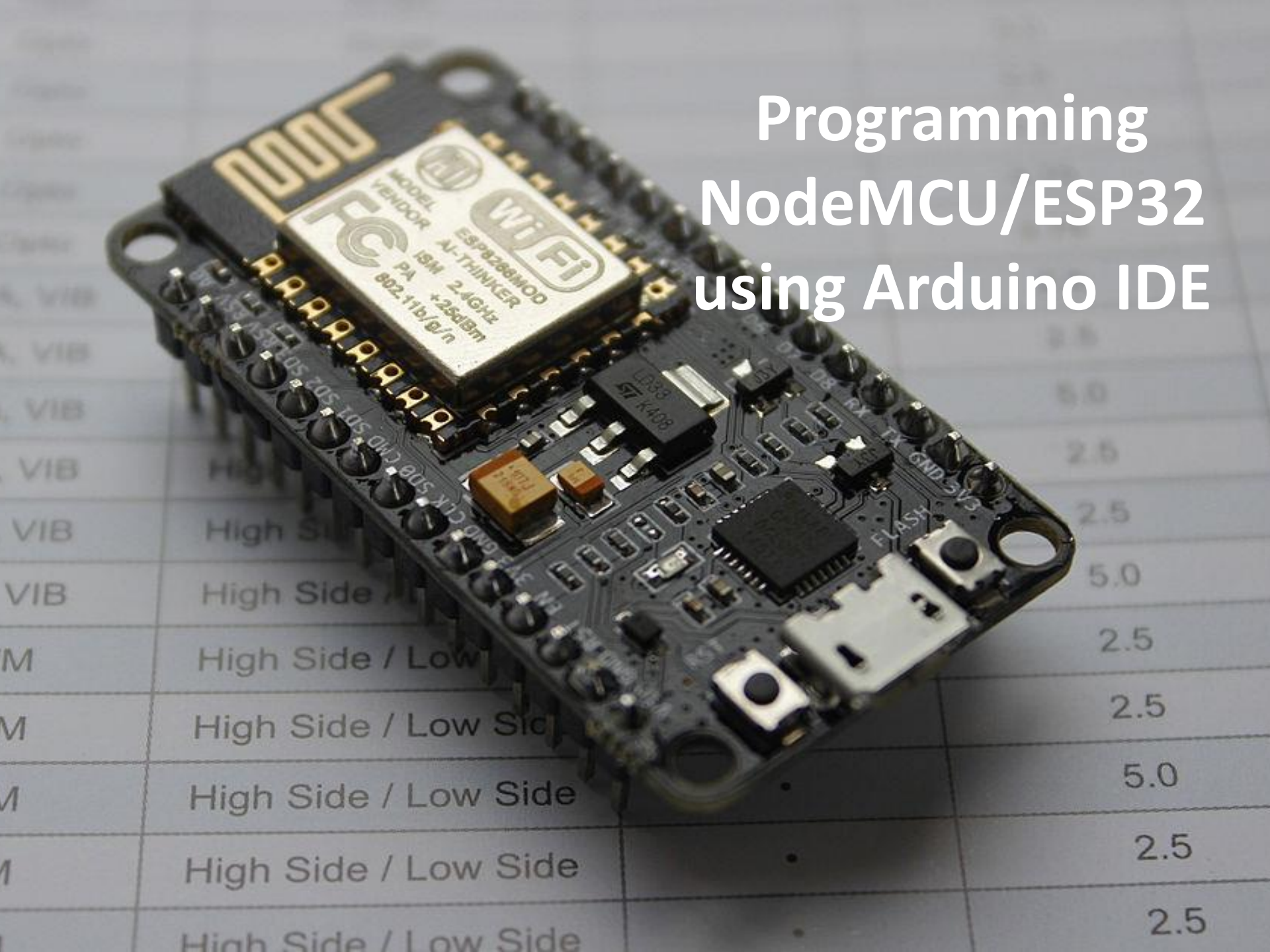
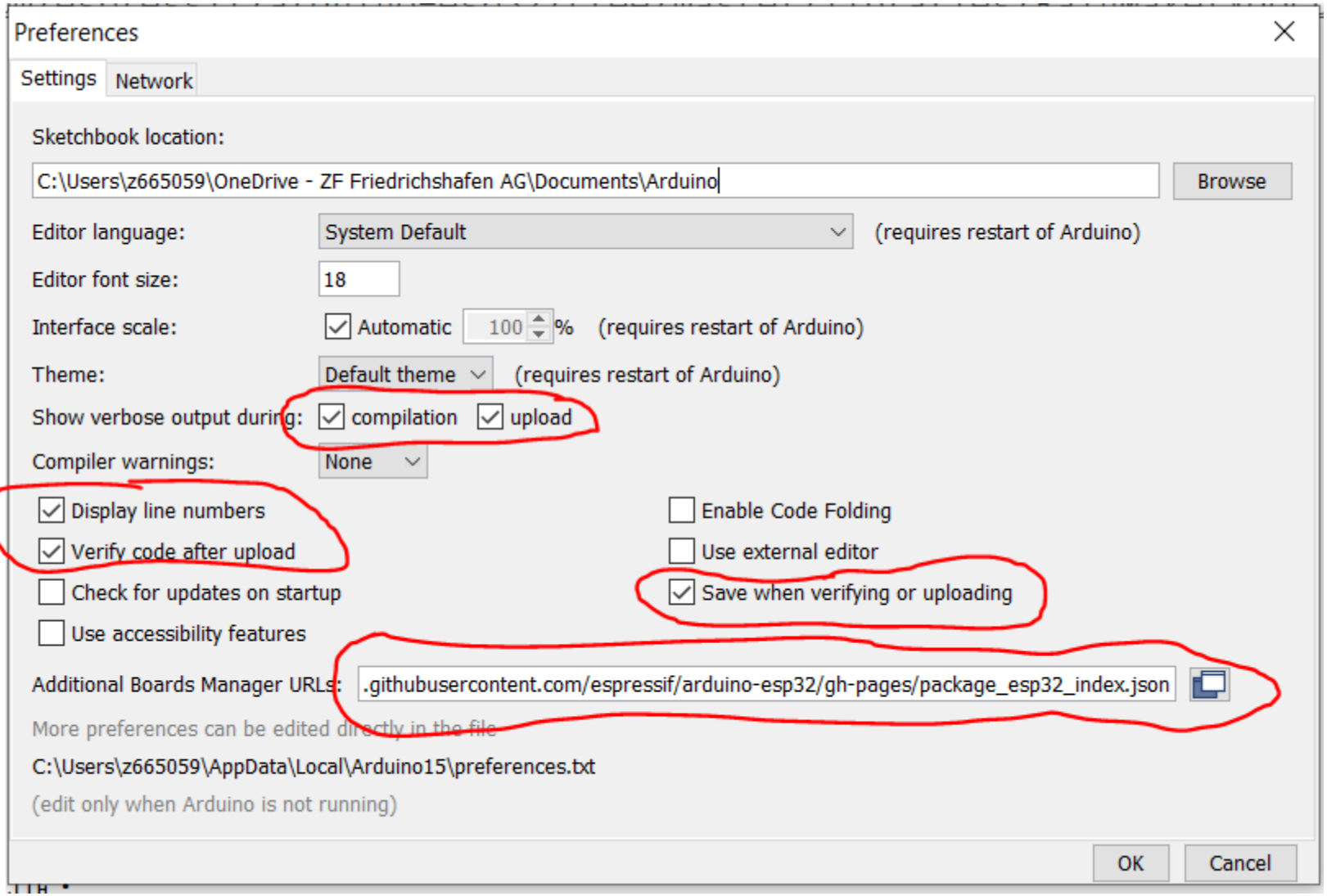

Programming
NodeMCU/ESP32
using Arduino IDE

ESP32

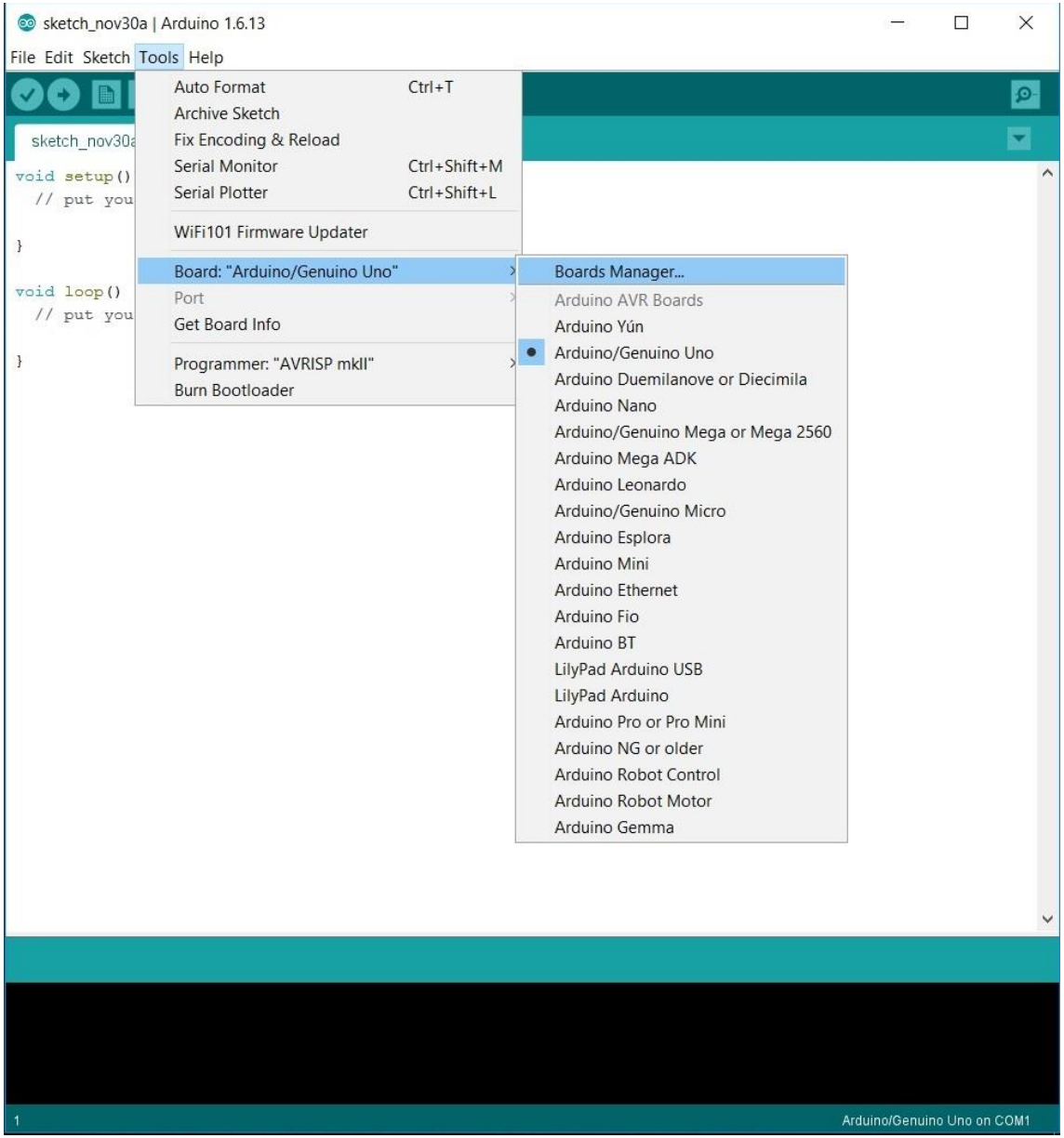
- Open the Arduino IDE.
- Go to **Files** and click on the **Preference** in the Arduino IDE.
- Copy the URL below in the **Additional boards Manager.**

