Tutorial 03 – Temperature and Humidity

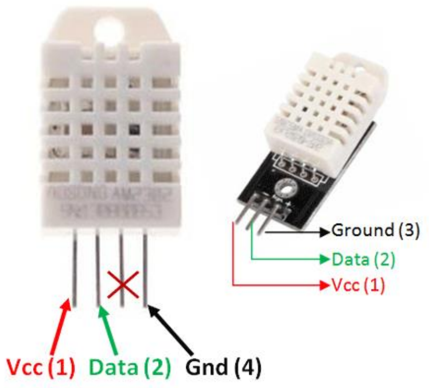
Make sure you complete **Setup** and
Test Your Adafruit Feather Huzzah32 Connection
document before starting this Tutorial 03.

Learning Objectives

1. Understand the use of DHT22 sensor and DHT.h library.
2. Understand the basics of programming, i.e., what is a source file, what is a header file, how to include a library (i.e., `#include`), how to install a library into a source file, how to create and call your own functions, the use of flag variables, static variables, different control structures (i.e., `while`, `do-while`), and arithmetic operators (i.e., `-`, `*`, `/`).
3. Implement and call your own functions to make a simple calculator.
4. Implement built-in functions from DHT.h library, i.e., DHT`()`, `begin()`, `readTemperature()`, `readHumidity()`, `computeHeatIndex()`, `convertCtoF()`, `convertFtoC()` in the code in order to operate DHT22 sensor via Adafruit Feather Huzzah32.
5. Understand maths functions in Arduino language i.e., `min()`, `max()`, `sq()`, and `sqrt()` and use them to process DHT22 sensor data.

Theory

DHT22 sensor is a commonly used temperature (-40 °C to 80 °C) and humidity sensor (0% to 100%). The sensor comes as a 4-pin package but only three pins are used. As shown in Figure 1, only pins 1, 2, and 4 are used for power, data, and Ground respectively.



(a)

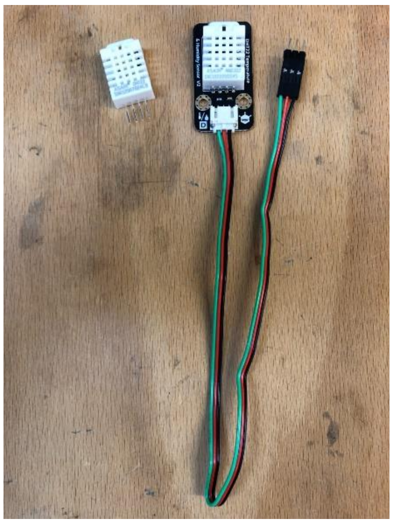

(b)

Figure 1: (a) DHT22 sensor (left) and DHT22 module (right) (Component 101, 2018), (b) DHT22 (left) and DHT22 module with jumper wires (right).

The operating voltage of DHT22 is 3.5 V to 5.5 V. We will be using only DHT22 sensor (Figure 1a left) and DHT22 module with jumper wires (Figure 1b right) for 5COM2004. The main difference between DHT22 sensor (Figure 1a left) and DHT22 module (Figure 1a right, and Figure 1b right) is that DHT22 module has an inbuilt capacitor (to store electrical energy - like a battery) and resistor (to reduce the current flow).

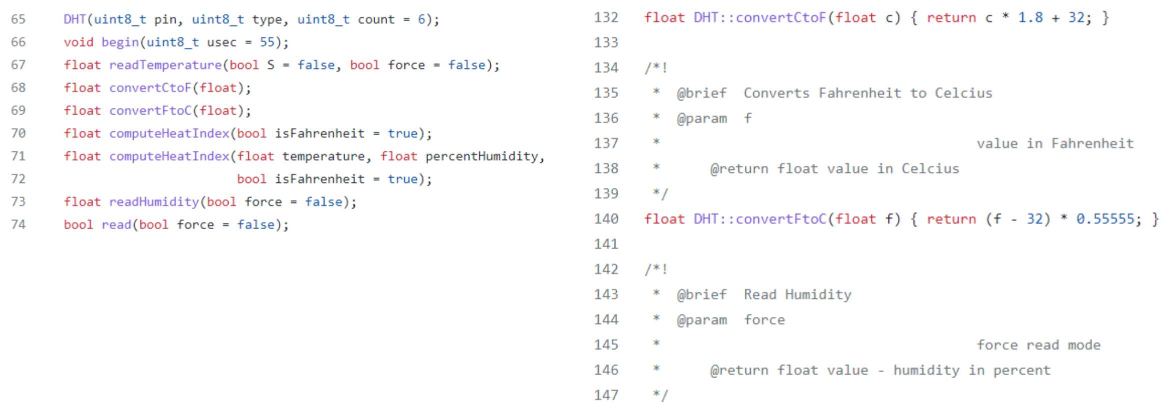
For Figure 1b (right), the DHT22 module (made by DF Robot) came with a female connector and jumper wires, whereas DHT22 and the DHT22 module in Figure 1a both have male connectors. The jumper wires connected to the female connector are colour coded, i.e., black (Ground), red (power) and green (data), and when DHT22 is faced upward (the PCB is flat on the table), Ground should be on the right-hand side.

Arduino libraries are files written in C or C++ which provide extra functionality to your program. Libraries make it easy for you to connect to hardware such as sensors, or display units. Each library consists of a source file (`.c` or `.cpp`) and a header file (`.h`). Although you can create your own libraries, we will focus only on the use of existing libraries within the scope of 5COM2004.

A source file or a source code file contains program instructions. In the Arduino language, a source file may include an entire program's source code (with `.ino` extension), or it may be one of many referenced libraries (with `.c` or `.cpp` extension) within a programming project.

