SEICHE 2022
Basic Arduino Programming

Instructor: Paul Frommeyer

www.paulfrommeyer.com

Corporate Sponsor: DXC Technology

Lesson Plan Overview

Lesson 1 – Intro and Setup
[may require 2 classes]

- Introduction to class format
- Overview of lesson plan
- Presentation format (monitor, camera, screen, whiteboard)
- Review of microcontrollers and types of boards
- SEICHE LED display architecture
  - ESP8266 pinout
  - High level architecture

Lesson 2 – Laptop operation review – Windows and Linux

- Inventory of USB drives
- Installation of Arduino IDE software
- Installation of CH340/ESP8266 serial port drivers (Windows only)
- Control panel/settings location
- Home directories and folder hierarchy
- Arduino file locations
- Search functions
- (Windows) Device Manager
- (Linux) Konsole
- Copying flash drive contents [critical]
- Open questions and issues
- **IDE essentials**
- Starting the Arduino IDE
- Basic Arduino sketch (program) structure
- Loading example sketches
- Loading and configuring new boards
- Connecting boards
- Identifying the microcontroller serial port
  - Linux
  - Windows

Lesson 3 – Libraries, Sketch structure, Serial Monitor, Variables, Binary Number System Pt1

- Libraries
- Sketch structure (A note on brace formatting)
- The serial port monitor
- Printing to the serial port monitor
- Variables and the assignment operator
- Binary number system Pt. 1.

Lesson 4/5 – The Binary Number System Cont. (may take 2-3 lessons)

- Numerals vs numbers
- Review: the base 10 system and digit place values
- New: the base 2 system and digit place values
- Bits and bytes and nybbles
- Binary addition and subtraction
- Formatting printed output
- Shifting and exponents

Lesson 6/7 – Integers and Arithmetic Operators (may take 2 lessons)

- **The integer data type**
- **Variables and the assignment operator**
- **Autoincrement and autodecrement**
- **Arithmetic Operators**
- **Assigning arithmetic results to a variable**
- **Bytes as data types**

Lesson 8 – Communicating with Sensors

- **Reading a potentiometer**
- **Reading a button**
- **Initializing I2C sensors**
- **Reading I2C sensors**

Lesson 9 – Characters and Arrays

- The hexadecimal number system
- Representing characters as numbers
- The ASCII character code
- The char data type
- Strings and character arrays

Lesson 10 – Comparison Operators

- ==, =>, <=
- strcmp
- Binary comparisons
- Uploading a sketch to the microcontroller

Lesson 11 – Control Structures

- if-then-else
- while
- for-next

Lesson 12 – Programming the LED matrix

- Initializing the SPI interface
- Controlling display brightness
- Lighting and clearing a single pixel
- Displaying text on the LED matrix
- Default fonts
- Nested for-next loops

Lesson 3 – Libraries, Sketches
Serial Monitor, Variables, Binary Numbers

Part A

- Libraries and the Library Manager
- Sketch Structure
  - Setup and Loop
  - The parts of a program (sketch): values, statements, functions
- The serial port monitor
- Printing to the serial port monitor

Part B

- Data types in detail: bytes, integers
- Variables
- Assigning values to variables
- Printing variables
- Simple arithmetic operators

Part C [Time permitting]

