



SEICHE 2023

Intermediate Arduino Programming for IoT

Instructor: Paul Frommeyer

www.paulfrommeyer.com

Corporate Sponsor: DXC Technology



REMINDER

**YOU MUST HAVE A
WORKING LED MATRIX
DISPLAY TO TAKE THIS
CLASS!**

**If displays are lost or damaged,
replacements are \$70 with 1-week
lead time for replacement**

Lesson Plan Overview

Lesson 1: 7Feb23 – Review, Addressing, and Pointers

- Overview of lesson plan
- Class Exercise: Validation of sketch uploading
- Class Exercise: Validation of binary create and upload
- More on the Preprocessor
- Memory Organization and Variables
- Review of addressing and pointers
- Using pointers to variables
- Declaring pointers in function calls
- Call-by-value Function Calls
- Call-by-reference Function Calls

Lesson 2: 14Feb23 – More Gory Details

- Class Exercise: Call-by-reference variables
- Introduction to Complex Data Structures
- Using complex data structures
- Storage declarations and Arduino

Lesson 3: 21Feb23 – Fonts Redux

- MD_Parola library font usage review
- The MD_Parola font format
- Designing your own fonts
- Declaring your own fonts
- Using your own fonts with MD_Parola

Lesson 4: 28Feb23 – Intro to Visual Studio Code

- Introduction to VSC
- <https://code.visualstudio.com/docs/introvideos/overview>
- VSCode Installation
- VSCode Integration with Arduino and ESP8266

Lesson 5: 7Mar23 – Filesystems

- Introduction to mass storage filesystems
- The SD FAT, FAT16, and FAT32 filesystems
- The exFAT filesystem
- Introduction to SPIFFS
- Class Exercise: Using SPIFFS

Lesson 6: 21Mar23 – WiFi

- Review of WiFi technology (but no gory details)
- Class Exercise: Review of WiFi access with WiFi Manager
- Web server and client HTTP architecture
- Introduction to HTML
- Creating a basic ESP8266 web server

Lesson 7: 28Mar23 – Web Technology

- More on HTML tags
- HTTP and Mime Types
- Using SPIFFS
- Class Exercise: Display A Web Page

Lesson 8: 4Apr23 – Doing More With Web Servers

- HTML Form Data
- Asynchronous Web Server Installation
- Class Exercise: Displaying Sensor Data With Async Web Server
- Class Exercise: Retrieving Form Data with Async Web Server

Lesson 9: 11Apr23 – Internet Access

- Class Exercise: Displaying form data with MD_Parola
- Obtaining information from an Internet web page
- Obtaining weather information from the Internet
- Class Exercise: Displaying weather information

Lesson 9: 11Apr23 – IoT Messaging and MQTT

- Introduction to IoT network messaging and MQTT
- Subscribing to an MQTT service
- Publishing to an MQTT service
- Class Exercise: Using MQTT and WiFi

Lesson 10: 18Apr23 – Version control and Git

- The need for document version control
- The Git version control system
- Installing Git
- Using Git
- Introduction to Github
- Using Github
- Class Exercise: Signing up for a Github account

Flash Drive Contents Reminder

- When I update parts of your flash drives, you can either *recopy the entire section*, or *just copy the updated part*.
- However, you will need to explicitly copy any new files so that they are locally accessible on your laptops
- Copying just *lesson* files will— wait for it!— *copy only the lesson files*.

IntermediateProgramming-Software

ESP8266FS-0.5.0.zip

Make certain you have copied everything in RED!
You should already have previously copied everything else!