A header file is a file with a `.h` extension, which contains function declarations and macro definitions to be shared between several source files. An example of a macro definition e.g., `#define` is discussed in Tutorial 01 (revisit Tutorial 01 if you have already forgotten how to use

`#define`). Figure 2 (left) illustrates examples of function declarations in DHT`.h` and Figure 2 (right) illustrates examples of function definitions from DHT`.cpp`. From Figure 2 (right), `convertCtoF()` and `convertFtoC()` are example function definitions from DHT`.cpp`. They contain program instructions i.e., how the convert degree Celsius to degree Fahrenheit and vice versa from a float input to the functions.



```cpp
DHT(uint8_t pin, uint8_t type, uint8_t count = 6);
void begin(uint8_t usec = 55);
float readTemperature(bool S = false, bool force = false);
float convertCtoF(float);
float convertFtoC(float);
float computeHeatIndex(bool isFahrenheit = true);
float computeHeatIndex(float temperature, float percentHumidity,
                       bool isFahrenheit = true);
float readHumidity(bool force = false);
bool read(bool force = false);
```

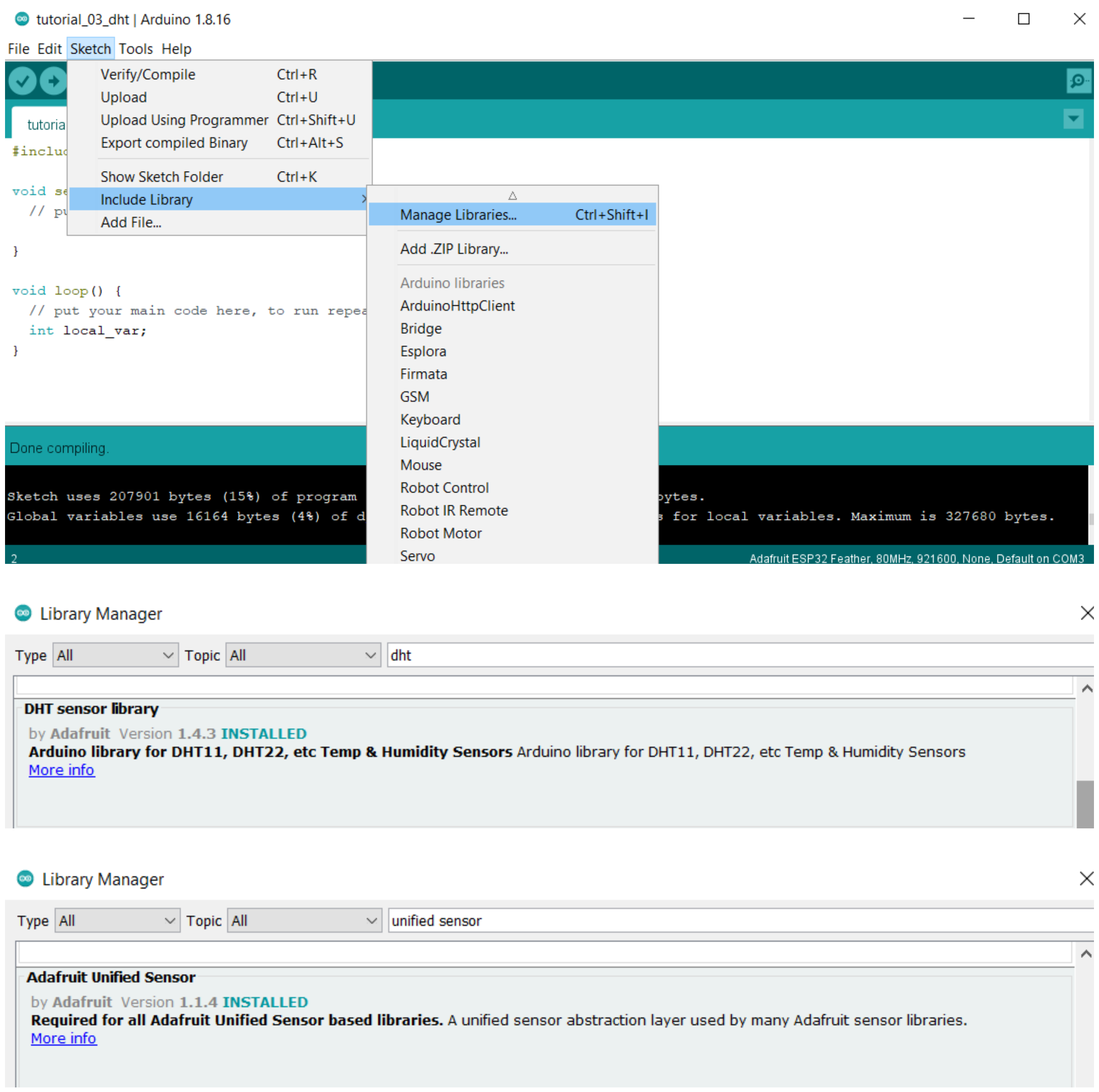
```cpp
float DHT::convertCtoF(float c) { return c * 1.8 + 32; }

/*!
 *  @brief  Converts Fahrenheit to Celcius
 *  @param  f
 *                                          value in Fahrenheit
 *      @return float value in Celcius
 */
float DHT::convertFtoC(float f) { return (f - 32) * 0.55555; }

/*!
 *  @brief  Read Humidity
 *  @param  force
 *                                          force read mode
 *      @return float value - humidity in percent
 */
```

Figure 2: Function declarations (left), and function definitions (right) (Adafruit, 2021).

You can include a library in your source file by going to Sketch, select Include Library and then select the relevant library from the list. You can also manually type in an `#include` statement at the top of your source code file (`.ino`) for each relevant header file (`.h`) of every required library. For example, insert `#include` <DHT`.h`> to include the DHT library to your source code. However, you may need to install the library before you can include it by going to Sketch, select Include Library, select Manage Libraries, enter the name of a required library, and then select Install. Once a library is installed, the word INSTALLED will be shown on the Library Manager following the version number of that library (as shown in Figure 3).

Read the following Arduino reference page on the macro `#include` which can be used to add a library to your source file.

1. `#include`: https://www.arduino.cc/reference/en/language/structure/further-syntax/include/ .


tutorial_03_dht | Arduino 1.8.16
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries... Ctrl+Shift+I
Add .ZIP Library...
Arduino libraries
ArduinoHttpClient
Bridge
Esplora
Firmata
GSM
Keyboard
LiquidCrystal
Mouse
Robot Control
Robot IR Remote
Robot Motor
Servo
tutoria
Done compiling.
Adafruit ESP32 Feather, 80MHz, 921600, None, Default on COM3

```
#incluc

void se
  // pu

}

void loop() {
  // put your main code here, to run repea
  int local_var;
}
```

```
Sketch uses 207901 bytes (15%) of program                          ytes.
Global variables use 16164 bytes (4%) of d                         s for local variables. Maximum is 327680 bytes.
```

DHT sensor library
by **Adafruit** Version **1.4.3** **INSTALLED**
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors
More info

Adafruit Unified Sensor
by **Adafruit** Version **1.1.4** **INSTALLED**
Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
More info

Figure 3: Installing DHT sensor library and Adafruit Unified Sensor library.

