Tutorial 02 – Input and Output

Make sure you complete **Setup** and
Test Your Adafruit Feather Huzzah32 Connection
document before starting this Tutorial 02.

Learning Objectives

1. Understand the use of breadboard in connecting simple sensors and actuator to the Adafruit Feather Huzzah32.
2. Understand the use of compile output to debug the program.
3. Understand the use of serial monitor in monitoring the output from sensors, debugging and communicating controls/settings to the Adafruit Feather Huzzah32.
4. Understanding the basics of programming including scope of variables, control structures (e.g., `if`, `else`, `for`), the return values from function calls, the arithmetic operator (%), comparison operators (e.g., >, >=, <, <=, ==), compound operators (i.e., ++), and the size of data.
5. Implement function calls related to serial monitor, i.e., `available()`, `begin()`, `read()`, `print()`, `println()`, and `parseInt()` , getting analogue input from the sensor i.e., `analogRead()`, and sending output i.e., `digitalWrite()` to the actuator on Adafruit Feather Huzzah32.

Theory

Breadboards are used for building circuits. Basically, a piece of plastic with a bunch of holes with internal wiring i.e., many metal strips that connect rows and columns together.

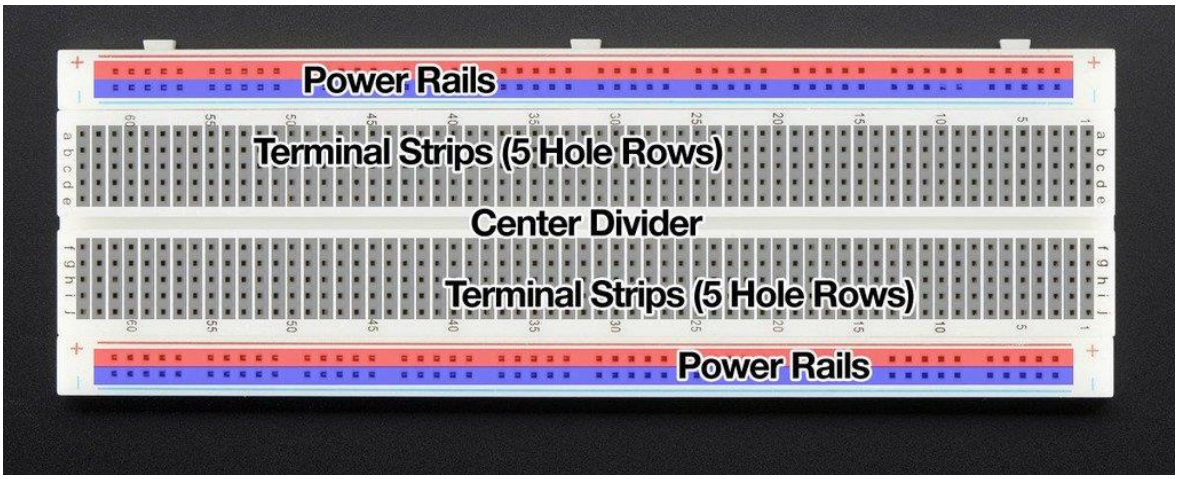


Figure 1: Breadboard and its metal strips (Adafruit, n.d.).

As shown in Figure 1, the number of teeth on the metal clips corresponds to the number of holes on the breadboard i.e., there are 5 teeth in each terminal strips and 50 teeth in each power rail. These teeth are used to grip electronic parts and any other electronic parts that are connected to the same metal clip are electronically connected together.

Note: DO NOT attempt to open the back of the School's breadboard to investigate the metal strips. Any student who does so will be asked to purchase a replacement for the School.

Components (e.g., sensors, actuators, boards) are usually placed on terminal strips. The power rails distribute power and ground connections along the entire board. There is no physical difference between positive and negative power rails, but it is a good practice to use the positive rails for power and negative rails for Ground.

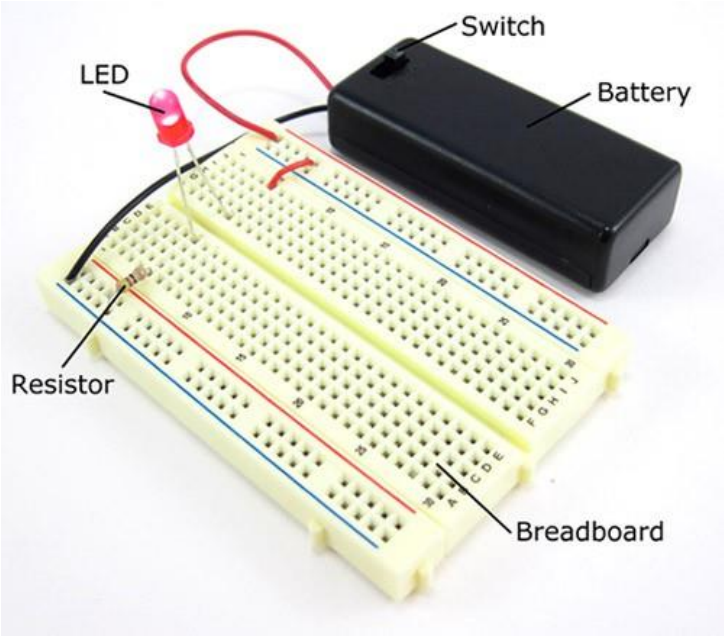


Figure 2: Connecting breadboard (Science Buddies, 2022).

Figure 2 illustrates how the right power rail on the breadboard is powered, and the left power rail is connected as Ground. The short red jumper wire (connecting the right power rail with J5) powers the entire 5th row of right terminal strip. The resistor (connecting the left power rail with A5) makes the entire 5th row of the left terminal strip connected to Ground. It should be noted that no other terminal strips are powered or connected to Ground in Figure 2.

