IoT Bootcamp - Virtual Lessons

Brian Rashap

April 22, 2020

Contents

List of Figures

List of Tables

List of Code Listings

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 E 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.

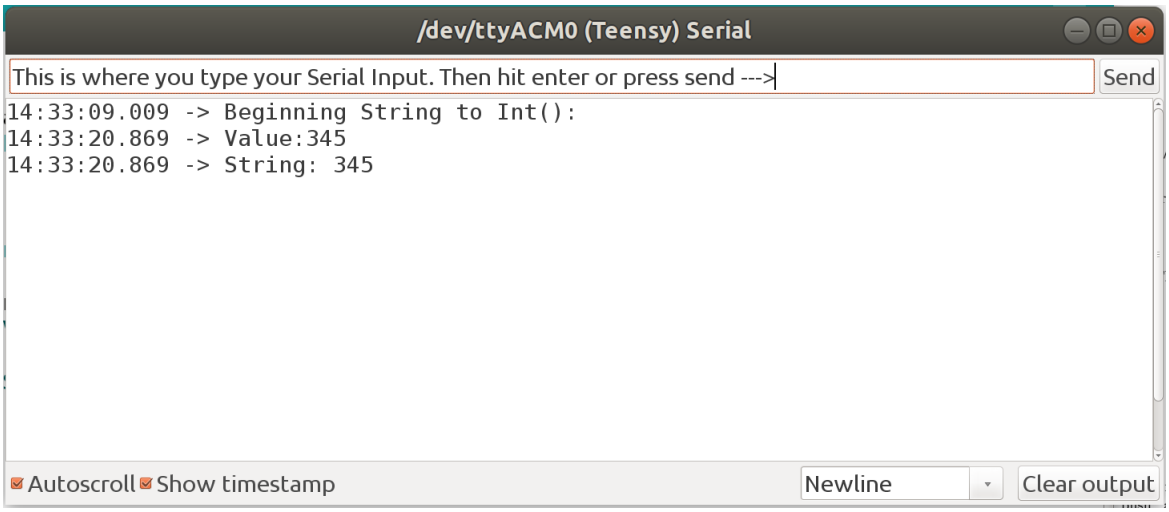


Figure 2.1: Serial Input

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 E.

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.

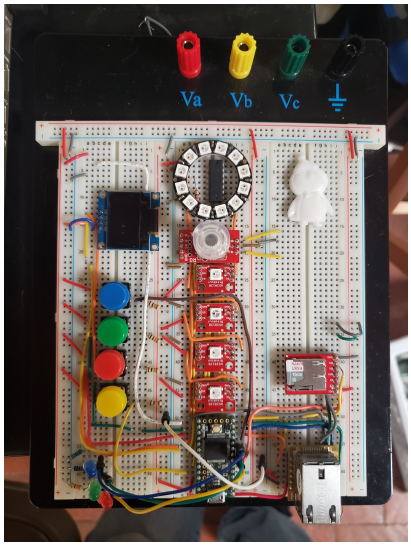


Figure 3.1: Example Board

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.

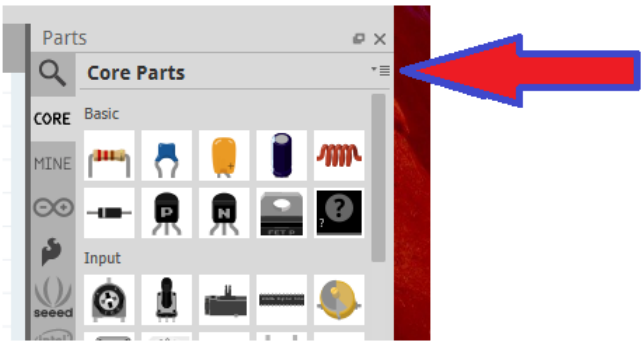


Figure 3.2: Importing Part into Fritzing

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`

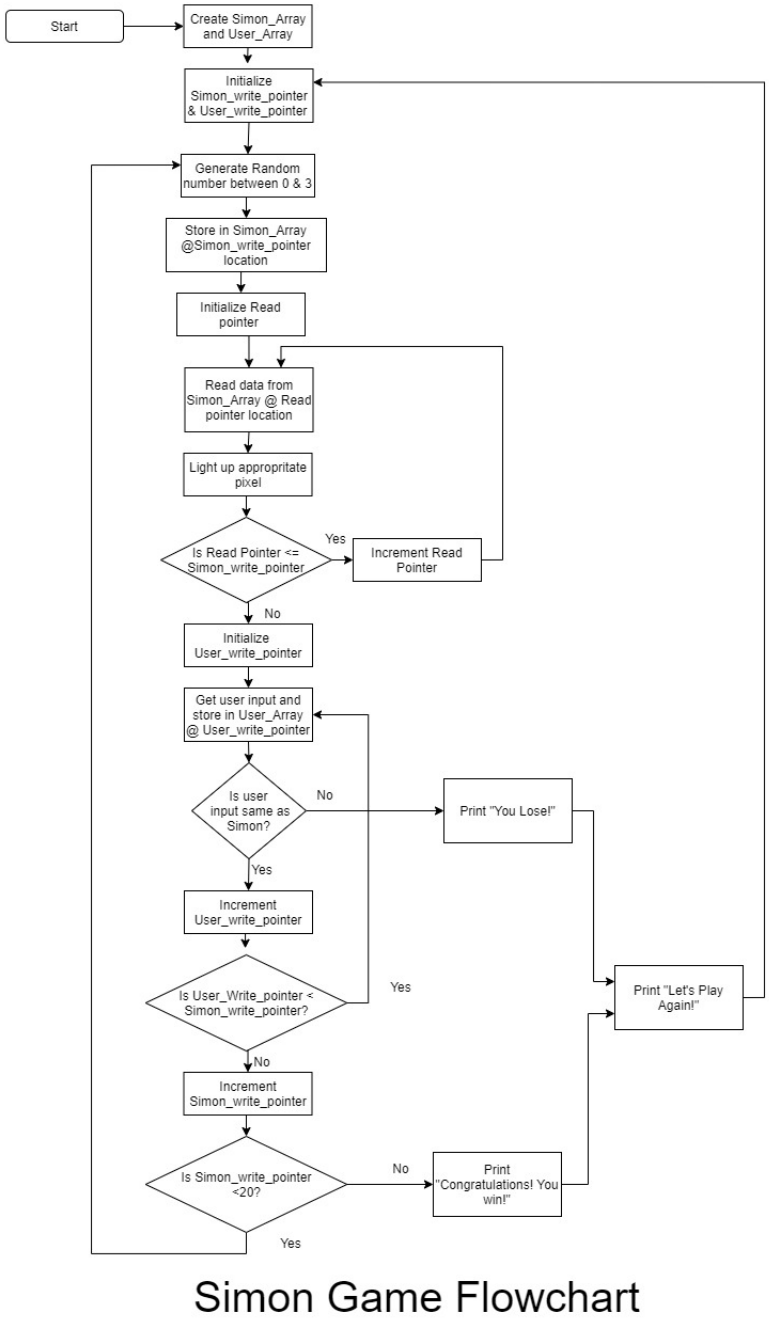


Figure 4.1: Simon ® Flow Chart

Chapter 5

Virtual Lesson 5

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. I will be updating the instructor_master repository with the Simon code at the end of today.

2. Start Coding your Smart Room Controller, see the following section.

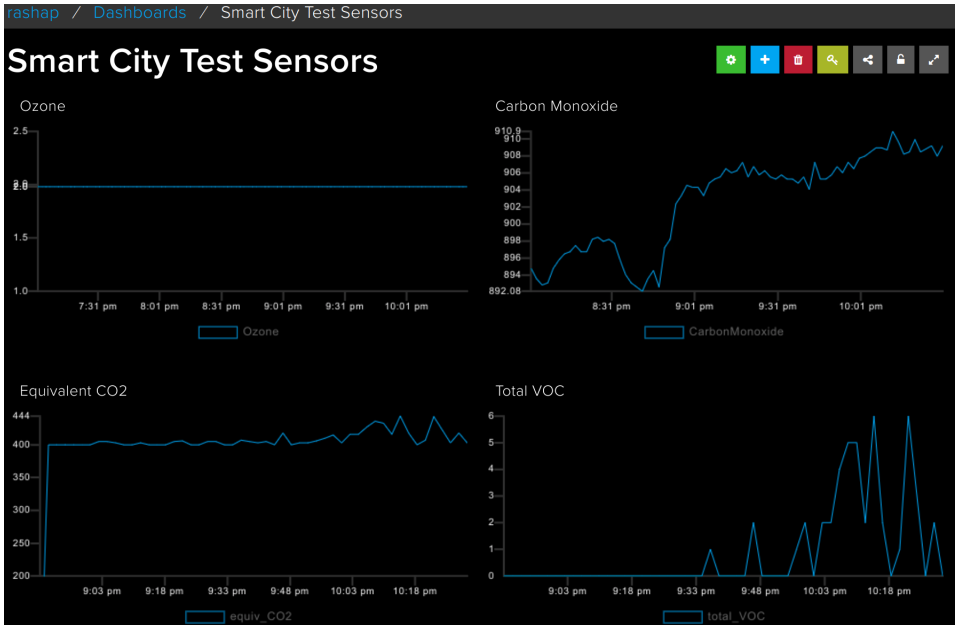


Figure 5.1: Adafruit Dashboard

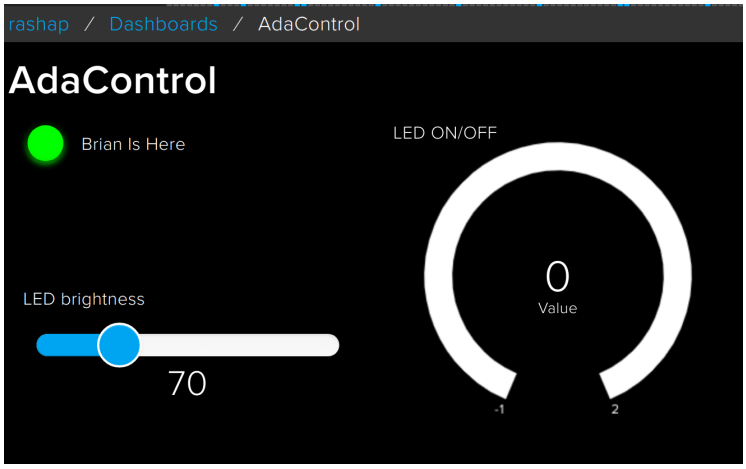


Figure 5.2: Controlling for Adafruit

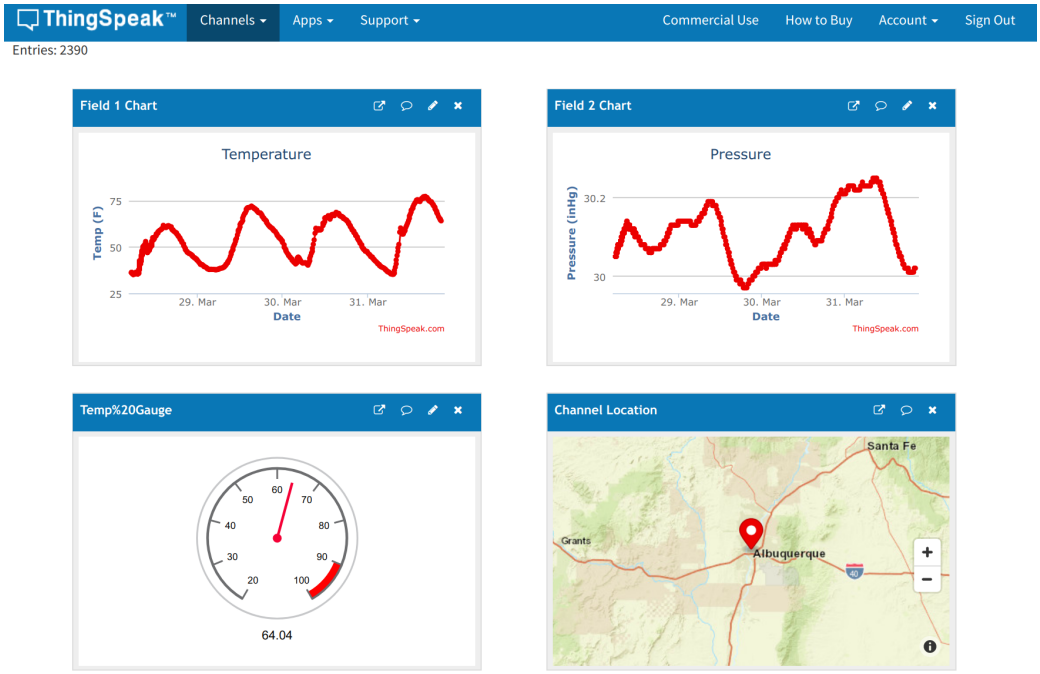


Figure 5.3: ThingSpeak Dashboard

Figure 5.4: Smart City Dashboard

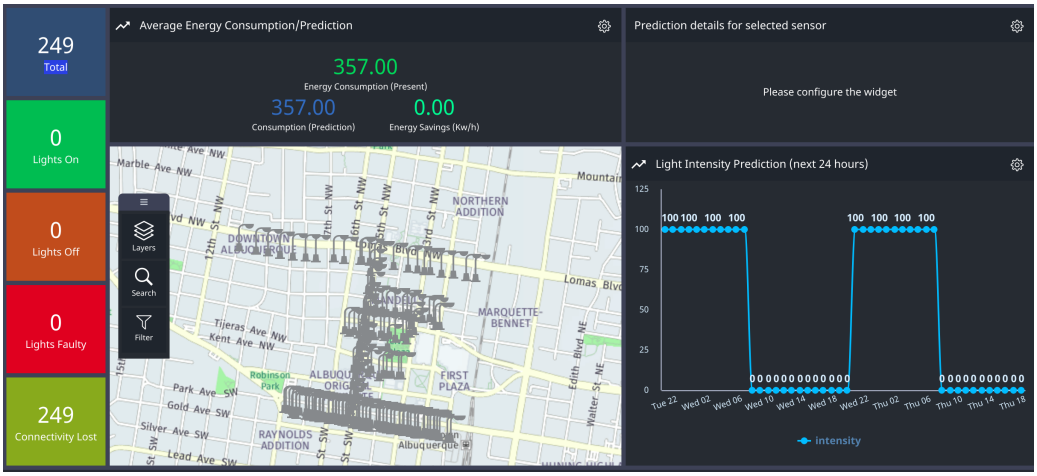


Figure 5.5: Smart City Lighting

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 WemoOn(outlet) - turn the Wemo "NeoPixel" to blue for the first outlet, yellow for the second outlet, and green for both outlets.
- void WemoOff(outlet) - turn off the appropriate "outlet."

5.2.2 Smart Room Controller Specifications

Your Smart Room Controller should have, at least, the following functionality. This will be easier to code if you write flow charts for each function and then combine them.

- 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." A NeoPixel near the button should indicate which outlet is selected (Blue or Yellow).
- OPTIONAL: Use the OLED Display to assist the user by showing the controller settings.

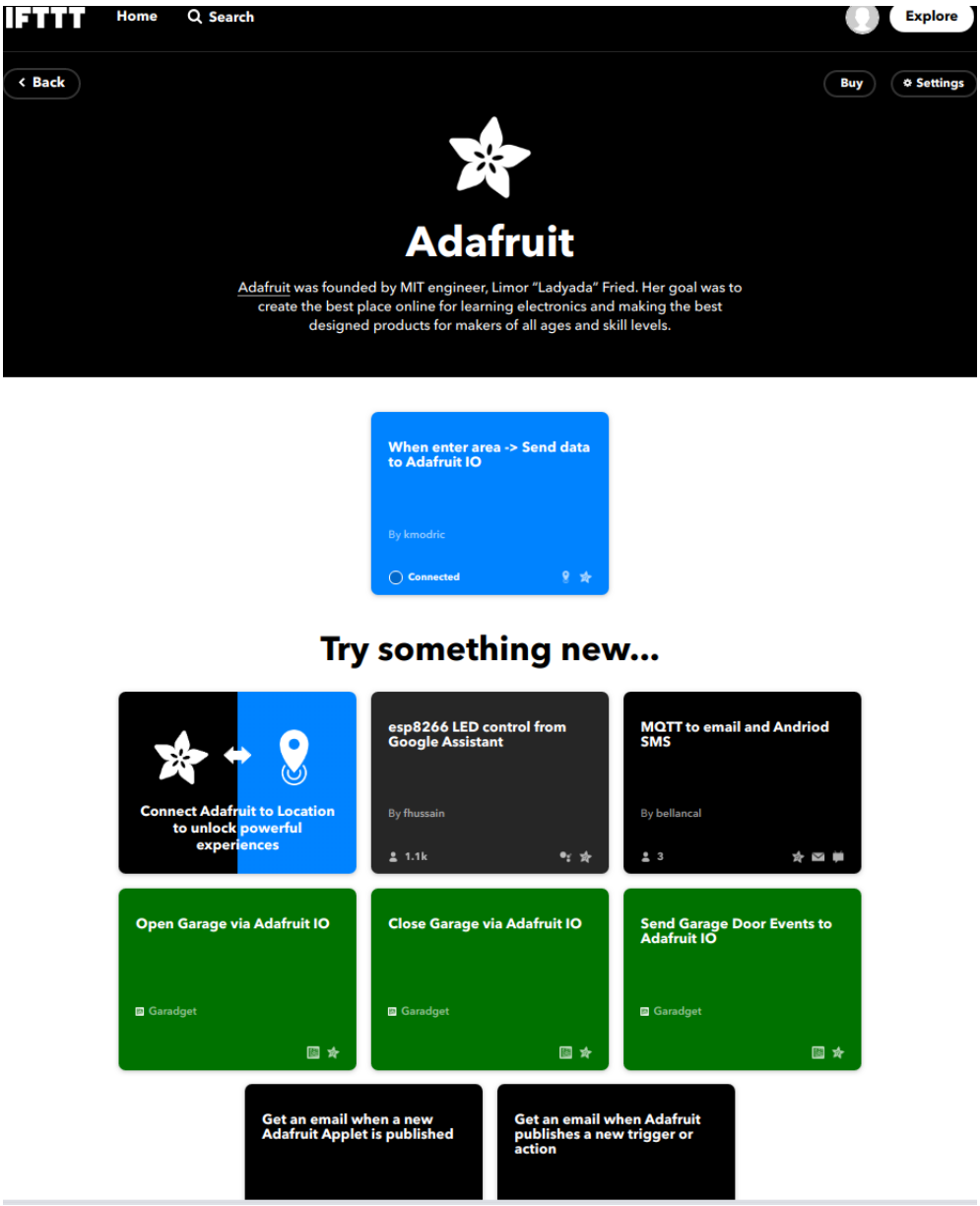


Figure 5.6: IF Then Then That

Chapter 6

Virtual Lesson 6

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

6.1 Argon Overview

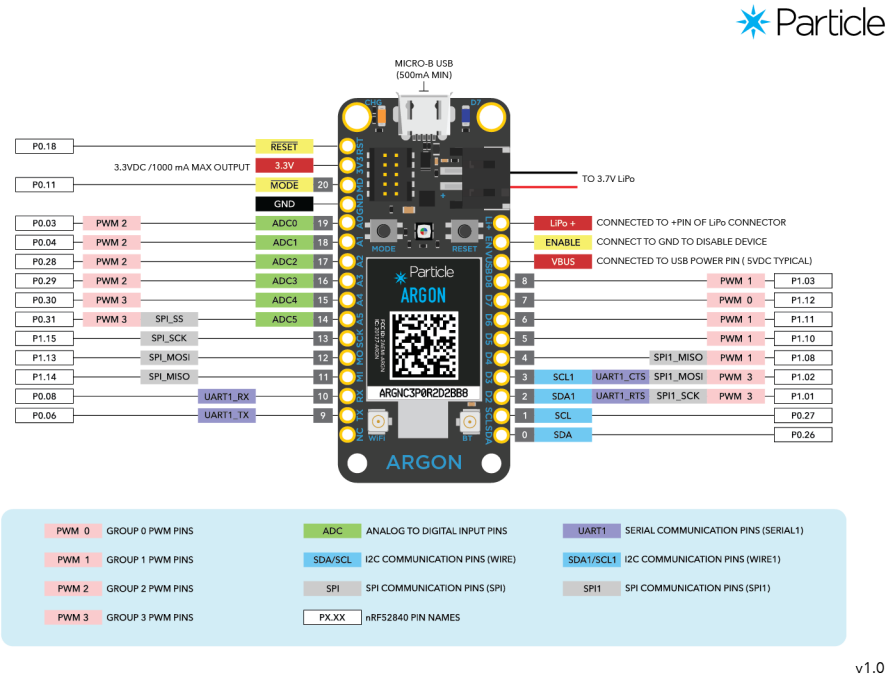
The Argon is a powerful Wi-Fi enabled development board that can act as either a standalone Wi-Fi endpoint or Wi-Fi enabled gateway for Particle Mesh networks. It is based on the Nordic nRF52840 and has built-in battery charging circuitry so it's easy to connect a Li-Po and deploy your local network in minutes.