Two libraries, i.e., DHT sensor library and Adafruit Unified Sensor, will be required for this Tutorial 03. You may search for the names as shown in Figure 3. If they are not INSTALLED, then you will need to install them before carrying out the Practice part of this tutorial.

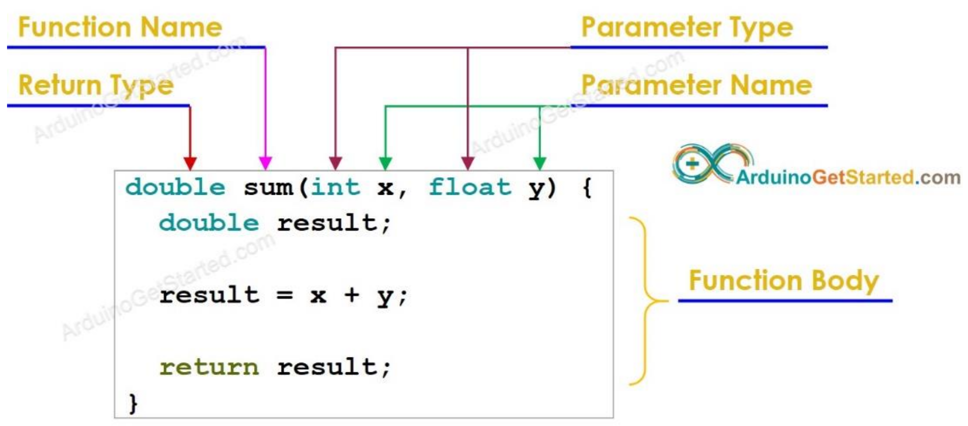
Unlike Arduino reference pages, libraries may be written by hardware manufacturers or other programmers, and they may only have short explanations on how the library can be used. Luckily, built-in function names are often self-explanatory, comments are often included in these libraries to explain the built-in functions, examples or tutorials are often available online, and the tasks required by 5COM2004 tutorials are generally simple enough to carry out without needing to understand every line of the code in the library (although it can be useful to understand them).

If you haven't done so already, check out the following built-in functions from DHT.cpp which you will be using in this Tutorial 03 https://github.com/adafruit/DHT-sensor-library/blob/master/DHT.cpp .

1. `DHT()`: to initialise the PIN no, and sensor type.
2. `begin()`: setup the pin.
3. `readTemperature()`
4. `readHumidity()`
5. `computeHeatIndex()`: to determine the apparent temperature or the human perceived temperature.
6. `convertCtoF()`
7. `convertFtoC()`

By including a library into your source file (`.ino`), you can refer to built-in functions from within the included libraries. Functions are chunks of code that can be reused rather than having to be rewritten multiple times. Figure 4 illustrates an example of a function that programmers may create themselves. You can call a function by simply calling its name and supplying relevant input parameters e.g., `sum(1, 2)` for the example function in Figure 4. Revisit Tutorial 01 and Lecture 03, if you have already forgotten what a function looks like, how to call it, and its purpose.

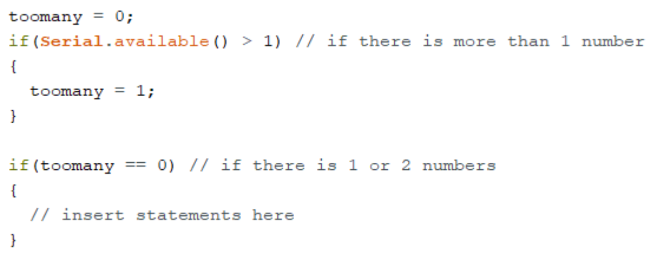
One difference which you may have noticed between built-in functions (Figure 2) and functions created by programmers such as yourself (Figure 4) is the lack of function declarations and the header file itself for your-created functions. Function declarations are required when you want to define functions in one source file and call it from another file (as with the case of your source file calling a library). It should also be noted that separating the header file from the source file is considered a good programming practice in C/C++ language, especially in large, complicated projects. All tutorials in 5COM2004 will be relatively small and simple, i.e., there is no need for multiple source files (`.ino`). Therefore, it won't be necessary to create your own library or header files, and functions that you will need such as the ones in this tutorial can be inserted directly into your source files.



```
double sum(int x, float y) {
  double result;

  result = x + y;

  return result;
}
```

Figure 4: A typical structure of a function (ArduinoGetStarted, n.d.).

A flag variable is a variable that has been defined to have one value until a certain condition is true, in which case the value of that variable is then changed to a different value. This is commonly used to control the flow of your program as it progresses. Figure 5 illustrates an example which may have been similar to the one you may have used in Tutorial 02's Challenge.



```
toomany = 0;
if(Serial.available() > 1) // if there is more than 1 number
{
  toomany = 1;
}

if(toomany == 0) // if there is 1 or 2 numbers
{
  // insert statements here
}
```

Figure 5: Flag variable.

From Figure 5, `toomany` is initialised to zero and only changed to 1 when there is more than one byte of data to read from the serial monitor. Flag variables such as `toomany` can minimise the number of times a program has to execute functions e.g., to process a data stream for a result and that result is being used several times within that function or program. By calling `Serial.available()` once and assigning its output value to a variable (in this case `toomany`), accessing the value from `toomany` is faster than calling the `Serial.available()` for its output value again and again. A programmer may then add statements in the if-clause (i.e., `if(toomany == 0)`) to instruct the program to carry out certain tasks when there is only one byte of data or lower from the serial monitor, as well as defining another set of tasks when there is more than one byte of data from the serial monitor (i.e., if an else-clause is added to the program). In this example, `toomany` is declared as `int`, but other data types e.g., `bool` can also be used for flag variables.

