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 – Internet Access

- Class Exercise: Displaying Sensor Data With Async Web Server
- 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.*

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-Software

ESP8266FS-0.5.0.zip

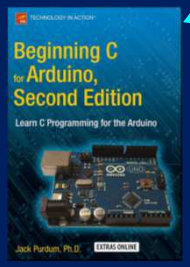
IntermediateProgramming-Lessons

- IntermediateProgramming-Lesson1
- IntermediateProgramming-Lesson2
- IntermediateProgramming-Lesson3
- IntermediateProgramming-Lesson7

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 7 – Web Technology

Part A

- More on HTML tags
- The Image Tag
- Class Exercise – Image Tags
- Hyperlinks
- Class Exercise – Hyperlinks
- Web Page Inspection
- HTTP and Mime Types

Part B

- Using SPIFFS
- Class Exercise: Display A Web Page

More on HTML Tags

- There are seemingly innumerable HTML tags
- The most critical tags do not just format text, they cause the browser, and sometimes the server to take actions
- One common action is to display another file, particularly images
- Another common action is to jump to another web page (a "hyperlink")
- And yet another common action is send input from the client browser back to the server (form input)
- Just like subroutines and CLI programs, certain tags take *arguments*, which are parameters specified *inside the tag* which tell the client browser or the server what to do
- Note also that not all tags necessarily take closing delimiters, e.g., `</b>`. This is particularly true of "action tags".
- We will look at all of these in turn

The IMG tag

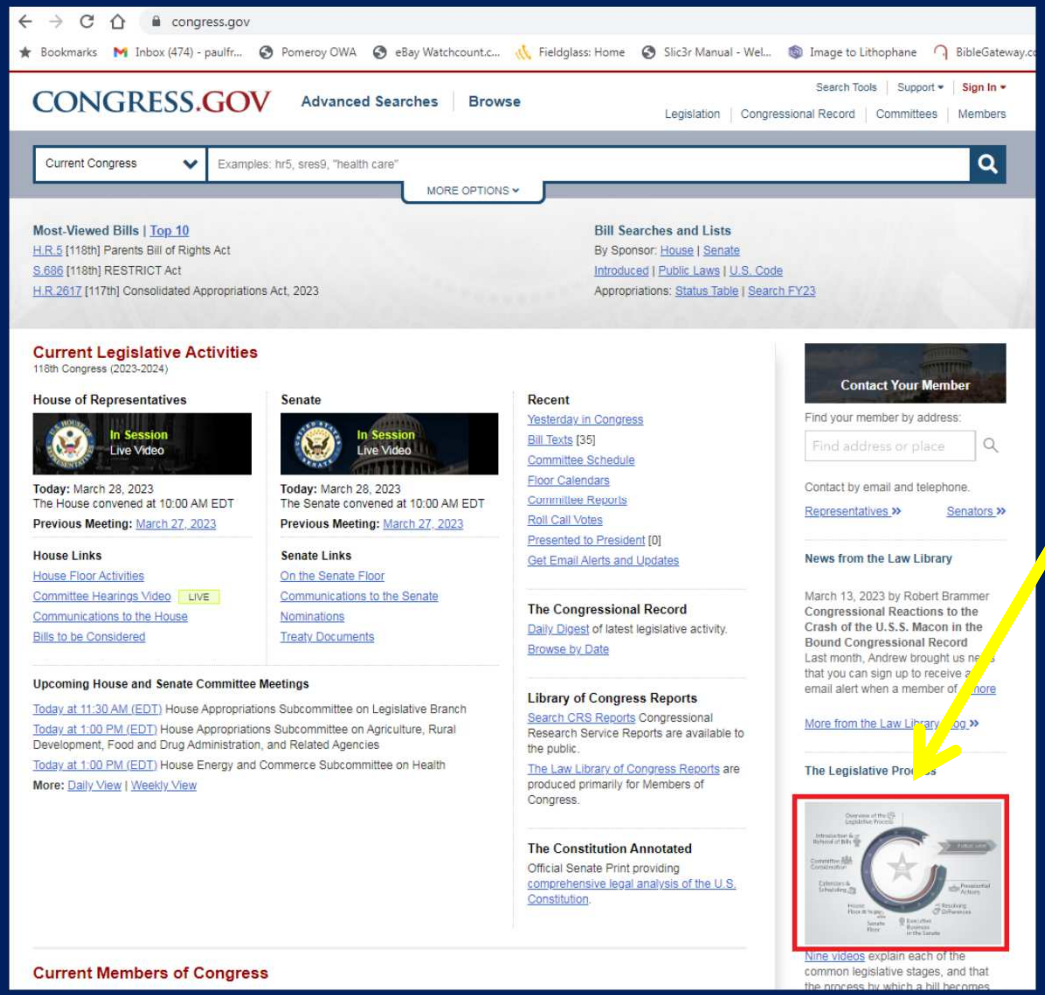
- The IMG tag is used to include images
- The images are typically on the same server as the web page with the IMG tag, *but they don't have to be*
- The img tag employs several *arguments*, and these arguments each take data called *parameters*
- Here is the full tag syntax:

```
<img src="url|filename" [alt="descriptive text" [width="x"] [height="y"]>
```

