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

- HTML Forms
- Processing form input with Arduino
- Class Exercise: Display messages from a web page

Lesson 8: 4Apr23 – Internet Access

- 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

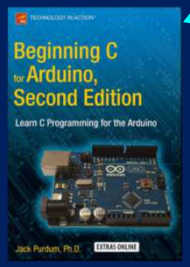
IntermediateProgramming-Lessons

- IntermediateProgramming-Lesson1
- IntermediateProgramming-Lesson2
- IntermediateProgramming-Lesson3
- IntermediateProgramming-Lesson6

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!



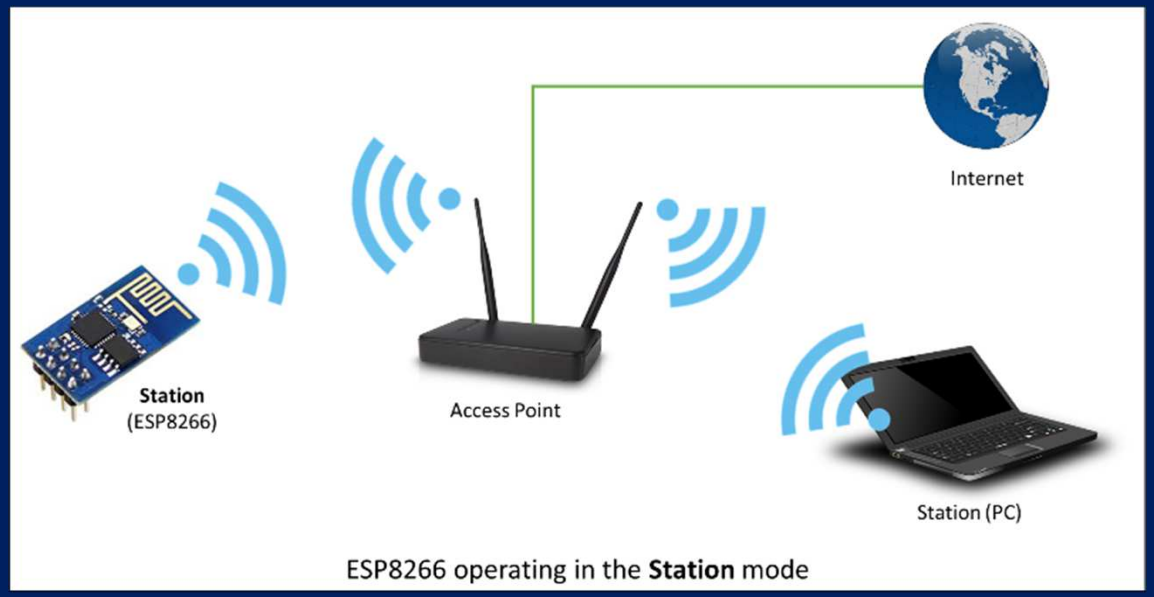
Lesson 6 – Web Servers, Browsers, and HTML

Part A

- WiFi Network Review
  - WiFi Operation
  - Joining a WiFi network
  - Getting an IP address
- Class Exercise: ESP8266 WiFi redux
- Basic web operation
- HTML markup
- Bare bones web page structure
- Class Exercise: Creating a web page

Wireless Network Review

- Recall that in most cases you will be connecting to an existing access point
- In this situation, each device that connects to an AP is called a *client*
- Typically, the AP is then uplinked to the Internet
- The ESP8266 mode for acting as a client is called, counterintuitively, *station mode*



ESP8266 operating in the **Station** mode

- AP – Access Point
- SSID – Service Set Identifier; the name of a wireless network
- IP Address – a unique 32 or 64 bit number assigned to each device connecting to a network (regardless of wired or wireless)
- DHCP – Dynamic Host Configuration Protocol – The means by which a network hands out IP addresses to devices
- Host – Catch-all term for a device connected to a network

