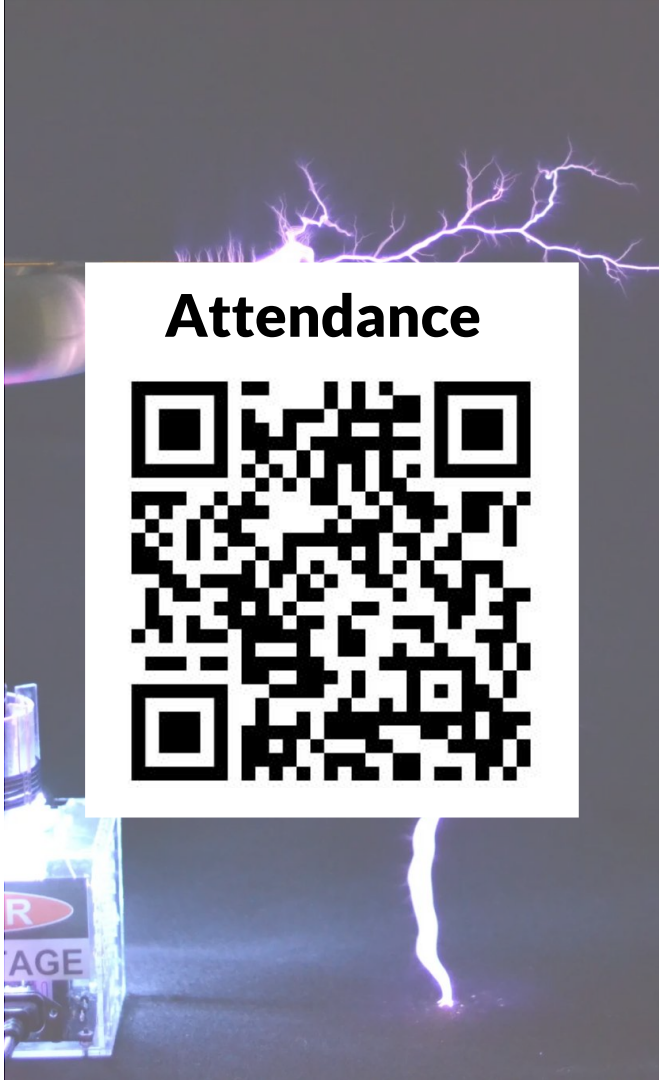
Arduino Workshop

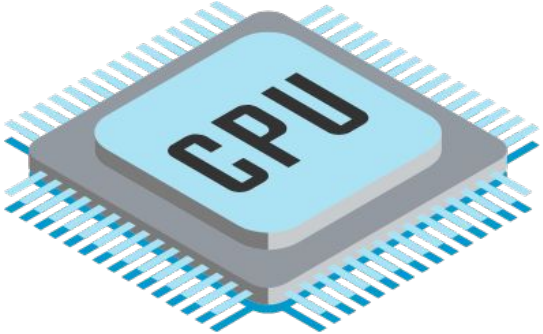
IEEE WPI Student Branch



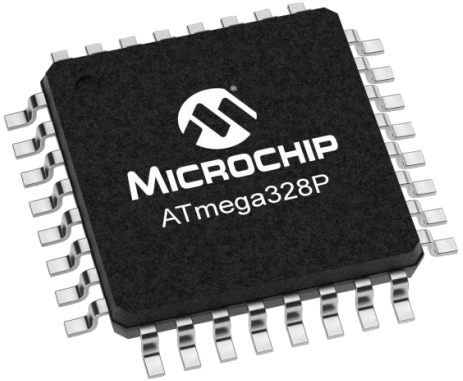
Background

CPUs

- General-purpose processor
- Requires external memory, peripherals
- Multi-threaded, multi process
- Runs an OS (Windows, Linux, macOS, etc)
- High speed, high power
- Used in computer, servers, phones



Microcontrollers



- Specialized, single-chip computer
- **Real-time** processing (typically no OS)
- Single-threaded execution
- Designed for **embedded systems**
- Processes data from **inputs** (sensors, buttons, etc.) and controls **outputs** (lights, motors, screens, etc.)
- Low power, small, but dedicated, reliable functionality

Use Cases

The Problem with Microcontrollers

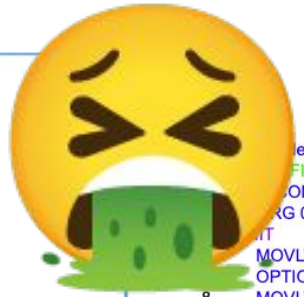
- Complex setup
- Low-level programming (C/Assembly)🤮
- Complicated, proprietary toolchains
- Limited hardware libraries
- **Steep learning curve**📈

ESP-IDF

STM32 CubeIDE

The Problem with Microcontrollers

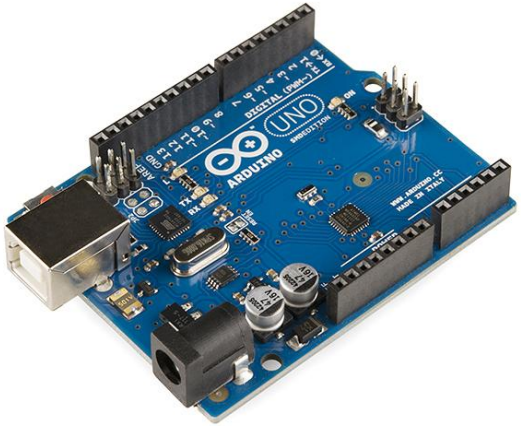
```
ISR(ADC_vect)
{
  TIFR1 = (1<<OCF1B);     //clear Timer compare match flag
  if (adcon ==1)
  {
    array[i] = ADCW;      //ADCW takes care of ADCL and ADCH
    i++;
  }
}
void check()
{
  i = 0;
  while(i<160)
  {
    SMCR |=0x02;          //SMCR - Sleep Mode Control Register
                          //bit 3:1, 001, ADC Noise Reduction Mode
                          //ADC noise reduction mode
    adcon = 1;
  }
  adcon = 0;
  SMCR &= 0xFD;           //idle mode
}
```

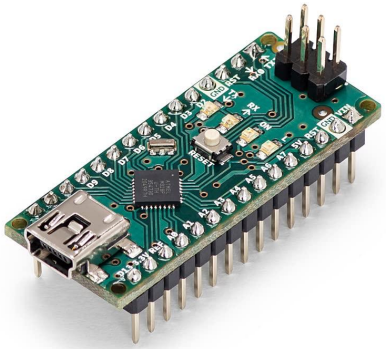


```
    e "p10f200.inc"
    FIG
    ONFIG _WDT_OFF & _CP_OFF & _MCLRE_OFF
    RG 0x0000
    IT
    MOVLW  ~(1<<T0CS)      ;Enable GPIO2
    OPTION
8   MOVLW ~(1 << GP2)      ;Set and GP2 as an output
9   TRIS GPIO
10  LOOP
11    BSF GPIO, GP2        ;Set GP2
12    CALL DELAY           ;Call DELAY subroutine
13    BCF GPIO, GP2        ;Reset GP2
14    CALL DELAY           ;Call DELAY subroutine
15    GOTO LOOP            ;loop forever
16
17  DELAY                  ;Start DELAY subroutine here
18    MOVLW D'162'         ;Load initial value for the delay
19    MOVWF 10             ;Copy the value to the register 0x10
20    MOVWF 11             ;Copy the value to the register 0x11
21  DELAY_LOOP             ;Start delay loop
22    DECFSZ 10, F         ;Decrement the register 0x10 and check if not zero
23    GOTO DELAY_LOOP      ;If not then go to the DELAY_LOOP label
24    DECFSZ 11, F         ;Else decrement the register 0x11, check if it is not 0
25    GOTO DELAY_LOOP      ;If not then go to the DELAY_LOOP label
26    RETLW 0              ;Else return from the subroutine
27
28    END
```

About Arduino

- **Open-source** hardware + software ❤️
- Unified programming + connectivity
- Easily program off-the-shelf microcontrollers ✅
- Large ecosystem of hardware libraries
- Community driven, cross-platform

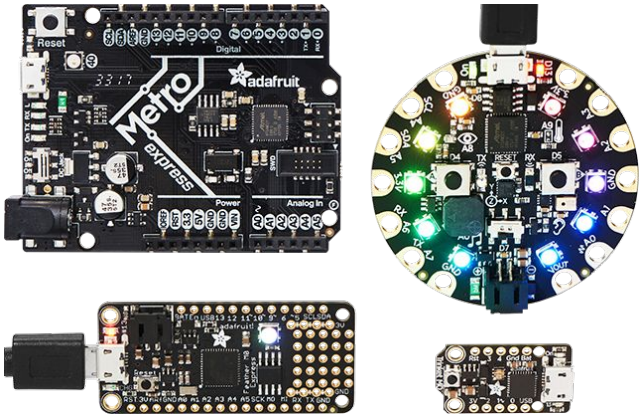


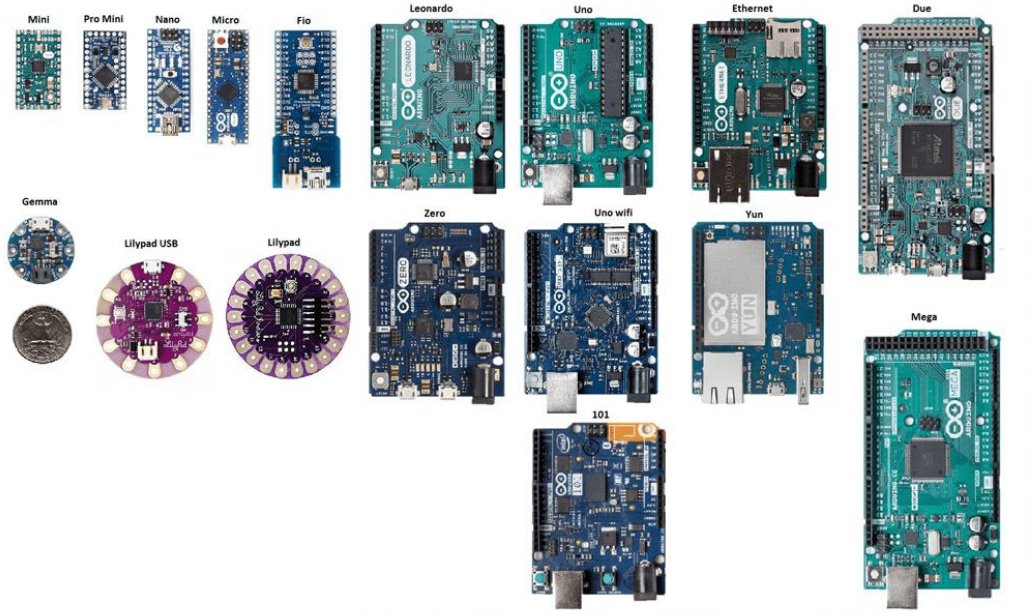


Why Arduino

- Plug and play 🔌
- High-level programming ☀️
- Unified tool chain
- Cross-compatible libraries
- Beginner friendly!

Variants





Other variants (Adafruit, ESP) can be programmed via Arduino IDE

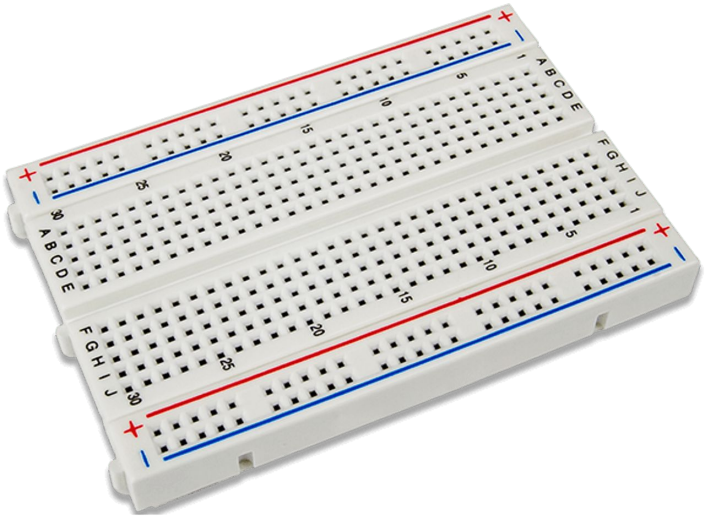
Arduino IDE

- Single IDE for all Arduinos
- Simple & beginner-friendly
- One click upload, debugging
- Large hardware library ecosystem
- High-level, uses C++

Beginner Track

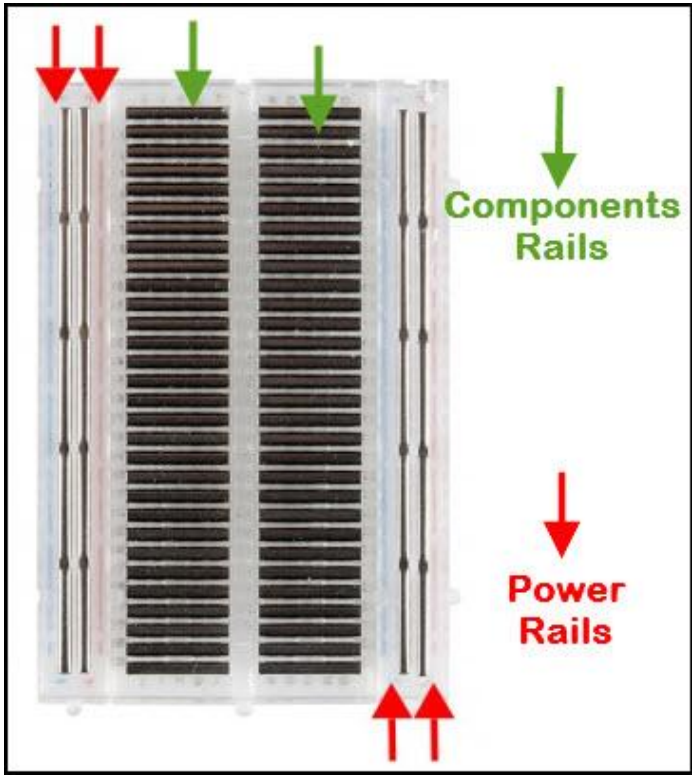
Breadboards

- No soldering required 🔧
- Easy to modify and test circuits ⚡
- Reusable for multiple projects 🔄
- Works with various components
- Essential for prototyping and learning!