- And here are what all those arguments and parameters do:
  - `<img` : the start of the tag itself, including the opening `<` delimiter
  - `src=` : specifies the image filename, *or url of the image*, to be included at this location on the page
  - `alt=` : Optional parameter which specifies plain text that will be displayed by a browser if it doesn't know how to handle the image type specified. Rarely a problem these days, but proper form
  - `width=` and `height=` : Optional parameter which allows rescaling of an image. If not provided, image will be displayed at its native resolution.
  - `>` : The closing delimeter for the tag
- Note that the parameters for arguments are always in "*double quotes*"!

The IMG tag (cont.)

- Let's see what all that looks like on the live internet. Here is a screen capture of the main page for http://congress.gov. We are interested in the graphic element (picture) outlined in red:



- <img src="https://www.congress.gov/img/video-legislative-process.jpg" alt="Legislative Process video" width="204" height="148">

Here is that graphic element "scraped" from the page (via right click -> Save As) and displayed on its own

Class Exercise: Image Tags

- Let's add that graphic to our web page from last time. Open the file seiche.html (in your Lesson7 folder) in your HTML editor (VS Code, Kate, or Notepad) and modify it to look like this:

```
<html>
  <head>
    <title>SEICHE Website</title>
  </head>
  <body>
    <h1>Welcome to our web page!</h1>
    <img video-legislative-process.jpg>
  </body>
</html>
```

- You will need to "scrape" the graphic from the congress.gov page by opening that page in a browser, then right-clicking on the image, then selecting "Save Image As". Navigate to the Lesson7 folder where your seiche.html file is and save the image

- Once you have modified your HTML file and saved the graphic, save your HTML file, then open seiche.html in a browser.
- You should see a display similar to that of Windows Firefox shown at right

Hyperlinks

- "In computing, a hyperlink, or simply a link, is a digital reference to data that the user can follow or be guided to by clicking or tapping. A hyperlink points to a whole document or to a specific element within a document. Hypertext is text with hyperlinks. The text that is linked from is known as anchor text." – Wikipedia
- Hyperlinks are the heart and soul of the web– they are what "make it work"
- A little known fact is that hyperlinks were used prior to the web. A hyperlinked data storage system famous in it's day was Apple's *Hypercard* for the Macintosh
- In HTML, no surprise, hyperlinks are implemented as tags. And the hyperlink tag syntax looks like this:
  `<a href="url"> HTML body content- any HTML body content! </a>`
- Example: `<a href="https://www.w3schools.com/tags/tag_a.asp">Hyperlink Tag Reference Page</a>`
- W3C Hyperlink reference page is here: https://www.w3schools.com/tags/tag_a.asp (did you see what I did there? ☺)
- Note that:
  - The hyperlinks tag has both opening <a> and closing delimeters
  - Since *any* HTML can be "made into" a hyperlink by enclosing it within the opening and closing tags, entire images can be made into tags:
    `<a href="https://congress.gov"><img src=video-legislative-process.jpg></a>`