FLASH DRIVE

- Archive-BasicProgrammingWith Arduino-Fall2022
- Archive-IntroductionToArduino-Spring2022
- IntermediateProgramming-Software
- IntermediateProgramming-Lessons
- IntermediateProgramming-Documentation

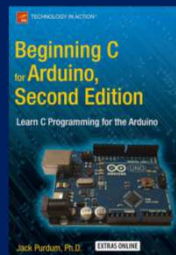
IntermediateProgramming-Lessons

- IntermediateProgramming-Lesson1
- IntermediateProgramming-Lesson2
- IntermediateProgramming-Lesson3
- **IntermediateProgramming-Lesson8**

IntermediateProgramming-Documentation

- ArduinoReference
 - **Beginning C for Arduino, 2nd Edition**
- BoardsGuides
- D1 Mini Reference
- ElectronicsReference
- IoT Reference
- LED Matrix Display Architecture
- **HackSpace Magazine**

Can't do reading homework without the files!



Lesson 8 – More With Web Servers

Part A

- **HTML Forms**
- **Synchronous vs Asynchronous Code**

Part B

- **Async Webserver Installation**
- **Class Exercise: Displaying Sensor Data On Web Page**
- **Class Exercise: GETting Form Data from Web Pages**

HTML Form Structure

- HTML has input elements displayed in different ways such as input field, checkbox (for selecting zero or more of multiple choices), radio buttons (for selecting one of multiple choices), submit button etc.
- The basic structure of a form consists of input fields and a submit button.
- The user fills out the input fields with the required information and upon clicking the submit button the data is sent to a form handler.
- Typically, the form *handler* is a file on the server with a script for processing input data.



```
<form action="home.html">  
  First Name:<br>  
  <input type="text" name="first_name">  
</br>  
  Last Name:<br>  
  <input type="text" name="last_name">  
</br>  
  Email:<br>  
  <input type="text" name="email">  
</br>  
  <input type="submit" name="Submit">  
  
</form>
```

First Name:

Last Name:

Email:

Submit Query

HTML Form Actions

- You add an action attribute to the form to define where the submitted data goes. In the example above the submitted information will be handled by the script of the home.html document.
- In this case, the URL is called relative. Relative URLs are compared to the current URL that is loaded in the Web browser. We can use slashes and the “double dot” notation to address a different folder or the parent folder of the virtual folder structure on the Web server.
- Full URLs are used to submit the form data to completely different Web site. For example, a Web site may embed an HTML form for newsletter subscription which submits its form fields to an external Web site, which provides email newsletter services.

HTML Form Methods

- In this example, an HTTP method attribute has been added to the form. Here again, the method can be either GET or POST. Both methods are used to transfer data from *client* to *server*.
- The GET method transfers data *in the URL* with a query string. Therefore, the length of the URL is limited. GET is preferable for images, word documents or data that does not require any security.
- POST is an HTTP method that encodes form data in a specified format (using MIME types, natch) and sends it to the server via the HTTP message body. The World Wide Web frequently uses POST to send user-generated data or an uploaded file to the web server.
- In the example below, you can see the standard URL encoding used to encode the HTML form fields and URLs. The URL encoding is a long string of name and value pairs. Each pair is separated from one another by an ampersand (&) sign and each name is separated from the value by an equals (=) sign. For example: key1=value1&key2=value2.
- This encoding can be used for text and other data fields, but it does not support file upload fields. We can overcome this limitation by switching to multipart encoding.

SoftUni Global

```
<form method="post">
Name: <input type="text" name="name"/> <br/>
Age: <input type="text" name="age"/> <br/>
<input type="submit" />
</form>
```

POST /index.html HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 23

URL-encoded form data

name=Maria+Smith&age=19

index.html

Name: Maria Smith
Age: 19
Submit

Page HTML

HTTP POST data stream

HTTP GET in URL

Differences Between GET and POST

If you want to send one or two simple variables (for example search parameters) to your server, then you use GET. However, if your form includes passwords, credit card information, or any other data that needs extra protection then POST is a better choice.