Read all the following Arduino reference pages on the additional data types that are discussed in Lecture 03:

1. `size_t`: https://www.arduino.cc/reference/en/language/variables/data-types/size_t/ .
2. `bool`: https://www.arduino.cc/reference/en/language/variables/data-types/bool/ .
3. `uint8_t`: this link refers to the actual uint8_t library https://pubs.opengroup.org/onlinepubs/009696899/basedefs/stdint.h.html . However, Arduino reference page has a simpler explanation of unsigned int data type, which are similar, but you may also find the following web page useful and simpler to understand https://www.arduino.cc/reference/en/language/variables/data-types/unsignedint/ .

As discussed in Lecture 03, curly braces i.e., { }, generally define the scope and the lifetime of variables. For example, a variable declared inside a function has its scope and its lifetime within that function i.e., the variable can only be accessed within that function, and that variable will be created when the function is called and destroyed when the program exits that function. The keyword `static` extends the lifetime of a variable to the entire duration of a program. A common usage example of static variables is counters e.g., the number of times a person walks passed a motion sensor. Your program may increase (or decrease – depending on the program requirements) the counter every time that function is called.

Read the following Arduino reference page on the variable score qualifiers `static` which can be used to preserve the value of a variable between function calls.

1. `static`: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/static/ .

We have looked at `if`, `else` and `for` in Tutorial 02, additional control structures which you may have considered using in Tutorial 02 Challenge are `while` and `do-while`. Unlike `for`, `while` and `do-while` loops are commonly used when the programmer does not have a clear number of repetitions required to execute a section of the code by the program. The difference between `while` and `do-while` is the fact that `do-while` guarantees at least one round of execution of the content of the loop itself (as the stopping condition is evaluated at the end of the loop). However, `while` does not guarantee this as the stopping condition is evaluated at the beginning of the loop. Read the following Arduino reference page on control structures:

1. `while`: https://www.arduino.cc/reference/en/language/structure/control-structure/while/ .
2. `do-while`: https://www.arduino.cc/reference/en/language/structure/control-structure/dowhile/ .

There are six arithmetic operators in the Arduino language. Four arithmetic operators are commonly used and have the same meanings as the arithmetic operators in algebra i.e., addition (+), subtraction (−), multiplication (*), and division (/). We have already used assignment operator (=), addition (+) and remainder operator (%) in Tutorial 01 and Tutorial 02. Revisit the previous two tutorials if you have already forgotten these three arithmetic operators.

Read all the following Arduino reference pages on the three additional arithmetic operators which you will be using in this tutorial.

1. Subtraction (−): https://www.arduino.cc/reference/en/language/structure/arithmetic-operators/subtraction/ .
2. Multiplication (*): https://www.arduino.cc/reference/en/language/structure/arithmetic-operators/multiplication/ .
3. Division (/): https://www.arduino.cc/reference/en/language/structure/arithmetic-operators/division/ .

There are 8 maths functions in the Arduino language. Read all the following Arduino reference pages on the 2 reference pages which you will be using in this tutorial. While sq() calculates the square of a number, sqrt() calculates the square root of a number.

1. `sq()`: https://www.arduino.cc/reference/en/language/functions/math/sq/ .
2. `sqrt()`: https://www.arduino.cc/reference/en/language/functions/math/sqrt/ .

The Arduino language also has min() and max() which identify the smaller and larger number of the two input values respectively. We won't be using these functions specifically in this tutorial, because of the limited number of input values. However, you should read all the following Arduino reference pages anyway to understand the basic of these functions.

1. `min()`: https://www.arduino.cc/reference/en/language/functions/math/min/ .
2. `max()`: https://www.arduino.cc/reference/en/language/functions/math/max/ .

Reference

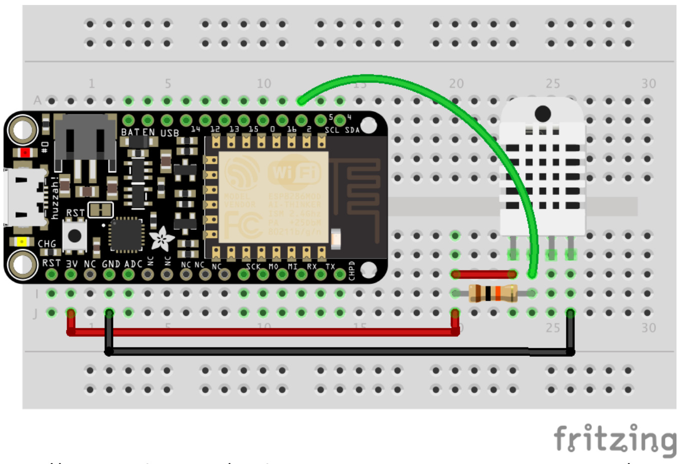
Adafruit. (2021). DHT-sensor-library. [Website] Available at: https://github.com/adafruit/DHT-sensor-library

ArduinoGetStarted. (n.d.). Function. [Website] Available at: https://arduinogetstarted.com/reference/arduino-function

Components 101. (2018). DHT22 – Temperature and Humidity Sensor. [Website] Available at: https://components101.com/sensors/dht22-pinout-specs-datasheet

Practice 3.1

1. Connect DHT22 and resistor according to the following schema. Note where the power (3V), the Ground (GND) and the data (Pin 14) are and connect them correctly.



Source: https://learn.adafruit.com/adafruit-io-basics-temperature-and-humidity/arduino-wiring

2. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
#include <DHT.h>

#define DHTPIN 14     // what pin we're connected to
#define DHTTYPE DHT22   // DHT 22  (AM2302)
DHT dht(DHTPIN, DHTTYPE); //// Initialize DHT sensor

float humidity;  //Stores humidity value
float temperature_c; // Stores Celcius temperature value
float temperature_f; // Store Fahrenheit temperature value

float hic; // Stores heat index for Celcius temperature value
float hif; // Stores heat index for Fahrenheit value

void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600);
  dht.begin();

}

void loop() {
  // put your main code here, to run repeatedly:

