IoT Bootcamp - Virtual Lessons

Brian Rashap

March 31, 2020

Contents

Preface **8**

1 Virtual Lesson 1 **9**

1.1 Tutorial 1 9

1.1.1 Data Types 9

1.1.2 Boolean 10

1.1.3 While Loops 12

1.2 Lesson 1 Assignments 12

1.2.1 Coding 12

1.2.2 Fusion 360 13

1.2.3 Product Planning 13

2 Virtual Lesson 2 **15**

2.1 Tutorial 2 15

2.1.1 Serial Input 15

2.1.2 Switch Case 17

2.2 Lesson 2 Assignments 18
2.2.1 Coding 18
2.2.2 Fusion 360 19
2.2.3 Product Planning 19

3 Virtual Lesson 3 **20**
3.1 Tutorial 3 20
3.1.1 Functions 20
3.1.2 Random Numbers 20
3.1.3 Arrays 22
3.2 Lesson 3 Assignments 24
3.2.1 Coding 24
3.2.2 Fusion 360 25
3.2.3 Fritzing 26

4 Virtual Lesson 4 **28**
4.1 Simon 28
4.2 OLED Display Revisited 29
4.3 Virtual 4 Assignments 30
4.3.1 Coding 30
4.3.2 Fusion 360 30

5 Virtual 5 Lessons **32**

5.1 Virtual 5 Assignment 32
5.2 Smart Room Controller 35
5.2.1 Student Project Repository 35
5.2.2 Smart Room Controller Specifications 36

6 Virtual Lesson 6 - UNDER CONSTRUCTION 37
6.1 Visual Studio Code 37

APPENDICES 39

A Functions 39

B Number Systems used by computers 43
B.1 Bases 43
B.2 Binary 44
B.3 Binary: Counting and Converting 44
B.4 Converting from DEC to BIN 45
B.5 Bits, Nibbles, and Bytes 46
B.6 Hexidecimal Basics 47
B.7 Counting in HEX 48
B.8 HEX identifiers 48
B.9 Converting Hex to Decimal 50

C GITHUB 51

C.1 Cloning an existing repository 51

C.2 Getting the latest updates 51

C.3 Placing your edits into the repository 51

D Lineage of Programming Languages 53

D.1 FORTRAN 53

D.2 ALGOL 54

D.3 CPL 55

D.4 BCPL 55

D.5 B 56

D.6 C 57

D.7 C++ 57

D.8 Python 58

D.9 PHP 59

List of Figures

2.1 Serial Input 17

3.1 Example Board 26

3.2 Importing Part into Fritzing 27

4.1 Simon ® Flow Chart 31

5.1 Adafruit Dashboard 33

5.2 Controlling for Adafruit 33

5.3 ThingSpeak Dashboard 34

5.4 Smart City Dashboard 34

5.5 Smart City Lighting 35

6.1 Visual Studio Code 38

A.1 Anatomy of a C function 39

D.1 Program Language Lineage 54

List of Tables

B.1 Decimal to Binary Conversion 45

B.2 Bits, Nibbles, and Bytes 47

B.3 Convert Decimal to Hexidecimal 48

B.4 Hex Identifiers 49

List of Code Listings

2.1 Serial Input 16

3.1 Roll a Die 21

A.1 Ardunio Functions 41

A.2 Example Functions 42

Preface

The IoT Bootcamp by its very nature is ideally an hands-on and collaborative environment, over the next few weeks we will make the best of having to work together virtually.

In this document, which will be updated daily and posted to a new Git Classroom assignment, will be

- Tutorial of IoT coding or hardware concepts
- References for learning more about a topic
- Exercises to be completed each day. These will include:
  - Starter code that you will need to complete
  - Online Fusion 360 lessons
  - Product design and layout (Fritzing) assignments

Please save all your work in the Git Classroom repository that you downloaded morning. Remember to "push" your repository back to the Git Classroom at the end of the day. A reminder of Git functions can be found in Appendix C of this document.

Chapter 1

Virtual Lesson 1

1.1 Tutorial 1

1.1.1 Data Types

The Arduino environment is really just C++ with library support and built-in assumptions about the target environment to simplify the coding process. C++ defines a number of different data types.

The Teensy's ARM Cortex processor is a 32-bit processor (Arduino boards use a 16-bit processor). This is why the int data type on the Teensy is 4-bytes (32 bits / 8 bits per byte). As a result, the data types are different from an Arduino and in order to keep the code compatible, some data types are duplicates (i.e., on the Teensy int and long are the same). Also, igned variables allow both positive and negative numbers, while unsigned variables allow only positive values.

- bool (8 bit) - simple logical true/false
- byte (8 bit) - unsigned number from 0-255
- char (8 bit) - while technically a signed number from -128 to 127. The compiler will interpret this data type as a character

- unsigned char (8 bit) - same as 'byte'; if this is what you're after, you should use 'byte' instead, for reasons of clarity
- unisigned short (16 bit) - unsigned number from 0-65535
- short (16 bit) - signed number from -32768 to 32767.
- unsigned int (32 bit) - unsigned number from 0-4,294,967,295.
- int (32 bit) - signed number from -2,147,483,648 to 2,147,483,647. This is most commonly what you see used for general purpose variables in Arduino example code provided with the IDE.
- unsigned long (32 bit) - on the Teensy, same as unsigned int. The most common usage of this is to store the result of the millis() function, which returns the number of milliseconds the current code has been running
- long (32 bit) - on the Teensy, same as int.
- float (32 bit) - signed number from -3.4028235E38 to 3.4028235E38. Floating point on the Arduino is not native; the compiler has to jump through hoops to make it work. If you can avoid it, you should. We'll touch on this later.

1.1.2 Boolean

A data type: bool holds one of two values, either true or false. For example, you can declare a boolean variable called buttonstate and set it to true.

```
1 bool buttonState;
2
3 buttonState = true;
```

Boolean Operators

There are three boolean operators

- NOT
- AND
- OR

The logical NOT (!) results in a true if the operand is false and vice versa.

This operator can be used inside the condition of an if statement.

```
if (!x) {     // if x is not true
    // statements
}
```

or, it can be used to invert the boolean value.

```
x = false;
y = !x;  // the inverted value of x is stored in y,
     // so now = y = true
x = !x;  // invert x, x = true
```

The logical AND (&&) results in true only if both operands are true.

Example Code This operator can be used inside the condition of an if statement.

```
// if BOTH the pins read HIGH
if (digitalRead(Pin2) == HIGH && digitalRead(Pin3) == HIGH)
    // statements
}
```

The logical OR (||) results in a true if either of the two operands is true. This operator can be used inside the condition of an if statement.

```
// if either x or y is greater than zero
if (x > 0 || y > 0) {
  // statements
}
```

1.1.3 While Loops

A while loop will loop continuously, and infinitely, until the expression inside the parenthesis () becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor or waiting for a button to be pressed.

```
while (buttonState == false) {
  // statement(s)
  }
```

There is also a do..while loop that works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

```
do {
  // statement block
} while (temperature < 72);
```

Tomorrow we will learn about the case statements.

1.2 Lesson 1 Assignments

1.2.1 Coding

Please complete the below coding assignments. The starter code is in your git classroom clone. Make sure that you are saving back into this repository so that your code is sent back to the instructor when you "push" as the end of the day. (HINT: you need to push your repository at the end of each day).

NOTE If you have only one button, look at the fritzing file in your respository (ifonlyonebutton.fzz) to see how you can approximate a button for the first lesson. And, please Slack your instructor letting them know you need additional buttons.

For each of the three below assignments, the instructions are in the commenets at the top of the file. Before you start coding, draw a flowchart in your lab notebook to sketch out your logic.

1. V1_1_booleans
2. V1_2_timer
3. V1_3_timer_oneButton

1.2.2 Fusion 360

As part of product design, we will need to be proficient at using Fusion 360. While we got some exposure to using Fusion 360 in class during Week 2, we are going to go through the basics of Fusion 360 a second time to help us build proficiency.

Fusion 360 updated its user interface and many of the tutorials on the Autodesk website use the old interface. As we are still beginners, we will use the below tutorial which matches the user interface you'll see when using Fusion 360. In later Fusion 360 lessons, I will be including tutorials with the old interface, but at that time we should be proficient enough in the new interface to follow along.