GET

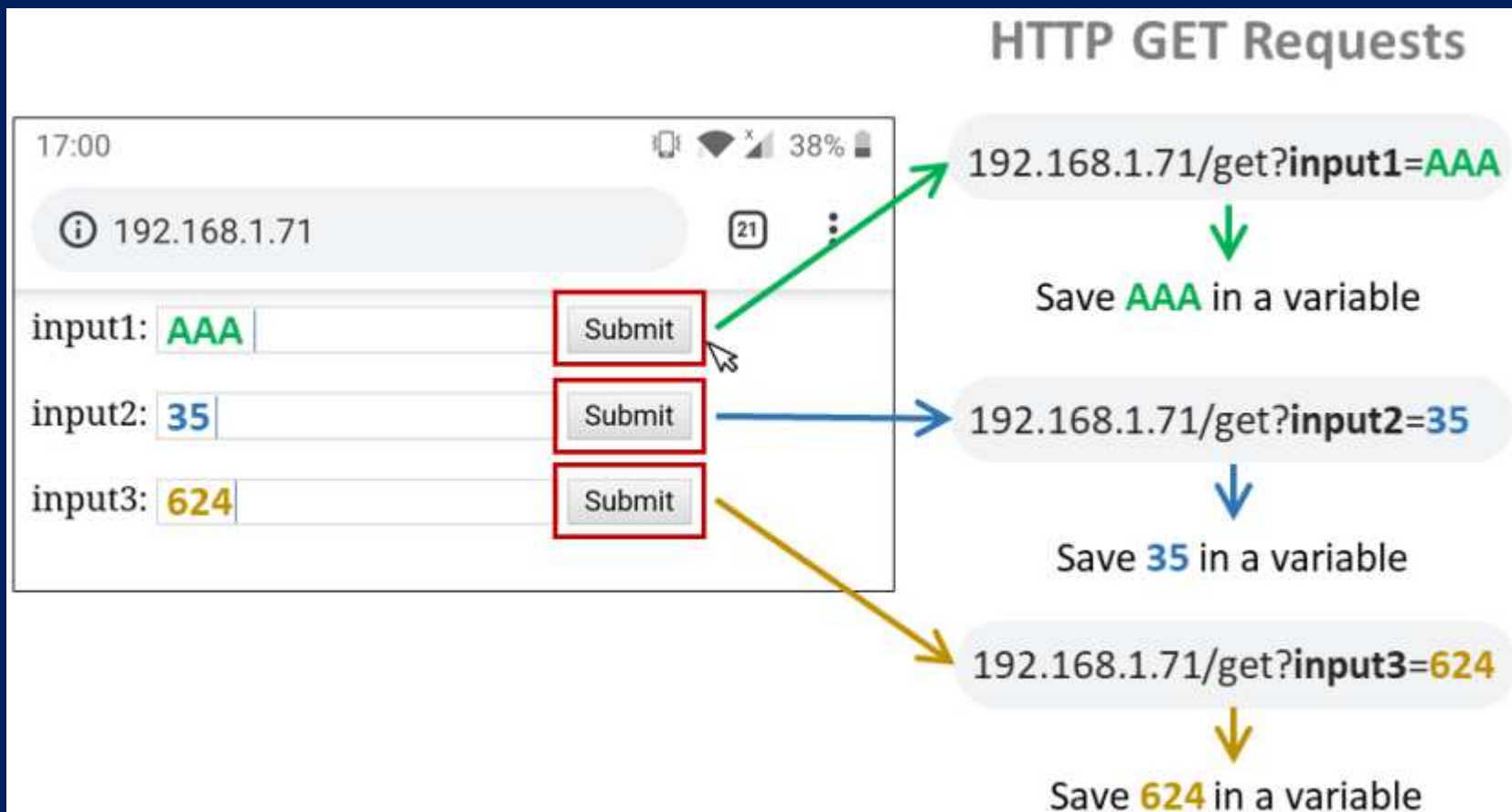
- Values are contained in the URL;
- Has a length limitation of 255 characters;
- It is often cacheable;
- Supports only string data types;
- Parameters are saved in browser history;
- Results can be bookmarked.

POST

- Values are contained in the message's body;
- Does not have a length limitation ;
- It is hardly cacheable;
- Supports different data types, such as string, numeric, binary, etc;
- Parameters are not saved in browser history;
- Results cannot be bookmarked.