Note: The example in Figure 2 powers the breadboard using a battery. However, all tutorials for 5COM2004 will use the power from Adafruit Feather Huzzah32 via USB connection. Revisit Test Your Adafruit Feather Huzzah32 Connection document again, if you do not understand what this USB connection is.

Read all the following Arduino reference pages for the functions being used to obtain input data from sensors and sending output to the actuators in this tutorial.

1. `analogRead()`: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

Functions may return a value. For example, `analogRead()` returns the analogue reading from the pin, i.e., a value between 0-4095 because Adafruit Feather Huzzah32 uses 12 bits ADC. The return value from a function can be used directly in an evaluation (Figure 6) or by assigning that return value to a variable (Figure 3).

Returns

The analog reading on the pin. Although it is limited to the resolution of the analog to digital converter (0-1023 for 10 bits or 0-4095 for 12 bits). Data type: `int`.

```
int analogPin = A3; // potentiometer wiper (middle terminal) connected to analog pin 3
                    // outside leads to ground and +5V
int val = 0;  // variable to store the value read

void setup() {
  Serial.begin(9600);           //  setup serial
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
  Serial.println(val);          // debug value
}
```

Figure 3: Return values from `analogRead()` and an example of assigning the return value from `analogRead()` to a variable (Arduino, 2022a).

A global variable `val` is declared to store the return value from `analogRead()` in Figure 3. The statement `int val = 0;` declares the variable `val` of type `int` and initialise it with a value of zero. `Val` is considered as a global variable here in Arduino language because it is declared outside of any functions.

A local variable is any variable that is declared within a pair of curly braces i.e., `{ }`, which can be within a function, or within a control structure such as a for-loop. The scope of a variable limits the access and modification to that variable, i.e., a global variable can be accessed by any function, whereas a local variable can only be accessed by code section within that scope. If you have forgotten about the use of curly braces, please revisit Tutorial 1.

```
int gPWMval;  // any function will see this variable

void setup() {
  // ...
}

void loop() {
  int i;    // "i" is only "visible" inside of "loop"
  float f;  // "f" is only "visible" inside of "loop"
  // ...

  for (int j = 0; j < 100; j++) {
    // variable j can only be accessed inside the for-loop brackets
  }
}
```

Figure 4: Variable scope (Arduino, 2022b).

Figure 4 illustrates an example of a global variable, i.e., `gPWMval`, and 3 examples of local variables i.e., `i`, `f`, and `j`. In this example, the program cannot access or modify the value of `i`, `f` and `j` from `setup()`. If you do not understand for-loop in Figure 4, Practice 2.1 uses this control structure and you can read its Arduino reference page from the link there.

Some functions, e.g., `pinMode()` do not return any value, and any attempt to obtain a return value from these functions would cause a compile error as shown in Figure 5. The two functions in Figure 4, i.e., `setup()` and `loop()` also do not return any value as indicated by the keyword `void` in front of the function name as shown on their function definition. Revisit Tutorial 1 if you have already forgotten what `pinMode()` and `void` are.

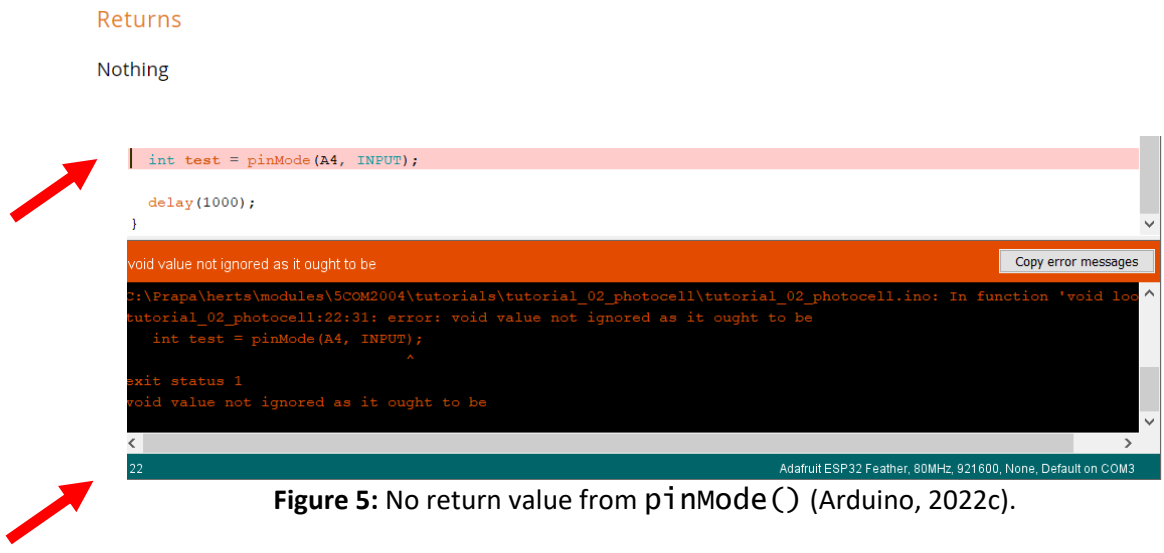


Figure 5: No return value from `pinMode()` (Arduino, 2022c).

The error message in the compile output window (Figure 5), explains the nature of the error as well as its possible location.

*"In function 'void loop()': tutorial_02_photocell:**22:31**: error: void value not ignored as it ought to be*

int test = pinMode(A4, INPUT);

^

exit status 1

void value not ignored as it ought to be"

The number 22:31 indicates the row number and the column number where the error is respectively. The number at the lower left of the Arduino IDE indicates the row number where the cursor is currently, i.e., row 22 in this example.

Control structures are used in branching the program based on given parameters. There are 10 different control structures in the Arduino language, similar to many other programming languages. Read all the Arduino reference pages on the control structures which will be used in this tutorial.