For today's lesson, please complete the tutorial Fusion 360 for Absolute Beginners. $https://www.youtube.com/watch?v = qvrHuaHhqHI$

After completing the tutorial, please export ($File- > Export$) your .f3d file to your repository.

1.2.3 Product Planning

It is time to start to create the specifications for your Smart Room Controller. You are not laying out your board yet, nor are you coding. Today, you are just righting a description of what your controller will be able to do.

The minimum requirements are:

- Control at least one Hue smart bulb in the classroom
  - Be able to turn the bulb on and off
  - Be able to change the color of the bulb
  - Be able to dim the bulb (using the encoder)
- Control at least one WEMO smart outlet in the classroom
- The Neopixels should be used to give an indication of what is going on with both the bulb and the outlet.

Optional features you might consider

- Integration of a BME280
- Use of the OLED display
- Adding a sleep timer function to the WEMO control (i.e., the outlet turns off after either a predefined or user entered time period).

For today, you are to create a document that describes the functions that your Smart Room Controller will be able to do. You should document this in the myRoomController.txt file that is in your git classroom respository. In addition, you should list the components that you'll be using for your project. NOTE: if you pull the latest class_slides repository, there is a spreadsheet with all the parts that we have been using in the class. This might be helpful as you are thinking about your Smart Room Controller.

Chapter 2

Virtual Lesson 2

2.1 Tutorial 2

2.1.1 Serial Input

Up until now, we have only written to our computer screen. Through the serial monitor, we can also provide inputs to the Teensy.

First, let's remember that Serial is actual a class/object that we can apply methods (actions) to. When we type Serial.begin(9600), we are telling the Serial object to begin with a communication rate of 9600 bits per second.

Another action we can take with the Serial Monitor is to input characters from it. This can be accomplished with the Serial.available() command which gets the number of bytes (characters) available for reading from the serial port. This is data that has already arrived and stored in the serial receive buffer (which holds 64 bytes). Serial.read() is then the method that actual reads from the Serial Monitor, one byte (or character at a time).

The code to bring in characters from the Serial Monitor is listed in Code Listing 2.1. A few things to note with this code:

```
1  /*
2   * Project: serialInput
3   * Description: take characters from the Serial Input and
       convert to an integrer
4   * Auther: <your name>
5   * Date: <today's date>
6   */
7
8
9  /*  Note: A String is a class of objects that behave
10  *   like an array of char (individual characters)
11  *   And, because it is an object, as we will see below,
12  *   it has methods associated with it.
13  */
14
15 // ASSIGNMENT: type (don't cut/paste) the code from
       iot_virtual.pdf and run it
```

Code Listing 2.1: Serial Input

- Line 29 - keep looping as long as there is something to read from the buffer
- Line 30 - read from the Serial Monitor and place the byte/character into inChar.
- Line 32 - the isDigit() function is true if inChar is a number.
- Line 33 - if inChar is a number, convert to char using (char)inChar, otherwise skip it. That is, only numbers, not characters, are added to the string.
- Line 37 - so, how do we tell the Teensy that we hit enter. The "\n" is the byte that gets returned when "enter" is pressed.

Once you run the code, open the Serial Monitor. In the top line, above where the data from the Teensy is displayed, is where your inputs are entered. Then you press the "enter" key or push the "send" button to the right.

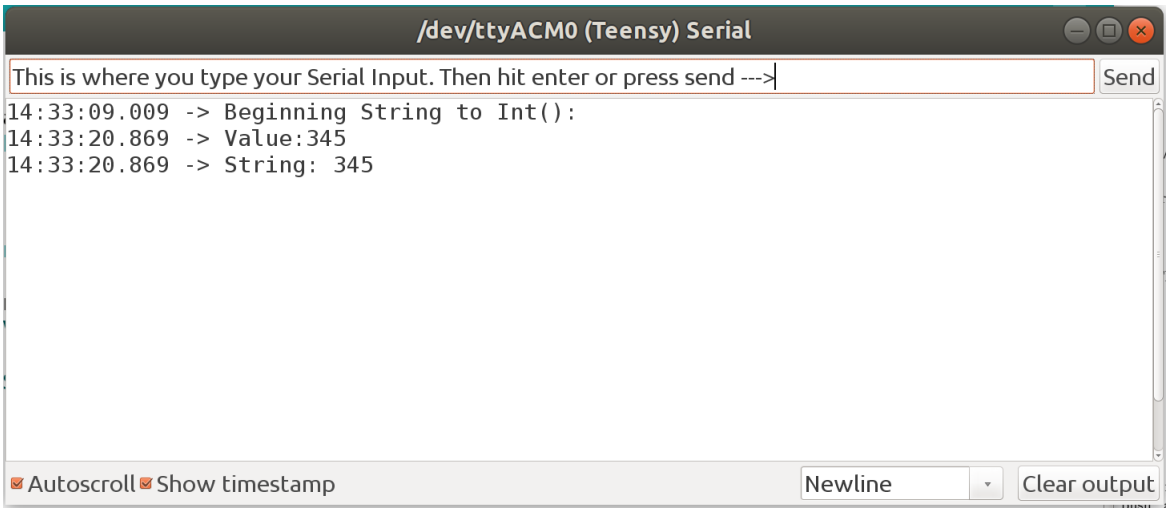


Figure 2.1: Serial Input

2.1.2 Switch Case

Like if statements, switch case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The "break" keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

```
switch (var) {
  case label1:
    // statements
    break;
  case label2:
    // statements
    break;
```

```
  default:
    // statements
    break;
}
```

In the above, "var" is a variable (either int or char) whose value to compare with various cases. And, "label1" and "label2" are constants (likewise, either int or char). For example:

```
int var    // variable used in swtich

switch (var) {
  case 1:
    //do something when var equals 1
    break;
  case 2:
    //do something when var equals 2
    break;
  default:
    // if nothing else matches, do the default
    // default is optional
    break;
}
```

2.2 Lesson 2 Assignments

2.2.1 Coding

Please complete the below coding assignments. The starter code is in your git classroom clone. Make sure that you are saving back into this repository so that your code is sent back to the instructor when you "push" as the end of the day. (HINT: you need to push your repository at the end of each day).

NOTE If you have only one button, please Slack your instructor letting them know you need additional buttons. We will all need 4 buttons for the next lessons.

For each of the three below assignments, the instructions are in the commenets at the top of the file. Before you start coding, draw a flowchart in your lab notebook to sketch out your logic.

1. V2_0_serialInput
2. V2_1_countdown
3. V1_2_switchcase

2.2.2 Fusion 360

For today's lesson, please complete the tutorial Fusion 360 for Absolute Beginners (2020) - Project # 2 at $https://youtu.be/XC-6AQksxHY$

After completing the tutorial, please export ($File->Export$) your .f3d file to your repository.

2.2.3 Product Planning

Continuing working on your Smart Room Controller planning, refining what you put together yesterday in Section 1.2.3.

Chapter 3

Virtual Lesson 3

3.1 Tutorial 3

3.1.1 Functions

We are going to get some more practice creating and using functions in today's lesson. Please review the details about functions in Appendix A.

3.1.2 Random Numbers

In the past lessons, we have occasionally used random numbers to make our code more interesting.

```
1 random(max)
2 random(min,max)
```

where min is the lower bound, inclusive, of the random value (it is assumed to be 0 if only max is used), and max is the upper bound, exclusive. This means that the function returns a random integer between min and (max-1). An example that uses the random function can be found in Code Listing 3.1.

```
/*
 * Project: Dice
 * Description: using switch case
 */
int die;
void setup() {
  Serial.begin(9600);
  while(!Serial);
}
void loop() {
  die = random(0,7); // return die roll between 0 and 6
  Serial.println(die);
  switch(die) {
    case 1:
      Serial.println("A one was rolled");
      break;
    case 2:
      Serial.println("A two was rolled");
      break;
    case 3:
      Serial.println("A three was rolled");
      break;
    case 4:
      Serial.println("A four was rolled");
      break;
    case 5:
      Serial.println("A five was rolled");
      break;
    case 6:
      Serial.println("A six was rolled");
      break;
    default:
      Serial.println("Illegal Die was Used");
      break;
  }
  delay(500);
}
```

Code Listing 3.1: Roll a Die

You may notice that every time you restart your program it repeats the same sequence of random numbers. The *random*() is not actually random, but is actually a pseudo-random number generator (PRNG). The PRNG uses an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of truly random numbers. A seed value is uses to start this pseudo-random sequence.

```
randomSeed(seed);
```

where *seed* is an unsigned long variable containing the seed value.