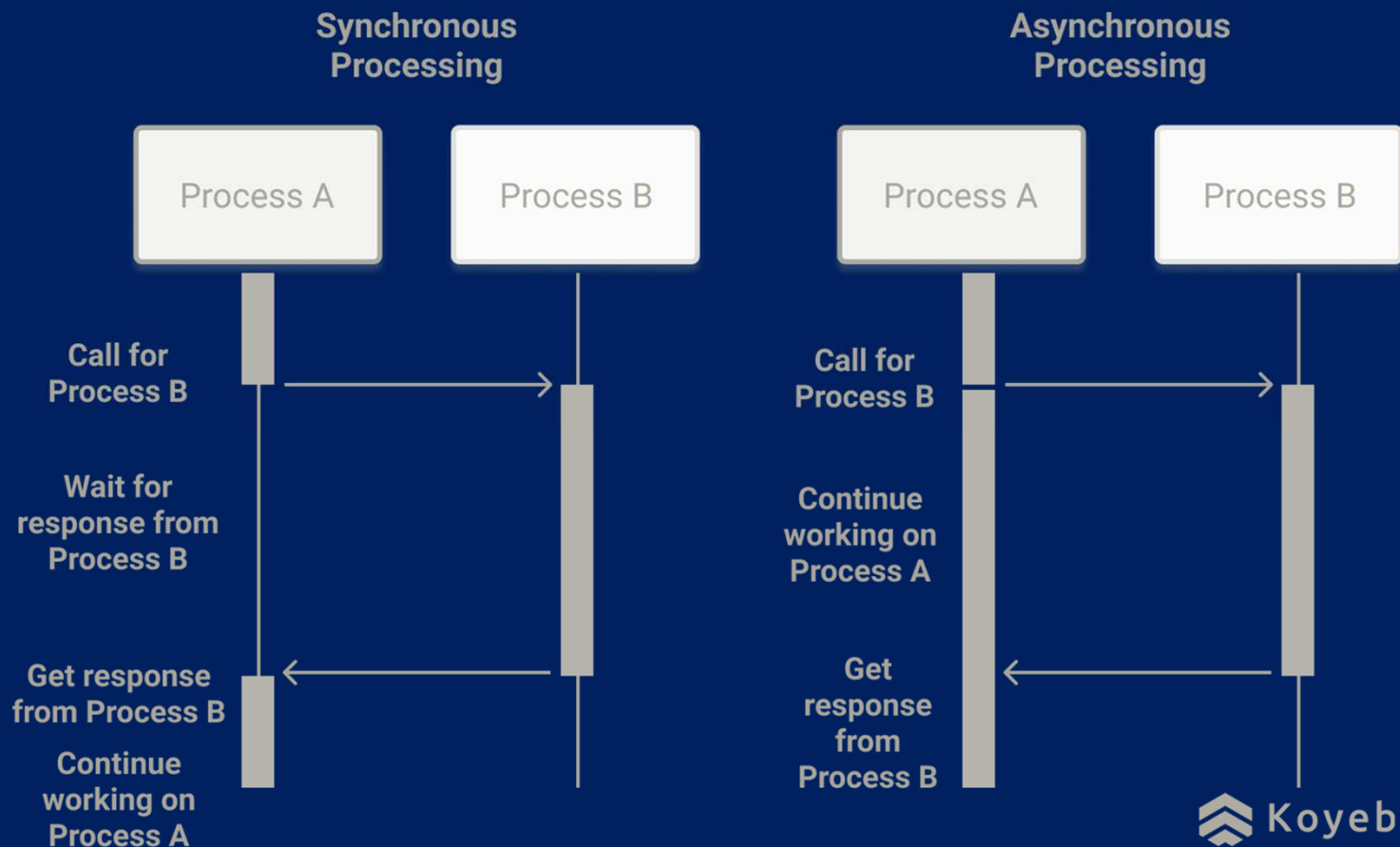
HTML GET Forms

- As mentioned previously, the GET HTTP data transfer can be used to send *form data* from a client browser to a server
- This is done by *including* the data *within* a URL.