  //Print temp and humidity values to serial monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %, Temperature C: ");
  Serial.print(temperature_c);
  Serial.print(" Celsius");
  Serial.print(", Heat Index C: ");
  Serial.print(hic);
  Serial.print(", Temperature F: ");
  Serial.print(temperature_f);
  Serial.print(" Fahrenheit");
  Serial.print(", Heat Index F: ");
  Serial.println(hif);
  delay(10000);
}
```

Note: Use practice3_1.json for WOKWI.

3. **Verify** the code. You will then be asked to save the code (or as commonly known as a sketch). Enter a **meaningful name** for your file, e.g., tutorial_03_dht. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

4. If you don't have a serial monitor open already, go to **Tools** and then select **Serial Monitor**.

5. **Upload** the code.

TO DO

1. Modify tutorial_03_dht.ino to read the temperature (both in Celsius and Fahrenheit) and the humidity using the functions readTemperature() and readHumidity() respectively as well as calculating the heat index using the function computeHeatIndex() for the temperature in Celsius and Fahrenheit.
2. **Verify** your modification and **Upload** the code.

Practice 3.2

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

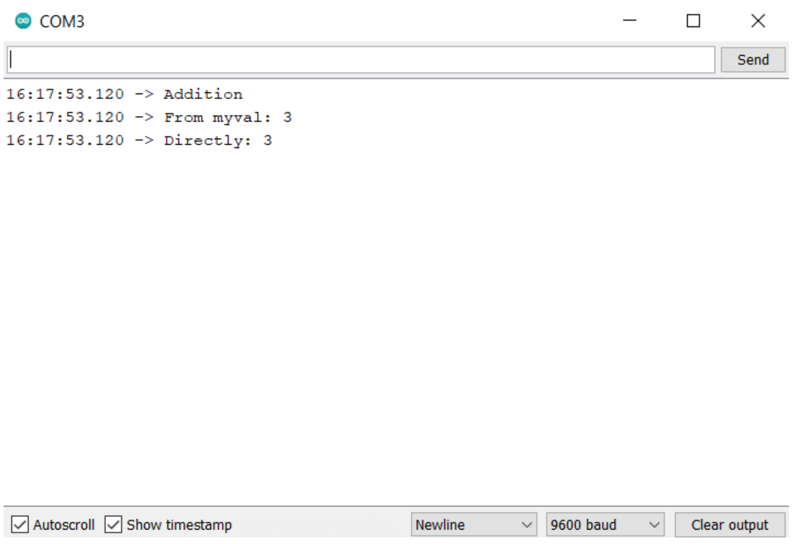
   ```
   int addition(int x, int y)
   {
     return (x + y);
   }

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

   void loop() {
     // put your main code here, to run repeatedly:
     int myval = addition(1, 2);

     Serial.println("Addition");
     Serial.print("From myval: ");
     Serial.println(myval);
     Serial.print("Directly: ");
     Serial.println(addition(1, 2));

     delay(10000);
   }
   ```

   Note: Use default.json for WOKWI.

2. **Verify** the code. You will then be asked to save the code (or as commonly known as a sketch). Enter a **meaningful name** for your file, e.g., tutorial_03_calculator. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

3. If you don't have a serial monitor open already, go to **Tools** and then select **Serial Monitor**.

4. **Upload** the code. You should now see the output of `addition()` as shown below.

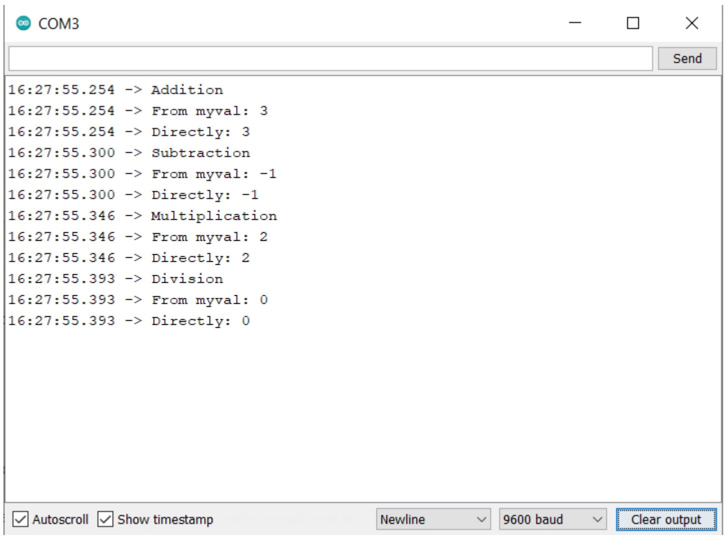


```
16:17:53.120 -> Addition
16:17:53.120 -> From myval: 3
16:17:53.120 -> Directly: 3
```

5. Make sure you understand the difference between the second and the third line of output above (i.e., where the value 3 is obtained from).

TO DO

1. Modify tutorial_03_calculator.ino to use 3 other arithmetic operators i.e., subtraction (−), multiplication (*) and division (/) which accept 2 integer inputs. You may choose whatever values you wish to use to test these functions. The below output example using 1 and 2 as the input for all the functions. It is a good programming practice to choose a meaningful name for variable and function name.



```
16:27:55.254 -> Addition
16:27:55.254 -> From myval: 3
16:27:55.254 -> Directly: 3
16:27:55.300 -> Subtraction
16:27:55.300 -> From myval: -1
16:27:55.300 -> Directly: -1
16:27:55.346 -> Multiplication
16:27:55.346 -> From myval: 2
16:27:55.346 -> Directly: 2
16:27:55.393 -> Division
16:27:55.393 -> From myval: 0
16:27:55.393 -> Directly: 0
```

2. **Verify** your modification and **Upload** the code to see if it still works correctly.

Practice 3.3

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

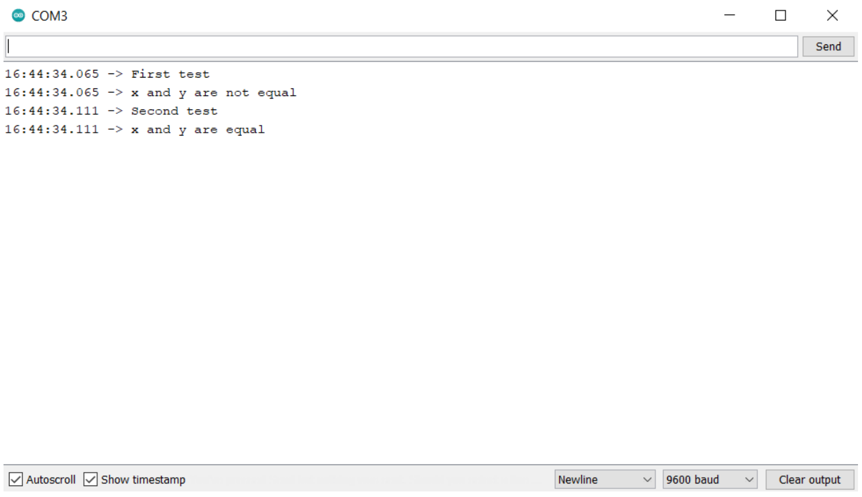
   ```
   void setup() {
     // put your setup code here, to run once:
     Serial.begin(9600);
   }

   void loop() {
     // put your main code here, to run repeatedly:

     int x = 9;
     int y = 10;