How do Breadboards work?

- The component rails are Horizontally connected
- The power rails are vertically connected (Used for PWR and GND)
- Easily move components and wires
- Great for experimenting with circuits

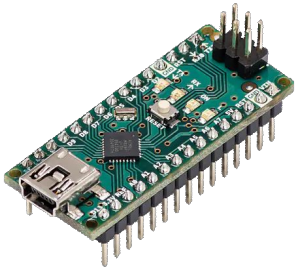


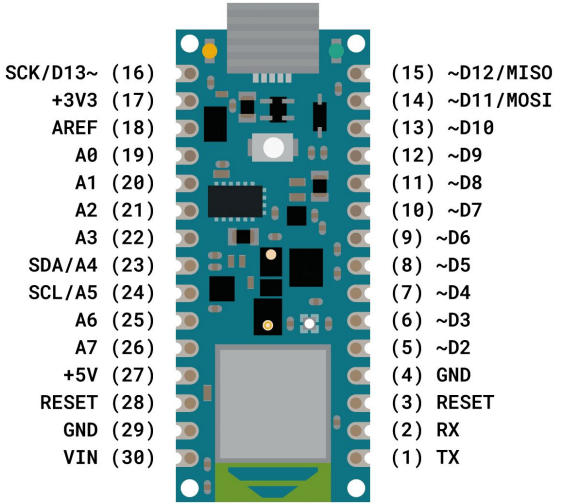
Circuit basics: Arduino

- In order to use your Arduino, place it **into** the breadboard.

  PLEASE DO NOT BEND THE PINS

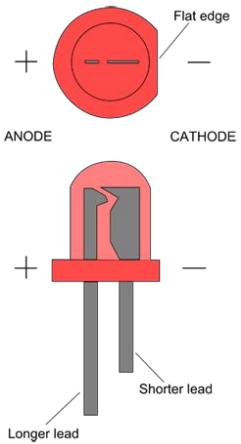
- The Arduino has several Pins
  - We'll be using Digital (D), GND, and 5V
- Reference the Pinout schematic when designing your circuits!

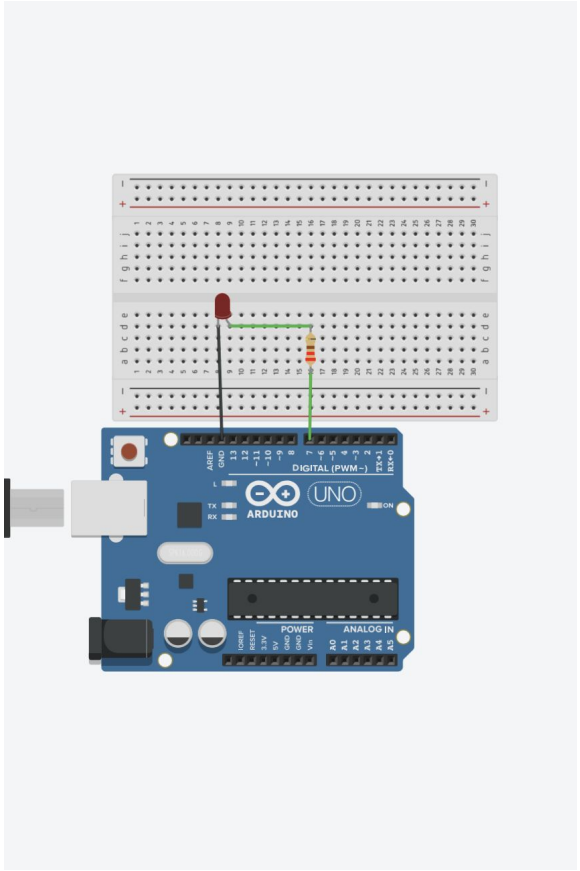




Circuit basics: LED Circuit

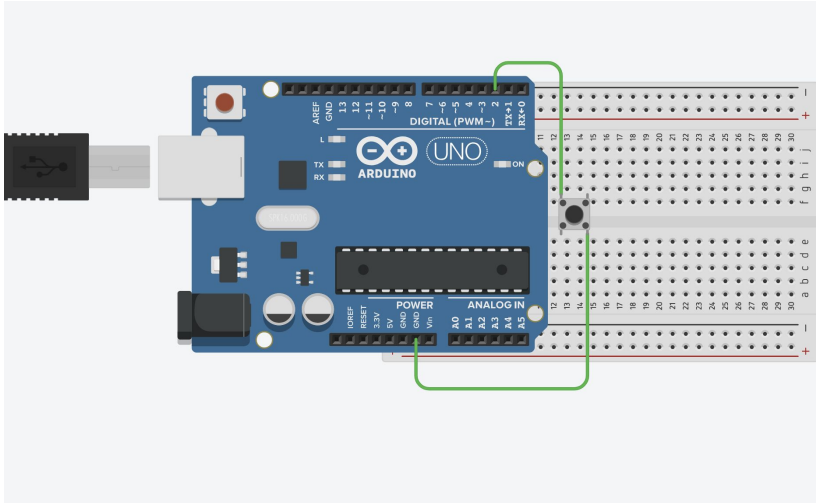
- An **LED has two leads**
  - An Anode + (Longer Side)
  - A Cathode - (Shorter side)
- The Cathode goes to **ground** while the Anode goes to **power**
- Instead of **connecting it to direct power**, we'll connect the Anode to one of our **Digital Pins**





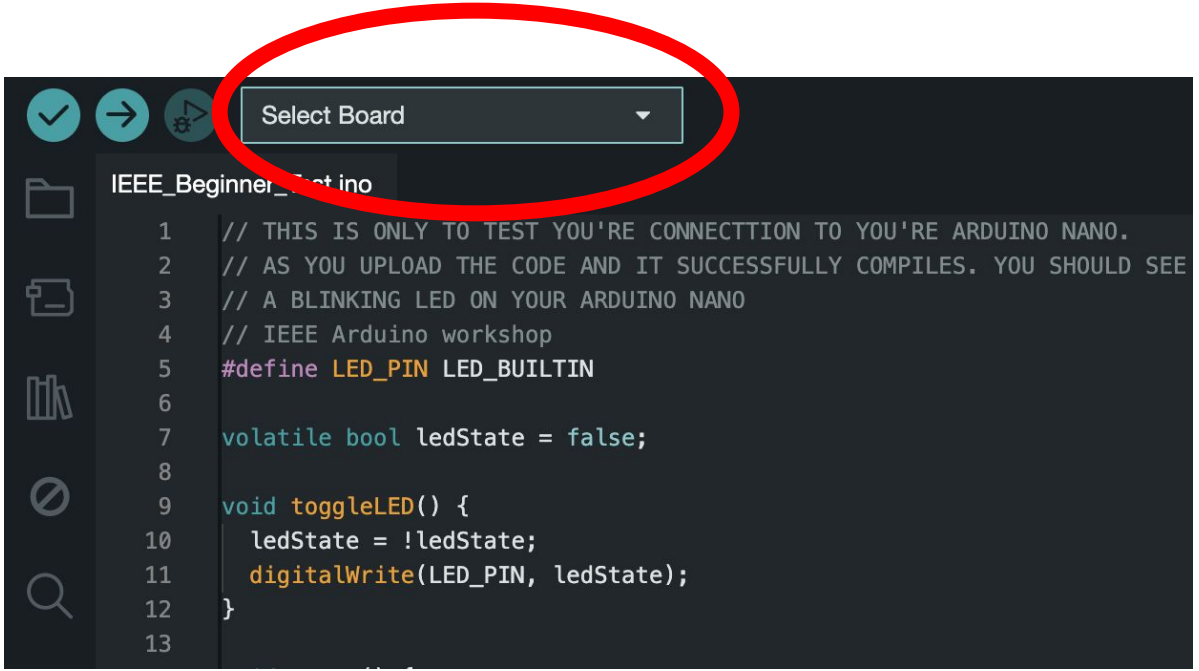
Circuit basics: Button

- One lead is connected to GND
- The other lead is connected to a **digital input pin**
- **Additionally**, we need to set a pull-up resistor to set a default state