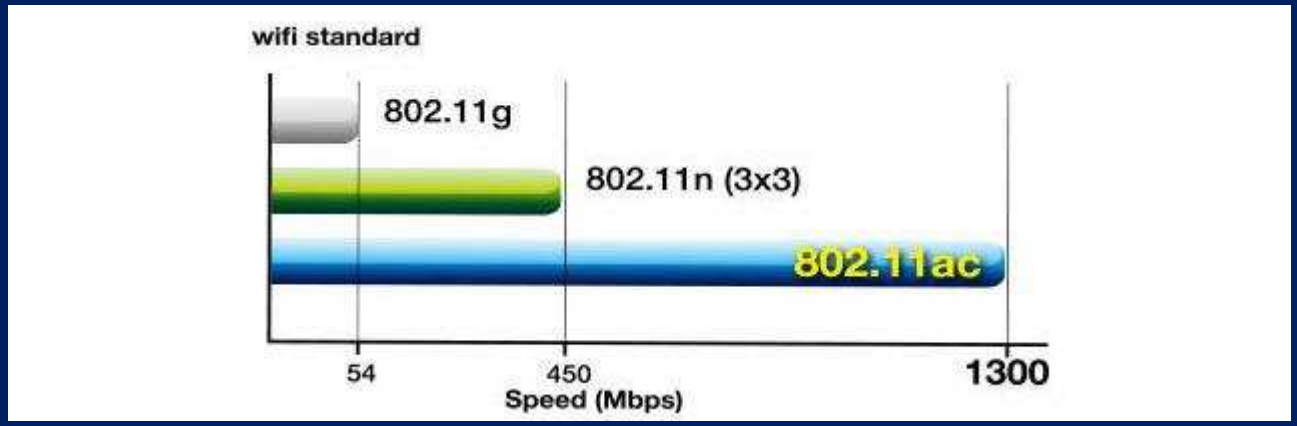
Wireless Network Review

- An ESP8266 can act as a *soft access point*
- In this mode, the ESP8266 *itself* provides WiFi connectivity for clients; it acts just like an AP
- It's called "soft" because it's programmable