If we utilize the a fixed seed value, then our program will always run through the same sequence of pseudo-random numbers. This ca be useful for debugging our programs. However, if we want a new pseudo-random sequence every time we start our code, we can take advantage of a floating input. Recall, we used pull-down and pull-up with our buttons to ensure they were in a known state (known voltage) when not pressed. Now we can take advantage of the floating input to seed our PRNG. Using a pin that is not connected to anything, we can use *analogRead*() to see out random function.

```
randomSeed(analogRead(0));
```

where Pin 0 is left floating (not connected to power or ground).

3.1.3 Arrays

An array is a collection of variables that are accessed with an index number. Arrays in the C++ programming language are written in can be complicated, but using simple arrays is relatively straightforward.

There are several different ways to declare an array.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

- You can declare an array without initializing it as in myInts.

- In myPins we declare an array without explicitly choosing a size. The compiler counts the elements and creates an array of the appropriate size.
- You can both initialize and size your array, as in mySensVals.
- For an array of characters (char), one more element than your initialization is required, to hold the required null character. So, the character array message needs to have a size of 6, five for the letters H-E-L-L-O, and one for the null character that ends the array.

To assign a value to an array:

```
mySensVals[0] = 10;
```

To retrieve a value from an array:

```
x = mySensVals[4];
```

Arrays and FOR Loops Arrays are often manipulated inside for loops, where the loop counter is used as the index for each array element. For example, to print the elements of an array over the serial port, you could do something like this:

```
for (byte i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

CAUTION: Arrays are zero indexed, that is, referring to the array initialization above, the first element of the array is at index 0. This means that in an array with ten elements, index nine is the last element. Hence:

```
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};
x = myArray[9];    // contains 11
y = myArray[10];   // is invalid and contains random
   information (other memory address)
```

For this reason you should be careful in accessing arrays. Accessing past the end of an array (using an index number greater than your declared array size - 1) is reading from memory that is in use for other purposes. Reading from these locations is probably not going to do much except yield invalid data. Writing to random memory locations is definitely a bad idea and can often lead to unhappy results such as crashes or program malfunction. This can also be a difficult bug to track down.

3.2 Lesson 3 Assignments

Before you start these current assignments, it might be a good time to clean up your file structure. If you saved files from the class outside of the GIT Classroom Repositories, then you can do some clean up by moving them to the appropriate locations. Also, please make sure that all your repositories from the in-person classes and the previous 2 virtual lessons have be pushed back to GIT Classroom. Instructions are in Appendix C.

3.2.1 Coding

Please complete the below coding assignments. The starter code is in your git classroom clone. Make sure that you are saving back into this repository so that your code is sent back to the instructor when you "push" at the end of the day. Don't forget to do the push. And, feel free to push after each assignment.

For each of the two below assignments, the instructions are in the comments at the top of the file. Before you start coding, draw a flowchart in your lab notebook to sketch out your logic.

1. V3_0_functions
2. V3_1_fillanArray

3.2.2 Fusion 360

Over the next week, you should work through these Fusion 360 tutorials. The first two you already did as part of Virtual Lesson 1 and 2. I include the links, but you can also just Google the name of the video and it should come up.

- Fusion 360 for Absolute Beginners at:

  `https://www.youtube.com/watch?v=qvrHuaHhqHI`

- Fusion 360 for Absolute Beginners (2020) - Project # 2 at:

  `https://youtu.be/XC-6AQksxHY`

- Fusion 360 Tutorial for Makers (2020) - Custom Name Plate at: `https://www.youtube.com/watch?v=ju247tQHKso`

- How to 3D Model a Lego Brick - Learn Autodesk Fusion 360 in 30 Days: Day #1 (REVISED) at:

  `https://www.youtube.com/watch?v=6yPKMSb6ja8`

After completing each tutorial, please export ($File- > Export$) your .f3d file to your repository.

3.2.3 Fritzing

Please create a Fritzing layout with the following elements and then wire this onto your board.

- The Teensy
- Four NeoPixels
- Four Buttons

As always, before you start your Fritzing layout, please draw the layout in your lab notebook. There is a starter fritzing file called buttons4neo4.fzz in your GIT Classroom respository pull for this lesson.

As a suggestion for things to come, it would make some sense if your buttons are in the proximity of the individual NeoPixels. While you don't have to copy my layout, I have included a picture of the instructor board of it in Figure 3.1 as a reference.

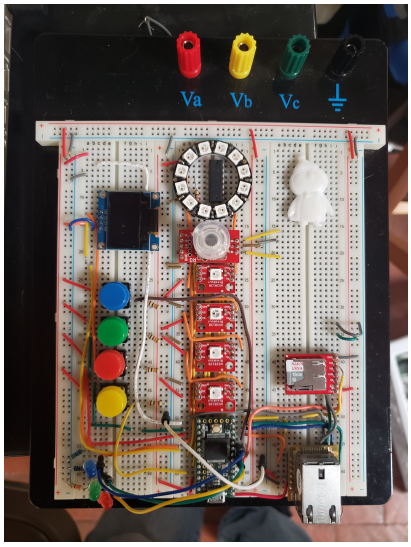


Figure 3.1: Example Board

Also, there are four button parts that are included in the Virtual 3 repository. Please import them into you Fritzing using the "Import" function. It can be accessed by clicking the 4 bars near the "search bar" in the Parts Window located by the arrow in Figure 3.2. And then you can access the parts by selecting the MINE tab at the left of the Parts Window.

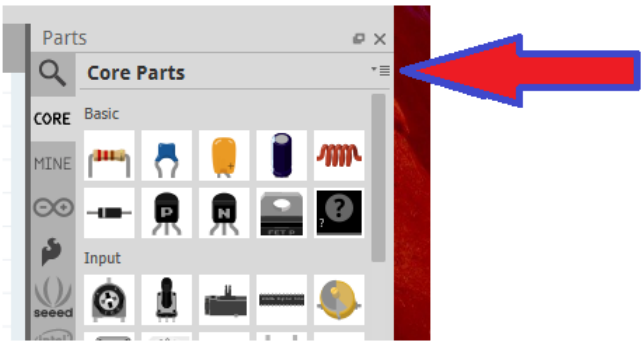


Figure 3.2: Importing Part into Fritzing

Chapter 4

Virtual Lesson 4

This week we are going to focus on integrating concepts that we have already learned. We will build and play the game Simon® and code up our Smart Room Controller. There are a few new concepts that will be introduced, but we will mostly focus on integration.

In Section 3.1.2, we were introduced to random numbers. This section has been updated, so please go back an learn about *randomSeed*().

4.1 Simon

Simon® is an electronic game of memory skill invented by Ralph H. Baer and Howard J. Morrison, working for toy design firm Marvin Glass and Associates, with software programming by Lenny Cope. The device creates a series of tones and lights and requires a user to repeat the sequence. If the user succeeds, the series becomes progressively longer and more complex. Once the user fails or the time limit runs out, the game is over. The original version was manufactured and distributed by Milton Bradley. If you are not familiar with Simon®, please check out this video:

`https://www.youtube.com/watch?v=PK7zc28IGPc`

4.2 OLED Display Revisited

Up until now, we have been using the ACROBOTIC_SSD1306 library with our OLED diplay. This library is very simple, but also very limited. It only allows characters of one size in 8 rows by 16 characters. An alternative library is the Adafruit_SSD1306 library (and the Adafruit_GFX graphics library). This library gives us full control of the 128x32 pixels on your OLED display and includes the ability to:

- Change font size
- Invert the screen (black on while)
- Draw lines, circles, triangles, and squares
- Scroll text
- Display bitmap images[1]

Here is some example code:

```
display.clearDisplay();

display.setTextSize(1);             // Normal 1:1 pixel scale
display.setTextColor(SSD1306_WHITE);// Draw white text
display.setCursor(0,0);       // Start at top-left corner
display.println(F("Hello, world!"));

display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
display.println(3.141592);

display.setTextSize(2);             // Draw 2X-scale text
display.setTextColor(SSD1306_WHITE);
display.print(F("0x")); display.println(0xDEADBEEF, HEX);

display.display();
```

[1] A bitmap describes a type of image that web-users encounter all the time without realizing it. Basically, it's a grid where each individual square is a pixel that contains color information. The key characteristics are the number of pixels (or squares in the grid), and the amount of information in each grid square (pixel).

4.3 Virtual 4 Assignments

4.3.1 Coding

Please complete the below coding assignments. The starter code is in your git classroom clone. Make sure that you are saving back into this repository so that your code is sent back to the instructor when you "push" at the end of the day. Don't forget to do the push. And, feel free to push after each assignment.

For each of the two below assignments, the instructions are in the comments at the top of the file. For the first coding assignment, before you start coding, draw a flowchart in your lab notebook to sketch out your logic. The second assignment will have us modify some existing code to utilize the Adafruit_SSD1306 library.

1. V04_01_UserArray
2. V04_02_BMErevisited: for this assignment, you will use Example $->$ Adafruit_SSD1306 $->$ ssd1306_128x32_i2c to learn about this library
3. V04_03_Simon: there is no starter code for this. Use your previous code and the flow chart in Figure 4.1 to code the Simon® game. Save in the Virtual-04 repository.

4.3.2 Fusion 360

We will continue to our Fusion 360 learning by completing the below tutorial.

How to 3D Model an Ice Cube Tray - Learn Autodesk Fusion 360 in 30 Days: Day #5 at: `https://www.youtube.com/watch?v=qvrHuaHhqHI`

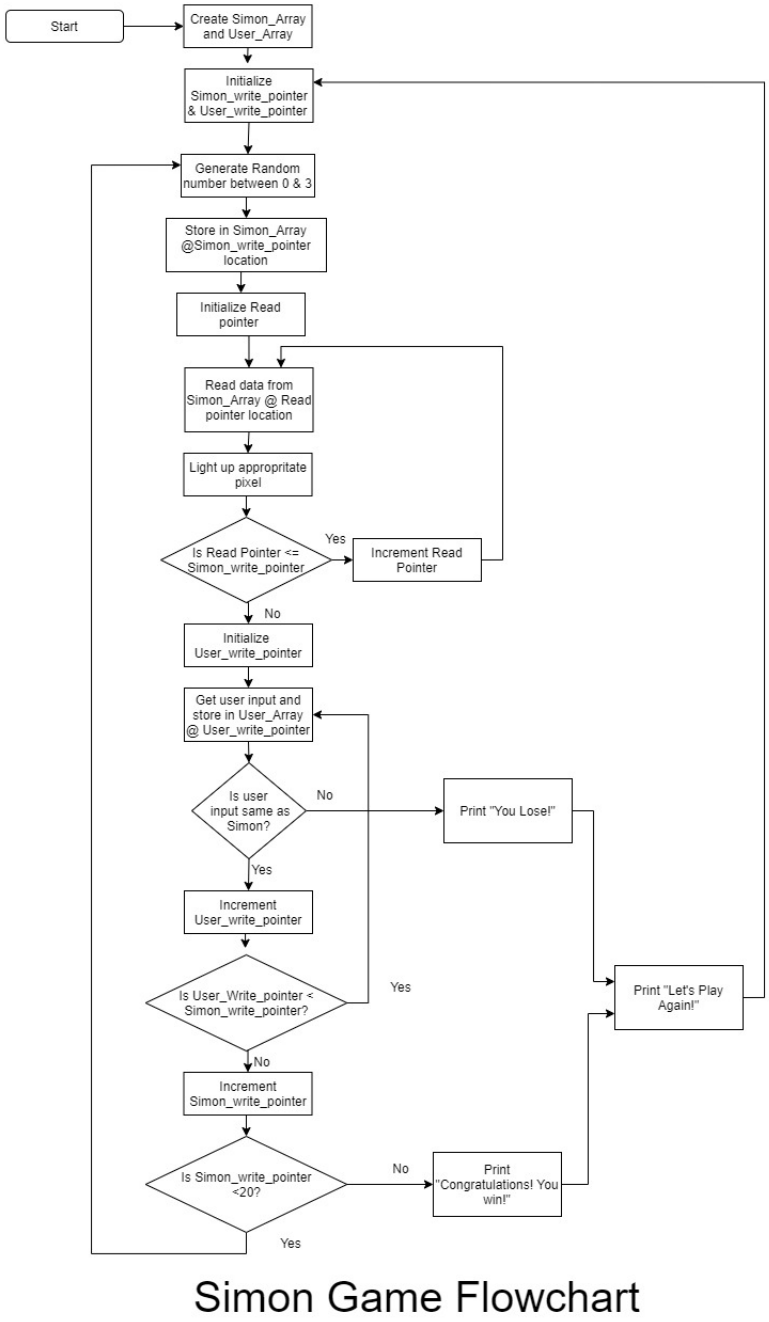


Figure 4.1: Simon ® Flow Chart

Chapter 5

Virtual 5 Lessons

Today we will be continuing with integration. You should be working on your Simon® game from Virtual Lesson 4.

In addition, today we have a special guest, Molly Blumhoefer, CNM's Sustainability Manager. She will be showing us a few of CNM's smart facilities dashboard. As preparation for this, I have included in this section a preview of some of the dashboards that we will be working with once we transition to the Particle Argon microcontroller. The can be seen in Figures 5.1 through 5.5.