  https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

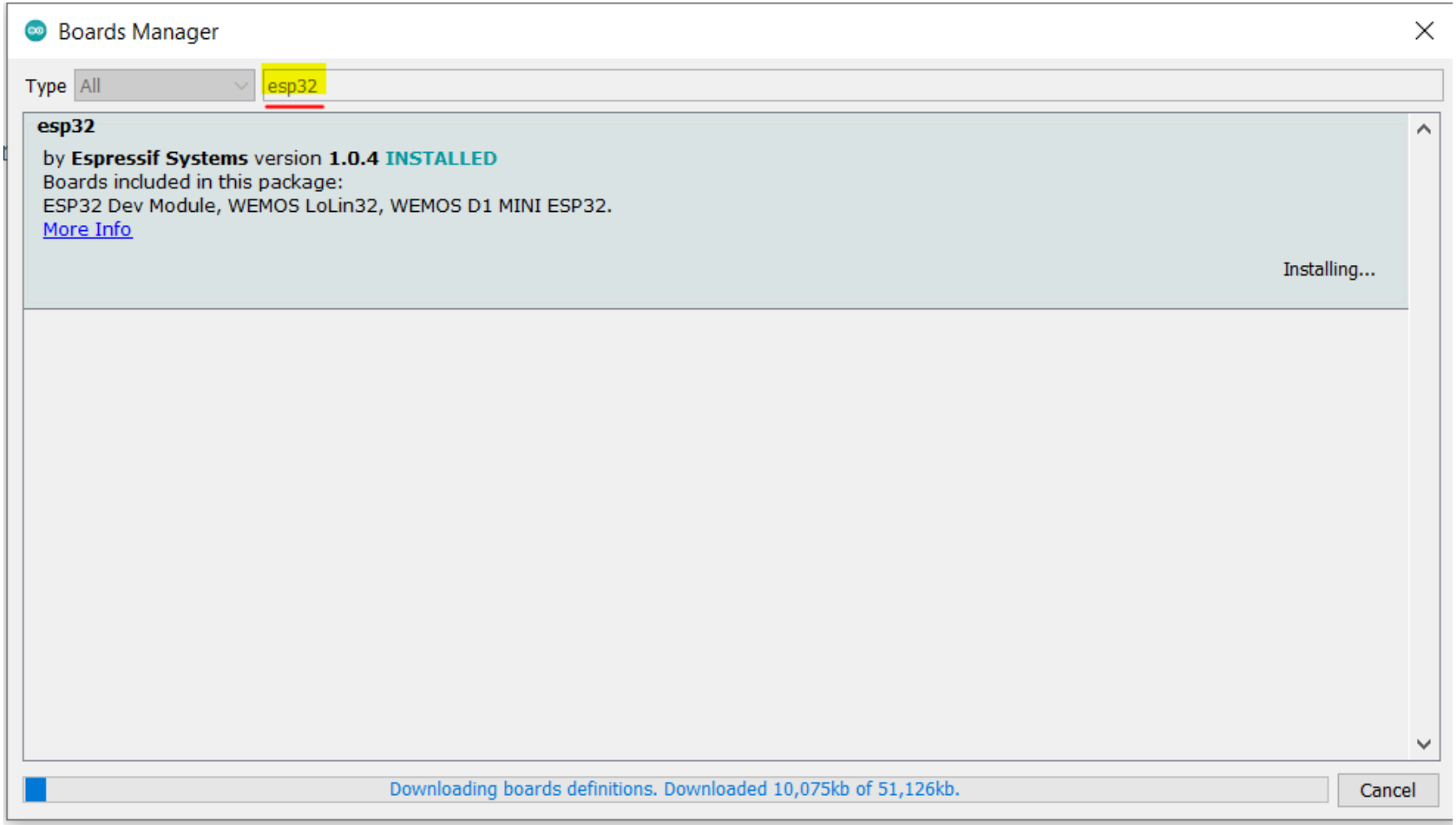
- Click **OK** to close the preference Tab.


Preferences
Settings
Network
Sketchbook location:
C:\Users\z665059\OneDrive - ZF Friedrichshafen AG\Documents\Arduino
Browse
Editor language:
System Default
(requires restart of Arduino)
Editor font size:
18
Interface scale:
Automatic
100
%
(requires restart of Arduino)
Theme:
Default theme
(requires restart of Arduino)
Show verbose output during:
compilation
upload
Compiler warnings:
None
Display line numbers
Verify code after upload
Check for updates on startup
Use accessibility features
Enable Code Folding
Use external editor
Save when verifying or uploading
Additional Boards Manager URLs:
.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
More preferences can be edited directly in the file
C:\Users\z665059\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)
OK
Cancel

- Go to **Tools** and **Board**, and then select **Board Manager.**

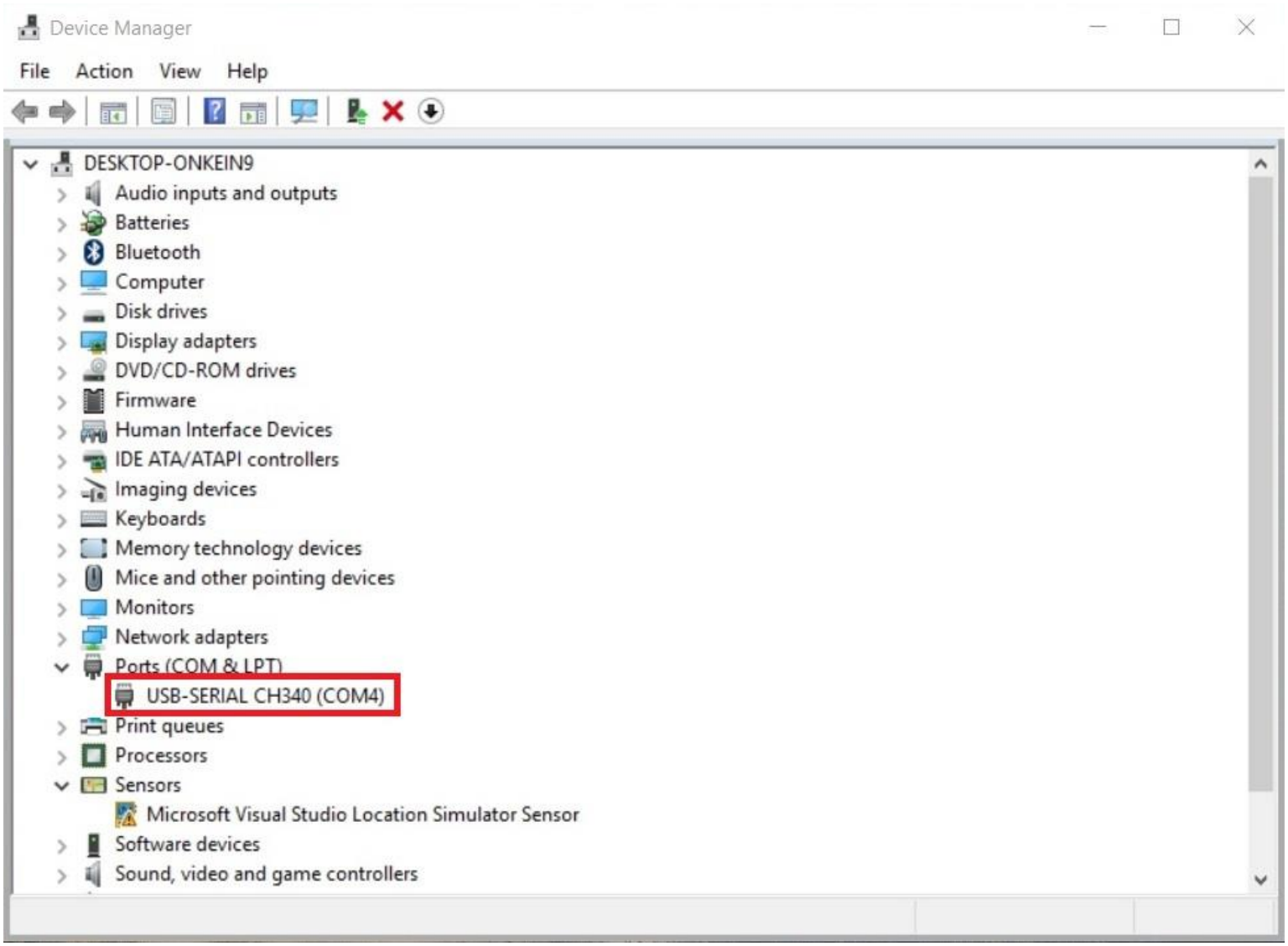


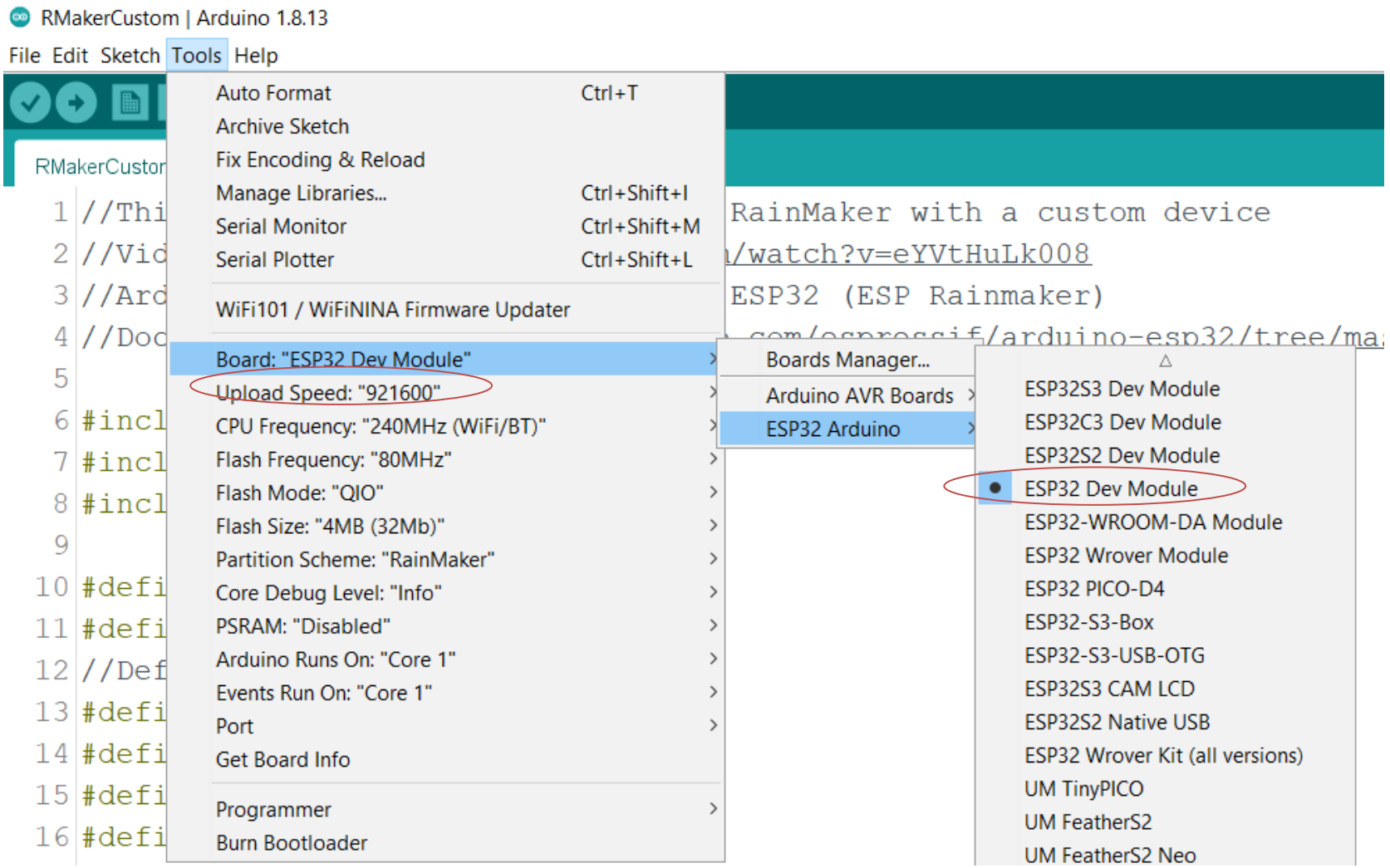
- Navigate to **esp32 by Espressif Systems** and install the software for Arduino.



- Once all the above process has been completed we are ready to program our **ESP32** with Arduino IDE.

- Connect ESP32 to PC with USB to microUSB cable .
- Windows should automatically download necessary drivers for it.



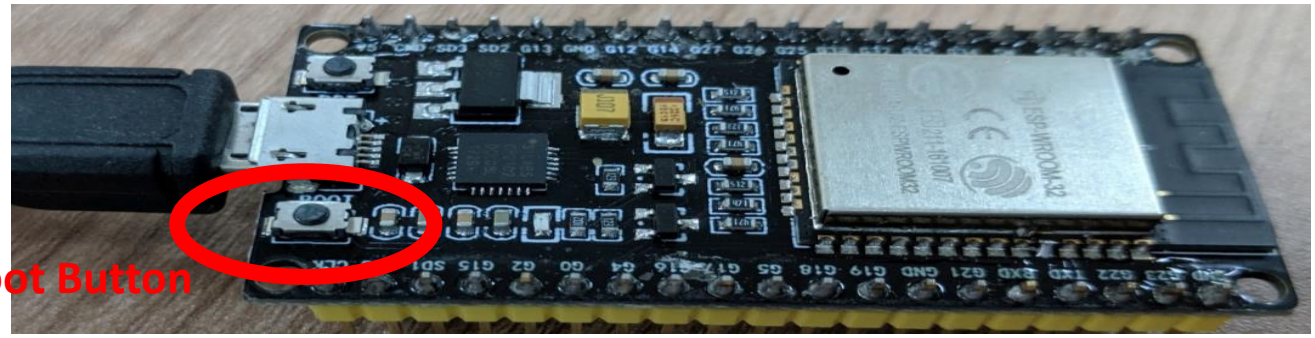

RMakerCustom | Arduino 1.8.13
File Edit Sketch Tools Help
RMakerCustor
1 //Thi
2 //Vid
3 //Ard
4 //Doc
5
6 #incl
7 #incl
8 #incl
9
10 #defi
11 #defi
12 //Def
13 #defi
14 #defi
15 #defi
16 #defi
RainMaker with a custom device
/watch?v=eYVtHuLk008
ESP32 (ESP Rainmaker)
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "RainMaker"
Core Debug Level: "Info"
PSRAM: "Disabled"
Arduino Runs On: "Core 1"
Events Run On: "Core 1"
Port
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Arduino AVR Boards
ESP32 Arduino
ESP32S3 Dev Module
ESP32C3 Dev Module
ESP32S2 Dev Module
ESP32 Dev Module
ESP32-WROOM-DA Module
ESP32 Wrover Module
ESP32 PICO-D4
ESP32-S3-Box
ESP32-S3-USB-OTG
ESP32S3 CAM LCD
ESP32S2 Native USB
ESP32 Wrover Kit (all versions)
UM TinyPICO
UM FeatherS2
UM FeatherS2 Neo