- Introduction to the binary number system

The Library Manager

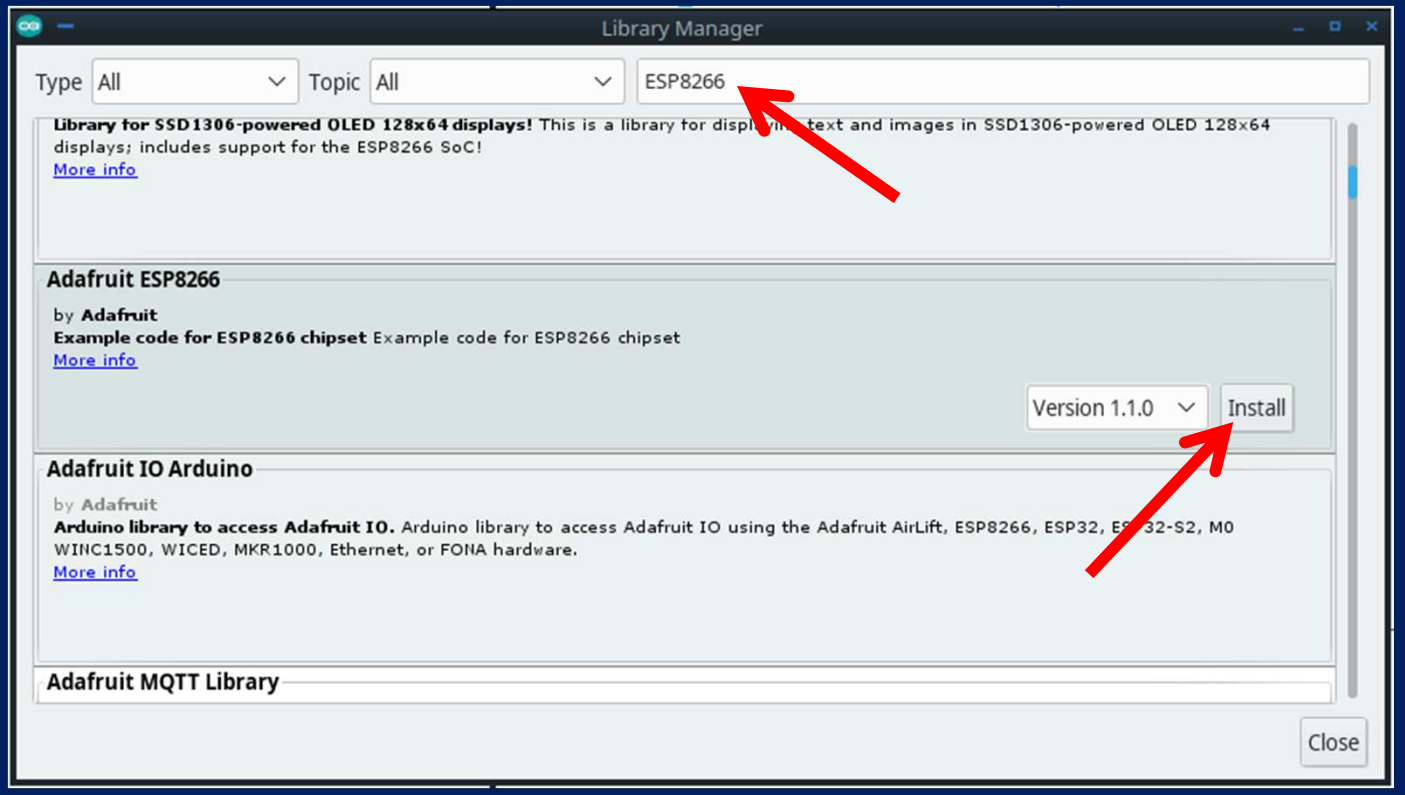
- A library is a collection of software written by others which performs a specific function or related functions
- Libraries allow us to use code written by others in our own projects
- In the Arduino IDE, the Library Manager handles the selection and installation of libraries
- Libraries are board-specific, so you must have the proper board definition(s) installed first, before you can install libraries for that board
- The Library manager is accessed from the Sketch menu

Let’s make sure everything is set up correctly for our Wemos D1 Mini board with its ESP8266 microprocessor…

Library Manager – ESP8266

ADD A LIBRARY

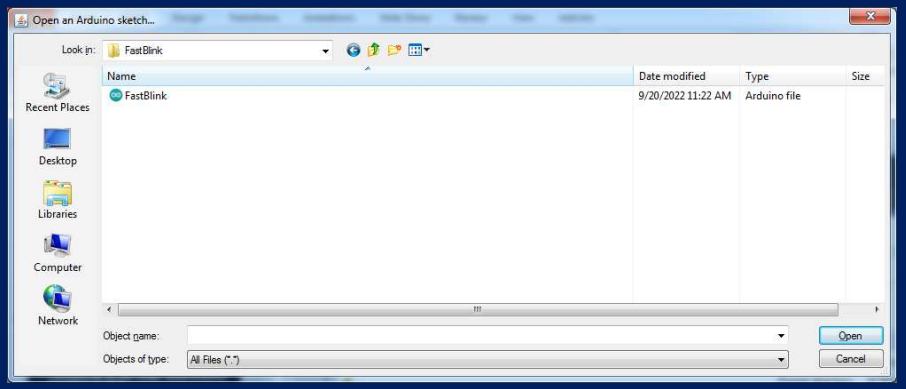
- Open “Manage Libraries” under “Include Library” in Sketch menu
- Enter “ESP266” in seach box
- Install the Adafruit ESP8266 library