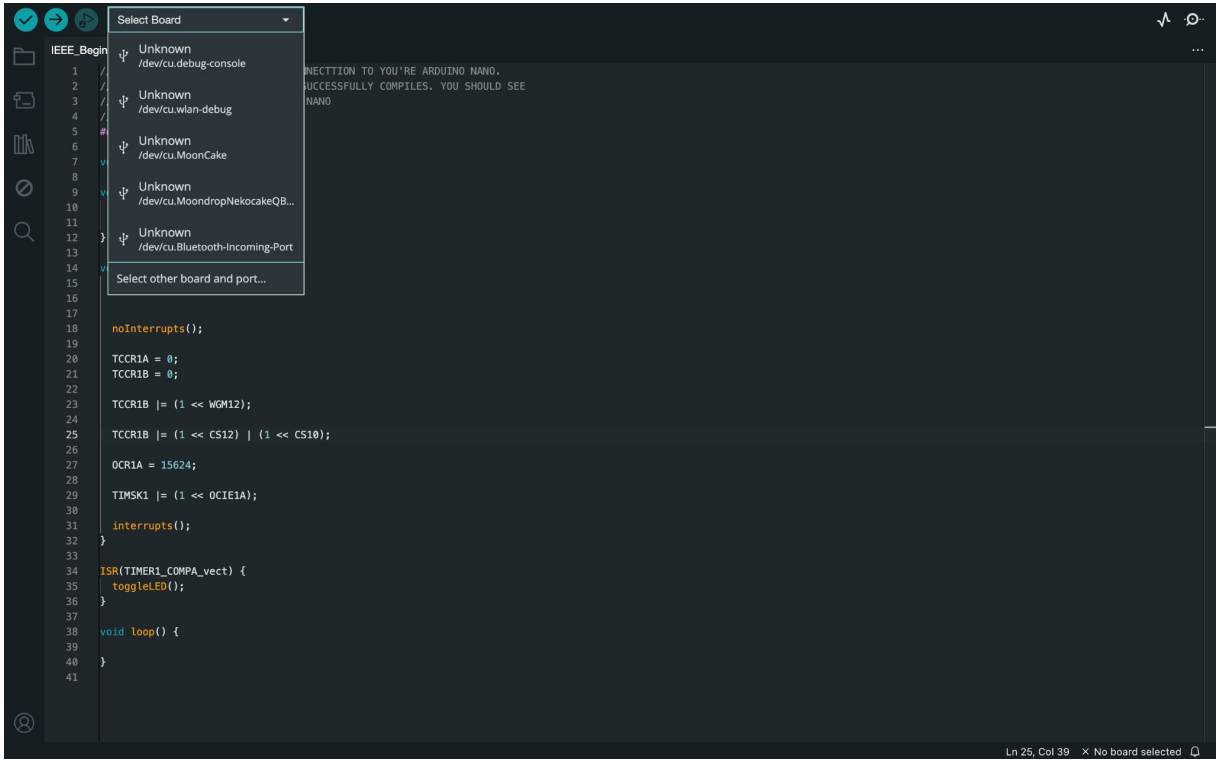
How to connect to your Arduino

- Open the Arduino IDE
- Select ***Board***



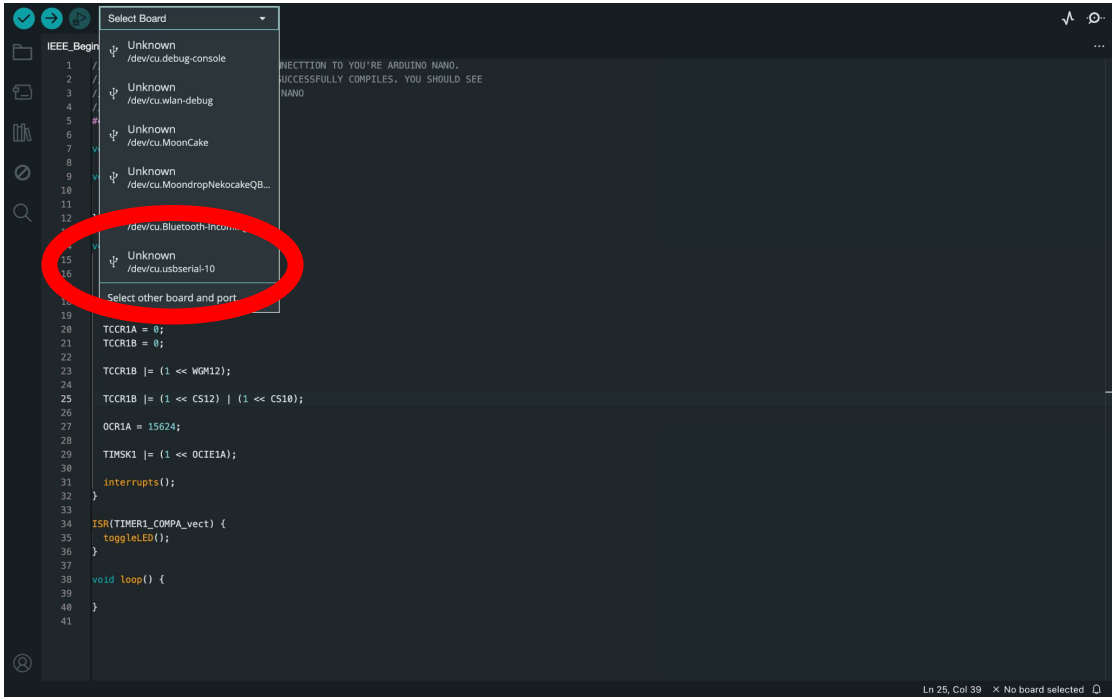
How to connect to your Arduino

- Click on it, and you'll see all the available ports to your computer



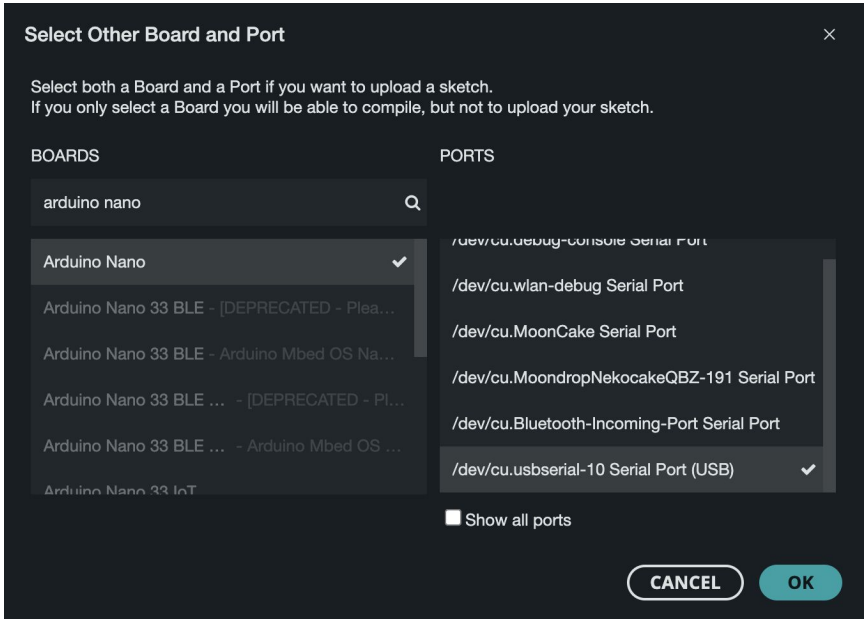
How to connect to your Arduino

- Take your USB cable and connect your computer to your Arduino
- You should see a **new usbserial pop-up**



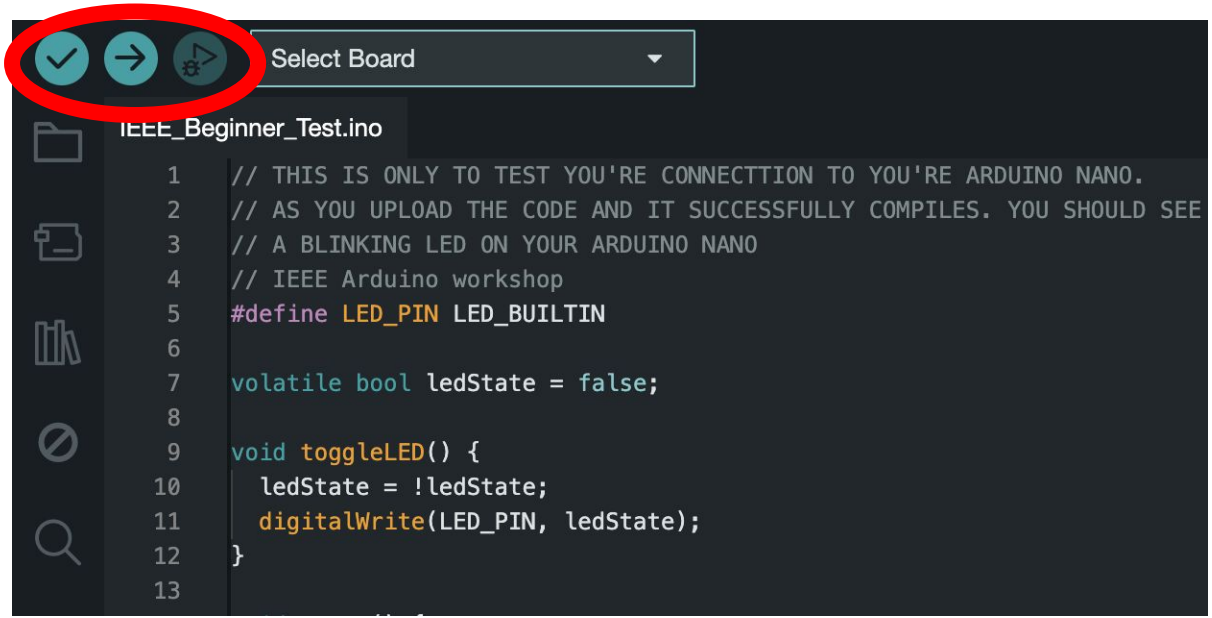
How to connect to your Arduino

- Click on the New port
- A pop-up menu will appear
- Type ***Arduino Nano*** under board
- Under ports, select the new port you plugged in



How to upload to your Arduino

- A **sketch** is what your code is written in
- After your code is written, to upload it click the **arrow button**



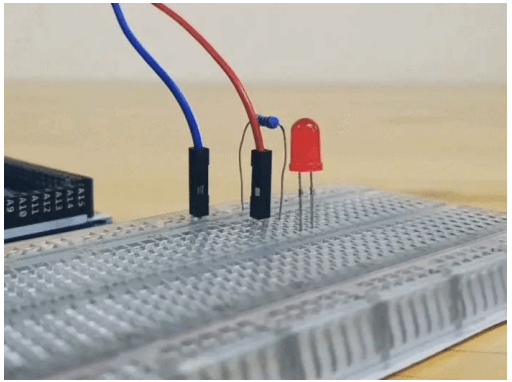
Check-in

We'll now take the time to connect to your Arduino Nano from your computer. If you have any clarifying for circuits as well, please ask us!

Please download the IEEE_Beginner_test.ino file attached to an email sent to you.

This is the code you'll be using to verify that your Arduino is connected.

What are we making…?



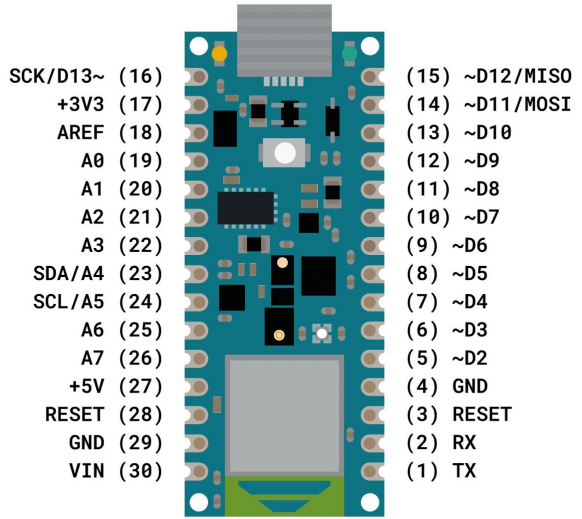
We’ll be controlling an LED with our Arduino!

- You’ll successfully set up an LED circuit
    - You’ll do this by connecting it to a GPIO pin and GND
- Program your Arduino to blink an LED on and OFF
- Control the LED using PWM signals
- Configure a button to turn an LED on and off

How do we program the Arduino?

Some key concepts include…

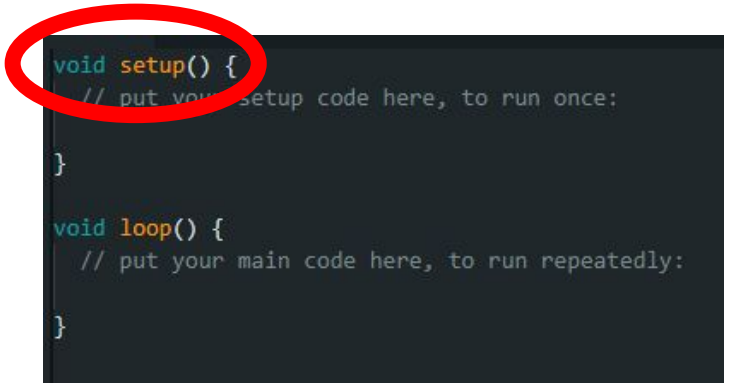
- Arduino Setup and Loop functions
- GPIO Configurations
  - Input vs. Output
  - Internal Pull-up resistors
- How to read and write to pins



Arduino IDE: setup()

What is setup()?

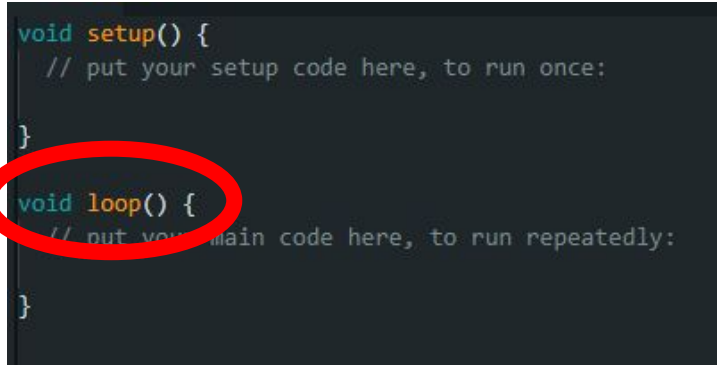
- Runs **once** when the board is powered on or reset
- Used to initialize variables and configure settings
- Essential for setting up hardware like pins, sensors, and displays
- A must-have function in every Arduino program!



Arduino IDE: loop()

What is loop()?

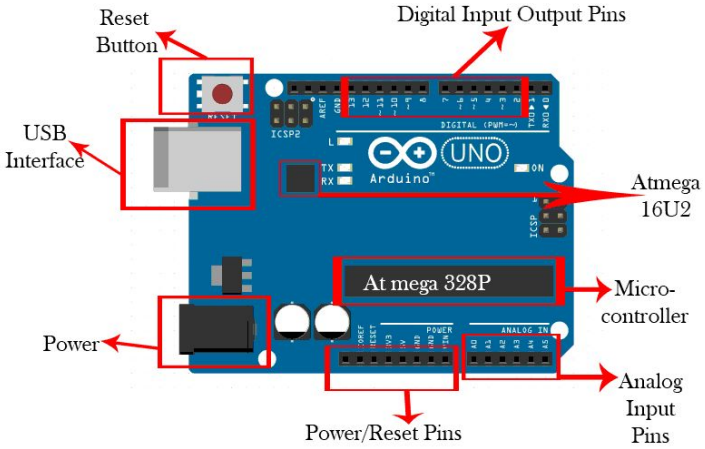
- Runs **continuously** after setup()
- Contains the **main logic** of the program
- **Keeps executing** unless powered off or reset
- Essential for automation and real-time processing



Arduino Pins

Input vs. Output

- **Input**
  - Used to read signals
  - Could be from sensors, buttons etc.
  - Reads a High (ON) or Low (OFF) signal
- **Output**
  - Use to control components
  - Could be LEDs, motors, buzzers
  - Writes a High (ON) or Low (OFF) signal



Arduino Pins

Pull up/down Resistors

- **Pull UP**
  - Ensures an input pin stays HIGH (ON) when no other signal is applied
- **Pull Down**
  - Ensures an input pin stays LOW (OFF) when no other signal is applied. **NOT available** on arduinos.
- Can be configured internally or externally depending on the microcontroller!

```
pinMode(pin1, INPUT_PULLUP);
```

Hint: We'll be using this declaration for buttons!

How do we control the pins?