ESP8266 operating in the **Soft Access Point** mode

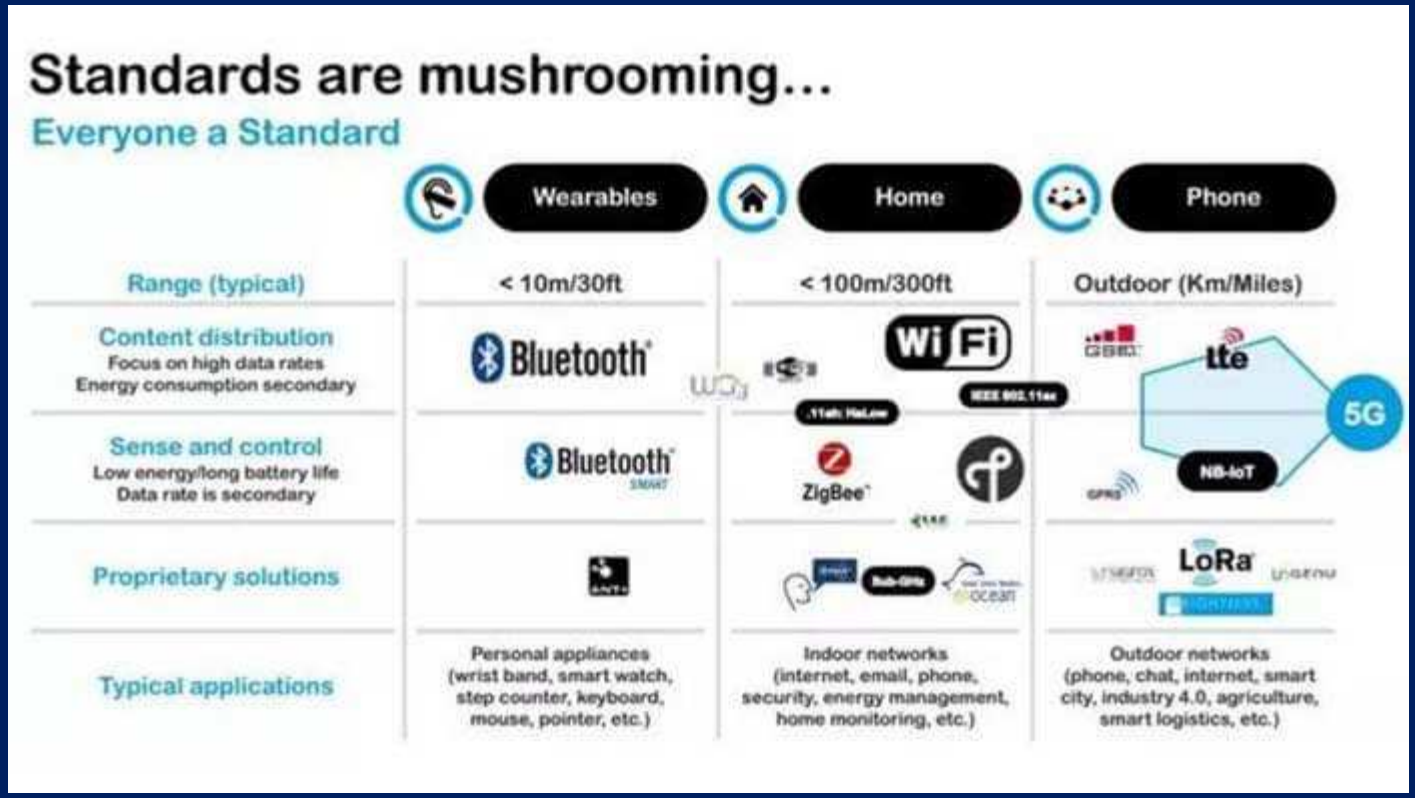
Wireless Network Types

- There are different types of wireless, mostly differentiated by speed and range
- Nearly all *Ethernet* based wireless networks support multiple speeds simultaneously for ***backwards compatibility***



Wireless Network Types (cont)

- There are many other types of wireless network
- For this class, we are concerned only with Ethernet based wireless, common known as “WiFi”



A Note On WiFi

- **WiFi is an industry trademark of the Wireless Alliance; it has a formal logo**
- **WiFi has *no technical meaning*, and is only a marketing term designed to replace technical naming with a simple, consumer-friendly moniker**
- **And the plan worked!**

WiFi Multi

- ESP8266WiFiMulti.h can be used to connect to a WiFi network with strongest WiFi signal (RSSI). This requires configuring one or more access points with SSID and password. It automatically switches to another WiFi network when the WiFi connection is lost.
- Having multiple SSID’s configured allows the ESP8266 to easily be moved between different networks

Classroom Exercise - WiFi

And here is our class sketch for WiFi. Go ahead and load and run!

SKETCH6A – WiFi Setup Excerpt

```
// Setup for wifiMulti
ESP8266WiFiMulti wifiMulti;
// WiFi connect timeout per AP. Increase when connecting takes longer.
const uint32_t connectTimeoutMs = 5000;
char ipaddrbuffer[20];


void setup(){
  Serial.begin(9600);

  // Don't save WiFi configuration in flash - optional
  WiFi.persistent(false);

  Serial.println("\nESP8266 Initializing Multi WiFi\n");

  // Set WiFi to station mode
  WiFi.mode(WIFI_STA);

  // Register multi WiFi networks
  wifiMulti.addAP("CFC_Public", "");
  wifiMulti.addAP("PALAS", “xxxxxx");
  // wifiMulti.addAP("ssid_from_AP_3", "your_password_for_AP_3");

  if (wifiMulti.run(connectTimeoutMs) == WL_CONNECTED)
  {
    Serial.println("WiFi connected!");
    Serial.print("WiFi SSID: ");
    Serial.println(WiFi.SSID());
    sprintf(ipaddrbuffer,"%s",WiFi.localIP().toString().c_str());
    Serial.print("WiFi IP Address - ipaddrbuffer: ");
    Serial.println(ipaddrbuffer);
  }
  else
  {
    Serial.println("WiFi not connected!");
  }
  Serial.println("\n\n");
}
```

ESP8266 WiFi

- We are now using WiFiMulti, which allows us to specify multiple SSID's (wireless networks) that our displays will try to connect to; whichever has the strongest signal will win. This means we can take our display different places and it will still work (as long as we know the network credentials in advance)

```
ESP8266WiFiMulti wifiMulti;
```

- This is how we set WiFi for "connect to an existing station":

```
WiFi.mode(WIFI_STA);
```

- The while() loop will keep looping as long as WiFi.status() is other than WL_CONNECTED. The loop will exit only if the status changes to WL_CONNECTED.
- Once the ESP8266 connects, we print out the SSID that it ended up using:

```
Serial.print("WiFi SSID: ");
Serial.println(WiFi.SSID());
```

- When we connect to a webserver, we need to know it's IP address. This statement prints to the Serial Monitor the IP address assigned to our ESP8266 by the WiFi network:

```
Serial.println(WiFi.localIP());
```

ESP8266 WiFi (cont.)