Synchronous vs Asynchronous

- In synchronous processing, one action must wait for another
- In asynchronous processing, actions can proceed in parallel



Async Web Server Installation

- The ESP8266 Asynchronous Web Server is faster and more efficient, and nearly all on-line projects use it instead of the single-threaded (synchronous) server included by default in the ESP8266 wifi library
- So, we will need to install the necessary libraries for the Async server before we can use it. **However, this time we must perform the installation manually!**
- Open your SKETCH8A folder in your file browser (Explorer or Dolphin)
- Copy the **AsyncTCP** and **ESPAsyncWebServer** folders
- Next, navigate to *your own* Arduino folder in your home directory:
 - C:\Users\YourUserName\Documents\Arduino [Windows]
 - /home/seiche/Arduino [Kubuntu Linux]
- Then open the **libraries** folder
- Now paste the two folders that you previously copied (to the file explorer clipboard) into the **libraries** folder

Class Exercise 8A: Display Sensor Data

- Go ahead and load SKETCH8A, **but do not upload it!! First**, upload the SPIFFS data. THEN send the sketch to your displays. Open the serial monitor to obtain the IP address assigned by the wifi, then surf to that IP in your browser.
- Remember, you probably cannot do a SPIFFS upload with Serial Monitor open
- Note: I already adapted the sketch for our BMP180 sensors, which do not provide humidity readings

WELL-COME TO SEICHE HTML TUTORIAL

ESP8266(NodeMCU)-BMP180 Sensor on Web Server

LED state ON



Temperature 112.60 °C

Humidity 9999.99 %

Pressure 897.02 hPa

You should see a page similar to that on the right, showing the various retrieved sensor values and their derived values.

BMP180 has not humidity return value; it's coded to all 9's in the sketch

Also, your page may or may not control your LED depending on your browser, as this example includes JavaScript

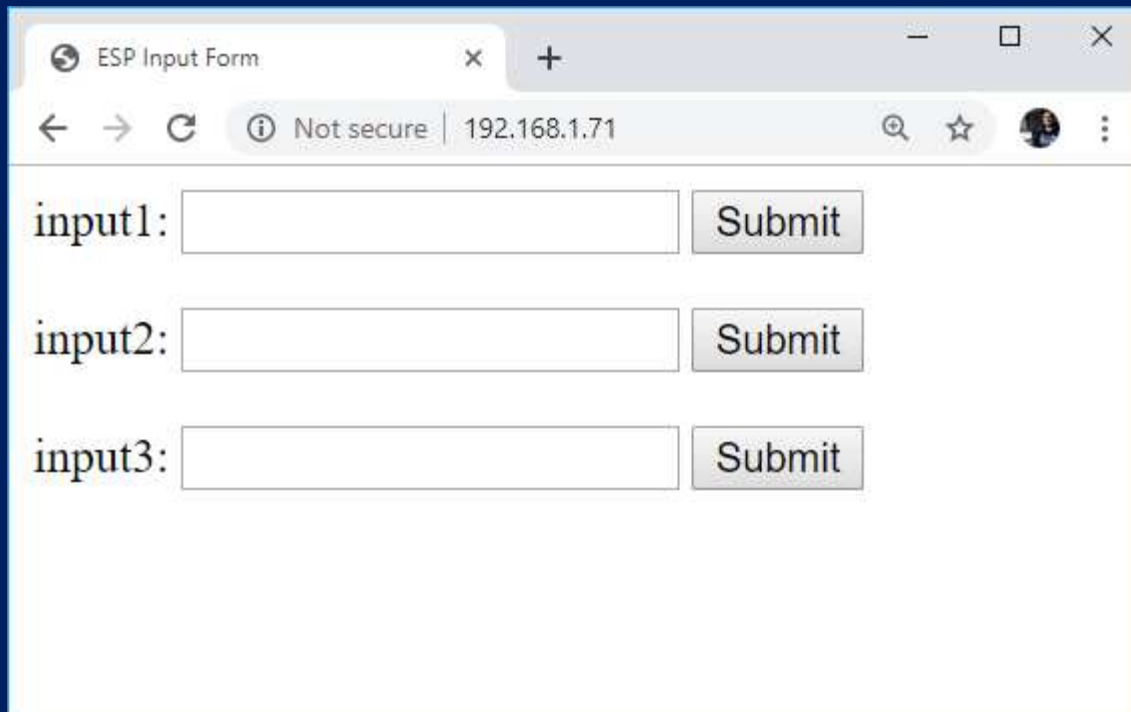
More gory details here: <https://randomnerdtutorials.com/esp8266-bme280-arduino-ide/>