`pinMode(pin, mode);`

- Defines a specific pin to **INPUT**, **INPUT_PULLUP**, or **OUTPUT**
- **Essential** for configuring **pins**

`digitalWrite(pin, value)`

- Sends a **HIGH (ON)** or **LOW (OFF)** signal to a pin.
- Works only on pins set as **OUTPUT** using pinMode().

`digitalRead(pin)`

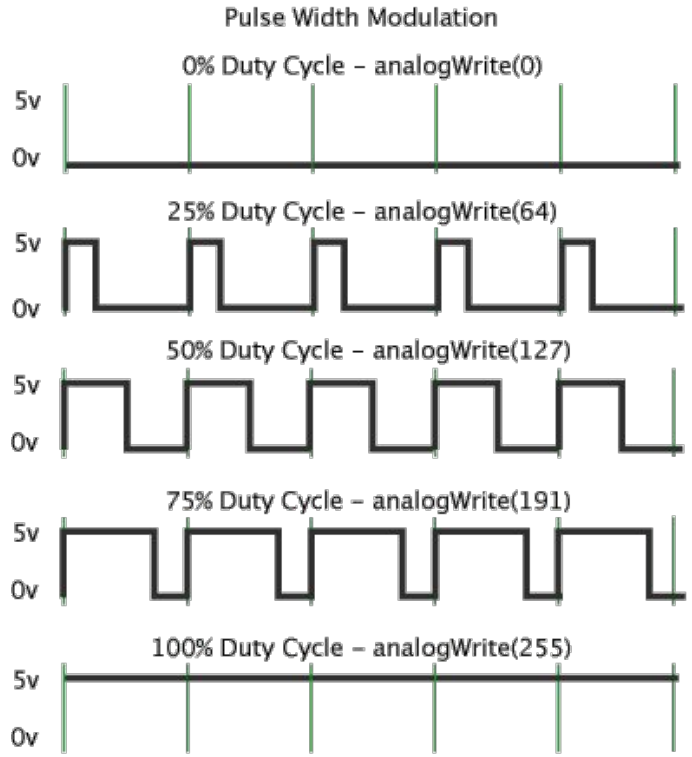
- **Reads whether the input is High (ON) or LOW (0)**
- Works only on pins set as INPUT or INPUT_PULLUP using pinMode().

```
1   #define buzzerPin 9
2
3   void setup() {
4       pinMode(buzzerPin, OUTPUT);  // Set pin 9 as an OUTPUT
5   }
6
7   void loop() {
8       digitalWrite(buzzerPin, HIGH); // Turn buzzer ON
9       delay(5000);                   // Wait 1 second
10      digitalWrite(buzzerPin, LOW);  // Turn buzzer OFF
11      delay(500);                   // Wait 0.5 a second
12  } // Loops
```

The code above sets up a buzzer, then turns it on for 5 seconds, then turns it off for 0.5 a second using the same code here!

PWM

- Available on all pins marked with **~**
- Alternates between High (ON) and Low (Off) **very fast** (490 or 980 Hz).
- Simulates **analog behavior** in digital circuits
- Very easy to program on Arduino's
  - Utilize **analogWrite(pin, level)**



This is the waveform that a PWM signal will generate. At a PWM frequency of 490 Hz, the green bars are ~2ms apart

Now it’s Your Turn!

- **Challenge 1:**
  - Program an LED to Blink on and off over a second
- **Challenge 2:**
  - Control the brightness of an LED using PWM
- **Challenge 3:**
  - Program an LED to turn on when a button is pressed
- **Bonus Challenge (Real ECE):**
  - Blink an LED without Arduino API

Bonus Challenge: Blink w/o Arduino API

The goal of this challenge is to blink an LED without using digitalWrite() and without using sleep() or (or millis() or micros() etc.)

- **Configuring Timer Interrupt**
  - Follow the tutorial linked here to generate a timer interrupt with a frequency of 1 second
    - What prescaler will you need? What threshold will you need? What mode is the counter in?
- **Writing to GPIO Port Register**
  - Now, change your code to blink the LED **without** calling digitalWrite()
    - Hint: PORTx = …..

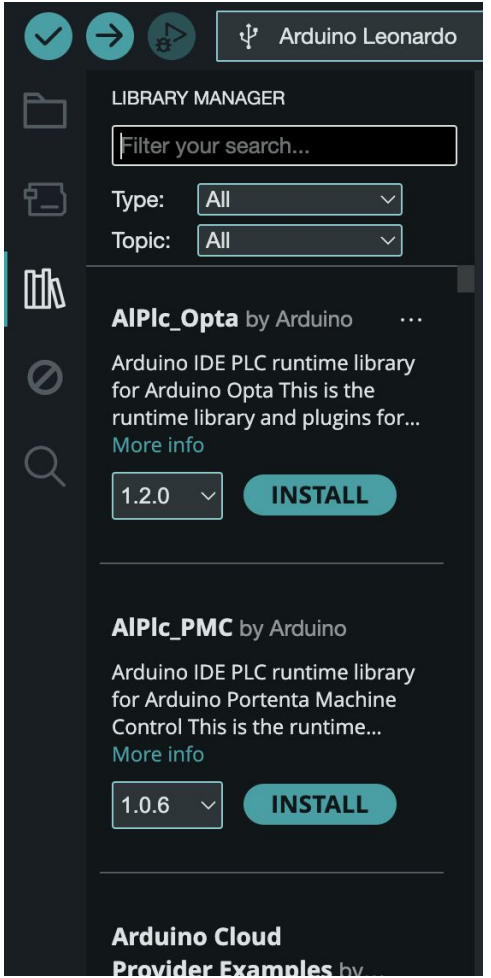
Arduino Libraries

What are libraries?

- Libraries are add-ons for Arduino
- Tons of libraries for various components
- Free to access and use

Why are libraries useful?

- Libraries save a **ton** of time
  - Don’t have to write code from scratch!
- Simplify Sensor and component integration
- Code becomes reusable and simpler!



Arduino Libraries

Using a library

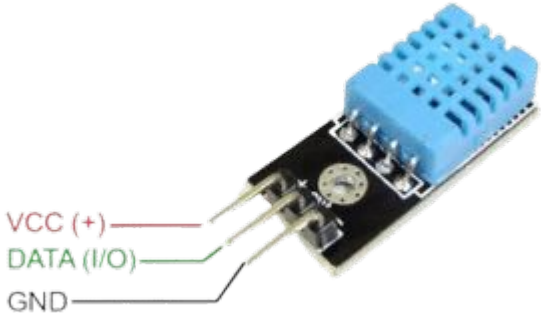
```
static Button myButton(0, buttonHandler);

void setup() {
  Serial.begin(115200);
  pinMode(PIN, INPUT_PULLUP);
}

static void pollButtons() {
  myButton.update(digitalRead(PIN));
}
```

Without a library

```
bool Button::update(uint8_t latestPoll) {
  latestPoll = (latestPoll != 0); // Collapse input into a 1/0 universe.

  if (latestPoll != _priorPoll) {
    // Input has changed since we last polled. Reset debounce timer.
    _readStartTime = millis();
  }

  // Save reading for next interrogation of update().
  _priorPoll = latestPoll;

  // Decide which debounce interval to use, depending on whether we're monitoring
  // for a next state change of "push" (0 to 1) or "release" (1 to 0).
  unsigned int debounceInterval = _pushDebounceInterval;
  if (_curState == BTN_PRESSED) {
    debounceInterval = _releaseDebounceInterval;
  }

  if ((millis() - _readStartTime) > debounceInterval) {
    // The reading has remained consistent for the debounce interval.
    // It's a legitimate state.

    if (latestPoll != _curState) {
      _curState = latestPoll; // Lock this in as the new state.
      (*_handlerFn)(_id, _curState); // Invoke callback handler.
      return true;
    }
  }

  return false; // No state change.
}
```

DHT11 Temperature and Humidity Sensor

- Reads Temperature and Humidity
- Middle Pin outputs the temperature data
- Left Pin is connected to 5V
- Right pin is connected to GND



One last challenge!

- **Challenge:**
  - Build a circuit for the DHT11 Sensor
  - Find a DHT11 Library and successfully install it
  - Print out the temperature and humidity from your sensor every second

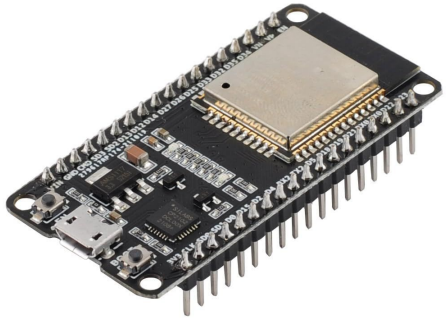
Hints

Click “more info” to go find some examples!

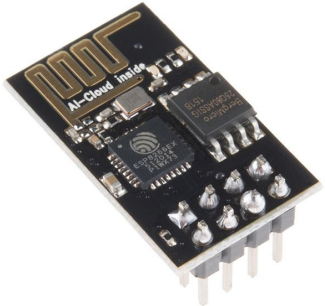
Some reputable libraries are from “Adafruit”

Intermediate Track

ESP Microcontrollers



- Microcontrollers made by Espressif Systems
- Built in WiFi, ESP32 also have Bluetooth
- More powerful than Arduinos (higher speed, memory)
- Supports Arduino IDE, MicroPython, Espressif IDE
- Great for iOT, web servers, automation, etc.



ESPRESSIF

NodeMCU

- We will be using NodeMCU, a dev board with an ESP 8266
- Adds USB, GPIO headers, voltage regulator to ESP 8266

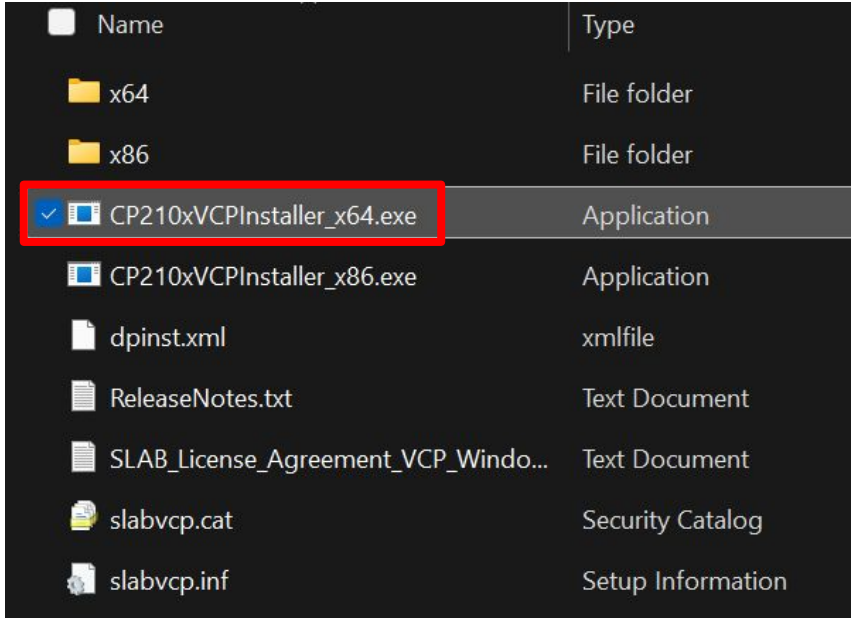
	ESP 8266	**Arduino Nano**
Clock Speed	80 MHz	16 MHz
RAM	80 KB	2 KB
Flash Storage	4MB	32 KB
WiFi	✅ Yes	❌ No
Operating Voltage	3.3V	5V

USB Drivers

- Windows users need to install a driver for the NodeMCU USB bridge from here: https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads

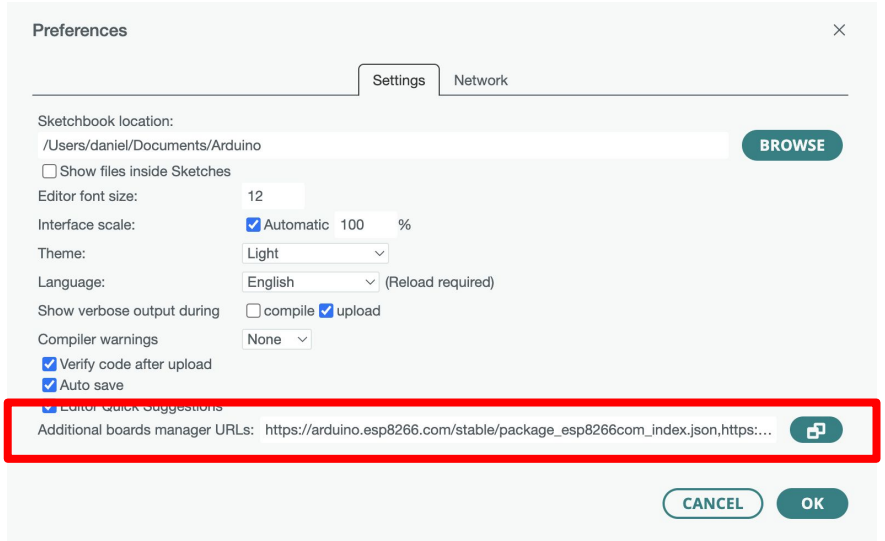
USB Drivers

Launch This →



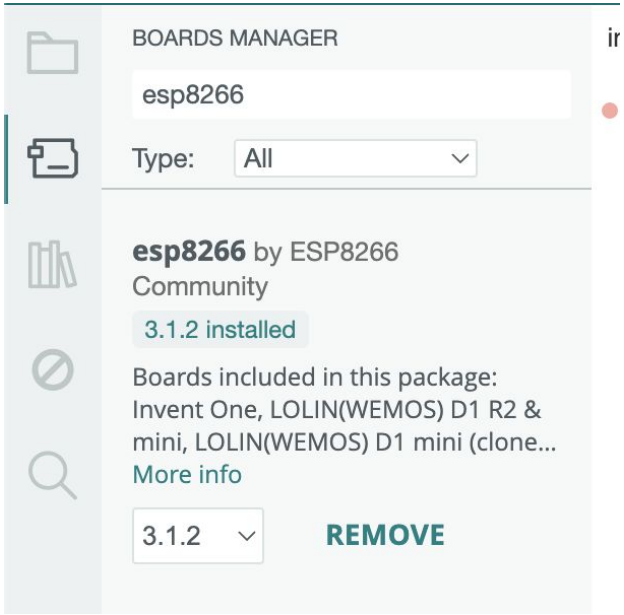
Adding ESP8266 to Arduino IDE

- Enter https://arduino.esp8266.com/stable/package_esp8266com_index.json into the *File>Preferences>Additional boards manager URLs* field.



