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 – Wifi Manager, API's, Web Scraping

- Class Exercise: Displaying form data with MD_Parola
- APIs
- Web Scraping

Lesson 10: 18Apr23 – JSON and REST API's

- Introduction to JSON
- Using JSON with Arduino
- Using REST with Arduino

Lesson 11: 25Apr23 – Internet Access

- Scraping information from an Internet web page
- Obtaining weather information from the Internet
- Class Exercise: Getting and displaying weather information

Lesson 12: 2May23 – 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
- Trimester Review and Graduation

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

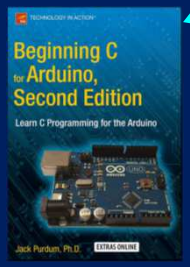
IntermediateProgramming-Lessons

- IntermediateProgramming-Lesson1
- IntermediateProgramming-Lesson2
- IntermediateProgramming-Lesson3
- IntermediateProgramming-Lesson10

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 10 – JSON and REST

Part A

- Web Scraping Followup
- Brief overview of JavaScript
- Introduction to JSON
- JSON API Examples
- Using JSON with Arduino

Part B

- Review of REST
- REST API Examples
- Using REST with Arduino

JavaScript

- JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS.
- The choice of the JavaScript name has caused confusion, implying that it is directly related to Java. JS is not related to Java, and has little to nothing in common with it.
- JS scripts are embedded in or included from HTML documents and interact with the Document Object Model (DOM). *All major web browsers have a built-in JavaScript engine that executes the code on the user's device*.
- jQuery is by far the most popular client-side library, used by over 75% of websites.[39] Facebook created the React library for its website and later released it as open source; other sites, including Twitter, now use it. Likewise, the Angular framework created by Google for its websites, including YouTube and Gmail, is now an open source project used by others.
- In contrast, the term "Vanilla JS" has been coined for websites not using any libraries or frameworks, instead relying entirely on standard JavaScript functionality.[40]
- Examples of client-side JavaScript usage:
  - Loading new web page content without reloading the page (via Ajax or a WebSocket.) For example, users of social media can send and receive messages without leaving the current page.
  - Web page animations, such as fading objects in and out, resizing, and moving them.
  - Playing browser games.
  - Controlling the playback of streaming media.
  - Generating pop-up ads or alert boxes.
  - Validating input values of a web form before the data is sent to a web server.
  - Logging data about the user's behavior then sending it to a server. The website owner can use this data for analytics, ad tracking, and personalization. (Attackers can leverage this, too)
  - Redirecting a user to another page.
  - Storing and retrieving data on the user's device, via the storage or IndexedDB standards. (A frequently exploited attack vector)

Source: Wikipedia

Web Scraping Followup

- **Here is the video from last week we did not have time to watch:**
  Scraping data for an ESP8266 or ESP32
  https://www.youtube.com/watch?v=kWdfCCwhQTE

JavaScript (cont.)

- The use of JavaScript has expanded beyond its web browser roots. JavaScript engines are now embedded in a variety of other software systems, both for server-side website deployments and non-browser applications.
- Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more (including microcontrollers).
- Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.
- Node.js lets developers use JavaScript to write command line tools and for *server-side scripting*.
- The ability to run JavaScript code on the server is often used to generate dynamic web page content before the page is sent to the user's web browser.
- Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, as opposed to using different languages for the server- versus client-side programming
- The VisualStudio Code IDE provides editing and debugging for Node.js, as well as JavaScript

Source: Amazon

JSON – JavaScript Object Notation

- JSON (JavaScript Object Notation) is a lightweight text-based open standard design for exchanging data.
- JSON is primarily used for serializing and transmitting structured data over network connection – exchanging data between a server and a client. It is often used in services like APIs (Application Programming Interfaces) and web services that provide public data.
- It is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values).
- It is a common data format with diverse uses in electronic data interchange, including that of web applications with servers.
- JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. JSON filenames use the extension .json
- JSON-RPC is a remote procedure call (RPC) protocol built on JSON, as a replacement for XML-RPC or SOAP. It is a simple protocol that defines only a handful of data types and commands. JSON-RPC lets a system send notifications (information to the server that does not require a response) and multiple calls to the server that can be answered out of order.
- Asynchronous JavaScript and JSON (or AJAJ) refers to the same dynamic web page methodology as Ajax, but instead of XML, JSON is the data format. AJAJ is a web development technique that provides for the ability of a webpage to request new data after it has loaded into the web browser. Typically it renders new data from the server in response to user actions on that webpage. For example, what the user types into a search box, client-side code then sends to the server, which immediately responds with a drop-down list of matching database items.