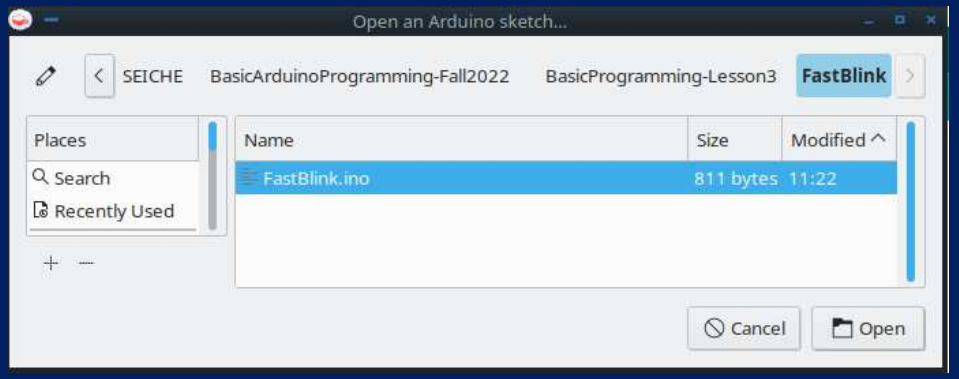
Let's load our own sketch!

- File menu->Open
- Navigate to the"FastBlink.ino" *file* in the **Lesson3->FastBlink** folder you copied from your USB drive at the beginning of class
- The sketch will open in the IDE
- Click on the "UPLOAD" arrow icon
- Once compile and upload are finished, the sketch will immediately run
- You should see your blue LEDs blink twice

Windows



Kubuntu



Note

Sketches live in folders *with the same name*

Lessons 1-3 Skills Review

At this point you should now be able to:

- Connect your ESP8266 Wemos D1 Mini board to your computer
- Have the computer recognize the board
- Identify the board serial port (Device Manager if Windows)
- Start the Arduino IDE
- Configure and install the board type in the Board Manager
- Configure and install a matching library in the Library Manager
- Select the serial port and port speed in the IDE
- Load an example sketch or your own sketch
- Validate (compile) a sketch
- Upload a sketch to your board

You can now start experimenting with different sketches from the IDE examples!

Parts of an Arduino Sketch

- All sketches have two main parts, or sections, of C++ code
- A setup section that runs once when the microcontroller is *powered up or reset.* This is why a reset (automatically) happens after a sketch is first loaded.
- A loop section which runs *continuously* after the setup section has completed
- Technically speaking, these sections are individual *functions* and are called by a master code module which the Arduino IDE elides (hides) from the user
- Setup is used to initialize variables, set pin operating modes, and other things to “set the stage” for the main program loop
- The loop is where the program steps (statements) for actual processing and events take place
- Certain special code may go outside of the setup or loop

Anatomy of a sketch

```
const byte LED_RED = 13;
const byte LED_GREEN = 12;
const byte LED_BLUE = 14;
```

Constants are one type of data defined outside the setup and loop sections

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pins as outputs.
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_BLUE, OUTPUT);
}
```

Here is the Setup section

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_RED, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_RED, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
  digitalWrite(LED_GREEN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_GREEN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
  digitalWrite(LED_BLUE, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BLUE, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

This is the Loop section

The serial port monitor

- It is possible to receive output from the microcontroller without having anything connected to it. The tool used to do this is called the Serial Port Monitor. Let's see how that works!
- Create a blank sketch (File -> New) and enter the following code:

```
Serial.begin(9600);
Serial.println("Hello world!");
```

Your resulting sketch should look like this:

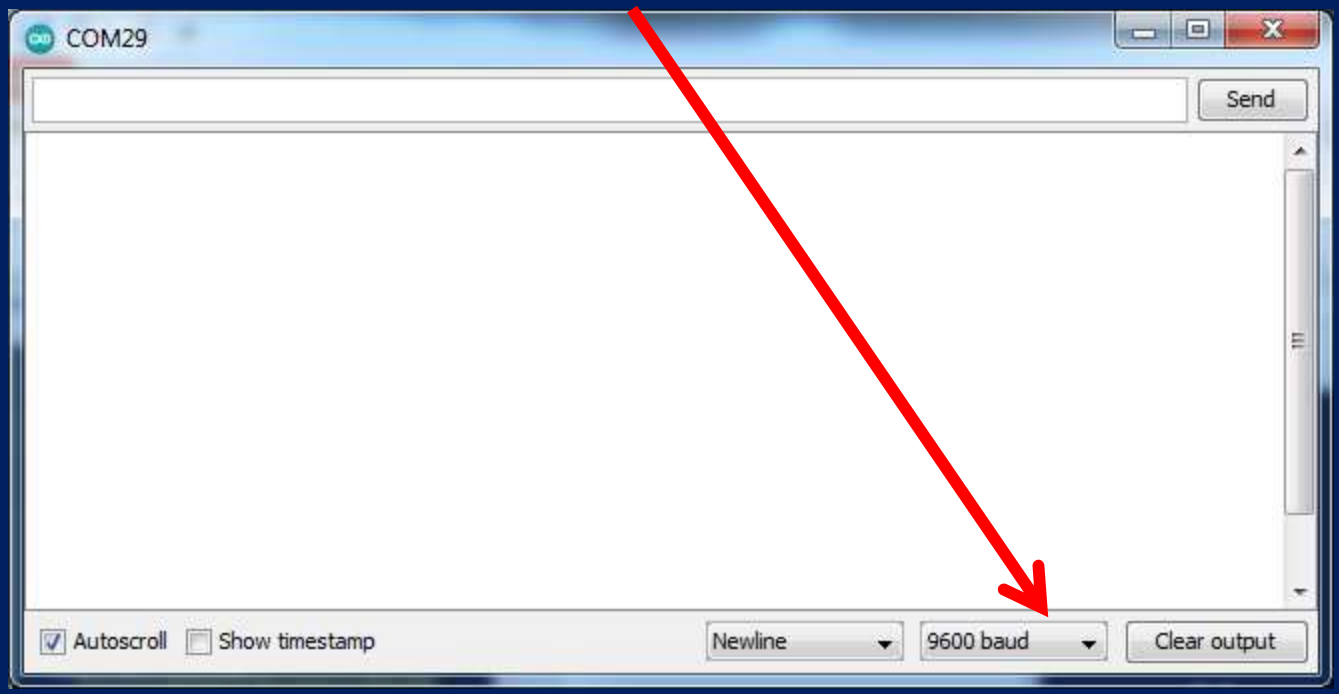
```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Hello world!");
}