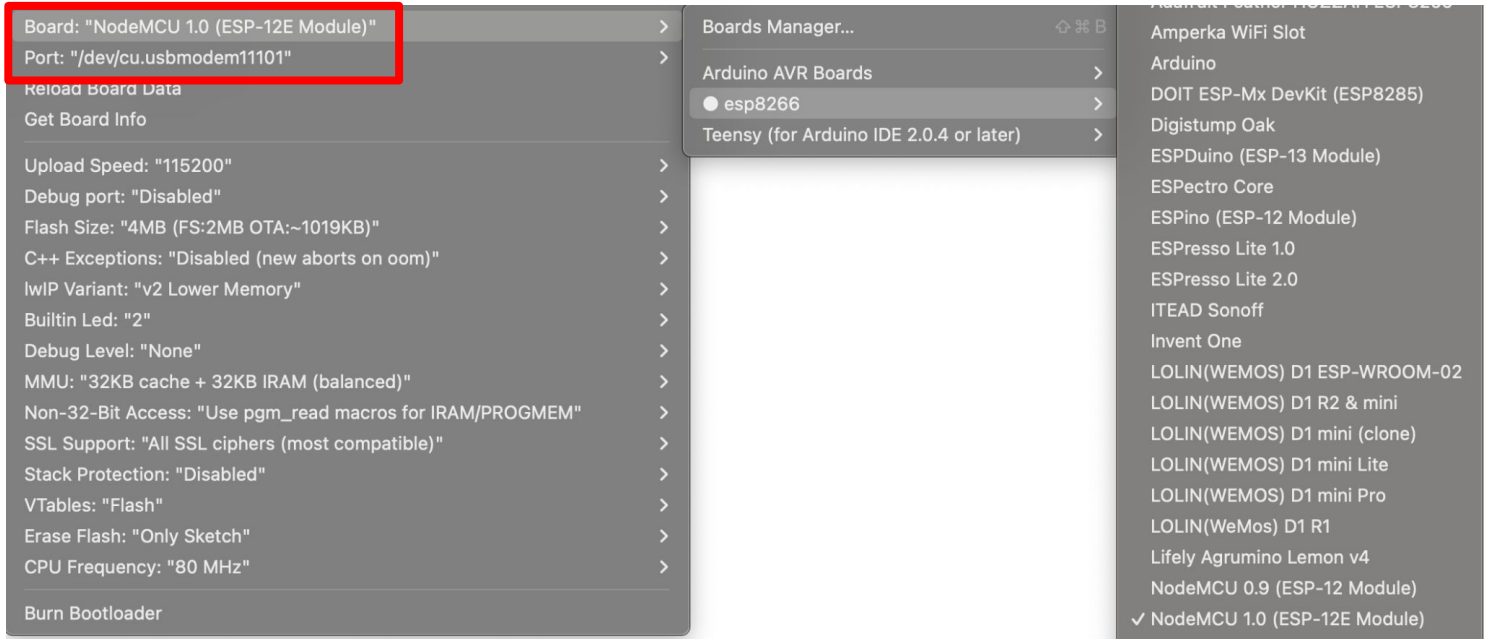
Adding ESP8266 to Arduino IDE

- Open Boards Manager from Tools > Board menu and install **esp8266** platform to install board files



Selecting Board

Select NodeMCU 1.0 from *Tools* > *Board* and the corresponding port



Port

Select NodeMCU 1.0 from *Tools* > *Board* and the corresponding port

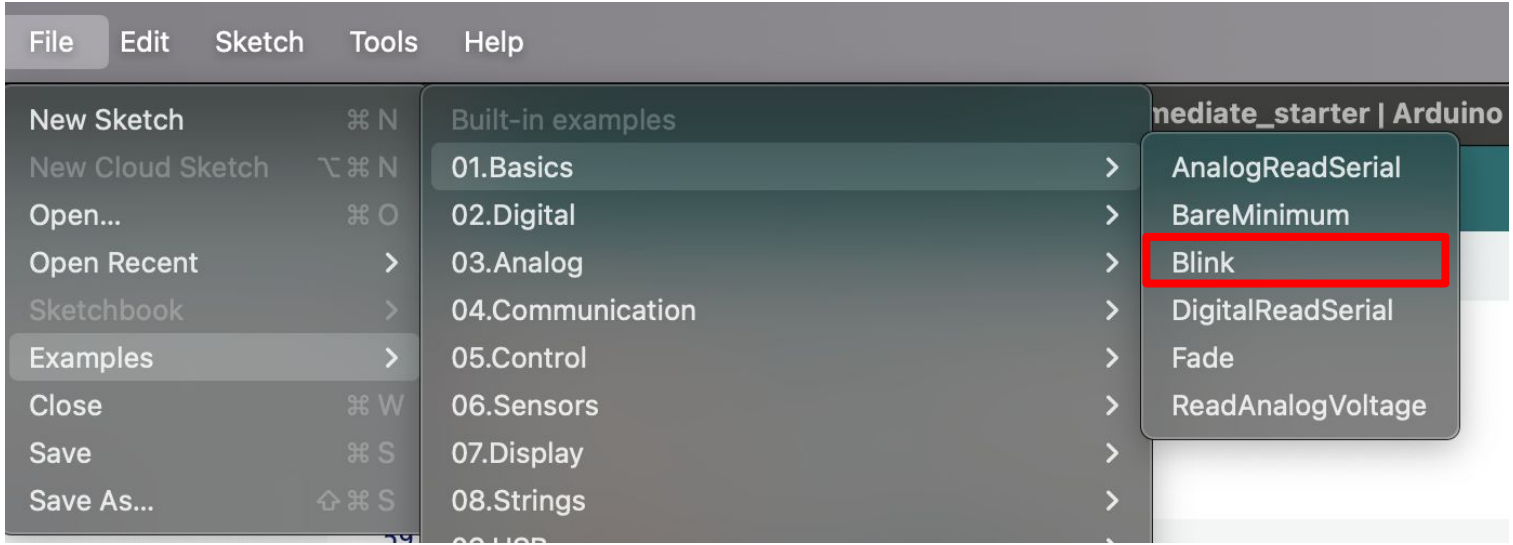
On macOS:
`/dev/usbserial`

On Linux:
`/dev/ttyUSB`

On Windows:
`COM*`

Blink

Let’s try uploading a basic sketch on the NodeMCU



Sanity Check

At this point, you should be able to compile and upload code to the NodeMCU and the Blink example should work.

Check-In

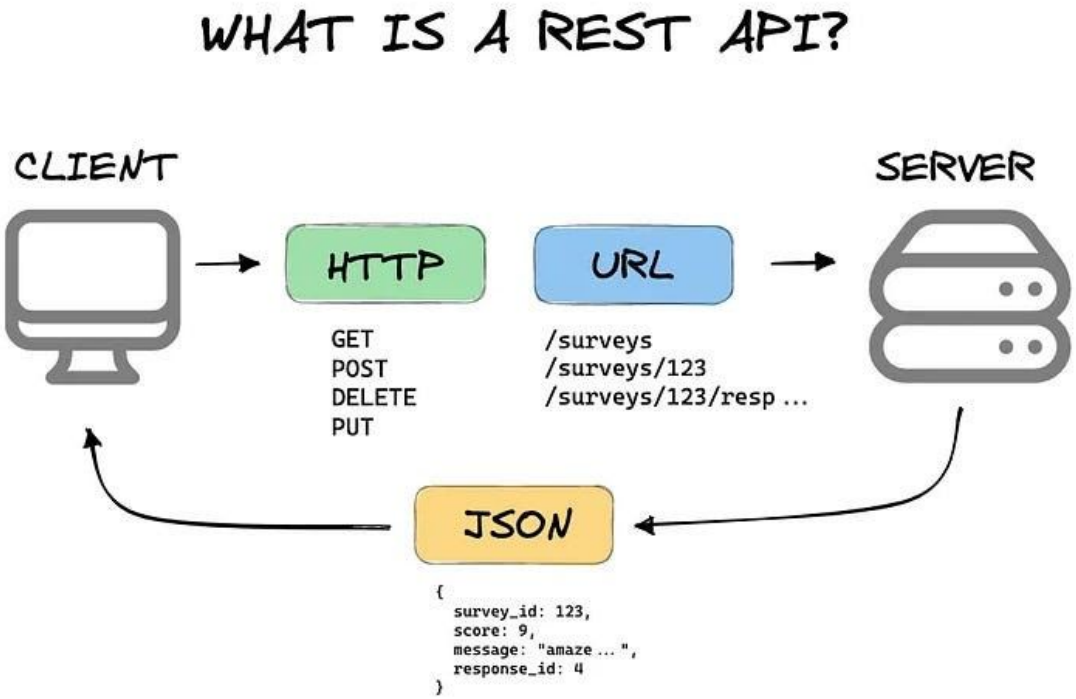
Project

- We will be displaying weather data from the internet on a 16x2 LCD display
- Using OpenMeteo API to fetch weather data

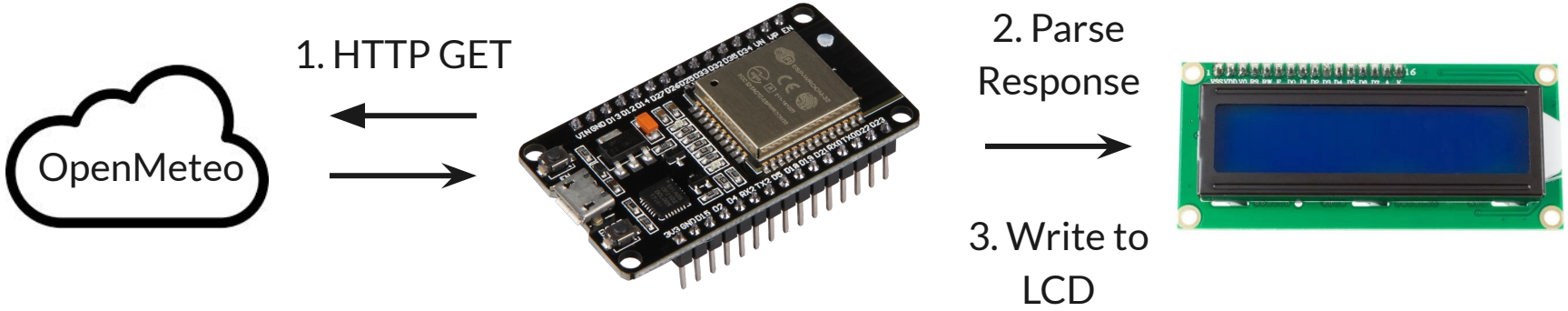
OpenMeteo

- https://open-meteo.com/en/docs
- Reference for all the different data you can fetch
- You can display whatever you want, we will focus on temperature + condition
- If you have extra time, try fetching more!