- If we want to show the IP address on our dot matrix displays, more steps are involved. This is because the IPaddress data type is in actuality a complex, multifarious C++ class– a class which Serial.Println() knows how to handle, but MD_Parola does not.
- So we have to convert the Ipaddress data structure into an array of char– a string– which MD_Parola will accept
- The incantation for this conversion is:

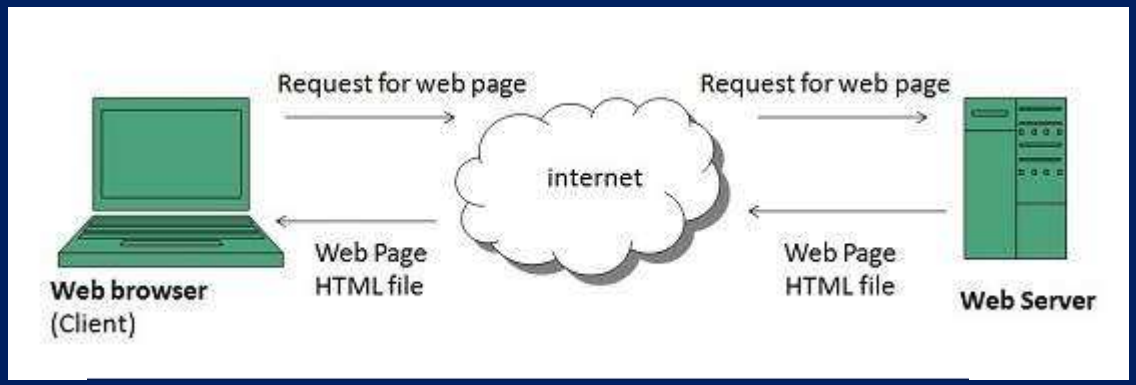
```
char ipaddrbuffer[20];
sprintf(ipaddrbuffer,"%s",WiFi.localIP().toString().c_str());
```

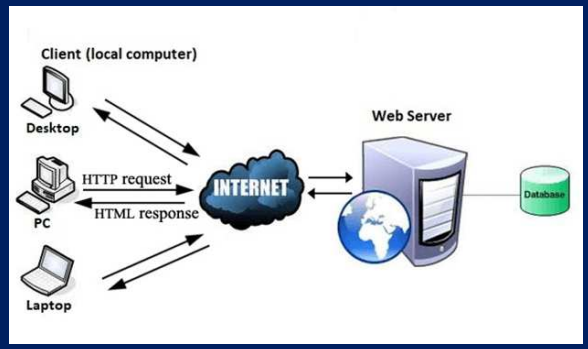
- Once we have the IP address as a string in ipaddrbuffer, we can display it with MD_Parola:

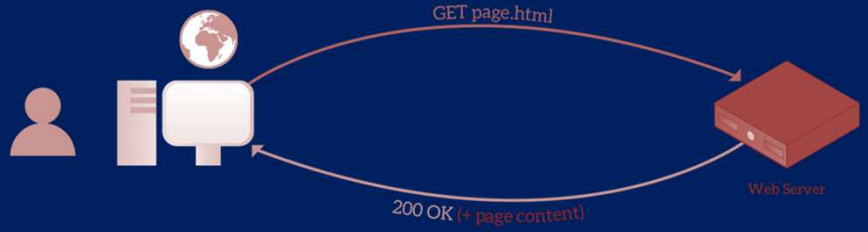
```
if(pmx.displayAnimate())
{
     pmx.displayText(ipaddrbuffer,PA_CENTER,50,0,PA_SCROLL_LEFT,PA_SCROL
L_LEFT);
    yield();
}
```

Basic Web Operation

- The World Wide Web (WWW), runs on a network protocol called Hypertext Transfer Protocol (HTTP)
- HTTP describes how clients, which are requestors, can obtain content from servers, which are content providers
- Thus, HTTP uses a *client-server* communication model