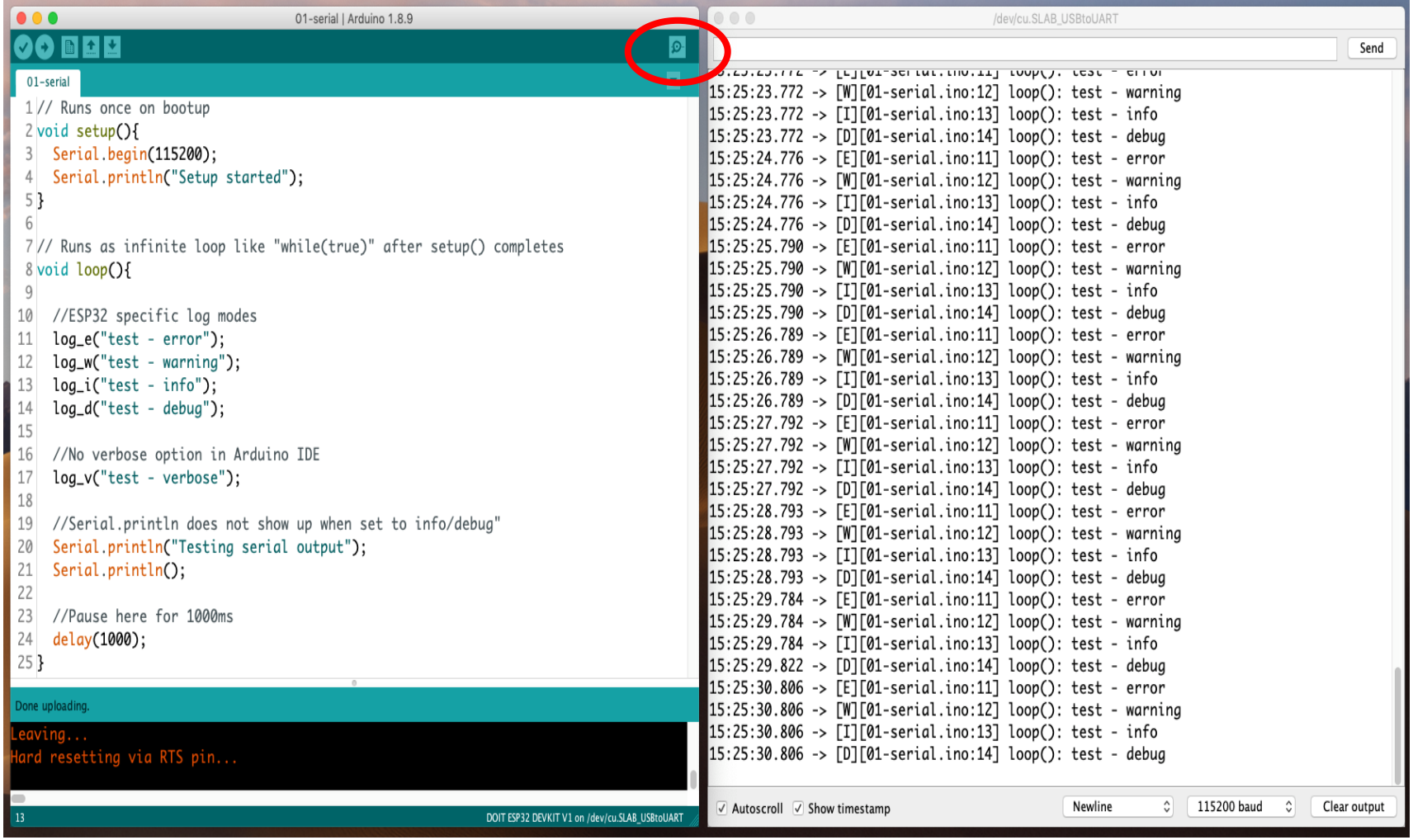
01-Serial

1. Verify Portable Arduino SDK installed correctly by Blinking OnBoard Led
2. How to flash an ESP32 program?

   By Pressing BOOT Button when *connecting……*

3. Using the serial console and ESP32 log facility





- ***Magnifying glass opens serial console***
- Show Timestamps
- Newline
- ***115200 baud***

02-Blinky

1. Do NOT Connect an LED, use OnBoard LED at pin no 2
2. Blink from the LED
3. #define PIN_LED 2

Blinky Video kept in 02-Blinky Folder

Pre-Conditions Before Flashing

1. sudo apt update
2. Python 2.7
   - sudo apt install python-pip
   - pip install pyserial
3. Python 3
   - sudo apt install python3-pip
   - pip3 install pyserial
4. Check USB Cable
   - In terminal type *dmesg --follow*
   - *Look for*

   "cp210x converter detected"

   "cp210x converter now attached to ttyUSB0"
5. *Set Permissions to Arduino for Accessing Board at port ttyUSB0*
   - *sudo usermod –a –G dialout $USER*
   - *sudo chmod a+rw /dev/ttyUSB0*
   - *Reboot your machine*

Basics of Wi-Fi

- Devices that connect to Wi-Fi network are called stations.
- Connection to Wi-Fi is provided by an access point, that acts as a hub for one or more stations.
- Each access point is recognized by a SSID (**S**ervice **S**et **ID**entifier).
- Clients can access services provided by servers in order to send, receive and process data.
- Server provide functionality to other programs or devices, called clients.
- We need to use ESP8266WiFi library to make Wi-Fi applications on NodeMCU.

WiFi library for ESP8266

- WiFi.begin(ssid, password, channel, bssid, connect)

  Meaning of parameters is as follows:

  ssid - a character string containing the SSID of Access Point we would like to connect to, may have up to 32 characters

  password to the access point, a character string that should be minimum 8 characters long and not longer than 64 characters

  channel of AP, if we like to operate using specific channel (optional)

  bssid - mac address of AP (optional)

  connect - a boolean parameter that if set to false, will instruct module just to save the other parameters without actually establishing connection to the access point

- WiFi.localIP()

Station Mode

- WiFi.status()
  Function returns one of the following connection statuses:

  WL_CONNECTED after successful connection is established
  WL_NO_SSID_AVAILin case configured SSID cannot be reached
  WL_CONNECT_FAILED if password is incorrect
  WL_IDLE_STATUS when Wi-Fi is in process of changing between statuses
  WL_DISCONNECTED if module is not configured in station mode

- WiFi.isConnected()

Soft Access Point Mode

- WiFi.softAP(ssid, password, channel, hidden)
- The first parameter of this function is required, remaining three are optional.
- Meaning of parameters

  channel - optional parameter to set Wi-Fi channel, from 1 -13. Default channel is 1.

  hidden - optional parameter, if set to true will hide SSID

- Function will return true or false depending on result of setting the soft-AP

- WiFi.softAPgetStationNum()
- The maximum number of stations that may be connected to ESP8266 soft-AP is five.

Scanning

- WiFi.scanNetworks()

- WiFi.scanNetworks(async, show_hidden)
- Both function parameters are of boolean type.

  asysnc - if set to true then scanning will start in background and function will exit without waiting for result. To check for result use separate function.

  show_hidden - set it to true to include in scan result networks with hidden SSID.

- WiFi.scanComplete()

Client Mode

- Create a client object using
- WiFiClient client;

- Declare server address using
- IPAddress server(74,125,115,105);

- client.connect(server, 80)
- client.stop()

- client.available()
- Returns the number of bytes available for reading

- client.write(data)
- Write data to the server the client is connected to

- client.read()
- Read the next byte received from the server the client is connected to (after the last call to read()).

Server Mode

- WiFiServer server(80);
- Creates a server that listens for incoming connections on the specified port.

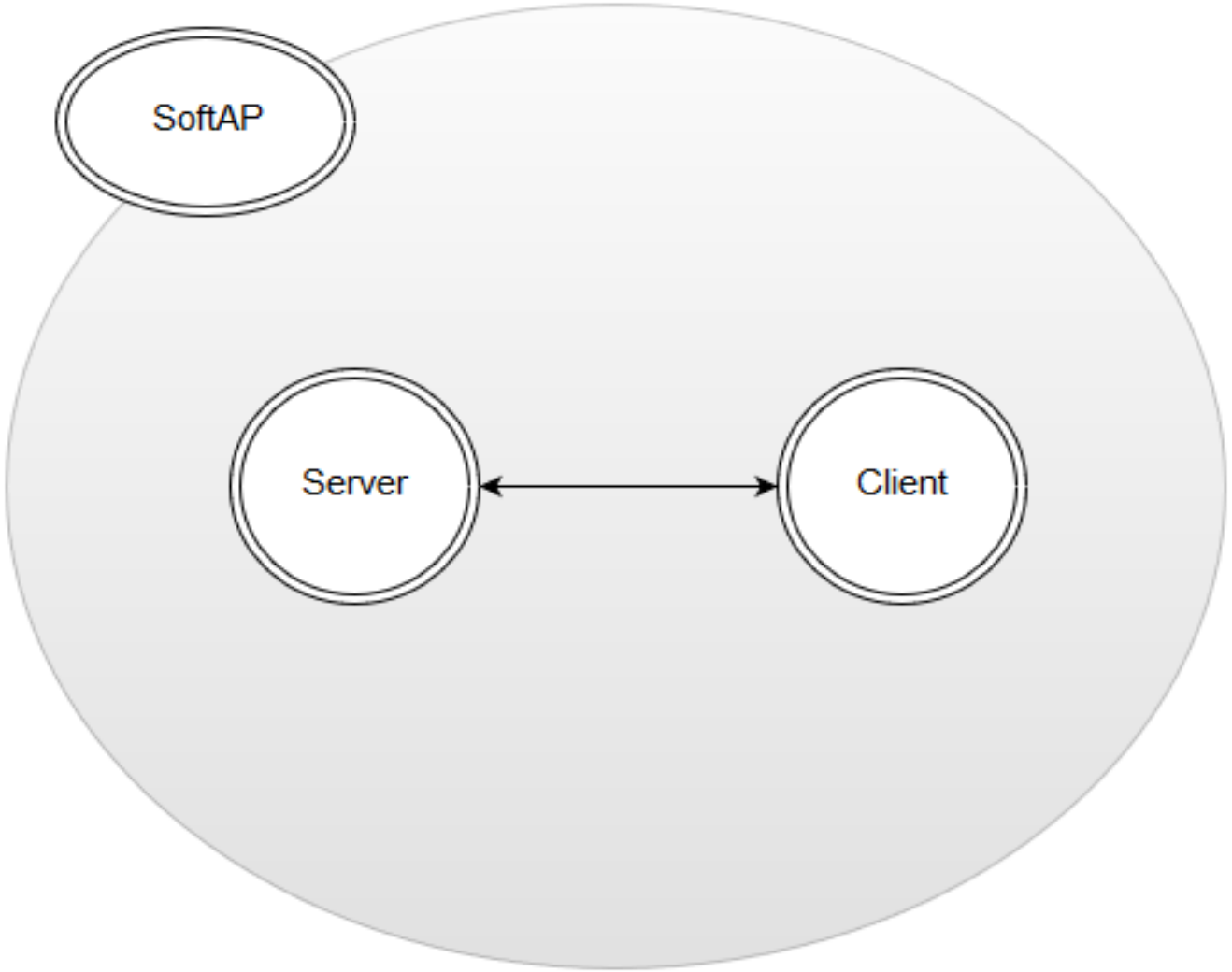
- server.begin()

- server.available()
- Returns a Client object, if no Client has data available for reading, this object will return false.

- server.write(data)
- Write data to all the clients connected to a server (one byte at a time).

- To read data from client use available to get client object and use client.read() to read data.