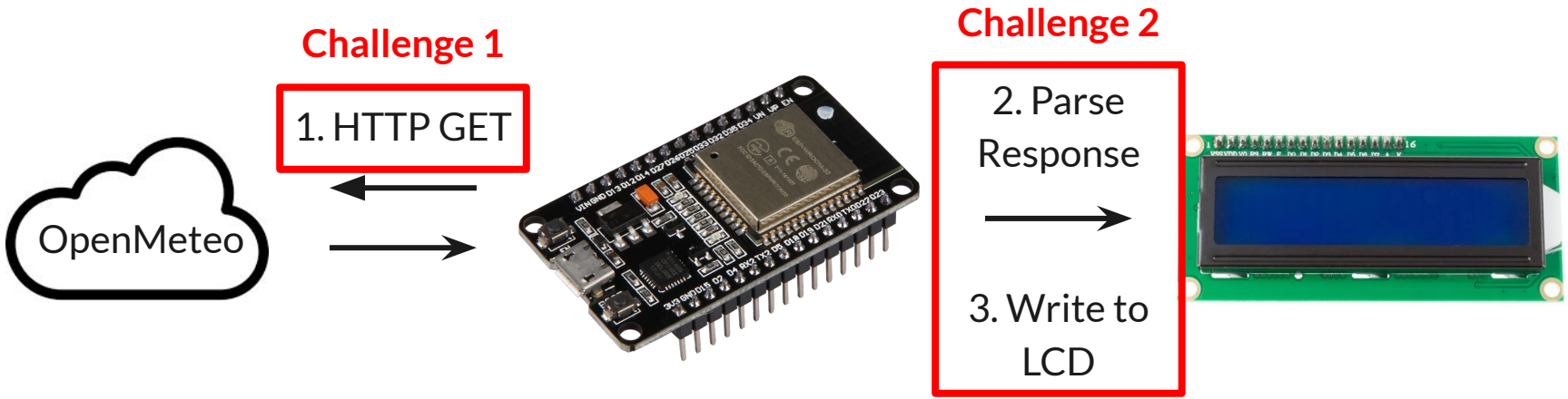
REST APIs


WHAT IS A REST API?
CLIENT
SERVER
HTTP
URL
GET
POST
DELETE
PUT
/surveys
/surveys/123
/surveys/123/resp ...
JSON
{
survey_id: 123,
score: 9,
message: "amaze ... ",
response_id: 4
}

Project

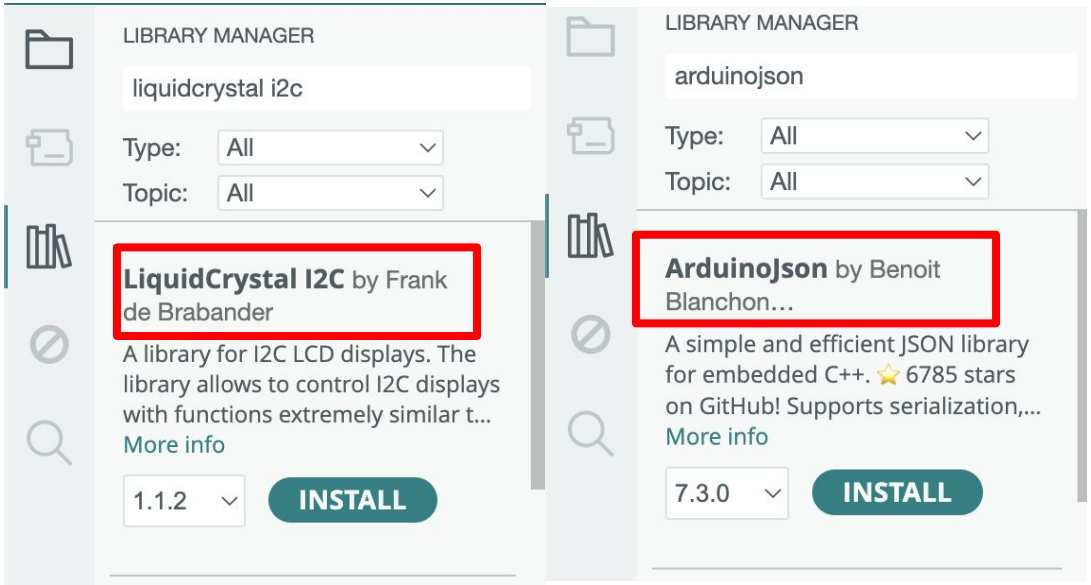


Project

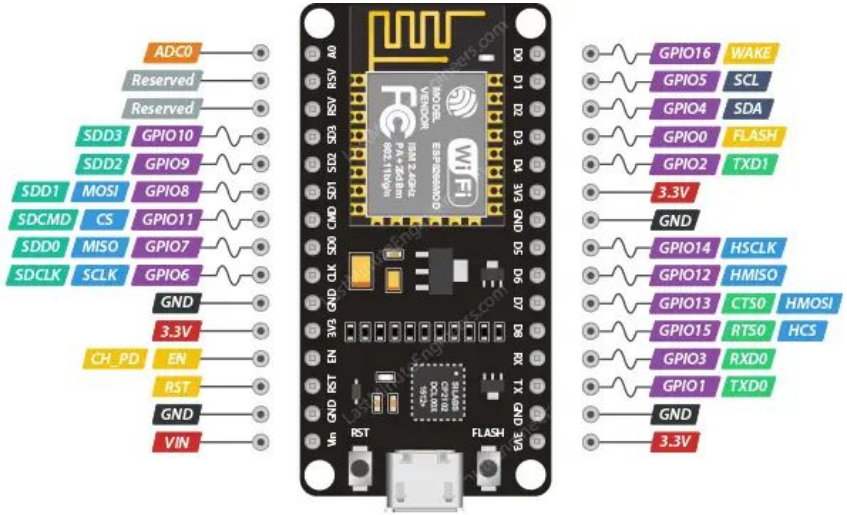

Challenge 1
1. HTTP GET
OpenMeteo
Challenge 2
2. Parse Response
3. Write to LCD

Libraries

- LiquidCrystal I2C
- ArduinoJson



NodeMCU Pinout



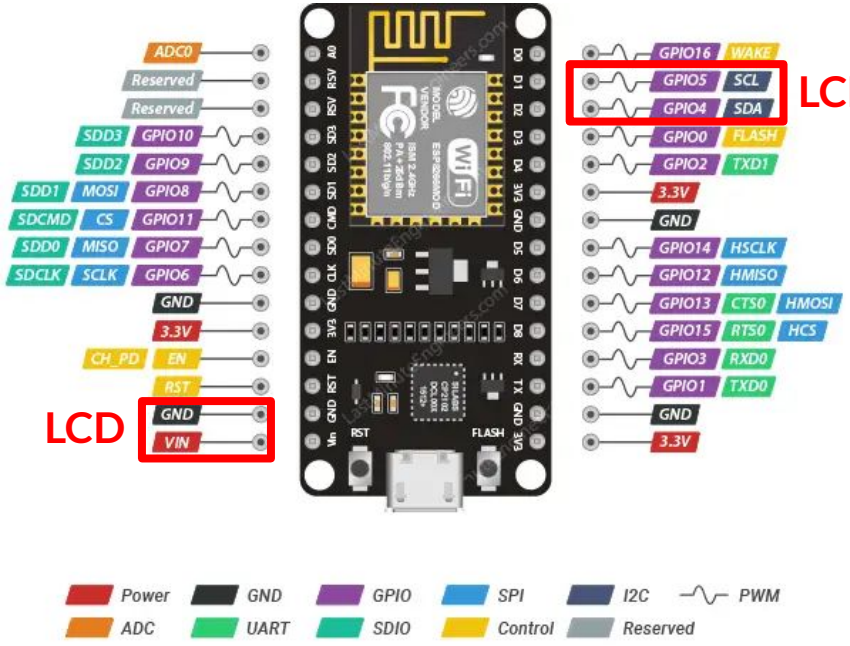
ESP8266 NodeMCU Pinout

Last Minute ENGINEERS.com

NodeMCU Pinout

D1 = SCL

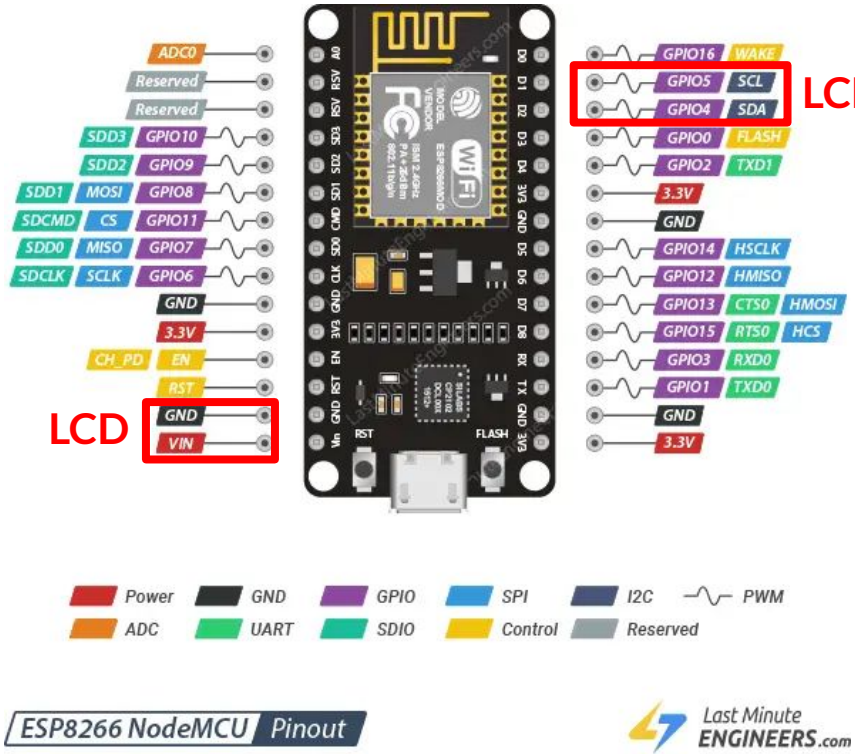
D2 = SDA



ESP8266 NodeMCU Pinout

Last Minute ENGINEERS.com

NodeMCU Pinout



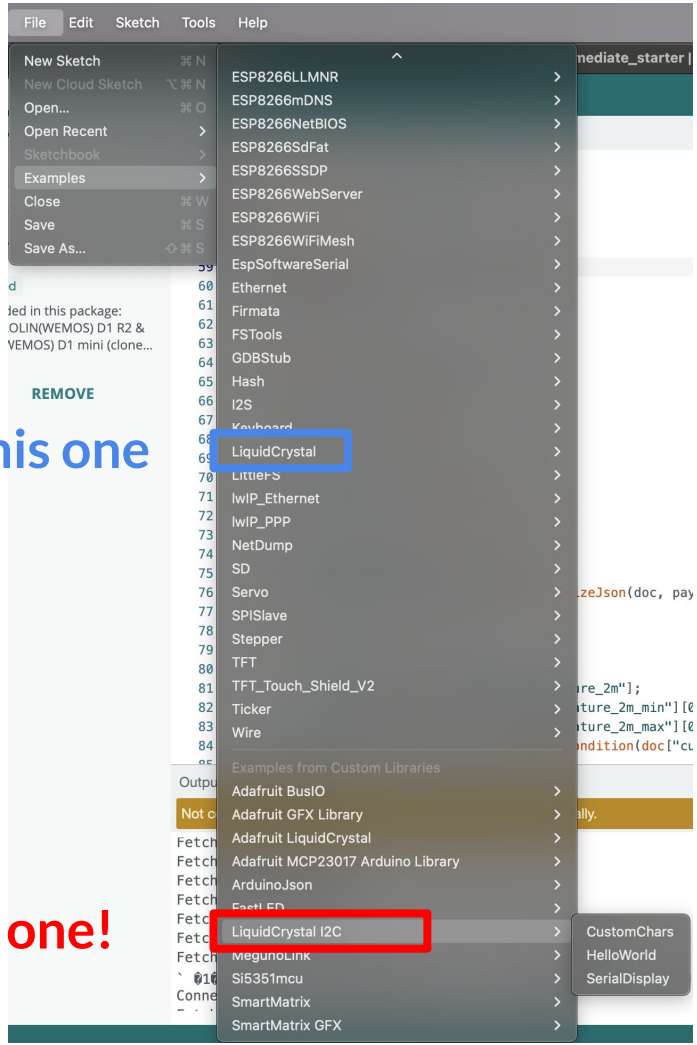
D1 = SCL

D2 = SDA

Connect LCD using 4 jumpers

Test LCD

- *File>Examples>LiquidCrystal I2C>HelloWorld*
- Contrast may need to be adjusted on the back of the LCD



Not this one

This one!

Sanity Check

At this point, you should be able to run the *HelloWorld* example on your NodeMCU and see output on the LCD.

Check-In

Download Starter Code

- github.com/ieee-wpi/arduino
- Starter code located in *intermediate_starter*
- You will be filling in the starter code wherever you see *TODO* using the slides as reference

LCD Code

```
// Initialize LCD object (I2C address, num col, num row)
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Initialize LCD
lcd.init();

// Turn on LCD backlight
lcd.backlight();

// Set cursor position (x [0-15], y [0-1])
lcd.setCursor(0,0);

// Clear LCD
lcd.clear();

// Print message
lcd.print("Bruh");
```

Serial

Make sure you can see print outs from Serial Monitor!

Open *Tools > Serial Monitor*

Select correct baud rate (115200)

```
void setup() {
 // Start Serial
 Serial.begin(115200);
}

void loop() {
 // Print this out
 Serial.println("Bruh");
}
```

Add Serial to HelloWorld

Add these lines to *HelloWorld* and make sure you can see Serial messages.