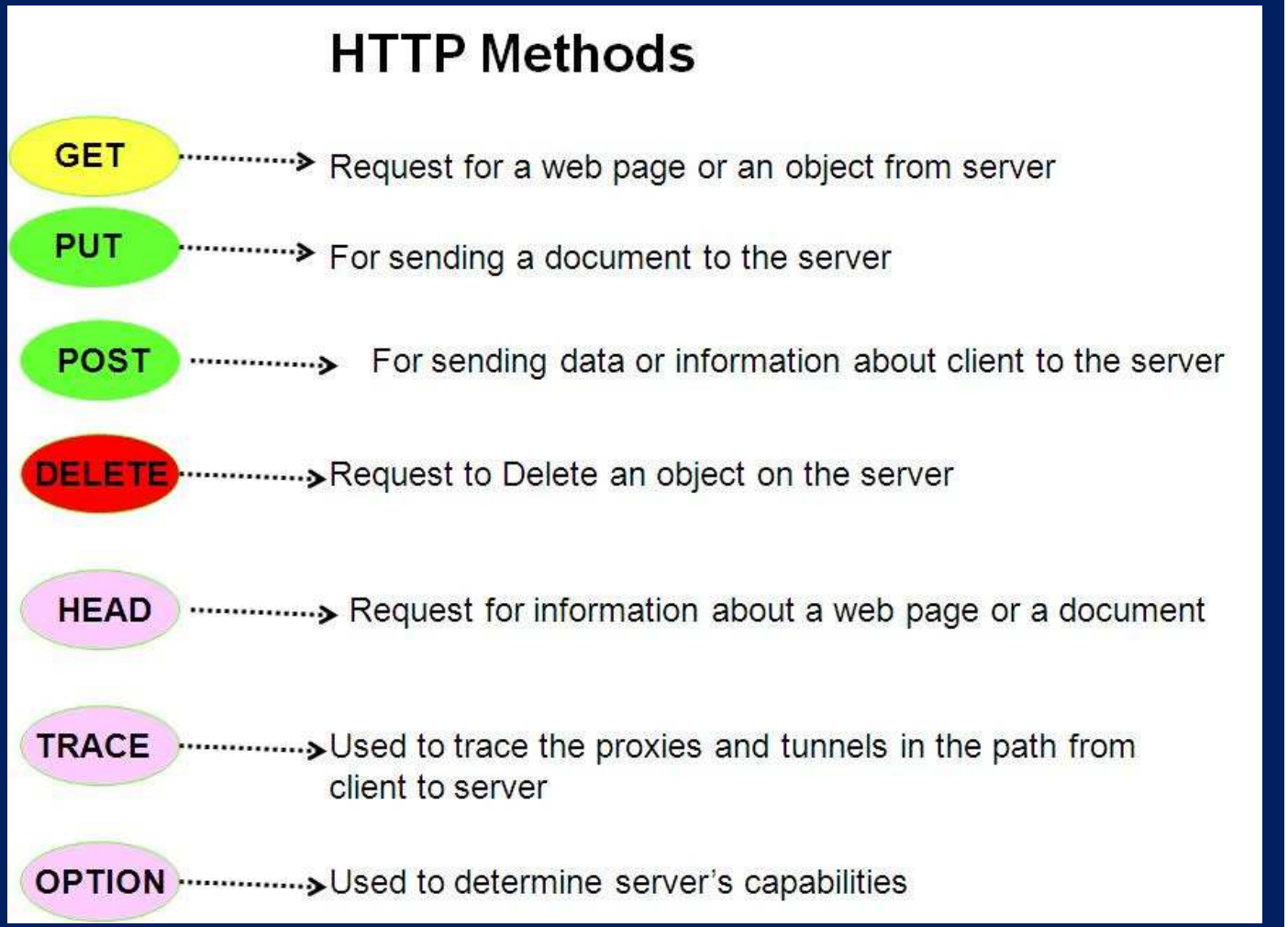






HTTP Operation

- HTTP has several modes which clients and servers use to exchange data
- Which mode is used has to be mutually agreed upon (negotiated)
- Certain modes are designed for this negotiation



HyperText Markup Language - HTML

- There is a specific document format that must be used by all webservers and clients, called HTML
- HTML is a formal *markup language*– a metagrammar and container notation that is used to “wrap” other content in “tags”
- Using tags to delineate different types of content gives the web servers and clients a *very* limited lexical understanding of that content

HTML tags	Produces
bold	**bold text**
italic	*italic text*
<u> underline </u>	underlined text
 example 	example
- one - two	• one • two
<hr />	

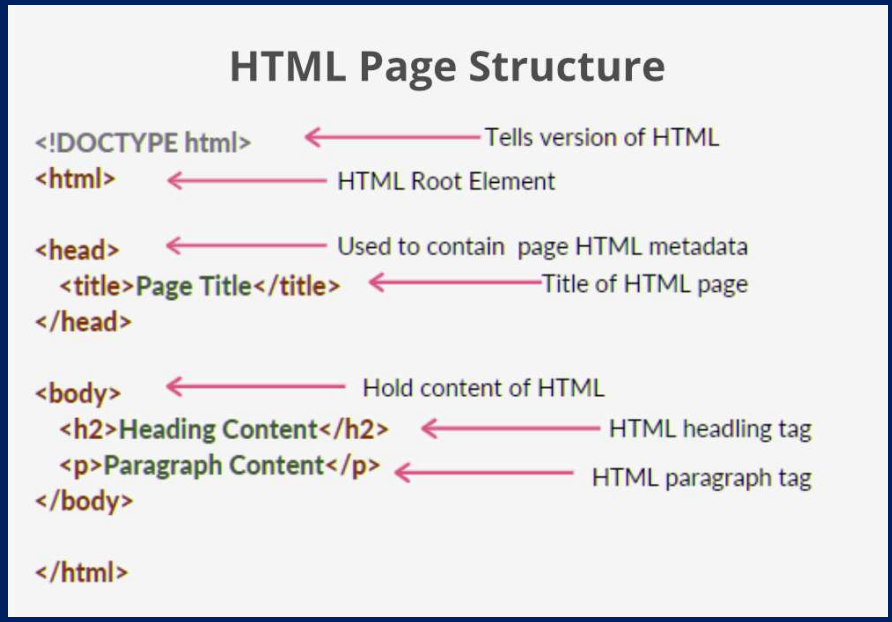
Italic	*Italic*
Bold	**Bold**
Emphasized	*Emphasized*
Strong	**Strong**
small	small
~~Deleted~~	~~Deleted~~
<ins>Inserted</ins>	Inserted
v_f	v_f
a²	a^2
<mark>Marked</mark>	Marked

