# 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 5 – Loading Binary Images; Arrays, Characters, Strings and Loops

## Part A – Loading Binary Images

- Windows Procedure for esptool
- Linux procedure for esptool
- Using esptool to load binary images

## Part B

- Arrays
- Characters and character codes
- Strings
- Conditional Loops Part 1

# Flashing Software Binaries

- The Arduino IDE takes Processing/C++ source code and compiles it into a single binary file which is then transmitted to the microcontroller where it is stored in flash memory
- Uploading of “raw” binary executable files to a microcontroller is called “flashing”
- It is possible to create precompiled binaries and flash them without having to compile from source code or use the Arduino IDE
- This is very, very similar to installing an app on a phone (and is identical to initially installing the *operating system* onto a phone, which is done at the factory)
- We are going to learn how to do this ourselves!

# Flashing ESP8266 Software on Linux

- Exit the Arduino IDE and/or ensure it is NOT running
- Assure you have copied the following file in today's Lesson folder to your hard drive (use the Dolphin GUI file explorer; if you copied the whole Lesson 5 folder, you're good to go, just make sure you can find the file)
  `LEDMatrixDisplay-V7.1-d1_mini`
- Open a Konsole window, and enter the following command:
  `which esptool`
- If your Linux system reports a path for the tool, you can skip to the installation step
- If you do not have esptool installed, enter the following command into the Konsole window (**enter your account password when/if prompted**)
  `sudo apt install esptool`
- One you have esptool installed, enter the following commands (you'll need to interpolate the path for the directory where you placed the binary; if in doubt, just copy the binary to your Documents folder:

```
cd [directory where the binary file is]
esptool --port /dev/ttyUSB0 write_flash 0x0 ./ LEDMatrixDisplay-
V7.1-d1_mini
```

- Your ESP8266 display should restart and begin running the image you just uploaded!

# Flashing on Windows with Python3

- If you have a Windows workstation, you'll need to install Python3 and then install esptool.py
- Assure you have copied the following file in today's Lesson folder to your hard drive (use the Windows file explorer; if you copied the whole Lesson 5 folder, you're good to go, just make sure you can find the file)
  `LEDMatrixDisplay-V7.1-d1_mini`
- For Windows 10/11, run the `python-3.10.4-amd64` installer from the Software folder (copy it from your flash drive FIRST!)
- Once Python is installed, open a CMD (DOS CLI) window
- Enter the following commands into the DOS CLI prompt (you can skip the REM comments):

```
pip install esptool
REM Change to the directory you copied the WLED binaries to
cd %userprofile%\Documents\folder_with_the_binary
REM Connect your board and find the port it's on with Device Mgr
REM Then flash the bootloader first
esptool.py --port COM?? write_flash 0x0 ./LEDMatrixDisplay-V7.1-d1_mini
```

# Arrays

- An array in C++ is *a sequential series of data types accessed via an index*
- You can think of an array as working like a bunch of mailboxes at a large apartment building; each mailbox has a number corresponding to an apartment.
- In C, what types of data that can be stored in the “mailbox” are determined by the data type of the array itself.
- Like all other variables, arrays are declared, and they are declared like this:

  *data_type variablename*[*number_of_storage_locations*]
- An actual example:

```
int smallarray[5];
```

- The previous statement declares smallarray as an array of type integer having five (5) storage locations, which we call *members*.

# Working With Arrays

| Element 1 | Element 2 | Element 3 | Element 4 | Element 5 |
|---|---|---|---|---|

- The elements of an array are stored *sequentially* in memory
- The index, then, determines how “far” into the array to “reach” to access a given element. This is called an *offset*.
- Because the index is an offset, *the first element is at zero offset from the beginning of the array*.
- With all of that in mind, we reference the first element like this:

```
Serial.println(smallarray[0],DEC);
```

- **Remember that in C, zero always counts!**

# Working With Arrays (cont.)


- To access the *fifth* element of the array, we would use this syntax:

```
smallarray[4] = 99;
```

- If the concepts of offsets is confusing (and it can be!), an easier way to think of it is that the maximum index is always *one less than the size of the array*.
- This works for all elements; the index is one less than the number of the element you want to use
- Thus, the third element of the array has an index offset of two: `smallarray[2]`
- **Remember that in C, zero always counts!**

# Working With Arrays (cont.)


- We can fill an array when we declare it like this:

```
int smallarray[4] = {10, 20, 30, 40};
```

- An element of an array is the “underlying” data type making up the array. So `smallarray[3]` is an `int`.