Open *Tools > Serial Monitor*

Select correct baud rate (115200)

```
void setup() {
 // Start Serial
 Serial.begin(115200);
}

void loop() {
 // Print this out
 Serial.println("Bruh");
 delay(1000);
}
```

Connecting to WiFi

- We first connect the ESP to WiFi

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#define SSID "SSID"
#define PASSWORD "PASSWORD"

void setup() {
 Serial.begin(115200);
 WiFi.begin(SSID, PASSWORD);

 // Wait for WiFi to connect
 while (WiFi.status() != WL_CONNECTED) {
   delay(100); // Ensure delay or yield is here — WDT will reset
 }
}
void loop() {
 if (WiFi.status() == WL_CONNECTED) {
   Serial.println("Bruh we good");
 } else {
   Serial.println("We cooked");
 }

 delay(10000); // Don't spam
}
```

Making HTTP Request

- We then make an HTTP request

```
WiFiClient client;
HTTPClient http;

http.begin(client, url); // Start HTTP request
int httpCode = http.GET(); // Perform GET request

if (httpCode == HTTP_CODE_OK) { // If successful
  String payload = http.getString(); // Get response
  Serial.println(payload);
}

 // Close connection, free up memory
 http.end();
```

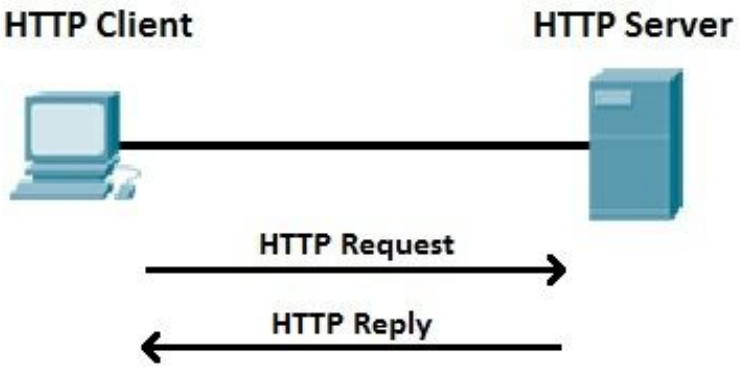
API URL

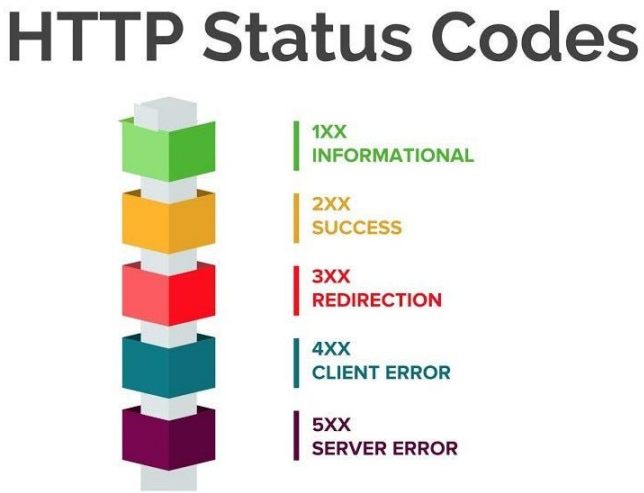
```
const char* WEATHER_URL = "http://api.open-meteo.com/v1/forecast?"
                          "latitude=?" // TODO: REPLACE ? WITH LATITUDE
                          "&longitude=?" // TODO: REPLACE ? WITH LONGITUDE
                          "&current=temperature_2m,relative_humidity_2m,apparent_temperature,weather_code"
                          "&daily=temperature_2m_max,temperature_2m_min"
                          "&timezone=America%2FNew_York"
                          "&forecast_days=1"
                          "&temperature_unit=fahrenheit";
```

HTTP Basics

200: OK This what we want

404: Not Found

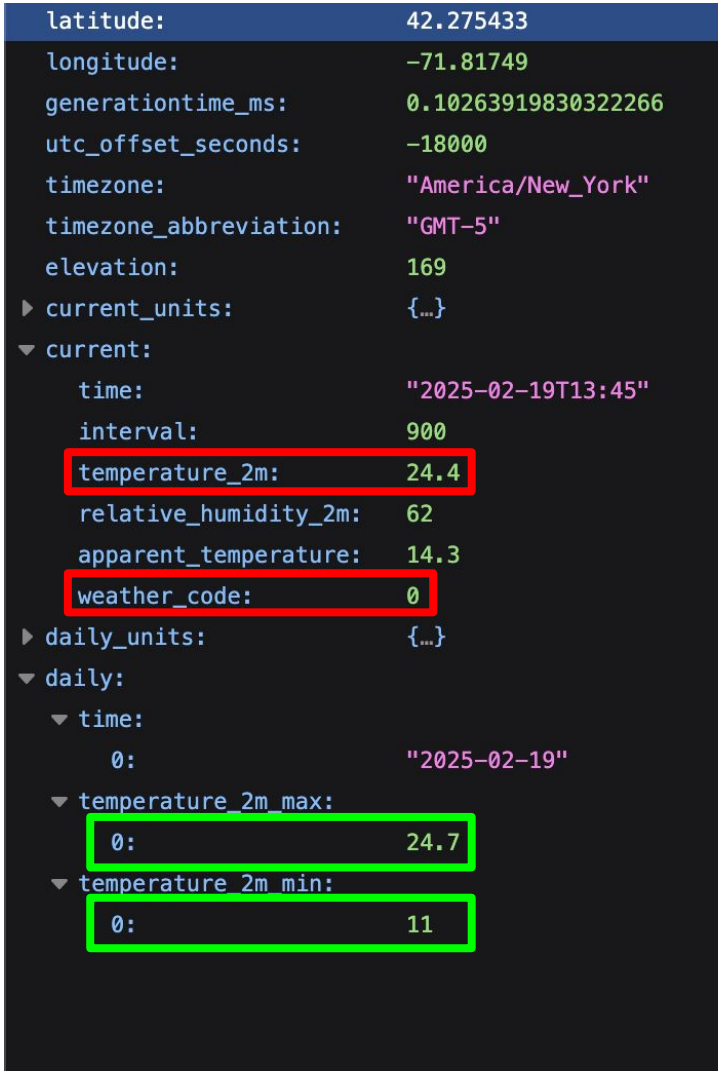




Parsing JSON

- Look at JSON structure
- Make sure keys are correct

You will need to parse temperature and condition at a minimum



Temp

Condition

Daily Min

Daily Max

(optional)

Parsing JSON

- Temp, condition you index using keys
  - [“key1”][“key2”]
- Daily min/max represent arrays. The last key is an array index.
  - [“key1”][“key2”][index]
  - index is a number, not a string!

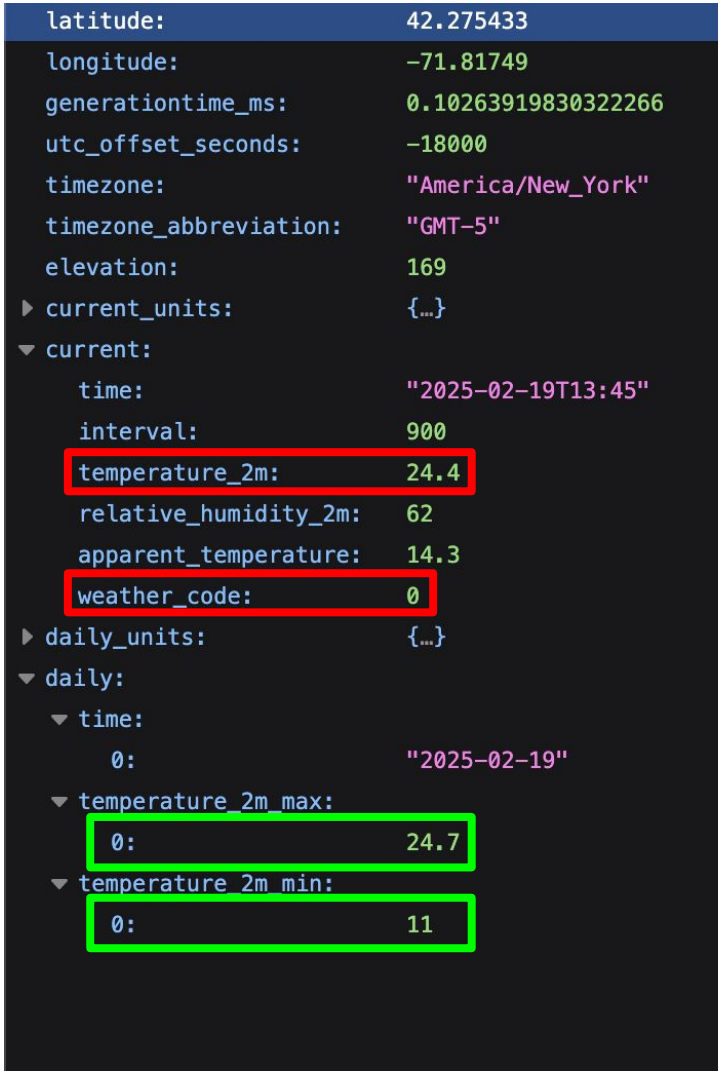
Temp

Condition

Daily Min

Daily Max

(optional)



JSON Code

- We then turn the response into a JSON which we can index

```
String response = http.getString();

DynamicJsonDocument json(1024); // Create JSON buffer
DeserializationError error = deserializeJson(json, response);

float bruh = json["key1"]["key2"];
float bruhButArray = json["key1"]["key2"][index];
```

You will need to parse temperature, and weathercode

Getting Weather Condition

- OpenMeteo returns a weather code we need to parse into a weather condition

```
// Returns weather condition from weather code
const char* getWeatherCondition(int code) {
 switch (code) {
   case 0: case 1: return "Clear";
   case 2: return "Partly Cloudy";
   case 3: return "Overcast";
   case 45: case 48: return "Fog";
   case 51: case 53: case 55: return "Drizzle";
   case 56: case 57: return "Freezing Drizzle";
   case 61: case 63: case 65: return "Rain";
   case 66: case 67: return "Freezing Rain";
   case 71: case 73: case 75: case 77: return "Snow";
   case 80: case 81: case 82: return "Rain";
   case 85: case 86: return "Snow";
   case 95: case 96: case 99: return "Thunderstorm";
   default: return "Unknown";
 }
}
```

ESP Exceptions

- This means you have a bug lol
- Use print statements to find the problematic code

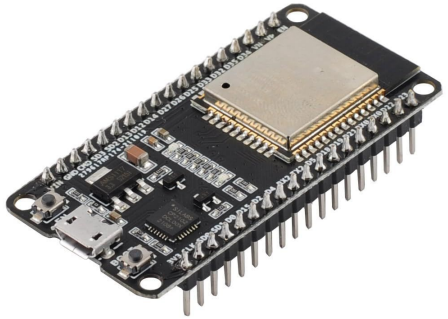
```
@@ @2@N@r@@o|@l @ $  c@ @@2r@@$@N@@o@l` @@r@$@$ @ @Connecting to WiFi...
---------------- CUT HERE FOR EXCEPTION DECODER ----------------

Soft WDT reset

Exception (4):
epc1=0x40201554 epc2=0x00000000 epc3=0x00000000 excvaddr=0x00000000 depc=0x00000000

>>>stack>>>

ctx: cont
sp: 3ffffe50 end: 3fffffd0 offset: 0160
3fffffb0:  3fffdad0 00000000 3ffee998 40206a44
3fffffc0:  feefeffe feefeffe 3fffdab0 40100f91
<<<stack<<<