1. `if`: https://www.arduino.cc/reference/en/language/structure/control-structure/if/
2. `else`: https://www.arduino.cc/reference/en/language/structure/control-structure/else/

The serial monitor can be a useful tool in debugging, by seeing the actual output, and communicating directly to the Adafruit Feather Huzzah32. Read all the following Arduino reference pages for the serial monitor related functions being used in this tutorial.

1. `begin()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/
2. `available()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/available/
3. `read()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/read/
4. `print()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/print/
5. `println()`: https://www.arduino.cc/reference/en/language/functions/communication/serial/println/

Figure 6 illustrates a simple example which setups the program to accept user input from the serial monitor. The variable `incomingByte` here is a global variable with type `int`. The return value from `Serial.available()` is evaluated directly in Figure 6, as opposed to assigning the return value from a function to a variable, e.g., `analogRead()` to a variable `val` in Figure 3.

```
int incomingByte = 0;   // for incoming serial data

void setup() {
        Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps
}

void loop() {

        // send data only when you receive data:
        if (Serial.available() > 0) {
                // read the incoming byte:
                incomingByte = Serial.read();

                // say what you got:
                Serial.print("I received: ");
                Serial.println(incomingByte, DEC);
        }
}
```

Figure 6: A simple serial communication example.

Reference

Adafruit. (n.d.). Breadboards for Beginners. [Website] Available at: https://learn.adafruit.com/breadboards-for-beginners/introduction

Arduino. (2022a). analogRead(). [Website] Available at: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

Arduino. (2022b). Scope. [Website] Available at: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/scope/

Arduino. (2022c). pinMode(). [Website] Available at: https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/

Science Buddies. (2022). How to Use a Breadboard for Electronics and Circuits. [Website] Available at: https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard

Practice 2.1

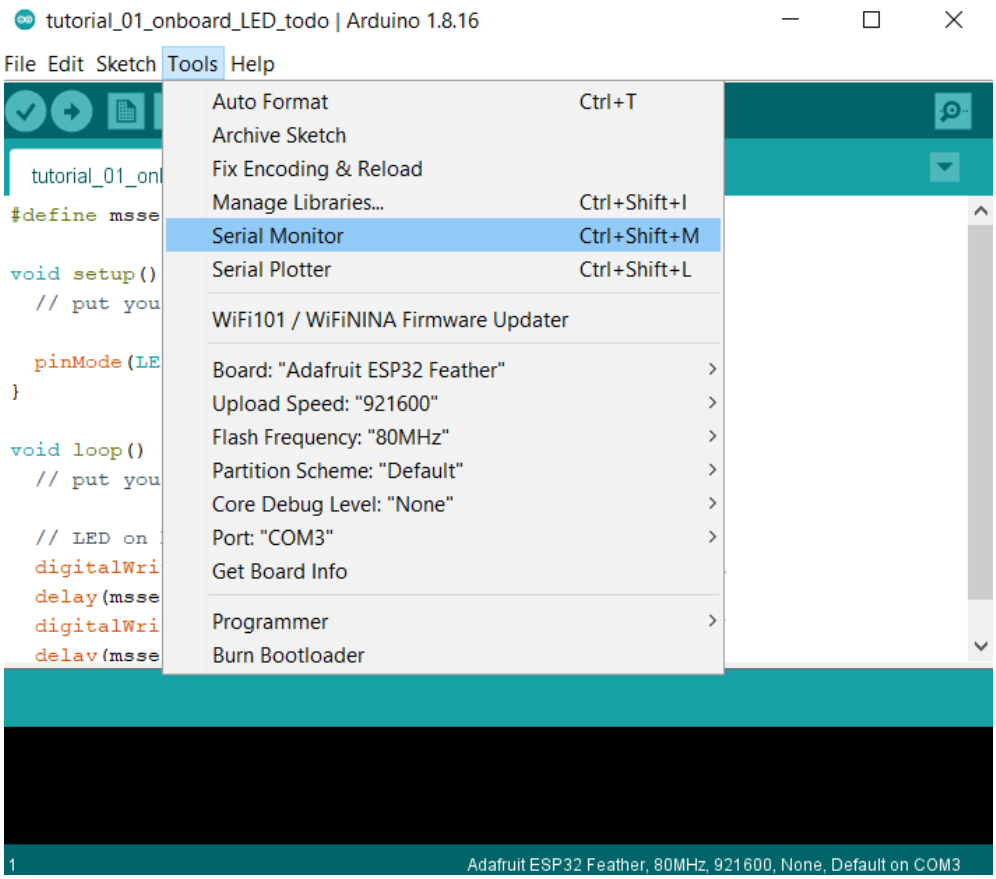
1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

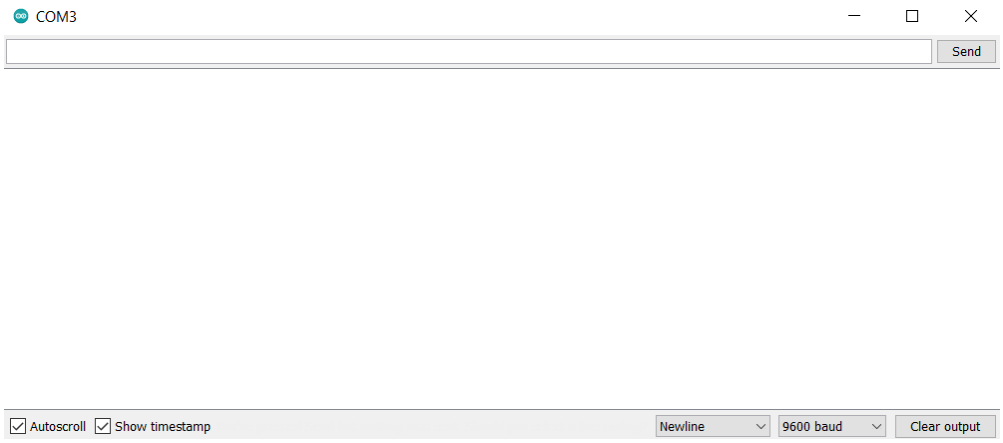
```
void setup() {
 // put your setup code here, to run once:
 Serial.begin(9600);
}

void loop() {
 // put your main code here, to run repeatedly:
 Serial.print("Hello world ");
}
```

Note: Use default.json for WOKWI.