Class Exercise: Hyperlinks

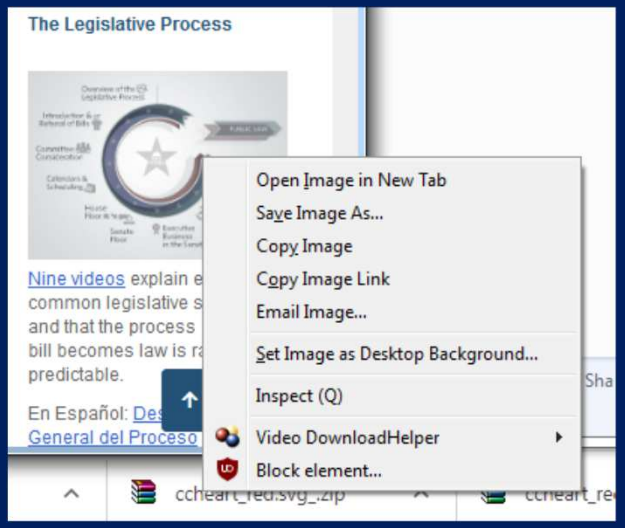
- Now let's add some hyperlinkage to our sample HTML file. Make a copy of your seiche.html file by saving it as hyperlink.html. Then modify the new file as shown below to turn the legislative process picture into a hyperlink.

```
<html>
  <head>
    <title>SEICHE Website</title>
  </head>
  <body>
    <h1>Welcome to our web page!</h1>
    <a href="https://congress.gov">
      <img src=video-legislative-process.jpg>
    </a>
  </body>
</html>
```

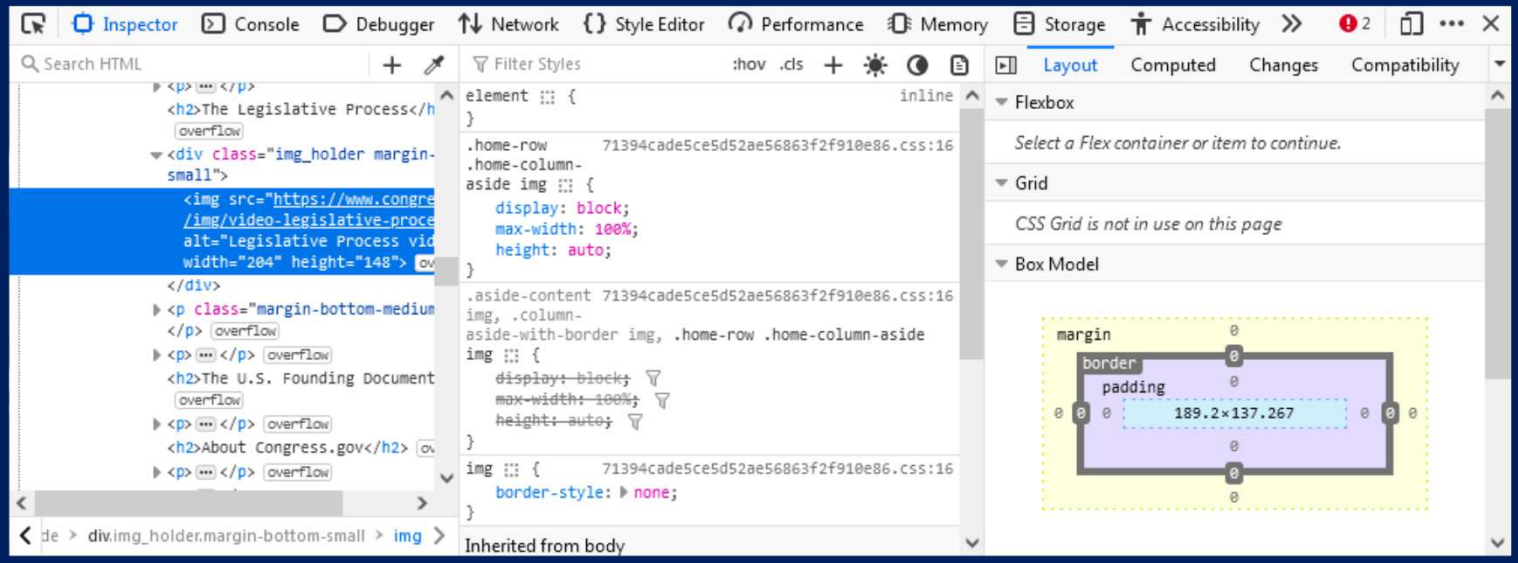
Doesn't *look* any different, does it?

- Once you have modified your HTML file and saved it, open the file in a browser. Your should see a page like the one at right.
- Click (right-click) on the image. What happens?
- Note that, as shown in the HTML here, HTML parsers (browsers) *ignore whitespace*, so you can arrange tags on their own lines to make the HTML easier to read.