5.1 Virtual 5 Assignment

1. Finish coding assignments from Lesson 4
2. Start Coding your Smart Room Controller, see the following section.

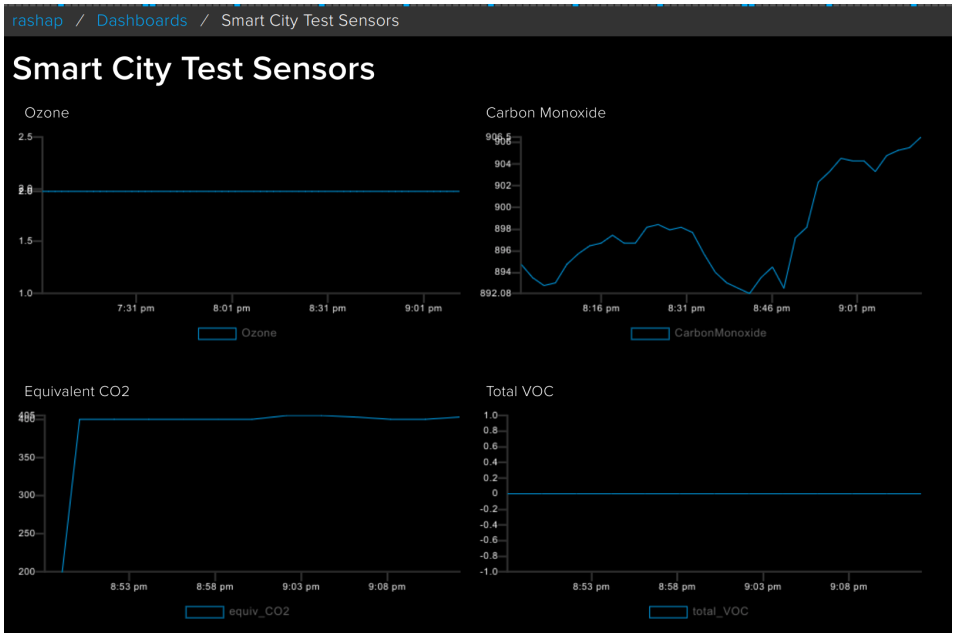


Figure 5.1: Adafruit Dashboard

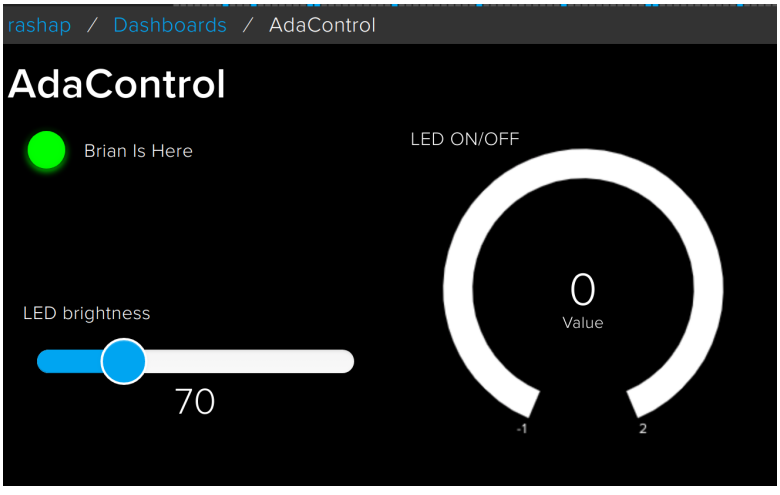


Figure 5.2: Controlling for Adafruit

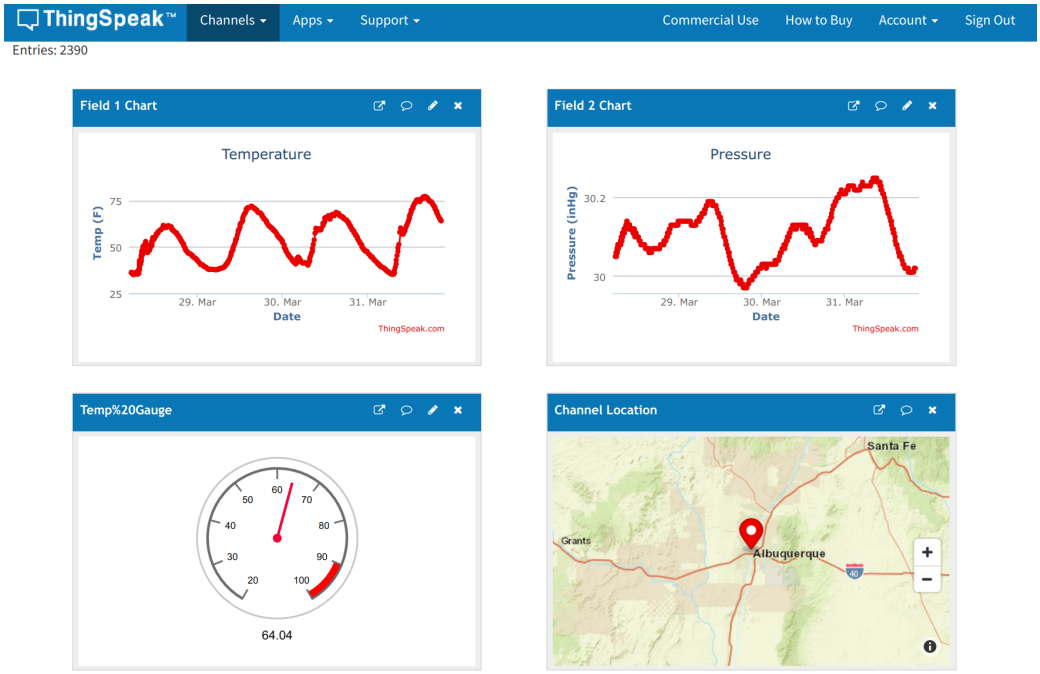


Figure 5.3: ThingSpeak Dashboard

Figure 5.4: Smart City Dashboard

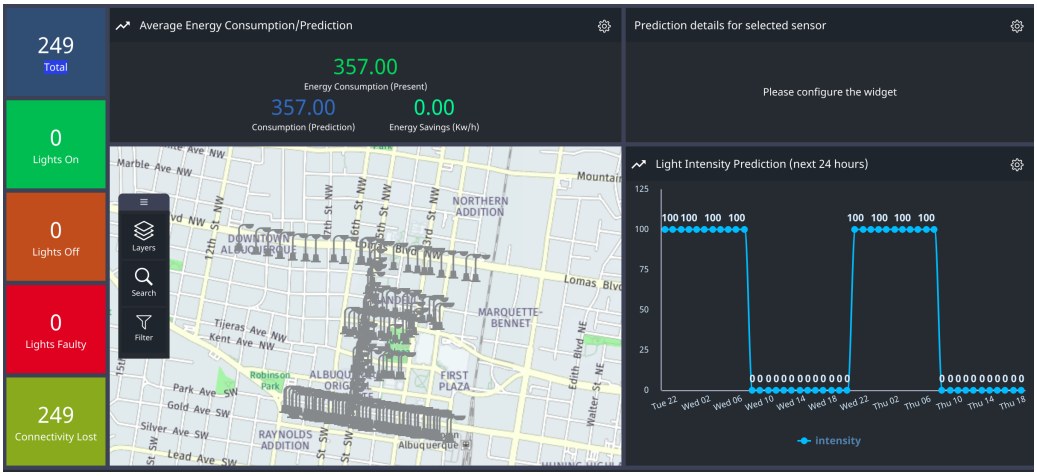


Figure 5.5: Smart City Lighting

5.2 Smart Room Controller

5.2.1 Student Project Repository

I have setup a GIT Classroom repository for each of you to store notes, flow charts, code, Fusion 360 models, etc. for your individual projects. Please go to `https://classroom.github.com/a/u8ljFbOJ` accept this respository and clone it in your Documents $->$ $<$ *yourname* $>$ directory.

Some of you completed the assignment from Virtual Lesson 1 to define the specifications for your Smart Room Controller. There are some fairly innovative ideas. For those of you that haven't had a chance create specifications, please use the specifications in the next section.

Given that we don't have access to the IoT Classroom with the Hue Smart Hub and the Wemo outlets we are going to simulate this portion. We will use two of our NeoPixels to be indicators for a light and outlets. In order to do this, you will need to create four functions that we will replace with actual code when we get back into the classroom.

- void SetHue(color, brightness) - turn the Hue "NeoPixel" to the required color and brightness
- void GetHue() - this can be a blank function that you call each time after you call SetHue. It is a nuance of the Hue API.
- void HueOn(outlet) - turn the Wemo "NeoPixel" to blue for the first outlet, yellow for the second outlet, and green for both outlets.
- void HueOff(outlet) - turn off the appropriate "outlet."

5.2.2 Smart Room Controller Specifications

Your Smart Room Controller should have, at least, the following functionality.

- Encoder button turns the "Hue light" on and off. The Encoder LED should indicate on (Green) or off (Red).
- Use the Encoder as a dimmer to change the brightness of the "Hue Light" from 0 to 255. The Pixel Ring should also show the brightness level. You should research the *map*() function as an easy way to map the 96 encoder positions to the 256 brightness levels of the "Hue Light." `https://www.arduino.cc/reference/en/language/functions/math/map/`