The Argon is great for connecting existing projects to the Particle Device Cloud or as a gateway to connect an entire group of local endpoints.

6.1.1 Features

- Espressif ESP32-D0WD 2.4 GHz Wi-Fi coprocessor
  - On-board 4MB flash for ESP32

Figure 6.1: Particle Argon Pin Layout



  - 802.11 b/g/n (2.4 GHz), up to 150 Mbps

- Nordic Semiconductor nRF52840 SoC
  - ARM Cortex-M4F 32-bit processor @ 64MHz
  - 1MB flash, 256KB RAM
  - IEEE 802.15.4-2006 (e.g. Zigbee): 250 Kbps
  - Bluetooth 5: 2 Mbps, 1 Mbps, 500 Kbps, 125 Kbps
  - NFC-A tag
- On-board additional 4MB SPI flash
- 20 mixed signal GPIO (6 x Analog, 8 x PWM), UART, I2C, SPI
- Micro USB 2.0 full speed (12 Mbps)
- Integrated Li-Po charging and battery connector

- JTAG (SWD) Connector

6.2 Why Particle

The Particle Argon was selected as our next microcontroller because:

- Global reach with over 200,000 IoT professionals
- Edge-to-Cloud infrastructure
- Prototyping to Production with same code
- WI-FI, Bluetooth, and Cellular connectivity
- Secure Device OS
- Built in cloud communication
- Real-Time OS that works across all products

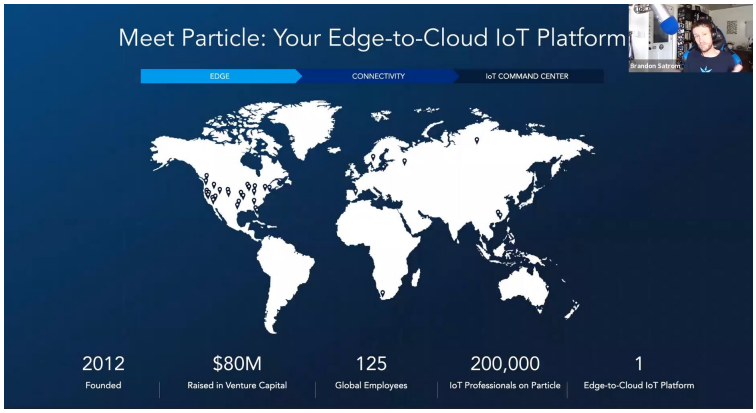


Figure 6.2: Particle's Global Reach

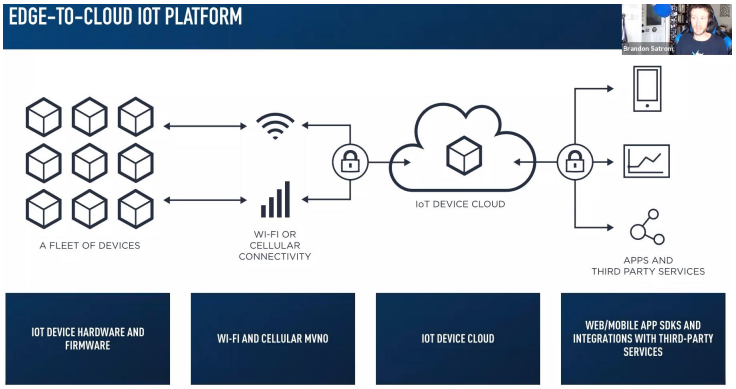


Figure 6.3: Edge to Cloud

Figure 6.4: Prototyping and Production

6.3 Virtual Lesson 6 Assignment

6.3.1 Coding

- Continue working on your Smart Room Controller

6.3.2 Fusion 360

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

How to 3D Model an iPhone X Phone Case - Learn Autodesk Fusion 360 in 30 Days: Day #10 at: `https://www.youtube.com/watch?v=eW9IrA3i4w4&list=PLrZ2zKOtC_-Bl4uHmsx5pZVUpJKFif3lR&index=2&t=1s`

6.3.3 Software for Particle Argon

Please do the following in the order below:

1. Create Particle login: `https://login.particle.io/signup`
2. Install Particle Command Line Interface. Download from `https://docs.particle.io/tutorials/developer-tools/cli/`. NOTE: you need to run this by right-clicking on it and selecting "Run As Administrator." Test that the Particle CLI installed correctly by going to PowerShell or Terminal and type particle. See Figure 6.5.
3. Download Particle Workbench / Visual Studio Code `https://docs.particle.io/quickstart/workbench/`. Select all default values during install. After it is installed, when you lauch it, it may ask you to Install Dependencies, if so, select yes. See Figure 6.6.

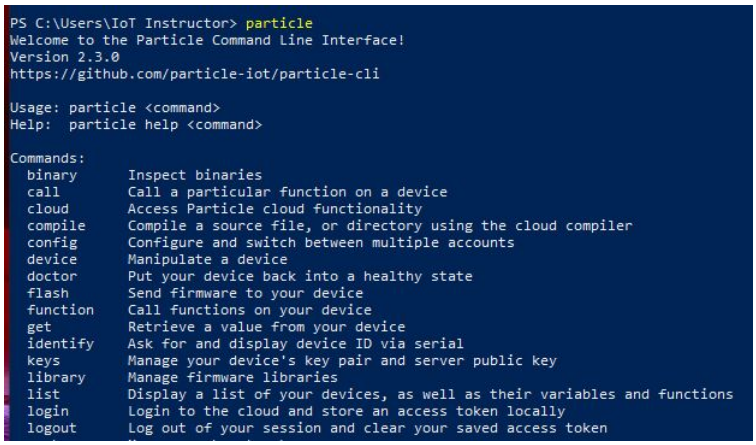


Figure 6.5: Visual Studio Code

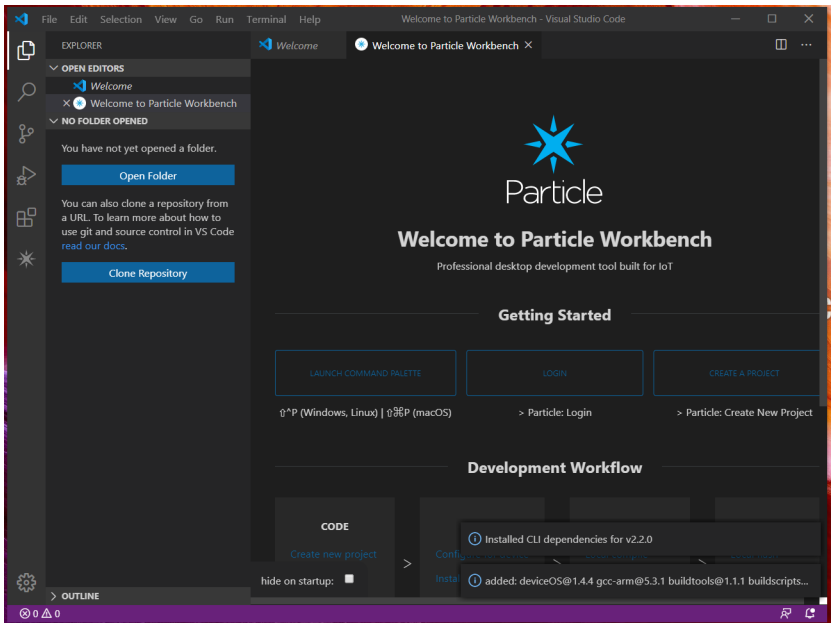


Figure 6.6: Visual Studio Code

Chapter 7

Virtual Lesson 7

7.1 Serial.Read() and Assignment

Today we are going to learn how to utilize the various Serial.Read() functions. We are going code an assignment where we get an input from the Serial Monitor three different ways. For the first one, you'll want to refer to V02_01_serialinput but remove the IF statements that differentiate integers from characters.

```
/*
 * Project: SerialRead
 * Description: More Practice Using Serial Inputs
 * Author: <your name>
 * Date: <today's date>
 */

 /*
  *  ASSIGNMENT
  *  Go to https://www.arduino.cc/reference/tr/language/
    functions/communication/serial/
  *  Read the descriptions of:
  *     read()
  *     readString()
  *     readStringUntil()
  *     setTimeout()
  *
```

```
17   *  Using each of the three read functions, have the user
18   *  enter their Name, Age, and Favorite Color
19   *
20   *  To the Serial Monitor write out:
21   *     It is nice to meet you, <name>
22   *     Did you know you are <xxx> months old
23   *         (note:you need to convert years to months)
24   *     Wow, my favoite color is also <color>
25   *
26   *  Remember: the enter key or send button returns a '\n'.
27   *  When running this program, experiment with being slow
28   *  to enter your string, as this will give you a feel
29   *  for how setTimeout() works.
30   */
31
32
33
34 void setup() {
35   // put your setup code here, to run once:
36
37 }
38
39 void loop() {
40   // put your main code here, to run repeatedly:
41
42 }
```

