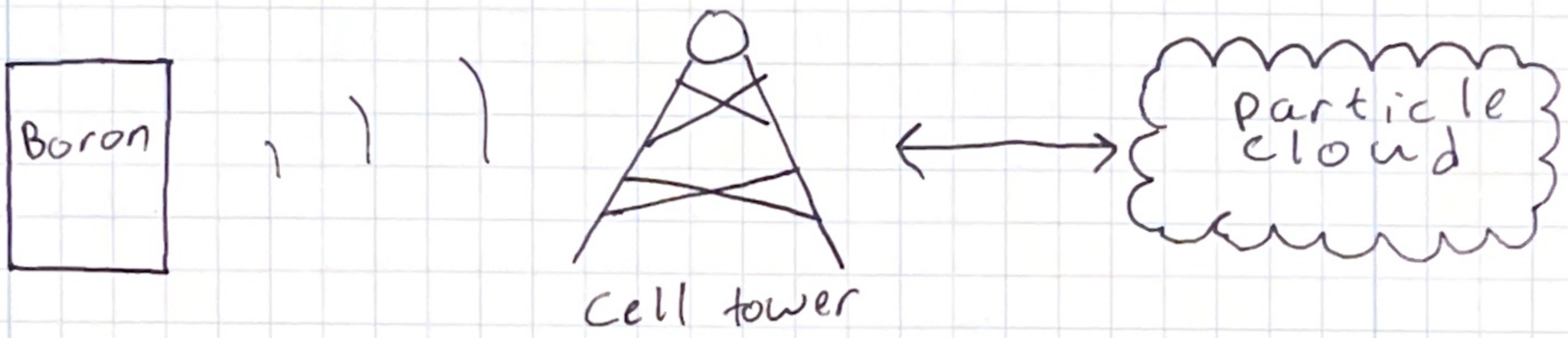
Setup

Particle as a company sells a web management interface for microcontrollers that have a cellular chip. This cellular chip allows the microcontroller to communicate over our cell towers.

We are using the particle board called "boron"



The particle cloud allows us to manage all of our devices. It even allows us to upload code entirley "OTA" (over the air).

Particle also has several other capabilities. Today, we learned about "Particle Functions" and "Particle Variables". These allow us to interact with our program over the web console.

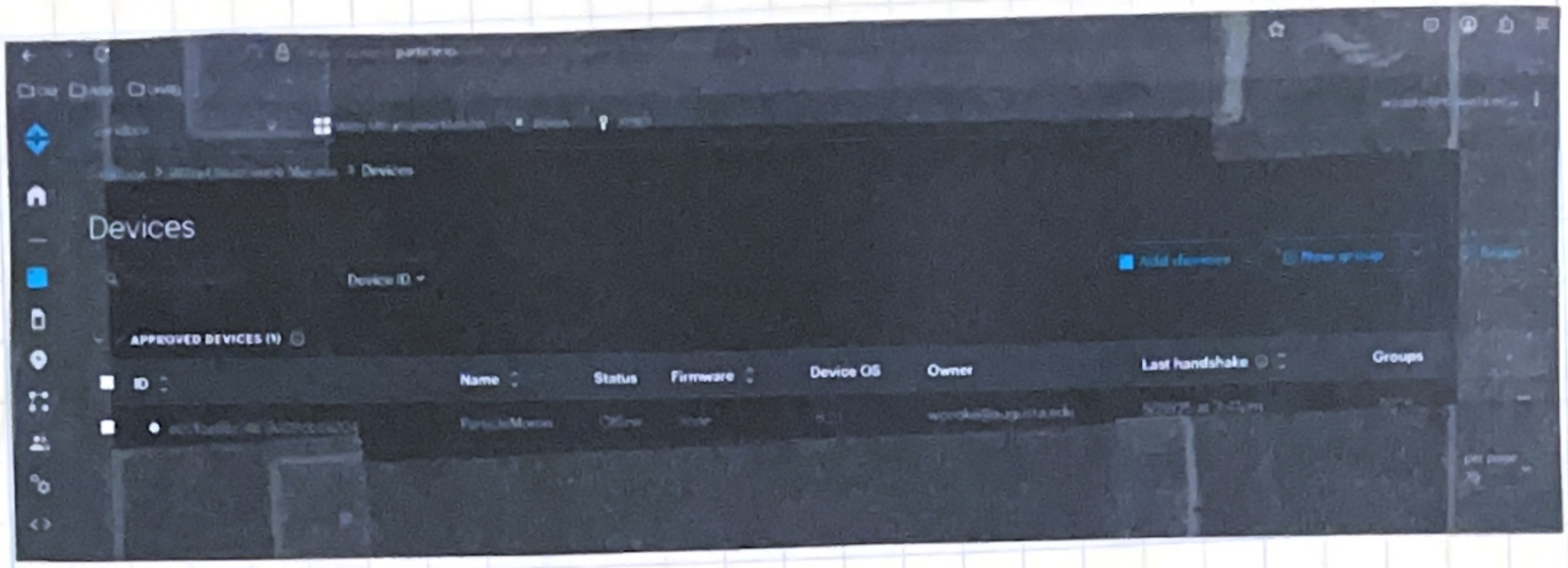
To set up a particle board, go to setup.particle.[illegible] :O