- Use one of the buttons to change the color of the "Hue Light" cycling through the colors of the rainbow. The Pixel Ring should change color along with the "Hue Light."
- Use the other button to turn on and off the "Wemo outlets." One click for on/off. Double click to select between the two "Wemo Outlets."
- OPTIONAL: Use the OLED Display to assist the user by showing the controller settings.

Chapter 6

Virtual Lesson 6 - UNDER CONSTRUCTION

In this lession we will prepare for transitioning to the Particle Argon microcontroller by installing Visual Studio Code and the Particle CLI.

6.1 Particle Login

6.2 Visual Studio Code

6.3 Particle CLI

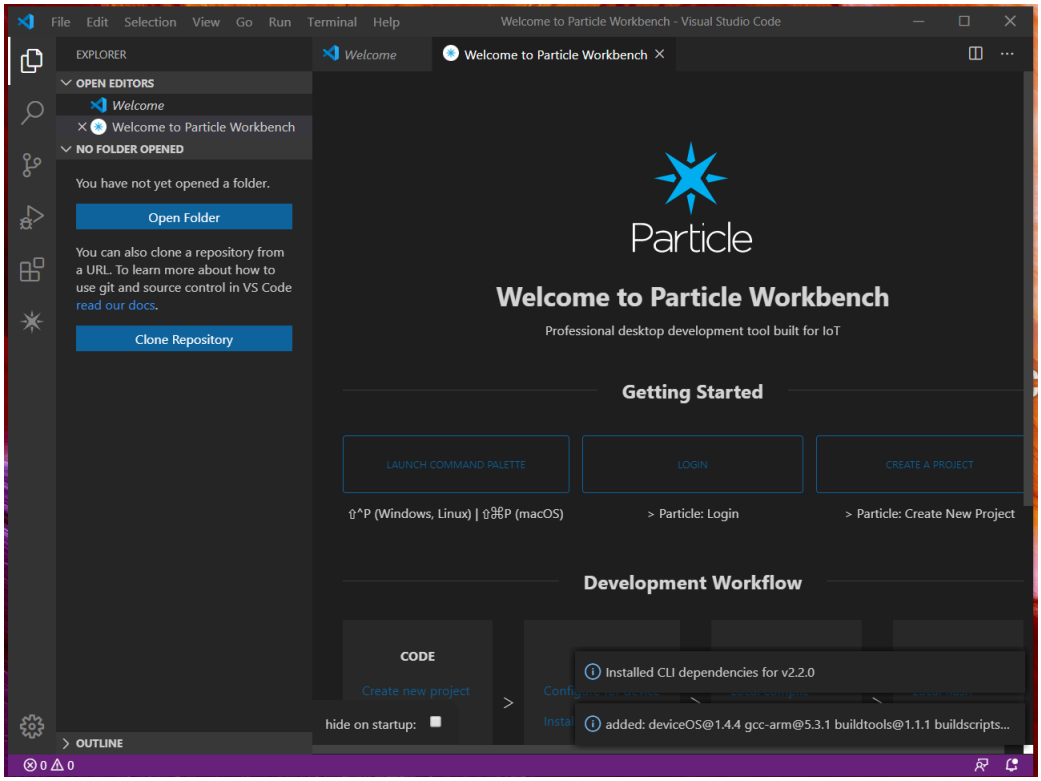


Figure 6.1: Visual Studio Code

Appendix A

Functions

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.

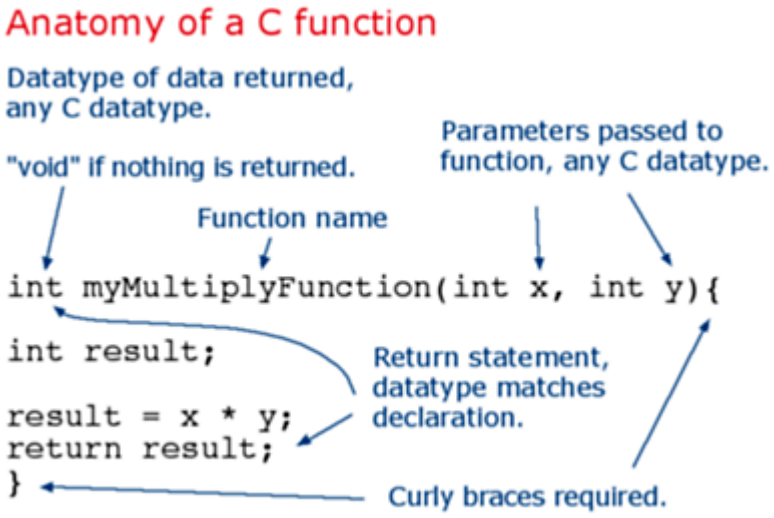


Figure A.1: Anatomy of a C function

Standardizing code fragments into functions has several advantages:

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought out and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.
- There are two required functions in an Arduino sketch, setup() and loop(). Other functions must be created outside the brackets of those two functions. As an example, we will create a simple function to multiply two numbers.

There are two required functions in an Arduino sketch, setup() and loop(). Other functions must be created outside the brackets of those two functions. As an example, we will create a simple function to multiply two numbers.

In Code Listing A.1, we create a function that needs to be declared outside any other function, so "myMultiplyFunction()" can go either above or below the "loop()" function.

To "call" our simple multiply function, we pass it parameters of the datatype that it is expecting. In our case k = myMultiplyFunction(i, j);

This function will read a sensor five times with analogRead() and calculate the average of five readings. It then scales the data to 8 bits (0-255), and inverts it, returning the inverted result. As you can see, even if a function does not have parameters and no returns is expected "(" and ")" brackets plus ";" must be given.

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

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

Code Listing A.1: Ardunio Functions

As you recall, we used a Function to convert from bits to voltage in our analog_read code that we wrote the first week. A version of this code can be found in Code Listing A.2.