Basic Example



Writing your first program (sketch)

- The code always has at least 2 functions
  - setup()
  - loop()

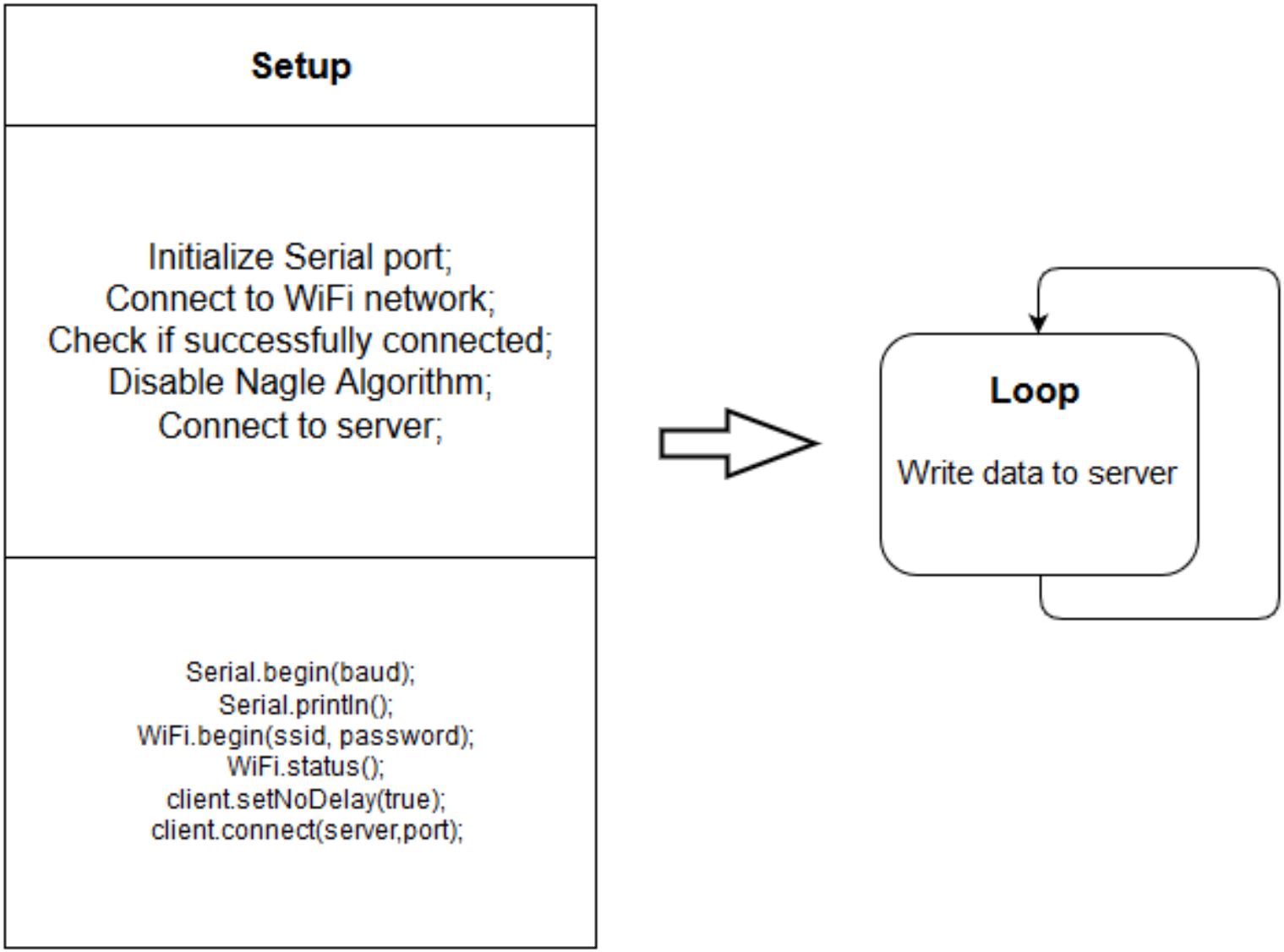
- The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc.

- The setup function will only run once, after each power up or reset.

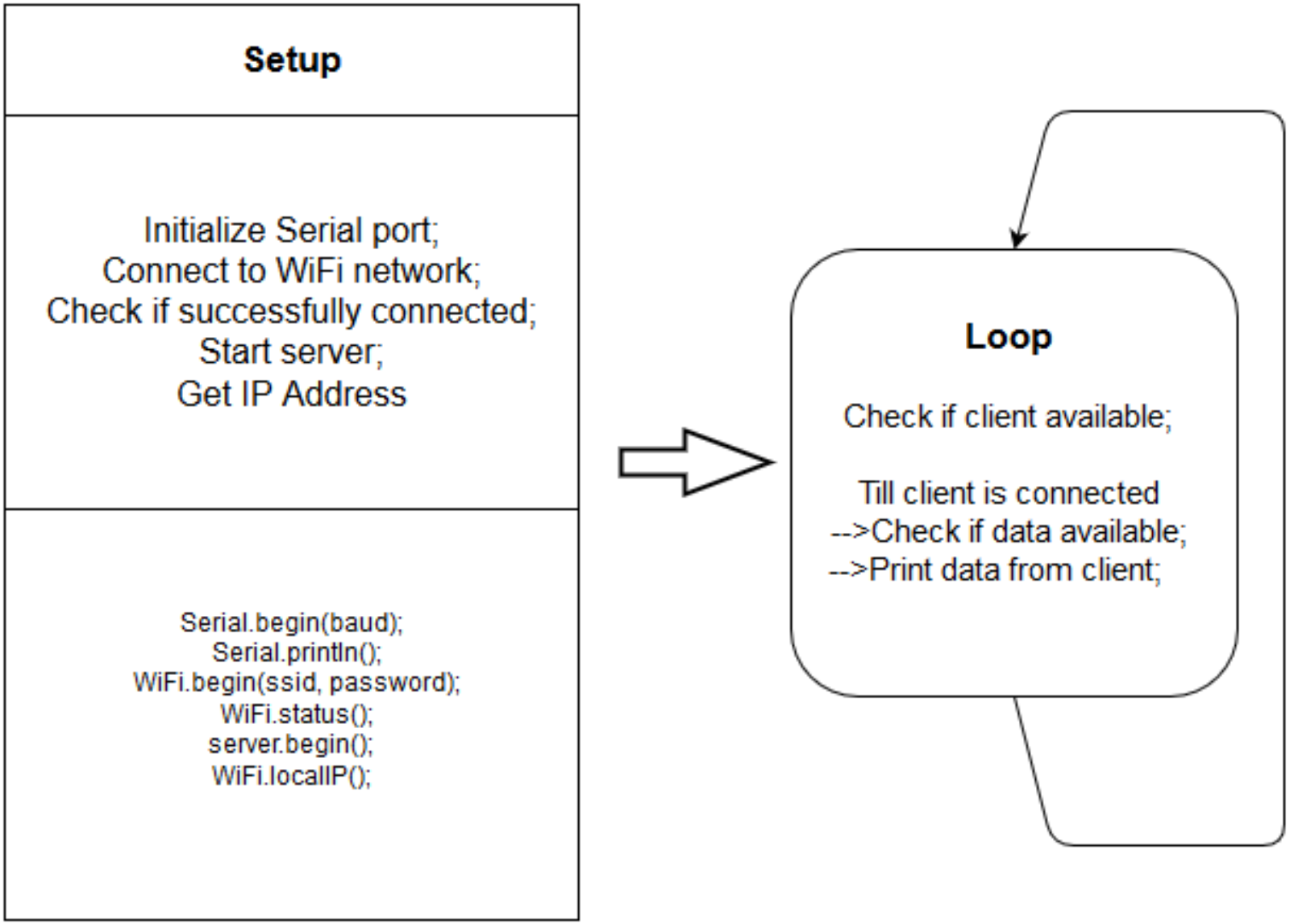
- After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively.

- Its very similar to C/C++.

Client


Setup
Initialize Serial port;
Connect to WiFi network;
Check if successfully connected;
Disable Nagle Algorithm;
Connect to server;
Serial.begin(baud);
Serial.println();
WiFi.begin(ssid, password);
WiFi.status();
client.setNoDelay(true);
client.connect(server,port);
Loop
Write data to server

Server


Setup
Initialize Serial port;
Connect to WiFi network;
Check if successfully connected;
Start server;
Get IP Address
Serial.begin(baud);
Serial.println();
WiFi.begin(ssid, password);
WiFi.status();
server.begin();
WiFi.localIP();
Loop
Check if client available;
Till client is connected
-->Check if data available;
-->Print data from client;

PART2

ESP32

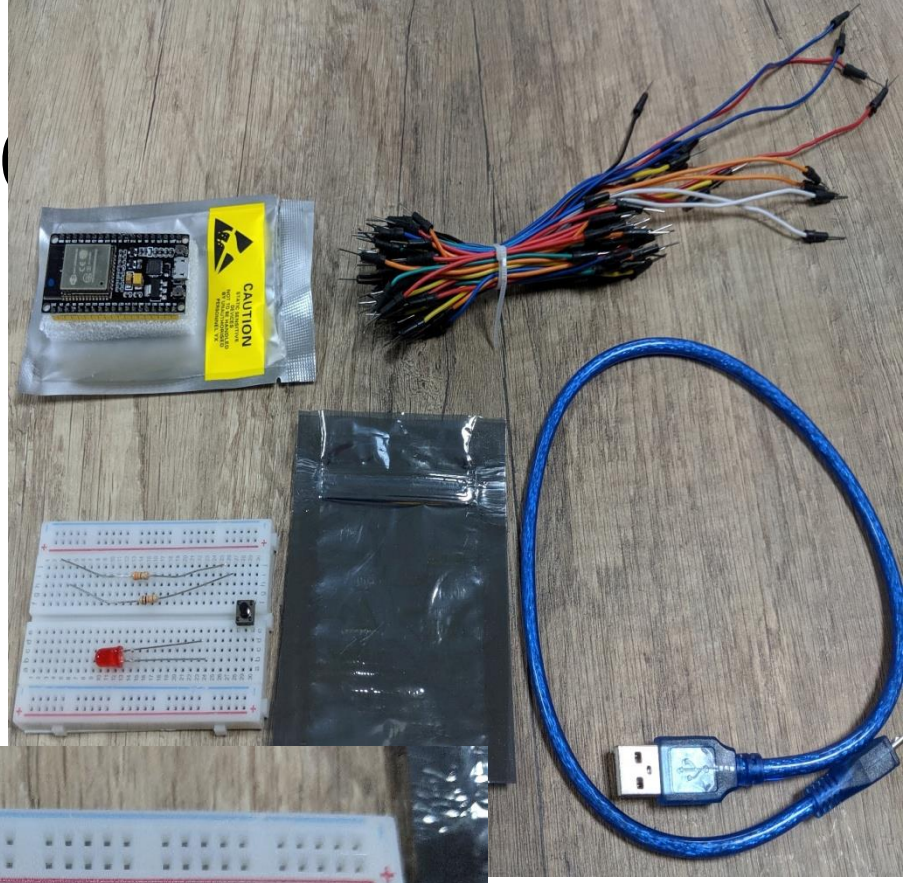
SDK installation

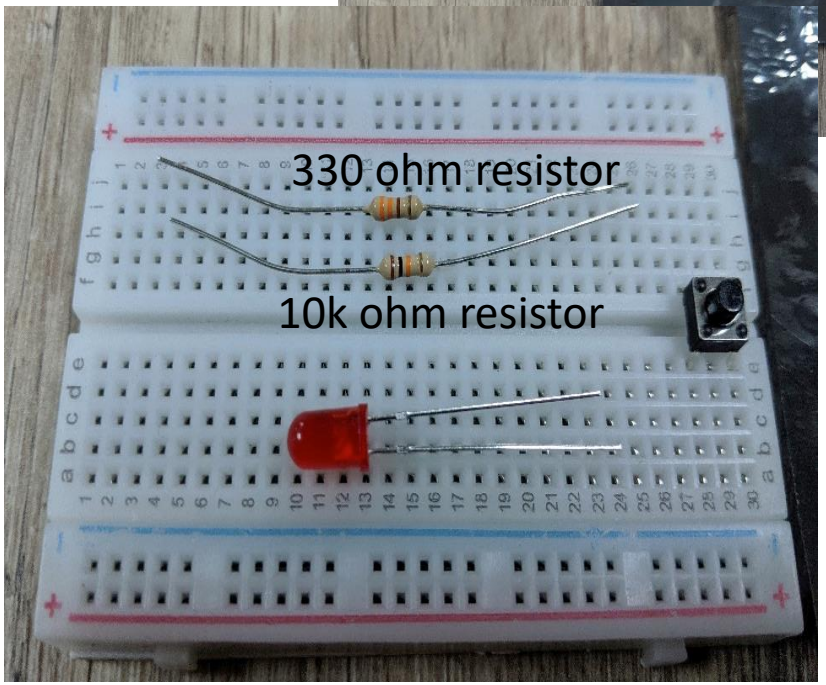
- https://github.com/yeokm1/iot-esp32-mcu-workshop
- Install Serial Virtual Com Port drivers for your operating system if needed
  - https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers
- Install Arduino IDE:
  - https://www.arduino.cc/en/main/software
- Install ESP32 SDK addon to Arduino IDE
  - Add the following path to Additional Boards URL configuration
  - https://dl.espressif.com/dl/package_esp32_index.json

Components

- ESP32 breakout board
- 50cm micro-USB cable
- Half-size breadboard
- Bunch of wires

- In ESD bag
  - Red LED
  - Push button
  - 330 ohm resistor
  - 10k ohm resistor