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

}
```

The serial port monitor

- Now, BEFORE uploading the sketch, open the Serial Port Monitor (Tools->Serial Monitor)
- At the bottom, use the pop-up menu to set the monitor speed to 9600 baud



- Reselect the new sketch window
- NOW you can upload the sketch! (And should!)
- You will be asked to save the sketch; call it anything you like

C++ Data Types

- There are many different ways in which data is stored in the C++ language
- A data type is a particular method for organizing and storing data
- All computer data is stored as some form of binary information– all 1's and 0's. This is because the transistors that our computers are built on can only be either on (one) or off (zero).
- Data types are designed as an aid to processing and displaying particular types of data
- The simplest data type is called a *bit*.
- A bit can *only* be a one or a zero; it is a single binary digit

C++ Data Types

- The most common data type is the *byte*. A byte is composed of eight (8) bits – of eight binary digits.
- Knowing how the bits in a byte are arranged is critical to understanding computer operation; more on this shortly
- A byte can represent any number from zero (0) to 255.
- Note and understand that this means that each byte can have one of 256 possible values
- In computers and computer programming, *zero always counts as a number.* When computers count, *they always start from zero.*

C++ Data Types

- The next most common data type is the *integer*. A an integer is typically 16 bits, or two bytes, in size.
- An integer can represent any number from zero (0) to 1023.
- Note and understand that this means that each integer can have one of 1024 possible values. Zero always counts!
- The term “integer” in computer science is *not* the same thing as the term “integer” as used in mathematics.
- In computer science, an integer refers to how a number is *stored* in a computer, and not (necessarily) to what *kind* of number it is.

Variables

- Simply put, a variable is a *container* for storing data
- How much data a variable can store depends upon its *data type*.
- In C++, variables are always *declared* to be a certain data type; this is no “automatic” or “default” typing as there is some programming languages.
- The variable declaration, then, determines what type of data and how much of that type a given variable can contain
- You know that two data types are bytes and integers, so how then do we declare a variable in the Arduino IDE?

Variable Declaration

- Words that the Arduino IDE “understands” are called keywords. These keywords are how values, statements, and functions are implemented in the IDE– they are how we tell the computer what to do.
- The keyword that is used to declare a byte is– wait for it!– the keyword “byte”.
- In C++, all statements are terminated with a semicolon “;”. The semicolon tells the IDE that the statement is over. In formal CS lingo, the semicolon is a *terminator*. It serves the same function for the IDE as a period does in English for us.
- Variables can be named anything, as long as the variable name does not use keywords or terminators; the IDE will complain loudly if it does.
- The way we declare a byte, then, is like this:

  byte b;

  Yes, it’s just that simple!
- Until you have a good understanding of variable scopes, you should declare variables before the setup() and loop() sections. This makes them *global variables*, so that both the setup() and loop() sections can use them.

Variable Declaration (cont.)

- We've now declared variable "b" as a byte.
- We know that variables are containers for information, and that byte can contain any number from 0 to 255, so how do we put information (a number) *into* a byte?
- We must use an assignment operator. In C++, the assignment operator is a *single* equals sign, =.
- We can assign a value to b when it is declared:

  byte b = 22;

  This means that the variable 'b' now "contains" the number twenty-two.
- Is there a way to display the contents of a variable? Sure is!

Displaying variables

- The easy way to display the contents of a variable is to use the Serial Monitor.
- We saw that we could display text in the Serial Monitor with the `Serial.println` statement, thus:
  `Serial.println(“Hello world”)`.
- The quotation marks are mandatory to tell the Serial.println statement that it is “being given” *text* to print.
- So how do we supply a variable? Like this:

```
Serial.println(b);
```

- If we do not tell the IDE *how* we want the variable printed, it will “make its best guess” based on the data type of the variable. But we *can* tell it how we want the variable printed:

```
Serial.println(b,DEC);
```

  That statement tells the IDE to print the variable as a *decimal* value– the type of number we use.

Simple Arithmetic

- We can use the assignment operator, =, inside either the setup() or loop() sections of a sketch to assign a value to a variable at any time, thus: `b = 99;` Note that there is no “int” declaration when we do this; the variable has *already* been declared.
- Computers would be pretty useless if all we could do with them is display numbers (or text) that we had manually entered. But what computers are really good at– because it’s *actually* all they do– is math. Specifically, addition and subtraction.
- Just as there is an assignment operator for storing information in a variable, there are arithmetic operators for performing math *using variables*.
- The basic arithmetic operators are + (addtion), - (subtraction), * (multiplication), / (division), % (modulus or modulo or remainder) and ^ (exponentiation).
- So if we want to add two numbers together, we can do it this way:

```
b = 2 + 2;
```

  Which will store the result in variable b.

Formal End of Lesson 3

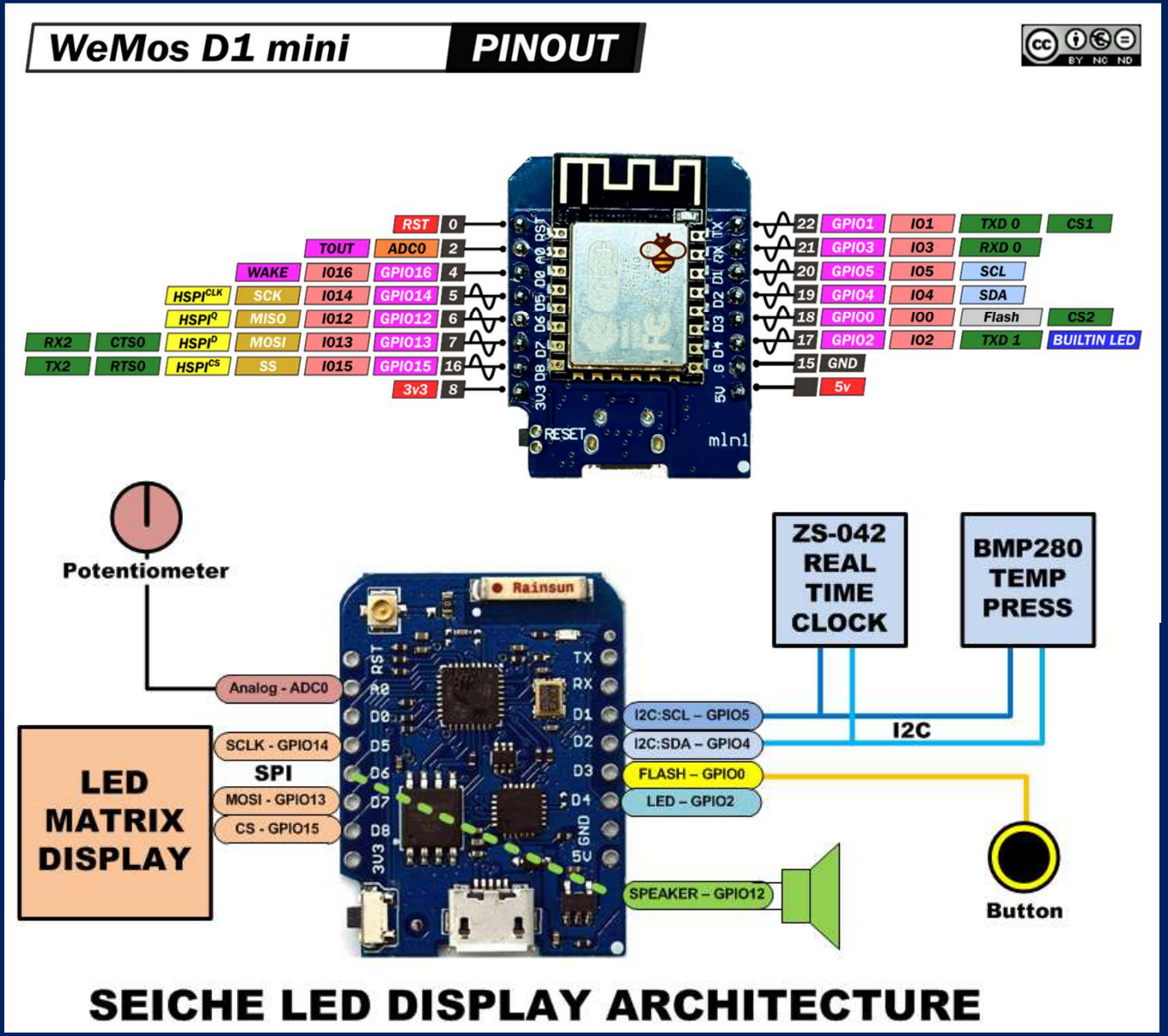
HOMEWORK!

- **Programming With Arduino – *Section 4. Arduino Programming***
- **Arduino Cookbook – *Chapter 2.1 through 2.9* (but feel free to read all of chapter 2, it's info you'll need to know)**

In next week's exciting episode

- Incrementing variables
- Arrays and strings
- Functions
- The binary number system, Part 1

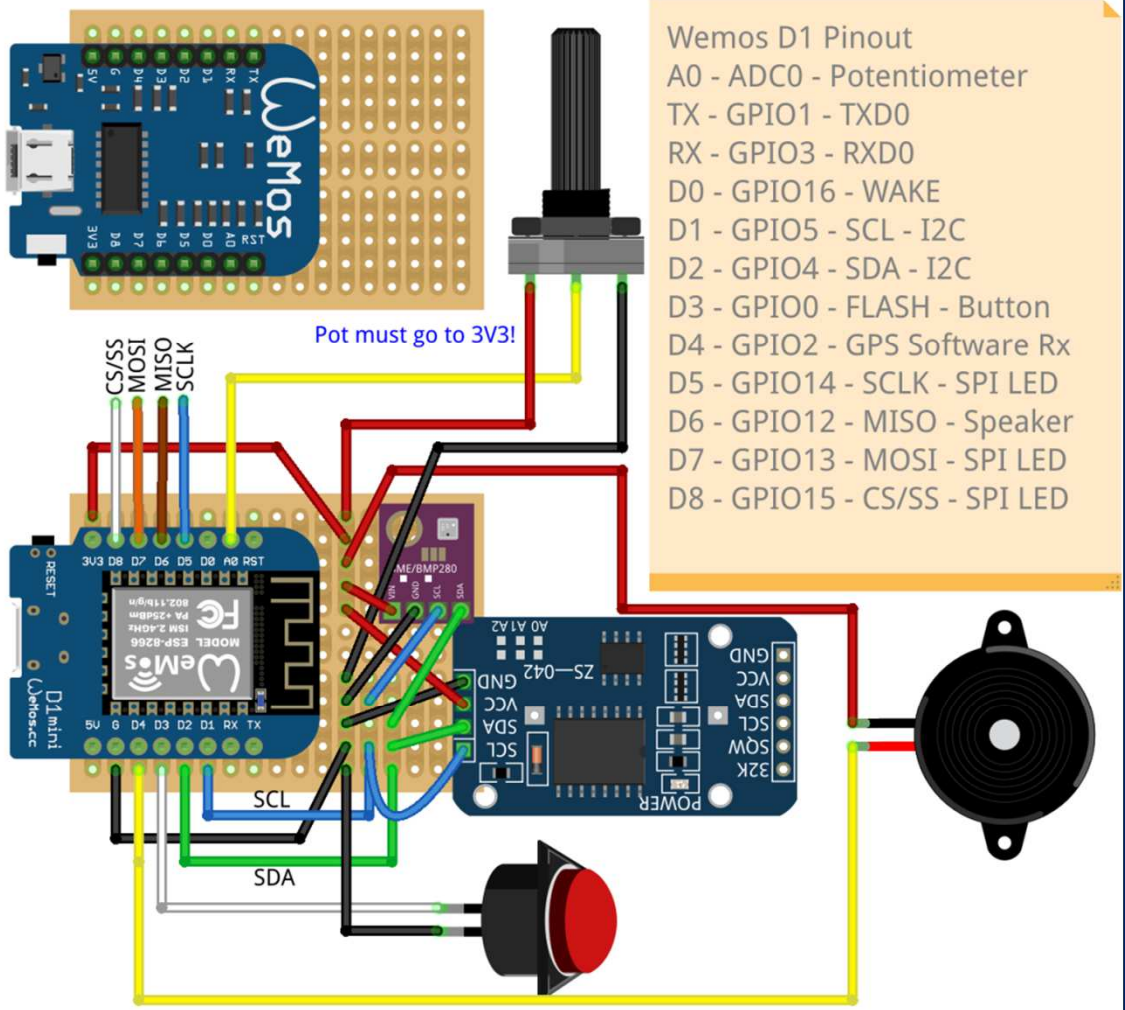
LESSON REFERENCE



LESSON REFERENCE

Pin Assignment Notes
GPIO16/D0 - HIGH at boot - No interrupt, no PWM or I2C - Unused
GPIO2/D4 - HIGH at boot - Input pulled up, output to onboard LED - - probable GPS RX software serial