Source: Wikipedia

JSON (cont.)

- **Syntax Basics**
  - In JSON, data is structured in a specific way. JSON uses symbols like { } , : ” ” [ ] and it has the following syntax:
  - Data is represented in key/value pairs
  - The colon (:) assigns a value to key
  - key/value pairs are separated with commas (,)
  - Curly brackets hold objects ({ })
  - Square brackets hold arrays ([ ])
- For example, to represent data in JSON, the key/value pairs come as follows:

```
{"key1":"value1", "key2":"value2", "key3":"value3"}
```

- Here is a real world example of structuring user data:

```
{"name":"Rui", "country": "Portugal", "age":24}
```

- Or in a IoT project, you may want to structure data from your sensors:

```
{"temperature":27.23, "humidity":62.05, "pressure":1013.25}
```

Source: https://randomnerdtutorials.com/decoding-and-encoding-json-with-arduino-or-esp8266/

JSON And APIs

Most APIs return data in JSON and most values are JSON objects themselves. The following example shows the data provided by a weather API:

```
{
  "coord":{
     "lon":-8.61,
     "lat":41.15
  },
  "weather":[
    {
      "id":803,
      "main":"Clouds",
      "description":"broken clouds",
      "icon":"04d"
    }
  ],
  "base":"stations",
  "main":{
    "temp":288.15,
    "pressure":1020,
    "humidity":93,
    "temp_min":288.15,
    "temp_max":288.15
  },
  (...)
}
```

Source: https://randomnerdtutorials.com/decoding-and-encoding-json-with-arduino-or-esp8266/

JSON and Arduino

- The easiest way to decode and encode JSON strings with the Arduino IDE is using the ArduinoJson library 5.13.5 which was designed to be the most intuitive JSON library, with the smallest footprint and most efficiently memory management for Arduino.
- It has been written with Arduino in mind, but it isn't linked to Arduino libraries so you can use this library in any other C++ project. There's also a documentation website for the library with examples and with the API reference.
- Features:
  - JSON decoding (comments are supported)
  - JSON encoding (with optional indentation)
  - Elegant API, very easy to use
  - Fixed memory allocation (zero malloc)
  - No data duplication (zero copy)
  - Portable (written in C++98)
  - Self-contained (no external dependency)
  - Small footprint
  - Header-only library
  - MIT License
- Compatible With:
  - Arduino boards: Uno, Due, Mini, Micro, Yun…
  - ESP8266, ESP32 and WeMos boards
  - Teensy, RedBearLab boards, Intel Edison and Galileo
  - PlatformIO, Particle and Energia

Source: https://randomnerdtutorials.com/decoding-and-encoding-json-with-arduino-or-esp8266/

The REST API (Review)

- REST stands for Representational State Transfer. REST defines a set of functions like GET, PUT, DELETE, etc. that clients can use to access server data. Clients and servers exchange data using HTTP.
- The main feature of REST API is statelessness. Statelessness means that servers do not save client data between requests. Client requests to the server are similar to URLs you type in your browser to visit a website. The response from the server is plain data, without the typical graphical rendering of a web page.
- A Web API or Web Service API is an application processing interface between a web server and web browser. All web services are APIs but not all APIs are web services. REST API is a special type of Web API that uses the standard architectural style explained above.
- The different terms around APIs, like Java API or service APIs, exist because historically, APIs were created before the world wide web. Modern web APIs are REST APIs and the terms can be used interchangeably.

Arduino REST API

- Arduino Rest API is a mechanism to exchange data between Arduino and other external systems. Using Arduino Rest API framework it is possible to control Arduino remotely.
- Generally speaking, this kind of mechanism is useful when an external application (client) sends a request to Arduino and it replies with some data.
- Arduino Rest API works over HTTP protocol so this kind of requests are synchronous.
- In IoT applications, there are other protocols that can be used like MQTT.
- Arduino API over HTTP plays an important role in a client-server scenario where Arduino acts as a server. MQTT, for example, uses a different pattern like publish-subscriber.