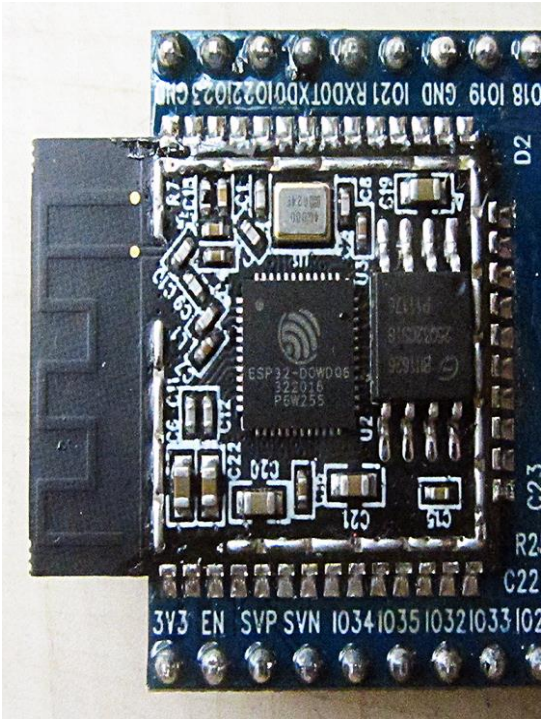


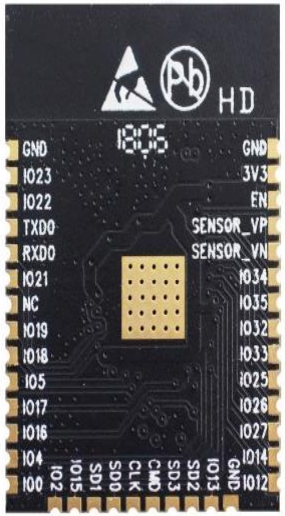


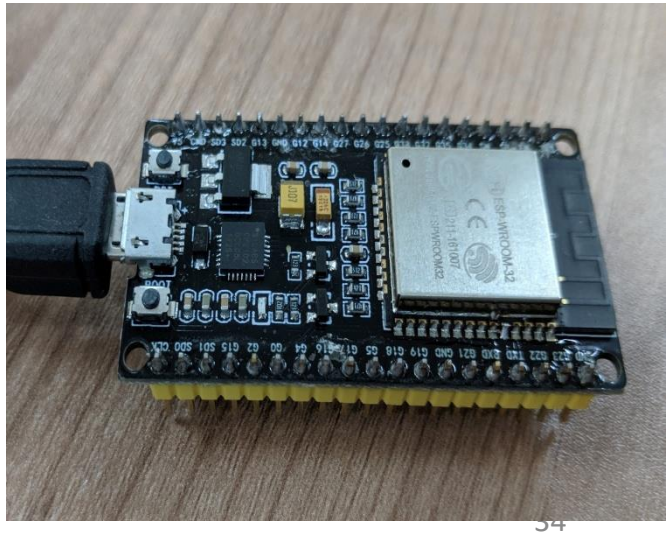
Agenda

- Basics of Wifi
  1. Setup ESP32 as a Station
  2. Setup ESP32 as a AP
- Basics of electronics
  1. Serial - Setting up of software SDK and testing
  2. Blink - Connecting and blinking LED
  3. Button - Connecting button and try to detect press
  4. Debounce - Concept of input debouncing
- The “Internet” portion of IoT
  5. Post - Posting Request to a server on button press
  6. Get - Repeatedly polling the server for on or off instruction

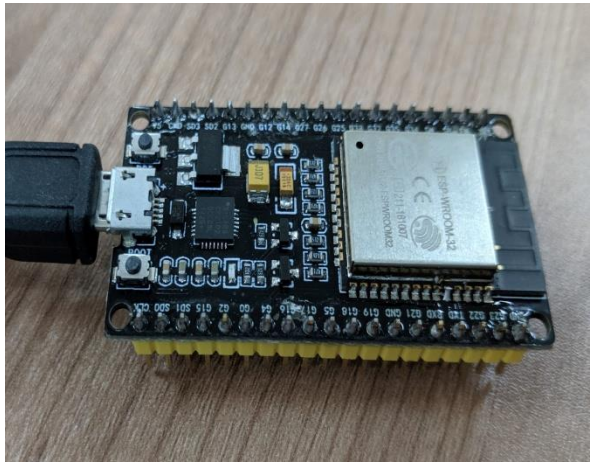
ESP32 chip vs ESP32 module vs ESP32 breakout



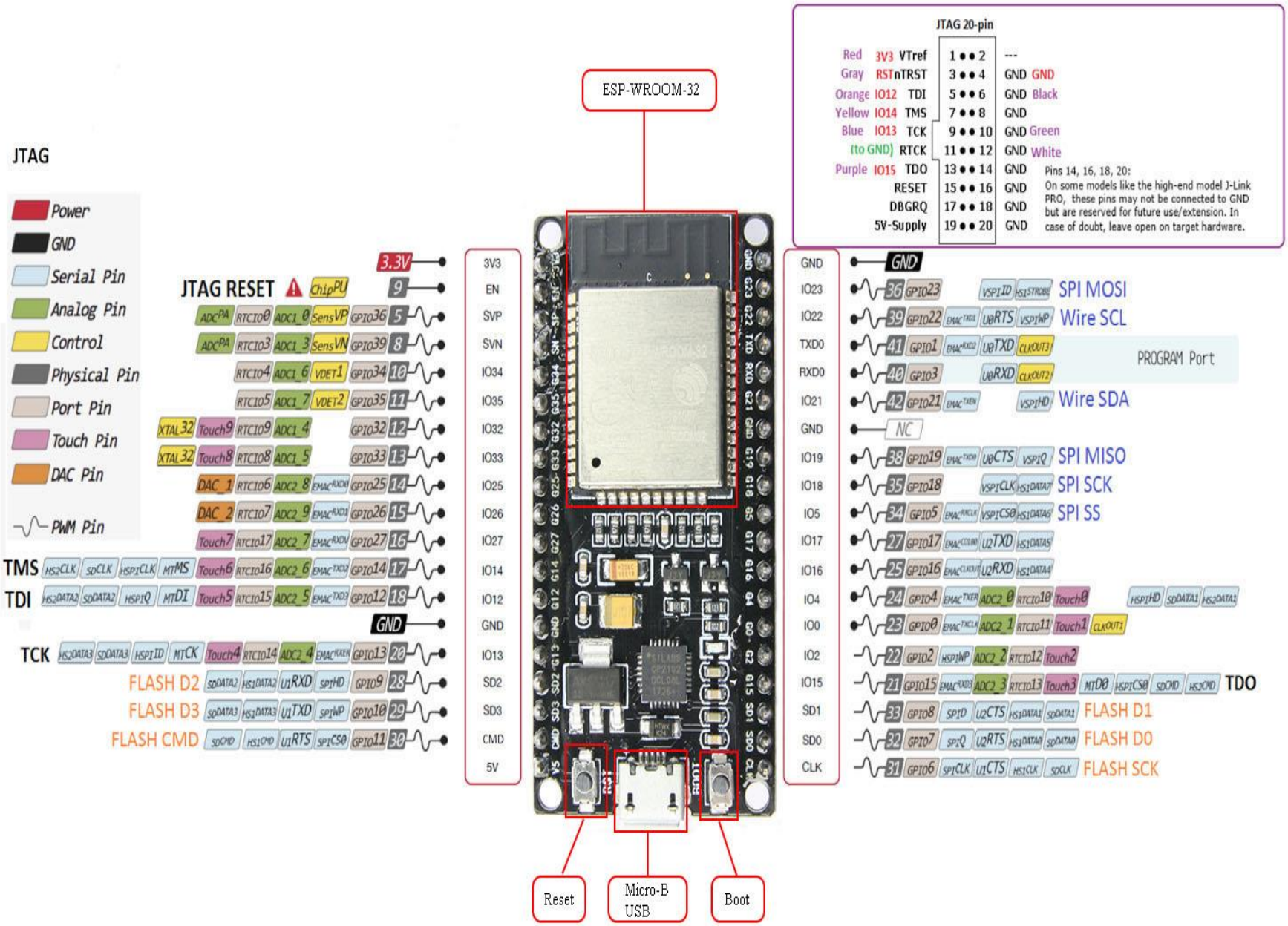




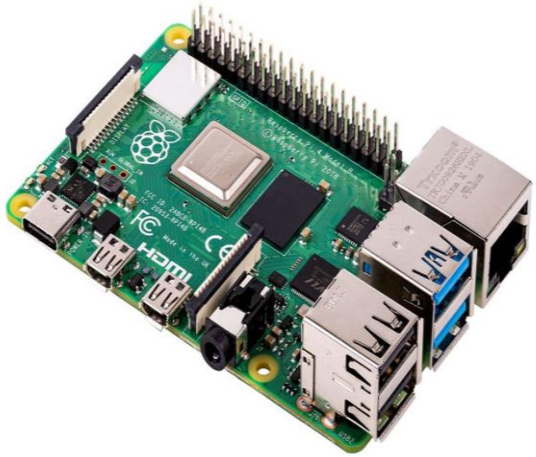
ESP32 Specs

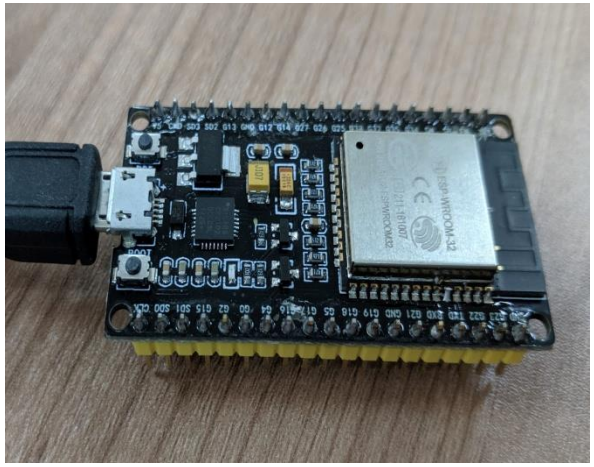


Specs	Value
CPU	Xtensa dual/single-core 32-bit LX6 microcontroller, operating at 160/240 MHz
RAM	520 KiB
Flash memory	4 - 16 MiB
Network Connectivity	Wifi 802.11n 2.4Ghz + BT 4.2
Peripheral Interfaces	3x UART, 2x SPI, 2x I2C, ADC, SD touch sensors, etc


JTAG
Power
GND
Serial Pin
Analog Pin
Control
Physical Pin
Port Pin
Touch Pin
DAC Pin
PWM Pin
JTAG 20-pin
Red 3V3 VTref 1 2 ---
Gray RST nTRST 3 4 GND GND
Orange IO12 TDI 5 6 GND Black
Yellow IO14 TMS 7 8 GND
Blue IO13 TCK 9 10 GND Green
(to GND) RTCK 11 12 GND White
Purple IO15 TDO 13 14 GND
RESET 15 16 GND
DBGRQ 17 18 GND
5V-Supply 19 20 GND
Pins 14, 16, 18, 20:
On some models like the high-end model J-Link PRO, these pins may not be connected to GND but are reserved for future use/extension. In case of doubt, leave open on target hardware.
ESP-WROOM-32
JTAG RESET
TMS
TDI
TCK
FLASH D2
FLASH D3
FLASH CMD
SPI MOSI
Wire SCL
PROGRAM Port
Wire SDA
SPI MISO
SPI SCK
SPI SS
TDO
FLASH D1
FLASH D0
FLASH SCK
Reset
Micro-B USB
Boot

Why not single-board computers like

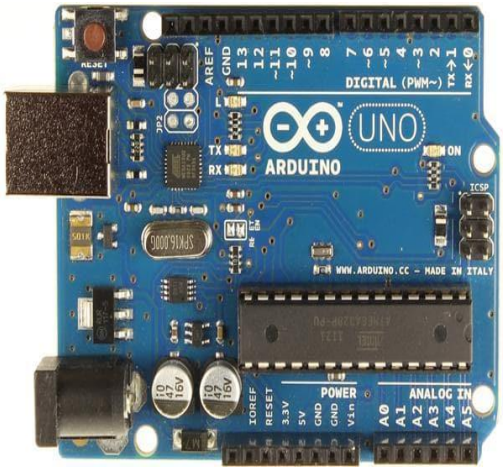


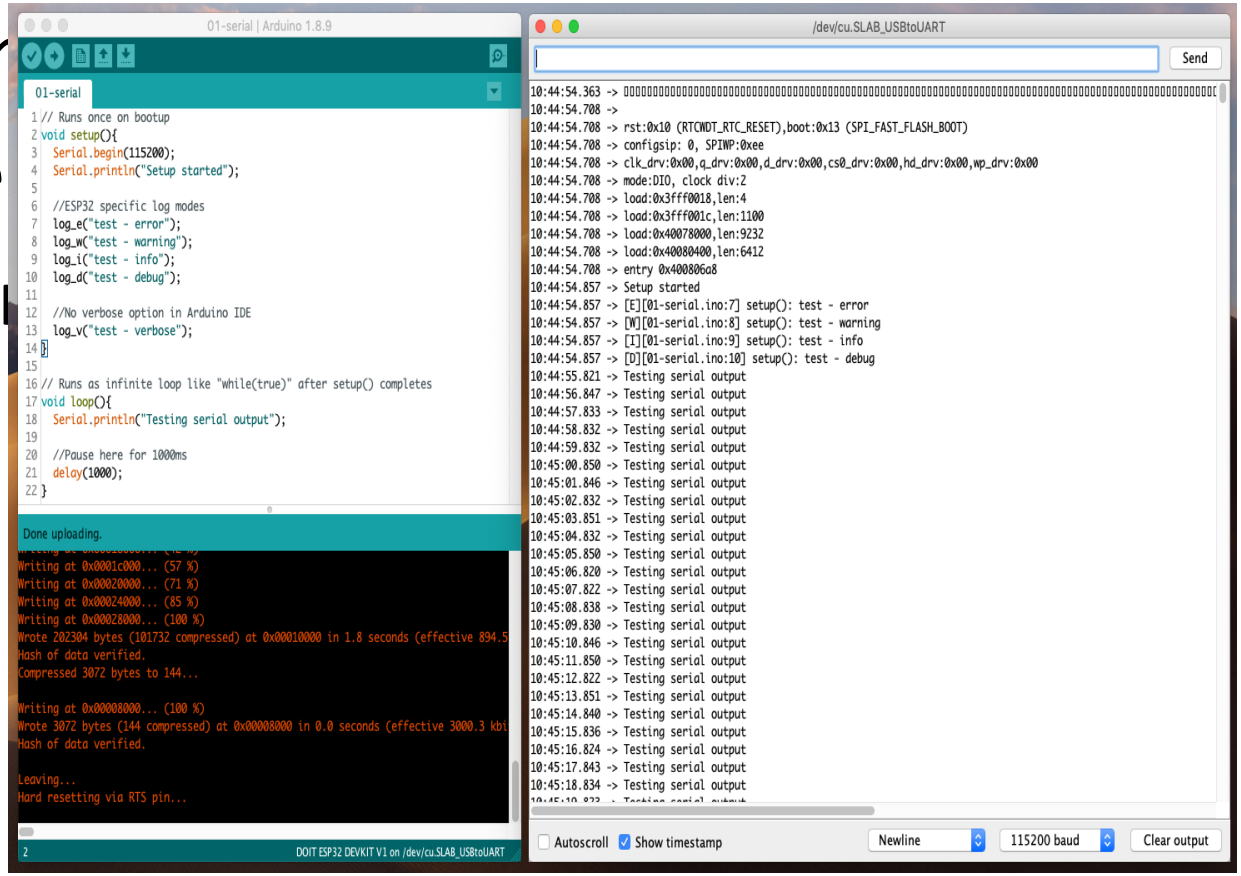


Specs	Arduino-compatible ESP32	Raspberry Pi 4 B
CPU type	Microcontroller	Microprocessor
Speed	Dual Core 160/240Mhz	Quad core 1.5 Ghz
RAM	520KB	1 - 4 GB
GPU/Display	None	VideoCore VI GPU, 4K
Disk	4 MiB	Depends on microSD card
Other connectivity	Wifi 802.11n 2.4Ghz + BT 4.2	USB, Ethernet, HDMI, audio, Wifi 2.4/5Ghz 802.11ac, BT 5.0
Operating System	None	Linux (usually Raspbian)
Real Time Operation	Better	Poorer
Typical Power	0.8W	8W