7.2 Argon Set-Up

7.2.1 Particle IoT App

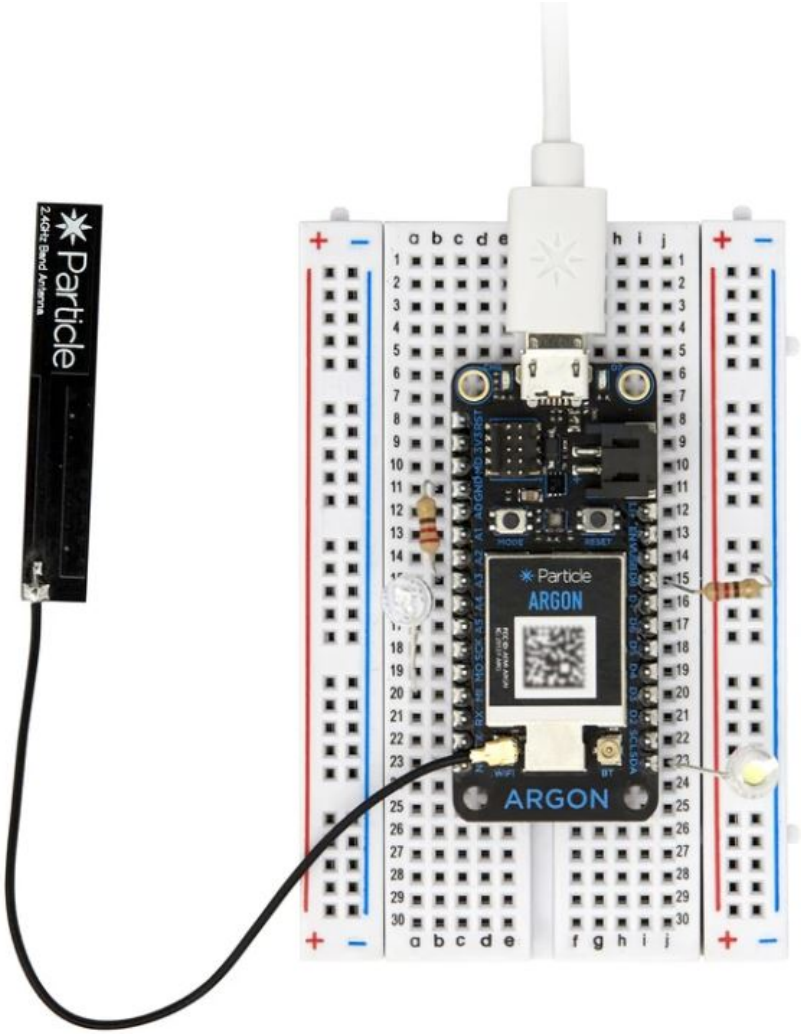


Figure 7.1: Particle Argon Setup

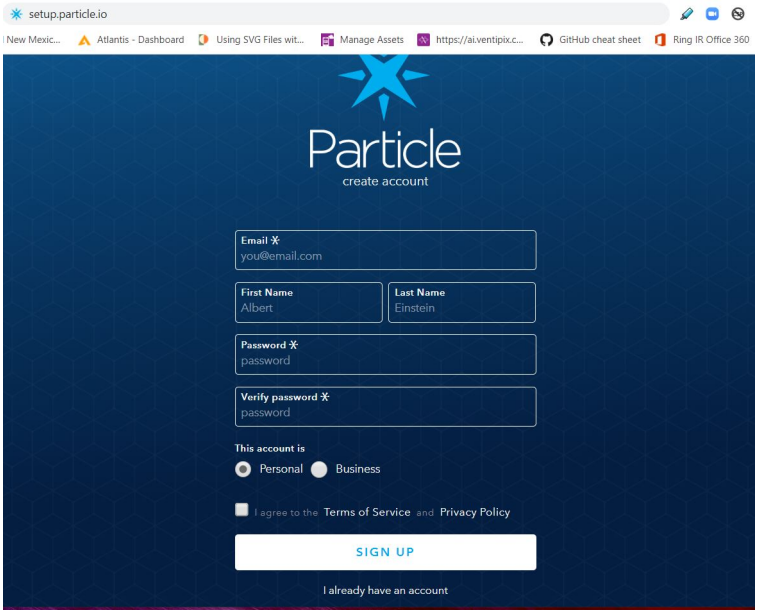


Figure 7.2: Particle Argon Setup

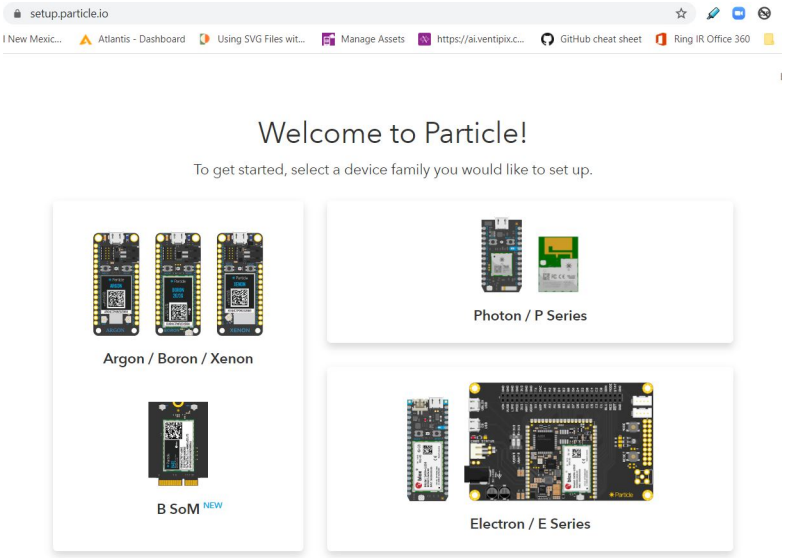


Figure 7.3: Particle Argon Setup

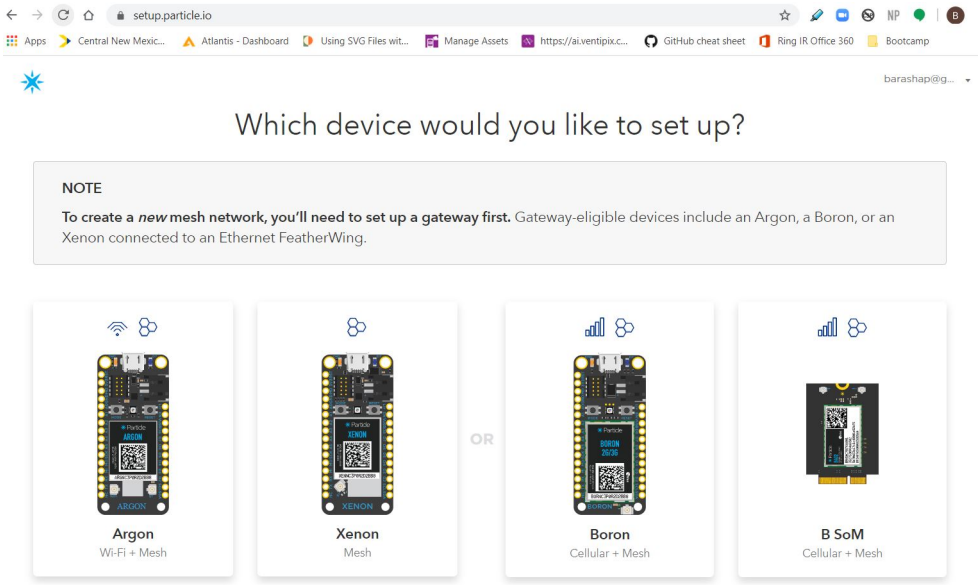


Figure 7.4: Particle Argon Setup

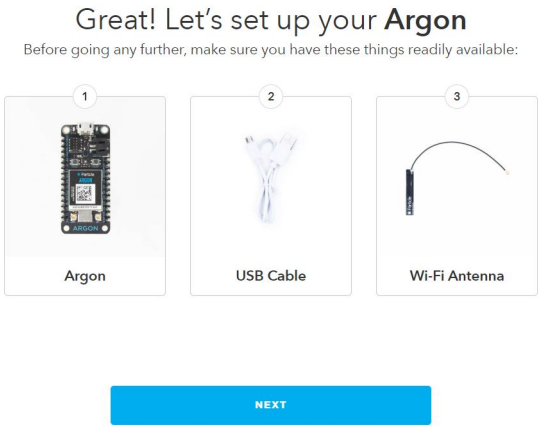


Figure 7.5: Particle Argon Setup

Figure 7.6: Particle Argon Setup

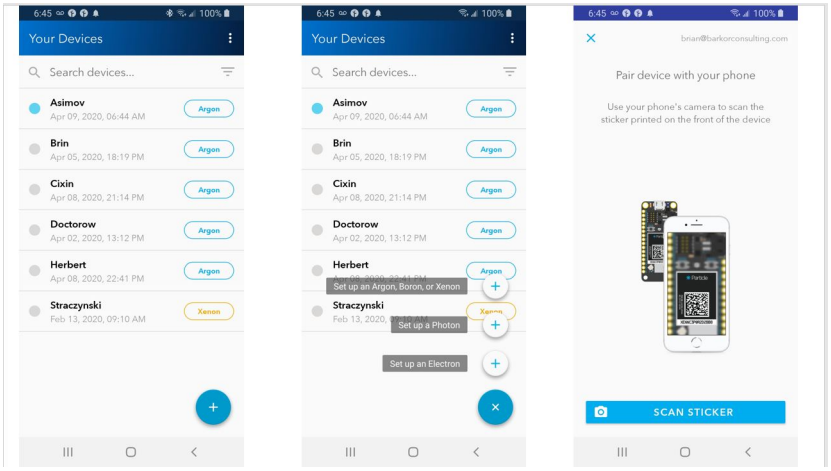


Figure 7.7: Particle Argon Setup

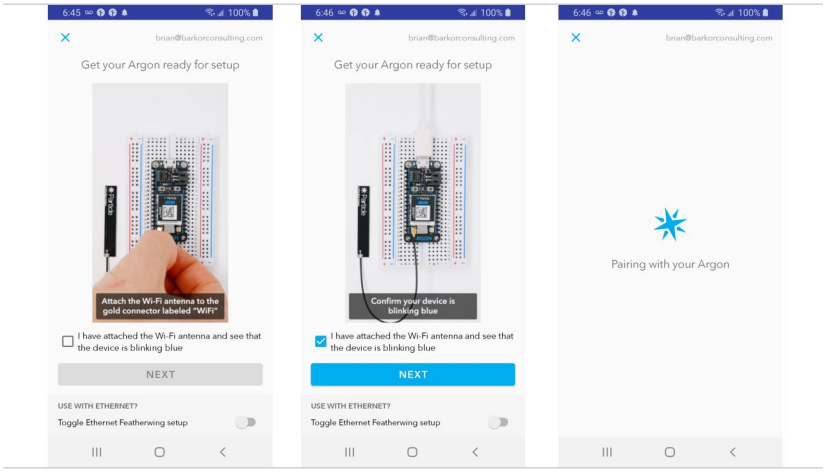


Figure 7.8: Particle Argon Setup

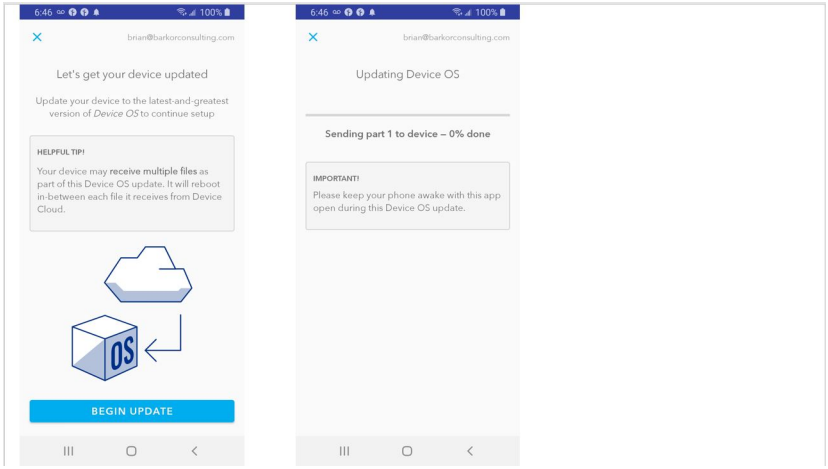


Figure 7.9: Particle Argon Setup

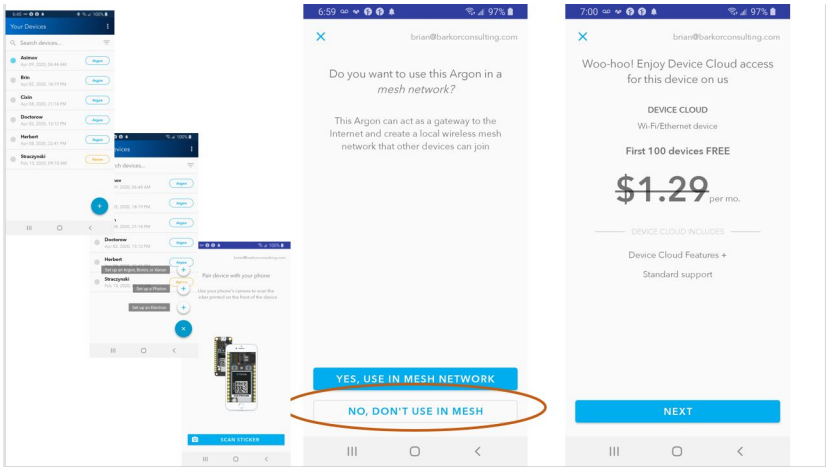


Figure 7.10: Particle Argon Setup

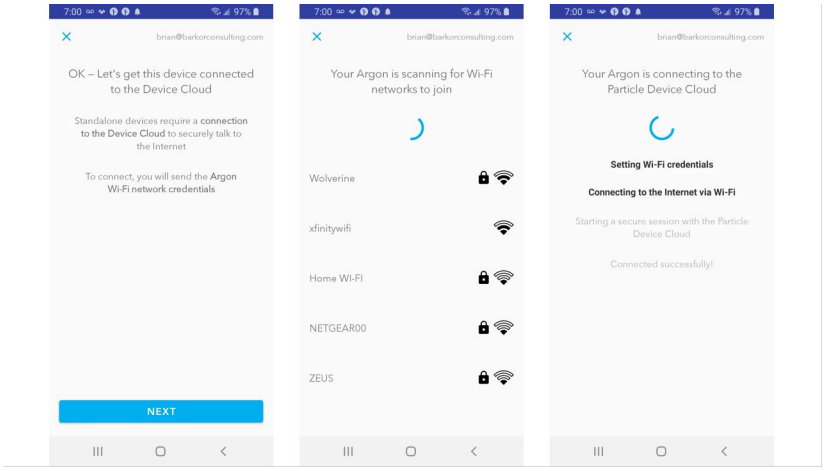


Figure 7.11: Particle Argon Setup

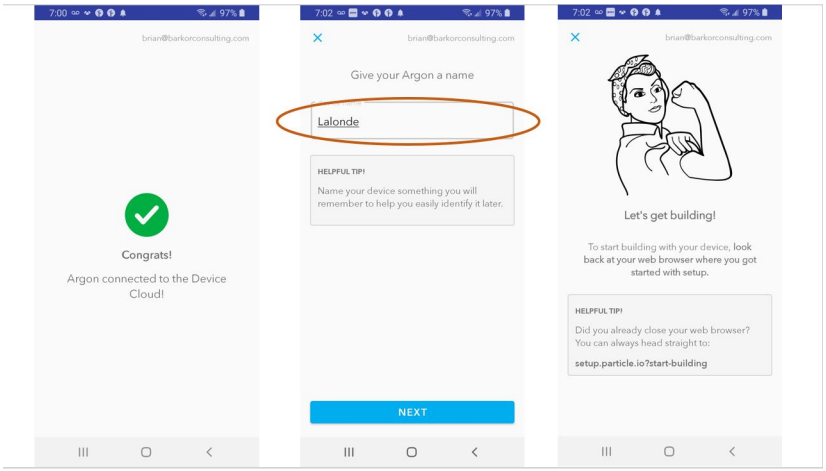


Figure 7.12: Particle Argon Setup

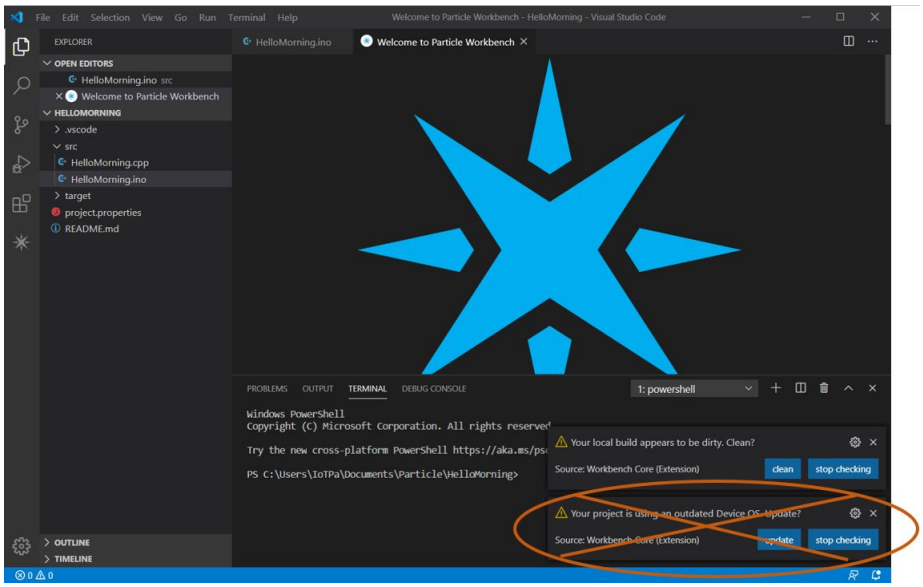


Figure 7.13: Particle Argon Setup

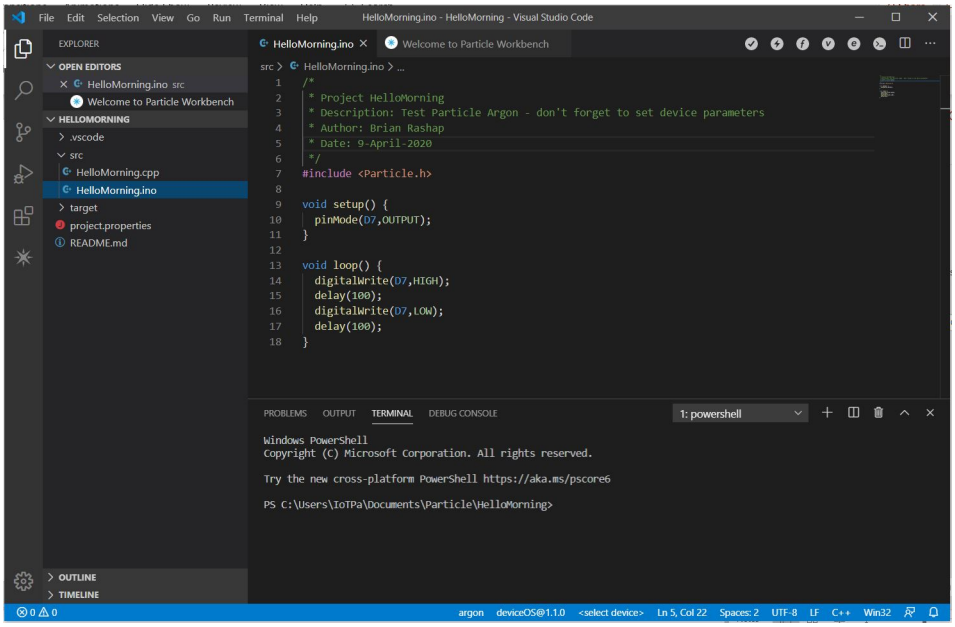


Figure 7.14: Particle Argon Setup

7.2.2 Particle CLI Setup

First, get your wifi credentials.

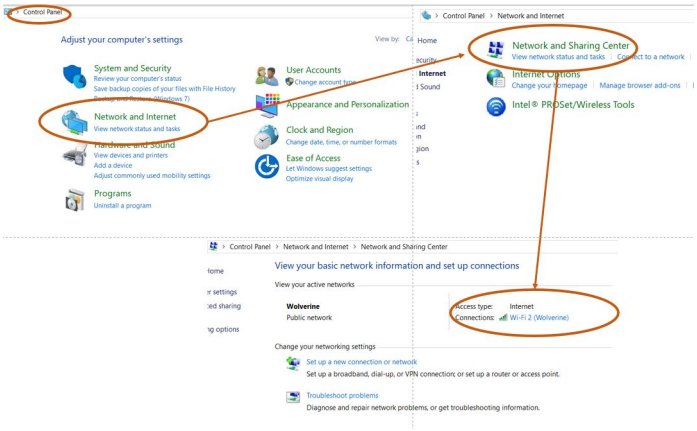


Figure 7.15: Particle Argon Setup

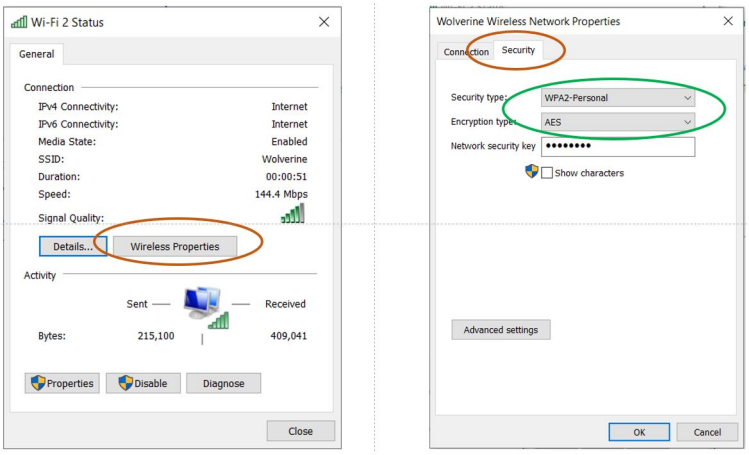


Figure 7.16: Particle Argon Setup

We will start by placing our Particle Argon onto our breadboard. Next, we will attach the WiFi antenna. And, then connect to your computer via the USB port. Check Device Manager $->$ Ports to see if your Argon is recognized. If not, then we will run Particle Setup through the Smartphone App. Start PowerShell or Terminal and execute the following instructions as shown in Figures 7.1 through 7.6.

1. particle login (Figure 7.1)

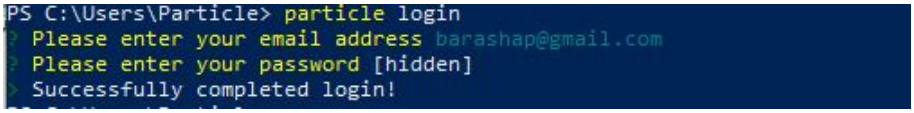


Figure 7.17: Particle Argon Setup: particle serial identify

2. particle serial identify (Figure 7.2)

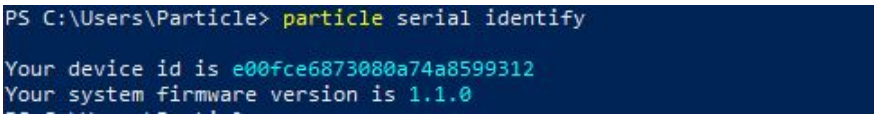


Figure 7.18: Particle Argon Setup: particle serial identify

3. particle serial wifi (Figure 7.3)

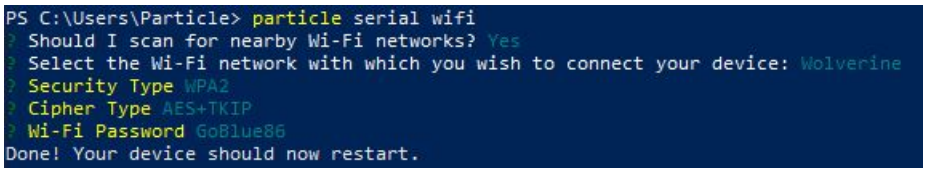


Figure 7.19: Particle Argon Setup: particle serial wifi

4. particle device add. You will use the device ID found with "particle serial identify" (Figure 7.4)

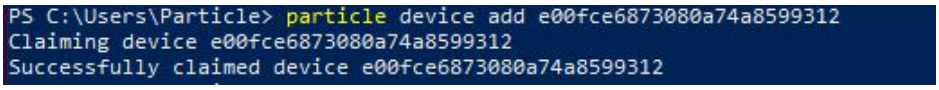


Figure 7.20: Particle Argon Setup: particle device add

5. particle device rename. Again, you will use the device ID found earlier, and then provide your device with a unique name (Figure 7.5)

```
PS C:\Users\Particle> particle device rename e00fce6873080a74a8599312 Herbert
Renaming device e00fce6873080a74a8599312
Successfully renamed device e00fce6873080a74a8599312 to: Herbert
```

Figure 7.21: Particle Argon Setup: particle device rename

6. particle list. We will validate that our device is setup by execution both particle serial list and particle list (Figure 7.6)

```
PS C:\Users\Particle> particle serial list
Found 1 device connected via serial:
COM9 - Argon - e00fce6873080a74a8599312
PS C:\Users\Particle> particle list
Cixin [e00fce68a3464adbfde1add3] (Argon) is online
Doctorow [e00fce687365c8dfdc103df8] (Argon) is offline
Asimov [e00fce68e5be047f5f12bcec] (Argon) is online
  variables:
    temp (double)
Brin [e00fce68be19b0df2ece5a7f] (Argon) is offline
Herbert [e00fce6873080a74a8599312] (Argon) is online
  Functions:
    int digitalread(String args)
    int digitalwrite(String args)
    int analogread(String args)
    int analogwrite(String args)
Straczynski [e00fce68cac1954a305932cd] (Xenon) is offline
```

Figure 7.22: Particle Argon Setup: particle serial list

7.3 Particle Not Connecting to Cloud

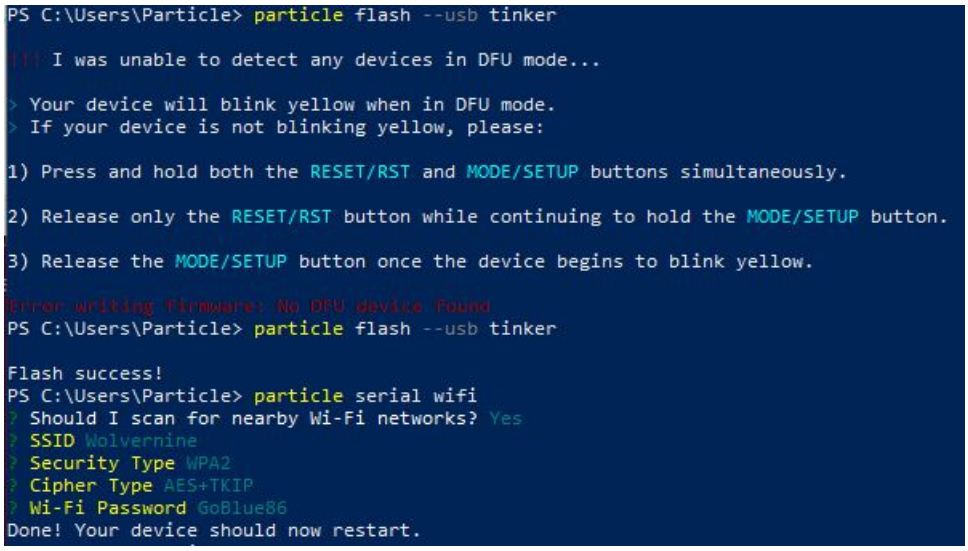


Figure 7.23: Particle Argon Setup: particle flash

Chapter 8

Virtual Lesson 8

8.1 Command Palette

The Command Palette will become a familiar part of the user interface that is used during the rest of this bootcamp. As the name implies, the Command Palette provides access to many commands such as open files, search for symbols, and see a quick outline of a file, all using the same interactive window. It can be invoked via cmd+shift+p on macOS or ctrl+shift+p on Linux and Windows.

Workbench adds custom Particle commands to the palette. Start typing Particle to see all the currently available commands. A sample list is seen in Figure 8.1.

8.2 Formatted Printing

C/C++ gives us the capability of embedding variables in print strings. Rather than having to write multiple Serial.print() and Serial.println() commands, we can combine text and variables by using Serial.printf() and Serial.printlnf(). The same methods can be used in our libraries, such as Adafruit_SSD1306.