web4college.com

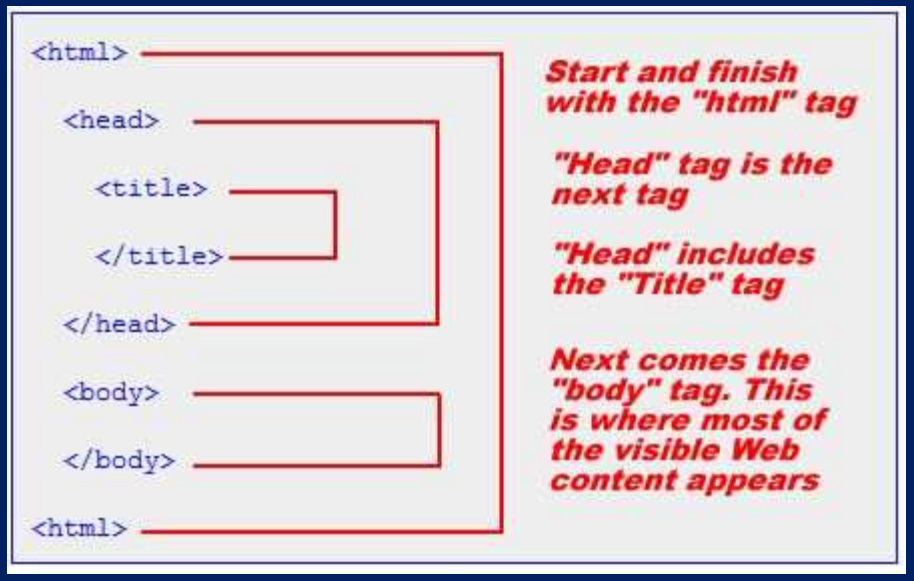
VERY basic HTML markup

Web Page Structure

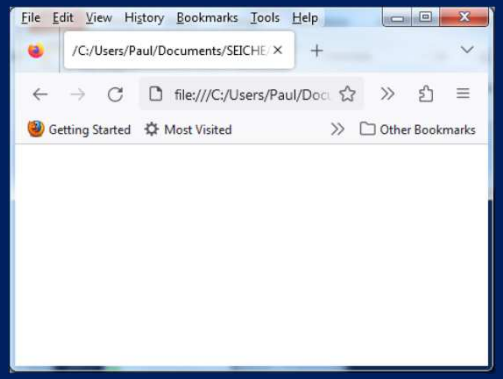
- HTML web pages use a very formal structure
- This structure is hierarchical
- The elements of the hierarchy are nested, some inside others




```
<!DOCTYPE html>
    <html>

        <head>
            <title> Title here </title>
        </head>

        <body>
            Web page content goes here.
        </body>

    </html>
```