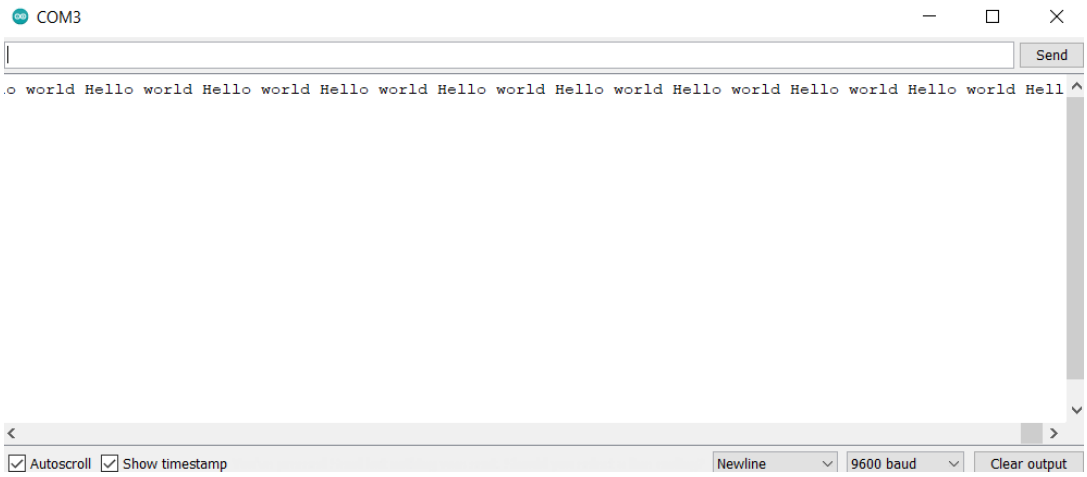
2. **Verify** the code. You will then be asked to save the code (or as it is commonly known, a sketch). Enter a **meaningful name** for your file, e.g., tutorial_02_helloworld. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

3. Go to **Tools** and then select **Serial Monitor**. The output should appear on the serial monitor, either in a separate window as shown below, or in a Serial Monitor tab under the coding area (if you use Arduino IDE 2.0).





4. **Upload** the code. You should now see Hello world messages on the serial monitor as shown continuously.



Trouble Shooting

1. Make sure that the baud rate specified in `Serial.begin()` is the same as the baud rate on the serial monitor. Revisit Lecture 02, if you don't remember how to change the baud rate on the serial monitor.

TO DO:

1. Read the following Arduino reference page on Remainder Operation (%)
   https://www.arduino.cc/reference/en/language/structure/arithmetic-operators/remainder/

   Note: you may find it useful to read the Arduino reference pages on other arithmetic operators too.

https://www.arduino.cc/reference/en#arithmetic-operators .

2. Reading the following Arduino reference page on < https://www.arduino.cc/reference/en/language/structure/comparison-operators/lessthan/ .

   Note: you may find it useful to read the Arduino reference pages on other comparison operators too, particularly >, <=, and >= which you may be using in this tutorial and Tutorial 03.
   https://www.arduino.cc/reference/en#comparison-operators .

3. Reading the following Arduino reference page on ++ https://www.arduino.cc/reference/en/language/structure/compound-operators/increment/ .
   Note: you may find it useful to read the Arduino reference pages on other compound operators too.
   https://www.arduino.cc/reference/en#compound-operators .

4. Read the following Arduino reference page on `for` https://www.arduino.cc/reference/en/language/structure/control-structure/for/ .

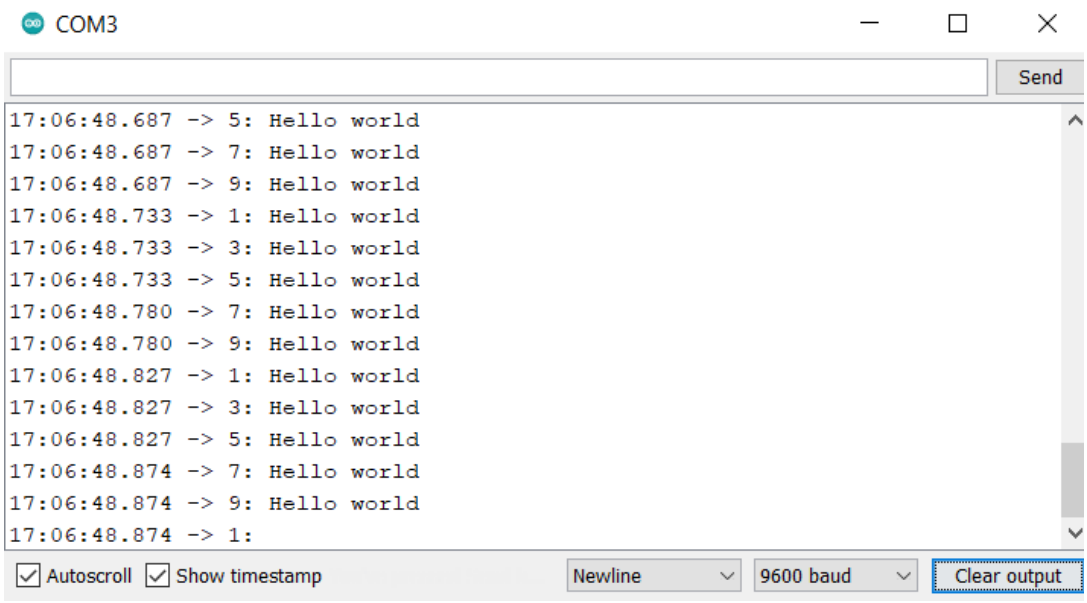
   Note: you may find it useful to read the Arduino reference pages on other control structures too.
   https://www.arduino.cc/reference/en#control-structure .