Web Page Inspection

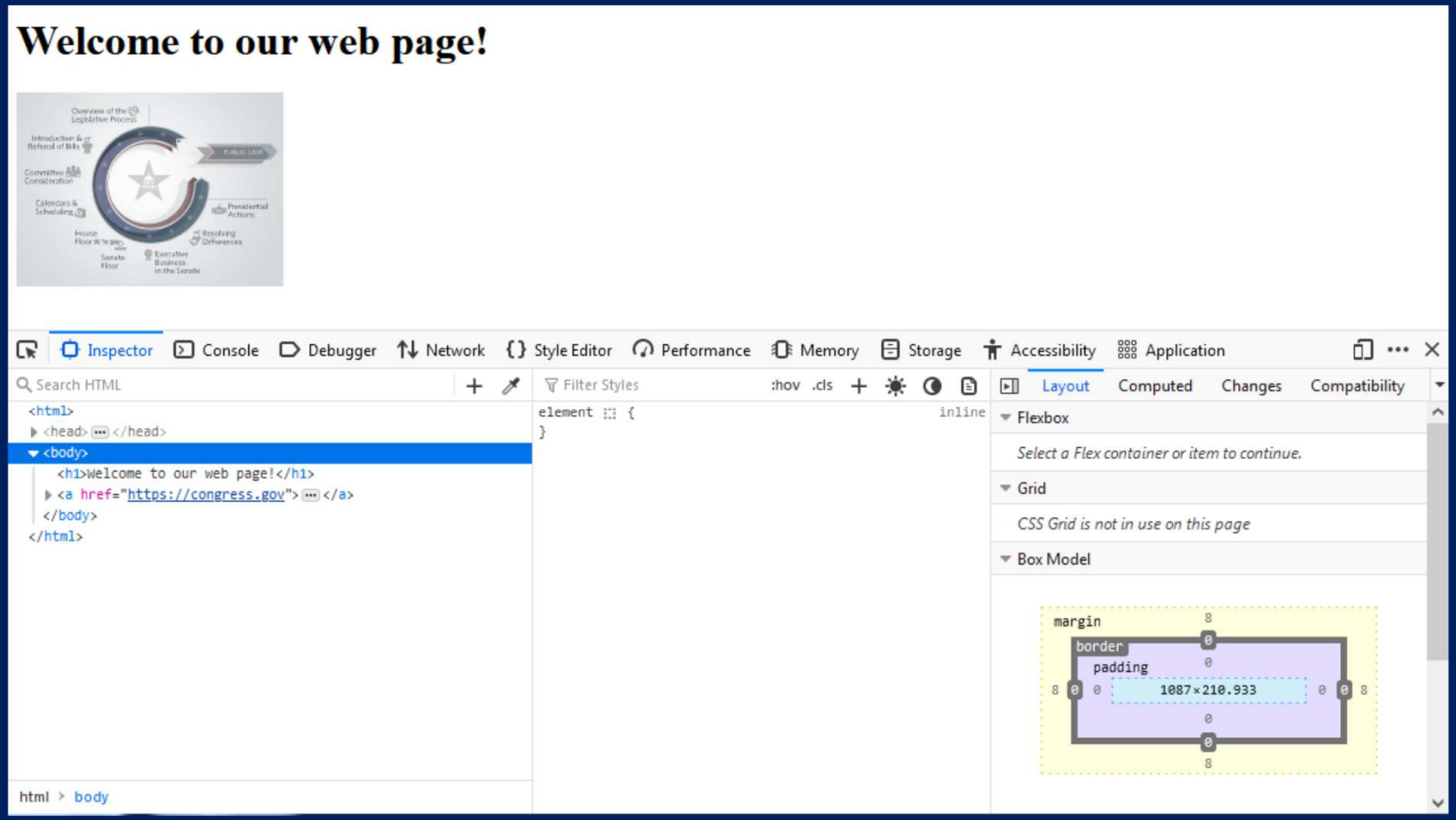


- If you navigate to good old http://congress.gov, and once again left click on the image we've been using, you should see the menu at right if you are using Firefox.
- This time, select "Inspect"
- This will pop some developers' HTML inspection windows, as see below.



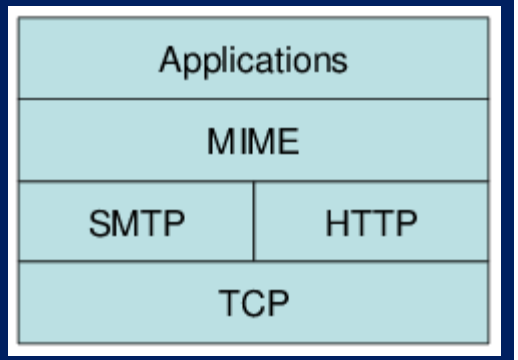
Web Page Inspection (cont.)

