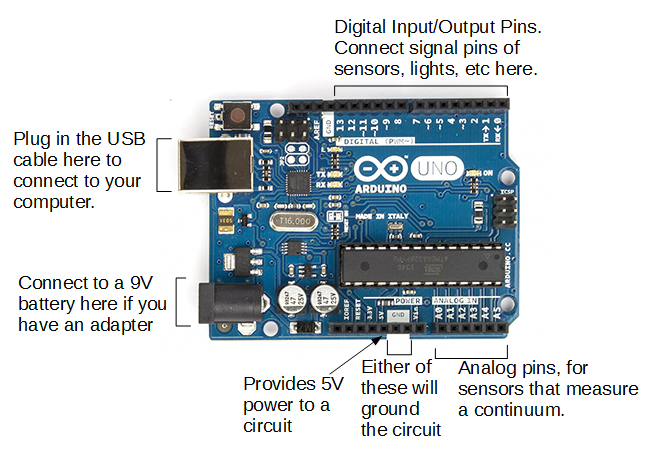
**Basic Arduino Workshop**

1. **Introduction to Arduino UNO**
   1. **Pins**

There are four main types of Pins

1. Digital
2. Analog
3. 5V / 3.3V
4. Ground



* + 1. **Digital Pins**

There are 14 digital pins (0 to 13), and the input is either 1 or 0. Similarly, the output is either 1 or 0. The value 1 corresponds to high voltage which is a range from (3.3V - 5V) while 0 corresponds to ground voltage.

Out of the 12 pins, there are digital pins with ~ sign. It means that these pins can support Pulse Width Modulation (PWM) and an analog value is modulated (i.e. represented) by a digital signal. It essentially allows the digital pin to simulate analog output.

Note that analog values can only be output via a PWM digital pin (i.e. Analog pins cannot produce analog values)

* + 1. **Analog Pins**

There are 6 analog pins (A0 to A5) and the input value ranges from 0 – 255. The analog output values ranges from 0 to 1023, however, these values cannot be output through the analog pins.

* + 1. **5V/3.3V**

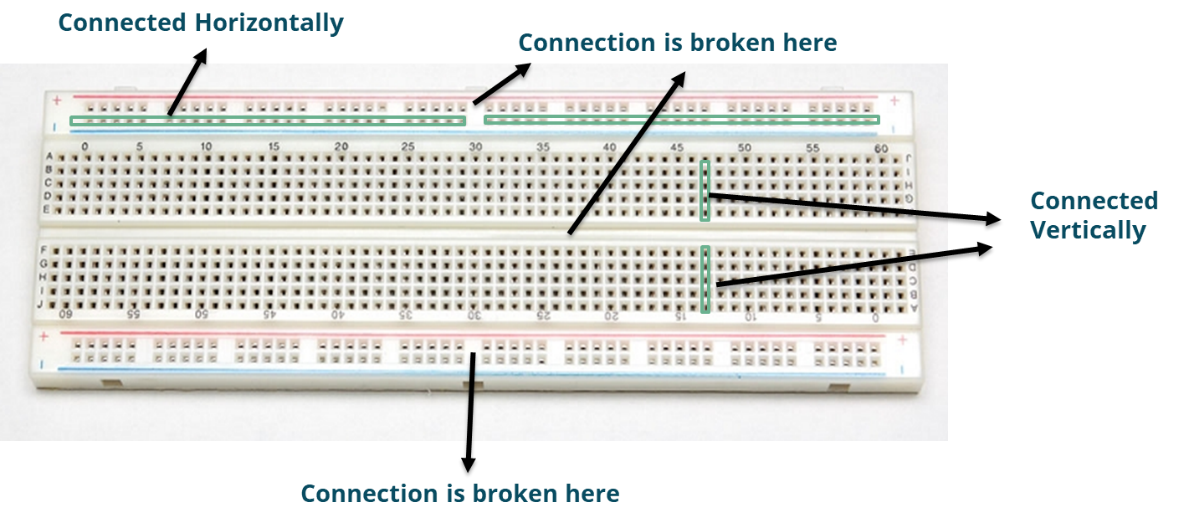
The 5V pin is used to produce the high voltage, however, the pin can only produce 5V unlike the digital pins which are able to change its output voltage.

The 3.3V pin is used to produce 3.3V, and its usage is dependent on the type of component used. Some components are sensitive to high voltages, hence 3.3V must be used instead. For instance, the Bluetooth Module[[1]](#footnote-1) should not be used with the 5V pin.