Class Exercise 8B: Get Webserver Input

- Go ahead and load SKETCH8B, and send it to your displays. Use the serial monitor to obtain the IP address assigned by the wifi, then surf to that IP in your browser

```
<form action="/get"> input1:  
<input type="text" name="input1">  
<input type="submit"  
value="Submit"> </form>
```

At left is the HTML used to generate the first form field and its button widget.
Note the action method, input type, and form type



The screenshot shows a web browser window with the title "ESP Input Form". The address bar shows "Not secure" and the IP address "192.168.1.71". The page content consists of three vertically stacked form elements. Each element has a label ("input1:", "input2:", "input3:") followed by a text input field and a "Submit" button.

You should see a page similar to that on the right, with three input fields and three *separate* submit buttons

Observe the SerialMonitor when you enter text into a field and then click the corresponding "Submit" button widget

More gory details here: <https://randomnerdtutorials.com/esp32-esp8266-input-data-html-form/>

Formal End of Lesson 5

HOMEWORK

- **No Homework This Week**

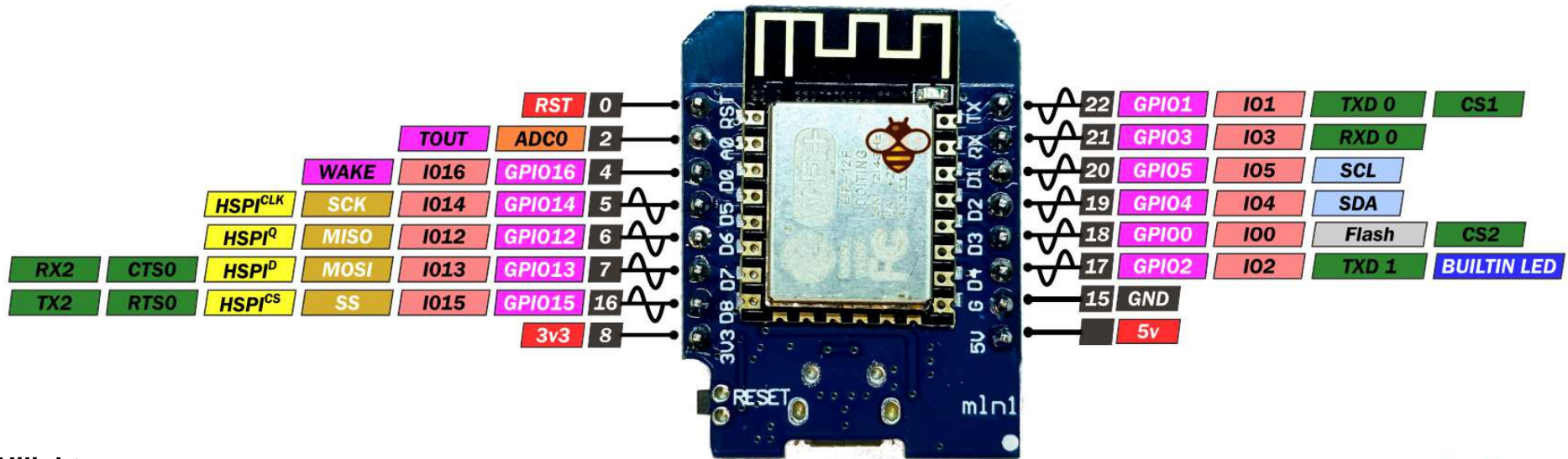
In next week's exciting episode

- Review of WiFi technology (but no gory details)
- Class Exercise: Review of WiFi access with WiFi Manager
- Uploading a web page using SPIFFS
- Class Exercise: Create a basic web server