Arduino REST API (cont.)

- To implement a Rest API framework there is an interesting library called aRest. This library is a framework that supports Restful services and provides several interesting features. This library supports different dev boards like Arduino, Raspberry, ES8266.
- This library is simple to use and can be downloaded directly from the Arduino library through Arduino IDE.
- Using this library we can implement the API because of aRest support:
  - reading pin values in rest style
  - writing pin values in rest style
  - Remote sketch function call
- For example, an external application or system can read the Arduino pin value using a simple HTTP request. Moreover, the same app or system can set the pin value using an HTTP Rest request. This is useful when we have an app that runs on a smartphone that wants to interact with Arduino board. This interaction takes place exploiting Arduino Rest API.
- One interesting aspect of aRest library is the capability to expose the Arduino sketch function in a Rest style. These sketch functions are called directly using a Rest HTTP request. Let us see how to use HTTP API in practice with Arduino.

Formal End of Lesson 9

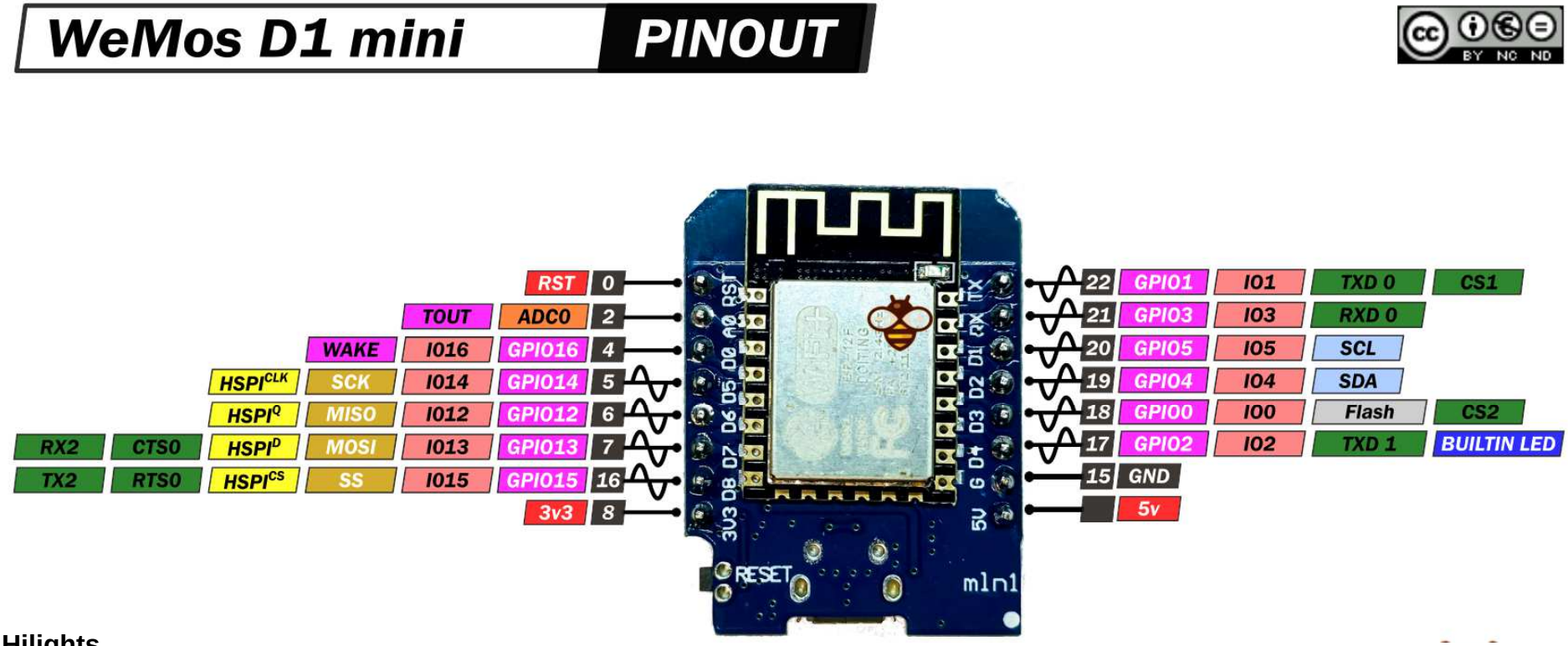
HOMEWORK

- **No Homework This Week**

In next week's exciting episode

- Scraping information from an Internet web page
- Obtaining weather information from the Internet
- Class Exercise: Getting and displaying weather information