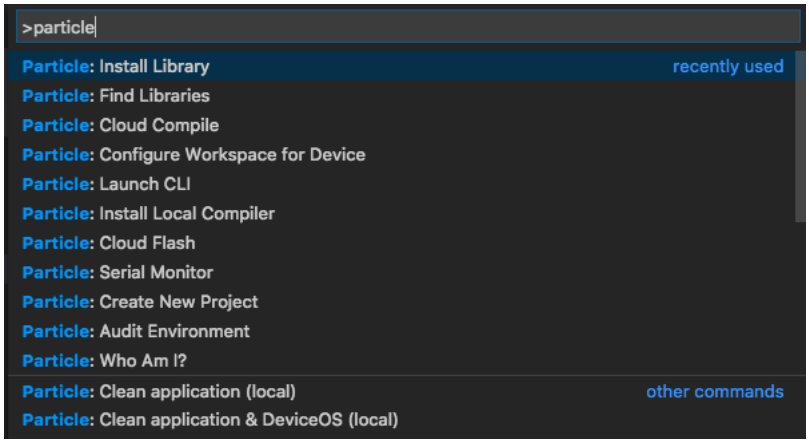


Figure 8.1: Visual Studio Code Command Palette

For example, *display.printf*(). The list of specifiers can be found in Figure 8.2.

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

Figure 8.2: Formatted Printing

We can revisit our HelloPaticle code with both the Serial Monitor and using formatted printing. In the below code, you'll notice that we start the Serial Monitor the same way as always. Line 12: *Serial.begin*(9600). When we Flash or code to the Argon, we'll then use invoke Particle:Serial Monitor by using cmd+shift+p on macOS or ctrl+shift+p on Linux and Windows. In our code, starting on Line 25, we can practice using printf() and printfln(). Notice, the \n can be used to include a carriage returning in the print string.

```
1  /*
2   * Project HelloParticle
3   * Description: Hello World with Particle Argon,
4   * with some Serial.printf learning
5   * Author: Brian Rashap
6   * Date: 09-APR-2020
7   */
8  int i;
9  int ledDelay = 750;
10 int ledPin = D7;
11 float pivalue = 3.141592653589;
12 void setup() {
13   Serial.begin(9600);
14   pinMode(D7,OUTPUT);
15   Serial.println("Hello World - Ready to Blink");
16 }
17 void loop() {
18   for(i=0;i<20;i++) {
19     Serial.println("Turning LED on");
20     digitalWrite(ledPin,HIGH);
21     delay(ledDelay);
22     Serial.println("Turning LED off");
23     digitalWrite(ledPin,LOW);
24     delay(ledDelay);
25   }
26 // Print Formatted Data - %i = int, %s = string, %f = float
27   Serial.printf("The delay is %i milliseconds \n",ledDelay);
28   Serial.printlnf("Values of %s = %1.2f or %1.4f or %1.6f","
       PI",pivalue, pivalue, pivalue);
29   Serial.printlnf("The Moon is %1.3E meters away",
       238855*1609.334);
30   delay(10*ledDelay);
31 }
```

8.3 Virtual Lesson 8 Assignment

8.3.1 Particle inputs and outputs

In Fritzing, create a breadboard layout to match this schematic. You will need to find the Particle Argon fritzing part online. When you search for it, you should find a github repository that you can clone. In PowerShell or Terminal, got to your Documents\Fritizing directory and clone the repository, and then import the part. Using the parts in your Particle Argon Development Kit, wire up this diagram on with your Argon.

- The anode of the photodiode is connected to Pin A0. Note, unlike an LED, the cathode (short pin) of the photodiode is connected to 3.3V.
- The anode (long pin) of the LED is connected to Pin D4 (which is a PWM pin on the Argon).

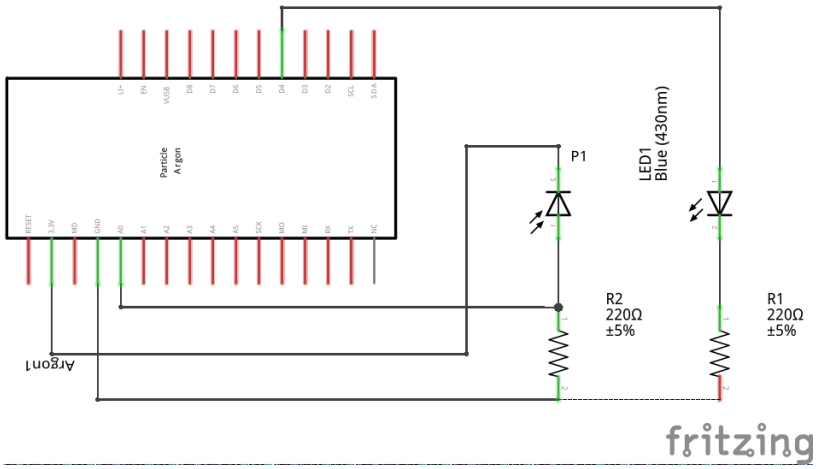
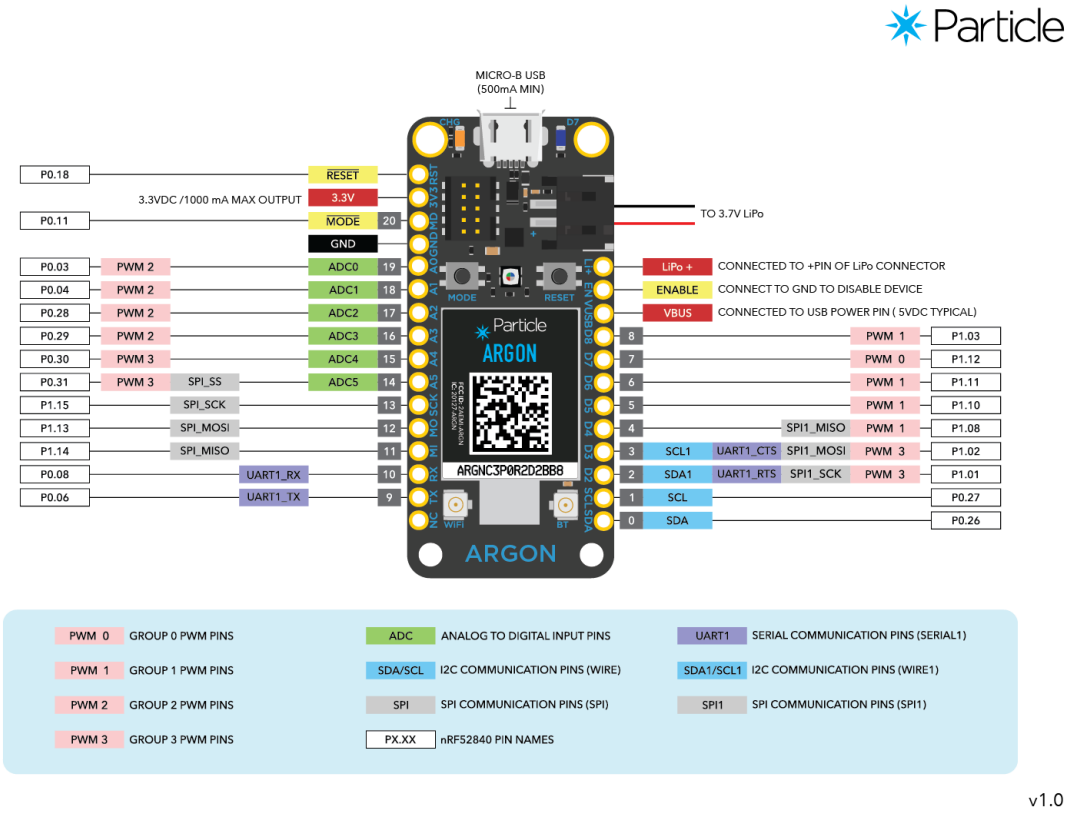


Figure 8.3: Hello Particle Schematic

Thought Pin D4 is labeled as a Digital Pin, you'll notice on the Particlue Pin Layout that it is also a PWM Pin, which allows you to use it for $analogWrite()$.

Figure 8.4: Particle Argon Pin Layout



For your first code with the Particle Argon. On Monday, I will be giving you a repository to place this code. For now, you can place it in Documents\<Your Name>\Particle.

1. Read the photodiode voltage as an analog input. Experiment with seeing what values you get with various light conditions - ambient light, covered with your hand, with your cellphone shining on it, etc.

2. Utilize formatted printing in this assignment to display the inputs from the photodiode and to tell the user the status of the LED.

3. From your measurements, try to determine how many bits of resolution

the Argon's analog input pins have. Then, connect an analog input directly to 3.3V and see if you are correct.

4. Use a digital output to turn on the LED if your hand is covering you photodiode and turn off if it is uncovered. You'll need to set a threshold based on your experiments in #1.

5. Now, have your LED brightness vary with the amount of light that your photodiode sees. Note, $analogWrite()$ on the Argon by default is 8-bits, so you'll need to convert based on the resolution of the $analogRead()$ found in #1.

6. Utilize formatted printing in this assignment to display the inputs from the photodiode and to tell the user the status of the LED.

8.3.2 Fusion 360

For our final Fusion 360 tutorial, we will be learning how to make a snap case for a Raspberry PI controller. Tanda is in the process of creating a Argon-based 3D model, and in two weeks you will be getting an assignment to make your own case using what you learn in this tutorial.

Fusion 360 Snap Fit Cases — 3D-Printable Raspberry Pi Case: `https://www.youtube.com/watch?v=E0NVC8xhf3I`

Additional information can be found at: `https://productdesignonline.com/fusion-360-tutorials/fusion-360-snap-fit-cases-3d-printable-raspberry-pi-case`

8.3.3 Smart Room Controller and Simon®

We will slowly introduce new topics for the Particle Argon controller next week. However, you need to continue to work on your last two Teensy projects, with priority for the second one.

- The Simon® Game

- The Smart Room Controller

The smart room controller is considered your midterm project, when we get back into the classroom, you will be showing off it's functionality for your classmates.

Chapter 9

Particle Lesson 1

9.1 Servo Motors

A servomechanism (servo) can refer to quite a few different machines that have been around longer than most may realize. Essentially, a servo is any motor-driven system with a feedback element built in. Servos are found everywhere from heavy machinery, to power steering in vehicles, to robotics and a wide variety of electronics.

A servo motor basically has three core components: a DC motor, a controller circuit, and a potentiometer or similar feedback mechanism. The DC motor is attached to a gearbox and output/drive shaft to increase the speed and torque of the motor. The DC motor drives the output shaft. The controller circuit interprets signals sent by the controller, and the potentiometer acts as the feedback for the controller circuit to monitor the position of the output shaft. Nearly all small servos have a standard three-pin. The color coding can vary between brands, but the pins are almost universally in the same order. When combined together, you can power and control the direction, speed and position of the output shaft with just three wires.

In order to move a servo to a position along its movement arc, or, in the case of continuous rotation servos the speed and direction of the motor, the controller needs to send a precisely timed signal for the servo to interpret.

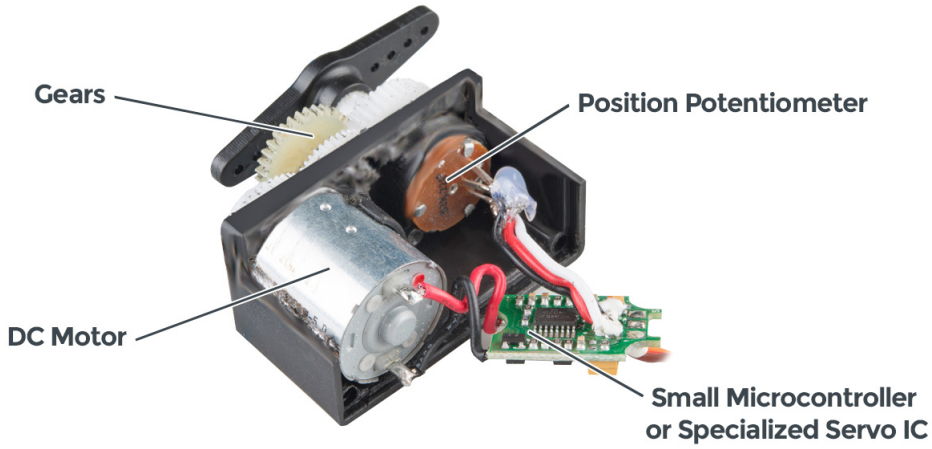


Figure 9.1: Servo Motor

Typical servos expect to see a pulse every 20ms, and the width of this signal determines the position. This width is usually between one and two milliseconds. This type of signal control is frequently referred to as Pulse Width Modulation (PWM).

9.2 Servo.h

We are going to use the Servo.h library. There are a number of libraries that are automatically added into your Particle DeviceOS code, Servo.h is one of them. To define object that is part of the class Servo, the following command is used:

```
1 // myServo is the name of an object from class Servo
2 Servo myServo
```

- servo.attach(pin) - Attach the Servo variable to a pin
- servo.write(angle) - Writes a value (integer) to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation.

- servo.read() - Read the current angle of the servo (the value passed to the last call to write()). This returns an integer.

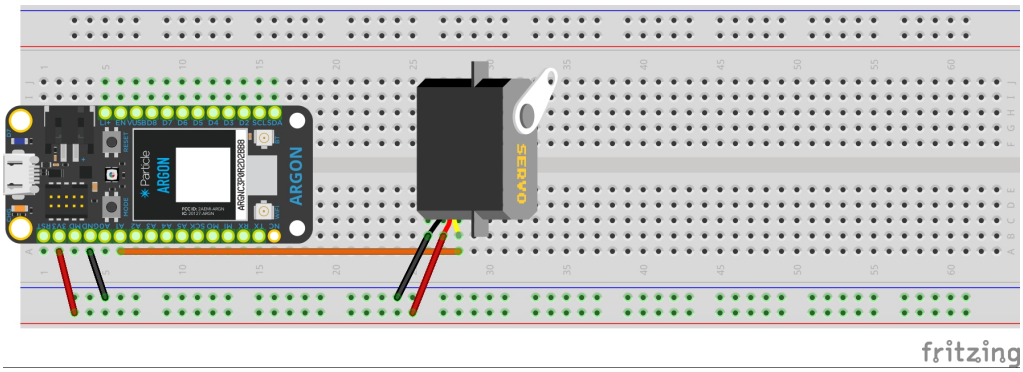


Figure 9.2: Servo Wiring

9.3 Assignment Particle 1 - Part I

You will find in the Particle_01 git classroom repository a few files, including the Fritzing part for the Particle Argon. Import this part in Fritzing.

1. First in your notebook, and the in Fritzing, create a breadboard diagram on the Particle Argon with two Servo motors and a button (with a pull down resistor).
2. Transfer this diagram to your breadboard
3. Create a new Visual Studio code (VScode) project (saving it into the Particle_01 repository as the Parent Folder). This code should move one of the Servo motors from 0 to 180 degrees (the motor's full range) and then back to 0.
4. Create a new project in VScode (also saved to the Particle_01 repository). Have one Servo move in along a sawtooth pattern from L02_HelloLED (L02_04_helloLEDsaw). For the other Servo, use the sin function from L02_05_helloLEDsin to move the Servo along a sine wave.

5. Next add in a button that when pressed, starts or stops the motion of the Servo.

NOTE:

- In HelloLED, the sine function ranged over analogWrite() range for the Teensy (0 - 255).
- You will need to #include < *math.h* > in order to the the function sin(). For the Servo, go from 0 to 180.

9.4 I2C Encoder - SparkFun Qwiic Twist RGB

We are going to incorporate an I2C encoder into our Servo project. We are going to utilize the SparkFun_Qwiic_Twist_Arduino_Library.h library. There is a copy of this library in the Particle_01 repository. In the Assignment, we will learn how to manually add the library into the Project Folder, so that it is accessible to your code.

The SparkFun_Qwiic_Twist_Arduino_Library.h library has a number of methods that allow us to work with the Qwiic Twist encoder. First, we need to define an object, twist, from the TWIST class and to initialize the encoder as follows:

```
#include <SparkFun_Qwiic_Twist_Arduino_Library.h>
TWIST twist; //Create instance of this object

void setup() {
  Serial.begin(9600);
  Serial.println("Qwiic Twist Example");

  if(twist.begin() == false)
  {
    Serial.print("Twist does not appear to be connected.");
    Serial.println("Please check wiring. Freezing...");
    while(true);
  }
}
```

The SparkFun_Qwiic_Twist_Arduino_Library.h library methods are:

- twist.begin() - initializes the sensor with basic settings and returns false if sensor is not detected
- int twist.getCount() - returns the number of indents the user has twisted the knob
- int twist.getDiff() - returns the difference between the last two getCount() calls
- bool twist.isMoved() - returns true if knob has been twisted
- int twist.timeSinceLastMovement() - returns the number of milliseconds since the last encoder movement
- bool twist.isClicked() - returns true if a click event has occurred
- int twist.timeSinceLastPress() - returns the number of milliseconds since the last button event (press and release)
- bool twist.isPressed() - returns true if button is currently being pressed
- void twist.setCount(pos) - set the encoder count to a specific pos
- void twist.setColor(R,G,B) - sets the color of the encoder. R, G, and B range from 0 - 255.

9.5 Assignment Particle 1 - Part II

- Add the I2C encoder to your Fritizing diagram. Unfortunately, there is not a Qwiic Twist fritizng part. So, use the BMP085 (which is not an encoder at all) as it has the same pin-out as your encoder. Connect the encoder to your breadboard using the Particle Argon I2C pins.
- From the File Menu, open the I2C_scan folder. Compile and Flash this code to your Argon. Validate in the Serial Monitor that your encoder is seen on the I2C bus.

```
Scanning...
I2C device found at address 0x3F  !
done
```

- Create a new project (called encoder) and use the SparkFun_Qwiic_Twist_Arduino_Library.h library write the encoder position to the Serial monitor. In order to use this library:
  1. Go to Documents/Particle/particle-1-redo-yourname/encoder
  2. Create a directory called lib
  3. Copy SparkFun_Qwiic_Twist_Arduino_Library folder into the new lib folder that you created
- Next, create code to control one of the Servo motors by turning the encoder.
- Finally, update your code to randomly change the color of the encoder when the encoder button is pressed.

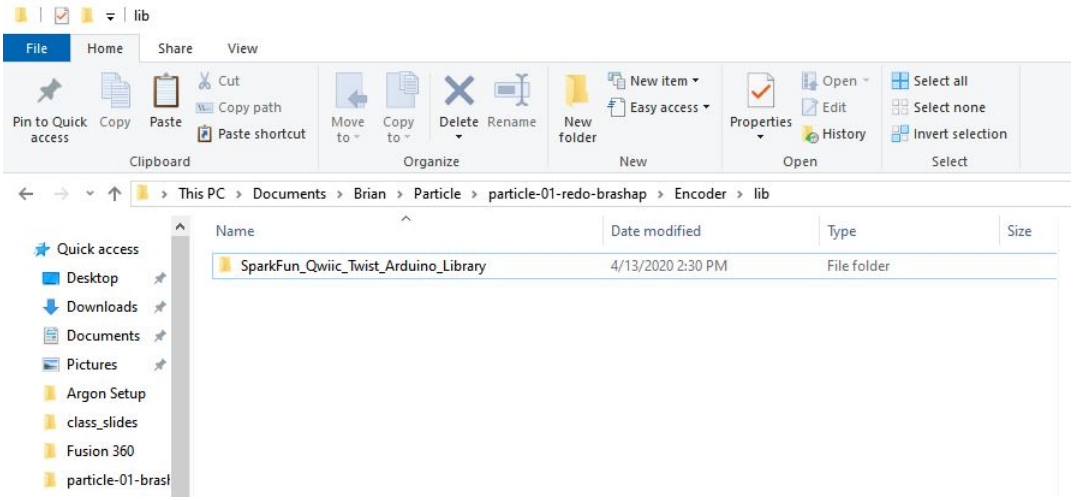


Figure 9.3: Library Added

In our next assignment, we will learn to use Particle:Install Library when we add the OLED to our board.

Chapter 10

Particle Lesson 2

Today we will learn how to install Libraries with VScode and how to keep track of time. First, let's discuss the Soil Moisture sensor.

10.1 Soil Moisture Readings

There are two types of soil moisture sensors. The first is a Resistive Moisture Sensor. The sensor consists of two probes which are used to measure the volumetric content of water. The two probes allow the current to pass through the soil and then it gets the resistance value to measure the moisture value.