Arduino SDK

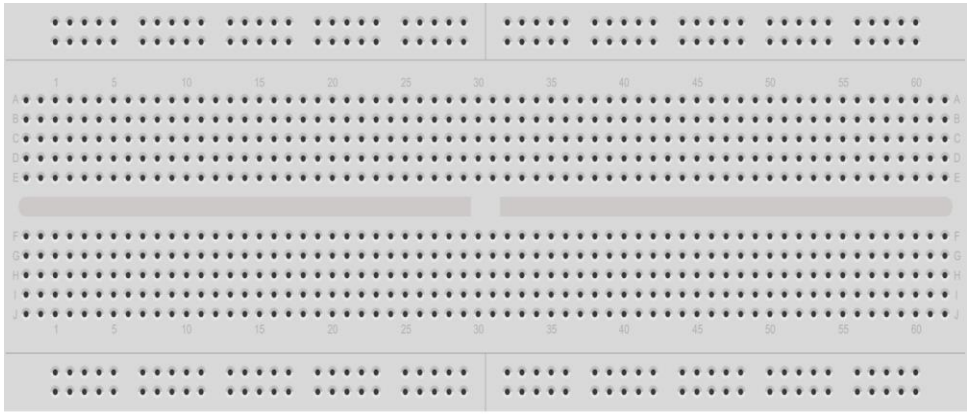
- Easy to use cross-platform IDE for microcontroller programming
- Support for mar
- C/C++ language
- Numerous libra

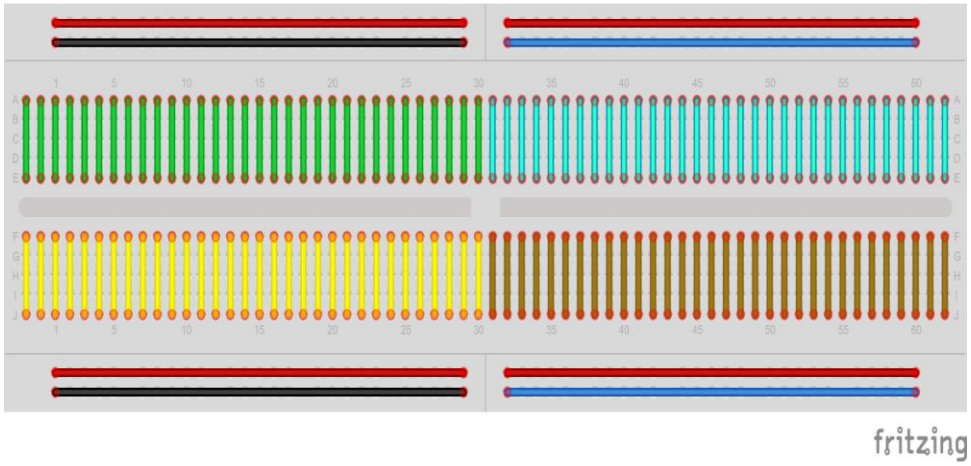




Half/Full Breadboard

- Base Prototyping component
- Continuous lines indicate those holes are

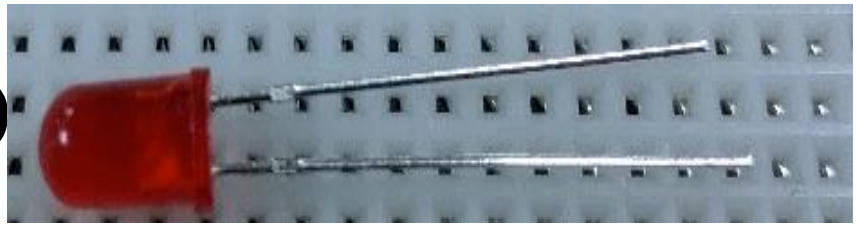




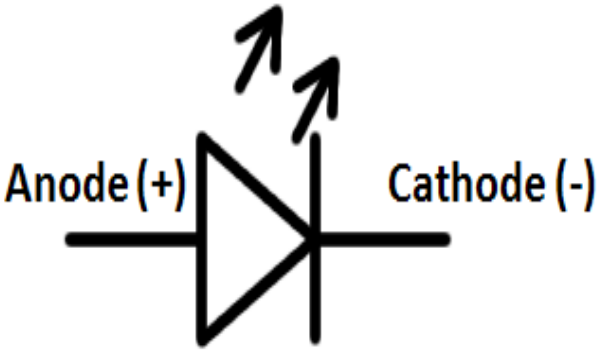
- Horizontal usually for power
- Convention: Red for +, Black/Blue for -

- Vertical for your

LED

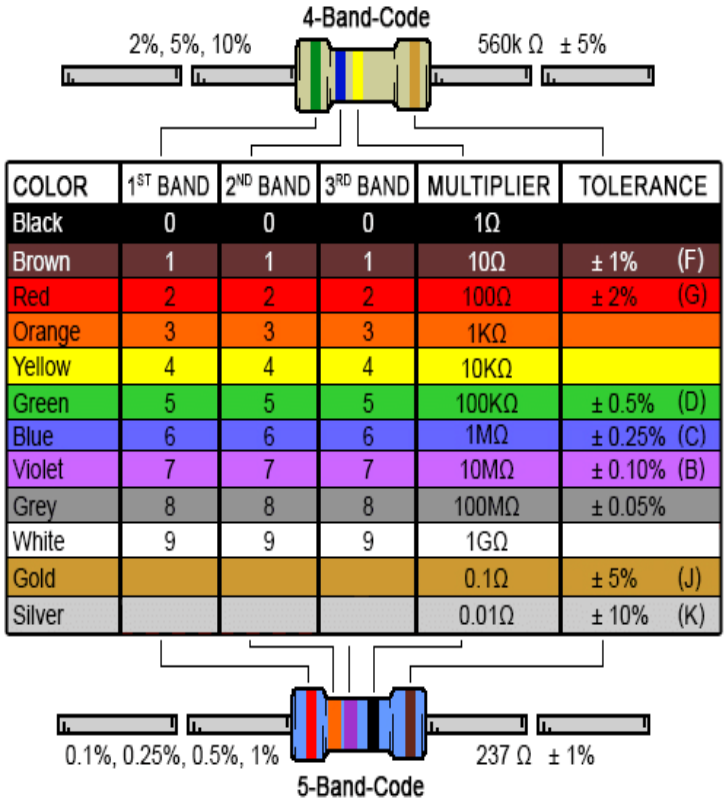


- "A **light-emitting diode** (**LED**) is a semiconductor light source that emits light when current flows through it. "
- Diode: Component allowing one way current flow



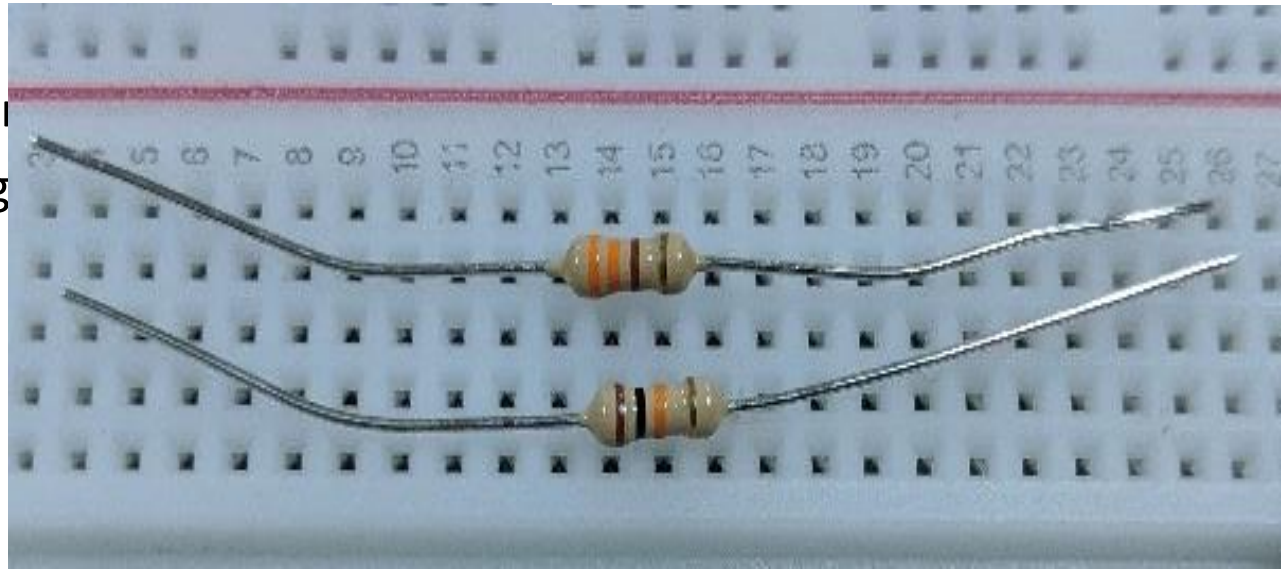
- Anode +: Longer leg
- Cathode -: Shorter Leg

Resistor



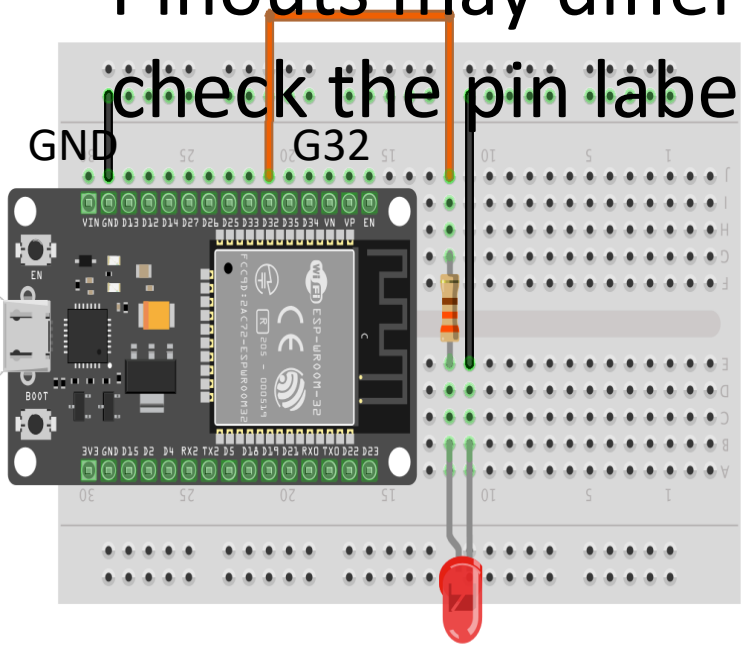
COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

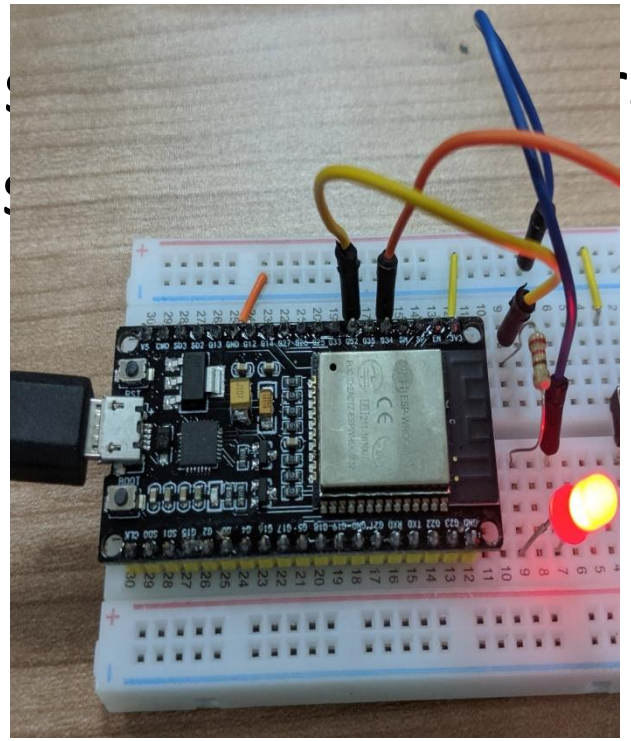
- Resists the flow of current
- Resistor Colour bands
- 4-band 330 ohm
  - Orange, Orange, Brown, Gold
  - 3, 3, x10, 5% tolerance
- 4-band 10K ohm
  - 1, 0, x1000, 5% toler
  - Blown, Black, Orang