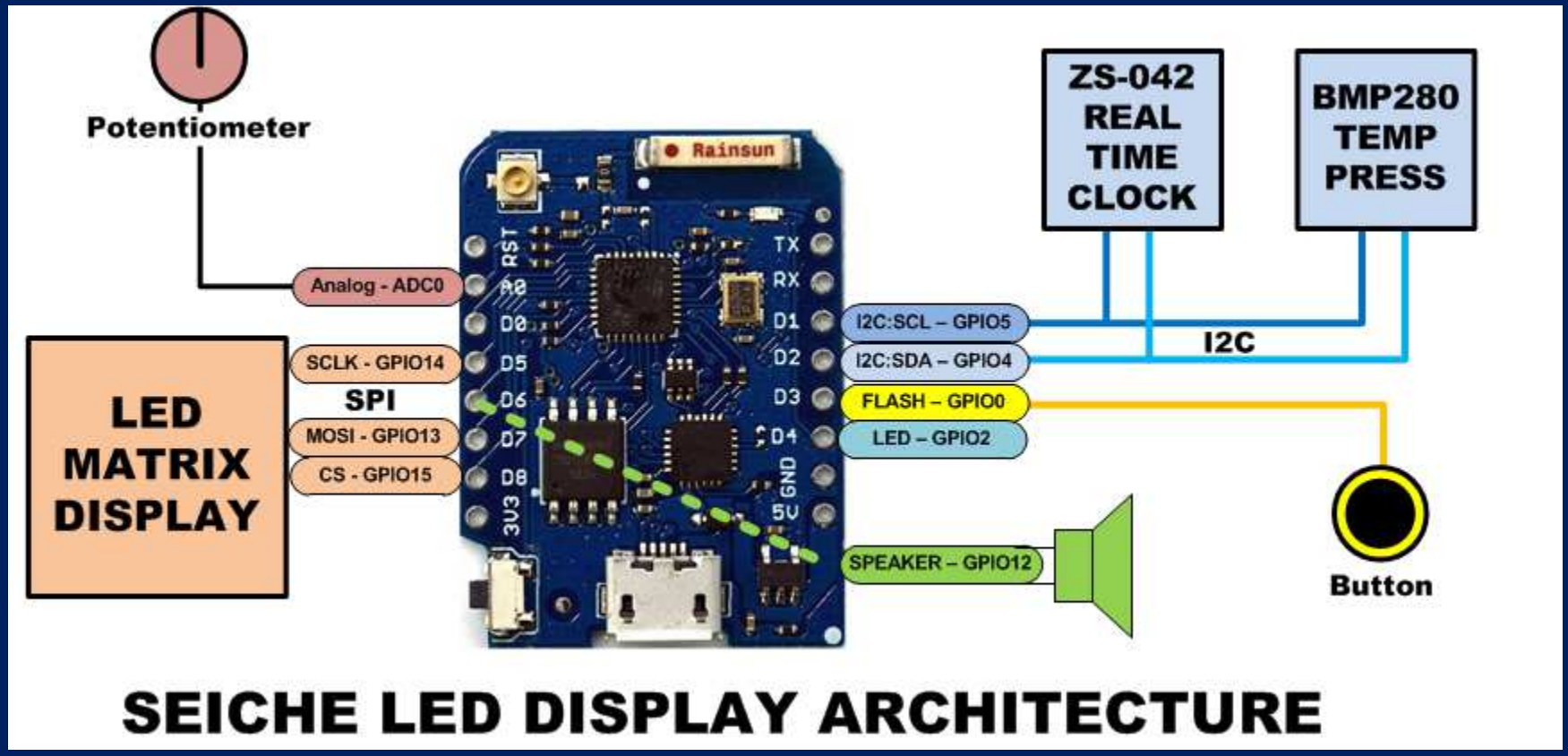
Our Microcontroller: The WeMos D1 Mini



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