Plug the board in and follow the instructions.

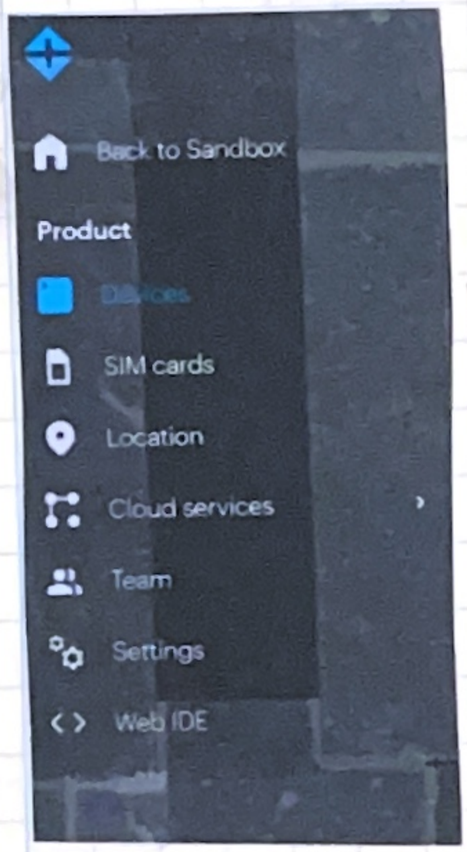
Next go to console.particle.io.
Here is like a main dashboard for everthing.
Go to "my product" and see if your device is there.

78 | Introduction to Particle Functions and Variables | 05/28/25 wesw Coda

Consde



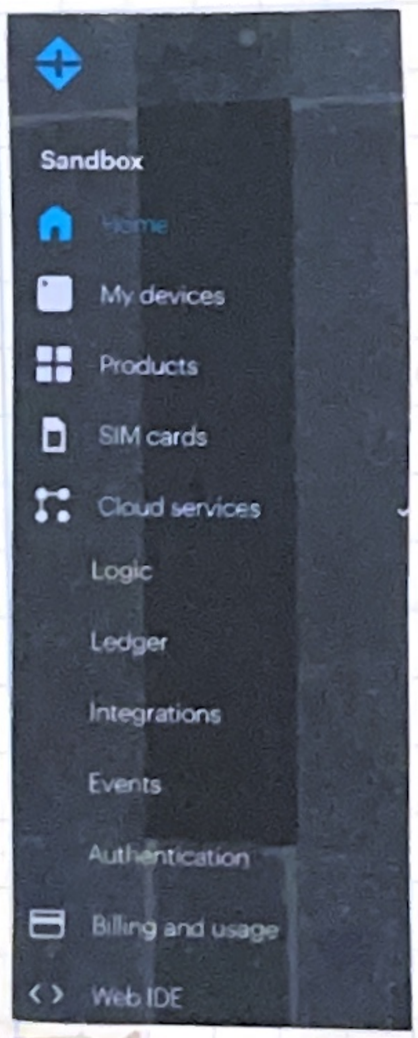
After clicking on my product, I see the device I setup earlier.