```
/*
 * Project: Analog_Input
 * Description: Introduce Analog Read
 * Author: Brian A Rashap
 * Date: 11-Dec-2019
 */

// Connect the rotary resistor and another resistor in Series
// Connect the middle of these two resistors to Pin 14

int inPin = 14;
int inputDelay = 3000;
int inVal;
float voltage;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while(!Serial);

  pinMode(inPin, INPUT);
}

void loop() {
  inVal = analogRead(inPin);
  Serial.println(inVal);
  voltage = bits2volts(inVal);
  Serial.println(voltage);
  delay(inputDelay);
}

float bits2volts(int bitVal) {
  float voltVal;
  voltVal = bitVal*(3.3/1028);
  return voltVal;
}
```

Code Listing A.2: Example Functions

Appendix B

Number Systems used by computers

B.1 Bases

Number systems are the methods we use to represent numbers. Since grade school, we've all been mostly operating within the comfy confines of a base-10 number system, but there are many others. Base-2, base-8, base-16, base-20, base...you get the point. There are an infinite variety of base-number systems out there, but only a few are especially important to electrical engineering.

The really popular number systems even have their own name. Base-10, for example, is commonly referred to as the decimal number system. Base-2, which we're here to talk about today, also goes by the moniker of binary. Another popular numeral system, base-16, is called hexadecimal.

The base of a number is often represented by a subscripted integer trailing a value. So in the introduction above, the first image would actually be 10010 somethings while the second image would be 100_2 somethings. This is a handy way to specify a number's base when there's ever any possibility of ambiguity.

B.2 Binary

Why binary you ask? Well, why decimal? We've been using decimal forever and have mostly taken for granted the reason we settled on the base-10 number system for our everyday number needs. Maybe it's because we have 10 fingers, or maybe it's just because the Romans forced it upon their ancient subjugates. Regardless of what lead to it, tricks we've learned along the way have solidified base-10's place in our heart; everyone can count by 10's. We even round large numbers to the nearest multiple of 10. We're obsessed with 10!

Computers and electronics are rather limited in the finger-and-toe department. At the lowest level, they really only have two ways to represent the state of anything: ON or OFF, high or low, 1 or 0. And so, almost all electronics rely on a base-2 number system to store, manipulate, and math numbers.

The heavy reliance electronics place on binary numbers means it's important to know how the base-2 number system works. You'll commonly encounter binary, or its cousins, like hexadecimal, all over computer programs. Analysis of Digital logic circuits and other very low-level electronics also requires heavy use of binary.

In this tutorial, you'll find that anything you can do to a decimal number can also be done to a binary number. Some operations may be even easier to do on a binary number (though others can be more painful).

B.3 Binary: Counting and Converting

The base of each number system is also called the radix. The radix of a decimal number is ten, and the radix of binary is two. The radix determines how many different symbols are required in order to flesh out a number system. In our decimal number system we've got 10 numeral representations for values between nothing and ten somethings: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Each of those symbols represents a very specific, standardized value.

In binary we're only allowed two symbols: 0 and 1. But using those two symbols we can create any number that a decimal system can.

Counting in binary You can count in decimal endlessly, even in your sleep, but how would you count in binary? Zero and one in base-two should look pretty familiar: 0 and 1. From there things get decidedly binary.

Remember that we've only got those two digits, so as we do in decimal, when we run out of symbols we've got to shift one column to the left, add a 1, and turn all of the digits to right to 0. So after 1 we get 10, then 11, then 100. Let's start counting...

Dec	Bin		Dec	Bin
0	0		16	10000
1	1		17	10001
2	10		18	10010
3	11		19	10011
4	100		20	10100
5	101		21	10101
6	110		22	10110
7	111		23	10111
8	1000		24	11000
9	1001		25	11001
10	1010		26	11010
11	1011		27	11011
12	1100		28	11100
13	1101		29	11101
14	1110		30	11110
15	1111		31	11111

Table B.1: Decimal to Binary Conversion

B.4 Converting from DEC to BIN

There's a handy function we can use to convert any binary number to decimal:

Binary to decimal conversion equation There are four important elements to that equation:

$a_n 2^n + a_{n-1} 2^{n-1} + ... + a_1 2^1 + a_0 2^0$

There are four important elements to that equation:

- a_n, a_{n-1}, a_1, etc., are the digits of a number. These are the 0's and 1's you're familiar with, but in binary they can only be 0 or 1.
- The position of a digit is also important to observe. The position starts at 0, on the right-most digit; this 1 or 0 is the least-significant. Every digit you move to the left increases in significance, and also increases the position by 1.
- The length of a binary number is given by the value of n, actually it's n+1. For example, a binary number like 101 has a length of 3, something larger, like 10011110 has a length of 8.
- Each digit is multiplied by a weight: the 2^n, 2^{n-1}, 2^1, etc. The right-most weight - 2^0 equates to 1, move one digit to the left and the weight becomes 2, then 4, 8, 16, 32, 64, 128, 256,... and on and on. Powers of two are of great importance to binary, they quickly become very familiar.

B.5 Bits, Nibbles, and Bytes

In discussing the make of a binary number, we briefly covered the length of the number. The length of a binary number is the amount of 1's and 0's it has.

Common binary number lengths Binary values are often grouped into a common length of 1's and 0's, this number of digits is called the length of a number. Common bit-lengths of binary numbers include bits, nibbles, and bytes (hungry yet?). Each 1 or 0 in a binary number is called a bit. From there, a group of 4 bits is called a nibble, and 8-bits makes a byte.

Bytes are a pretty common buzzword when working in binary. Processors are all built to work with a set length of bits, which is usually this length is a multiple of a byte: 8, 16, 32, 64, etc.

Length	Name	Example
1	Bit	0
4	Nibble	1011
8	Byte	10110101

Table B.2: Bits, Nibbles, and Bytes

Word is another length buzzword that gets thrown out from time-to-time. Word is much less yummy sounding and much more ambiguous. The length of a word is usually dependent on the architecture of a processor. It could be 16-bits, 32, 64, or even more.

B.6 Hexidecimal Basics

Hexadecimal – also known as hex or base 16 – is a system we can use to write and share numerical values. In that way it's no different than the most famous of numeral systems (the one we use every day): decimal. Decimal is a base 10 number system (perfect for beings with 10 fingers), and it uses a collection of 10 unique digits, which can be combined to positionally represent numbers.

Hex, like decimal, combines a set of digits to create large numbers. It just so happens that hex uses a set of 16 unique digits. Hex uses the standard 0-9, but it also incorporates six digits you wouldn't usually expect to see creating numbers: A, B, C, D, E, and F.

Hex, along with decimal and binary, is one of the most commonly encountered numeral systems in the world of electronics and programming. It's important to understand how hex works, because, in many cases, it makes more sense to represent a number in base 16 than with binary or decimal.

B.7 Counting in HEX

Counting in hex is a lot like counting in decimal, except there are six more digits to deal with. Once a digit place becomes greater than "F", you roll that place over to "0", and increment the digit to the left by 1.

DEC	HEX		DEC	HEX		DEC	HEX		DEC	HEX
0	0		8	8		16	10		24	8
1	1		9	9		17	11		25	19
2	2		10	A		18	12		26	1A
3	3		11	B		19	13		27	1B
4	4		12	C		20	14		28	1C
5	5		13	D		21	15		29	1D
6	6		14	E		22	16		30	1E
7	7		15	F		23	17		31	1F

Table B.3: Convert Decimal to Hexidecimal

B.8 HEX identifiers

"BEEF, it's what's for dinner". Am I channelling my inner Sam Elliott (McConaughey?), or expressing my hunger for the decimal number 48879? To avoid confusing situations like that, you'll usually see a hexadecimal number prefixed (or suffixed) with one of these identifiers:

There are a variety of other prefixes and suffixes that are specific to certain programming languages. Assembly languagues, for example, might use an "H" or "h" suffix (e.g. 7Fh) or a ""$prefix$(6AD). Consult examples if you're not sure which prefix or suffix to use with your programming language.

The "0x" prefix is one you'll see a lot, especially if you're doing any Arduino programming. We'll use that from now on in this tutorial.

In summary: DECAF? A horrible abomination of coffee. 0xDECAF? A perfectly acceptable, 5-digit hexadecimal number.

Identifier	**Example**	**Notes**
0x	0x47DE	This prefix shows up a lot in UNIX C-based programming languages
#	#FF7734	Color references in HTML and image editting programs
%	%20	Often used in URLs to express characters like "Space" (%20)
\x	\x0A	Often used to express character control codes like "Backspace" (\x08), "Escape" (\x1B), and "Line Feed" (\x0A)
&#x	&#x3A9	Used in HTML, XML, and XHTML to express unicode characters (e.g. Ω prints an Ω).
0h	0h5E	A prefix used by many programmable graphic calculators (e.g. TI-89).
Numeral / Text Subscript	BE3716, 13Fhex	This is more of a mathematical representation of base 16 numbers. Decimal numbers can be represented with a subscript 10 (base 10). Binary is base 2.

Table B.4: Hex Identifiers

B.9 Converting Hex to Decimal

There's an ugly equation that rules over hex-to-decimal conversion:

$h_n 16^n + h_{n-1} 16^{n-1} + \cdots + h_1 16^1 + h_0 16^0$

There are a few important elements to this equation. Each of the h factors (hn, hn-1) is a single digit of the hex value. If our hex value is 0xF00D, for example, h0 is D, h1 and h2 are 0, and h3 is F.

Powers of 16 are a critical part of hexadecimal. More-signficant digits (those towards the left side of the number) are multiplied by larger powers of 16. The least-significant digit, h0, is multiplied by 160 (1). If a hex value is four digits long, the most-significant digit is multiplied by 163, or 4096.

To convert a hexadecimal number to decimal, you need to plug in values for each of the h factors in the equation above. Then multiply each digit by its respective power of 16, and add each product up. Our step-by-step approach is:

1. Start with the right-most digit of your hex value. Multiply it by 160, that is: multiply by 1. In other words, leave it be, but keep that value off to the side. [1]
2. Move one digit to the left. Multiply that digit by 161 (i.e. multipy by 16). Remember that product, and keep it to the side.
3. Move another digit left. Multiply that digit by 162 (256) and store that product.
4. Continue multiplying each incremental digit of the hex value by increasing powers of 16 (4096, 65536, 1048576, ...), and remember each product.
5. Once you've multiplied each digit of the hex value by the proper power of 16, add them all up. That sum is the decimal equivalent of your hex value.

[1] Remember to convert alphabetic hex values (A, B, C, D, E, and F) to their decimal equivalent (10, 11, 12, 13, 14, and 15)

Appendix C

GITHUB

C.1 Cloning an existing repository

Use this to get a repository that already exists and pull it to your local machine. Go the directory where you want to place the repository and:

```
1  git clone <URL of repository>
2
```

C.2 Getting the latest updates

To get the latest updates from the repository:

```
1  git pull
2
```

C.3 Placing your edits into the repository

To get the latest updates from the respository:

```
git add .
git commit -m "<comment about what you are adding>"
git push

```

Appendix D

Lineage of Programming Languages

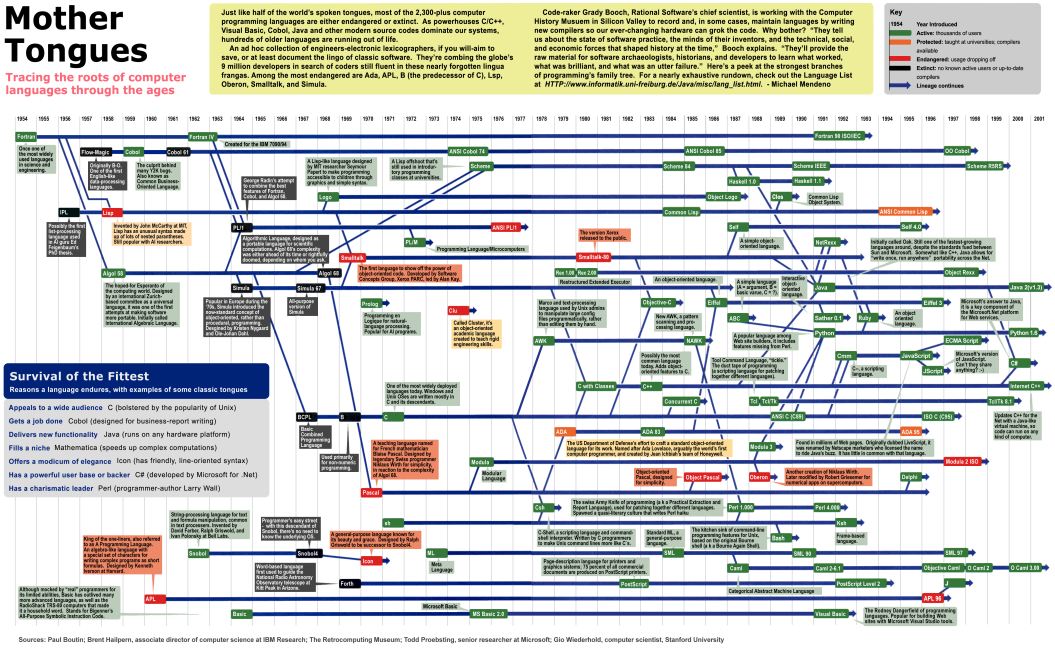
D.1 FORTRAN

Fortran; formerly FORTRAN, derived from Formula Translation[2]) is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.

Originally developed by IBM[3] in the 1950s for scientific and engineering applications, FORTRAN came to dominate this area of programming early on and has been in continuous use for over six decades in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing[4] and is used for programs that benchmark and rank the world's fastest supercomputers.[5][6]

Fortran encompasses a lineage of versions, each of which evolved to add extensions to the language while usually retaining compatibility with prior versions. Successive versions have added support for structured programming and processing of character-based data (FORTRAN 77), array programming, modular programming and generic programming (Fortran 90), high perfor-

Figure D.1: Program Language Lineage



mance Fortran (Fortran 95), object-oriented programming (Fortran 2003), concurrent programming (Fortran 2008), and native parallel computing capabilities (Coarray Fortran 2008/2018).

Fortran's design was the basis for many other programming languages. Among the better known is BASIC, which is based on FORTRAN II with a number of syntax cleanups, notably better logical structures,[7] and other changes to work more easily in an interactive environment.[8]

D.2 ALGOL

ALGOL 58, originally named IAL, is one of the family of ALGOL computer programming languages. It was an early compromise design soon superseded by ALGOL 60. According to John Backus[2]

"The Zurich ACM-GAMM Conference had two principal motives in proposing the IAL: (a) To provide a means of communicating numerical methods and other procedures between people, and (b) To provide a means of realizing a stated process on a variety of machines..."

ALGOL 58 introduced the fundamental notion of the compound statement, but it was restricted to control flow only, and it was not tied to identifier scope in the way that Algol 60's blocks were.

ALGOL 60 (short for Algorithmic Language 1960) is a member of the ALGOL family of computer programming languages. It followed on from ALGOL 58 which had introduced code blocks and the begin and end pairs for delimiting them. ALGOL 60 was the first language implementing nested function definitions with lexical scope. It gave rise to many other programming languages, including CPL, Simula, BCPL, B, Pascal, and C.

Niklaus Wirth based his own ALGOL W on ALGOL 60 before moving to develop Pascal. Algol-W was intended to be the next generation ALGOL but the ALGOL 68 committee decided on a design that was more complex and advanced rather than a cleaned simplified ALGOL 60. The official ALGOL versions are named after the year they were first published. Algol 68 is substantially different from Algol 60 and was criticised partially for being so, so that in general "Algol" refers to dialects of Algol 60.

D.3 CPL

CPL (Combined Programming Language) is a multi-paradigm programming language, that was developed in the early 1960s. It is an early ancestor of the C language via the BCPL and B languages.

D.4 BCPL

BCPL ("Basic Combined Programming Language") is a procedural, imperative, and structured computer programming language. Originally intended

for writing compilers for other languages, BCPL is no longer in common use. However, its influence is still felt because a stripped down and syntactically changed version of BCPL, called B, was the language on which the C programming language was based. BCPL introduced several features of many modern programming languages, including using curly braces to delimit code blocks.[3] BCPL was first implemented by Martin Richards of the University of Cambridge in 1967.

D.5 B

B is a programming language developed at Bell Labs circa 1969. It is the work of Ken Thompson with Dennis Ritchie.

B was derived from BCPL, and its name may be a contraction of BCPL. Thompson's coworker Dennis Ritchie speculated that the name might be based on Bon, an earlier, but unrelated, programming language that Thompson designed for use on Multics.

B was designed for recursive, non-numeric, machine-independent applications, such as system and language software.[3] It was a typeless language, with the only data type being the underlying machine's natural memory word format, whatever that might be. Depending on the context, the word was treated either as an integer or a memory address.

As machines with ASCII processing became common, notably the DEC PDP-11 that arrived at Bell, support for character data stuffed in memory words became important. The typeless nature of the language was seen as a disadvantage, which led Thompson and Ritchie to develop an expanded version of the language supporting new internal and user-defined types, which became the C programming language.

D.6 C

C, as in the letter c) is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, while a static type system prevents unintended operations. By design, C provides constructs that map efficiently to typical machine instructions and has found lasting use in applications previously coded in assembly language. Such applications include operating systems and various application software for computers, from supercomputers to embedded systems.

C was originally developed at Bell Labs by Dennis Ritchie between 1972 and 1973 to make utilities running on Unix. Later, it was applied to reimplementing the kernel of the Unix operating system.[6] During the 1980s, C gradually gained popularity. It has become one of the most widely used programming languages,[7][8] with C compilers from various vendors available for the majority of existing computer architectures and operating systems. C has been standardized by the ANSI since 1989 (see ANSI C) and by the International Organization for Standardization.

C is an imperative procedural language. It was designed to be compiled using a relatively straightforward compiler to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code. The language is available on various platforms, from embedded microcontrollers to supercomputers.

D.7 C++

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ has

object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms.[6]

C++ was designed with a bias toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency¸ and flexibility of use as its design highlights.[7] C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications,[7] including desktop applications, servers (e.g. e-commerce, Web search, or SQL servers), and performance-critical applications (e.g. telephone switches or space probes).[8]

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2017 as ISO/IEC 14882:2017 (informally known as C++17).[9] The C++ programming language was initially standardized in 1998 as ISO/IEC 14882:1998, which was then amended by the C++03, C++11 and C++14 standards. The current C++17 standard supersedes these with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs since 1979 as an extension of the C language; he wanted an efficient and flexible language similar to C that also provided high-level features for program organization.[10] C++20 is the next planned standard, keeping with the current trend of a new version every three years.[11]

D.8 Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.[27]

Python is dynamically typed and garbage-collected. It supports multiple pro-

gramming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.[28]

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, is "sunsetting" on January 1, 2020 (after extension; first planned for 2015), and the Python team of volunteers will not fix security issues, or improve it in other ways after that date.[29][30] With the end-of-life, only Python 3.5.x and later will be supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[31] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

D.9 PHP

PHP: Hypertext Preprocessor (or simply PHP) is a general-purpose programming language originally designed for web development. It was originally created by Rasmus Lerdorf in 1994;[6] the PHP reference implementation is now produced by The PHP Group.[7] PHP originally stood for Personal Home Page,[6] but it now stands for the recursive initialism PHP: Hypertext Preprocessor.[8]

PHP code may be executed with a command line interface (CLI), embedded into HTML code, or used in combination with various web template systems, web content management systems, and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in a web server or as a Common Gateway Interface (CGI) executable. The web server outputs

the results of the interpreted and executed PHP code, which may be any type of data, such as generated HTML code or binary image data. PHP can be used for many programming tasks outside of the web context, such as standalone graphical applications[9] and robotic drone control.[10]

The standard PHP interpreter, powered by the Zend Engine, is free software released under the PHP License. PHP has been widely ported and can be deployed on most web servers on almost every operating system and platform, free of charge.[11]

The PHP language evolved without a written formal specification or standard until 2014, with the original implementation acting as the de facto standard which other implementations aimed to follow. Since 2014, work has gone on to create a formal PHP specification.[12]

As of September 2019, over 60% of sites on the web using PHP are still on discontinued/"EOLed"[13] version 5.6 or older;[14] versions prior to 7.1 are no longer officially supported by The PHP Development Team,[15] but security support is provided by third parties, such as Debian.[16]