Our Microcontroller: The WeMos D1 Mini

WeMos D1 mini

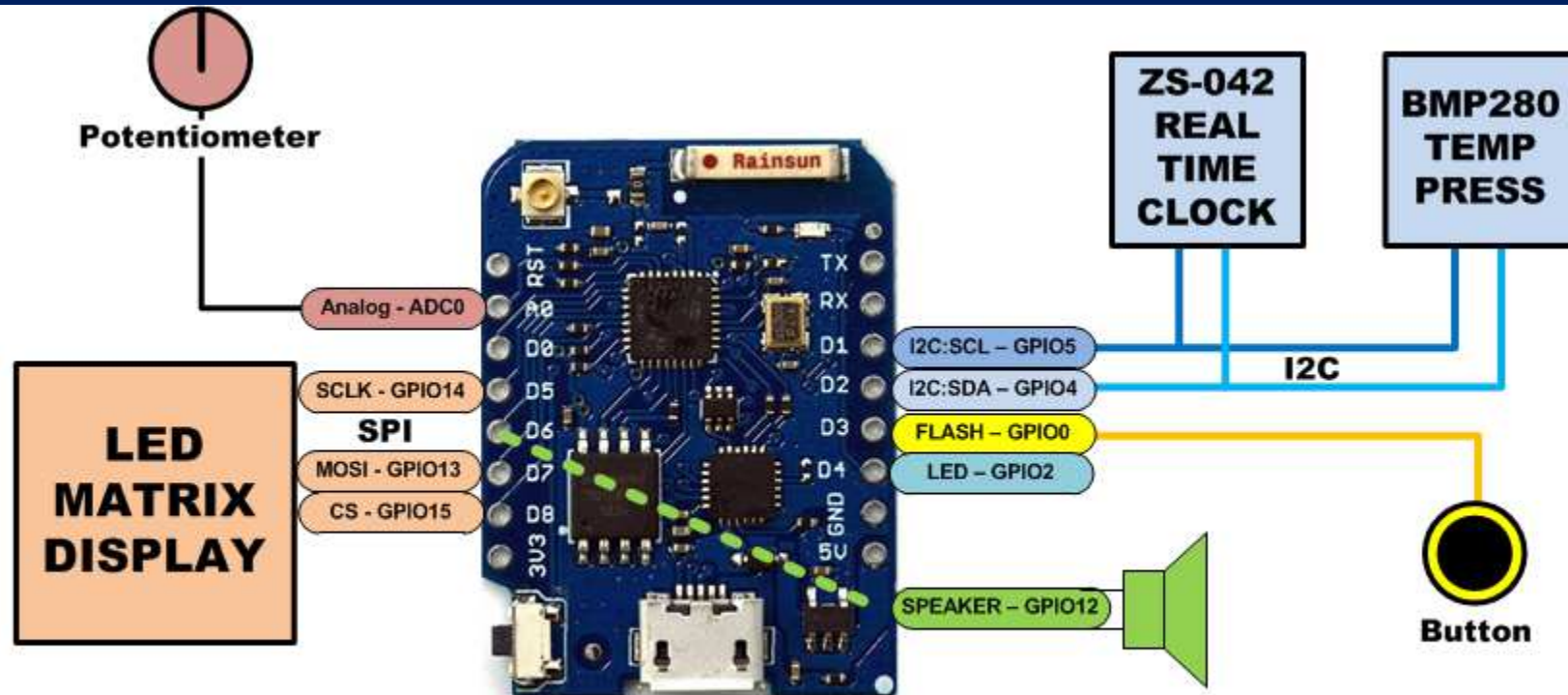
PINOUT



Highlights

- 11 digital IO: all are interrupt and pwm capable (except D0/GPIO16)
- 1 analog input (3.2V max input): A0/ADC0
- Micro USB or Type-C USB Port (clones usually have micro USB)
- Two SPI interfaces (one is used for on-board flash memory), one I2C interface, two serial ports
- Built-in WiFi (client or standalone access point modes) and Bluetooth
- Compatible with MicroPython, Arduino, NodeMCU
- Uses the CH340 USB-to-serial driver (installation usually needed on Windows)
- Extremely low cost (approx \$3.00 US on Amazon; one of the two least expensive components in your kits)

SEICHE LED Display Architecture



SEICHE LED DISPLAY ARCHITECTURE

- Red MAX7219 8x32 LED matrix display (SPI)
- ZS-042 real-time clock module (I2C)
- BMP280 Temperature and Pressure Sensor (I2C)
- Piezoelectric speaker (PWM)
- 10K Ω Potentiometer (Analog-to-Digital Converter)
- Button (Pullup and interrupt)

sprintf() Function Syntax

- sprintf()'s formal syntax is:
sprintf(char ***buffer**, const char ***format**, *variable list*)
- **buffer** is an array of type char (the parameter when passed can also be a pointer [address] for such an array) which will contain the formatted output of the function
- **format** is an unmodifiable (constant) array of char containing the format descriptors for all variables subsequently passed to the function, which is to say, it's the *format string*.
- The variable list are just the comma separated variables that are to be formatted
- All variables passed in a single call to sprintf() are combined into a single output string
- sprintf() automatically puts a terminating NULL (\0) at the end of the ASCII output

sprintf() Format Strings

- **sprintf()** format strings contain conversion specifiers that specify how each individual variable is to be converted
- **sprintf()** conversion specifiers have the following general syntax (all begin with a percent-sign):

%[flags][minimum field width][.][precision][length][conversion character]

- **%** - special token that indicates the start of a conversion specifier