If we hover over the side menu, we get some options.

»Notice that we are in a product.
↰Devices associated with this product

→ Settings of this product.



What if we click "back to sandbox"

→ all devices on any product.
→ Products

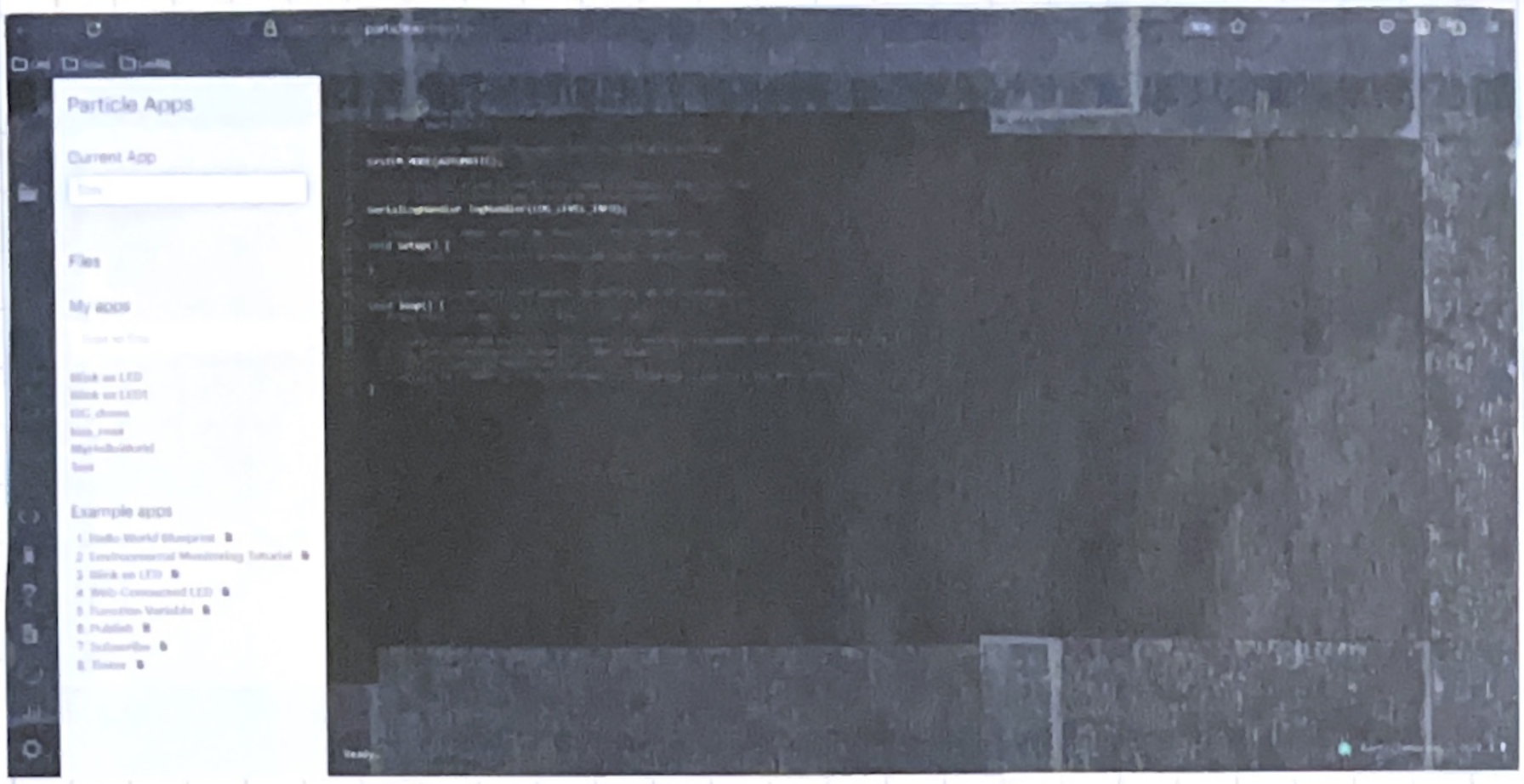
"Sand box" → Collection of "Products"