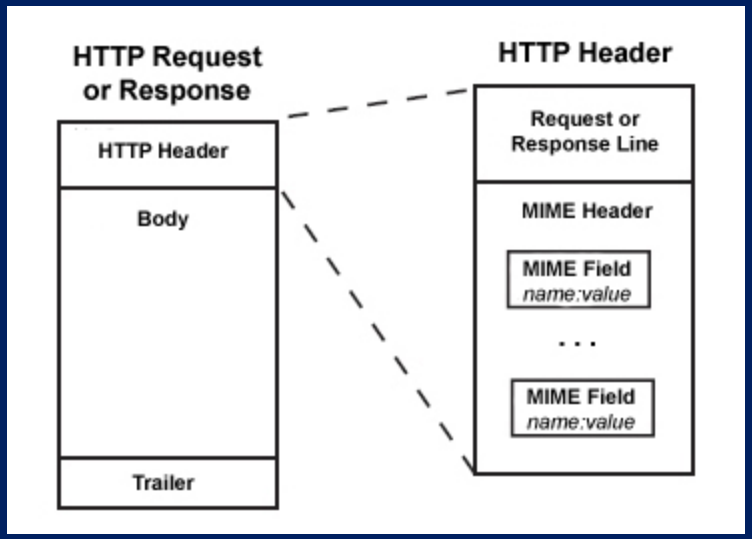
- And of course, you can do the same thing for the page(s) y'all just made!

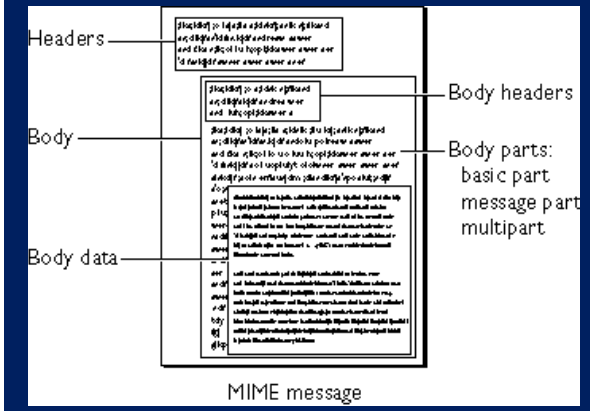


HTTP and MIME Types

- Multipurpose Internet Mail Extensions (MIME) were a series of protocols developed for encapsulating binary data (applications, images) in plain text.
- MIME was originally used for the plain-text based e-mail systems of the 1990's
- When the W3C (Word Wide Web Consortium, natch) needed a protocol for identifying data types in HTTP, they chose the already-existing MIME, rather than reinvent the wheel