5. Read the following Arduino reference page on Equal to (==) https://www.arduino.cc/reference/en/language/structure/comparison-operators/equalto/ .

6. Modify tutorial_02_helloworld.ino to print out XXX: Hello world as shown, where XXX represents the odd numbers between 0 and 10.

   You will need to declare some variables for this modification. As these variables are not needed outside of `loop()`, it is suggested that you use local variables for the this task. Do select meaningful names for these variables.

   You can use `delay()` to slow down the output, if you wish. If you have forgotten how to use `delay()`, refers back to Tutorial 1.



7. **Verify** your modification and **Upload** the code to see if it produce the required output.

Practice 2.2

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.

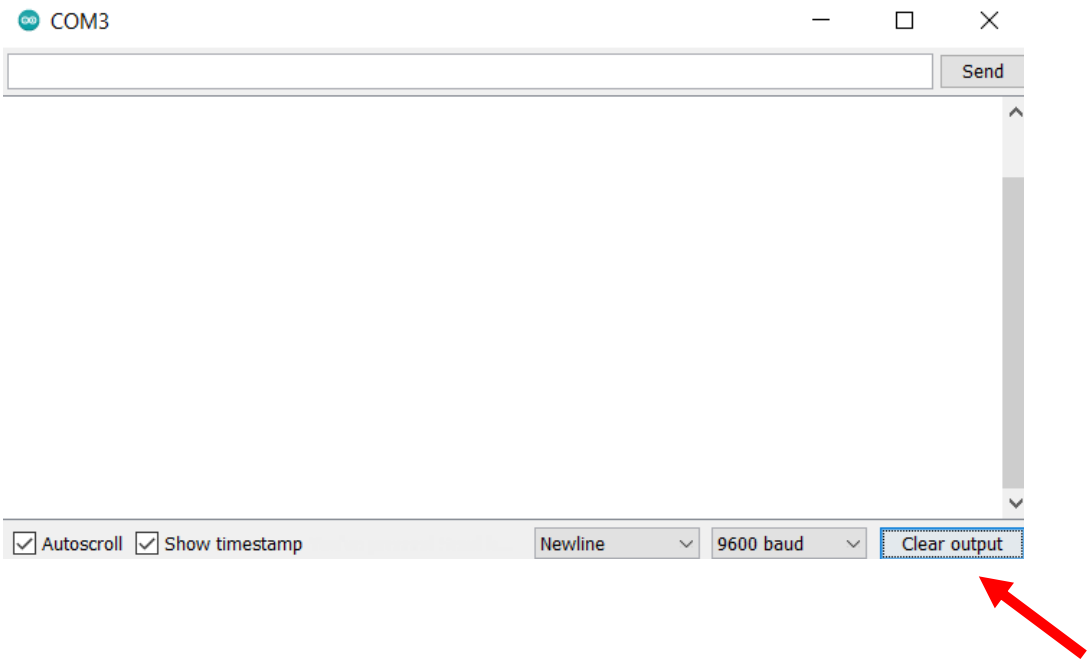
```
int incoming = 0;

void setup() {
 // put your setup code here, to run once:
 Serial.begin(9600);                    – change 9600 to 115200 for WOKWI
}

void loop() {
 // put your main code here, to run repeatedly:
 if(Serial.available() > 0)
 {
  incoming = Serial.read();
  Serial.print("Received: ");
  Serial.println(incoming);
 }
}
```

Note: Use default.json for WOKWI.

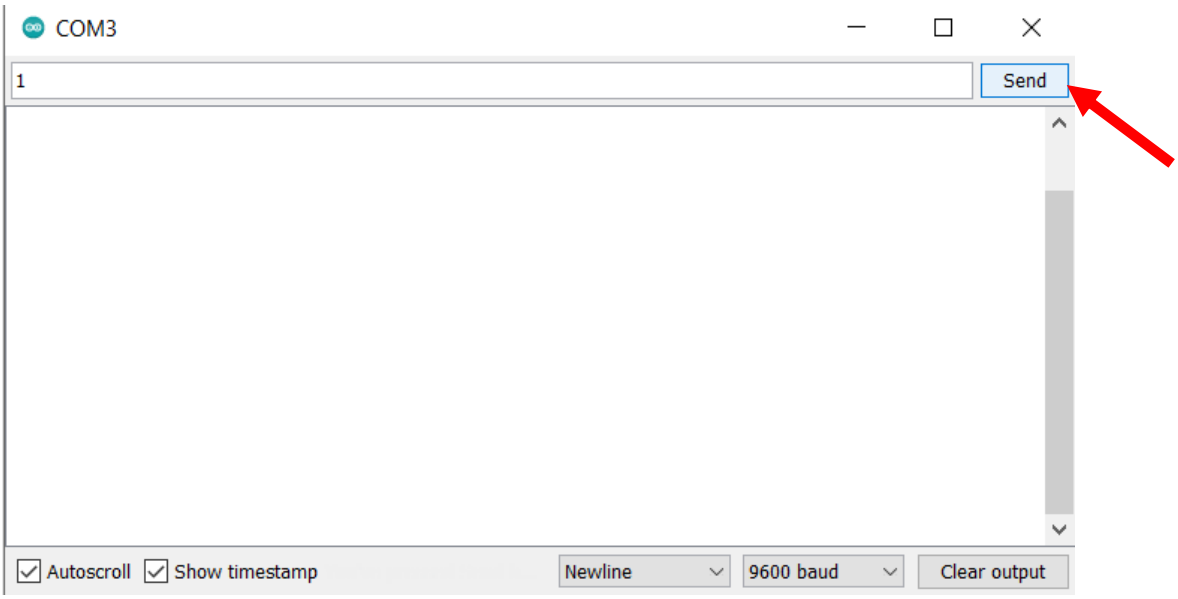
2. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_02_read. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