"Product" → Collection of devices

05/28/25 Wesley Cook

Introduction to Particle Functions and Variables

Web IDE



If we click the web IDE, we get this page. Here, we can write code and upload that code to the particle devices.

→ Flash device

→ Verify code

→ Save code

→ Code, list files

→ Libraries

→ help?

→ DOCS

→ Devices, Select device to flash

→ Consde

→ Account settings

Introduction to Particle Functions and Methods

Particle functions

Particle functions allow us to interact with our code from the web IDE.
They must return an integer and take in a string argument.

Syntax:

```
Particle.function("Name on Console", Function Handle);
```

"Name on Console": Can be anything like "start a rave" or "console light"

This is the name you will look for on Console.particle.io.

The function handle is the name of the function that it should be associated with in code. Similar to how we tell an interrupt what function to call.

~~On the tool~~ On consde.particle.io, search for your product and click on the device that has a particle function.

It will show up on the right menu;

```
1   SYSTEM_MODE(AUTOMATIC);
2   SerialLogHandler logHandler(LOG_LEVEL_INFO);
3
4   int pcbs;
5
6   const int LEDP = D0;
7
8   void setup() {
9     pinMode(LEDP, OUTPUT);
10    pinMode(D7, OUTPUT);
11    pinMode(D2, INPUT);
12
13    Particle.function("ConsoleLight", blink);
14  }
15
16  void loop() {}
17
18  int blink(String cmd){
19      if (cmd == "ON")
20      {
21          digitalWrite(D0, HIGH); // Turn on
22          return 1;
23      }
24      else if (cmd == "OFF")
25      {
26          digitalWrite(D0, LOW);  // Turn off
27          return 0;
28      }
29      else
30      {
31          return -1;
32      }
33  }
```

So if we type "ON" into the console, it will turn on D0. Notice that we didn't have to put this in the loop!

Our particle function can be called from anywhere we have access to console.particle.io, assuming the board is powered and it can reach a cell tower.

Introduction to Particle Functions and Variables

particle Variables

Particle Variables can be found in the same place as particle functions: console.particle.io and select the specific device.

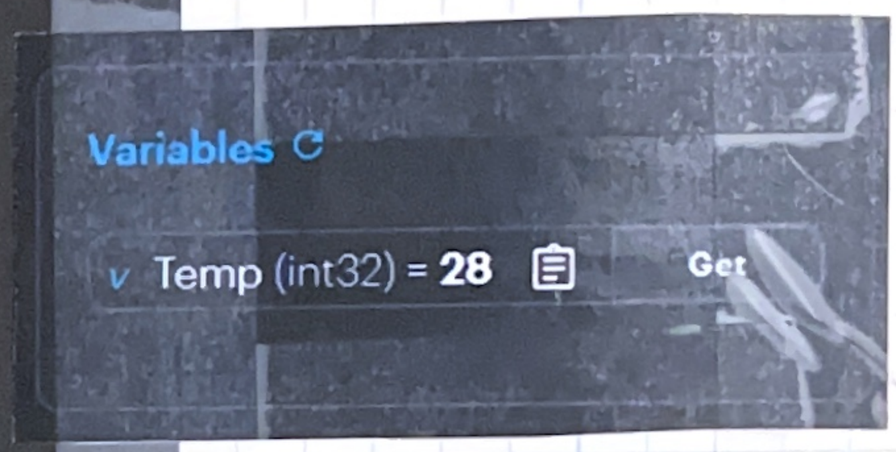
Particle Variables allow us to see the value of some variable in our program from the console. They can not be a float data type.