# Character Encoding – Cyph3rpunks R001!

- You may already be familiar with Morse code, which uses combinations of dots and dashes to represent each alphabetic character and numeral
- Computer character encoding is based on historic, very simple ciphers used since ancient times to hide messages
- You may have used this simple cipher when you were young kids, based on alphabetical position: A = 1, B = 2, C=3, etc.
- Using that simple cipher, can you decode this ciphertext?

  **8 5 12 12 15 23 15 18 12 4**

- Since computers can only store ones and zeroes, we must have a way of "mapping" alphabetic characters– and *printed* numerals– into numbers the computer can store and retrieve.
- In days of yore when electric typewriters were used instead of video terminals, it was also necessary to control the typewriter mechanism and cause it to return the print carriage to the left (carriage return) and feed the paper (line feed) etc.

# Character Encoding – ASCII

- Several schemes were tried at the dawn of the information age: Morse code, Baudot code, EBCDIC, and finally ASCII
- ASCII – the American Standard Code for Information Interchange– ultimately became the cornerstone for worldwide computer character encoding.
- ASCII was originally a seven bit code– it could only encode 127 characters; this was done due to memory limitations
- ASCII was subsequently expanded to 8 bits, but the older 7-bit encodings are included in the 8-bit ASCII standard
- Very roughly, 7-bit ASCII is divided up like this:
  - 0 – 31 : Non-printing “control codes” used with physical teletype terminals and some video display terminals
  - 32-126 : Alphanumeric characters
  - 127 : The so-called “NUL” character that “does nothing”
  - 128 – 255 : So-called “Extended” characters; international symbols, “text” graphics. These characters can, and often did, vary by terminal manufacturer