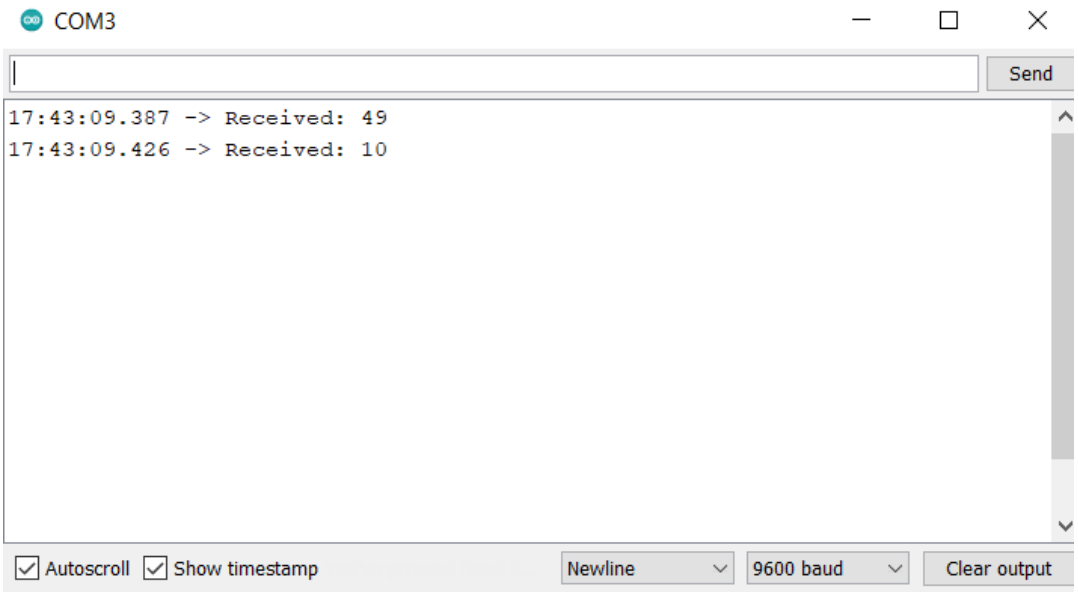
3. If a serial monitor isn't already opened, then go to **Tools** and select **Serial Monitor**.

4. **Upload** the code. You may wish to **Clear output** by using the button at the bottom right of the serial monitor. You should now see an empty serial monitor as shown.



5. Type in 1 into the input area of the serial monitor and press **Send**. You should now see the output from the code on the serial monitor as shown.





6. Read the following page about ASCII value https://www.asciitable.com/ . Make sure you understand why the value 49 and 10 is received.

7. Type in 10 into the input area of the serial monitor. Before pressing **Send** have a guess as to what values will be shown on the serial monitor. If the values that are shown on the serial monitor after you press `Send` aren't the same as what you have guessed, revisit the ASCII table in the link provided in the bullet point #6 (above), and the Arduino reference pages on:
    a. `Serial.read()`: https://www.arduino.cc/en/serial/read (see the section on Return value from the function),
    b. `Int`: https://www.arduino.cc/en/reference/int (see the Description specifically on data size).

Trouble Shooting

1. Make sure that the baud rate specified in `Serial.begin()` is the same as the baud rate on the serial monitor. Revisit Lecture 02, if you don't remember how to change the baud rate on the serial monitor.

TO DO:

1. Read the following Arduino reference page on `Serial.parseInt()` https://www.arduino.cc/reference/en/language/functions/communication/serial/parseint/ .

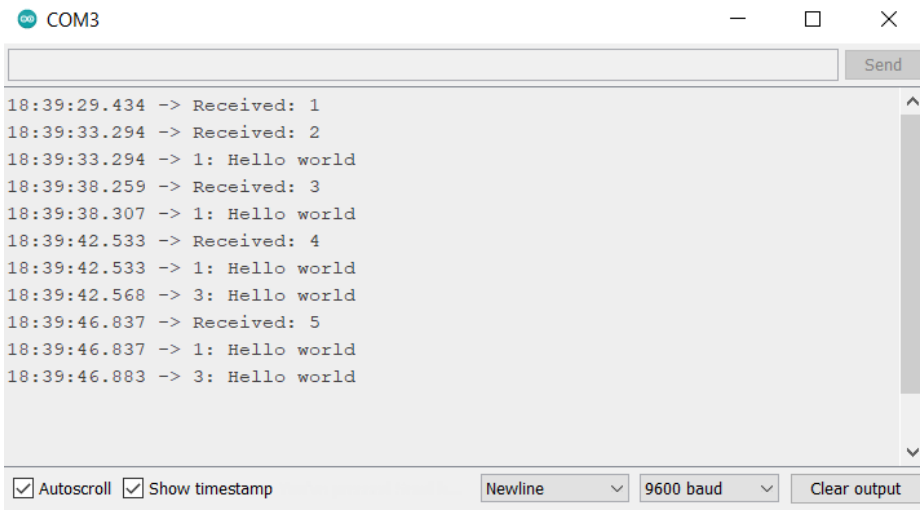
2. Reuse/combine the tutorial_02_helloworld.ino and tutorial_02_read.ino. Modify the combined program to dynamically print out

   Received: YYY
   XXX: Hello world

   where YYY represents the number the user types into the serial monitor, and XXX is the odd numbers between 0 and YYY.

   Your output should look like the example shown below, when the user entered 1 and then press **Send**, 2 and then press **Send**, 3 … up to 5.