GPIO12/D6 - Piezo Speaker (not used in SPI LED Matrix)
GPIO[12],13,14,15/D6,D7,D5,D8 - MISO,MOSI,SCLK,CS - SPI
GPIO4,5/D2,D1 - SDA,SCL - I2C
ADC0/A0 - Analog Input - Potentiometer 3.3V divider
GPIO0/D3 - Input pulled up - FLASH button, boot fails if pulled low - button to ground

SPI - LED matrix : 12,13,14,15
I2C - RTC,BMP280 : 4,5
Serial RX - GPS : 16 SS
Input pullup with interrupt - Button : 0
Piezo Speaker : 2
Analog Input - Potentiometer : ADC0



Wemos D1 Pinout
A0 - ADC0 - Potentiometer
TX - GPIO1 - TXD0
RX - GPIO3 - RXD0
D0 - GPIO16 - WAKE
D1 - GPIO5 - SCL - I2C
D2 - GPIO4 - SDA - I2C
D3 - GPIO0 - FLASH - Button
D4 - GPIO2 - GPS Software Rx
D5 - GPIO14 - SCLK - SPI LED
D6 - GPIO12 - MISO - Speaker
D7 - GPIO13 - MOSI - SPI LED
D8 - GPIO15 - CS/SS - SPI LED