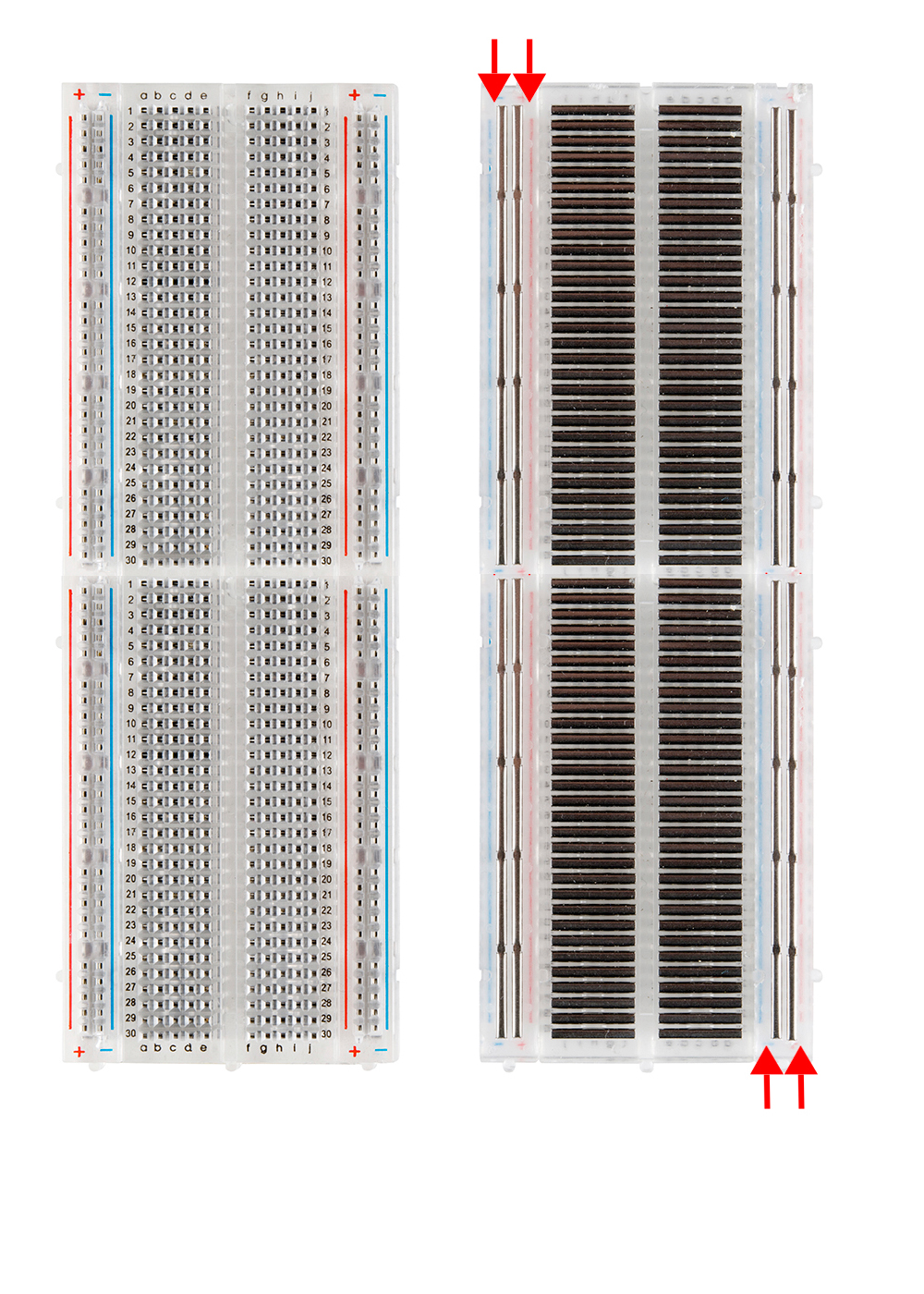
* + 1. **Ground**

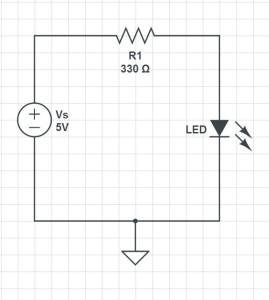
It is abbreviated by “GND” and it is used to complete the circuit.