     Serial.println("First test");
     if(x == y)
     {
       Serial.println("x and y are equal");
     }
     else
     {
       Serial.println("x and y are not equal");
     }

     Serial.println("Second test");
     if(x = y)
     {
       Serial.println("x and y are equal");
     }
     else
     {
       Serial.println("x and y are not equal");
     }

     delay(10000);
   }
   ```

   Note: Use default.json for WOKWI.

2. **Verify** the code. You will then be asked to save the code (or as commonly known as a sketch). Enter a **meaningful name** for your file, e.g., tutorial_03_equal_signs. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

3. If you don't have a serial monitor open already, go to **Tools** and then select **Serial Monitor**.

4. **Upload** the code. You should now see the output of `if` and `else` clauses as shown below.

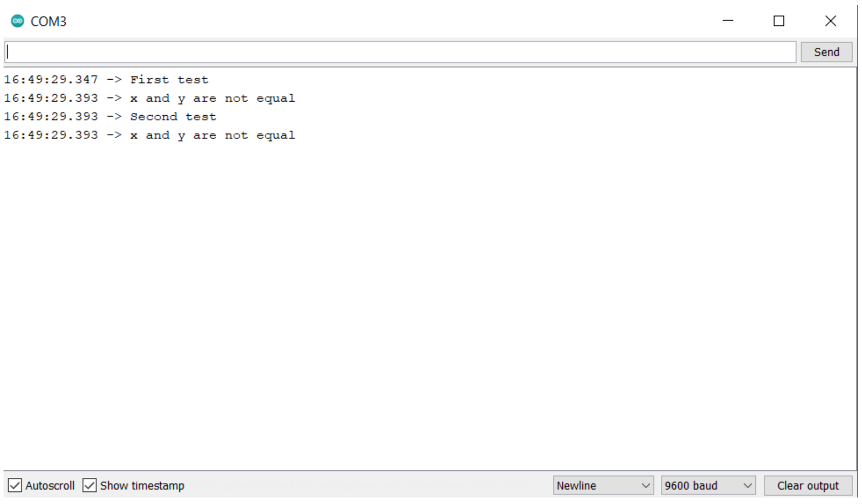


```
16:44:34.065 -> First test
16:44:34.065 -> x and y are not equal
16:44:34.111 -> Second test
16:44:34.111 -> x and y are equal
```

5. Make sure you understand the difference between the second and the fourth line of output above, i.e., `x and y are not equal` and `x and y are equal`.

<u>TO DO</u>

1. Read the following Arduino reference page on Boolean constants i.e., `true` and `false` https://www.arduino.cc/reference/en/language/variables/constants/constants/ .
2. Modify tutorial_03_equal_signs.ino by changing the value of `y` to 0 (instead of 10). It may be a good idea to save the modification in a different name, e.g., tutorial_03_equal_signs_todo as this will allow you to run the two files to compare the results from this modification and the original.
3. **Verify** your modification and **Upload** the code to compare the difference. You should now see the output of `if` and `else` clauses as shown below.



```
16:49:29.347 -> First test
16:49:29.393 -> x and y are not equal
16:49:29.393 -> Second test
16:49:29.393 -> x and y are not equal
```

4. Do make sure that you understand why the code does what it does, i.e., the program now produce the same output for both the second and the fourth output line.

<u>Practice 3.4</u>

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
#define ntemp 10

float temp[ntemp];
static int measurement_counter = 0;

float getMin()
{
  float result = temp[0];
  for(int i = 1; i < ntemp; i++)
  {
    if(temp[i] < result)
    {
      result = temp[i];
    }
  }
  return result;
}

float getMax()
{
  float result = temp[0];
  for(int i = 1; i < ntemp; i++)
  {
    if(temp[i] > result)
    {
      result = temp[i];
    }
  }
  return result;
}

float getAvg()
{
  float result = 0.0;
  for(int i = 0; i < ntemp; i++)
  {
    result = result + temp[i];
  }
  return (result / ntemp);
}

float getStdv()
```

```
{
  float avg = getAvg();
  float deviation = 0.0;
  float sumsqr = 0.0;
  for(int i = 0; i < ntemp; i++)
  {
    deviation = temp[i] - avg;
    sumsqr = sumsqr + sq(deviation);
  }
  float variance = sumsqr / ntemp;
  return sqrt(variance);
}

void initialisation()
{
  for(int i = 0; i < ntemp; i++)
  {
    temp[i] = 0.0;
  }
}

void addCounter()
{
  measurement_counter = measurement_counter + 1;

  if(measurement_counter >= ntemp)
  {
    printProcessedData();
    resetCounter();
  }
}

void resetCounter()
{
  measurement_counter = 0;
}


void addReading(float t)
{
  temp[measurement_counter] = t;
  addCounter();
}

void printProcessedData()
{
  Serial.print("Min: ");
  Serial.print(getMin());
```

```
 Serial.print(", Max: ");
 Serial.print(getMax());
 Serial.print(", Avg: ");
 Serial.print(getAvg());
 Serial.print(", Stdev: ");
 Serial.println(getStdv());
}

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

void loop() {
 // put your main code here, to run repeatedly:

 delay(1000);
}
```

Note: Use practice3_1.json for WOKWI.