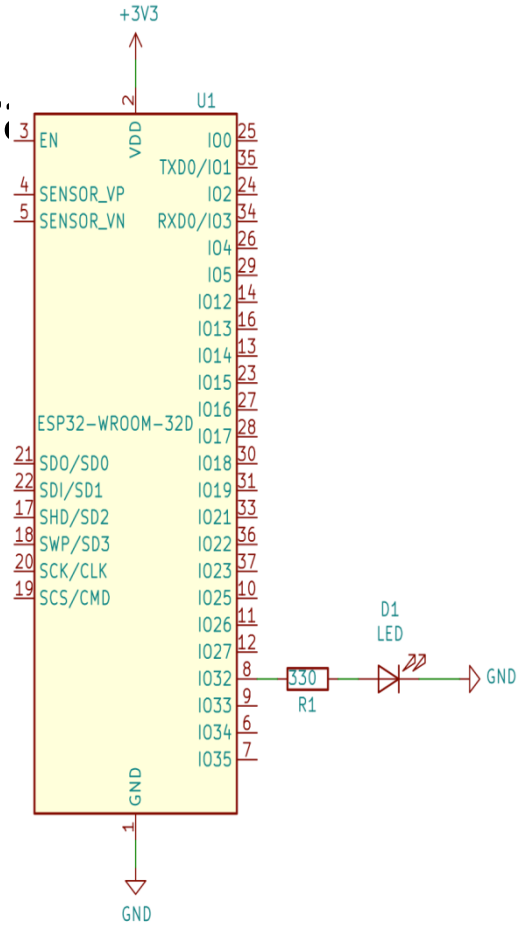


Connecting the LED setup

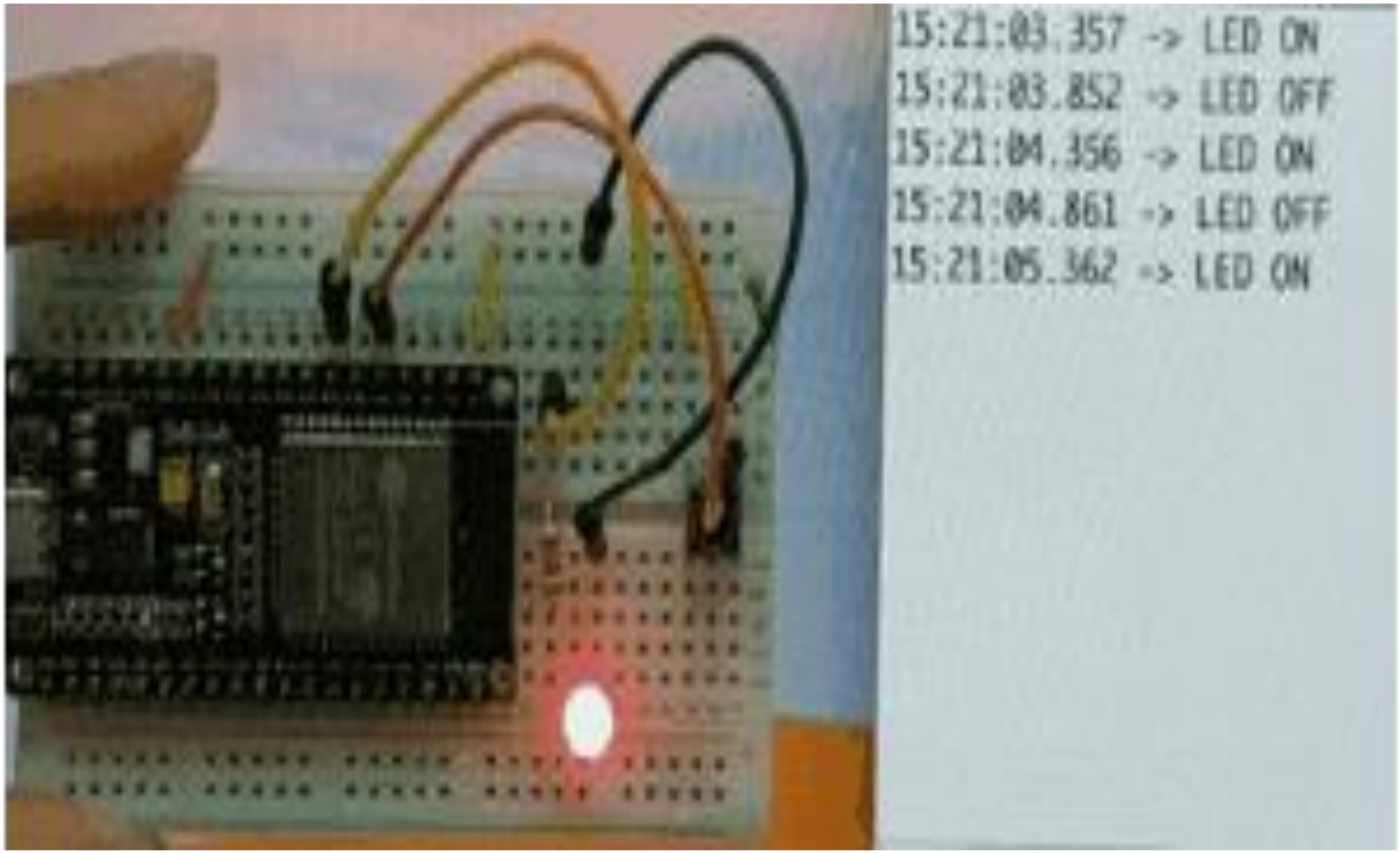
- Unplug USB cable before working on electronics
- Breadboard + ESP32 board (G32, GND) + 330 ohm resistor + LED + wires
- Pinouts may differ s a
  check the pin labels





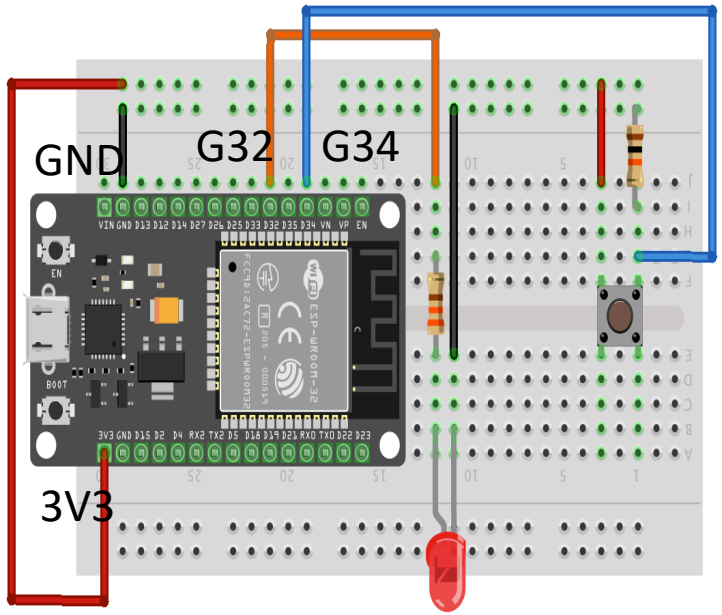


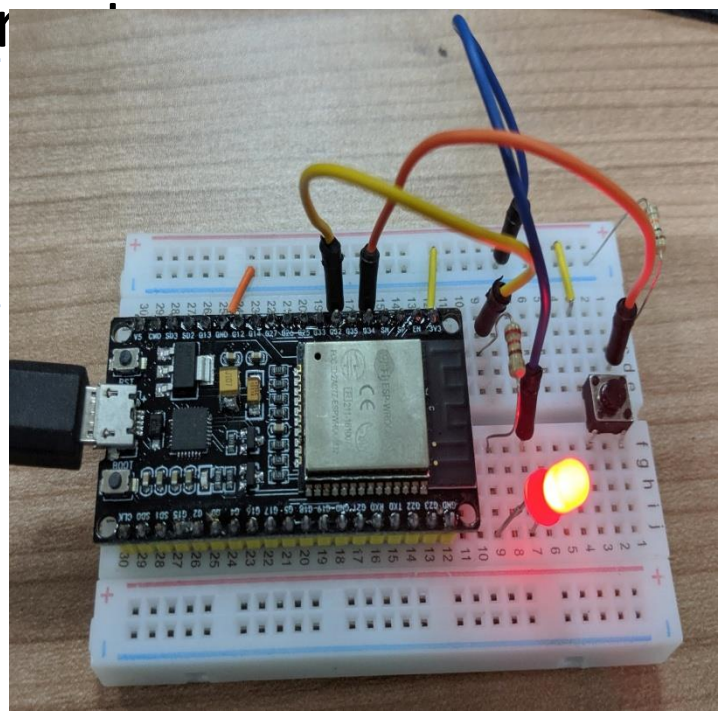
Program the board with 02-blinky.ino

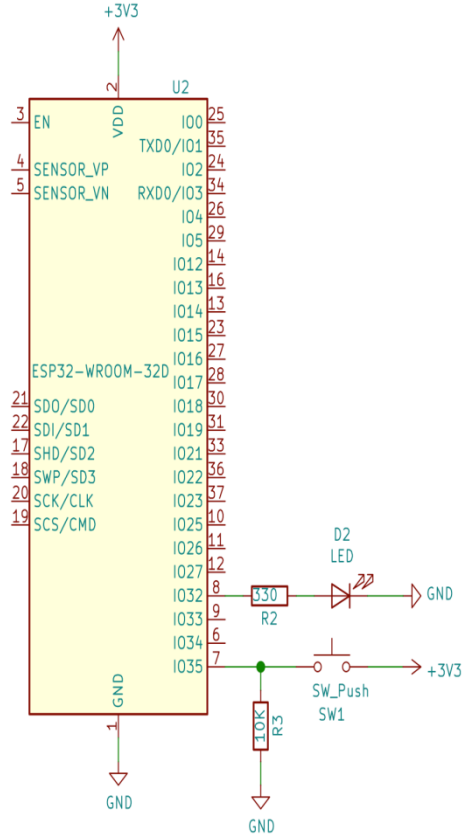


03-button button setup

- 10k ohm resistor + button + wires -> G34
- Actual breakout 3V3 pin position **differs from picture**
- “Pull down” resistor