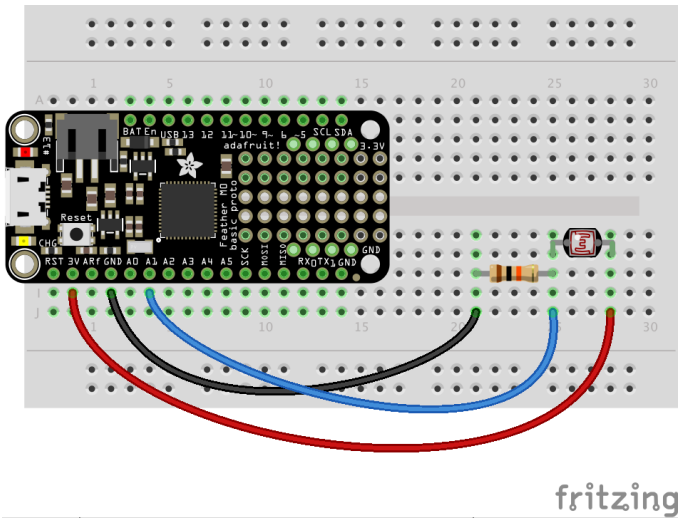
   Note: WOKWI may not work with 1.



3. **Verify** your modification and **Upload** the code to see if it produce the required output.

Practice 2.3

1. Connect photocell and resistor according to the following schema. Note where the power (3V), the Ground (GND) and the data (A1) are and connect them correctly.



Source: https://learn.adafruit.com/photocells/circuitpython

2. Type the skeleton code below into the coding area of your Arduino IDE.

```
int photocellPin = A1;    // the photocell and 10K pulldown are connected to a1 – change A1 to 4 for WOKWI
int photocellReading;     // the analogue reading from the sensor

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

}

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

  delay(1000);
}
```

Note: Use practice2_3.json for WOKWI. The simulator cannot measure the real brightness obviously.

Trouble Shooting

1. Make sure that the value of `photocellPin` specified in your program is the same as the schema, i.e., if you don't stick the jumper wire in A1 pin, then that value in your code will have to change accordingly.

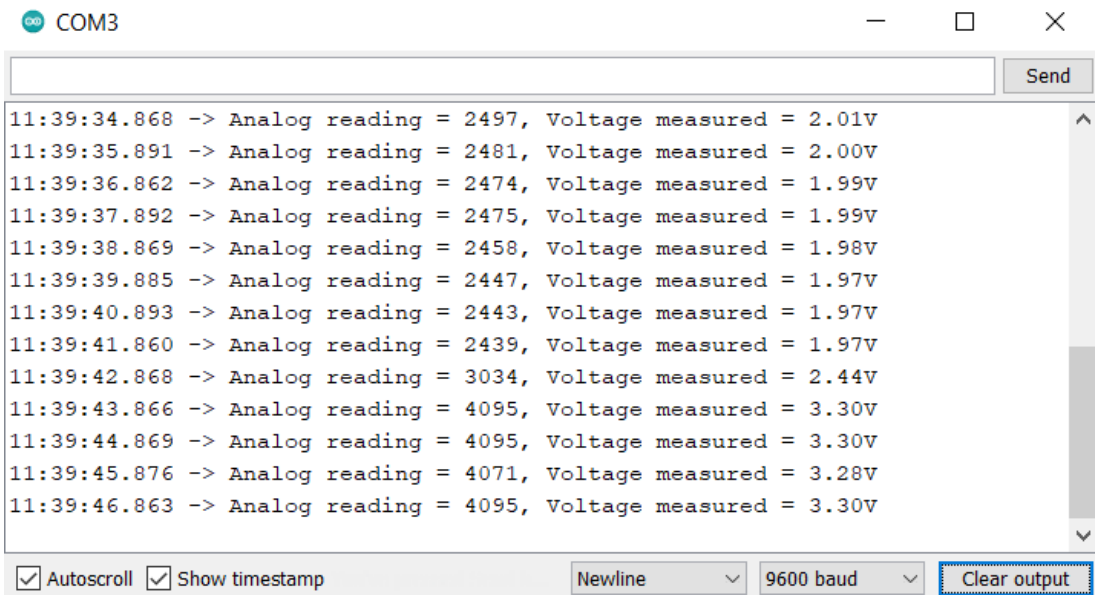
TO DO:

1. If you haven't done so already, use the link provided in the Theory part of this tutorial to make sure that you thoroughly understand how to get the analogue readings:
    a. `analogRead()`: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/ .

2. Revisit Lecture 02 for the equation to convert Analogue to Digital Converter (ADC) reading to Voltage measured.

    Note: If you haven't done so already, you may find it useful to read the Arduino reference pages on other arithmetic operators too.
    https://www.arduino.cc/reference/en#arithmetic-operators

3. Modify the code so that it will display the real-time analogue reading from photocell and Voltage measured as shown.



4. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_02_photocell. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

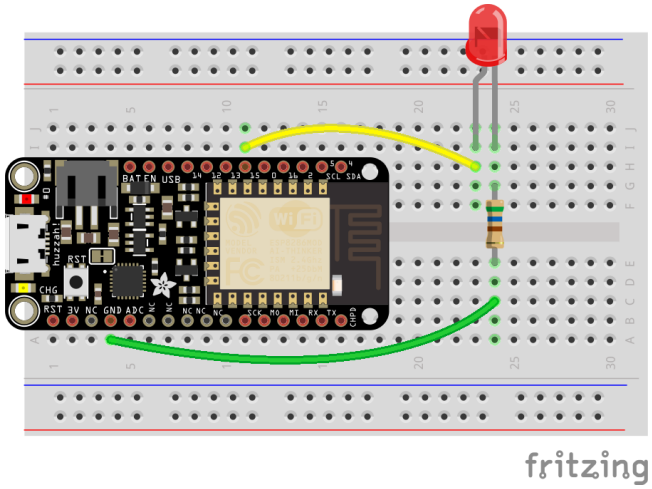
5. **Upload** the code to see if it produces the required output.