# 7-Bit ASCII Character Code

## Decimal - Binary - Octal - Hex – ASCII Conversion Chart

| Decimal | Binary | Octal | Hex | ASCII | Decimal | Binary | Octal | Hex | ASCII | Decimal | Binary | Octal | Hex | ASCII | Decimal | Binary | Octal | Hex | ASCII |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00000000 | 000 | 00 | NUL | 32 | 00100000 | 040 | 20 | SP | 64 | 01000000 | 100 | 40 | @ | 96 | 01100000 | 140 | 60 | ` |
| 1 | 00000001 | 001 | 01 | SOH | 33 | 00100001 | 041 | 21 | ! | 65 | 01000001 | 101 | 41 | A | 97 | 01100001 | 141 | 61 | a |
| 2 | 00000010 | 002 | 02 | STX | 34 | 00100010 | 042 | 22 | “ | 66 | 01000010 | 102 | 42 | B | 98 | 01100010 | 142 | 62 | b |
| 3 | 00000011 | 003 | 03 | ETX | 35 | 00100011 | 043 | 23 | # | 67 | 01000011 | 103 | 43 | C | 99 | 01100011 | 143 | 63 | c |
| 4 | 00000100 | 004 | 04 | EOT | 36 | 00100100 | 044 | 24 | $ | 68 | 01000100 | 104 | 44 | D | 100 | 01100100 | 144 | 64 | d |
| 5 | 00000101 | 005 | 05 | ENQ | 37 | 00100101 | 045 | 25 | % | 69 | 01000101 | 105 | 45 | E | 101 | 01100101 | 145 | 65 | e |
| 6 | 00000110 | 006 | 06 | ACK | 38 | 00100110 | 046 | 26 | & | 70 | 01000110 | 106 | 46 | F | 102 | 01100110 | 146 | 66 | f |
| 7 | 00000111 | 007 | 07 | BEL | 39 | 00100111 | 047 | 27 | ‘ | 71 | 01000111 | 107 | 47 | G | 103 | 01100111 | 147 | 67 | g |
| 8 | 00001000 | 010 | 08 | BS | 40 | 00101000 | 050 | 28 | ( | 72 | 01001000 | 110 | 48 | H | 104 | 01101000 | 150 | 68 | h |
| 9 | 00001001 | 011 | 09 | HT | 41 | 00101001 | 051 | 29 | ) | 73 | 01001001 | 111 | 49 | I | 105 | 01101001 | 151 | 69 | i |
| 10 | 00001010 | 012 | 0A | LF | 42 | 00101010 | 052 | 2A | * | 74 | 01001010 | 112 | 4A | J | 106 | 01101010 | 152 | 6A | j |
| 11 | 00001011 | 013 | 0B | VT | 43 | 00101011 | 053 | 2B | + | 75 | 01001011 | 113 | 4B | K | 107 | 01101011 | 153 | 6B | k |
| 12 | 00001100 | 014 | 0C | FF | 44 | 00101100 | 054 | 2C | , | 76 | 01001100 | 114 | 4C | L | 108 | 01101100 | 154 | 6C | l |
| 13 | 00001101 | 015 | 0D | CR | 45 | 00101101 | 055 | 2D | - | 77 | 01001101 | 115 | 4D | M | 109 | 01101101 | 155 | 6D | m |
| 14 | 00001110 | 016 | 0E | SO | 46 | 00101110 | 056 | 2E | . | 78 | 01001110 | 116 | 4E | N | 110 | 01101110 | 156 | 6E | n |
| 15 | 00001111 | 017 | 0F | SI | 47 | 00101111 | 057 | 2F | / | 79 | 01001111 | 117 | 4F | O | 111 | 01101111 | 157 | 6F | o |
| 16 | 00010000 | 020 | 10 | DLE | 48 | 00110000 | 060 | 30 | 0 | 80 | 01010000 | 120 | 50 | P | 112 | 01110000 | 160 | 70 | p |
| 17 | 00010001 | 021 | 11 | DC1 | 49 | 00110001 | 061 | 31 | 1 | 81 | 01010001 | 121 | 51 | Q | 113 | 01110001 | 161 | 71 | q |
| 18 | 00010010 | 022 | 12 | DC2 | 50 | 00110010 | 062 | 32 | 2 | 82 | 01010010 | 122 | 52 | R | 114 | 01110010 | 162 | 72 | r |
| 19 | 00010011 | 023 | 13 | DC3 | 51 | 00110011 | 063 | 33 | 3 | 83 | 01010011 | 123 | 53 | S | 115 | 01110011 | 163 | 73 | s |
| 20 | 00010100 | 024 | 14 | DC4 | 52 | 00110100 | 064 | 34 | 4 | 84 | 01010100 | 124 | 54 | T | 116 | 01110100 | 164 | 74 | t |
| 21 | 00010101 | 025 | 15 | NAK | 53 | 00110101 | 065 | 35 | 5 | 85 | 01010101 | 125 | 55 | U | 117 | 01110101 | 165 | 75 | u |
| 22 | 00010110 | 026 | 16 | SYN | 54 | 00110110 | 066 | 36 | 6 | 86 | 01010110 | 126 | 56 | V | 118 | 01110110 | 166 | 76 | v |
| 23 | 00010111 | 027 | 17 | ETB | 55 | 00110111 | 067 | 37 | 7 | 87 | 01010111 | 127 | 57 | W | 119 | 01110111 | 167 | 77 | w |
| 24 | 00011000 | 030 | 18 | CAN | 56 | 00111000 | 070 | 38 | 8 | 88 | 01011000 | 130 | 58 | X | 120 | 01111000 | 170 | 78 | x |
| 25 | 00011001 | 031 | 19 | EM | 57 | 00111001 | 071 | 39 | 9 | 89 | 01011001 | 131 | 59 | Y | 121 | 01111001 | 171 | 79 | y |
| 26 | 00011010 | 032 | 1A | SUB | 58 | 00111010 | 072 | 3A | : | 90 | 01011010 | 132 | 5A | Z | 122 | 01111010 | 172 | 7A | z |
| 27 | 00011011 | 033 | 1B | ESC | 59 | 00111011 | 073 | 3B | ; | 91 | 01011011 | 133 | 5B | [ | 123 | 01111011 | 173 | 7B | { |
| 28 | 00011100 | 034 | 1C | FS | 60 | 00111100 | 074 | 3C | < | 92 | 01011100 | 134 | 5C | \ | 124 | 01111100 | 174 | 7C | \| |
| 29 | 00011101 | 035 | 1D | GS | 61 | 00111101 | 075 | 3D | = | 93 | 01011101 | 135 | 5D | ] | 125 | 01111101 | 175 | 7D | } |
| 30 | 00011110 | 036 | 1E | RS | 62 | 00111110 | 076 | 3E | > | 94 | 01011110 | 136 | 5E | ^ | 126 | 01111110 | 176 | 7E | ~ |
| 31 | 00011111 | 037 | 1F | US | 63 | 00111111 | 077 | 3F | ? | 95 | 01011111 | 137 | 5F | _ | 127 | 01111111 | 177 | 7F | DEL |