- **Flags** – these modify the behavior of the specification
- **Minimum field width** – as it says on the tin; this the minimum number of characters to be converted
- **.** – The period is a separator between field width and precision
- **Precision** – means one of the following depending on the variable type and conversion specifier
 - The maximum number of characters to be generated from a string
 - The number of digits after the decimal point for type float conversions (e, E, or f)
 - The zero-filled minimum number of digits for an integer
- **Conversion character** – A single character which determines the output type of the conversion specifier; a single character that specifies the type of output format for the corresponding data or variable

sprintf() Conversion Specifiers

Below are the general conversion specifiers and what they do.

Specifier	What it does
d, i	int - integer; signed decimal notation
o	int – unsigned octal (no leading zero)
x, X	int – unsigned hexadecimal, no leading 0x
u	int – unsigned decimal
c	int – single character, after conversion to unsigned char
s	char * - characters from string are printed until \0 (NULL) or <i>precision</i> is reached
f	double – decimal notation of form [-]mmm.ddd where number of decimals is specified by precision; precision of zero (0) suppresses the decimals altogether
e, E	double - exp notation; default precision of 6, 0 suppresses
g, G	double – Use %f for $<10^4$ or %e for $>10^4$
p	void * - print output as a pointer, platform dependent
n	Number of characters generated so far; goes into output
%	No conversion, put a % percent sign in the output

sprintf() Conversion Specifiers

Below are the flags and what they do.

Flag	What it does
-	Left justification
+	Always print number with a sign
<i>spc</i> (space)	Prefix a space if first character is not a sign
0 (zero)	Zero fill left for numeric conversions
#	Alternate output form depending on conversion character o – first digit will be zero x or X – 0x or 0X (respectively) prefixed to non-zero results e, E, f, g and G – Output will always have a decimal point g and G – trailing zeroes will never be removed