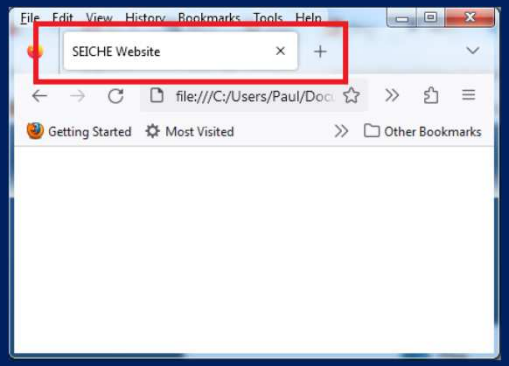


Bare Bones Web Page Structure



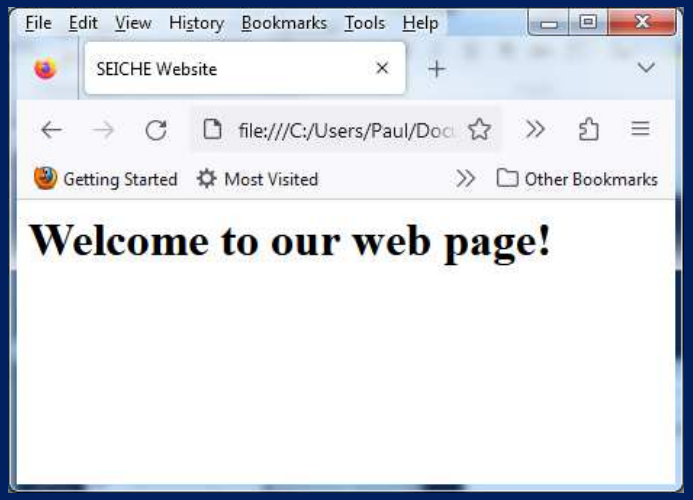

```
<html>
        <head>
        </head>
        <body>
        </body>
</html>
```

Absolute minimum; no content




```
<html>
  <head>
    <title>SEICHE Website</title>
  </head>
  <body>
  </body>
</html>
```

Just a title; title displays differently in different browsers




```
<html>
  <head>
    <title>SEICHE Website</title>
  </head>
  <body>
    <h1>Welcome to our web page!</h1>
  </body>
</html>
```

Both a title and a header

CLASS EXERCISE

- Using either VS Code, Kate (Linux), or Notepad (Windows), create a new web page, seiche4.html, in your class directory. [Hint: You can just open seiche3.html and then Save As ☺]
- Add body text to your new web page, then open it in Firefox (or the browser of your choice)
- Using the formatting tags show in previous slides, modify the web page you just created to change the style of the various sections
- Remember to reload the page to see your edits!