# 8-Bit ASCII Character Code

## ASCII control characters

| | | |
|---|---|---|
| 00 | NULL | (Null character) |
| 01 | SOH | (Start of Header) |
| 02 | STX | (Start of Text) |
| 03 | ETX | (End of Text) |
| 04 | EOT | (End of Trans.) |
| 05 | ENQ | (Enquiry) |
| 06 | ACK | (Acknowledgement) |
| 07 | BEL | (Bell) |
| 08 | BS | (Backspace) |
| 09 | HT | (Horizontal Tab) |
| 10 | LF | (Line feed) |
| 11 | VT | (Vertical Tab) |
| 12 | FF | (Form feed) |
| 13 | CR | (Carriage return) |
| 14 | SO | (Shift Out) |
| 15 | SI | (Shift In) |
| 16 | DLE | (Data link escape) |
| 17 | DC1 | (Device control 1) |
| 18 | DC2 | (Device control 2) |
| 19 | DC3 | (Device control 3) |
| 20 | DC4 | (Device control 4) |
| 21 | NAK | (Negative acknowl.) |
| 22 | SYN | (Synchronous idle) |
| 23 | ETB | (End of trans. block) |
| 24 | CAN | (Cancel) |
| 25 | EM | (End of medium) |
| 26 | SUB | (Substitute) |
| 27 | ESC | (Escape) |
| 28 | FS | (File separator) |
| 29 | GS | (Group separator) |
| 30 | RS | (Record separator) |
| 31 | US | (Unit separator) |
| 127 | DEL | (Delete) |

## ASCII printable characters

| | | | | | |
|---|---|---|---|---|---|
| 32 | space | 64 | @ | 96 | ` |
| 33 | ! | 65 | A | 97 | a |
| 34 | " | 66 | B | 98 | b |
| 35 | # | 67 | C | 99 | c |
| 36 | $ | 68 | D | 100 | d |
| 37 | % | 69 | E | 101 | e |
| 38 | & | 70 | F | 102 | f |
| 39 | ' | 71 | G | 103 | g |
| 40 | ( | 72 | H | 104 | h |
| 41 | ) | 73 | I | 105 | i |
| 42 | * | 74 | J | 106 | j |
| 43 | + | 75 | K | 107 | k |
| 44 | , | 76 | L | 108 | l |
| 45 | - | 77 | M | 109 | m |
| 46 | . | 78 | N | 110 | n |
| 47 | / | 79 | O | 111 | o |
| 48 | 0 | 80 | P | 112 | p |
| 49 | 1 | 81 | Q | 113 | q |
| 50 | 2 | 82 | R | 114 | r |
| 51 | 3 | 83 | S | 115 | s |
| 52 | 4 | 84 | T | 116 | t |
| 53 | 5 | 85 | U | 117 | u |
| 54 | 6 | 86 | V | 118 | v |
| 55 | 7 | 87 | W | 119 | w |
| 56 | 8 | 88 | X | 120 | x |
| 57 | 9 | 89 | Y | 121 | y |
| 58 | : | 90 | Z | 122 | z |
| 59 | ; | 91 | [ | 123 | { |
| 60 | < | 92 | \ | 124 | \| |
| 61 | = | 93 | ] | 125 | } |
| 62 | > | 94 | ^ | 126 | ~ |
| 63 | ? | 95 | _ | | |