When there is more water, the soil will conduct more electricity which means that there will be less resistance which corresponds to a higher moisture level. Dry soil conducts electricity poorly, so when there will be less water, then the soil will conduct less electricity which means that there will be more resistance. And, thus, a lower moisture level.

Unfortunately, if the resistive sensor is left with current applied in wet soil, then electroplating activity occurs that strips one electrode in the matter of a month. While there are ways of getting around this but only intermittently applying current to take a measurement, there are alos Capacitive Moisture Sensors that don't suffer this issues.

Figure 10.1: Resistive Soil Moisture Sensor

As the name suggests, the sensor is based on a capacitor. These consist of a positive plate, a negative plate and the space in-between the plates, known as the dielectric. A capacitive moisture sensor works by measuring the changes in capacitance caused by the changes in the dielectric. It does not measure moisture directly (pure water does not conduct electricity well), instead it measures the ions that are dissolved in the moisture These ions and their concentration can be affected by a number of factors, for example adding fertilizer for instance will decrease the resistance of the soil. Capacitive measuring basically measures the dielectric that is formed by the soil and the water is the most important factor that affects the dielectric.

Capacitive measuring has some advantages, It not only avoids corrosion of the probe but also gives a better reading of the moisture content of the soil as opposed to using a resistive soil moisture sensor. Since the contacts (the plus plate and the minus plate of the capacitor) are not exposed to the soil, there is no corrosion of the sensor itself.

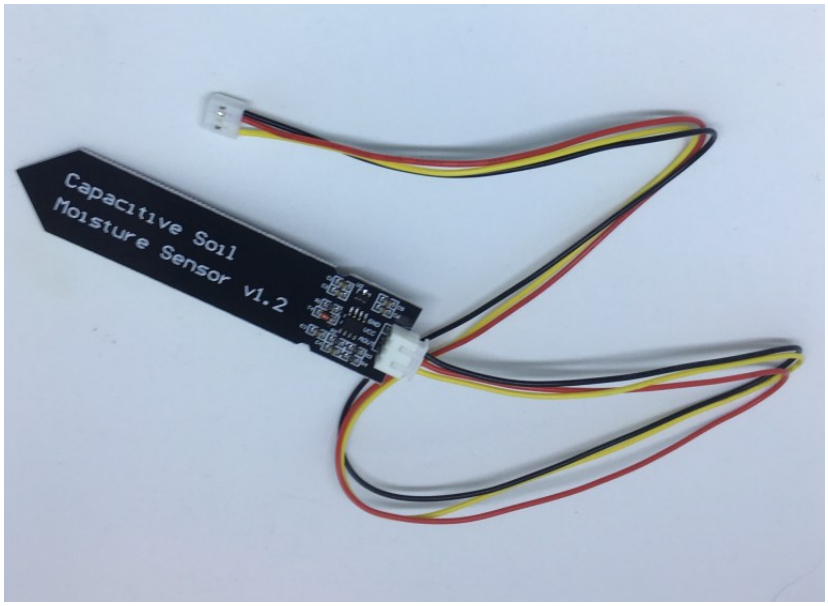


Figure 10.2: Capacitive Soil Moisture Sensor

10.2 Current Time

The Argon has the ability to get the current time from the Particle Cloud. This is done by using the time functions that are included with Particle.h. This can be accomplished using the following code:

```
/*
 * Project Time_Sync
 * Description: Function to Sync Time to Particle Cloud
 * Author: Brian Rashap
 * Date:   13-Jan-2020
 */

#include <Particle.h>

void setup() {
  Serial.begin();
  while(!Serial); // wait for Serial monitor
  Serial.println("-------- Begin Time Sync --------");
}

```

```
16 void loop() {
17   sync_my_time();
18   delay(random(10000,60000));
19 }
20
21 void sync_my_time() {
22   Time.zone(-6); // Set Time Zone to MDT (UTC - 7)
23   unsigned long cur = millis();
24
25     // Request time synch from Particle Device Cloud
26     Particle.syncTime();
27
28     // Wait to receives time from Particle Device Cloud
29     waitUntil(Particle.syncTimeDone);
30     // Check if synchronized successfully
31     if (Particle.timeSyncedLast() >= cur)
32     {
33       // Print current time
34       Serial.printlnf("Current time is %s",Time.timeStr());
35     }
36 }
```

10.3 Particle: Install Library

In order to Install Libraries directly from Visual Studio Code, you can use the Command Palette.

- Use control-shift-p (or command-shift-p on a MAC) to open the Command Palette.
- Use the command Particle: Find Libraries to search for available libraries
- Use the command Particle: Install Library to install the library that you intend to use. Unlike the Arduino IDE, you need to install the library into each individual project where you expect to use it.

Now, let's practice. From a new window in Visual Studio Code, open up the Folder OLED_Test for the Git Classroom repository. First find the available

libraries using the command Particle: Find Libraries and search for SSD1306. Now, install the Adafruit_SSD1306 library. Compile and then Flash the code to validate that you installed this correctly. Notice that this library allows us to use the full 128x64 pixels on the OLED display.

10.4 Assignment Particle 2

1. Draw a fritzing diagram of a soil moisture sensor (using the sm_cap.fzpz in the Particle_02 repository), along with a OLED display. Using your OLED from your Teensy board, build this circuit on our breadboard.
2. Run, I2C_Scan to validate that the OLED display is correctly wired.
3. Create a new project in Particle_02-yourname respository. Write code to
   - Take an analogRead() from the soil moisture sensor every 3 minutes.
   - Get the current time from the Particle Cloud
   - Display the current time and the moisture reading on OLED
4. Take a plant or cup of soil (relatively dry) and do the following four measurements. Take four measurements of each (over a 10 minute period) and record the results in your lab notebook.
   - Moisture sensor in an empty cup
   - Moisture sensor in a cup of water filled to the "notch" on the sides of the sensor.
   - Moisture sensor in the soil with (our without) a plant
   - Moisture sensor in that same soil after it has been watered.

Chapter 11

Particle Lesson 3 - Adafruit.io

We will be using the MQTT protocol to send our first data to the cloud, in this case adafruit.io.

11.1 MQTT

MQTT stands for Message Queuing Telemetry Transport. It is a publish/-subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

11.1.1 MQTT security

You can pass a user name and password with an MQTT packet in V3.1 of the protocol. Encryption across the network can be handled with SSL,

Figure 11.1: MQTT

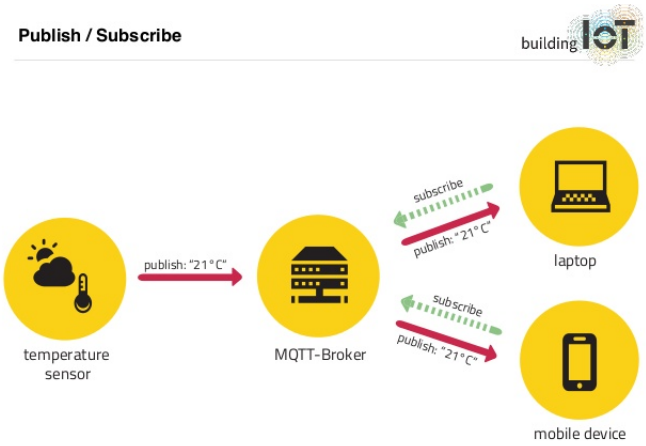
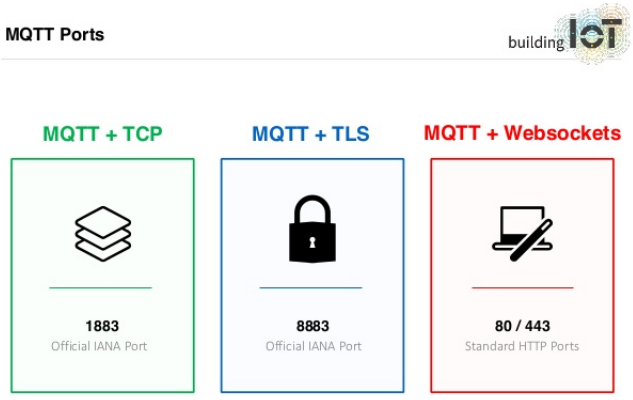


Figure 11.2: MQTT Sockets



independently of the MQTT protocol itself (it is worth noting that SSL is not the lightest of protocols, and does add significant network overhead). Additional security can be added by an application encrypting data that it sends and receives, but this is not something built-in to the protocol, in order

to keep it simple and lightweight.

11.2 Adafruit.io

Figure 11.3: Adafruit.io

Adafruit.io is a cloud service - that just means Adafruit run it for you and you don't have to manage it. You can connect to it over the Internet. It's meant primarily for storing and then retrieving data but it can do a lot more than just that!

- Display your data in real-time, online
- Make your project internet-connected: Control motors, read sensor data, and more!
- Connect projects to web services like Twitter, RSS feeds, weather services, etc.
- Connect your project to other internet-enabled devices
- The best part? All of the above is do-able for free with Adafruit IO

11.2.1 Getting Started with Adafruit.io

Here are the steps to set-up your first Adafruit.io Dashboard:

1. Create an adafruit.io account at `https://accounts.adafruit.com/users/sign_up/`
2. Next, you will need your adafruit.io username and key. There is a link in the upper right of your adafruit.io webpage labeled: Adafruit IO Key. Click on it and copy your username and io_key.
3. Now, we will create variables (feeds) within adafruit.io where your data will be stored. Go to the feeds tab (view all, if needed) and click on actions and select Create a New Feed. Create two feeds: Moisture and Temperature.
4. Next, we will create a dashboard. On the dashboard tab, click on actions, and then Create a New Dashboard. Name your dashboard. You can also at this point give it a description and/or upload an image.
5. Finally, we will add Blocks to your dashboard. Click on your dashboard to open it.
    - Click the + symbol to add your first block

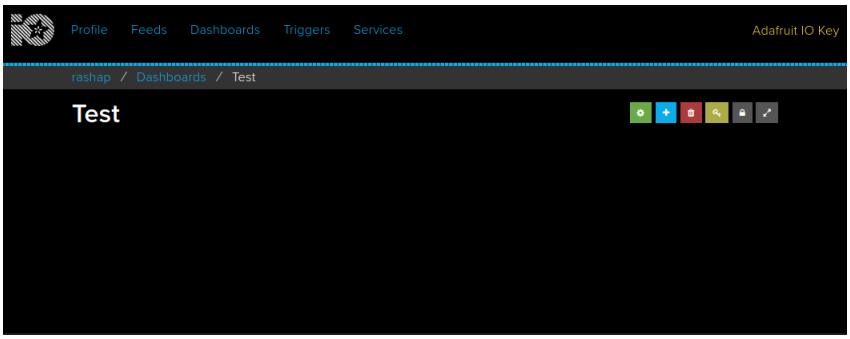


Figure 11.4: Black Dashboard

- Select the Line Chart

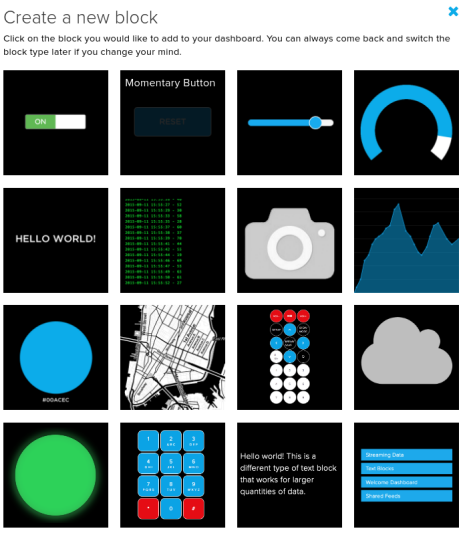


Figure 11.5: Types of Blocks

- Select the Feeds (up to 5) that you want on the chart.

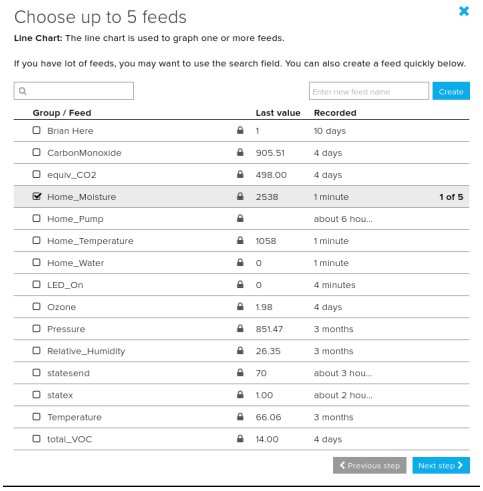


Figure 11.6: Adding Feeds

When all is done, you can make a dashboard like this (but without data):

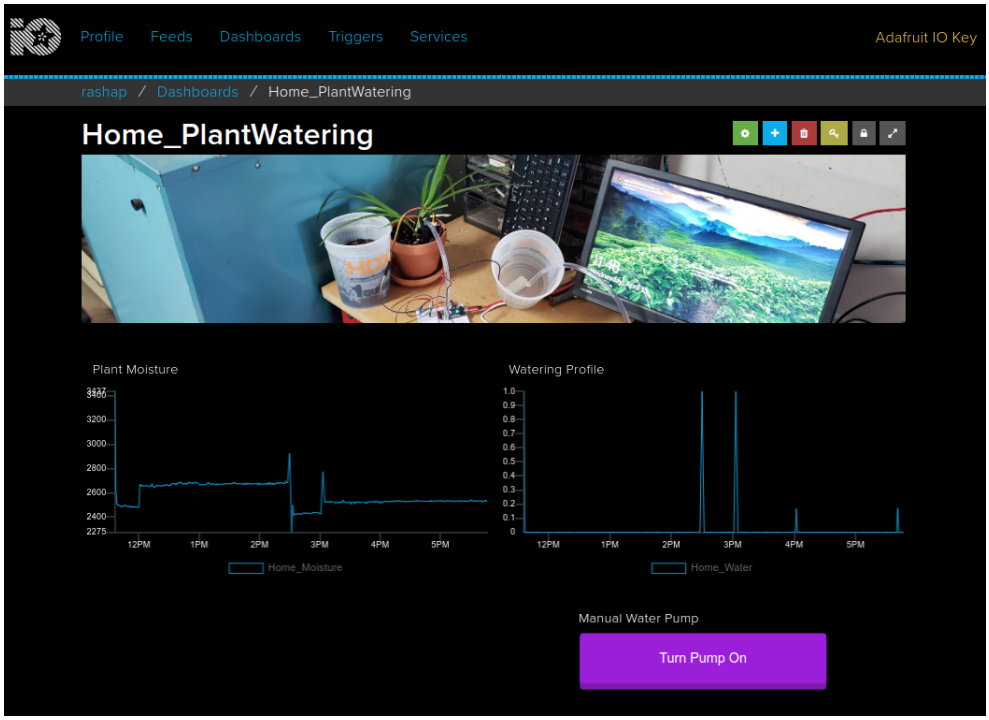


Figure 11.7: Brian's PlantWater Dashboard

11.3 Using MQTT with Adafruit IO

The following is needed to connect a MQTT client to Adafruit IO:

- Host: *io.adafruit.com*
- Port: 1883 (or 8883 for SSL encrypted connection)
- Username: your Adafruit account username (see the accounts.adafruit.com page here to find yours)
- Password: your Adafruit IO key (click the AIO Key button on a dashboard to find the key)

Adafruit IO's MQTT API exposes feed data using special topics. You can publish a new value for a feed to its topic, or you can subscribe to a feed's topic to be notified when the feed has a new value:

<username>/feeds/<feedname>

Where <username> is your Adafruit IO username (the same as specified when connecting to the MQTT server) and <feedname> is the feed's name or key.

Here is the example setup for the Particle DeviceOS:

```
#include <Adafruit_MQTT.h>

// All this needs to be included too
#include "Adafruit_MQTT/Adafruit_MQTT.h"
#include "Adafruit_MQTT/Adafruit_MQTT_SPARK.h"
#include "Adafruit_MQTT/Adafruit_MQTT.h"

/******** Adafruit.io Setup ********/
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883      // use 8883 for SSL
#define AIO_USERNAME  "<username>"
#define AIO_KEY       "<key>>"

/*****Global State (you don't need to change this!) *****/
TCPClient TheClient;

/* Setup the MQTT client class by passing in the WiFi client
   and MQTT server and login details.  */
Adafruit_MQTT_SPARK mqtt(&TheClient,AIO_SERVER,AIO_SERVERPORT
   ,AIO_USERNAME,AIO_KEY);

/************ Feeds ************/
// Setup a feed called <object> for publishing.
// Notice MQTT paths for AIO follow the form: <username>/
   feeds/<feedname>

Adafruit_MQTT_Publish <object1> = Adafruit_MQTT_Publish(&mqtt
   , AIO_USERNAME "/feeds/<feedname1>");
Adafruit_MQTT_Publish <object2> = Adafruit_MQTT_Publish(&mqtt
   , AIO_USERNAME "/feeds/<feedname2>");
```

In order to publish to Adafruit IO, you simply use

```
void loop() {
  feed1 = random(1,100);
  feed2 = random(1,1000);

 if(mqtt.Update()){
  <object1>.publish(feed1);
  <object2>.publish(feed2);
 }
 delay(30000);
}
```

11.4 Assignment Particle 3 - Part I

1. Create your first dashboard using the instructions in Section 11.2.1.
2. In Visual Studio Code, open the Folder AdaTrial and set it up to send two data streams to Adafruit.io. Note, the free version of Adafruit.io has a limit on the number of data points sent every hour, so don't shrink the delay too much.
3. Add your BME280 sensor to your fritzing diagram of the soil monitor. Save the new fritzing in Particle_03 repository. Build this circuit on your board, testing BME280 with I2C_Scan.
4. Create a new Project (copying over your code from the AdaTrial) using Particle_03 repository as our Parent Folder. Add in the code for BME280 measurements with the appropriate library. Use L10_02_BME280 in instructor_master as a reference if needed. Also, be sure to put the appropriate I2C address into bme.begin().
5. Add in the Adafruit.io header information, set up feeds from Soil Moisture and Temperature. Publish to Adafruit.io.
6. In Adafruit.io, setup a dashboard with your two feeds. Send a screen shot of your dashboards to your fellow students in iot-cohort1.

11.5 Subscribing to Data from Adafruit.io

In the assignment, we will add to the Adafruit.io dashboard. However, there is also changes you need to make to your code in all three sections: the Header, void setup(), and void loop(). Can you tell what the below will do, just by reading the below code?

In the Header add:

```
Adafruit_MQTT_Subscribe onoffbutton = Adafruit_MQTT_Subscribe
    (&mqtt, AIO_USERNAME "/feeds/LED_On");

int button;
int i;
```

In void setup() add:

```
pinMode(D7,OUTPUT);   // set the onboard LED as an output

// Setup MQTT subscription for LED_On feed.
mqtt.subscribe(&onoffbutton);
```

In void loop() add:

```
for(i=0;i<18;i++) {    //18x10sec loops = 3min delay
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(10000))) {
    // do this loop for 10 seconds
    if (subscription == &onoffbutton) {
      button = atoi((char *)onoffbutton.lastread);
      //atoi converts adafruit string to int
      Serial.printf("Button State is %i",button);
      digitalWrite(D7,button);
    }
  }
}
```

11.6 Assignment Particle 3 - Part II

1. Subscribe from Dashboard

(a) Start by adding a button to your dashboard to control the LED on your Argon. Follow the steps in Section 11.2.1 to create a feed called LED_On and then add a button associated with this feed to your dashboard. The button on your dashboard should have Pressed Value of 1 and a Released Value of 0.