6. You can test your program and photocell by blocking the light e.g., using your hand(s) to cover the photocell, and see if the readings are changing.

Practice 2.4

1. Connect LED and resistor according to the following schema. Note where the Ground (GND) and the data (15) are and connect them correctly. The short leg of the LED (Cathode) should be connected to Ground.

   Note: Avoid looking right at the LED!!! They can be very bright.



Source: https://learn.adafruit.com/micropython-basics-blink-a-led/hardware

2. Modify your tutorial_01_onboard_LED.ino from blinking onboard LED to blink this new LED instead. Declare a global variable e.g., `LEDPin`, to store the value of the LED data pin, as you will need to use this value in both `setup()` and `loop()`.

   Note: Use practice2_4.json for WOKWI.

3. **Verify** your modification. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_02_LED. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

4. **Upload** the code to see if it now blinks the LED.

TO DO:

1. If you haven't completed the TO DO for Practice 2.3, do so first.

2. Reuse/combine tutorial_02_photocell.ino and tutorial_02_LED.ino so that it will:
   a. Display the real-time analogue reading from photocell and Voltage measured as shown in the TO DO for Practice 2.3.
   b. Turn on the LED when the environment is "dark".

   Note: you can specify the value for "dark", as this will depend on the environment you are working in, i.e., "dark" inside a room will be different from "dark" outside the room on a sunny day.

3. Combine the photocell and resistor connection (from Practice 2.3) with the connections of the LED and resistor (from Practice 2.4) together. You will obviously need to be a bit creative here, as both schemas are currently using some of the terminal strips.

   Note: Do make use of the power rails to make the connections of all components easier. For WOKWI, similar to Practice 2.3, photocellPin should be changed to 4.

4. If a serial monitor isn't already opened, then go to **Tools** and select **Serial Monitor**.

5. **Upload** the code to see if it produces the required output.

6. You can test your program and photocell by blocking the light e.g., using your hand(s) to cover the photocell, and see if the readings are changing.

Trouble Shooting

1. Make sure that the value of `photocellPin` and `LEDPin` specified in your program is the same as your connections, i.e., if you don't stick the jumper wire in A1 pin for `photocellPin` or 15 pin for `LEDPin`, then that values in your code will have to change accordingly.

2. LED lights are on when the current flow from anode (+) to cathode (-). Make sure that you connect the LED long leg (+) to the data pin, and the short leg (-) to Ground. With this set up, when you set the data pin to HIGH then the LED will light up and when you set the data pin to LOW then the LED will be off.

   However, if you connect the LED short leg (-) to the data pin, and the long leg (+) to power, the opposite will happen. So, when you set the data pin to LOW then the LED will light up, and when you set data pin to HIGH then the LED will be off.

Challenge 2.1

1. Modify tutorial_02_read.ino to dynamically process the user input as follows:
    a. When the user enters one number, the program should print the following out on the serial monitor

    Received: 0 YYY
    XXX: Hello world

    where YYY represents the number the user types into the serial monitor, and XXX is the odd numbers between 0 and YYY.

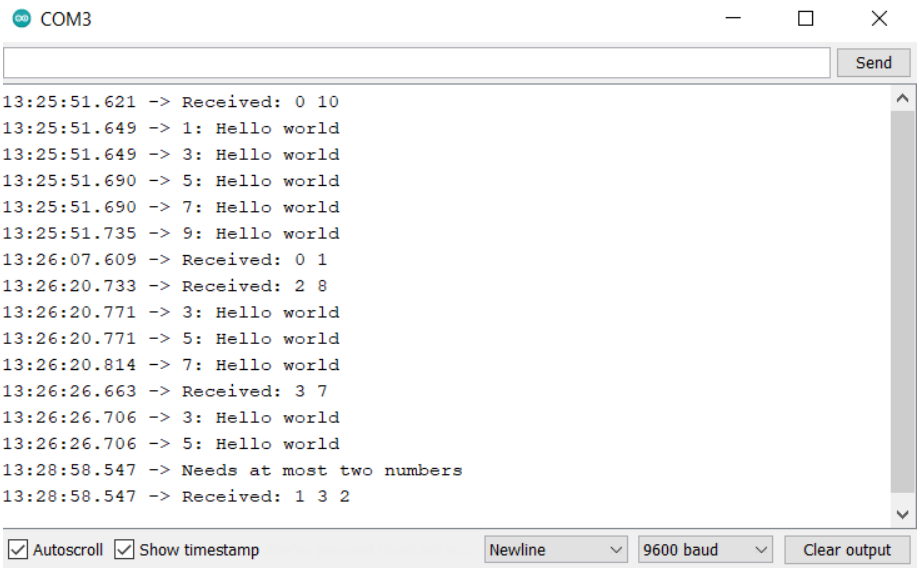
    b. When the user enters two numbers, the program should print the following out on the serial monitor

    Received: YYY ZZZ
    XXX: Hello world

    where YYY represents the smaller number of the two numbers that the user types into the serial monitor, ZZZ represents the larger number of the two numbers that the user types into the serial monitor and XXX is the odd numbers between YYY and ZZZ.

    Note: Use default.json for WOKWI.

    Your output should look like the example shown below, when the user entered 10 and then press **Send**, 1 and then press **Send**, 2 8 and then press **Send**, 7 3 and then press **Send**, 1 3 2 and then press **Send**.



    Note: WOKWI will behave slightly differently when more than 2 numbers are entered, i.e., it will not print “Needs at most two numbers”.

2. **Verify** your modification. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_02_parseInt_challenge. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.

3. **Upload** the code to see if it now produces the required output.