Syntax:

Particle.variable("Name on console", variable in code);

← using TC74

```ino
// Include Particle Device OS APIs
#include "Particle.h"
#include "Wire.h"

// Let Device OS manage the connection to the Particle Cloud
SYSTEM_MODE(AUTOMATIC);
SerialLogHandler logHandler(LOG_LEVEL_INFO);

const int led    = D9;
const int button = D2;
int myAdd;
int tempC;

// setup() runs once, when the device is first turned on
void setup() {
    Wire.begin();
    pinMode(led, OUTPUT);
    pinMode(button, INPUT);
    while(!Particle.connected());

    Serial.begin(9600);
    // delay(5000);
    for (int addr = 0; addr <= 127; addr++)
    {
        Wire.beginTransmission(addr);
        if (Wire.endTransmission() == 0) // Sensor sent back ACK
        {
            myAdd = addr;
            Serial.print("0x"); Serial.println(addr, HEX);
        }
    }

    Particle.variable("Temp", tempC);
}

// loop() runs over and over again, as quickly as it can execute.
void loop() {
    Wire.beginTransmission(myAdd);S
    Wire.write(0x00);
    Wire.endTransmission();
    delay(100);

    Wire.requestFrom(myAdd, 1);
    tempC = Wire.read();

    Serial.print("Temp: "); Serial.println(tempC);
}
```



See page 71-72 for old code that this replaces.

In the previous code, there is a delay of ~1s for reading each sensor. We would request the data, wait one second, and then read the data for each sensor. But the I2C Bus is free to use during the delay time...

Idea: ~~Read~~ Request data from all sensors back to back. Wait ~1 Second. Read all sensors response back to back.

```
18 void requestAtlas(int sensorAddress)
19 {
20   Wire.beginTransmission(sensorAddress);
21   Wire.write(82);                  // (R)ead
22   Wire.endTransmission();
23 } // end of request()
24
25 float readResponseAtlas(int sensorAddress)
26 {
27   Wire.requestFrom(sensorAddress, 20, 1);
28   code = Wire.read();
29
30   // 1 is success
31   // 2 syntax error
32   // 254 not ready
33   // 255 no data to send
34
35   // Read the data from the sensor
36   for (int ii = 0; ii < 20; ii++)
37   {
38     charIn = Wire.read();
39     sensorData[ii] = charIn;
40   }
41   Wire.endTransmission();
42   // Parse the data
43   data = atof(sensorData);
44
45   // Reset Variables
46   charIn = 0;
47   for (int jj = 0; jj < 20; jj++)
48   {
49     sensorData[jj] = 0;
50   }
51   return data;
52 } //end of readResponseAtlas()
```

```
73 void loop()
74 {
75   requestAtlas(DOaddress);
76   requestAtlas(ORPaddress);
77   requestAtlas(pHaddress);
78   requestAtlas(Caddress);
79
80   delay(timeDelay);
81
82   float DO  = readResponseAtlas(DOaddress);
83   float ORP = readResponseAtlas(ORPaddress);
84   float pH  = readResponseAtlas(pHaddress);
85   float C   = readResponseAtlas(Caddress);
86
87   Serial.print(DO);  Serial.print(" ");
88   Serial.print(ORP); Serial.print(" ");
89   Serial.print(pH);  Serial.print(" ");
90   Serial.print(C);   Serial.println();
91 }
```

only one delay!

~1s

```
10:39:39.687 -> 6.93 821.50 7.16 0.00
10:39:40.600 -> 6.93 821.60 7.16 0.00
10:39:41.492 -> 6.93 821.50 7.16 0.00
10:39:42.425 -> 6.93 821.60 7.16 0.00
10:39:43.327 -> 6.93 821.80 7.16 0.00
```

This is very useful for the autonomous version. We can gather data more often.