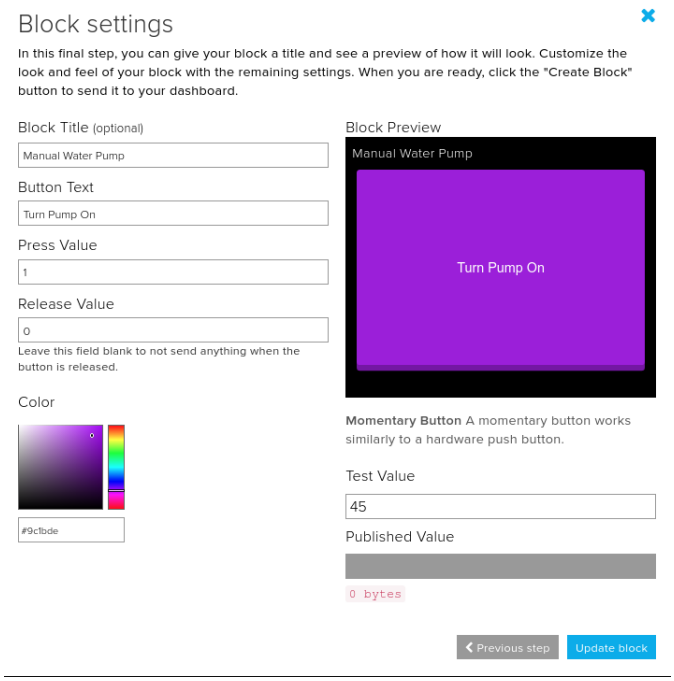


Figure 11.8: LED_On Setup

(b) Incorporate the code from Section 11.5 to your latest Soil Monitoring project.

(c) Run your code and see if you can control the on-board LED from your dashboard. Troubleshoot as needed.

(d) Explore the various blocks available to be added to your dashboard. Implement a few of them. Send a screen shot of your dashboards to your fellow students in iot-cohort1.

2. Seeed Air Quality Sensor - the air quality sensor needs to initially burn in its element for a few days to get accurate readings. Connect the V_{cc}

(3.3V) and GND to one of your microcontrollers and let it stay powered on for 96 hours. If you have the NCD CO sensor, do the same with it as well.

3. We will start focusing on our capstone projects over the next week. You can either do your capstone individually in a group of two or three. Please use the Student Projects GIT classroom `https://classroom.github.com/a/u8ljFbOJ` to document your capstone idea.

4. Continue to work on your Smart Room Controller. We will be testing them when we get back into class the first week in May. Remember, this is your midterm assignment and needs to be working to graduate.

Chapter 12

Particle Lesson 4

In this chapter, we will learn to send data to our second cloud service, Thingspeak™. While Thingspeak™ allows for MQTT data transfer, similar to Adafruit.io, we are going to learn how to send data to the Particle Cloud and then use a WebHook to transfer the data from the Particle Cloud to a third-party app. In order to do this, we need to "package the data in a data-interchange format JSON. We will update our last project to send data to both cloud services. We will also in this chapter be adding the "watering" part to our Plant Watering Project by integrating a pump into our project.

12.1 Particle Cloud

The Particle Cloud is the powerful centerpiece of the Particle platform, handling many of the most complex pieces of creating an IoT product. From robust security, to reliable infrastructure, to a flexible integrations system, the Particle Cloud has everything you need to move quickly and succeed. The Particle Cloud provides many things needed to create and build a connected product :

- Pre-provisioned connectivity modules with reprogrammable microcontrollers

- Managed cloud infrastructure for two way communications
- Pre-built integrations with and tools to stream data to databases and third-party web-services
- Resource efficient communications protocols and messaging encryption stack
- APIs for interacting with devices virtually
- Device management software for overseeing a fleet of connected products
- Firmware release tooling for reprogramming a fleet of connected devices

12.1.1 JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

12.1.2 JsonParserGeneratorRK.h

Creating objects in JSON is straighforward, but is also tedious. Here's an example of code needed to Publish to the Particle Cloud in a format that is ready to be used with a WebHook.

```
// Prepare the Publish Payload using snprintf
snprintf(msg, sizeof(msg),
//ThingSpeak Field #1,ThingSpeak Field #2,ThingSpeak WriteKey
  "{\"1\":\"%.1f\"      ,\"2\":\"%.1f\"      \"k\":\"%s\"}",
// Float for Field #1,Float for Field #2 ,ThingSpeak WriteKey
    field1          ,   field2          ,   myWriteAPIKey);

// More fields can be published by adding to the snprintf.

//Publish to the particle cloud using:
Particle.publish(eventName, msg, PRIVATE, NO_ACK);
```

Luckily, there is a JSON Parser available to simplifies this process. We begin by including the header file. Make this the first #include

```
#include <JsonParserGeneratorRK.h>
```

Next we create a function to package and send your data to the Particle Cloud.

```
void createEventPayLoad(int moistValue, float tempValue,
    float presValue, float humValue, int waterED) {
  JsonWriterStatic<256> jw;
  {
    JsonWriterAutoObject obj(&jw);

    jw.insertKeyValue("Moisture", moistValue);
    jw.insertKeyValue("Temperature", tempValue);
    jw.insertKeyValue("Pressure", presValue);
    jw.insertKeyValue("Humidity", humValue);
    jw.insertKeyValue("Plant Watered", waterED);
  }
  Particle.publish("env-vals",jw.getBuffer(), PRIVATE);
}
```

Finally, we call the function in void loop() using:

```
createEventPayLoad(moist,temp,pres,hum,watered);
```

12.2 ThingSpeak Visualization and Analytics

ThingSpeak™ is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak. With the ability to execute MATLAB® code in ThingSpeak you can perform online analysis and processing of the data as it comes in. ThingSpeak is often used for prototyping and proof of concept IoT systems that require analytics.

12.2.1 Creating a ThingSpeak™ Channel

After creating a ThingSpeak™ account. We begin by creating our first Channel.

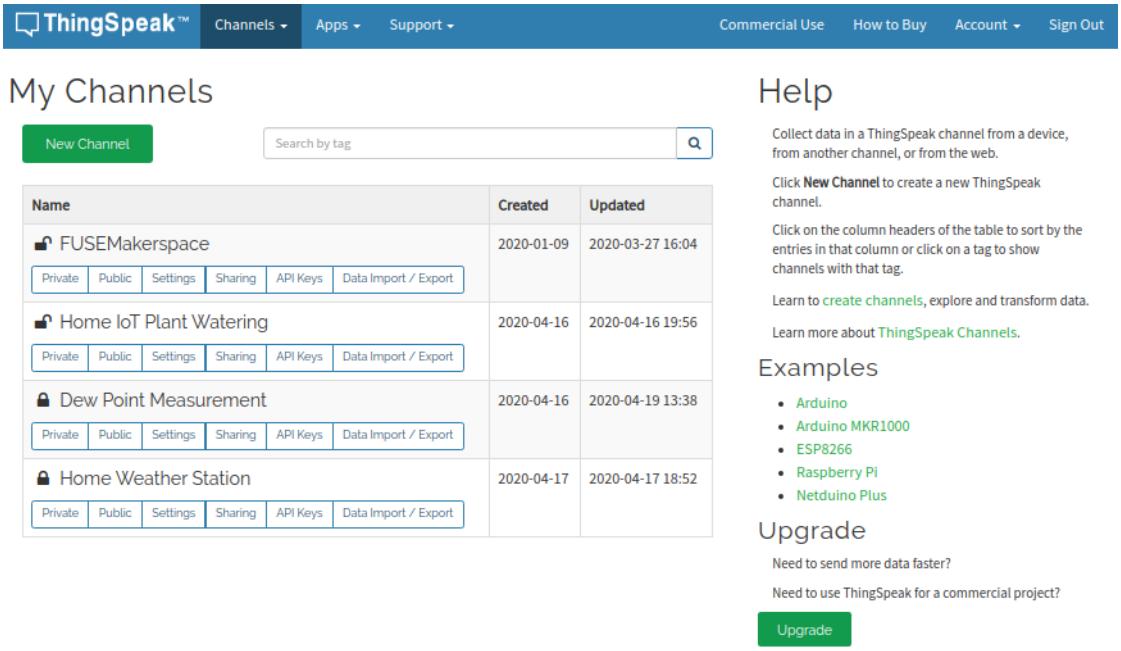


Figure 12.1: ThingSpeak Channel

Once the Channel is created, we have several tabs. Private/Public View, Settings, Sharing, API Keys, and Data Import/Export. Selecting the API Keys tab, we copy down our Write API Key. This will allow our WebHook to send data to the Channel.

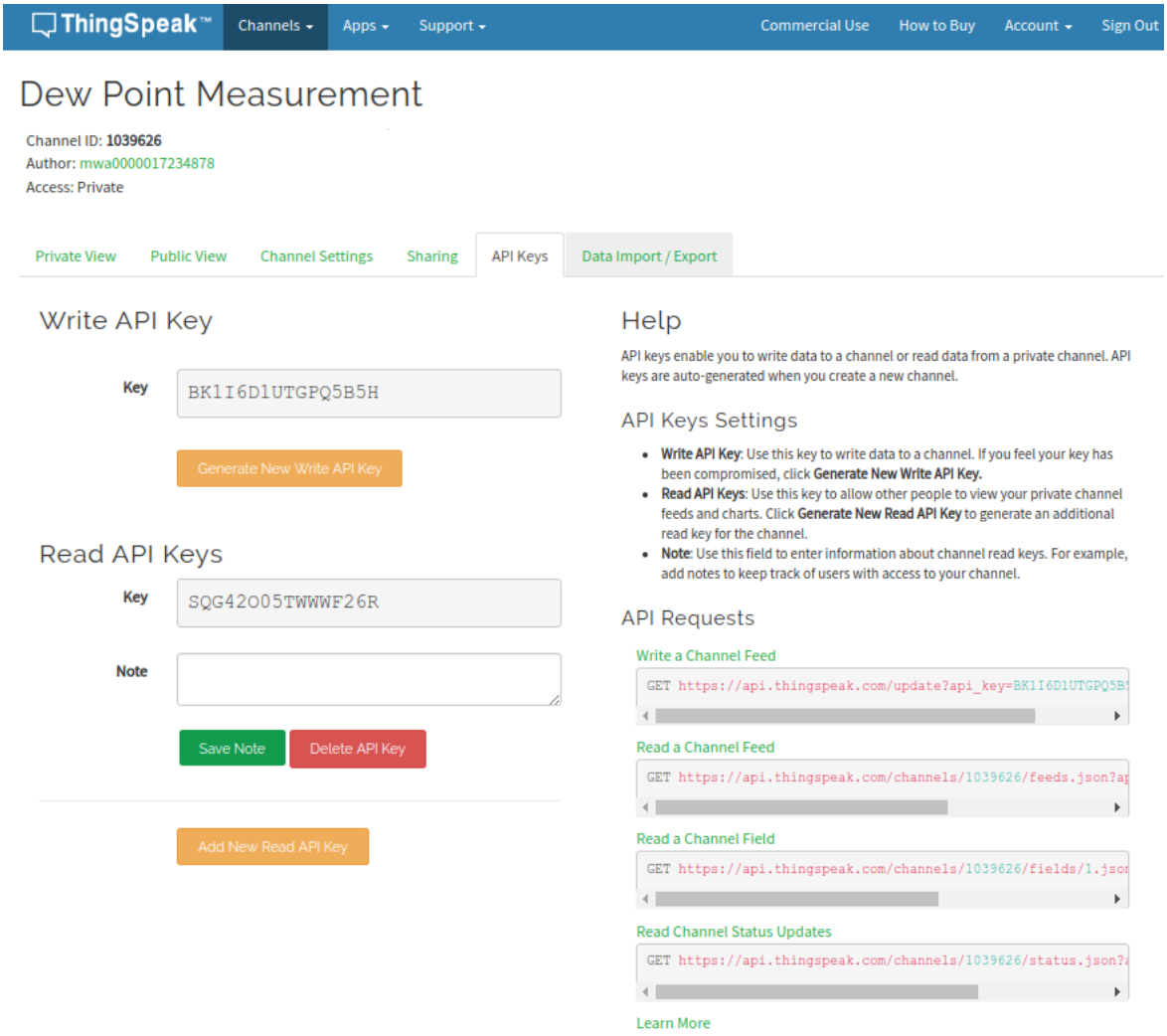


Figure 12.2: ThingSpeak API Tab

12.2.2 Particle Cloud Webhooks

To send data to ThingSpeak™, we next create a Webhook from the Paricle Console (`console.particle.io`). From the Integrations tab, we start by creating a New Integration.

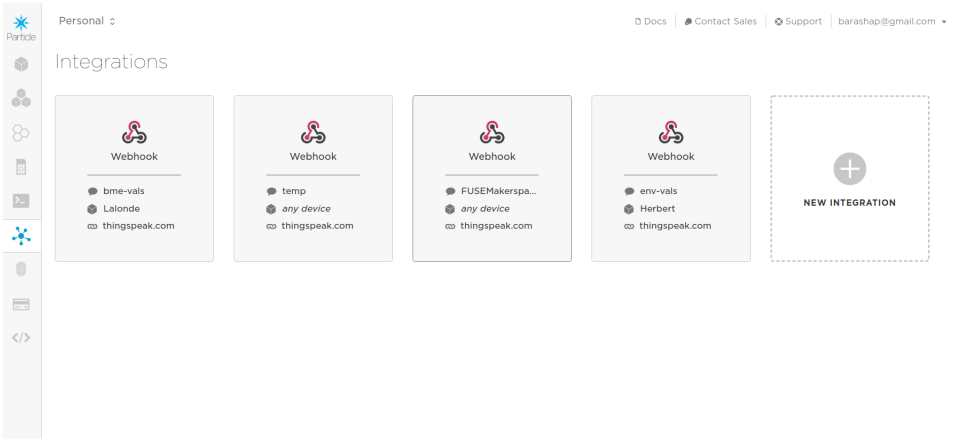


Figure 12.3: Particle Cloud Integrations

We fill in the appropriate fields, being use to use Advanced Settings and Custom. The Device field can be left blank if you don't want to restrict with of your Particle Argons can send data via this Webhook. Also, note that the JSON object is enclosed in three {}, for example {{{Moisture}}}. These object names should match the names used the createEventPayLoad() function from the preceeding section.

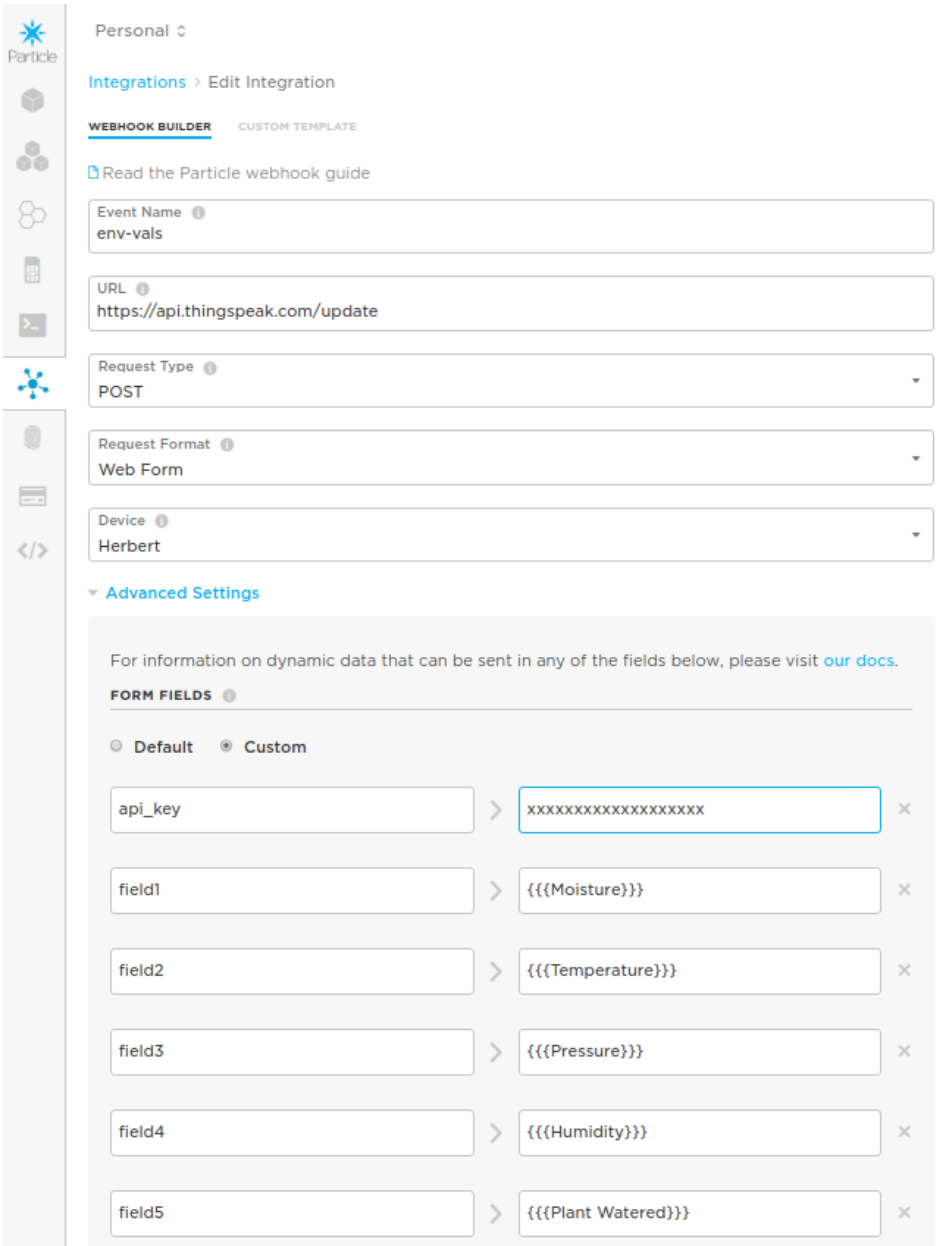


Figure 12.4: Particle Cloud Webhooks

12.2.3 Updating your ThingSpeak™ Channel

Finally, you go back to your ThingSpeak™ Channel, select the fields you want to display, give them the names of your JSON objects, and set the Metadata field to JSON.

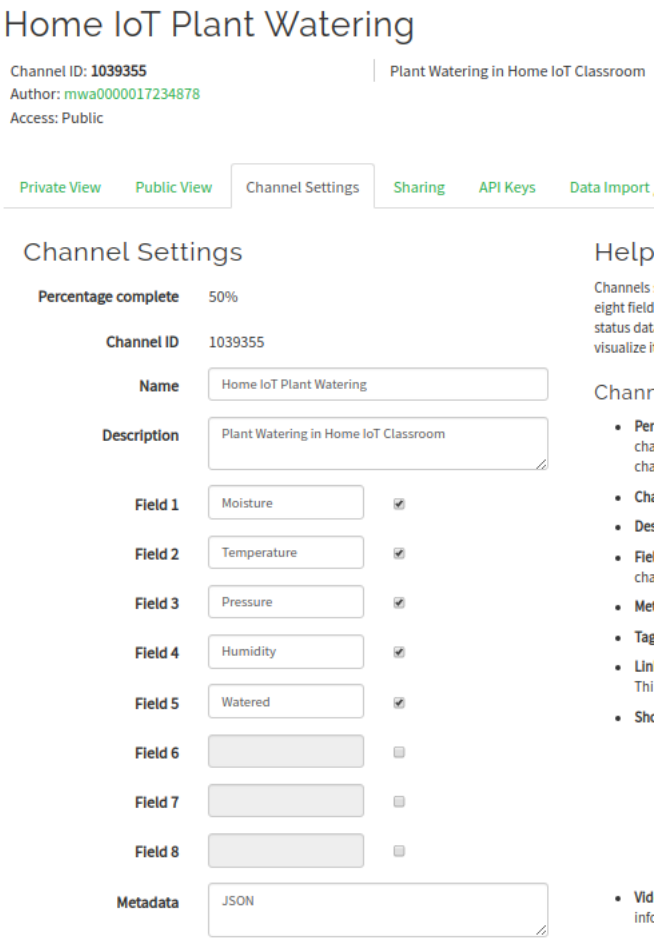


Figure 12.5: ThingSpeak Channel Setup

Your dashboard will start to update with the next data push from your code. You can first see the data on the Particle Console from the Events Tab.