## Extended ASCII characters

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 128 | Ç | 160 | á | 192 | └ | 224 | Ó |
| 129 | ü | 161 | í | 193 | ┴ | 225 | ß |
| 130 | é | 162 | ó | 194 | ┬ | 226 | Ô |
| 131 | â | 163 | ú | 195 | ├ | 227 | Ò |
| 132 | ä | 164 | ñ | 196 | ─ | 228 | õ |
| 133 | à | 165 | Ñ | 197 | ┼ | 229 | Õ |
| 134 | å | 166 | ª | 198 | ã | 230 | µ |
| 135 | ç | 167 | º | 199 | Ã | 231 | þ |
| 136 | ê | 168 | ¿ | 200 | ╚ | 232 | Þ |
| 137 | ë | 169 | ® | 201 | ╔ | 233 | Ú |
| 138 | è | 170 | ¬ | 202 | ╩ | 234 | Û |
| 139 | ï | 171 | ½ | 203 | ╦ | 235 | Ù |
| 140 | î | 172 | ¼ | 204 | ╠ | 236 | ý |
| 141 | ì | 173 | ¡ | 205 | ═ | 237 | Ý |
| 142 | Ä | 174 | « | 206 | ╬ | 238 | ¯ |
| 143 | Å | 175 | » | 207 | ¤ | 239 | ´ |
| 144 | É | 176 | ░ | 208 | ð | 240 | ≡ |
| 145 | æ | 177 | ▒ | 209 | Ð | 241 | ± |
| 146 | Æ | 178 | ▓ | 210 | Ê | 242 | ‗ |
| 147 | ô | 179 | │ | 211 | Ë | 243 | ¾ |
| 148 | ö | 180 | ┤ | 212 | È | 244 | ¶ |
| 149 | ò | 181 | Á | 213 | ı | 245 | § |
| 150 | û | 182 | Â | 214 | Í | 246 | ÷ |
| 151 | ù | 183 | À | 215 | Î | 247 | ¸ |
| 152 | ÿ | 184 | © | 216 | Ï | 248 | ° |
| 153 | Ö | 185 | ╣ | 217 | ┘ | 249 | ¨ |
| 154 | Ü | 186 | ║ | 218 | ┌ | 250 | · |
| 155 | ø | 187 | ╗ | 219 | █ | 251 | ¹ |
| 156 | £ | 188 | ╝ | 220 | ▄ | 252 | ³ |
| 157 | Ø | 189 | ¢ | 221 | ¦ | 253 | ² |
| 158 | × | 190 | ¥ | 222 | Ì | 254 | ■ |
| 159 | ƒ | 191 | ┐ | 223 | ▀ | 255 | nbsp |

# C and ASCII

- Why do we care about ASCII? Because it's how most computers store basic character data (there are more sophisticated character encodings, like UTF-8– we will study those next semester.)
- You may have noticed that any of the numbers for the 255 characters of the extended 8-bit ASCII character set will "fit" in a single byte. This is not a coincidence.
- Thus: A single byte of data can hold the number equivalent of any one of the 255 8-bit ASCII characters
- The data type used to hold character data in C is `char`
- Let's see how that works!