Formal End of Lesson 5

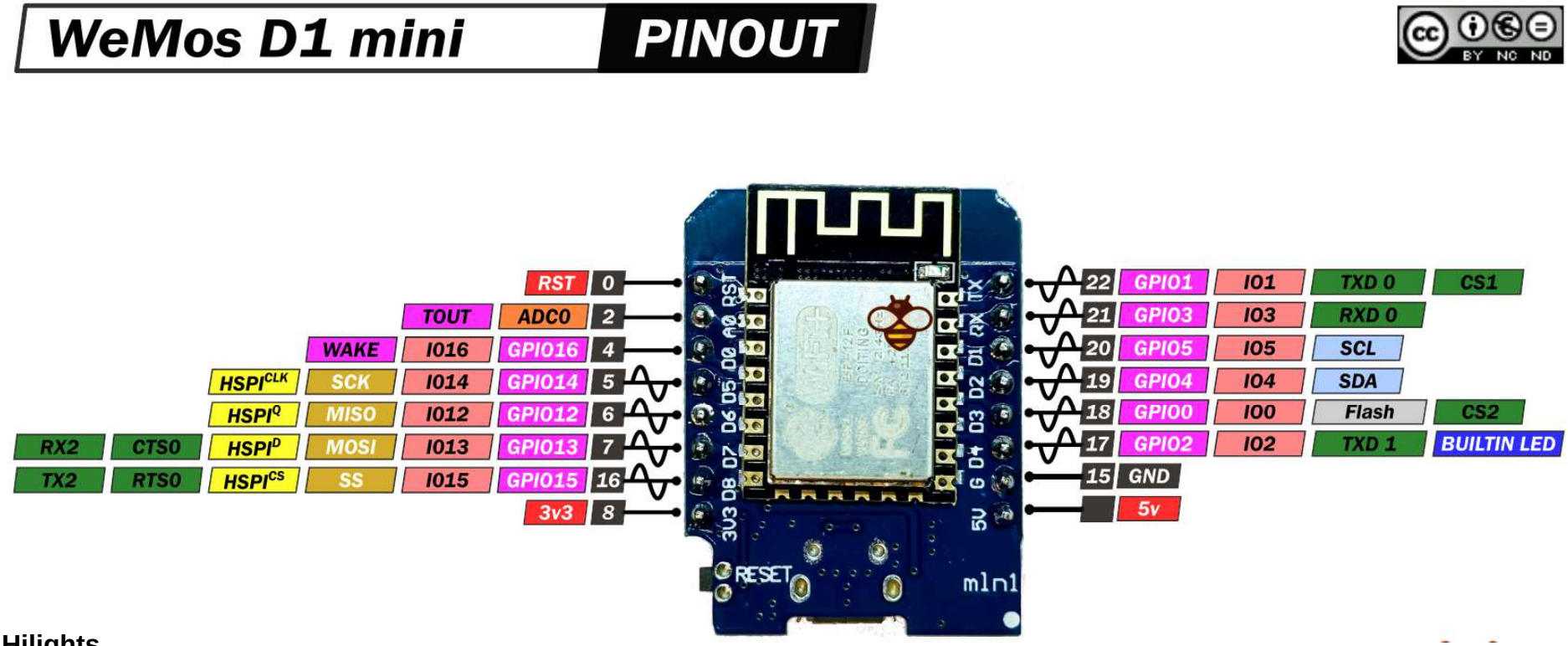
HOMEWORK

- **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

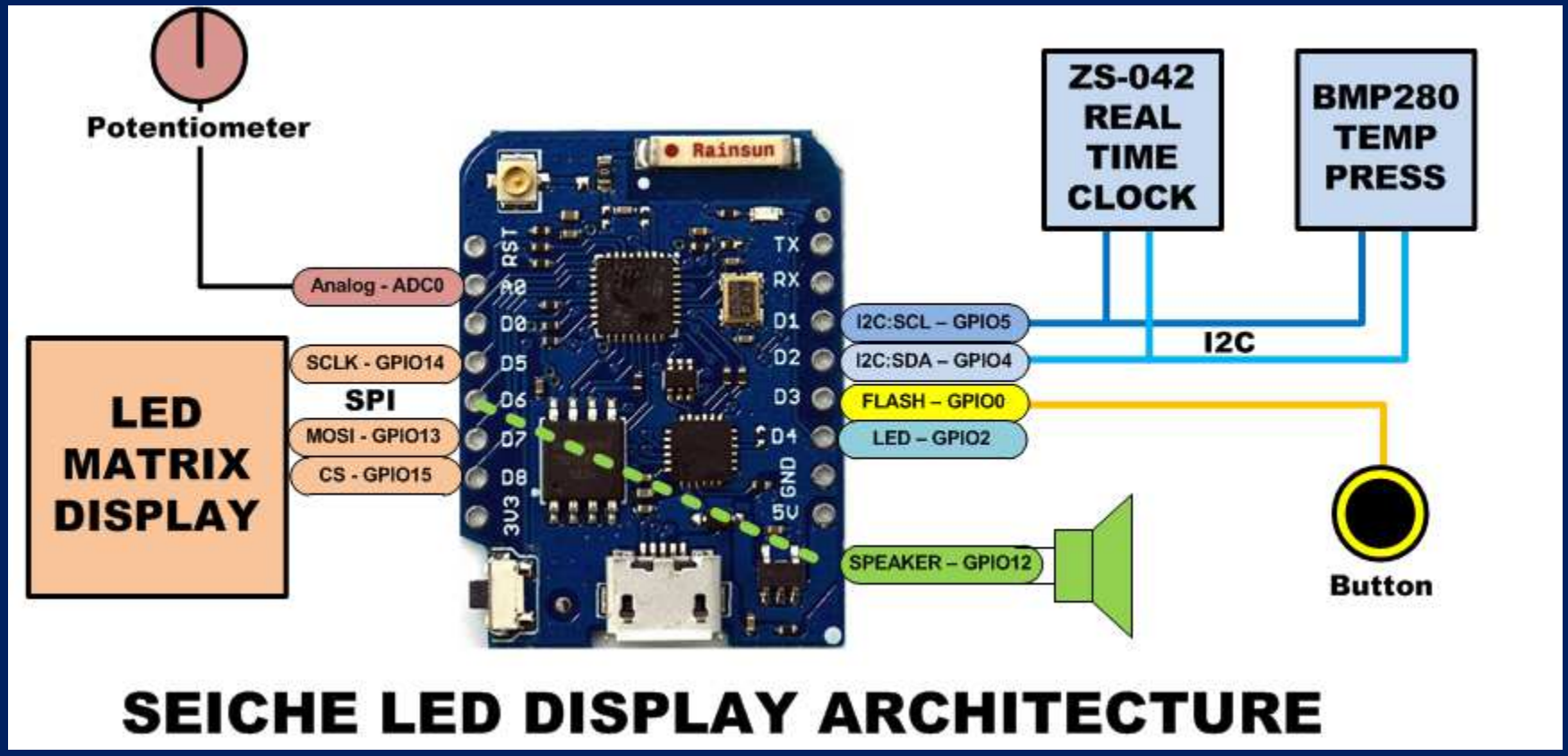
Our Microcontroller: The WeMos D1 Mini



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