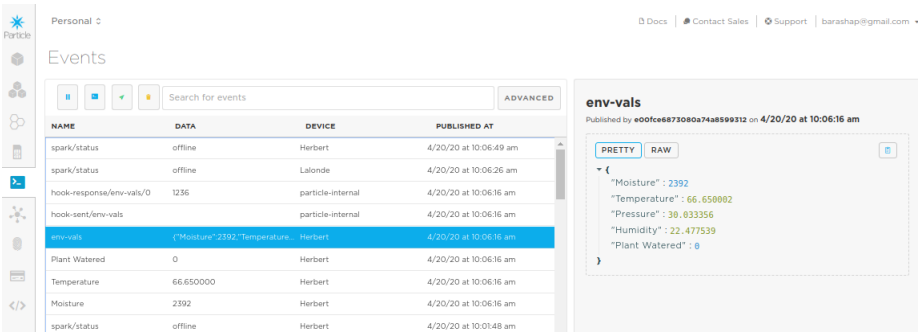


Figure 12.6: Particle Cloud Events

And, then you can see your Dashboard in action. You can add guages and indicators using the Add Widgets button. You can change the colors of your lines, by editing the graph. The hex codes are found at `https://htmlcolorcodes.com/color-picker/`.

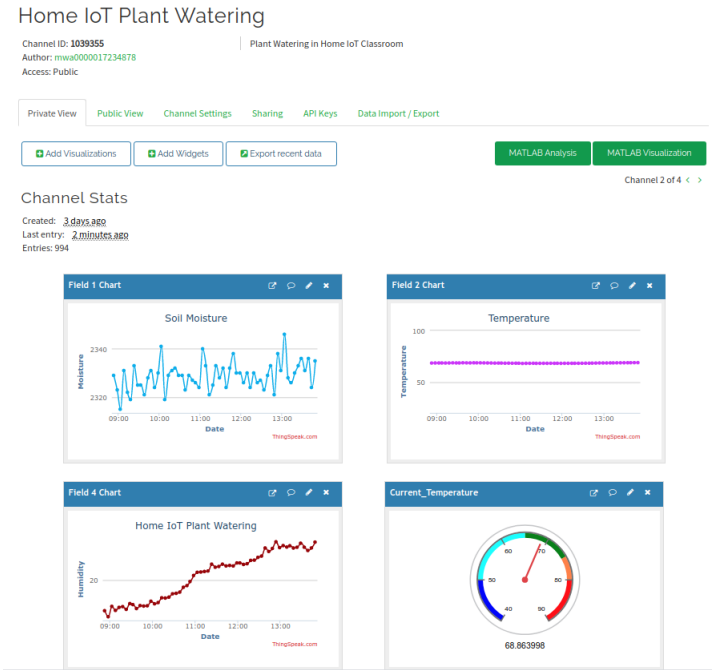


Figure 12.7: ThingSpeak Dashboard

12.3 Relays

Our microcontroller operates at 3.3V and can send digital signals at 3.3V or GND to operate devices. However, often the instrument being operated requires more voltage (5V, 12V, 24V, 120V, 240V, etc.). Control of these devices is accomplished by the use of a relay. Relays are simple switches which are operated both electrically and mechanically. Relays consist of an electromagnet and a set of contacts. The switching mechanism is carried out with the help of the electromagnet.

The main operation of a relay comes in places where only a low-power signal can be used to control a circuit. It is also used in places where only one signal can be used to control a lot of circuits. The application of relays started during the invention of telephones. They played an important role in switching calls in telephone exchanges. They were also used in long distance telegraphy. They were used to switch the signal coming from one source to another destination. After the invention of computers they were also used to perform Boolean and other logical operations.

The high end applications of relays require high power to be driven by electric motors and so on. Such relays are called contactors. It is an electro-magnetic relay with a wire coil, surrounded by an iron core. A path of very low reluctance for the magnetic flux is provided for the movable armature and also the switch point contacts. The movable armature is connected to the yoke which is mechanically connected to the switch point contacts. These parts are safely held with the help of a spring. The spring is used so as to produce an air gap in the circuit when the relay becomes de-energized.

The diagram shows an inner section diagram of a relay. An iron core is surrounded by a control coil. As shown, the power source is given to the electromagnet through a control switch and through contacts to the load. When current starts flowing through the control coil, the electromagnet starts energizing and thus intensifies the magnetic field. Thus the upper contact arm starts to be attracted to the lower fixed arm and thus closes the contacts causing a short circuit for the power to the load. On the other hand, if the relay was already de-energized when the contacts were closed, then the contact move oppositely and make an open circuit.

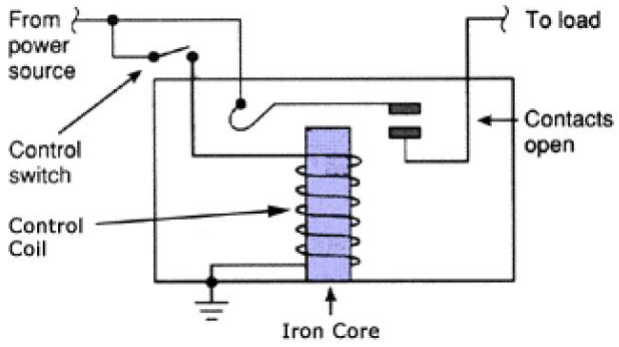


Figure 12.8: ThingSpeak

As soon as the coil current is off, the movable armature will be returned by a force back to its initial position. This force will be almost equal to half the strength of the magnetic force and is provided by a spring.

12.4 Assignment Particle 3 - Part III

Please continue to use /Documents/<Your Name¿>/particle-03 as the Parent folder for the below assignments. Be sure to git push at the end of each day.

1. Modify your Soil Monitor code to send measurements to the Particle cloud in JSON format.
2. Go to ThingSpeak and create an account: `https://thingspeak.com/users/sign_up`
3. Create your first ThingSpeak channel. Go the the API Keys tab and copy down your Write API Key.
4. Go to Particle Console and setup a WebHook between the data from your Argon and your ThingSpeak Channel.

5. Now, using the wiring diagrams from Section 12.4.1, add the Relay and Water Pump to your project. Submerge the pump in a glass of water.

   (a) Draw your own Fritzing diagram.

   (b) Set a threshold variable to your current soil measurement minus 100. If you have scaled your analog input, then adjust the subtraction by the same scaling. Add in an IF-statement that turns on the pump for approximately 1.5 seconds each half hour is the moisture measurement is above the threshold.

   (c) Add a button to your Adafruit.io dashboard that manually turns on the pump.

   (d) Add a variable to your dashboard(s) that is a 1 if the pump was turned on, and a 0 if it was not.

12.4.1 Wiring the Relay

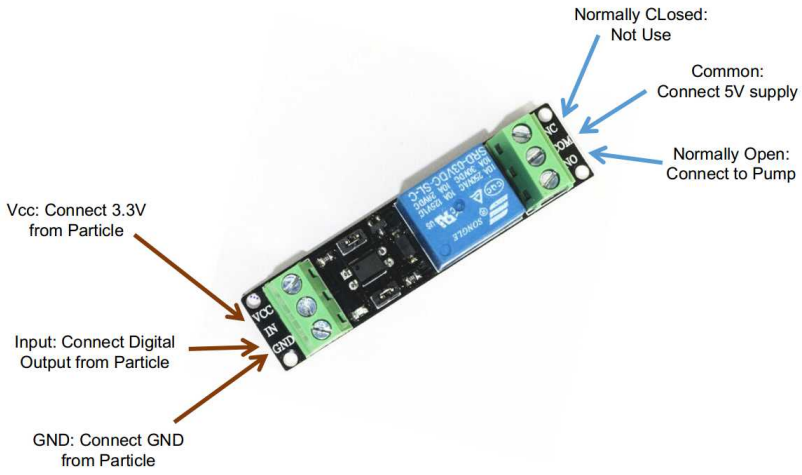


Figure 12.9: Wiring the Relay

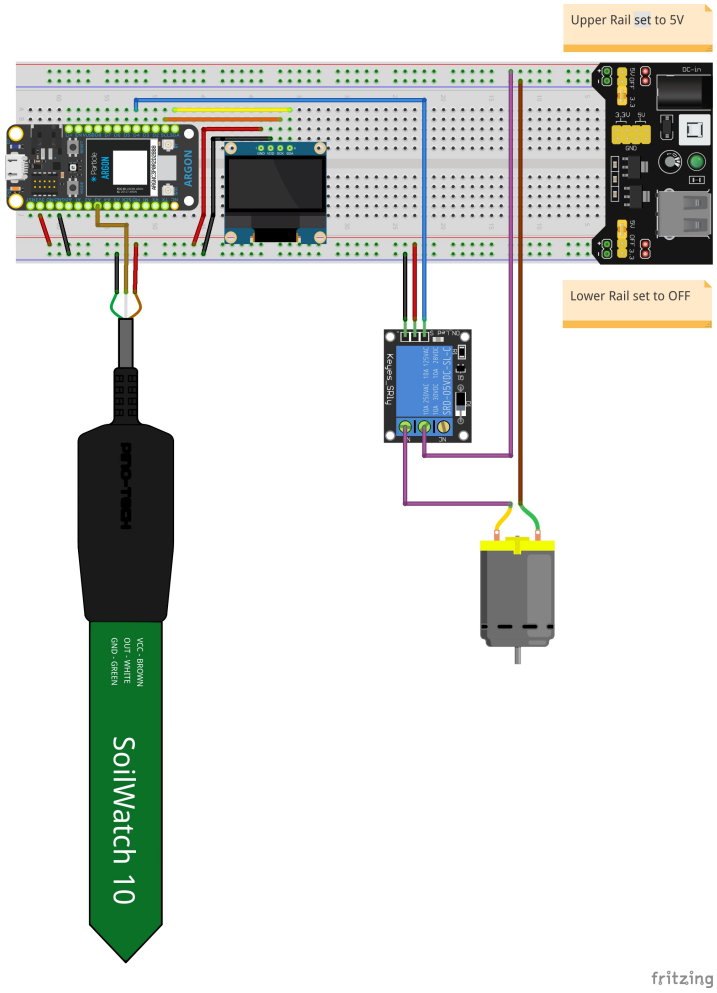


Figure 12.10: Using BreadBoard Power Supply

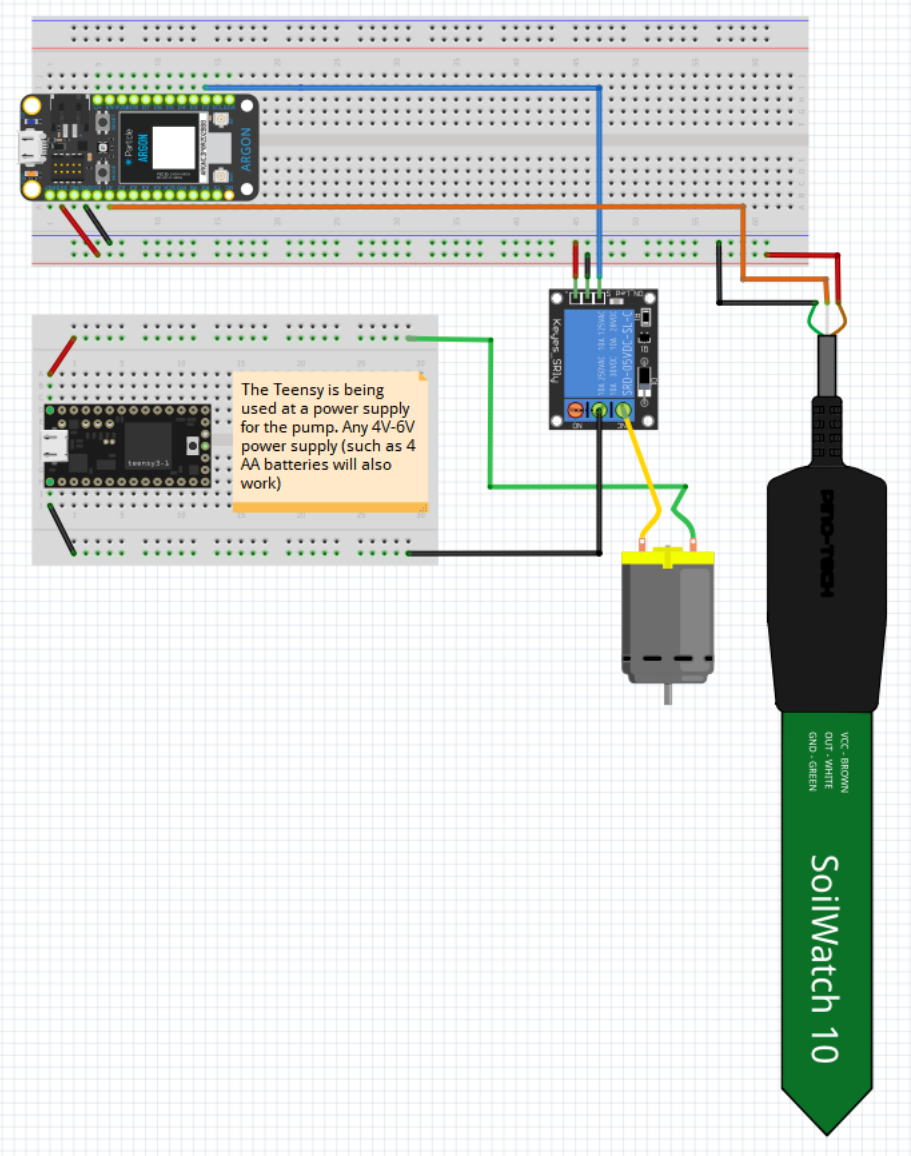


Figure 12.11: Using Teensy for 5V supply

Chapter 13

Particle Lesson 5

In this lesson, we are going to learn how to integrate two sensors into our Soil Moisture project by reviewing online information about the sensors.

13.1 pulseIn()

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

The syntax is as follows

```
pulseIn(pin, value)
pulseIn(pin, value, timeout)
```

Here is an example of the pulseIn being used

```
int pin = 7;
unsigned long duration;

void setup() {
  Serial.begin(9600);
  pinMode(pin, INPUT);
}

void loop() {
  duration = pulseIn(pin, HIGH);
  Serial.println(duration);
}
```

As we will see in the next section, pulseIn() is used by a Particle or Dust sensor to count particulates in the air.

13.2 Seeed Dust Sensor

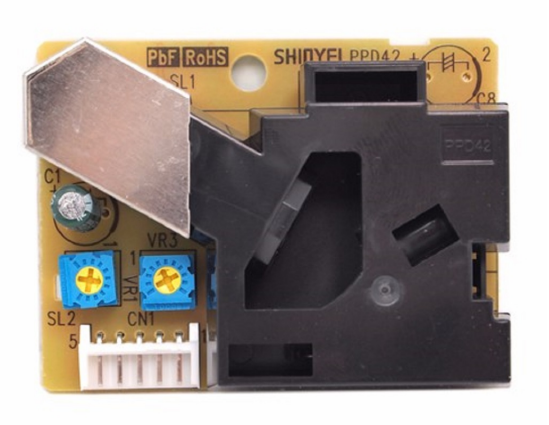


Figure 13.1: Seeed Dust Sensor

The dust sensor uses an optical sensing method to detect dust. A photosensor and an infrared light-emitting diode which is known as an IR LED are optically arranged in the dust sensor module. The photo-sensor (PT) detects the reflected IR LED rays which are bounced off of the dust particles in the

air. In the case of the Seeed sensor, it sends a low pulse everytime a dust particle is detected.

When you look at the Seeed Dust Sensor example code, you will notice three parameters are used to provide information on the dust concentration:

- "Lowpulseoccupancy" represents the Low Pulse Occupancy Time(LPO Time) detected in given 30s. Its unit is microseconds.
- "Ratio" reflects on which level LPO Time takes up the whole sample time.
- "Concentration" is a figure that has a physical meaning. It is calculated from the characteristic graph below by using the LPO,

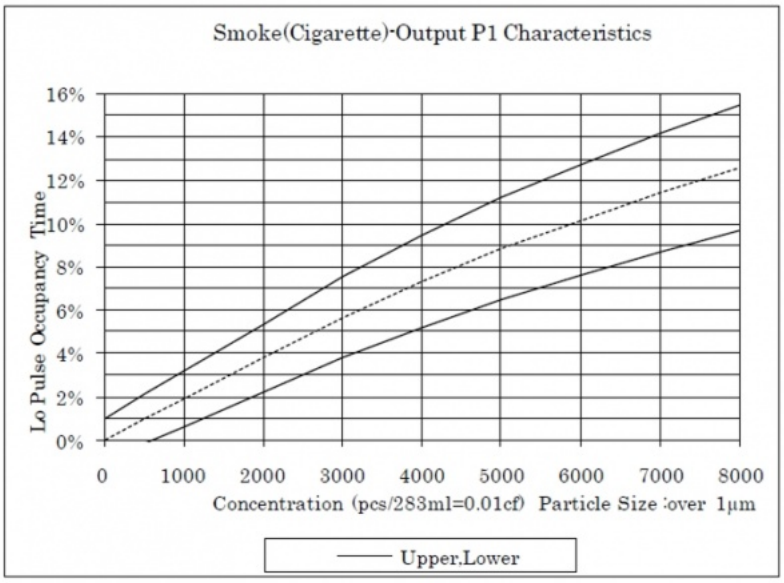


Figure 13.2: Dust Sensor Characterization

13.3 Seeed Air Quality Sensor v1.3

Figure 13.3: Air Quality Sensor

The Seeed Air Quality Sensor is an electrochemical sensor that is responsive to a wide scope of harmful gases, as carbon monoxide, alcohol, acetone, thinner, formaldehyde and so on. Due to the measuring mechanism, this sensor can't output specific data to describe target gases' concentrations quantitatively. But it's still competent enough to be used in applications that require only qualitative results, like auto refresher sprayers and auto air cycling systems.

The basic components of an electrochemical sensor are a working (or sensing) electrode, a counter electrode, and usually a reference electrode. These electrodes are enclosed in the sensor housing in contact with a liquid electrolyte. The working electrode is on the inner face of a Teflon membrane that is porous to gas, but impermeable to the electrolyte.

The gas diffuses into the sensor and through the membrane to the working electrode. When the gas reaches the working electrode, an electrochemical reaction occurs; either an oxidation or reduction depending on the type of gas. For example, carbon monoxide may be oxidized to carbon dioxide, or oxygen may be reduced to water. An oxidation reaction results in the flow of electrons from the working electrode to the counter electrode through

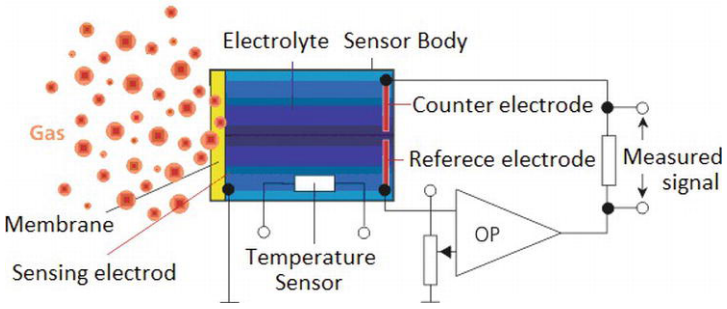


Figure 13.4: ElectroChemcial Sensor

the external circuit; and conversely a reduction reaction results in flow of electrons from the counter electrode to the working electrode. This flow of electrons constitutes an electric current, which is proportional to the gas concentration. The electronics in the instrument detects and amplifies the current and scales the output according to the calibration. The instrument then displays the gas concentration in, for example, parts per million (PPM) for toxic gas sensors and percent volume for oxygen sensors.