# Class Sketch 5A

- Find and open Sketch5A.ino in your IDE

```
char aletter;

void setup()
{
  // Setup port for serial monitor
  Serial.begin(9600);
}

This time around, we will print the ASCII character set!
void loop()
{
  Serial.println(aletter, DEC);
  Serial.println(aletter,DEC);
  Serial.println();

  aletter++; // Wait, what?!

// Leave delay at 500 or higher to have output be readable
  delay(500);
}
```

**Describe your results. What is happening, *and why*?**

# Strings

- In computer science, sequential groups of alphanumeric data are called “strings”. (Can’t call ‘em “words”, that’s another data type, right?)
- Strings can contain numerals, letters, *or any other valid ASCII character*
- In C, strings are implemented as *arrays* of type `char`. They are declared like this:

  ```
  char   somestring[10].
  ```

- **Note that a string can only contain the number of characters it has been declared for!**
- There are two primary types of strings:
  - Raw ASCII data
  - Null-terminated

# Strings (cont.)

- Raw ASCII data strings are filled with whatever, and it's up to the programmer to know when the string "ends". There is no way to automagically "read through" the characters of a string.
- Because it is hugely convenient to let the computer worry about pulling the individual characters out of a string array, the convention was developed to use the NULL character, ASCII 0, to terminate a string, that is, be the *last* character.
- It's the programmer's job to ensure that there is a NULL terminating a string. This is a requirement for certain function calls and object methods, such as Serial.println(somestring);
- Best practice is to add the NULL "as you go"
- Arduino C++ offers a String datatype that is unique to the Arduino environment. It is very memory and processor intensive, thus not used often by developers, and is not portable to most C++ compilers, so is covered in this class (see "Choosing between Arduino Strings and C character arrays" in the Arduino Cookbook.) You are welcome to use it in your own code, however, and the Cookbook has good instructions on using it.

# Relational Operators Redux

- Last lesson we saw that the relational operators in C++ are:
  - Logical equals; this is done with a double equals sign: ==
  - Logical not equal; this is a negated equals sign: !=
  - Greater than; this is the (wait for it) greater-than sign: >
  - Less than; this is the (no surprise) less-than sign: <
  - Equal to or greater than: =>
  - Less than or equal to: <=
  - And, as you've already seen, a simple negation, !
- This lesson, we are introducing two more. These are used to *compare* two logical expressions *together*:
  - && Double ampersand is relational "and". Evaluates to true if both expressions are true

    (foo < 30 && bar > 500)
  - || Double "bar" or "pipe" is the relational "or". Evaluates to true if either expression is true

    (mph == 88 || c == 300,000)

# Conditional Loops Part 1

- You are already familiar with the Loop() function in the IDE. This is an example of an endless or *infinite* loop, because it repeats indefinitely
- Much more useful are conditional loops. Conditional loops execute until a logical conditional is (or isn't) satisfied.
- Today we will learn about the while loop, whose syntax is:

```
while (conditional expr)
        statement;
```

- `when` is usually used with a code block

```
while (mph < 88)
{
        Serial.println(a, DEC);
        mph++
}
```

# While – Let's Give It A Go

Let's modify our sketch to use a while loop to only print the first 127 ASCII characters.

```
char aletter;
void setup()
{
  // Setup port for serial monitor
  Serial.begin(9600);
}

This time around, we will print the ASCII character set!
void loop()
{
  aletter = 0;
  while(aletter < 128)
  {
    Serial.println(aletter, DEC);
    Serial.println(aletter,DEC);
    Serial.println();
    aletter++;
    delay(500);
  }
}
```

Describe your results. What is happening, and why?