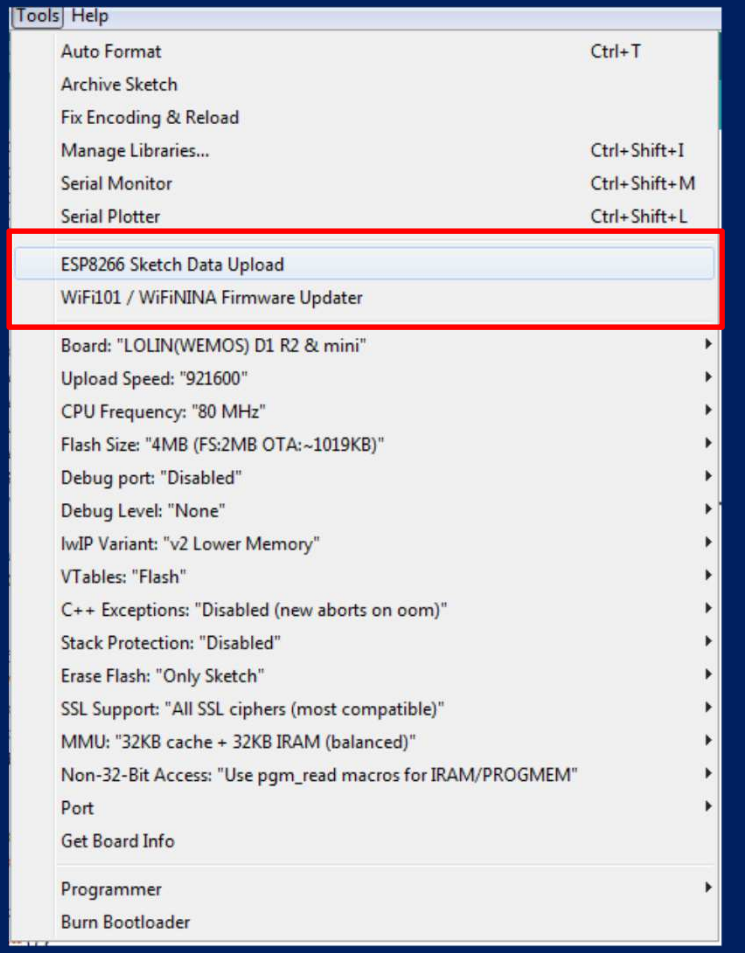


You don't need to know the gory details for *this* class, but you'll see MIME references *frequently* when working with HTML and webservers!

Using SPIFFS

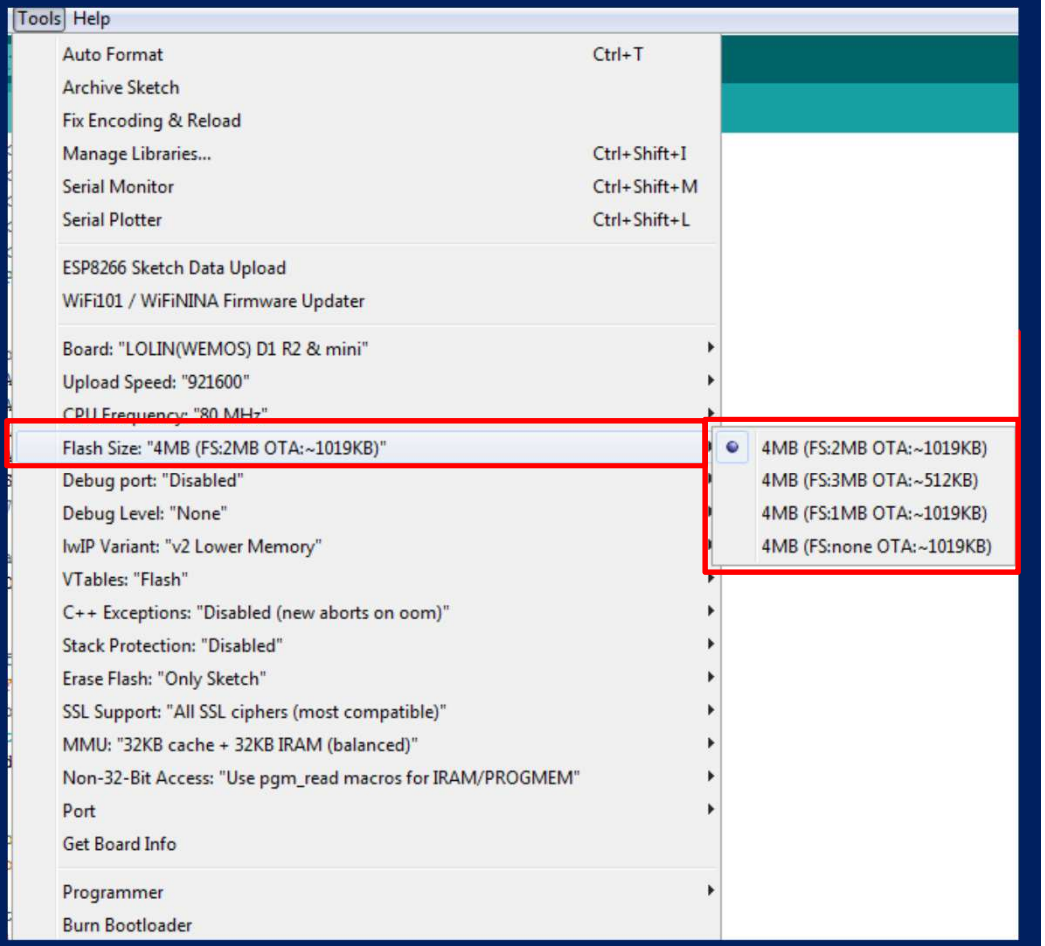
- We already loaded the SPIFFS tool into our IDE's. Now we will learn how to use it.
- SPIFFS requires that any data to be uploaded be present in a folder named "data" in the same folder as the sketch that will be using it:

  [SKETCH7A]
  - SKETCH7A.ino
  - [data]
    - file1
    - file2
    - .....
- This data is uploaded by selecting the ESP8266 Sketch Data Upload item from the Tools menu in the IDE.
- When the menu item is selected, all files in the "data" folder are uploaded to the ESP8266 microcontroller flash



Using SPIFFS (cont.)