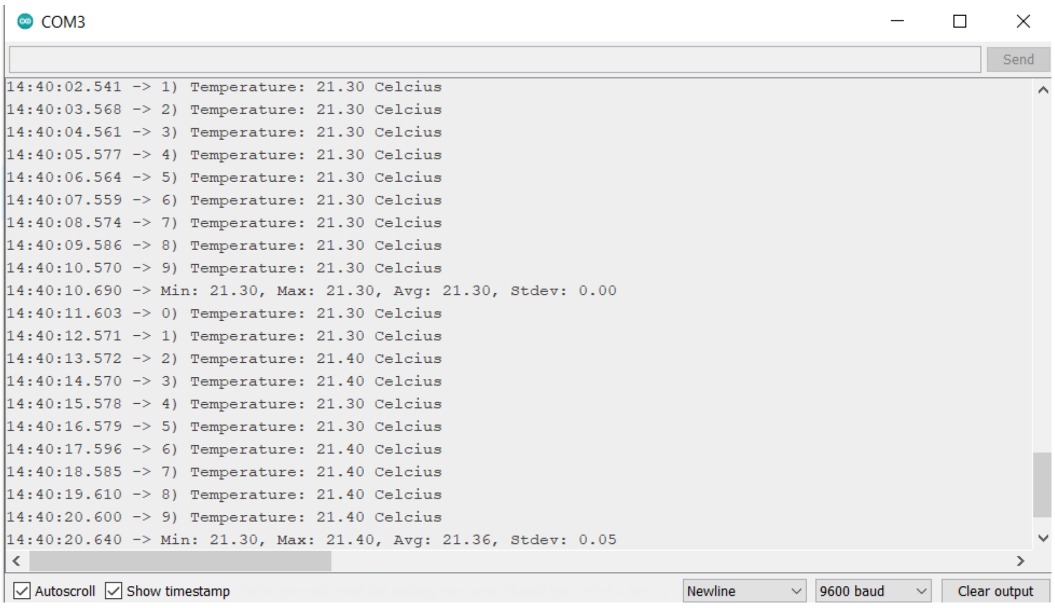
2. **Verify** the code. You will then be asked to save the code (or as commonly known as a sketch). Enter a **meaningful name** for your file, e.g., tutorial_03_maths. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

3. If you don't have a serial monitor open already, go to **Tools** and then select **Serial Monitor**.

4. **Upload** the code.

TO DO

1. Modify tutorial_03_maths.ino by combining sections from tutorial_03_dht.ino (Practice 3.1) to now record a Celsius temperature reading every second and generate some statistics after every 10 readings e.g. minimum, maximum, average, and standard deviation. Your code should produce an output similar to what is shown below.

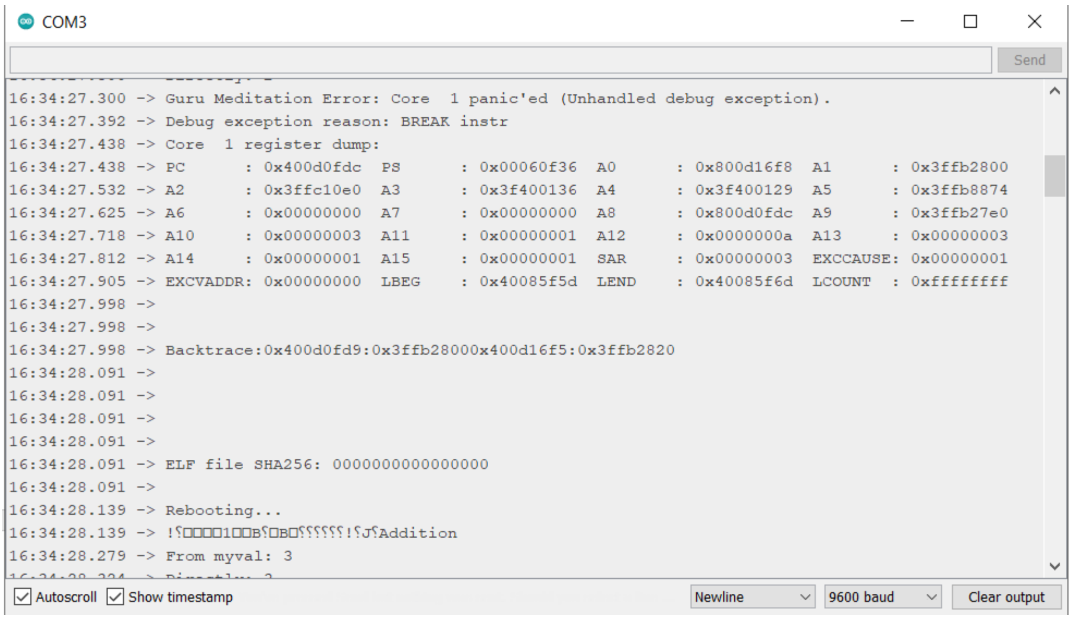


```
14:40:02.541 -> 1) Temperature: 21.30 Celcius
14:40:03.568 -> 2) Temperature: 21.30 Celcius
14:40:04.561 -> 3) Temperature: 21.30 Celcius
14:40:05.577 -> 4) Temperature: 21.30 Celcius
14:40:06.564 -> 5) Temperature: 21.30 Celcius
14:40:07.559 -> 6) Temperature: 21.30 Celcius
14:40:08.574 -> 7) Temperature: 21.30 Celcius
14:40:09.586 -> 8) Temperature: 21.30 Celcius
14:40:10.570 -> 9) Temperature: 21.30 Celcius
14:40:10.690 -> Min: 21.30, Max: 21.30, Avg: 21.30, Stdev: 0.00
14:40:11.603 -> 0) Temperature: 21.30 Celcius
14:40:12.571 -> 1) Temperature: 21.30 Celcius
14:40:13.572 -> 2) Temperature: 21.40 Celcius
14:40:14.570 -> 3) Temperature: 21.40 Celcius
14:40:15.578 -> 4) Temperature: 21.30 Celcius
14:40:16.579 -> 5) Temperature: 21.30 Celcius
14:40:17.596 -> 6) Temperature: 21.40 Celcius
14:40:18.585 -> 7) Temperature: 21.40 Celcius
14:40:19.610 -> 8) Temperature: 21.40 Celcius
14:40:20.600 -> 9) Temperature: 21.40 Celcius
14:40:20.640 -> Min: 21.30, Max: 21.40, Avg: 21.36, Stdev: 0.05
```

2. **Verify** your modification and **Upload** the code.
3. You may use the readings to manually recheck the calculation to ensure that your code is performing as expected.