---------------- CUT HERE FOR EXCEPTION DECODER ----------------

 ets Jan  8 2013,rst cause:2, boot mode:(3,6)

load 0x4010f000, len 3424, room 16
tail 0
chksum 0x2e
load 0x3fff20b8, len 40, room 8
tail 0
chksum 0x2b
csum 0x2b
v00049110
~ld
```

Your Turn!

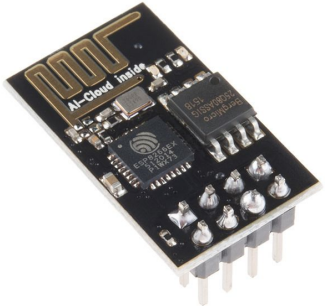
- **Challenge 1:**
  - Send HTTP GET request to OpenMeteo
- **Challenge 2:**
  - Parse response and write to LCD
- **Extra Challenge:**
  - Min and max daily temp
  - Custom arrow characters

Advanced Track

ESP Microcontrollers



- Microcontrollers made by Espressif Systems
- Built in WiFi, ESP32 also have Bluetooth
- More powerful than Arduinos (higher speed, memory)
- Supports Arduino IDE, MicroPython, Espressif IDE
- Great for iOT, web servers, automation, etc.



ESPRESSIF

NodeMCU

- We will be using NodeMCU, a dev board with an ESP 8266
- Adds USB, GPIO headers, voltage regulator to ESP 8266

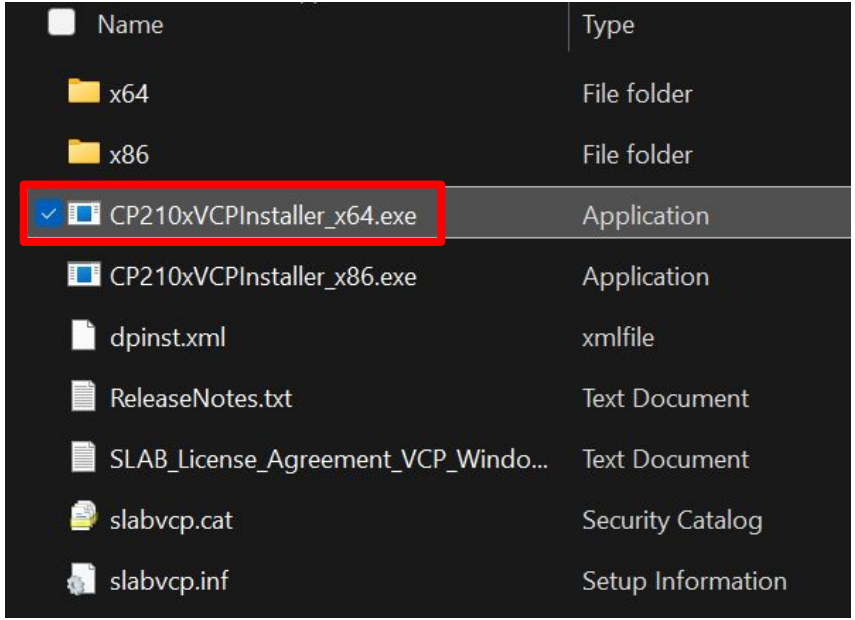
	ESP 8266	**Arduino Nano**
Clock Speed	80 MHz	16 MHz
RAM	80 KB	2 KB
Flash Storage	4MB	32 KB
WiFi	✅ Yes	❌ No
Operating Voltage	3.3V	5V

USB Drivers

- Windows users need to install a driver for the NodeMCU USB bridge from here: https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads

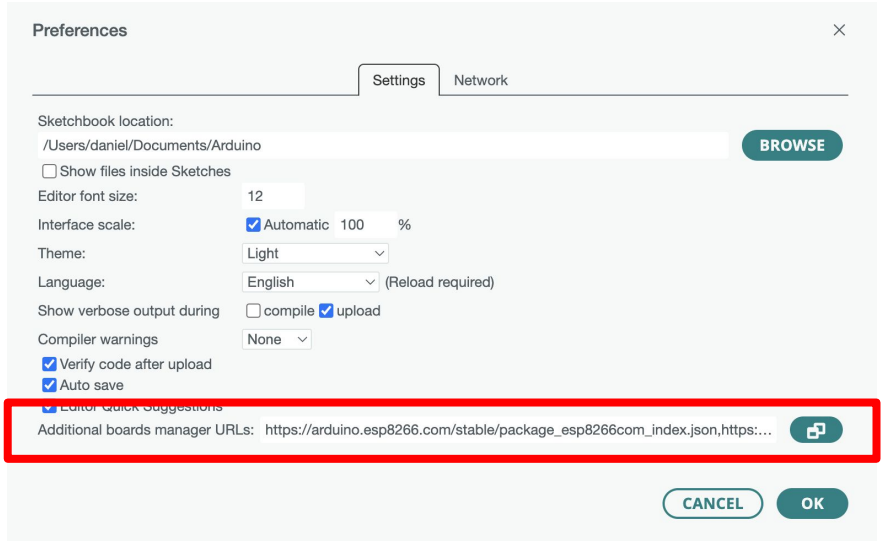
USB Drivers

Launch This →



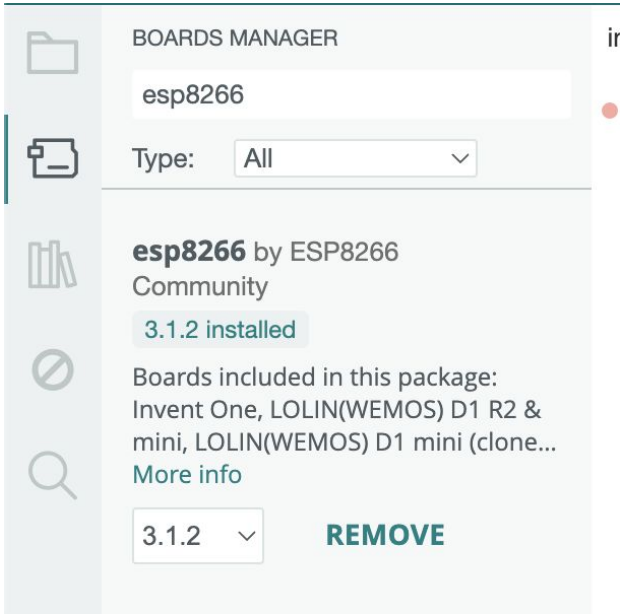
Adding ESP8266 to Arduino IDE

- Enter https://arduino.esp8266.com/stable/package_esp8266com_index.json into the *File>Preferences>Additional boards manager URLs* field.



Adding ESP8266 to Arduino IDE

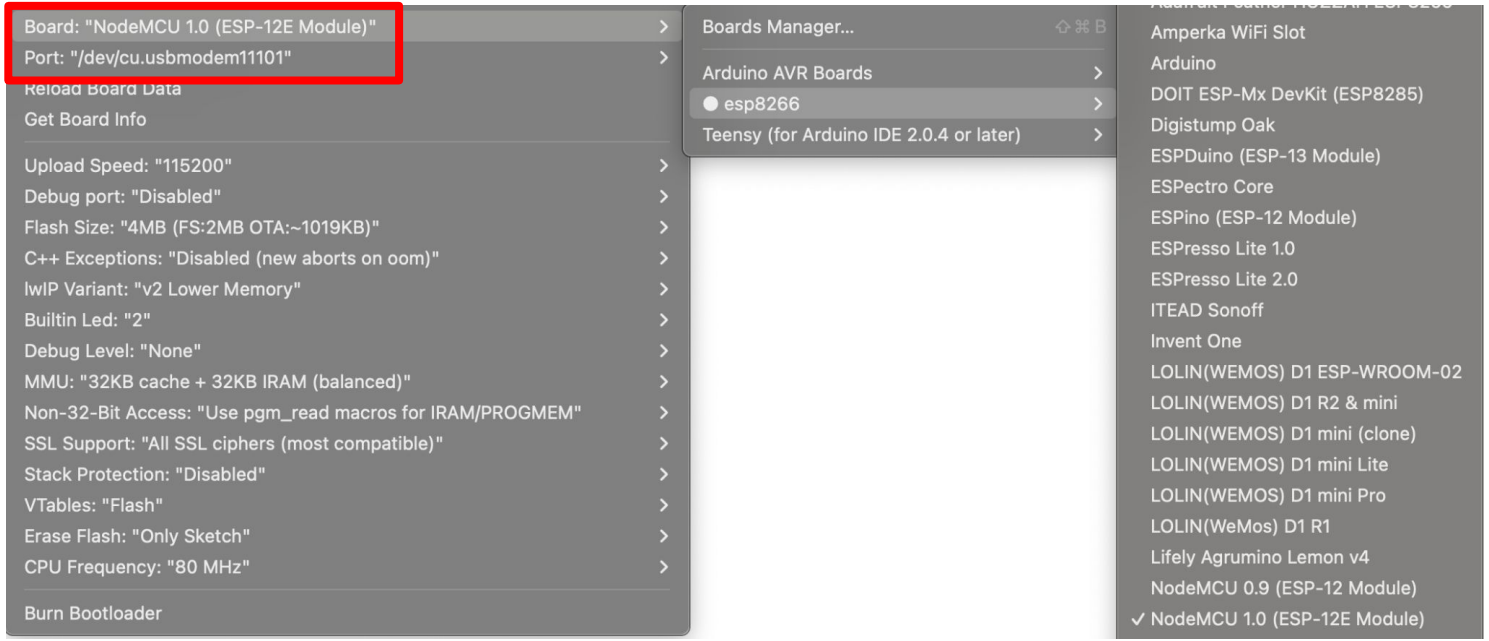
- Open Boards Manager from Tools > Board menu and install **esp8266** platform to install board files



Selecting Board and Port

Select NodeMCU 1.0 from Tools > Board and the corresponding port

Double check!



Port

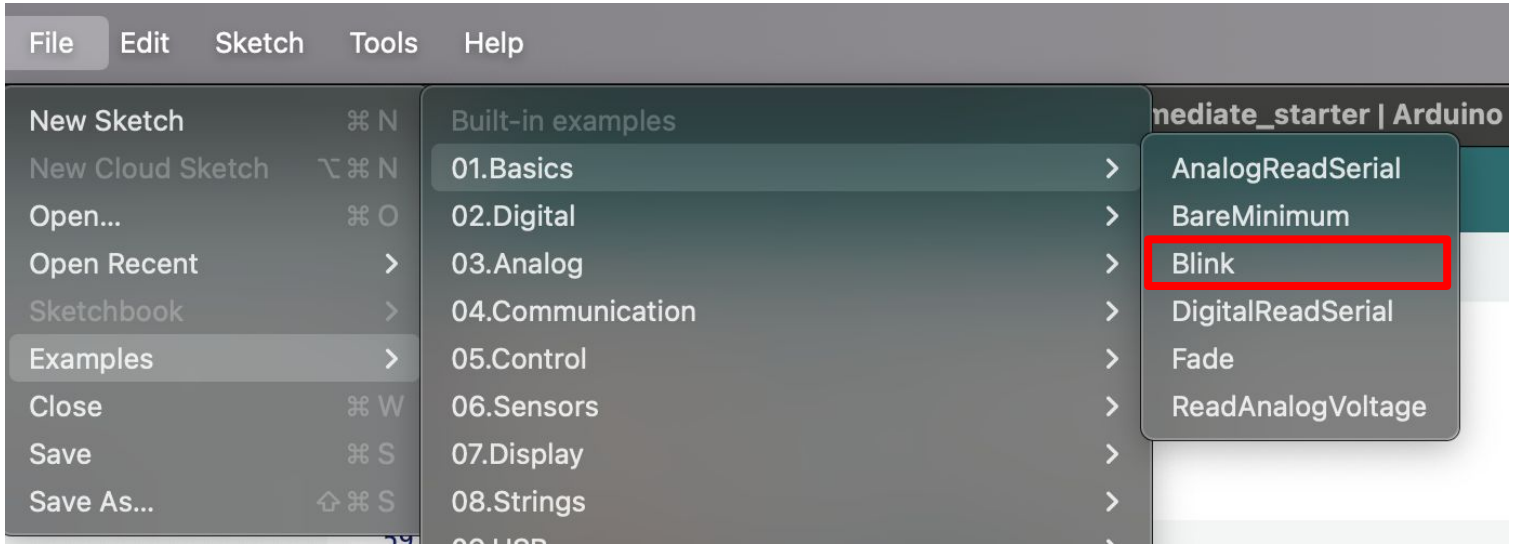
Select NodeMCU 1.0 from Tools > Board and the corresponding port

On macOS/Linux:
`/dev/*`

On Windows:
`COM*`

Blink

Let’s try a basic sketch on the NodeMCU



Sanity Check

At this point, you should be able to compile and upload code to the NodeMCU and the Blink example should work.

Check-In

ESP Now

- Wireless protocol created by EspressIf
- Fast and Low Power
- Easy to set up
- Works with all configurations
  - One-to-one
  - One-to-many
  - Many-to-one
  - Many-to-many
  - Broadcast

MAC Address

- Each device has a **unique** MAC Address
- We need this to send data to a **specific** device
- **REMEMBER THIS VALUE**

```
#include <ESP8266WiFi.h>

void setup(){
  Serial.begin(115200);
  Serial.println();
  Serial.print("ESP Board MAC
Address:  ");
  Serial.println(WiFi.macAddress());
}

void loop(){


  }
```

Callbacks

- ESP-Now uses **callbacks** to work
- A callback is a special name for a function
- Functions can be **registered** to run after **specific** events
  - esp_now_register_send_cb(OnDataSent);
  - esp_now_register_recv_cb(OnDataRecv);

Callback Example

```
void OnDataRecv(uint8_t * mac, uint8_t *incomingData, uint8_t len) {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
  Serial.println(len);
  Serial.print("Enable value: ");
  Serial.println(myData.enabled);
  // Do stuff with the data
}

void setup() {
  // Other setup code...

  esp_now_register_recv_cb(OnDataRecv);
}
```

Setup

- WiFi must be enabled for ESP-Now to work
- Init ESP-Now with esp_now_init() :D
- Set ESP-Now role
  - ESP_NOW_ROLE_IDLE
  - ESP_NOW_ROLE_CONTROLLER
  - ESP_NOW_ROLE_SLAVE
  - ESP_NOW_ROLE_COMBO
  - ESP_NOW_ROLE_MAX
- Add peers
- Register callbacks

Message Structure

- Predetermined “format” of the message to send
- **MUST** be the same on both the sender and the receiver
- Can be up to 250 bytes large
- Structure this however you need!
-

```
typedef struct struct_message {
 bool enabled;
 bool hasString;
 char topline[17];
 char bottomline[17];
} struct_message;
```