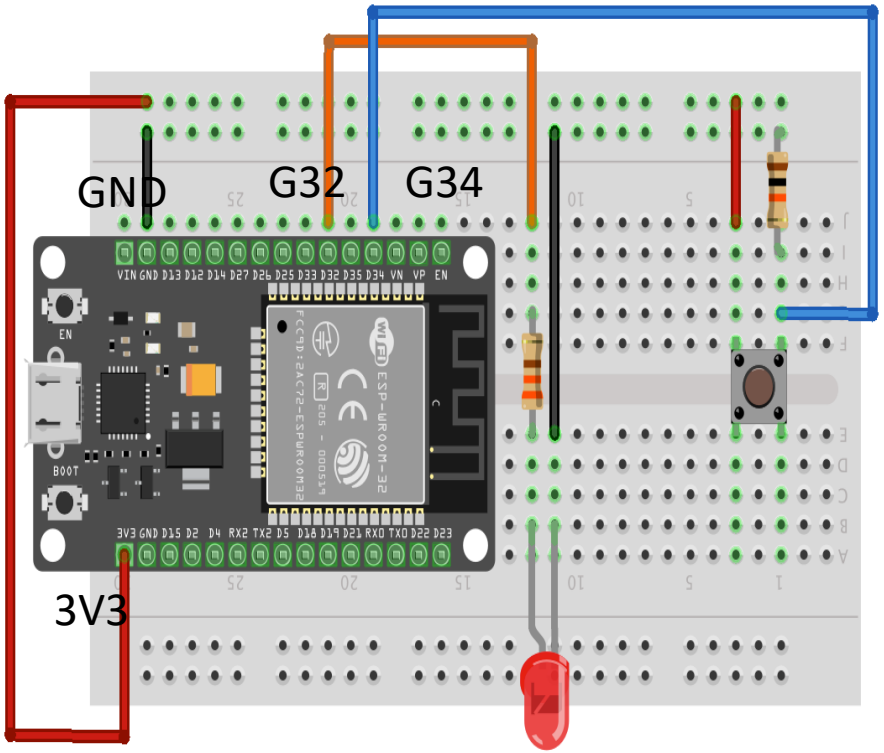


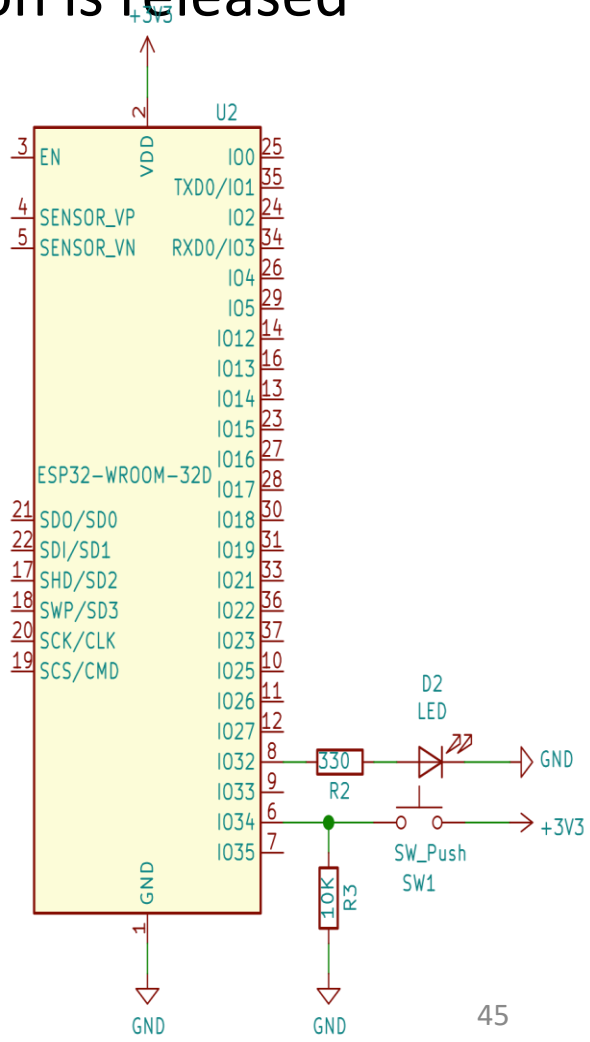




Pull down resistor for button

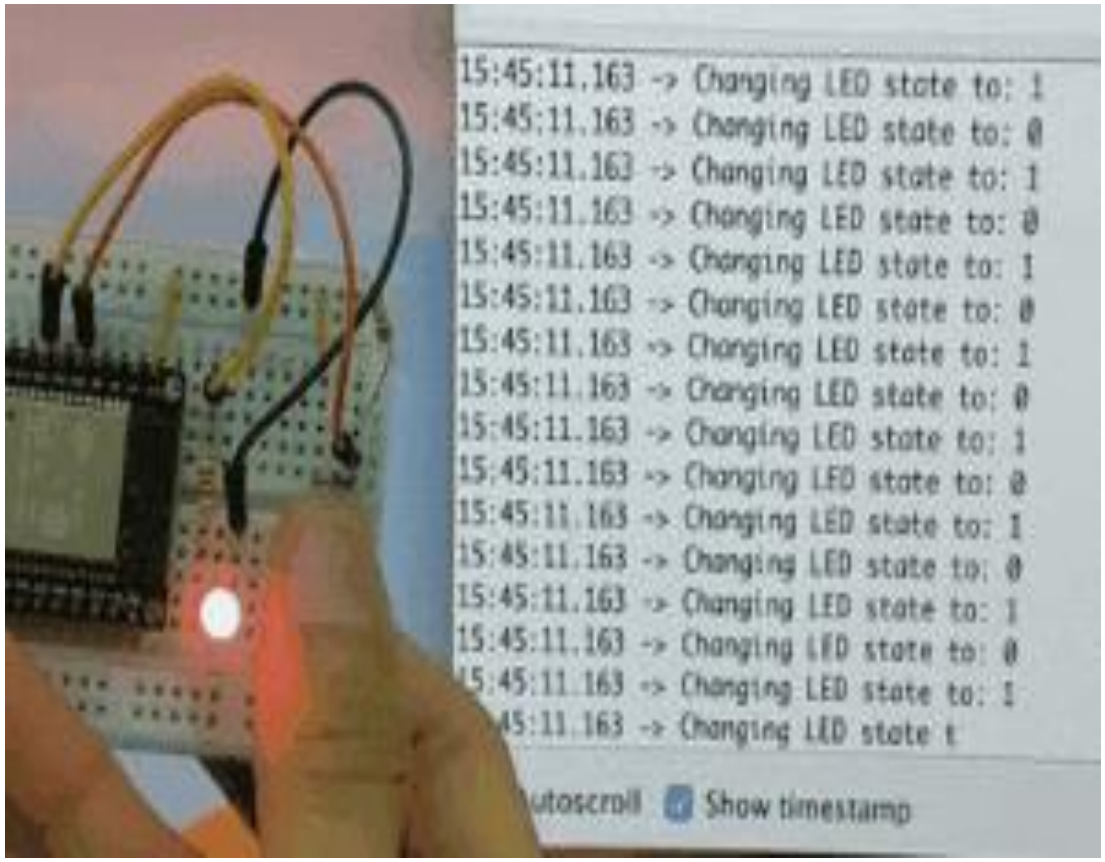
- Resistor “pulls” IO34 to ground when button is released
- Guarantees that IO34 pin goes to 0





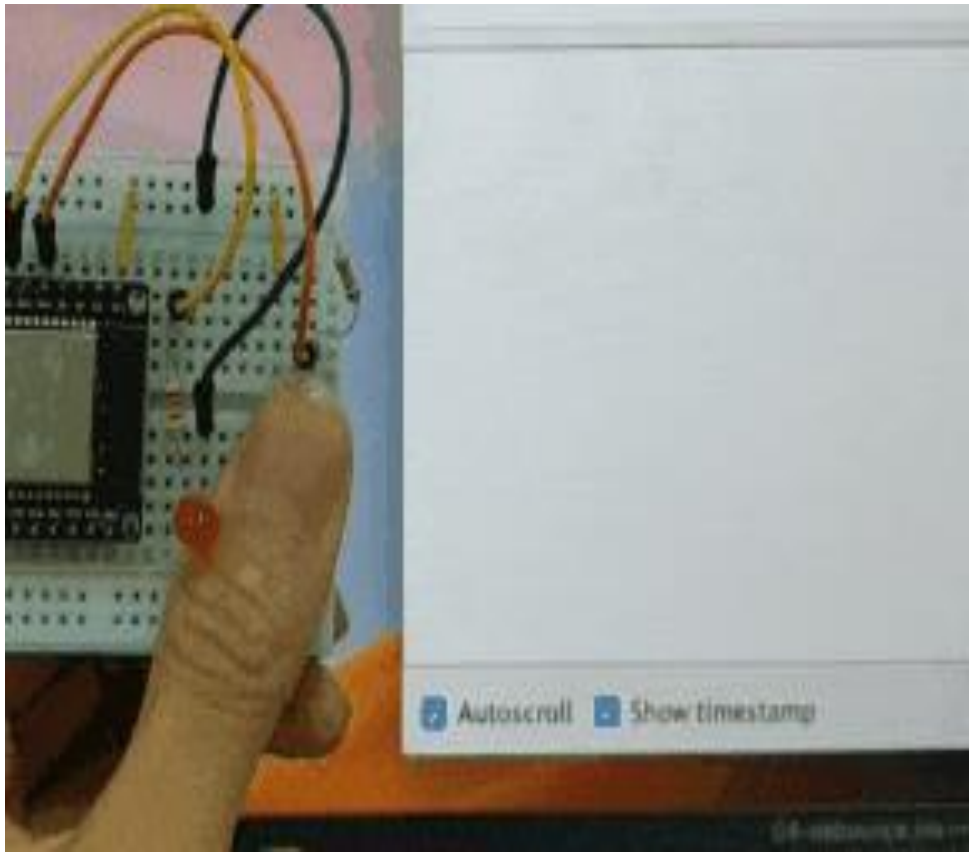
Program the board with 03-button.ino

- We want to toggle LED on each button press
- Observe looping speed of button press detection is too fast to be useful



04-debounce

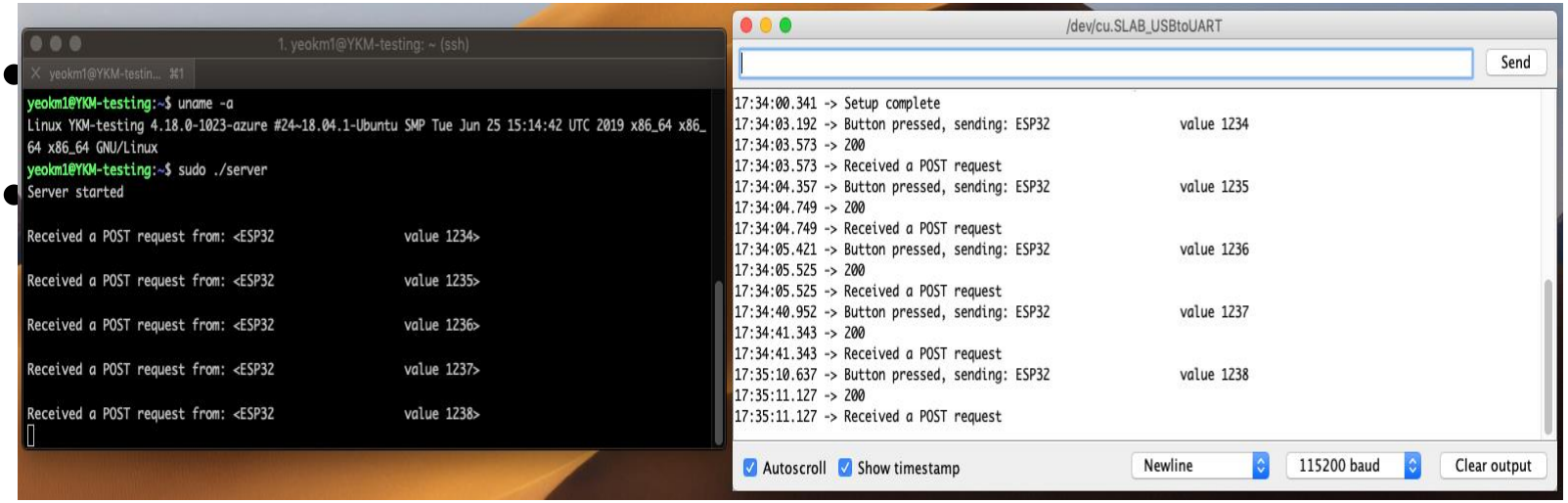
- Process one input within an interval



05-wifi-post

1. Making a POST request to the server via a button press
2. Able to receive a POST request using a Go program

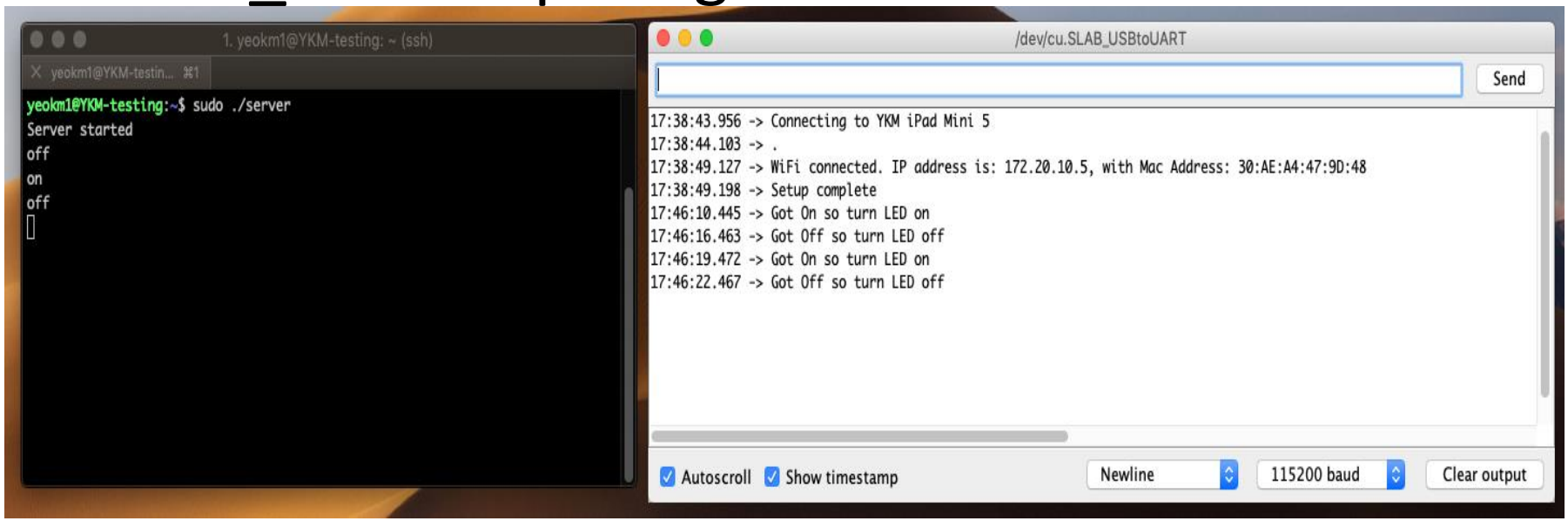
- WIFI_SSID: iot-workshop



06-wifi-get

1. Making a Get Request to a server to get instruction
2. Serve a GET request using a Go program

- WIFI_SSID: iot-workshop
- WIFI_PASS: esp32isgreat



Useful websites

- http://frightanic.com/iot/comparison-of-esp8266-nodemcu-development-boards/
- https://www.gitbook.com/book/krzychb/esp8266wifi-library/details
- https://github.com/esp8266/Arduino/blob/master/doc/reference.md
- http://www.esp8266.com/wiki/doku.php
- https://github.com/esp8266/Arduino/blob/master/doc/libraries.md
- http://esp8266.github.io/Arduino/versions/2.0.0/doc/libraries.html

Thank You