# Formal End of Lesson 5

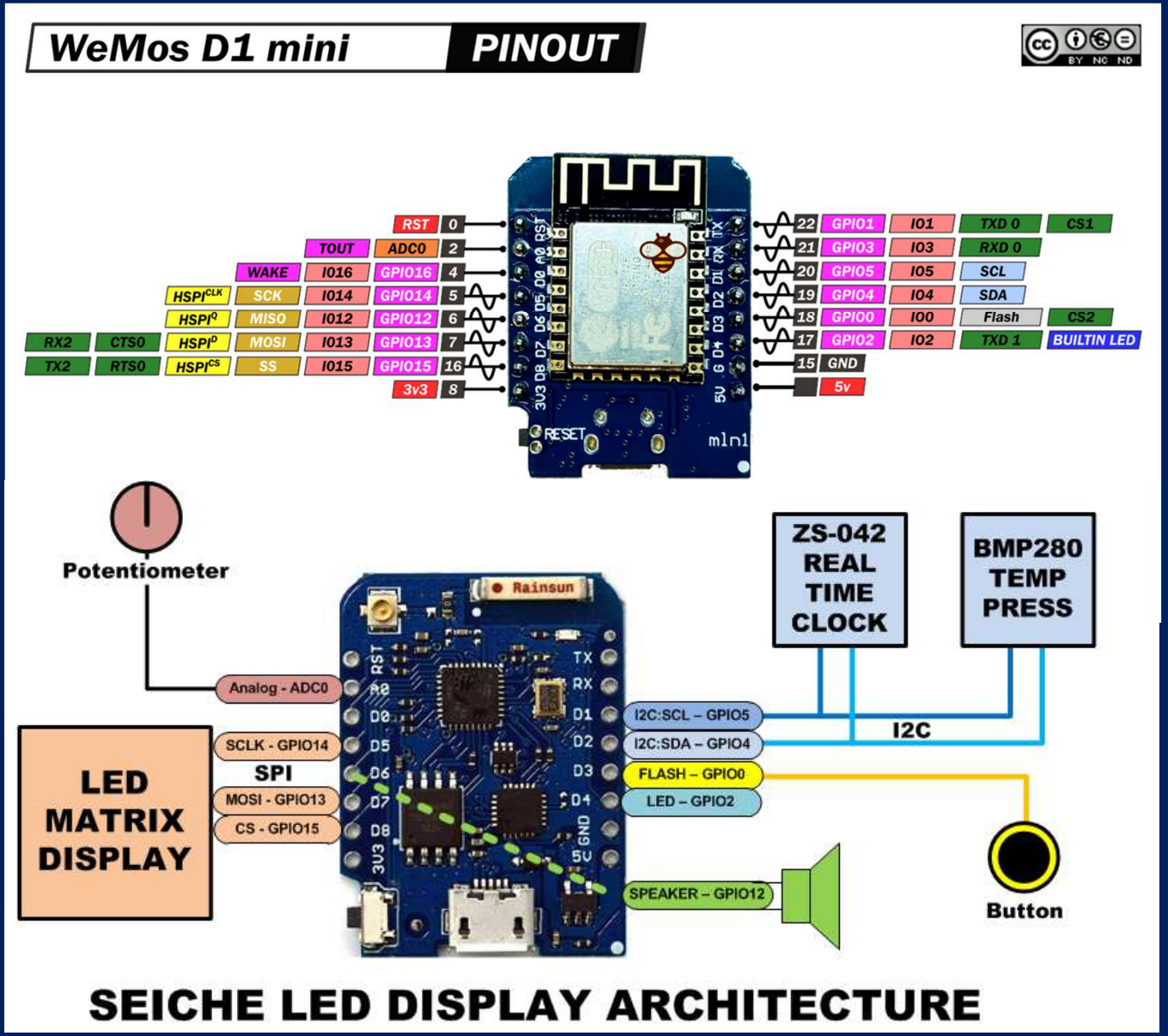
**HOMEWORK!**

- **Arduino Cookbook – *Make sure you have read ALL of Chapter 2 and Chapter 3***
- **A programming assignment! Yay!**
  **Take your homework sketch from last week and modify it to print the contents of an integer array of 99 elements, starting from the first element to the last, then from the last to the first. Use while instead of if-then. You'll execute the sketch next week.**
  **Yeah, it's another desk-checked exercise– although feel free to use a GNU C compiler locally if you just can't wait. ☺**

**In next week's exciting episode**

- For-Next Loops
- Nested loops
- Functions and parameters
- Nested for-next loops
- The binary number system (still assembling the slide deck!)