Challenge 3.1

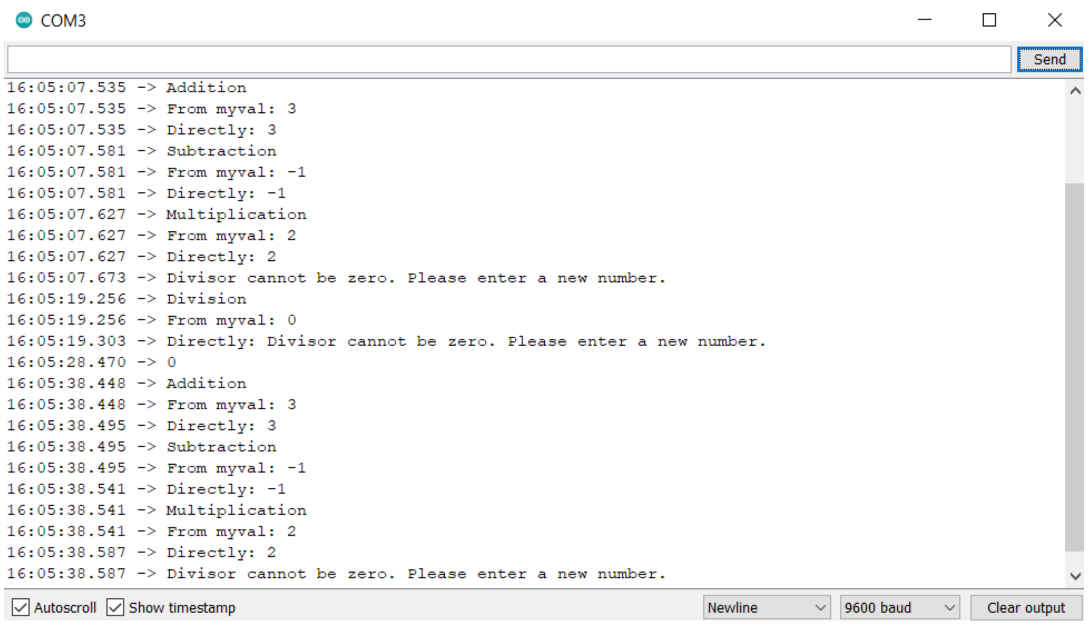
1. Revisit your tutorial_03_calculator.ino and test what happen when your divisor is zero. For example, if you have previously used 2/3, i.e., 2 is the numerator and 3 is your divisor (or denominator), change 3 to zero, **Verify** your modification, and **Upload** the code.
2. If your code works perfectly, then you have now completed this Challenge. However, if your code produces a similar output as shown below, you will then need to complete this Challenge.



```
16:34:27.300 -> Guru Meditation Error: Core  1 panic'ed (Unhandled debug exception).
16:34:27.392 -> Debug exception reason: BREAK instr
16:34:27.438 -> Core  1 register dump:
16:34:27.438 -> PC      : 0x400d0fdc  PS      : 0x00060f36  A0      : 0x800d16f8  A1      : 0x3ffb2800
16:34:27.532 -> A2      : 0x3ffc10e0  A3      : 0x3f400136  A4      : 0x3f400129  A5      : 0x3ffb8874
16:34:27.625 -> A6      : 0x00000000  A7      : 0x00000000  A8      : 0x800d0fdc  A9      : 0x3ffb27e0
16:34:27.718 -> A10     : 0x00000003  A11     : 0x00000001  A12     : 0x0000000a  A13     : 0x00000003
16:34:27.812 -> A14     : 0x00000001  A15     : 0x00000001  SAR     : 0x00000003  EXCCAUSE: 0x00000001
16:34:27.905 -> EXCVADDR: 0x00000000  LBEG    : 0x40085f5d  LEND    : 0x40085f6d  LCOUNT  : 0xffffffff
16:34:27.998 ->
16:34:27.998 ->
16:34:27.998 -> Backtrace:0x400d0fd9:0x3ffb28000x400d16f5:0x3ffb2820
16:34:28.091 ->
16:34:28.091 ->
16:34:28.091 ->
16:34:28.091 ->
16:34:28.091 -> ELF file SHA256: 0000000000000000
16:34:28.091 ->
16:34:28.139 -> Rebooting...
16:34:28.139 -> !ʕ□□□□1□□Bʕ□B□ʕʕʕʕʕʕ!ʕJʕAddition
16:34:28.279 -> From myval: 3
```

3. Modify your code to generate a message “Divisor cannot be zero. Please enter a new number.” to the user to input a new divisor as shown.

   Note: You may use sections from tutorial_02_read.ino to get user input. Use default.json for WOKWI.



```
16:05:07.535 -> Addition
16:05:07.535 -> From myval: 3
16:05:07.535 -> Directly: 3
16:05:07.581 -> Subtraction
16:05:07.581 -> From myval: -1
16:05:07.581 -> Directly: -1
16:05:07.627 -> Multiplication
16:05:07.627 -> From myval: 2
16:05:07.627 -> Directly: 2
16:05:07.673 -> Divisor cannot be zero. Please enter a new number.
16:05:19.256 -> Division
16:05:19.256 -> From myval: 0
16:05:19.303 -> Directly: Divisor cannot be zero. Please enter a new number.
16:05:28.470 -> 0
16:05:38.448 -> Addition
16:05:38.448 -> From myval: 3
16:05:38.495 -> Directly: 3
16:05:38.495 -> Subtraction
16:05:38.495 -> From myval: -1
16:05:38.541 -> Directly: -1
16:05:38.541 -> Multiplication
16:05:38.541 -> From myval: 2
16:05:38.587 -> Directly: 2
16:05:38.587 -> Divisor cannot be zero. Please enter a new number.
```

4. **Verify** your modification. You will then be asked to save the code (or as commonly known as a sketch). Enter a **meaningful name** for your file, e.g., tutorial_03_calculator_challenge. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.
5. **Upload** the code to see if it now produces the required output.