- A critical aspect of using SPIFFS is that you must be aware of just how much storage the files you are uploading use.
- The ESP8266 Wemos D1 Mini we use by default has 4MB of flash. That has to be divided between code and SPIFFS storage.
- You can view, and change, the allocation from the Tools menu as shown at right
- The default 2:1 ratio of SPIFFS (FS) to code (OTA) is usually adequate in most cases
- If you need a gob of storage for web files, you'll need to move to an ESP32. Which will also be a *much* faster server.



CLASS EXERCISE – DISPLAY A WEB PAGE!

- Open SKETCH7A; ***do not upload it!!***
- The necessary web files are already in the “data” directory, however, you will need to upload them
- As shown previously, select ESP8266 Data Upload from the Tools menu to initiate the upload of the data files to your display
- Once the data has uploaded, upload your sketch as usual
- You will need to obtain the IP address of your WiFi connection from the Serial Monitor (I haven’t added a Parola IP display to this sketch yet.)

DISPLAY A WEB PAGE! (cont.)

- Once you have the IP address of your display, open it in a browser using a URL, thus: `http://`*ip-address*
- You should see a browser display like the one below!

Formal End of Lesson 5

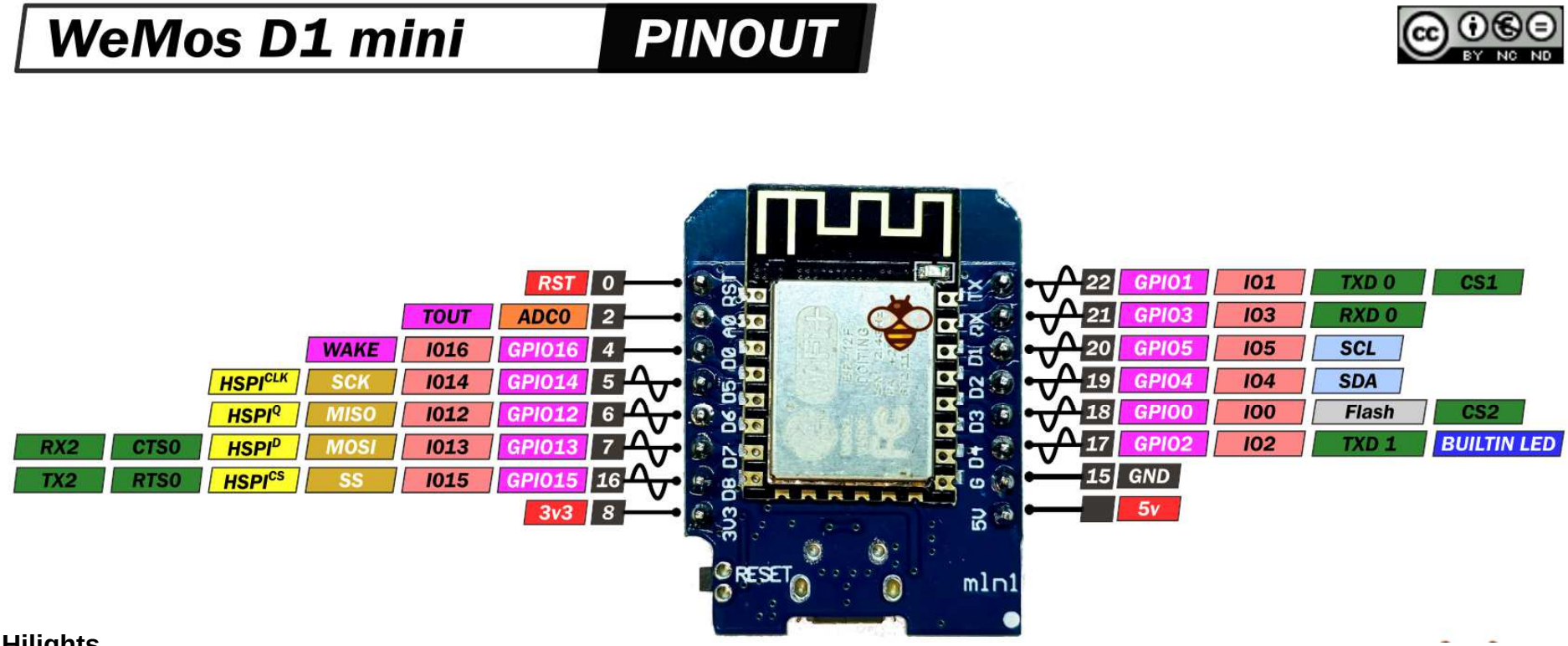
HOMEWORK

- **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

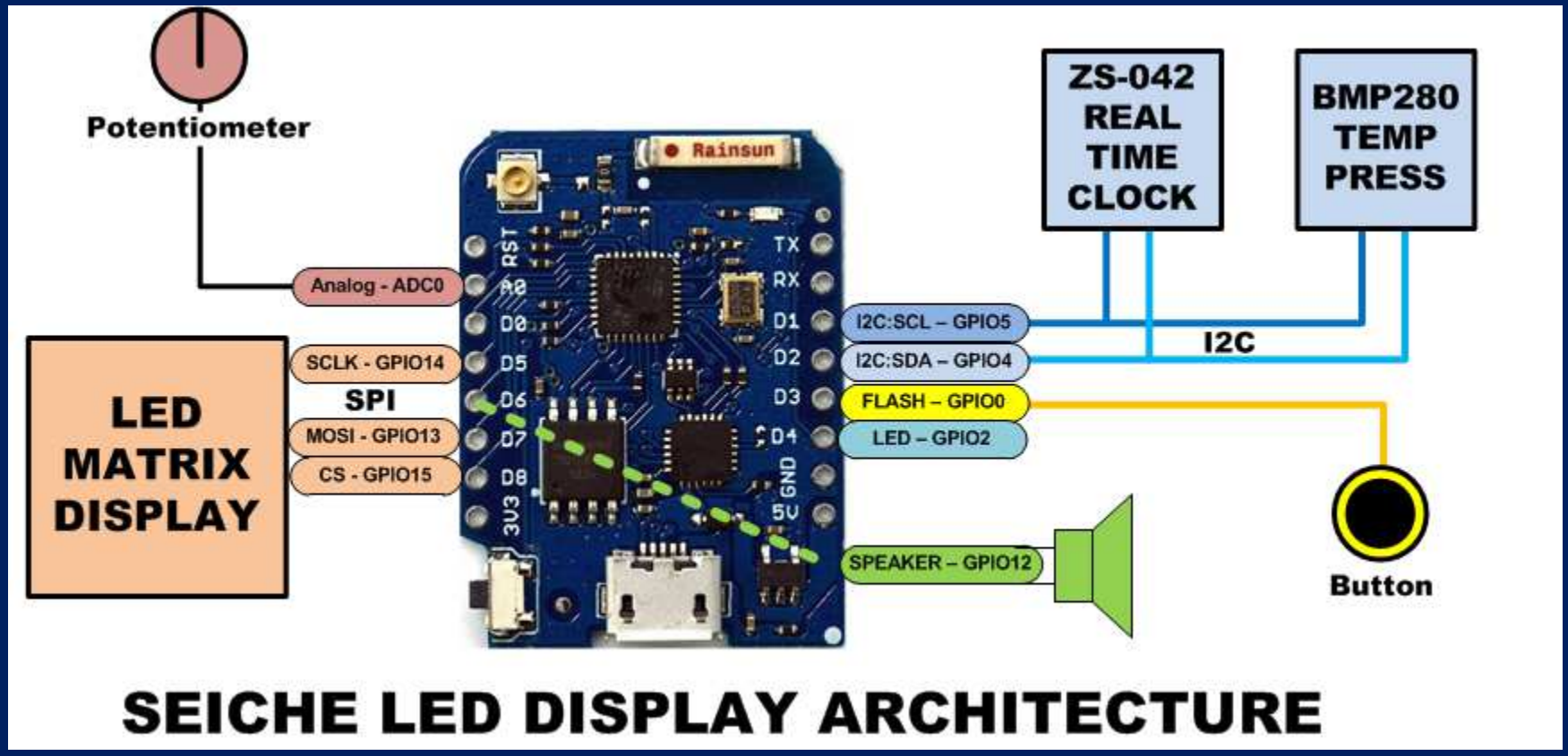
Our Microcontroller: The WeMos D1 Mini



Hilights

- **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



- Red MAX7219 8x32 LED matrix display (SPI)
- ZS-042 real-time clock module (I2C)
- BMP280 Temperature and Pressure Sensor (I2C)
- Piezoelectric speaker (PWM)
- 10KΩ 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()` automagically 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
  - Minumum 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 *precision* 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
spc (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