13.4 Assignment Particle 3 - Part IV

Please continue to use /Documents/<Your Name>/particle-03 as the Parent folder for the below assignments. Be sure to git push at the end of each day. The below exercise requires you to find information on the listed websites to complete the assignments. Work with your peers, before contacting the instructors, if you have issues.

1. Create a new project called PulseTest in your particle-03 repository. Add a button with a pull-up resistor to your Particle Argon. (Specifically use a pull-up resistor as to remind yourself how this is implemented. If you don't recall, look in instructor_master/L3_Buttons for the Fritzing diagram. Implement simple code that uses pulseIn() to print out the number of microseconds that your button is pushed. It should print nothing if pulseIn() returns a 0 due to timeout.

2. Create a new project called AirQuality in your particle-03-repository. Using the information found in `http://wiki.seeedstudio.com/Grove-Dust_Sensor/` implement the Grove Dust Sensor on your board. Test that it is working about taking baseline measurements and then lightly sprinkling a small amount of flour or powder near the sensor. Please note, the sensor needs to be in a vertical configuration (with the wiring pointing down) to function properly.

3. Using the information found in `http://wiki.seeedstudio.com/Grove-Air_Quality_Sensor_v1.3/`, add the Air Quality Sensor v1.3 to your AirQuality code. In addition to the qualitative air quality assessment, print out the detected concentration of gases in the air. After taking a baseline reading, test your sensor by placing a cotton ball of isopropyl alcohol or nail polish remover near your sensor. A few things of note:

    - The library you need can be installed via the Commmand Palette - Install Libraries.
    - Rather than use the Arduino sample code on the website, there is an example that gets downloaded with the library. It is easier to follow.
    - In addition to sensor.slope() which is in the example code, you might want to use sensor.getValue(). You can look in the library's C++ code to understand the methods that go with the class AirQualitySensor.

4. Finally, integrate both of these sensors into your Soil Moisture code and send the data to the ThingSpeak cloud.

Chapter 14

NCD.io ControlEverything Sensors

The NCD IoT Interface provides users with a means of changing or upgrading the IoT communications technology as new technologies emerge. The NCD IoT Interface is directly compatible with Particle Photon for WiFi communications, Electron for Cellular Communications, Bluz for Bluetooth. Optionally, adapters may be installed to provide a direct interface to Arduino Nano, Micro, USB, PyCom WyPy, Onion Omega, Raspberry Pi, and much more. The NCD IoT interface allows you to re-use your hardware so it never becomes obsolete! Based on I2C communications, the NCD IoT Interface uses only 2 GPIO lines of your microcontroller, freeing the rest of your CPU for other tasks.

14.1 NodeLynk IoT Device Expansion

This is a IoT Device which accepts a common processor and provides onboard sensing or control capabilities. This IoT device may be expanded to include additional hardware functionality using nodeLynk Expansion Devices. The NCD IoT socket found on IoT devices is capable of directly handling NCD ESP8266 and ESP32 series processors as well as Particle Photon

and Particle Electron. We also manufacture many adapters for the NCD IoT socket for Arduino Nano, Micro, and Feather microprocessor modules. Between the socket is a nodeLynk connector, which is used for I2C Expansion.

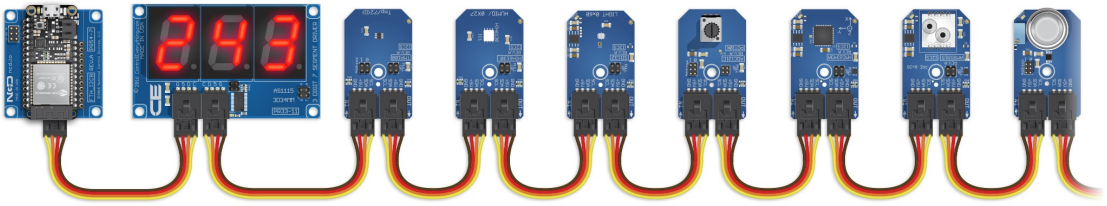


Figure 14.1: ControlEverything I2C Sensors

Figure 14.2: Feather Battery I2C Shield

14.2 Sensors

14.2.1 Gas Monitoring

The MQ-9 Gas sensor makes it easy to monitor carbon monoxide and combustible gas concentration levels using our I2C Mini Module form factor. The

MQ9 is connected to an ADC121C 12-Bit Analog to Digital converter, which is capable expanding to 9 gas sensors per I2C port using just two address jumpers (making full use of the floating address system).

Figure 14.3: MQ-9 Carbon Monoxide Combustible Gas Sensor

The MQ-9 is capable of sensing carbon monoxide air concentration levels between 10 and 1,000ppm and combustible gas air concentration levels between 100 and 10,000ppm. The ideal sensing condition for the MQ9 is $20°C \pm 2°C$ at $65\% \pm 5\%$ humidity. An internal preheater inside the sensor helps achieve the ideal sensing conditions, but the datasheet recommends over 48 hours for preheating to achieve optimal accuracy. Because of the internal preheater, this sensor requires more current than most of our I2C Mini Modules. We measured 139mA for this I2C Mini Module so we strongly recommend planning a power strategy for your I2C master device to deliver no-less than 150mA per sensor! This sensor comes pre-calibrated to the datasheet's recommended values; however, final calibration may be required for accurate measurements, as we do not stock the equipment for full-scale calibration of these sensors.

More details can be found at:
`https://store.ncd.io/product/mq-9-carbon-monoxide-combustible-gas-sensor-adc121c-`

Figure 14.4: AMS5812-0008-D Low Pressure Sensor

14.2.2 Pressure Sensor

AMS5812 pressure sensors are a series of high-precision OEM sensors with an analog 0.5–4.5V voltage output and a digital I2C-interface. They are calibrated and compensated for across a wide temperature range of -25 to $+85°C$. AMS5812 comes as a dual in-line package (DIP) for assembly on printed circuit boards (PCBs) and is fully operational without the need for any additional components. The electrical connection is made via the DIP solder pins; pressure is connected via two vertical metal tubes. AMS5812 combines micromachined, high quality piezo-resistive measuring cells with a modern, signal conditioning mixed-signal ASIC on a ceramic substrate. This enables high precision measurements and excellent drift and long-term stability. The sensors in the AMS5812 series are available for various applications and pressure ranges: differential (relative) devices in pressure ranges from 0–0.075psi up to 0–100psi, absolute pressure variants for 0–15PSI, 0–30psi and a barometric type. Bidirectional differential devices are available from $-0.075/+0.075psi$ up to $-15/+15psi$. Typical applications of AMS5812 include Barometric pressure measurement, Vacuum monitoring, Gas flow, Medical instrumentation, Heating, Ventilation and Air Conditioning (HVAC).

More details can be found at:
`https://store.ncd.io/product/ams5812-0008-d-low-pressure-sensor/`

14.2.3 Light Sensor

The TMG39931 features advanced gesture detection, proximity detection, digital ambient light sensor (ALS), Color Sensor (RGBC), and optical pattern generation/transmission.

The slim modular package incorporates an IR LED and factory calibrated LED driver.

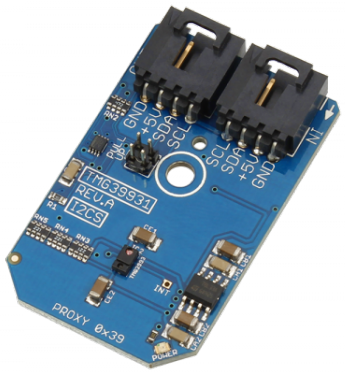


Figure 14.5: TMG39931 Light Sensor Gesture, Color, Proximity Sensor

Gesture detection utilizes four directional photodiodes to sense reflected IR energy (sourced by the integrated LED) to convert physical motion information (i.e. velocity, direction and distance) to digital information.

The gesture engine accommodates a wide range of mobile device gesturing requirements: simple North-South-East-West gestures or more complex gestures can be accurately sensed.

Power consumption and noise are minimized with adjustable IR LED timing.

More details can be found at:
`https://store.ncd.io/product/tmg39931-light-sensor-gesture-color-als-and-proximit`

14.2.4 Power Monitoring

This 1-Channel high-accuracy current monitoring controller makes it easy to get AC current measurements into your favorite IoT cloud platform. This current monitoring controller is available pre-tuned in your choice of 10, 20, 30, 50, 70, and 100-Amp ranges.

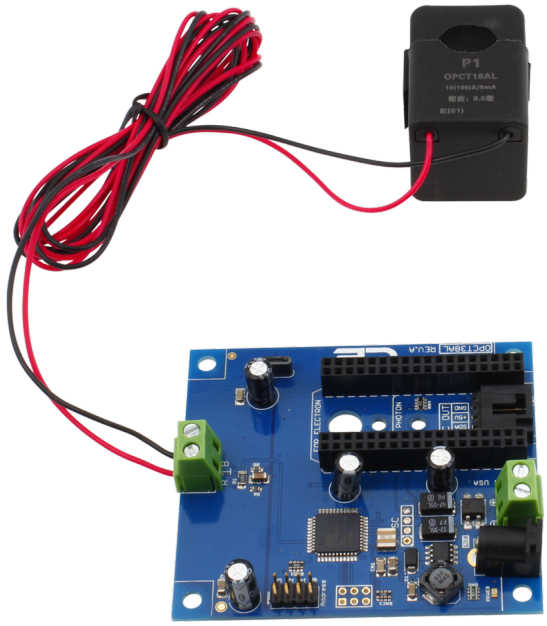


Figure 14.6: 1-Channel Off-Board AC Current Monitor

This controller handles all of the current calculations for energy monitoring applications. Use our Particle library to communicate energy monitoring data to the Particle cloud or adapt our library for your specific application

using the Arduino wire language.

This device provides an ideal energy monitoring solution for most industrial and commercial applications at a low cost with unmatched accuracy.

This controller includes one off-board current sensor and an I2C cable.

To use this device, simply run an AC power wire through the opening of the current sensor. This controller will read the magnetic field inducted onto the current sensor and provide you with a real-world current measurement value that is 98% accurate (prior to calibration).

This current monitoring controller includes our second generation firmware, which maintains best possible accuracy when measuring inductive loads, minimizing the need for calibration. This controller is pre-calibrated for resistive loads, but has demonstrated 98% accuracy when measuring most inductive load sources.

This controller includes split-core sensors with 1.5 Meter wires (around 5 feet). Split-core sensors can be opened and clamped around existing wire, greatly simplifying installation. For AC Current Measurement Only. This controller is tuned specifically for the included sensors, and may not be used with any other sensor. The sensor rating of this controller is 100 Amps; however, this controller is tuned according to your order. It is not possible to measure current outside the tuned range of this controller. Never exceed 110% of the rated current range of this controller (10, 20, 30, 50, 70, or 100-Amp depending on variation) or permanent damage to the controller may result. To achieve highest possible accuracy, each channel may require calibration for the type of load under measurement. As with all current sensors of this type, loads below 1 Amp (1% of the sensor rating) are known for significantly decreased accuracy (around 90% or less). To increase accuracy of low-current loads, consider doubling the readings by looping your AC load wire through the sensor a second time. This will effectively double all readings and provide greater low-end accuracy.

Excessively noisy loads may greatly affect accuracy, but our testing has shown 98% or better accuracy with most loads tested. This controller dedicates 1.3 seconds for measurement of each channel and cycles through all available channels continuously. We recommend querying this controller every 10 sec-

onds for best results.

More details can be found at:
https://store.ncd.io/product/1-channel-off-board-98-accuracy-100-amp-ac-current-m

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.

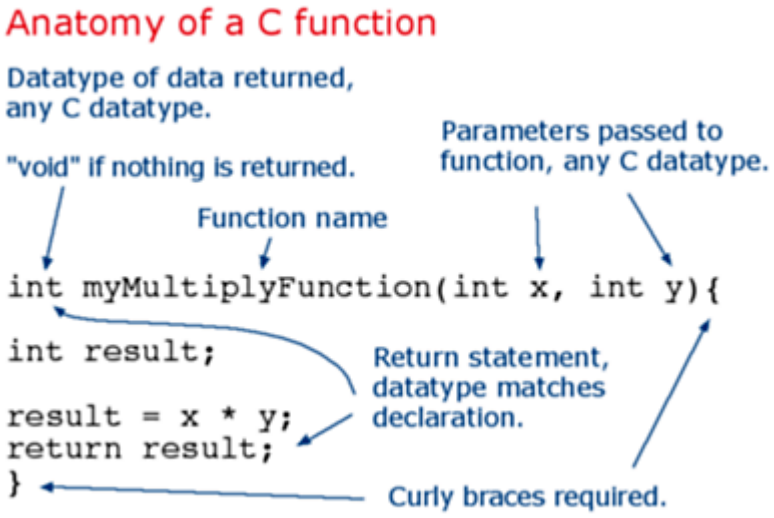


Figure A.1: Anatomy of a C function

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.

```
1  void setup(){
2    Serial.begin(9600);
3  }
4
5  void loop() {
6    int i = 2;
7    int j = 3;
8    int k;
9
10   k = myMultiplyFunction(i, j); // k now contains 6
11   Serial.println(k);
12   delay(500);
13 }
14
15 int myMultiplyFunction(int x, int y){
16   int result;
17   result = x * y;
18   return result;
19 }
```

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

Useful Code Examples

B.1 Fix

This code cements the bit-change that keeps the Particle Argon out of Listening Mode.

```
/*
 * Project fix
 * Description: fix the argon going into listening mode
 * Author: Brian Rashap
 * Date: 09-APR-2020
 */
#include "Particle.h"
#include "dct.h"
SYSTEM_MODE(SEMI_AUTOMATIC);
void setup() {
    const uint8_t val = 0x01;
    dct_write_app_data(&val, DCT_SETUP_DONE_OFFSET, 1);
    WiFi.on();
    WiFi.connect();
    Particle.connect();
}
```

B.2 I2C Scan

```
/*
 * Project I2C_Scan
 * Description: Scan I2C bus and return found addresses
 * Author: Brian Rashap
 * Date: 13-Jan-2020
 */

#include "Particle.h"

unsigned long delayTime;

void setup()
{
  Wire.begin();
  Serial.begin(9600);
  while(!Serial);    // time to get serial running
  Serial.println("\nI2C Scanner");

      unsigned status;

    // default settings
    // (you can also pass in a Wire library object like &
   Wire2)
}
```

B.3 Multiple Time Loops

```
/*
 * Project TimeLoops
 * Description: How to have loops run a different times
 * Author: Brian Rashap
 * Date: 16-APR-2020
 */
int lastSecond;
int lastMinute;
int lastHour;
int currentTime;

void setup() {
  lastSecond =  -1000;  // set to negative one second
  lastMinute = -60000;  // set to negative one minutes
```

```
  lastHour =  -360000;  // set to negative one hour
}

void loop() {
  //run constantly
  currentTime = millis();

  //run once per second
  if((currentTime-lastSecond)>1000) {
    Serial.print(".");
    lastSecond = millis();
  }

  //run once per minute
  if((currentTime-lastMinute)>60000) {
    Serial.println();
    Serial.println("Minute");
    lastMinute = millis();
  }

  //run once per hour
  if((currentTime-lastHour)>360000) {
    Serial.println();
    Serial.println("Hour");
    lastHour = millis();
  }
}
```

Appendix C

Particle Argon CLI Commands

C.1 Entering DFU-Mode

If you wish to program your device with a custom firmware via USB, you'll need to use this mode. This mode triggers the on-board bootloader that accepts firmware binary files via dfu-util.

To enter DFU-Mode:

- Hold down BOTH buttons
- Release only the RESET (right) button, while holding down the MODE button.
- Wait for the LED to start flashing yellow (it will flash magenta first)
- Release the MODE (left) button
- The device now is in the DFU mode and will continue to flash yellow.

C.2 Particle Not Connecting to Cloud

If you Particle Argon doesn't enter the connected mode (breathing cyan), then it is not connecting to the cloud. If this happens it often requires a firmware reset.

1. Place the Argon in DFU-Mode as outlined in Section C.1.
2. From Terminal/PowerShell type: particle update. This will update the devices firmware.
3. Once it has restarted, flashing green or blue, enter DFU-Mode again.
4. Execute the command: particle flash `--`usb tinker
5. Once it has restarted, it should return to breathing cyan.
6. If, instead, it returned to blinking blue, then you need to reset your wifi credentials using: particle serial wifi.

More information can be found at `https://docs.particle.io/tutorials/device-os/led/argon/`

Appendix D

Number Systems used by computers

D.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.

D.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).

D.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 D.1: Decimal to Binary Conversion

D.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.

D.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 D.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.

D.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.

D.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 D.3: Convert Decimal to Hexidecimal

D.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 D.4: Hex Identifiers

D.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 E

GITHUB

E.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
```

E.2 Getting the latest updates

To get the latest updates from the repository:

```
1  git pull
2
```

E.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 F

Lineage of Programming Languages

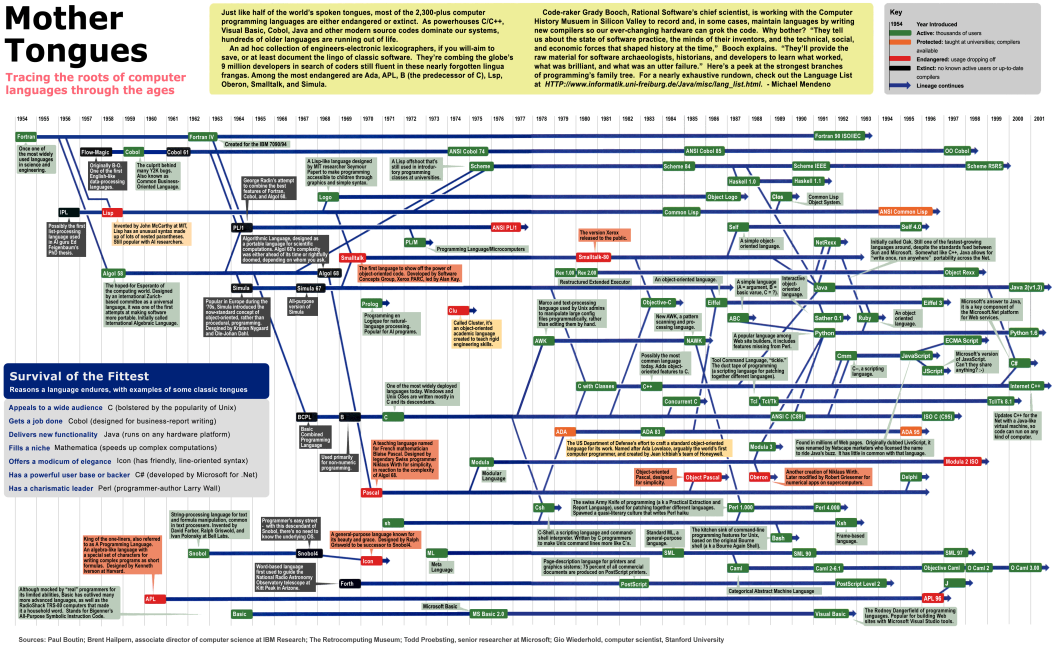
F.1 FORTRAN

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 F.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]

F.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.

F.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.

F.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.

F.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.

F.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.

F.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]

F.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.

F.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]