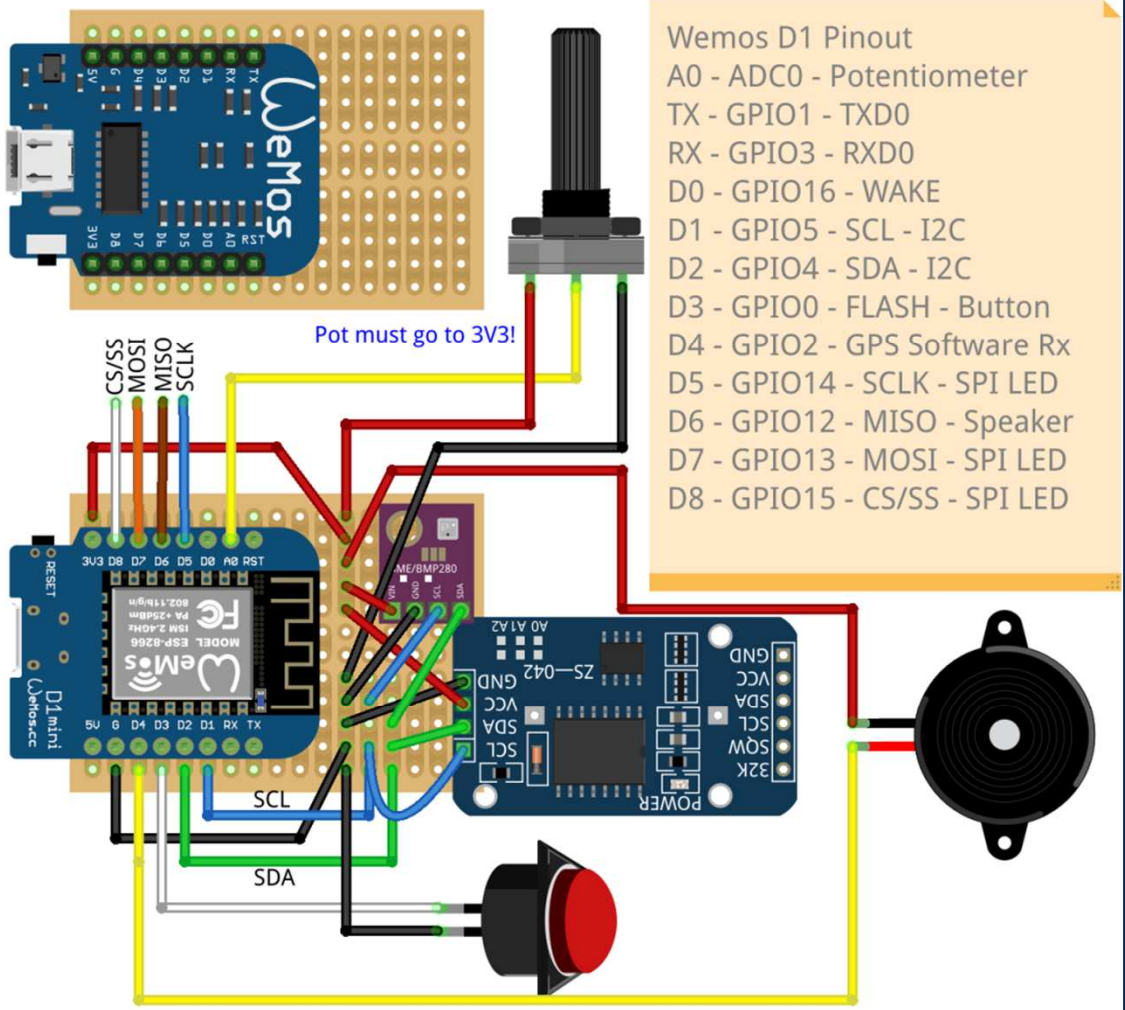
# LESSON REFERENCE



# 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