* 1. **Breadboard**



The pins are connected to each other through the underlying connections within the breadboard



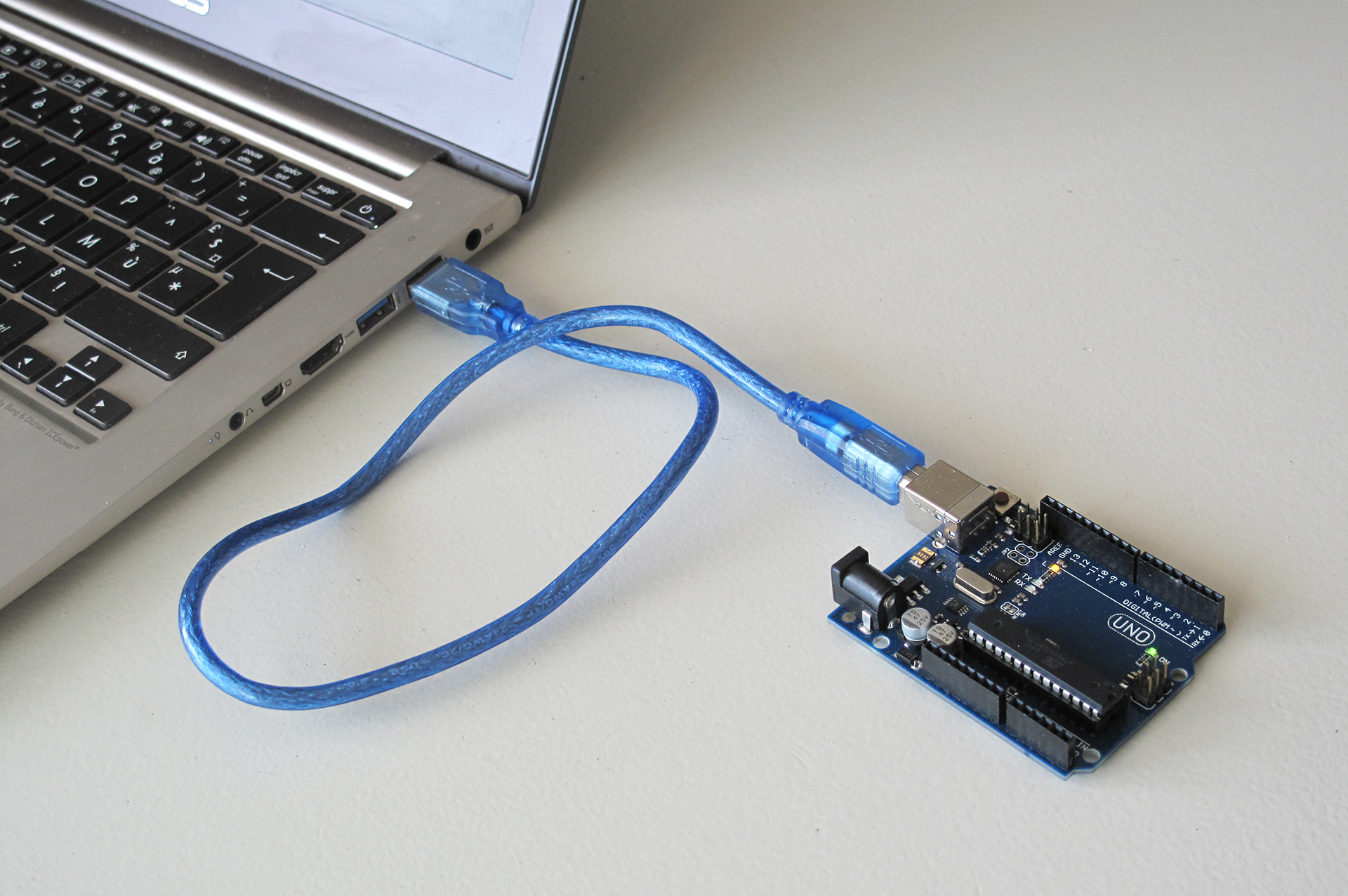
**** 

Example 1: Connecting a LED to an Arduino using a breadboard

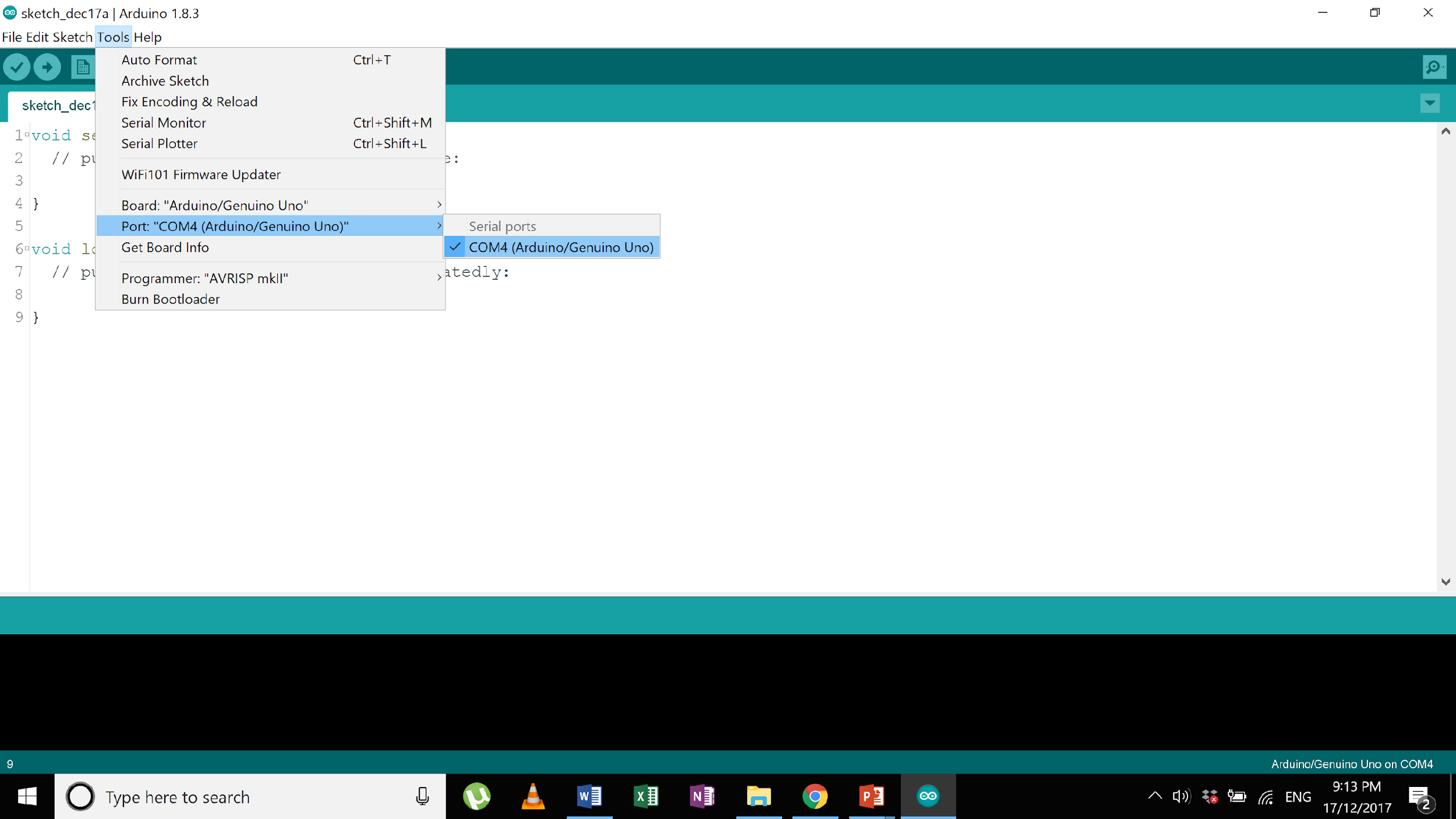
* 1. **Basic Functions**

|  |  |  |
| --- | --- | --- |
| Function | Values for each field | Remarks |
| pinMode(pin,mode) | pin   * Pin number (i.e. 5, A3)   mode   * INPUT * OUTPUT * INPUT\_PULLUP → activate the pull up circuit | * Changes what the Pin does * Included within void setup()   INPUT\_PULLUP   * Ensures that the digital input is either 0 or 1 * Built within the Arduino Board * Important when making a button |
| digitalWrite(pin,mode) | pin   * Pin number (i.e. 5, A3)   mode   * HIGH or LOW (5V or 0V) | * Change the output of the digital pin * If the pin has been set to input or input pull up, this function cannot be used as this function is used to produce an output |
| analogWrite(pin,value) | pin   * Pin number (i.e. 5, A3)   value   * 0-255 (Corresponds to a range from 0V to 5V) | * Change the output of the analog pin * Only digital pins with PWM can use this function * If the pin has been set to input or input pull up, this function cannot be used as this function is used to produce an output |
| digitalRead(pin) | pin   * Pin number (i.e. 5, A3) | * Returns the value which the pin is reading (0 or 1) * If the pin has been set to output, this function cannot be used as this function is used to read an input |
| analogRead(pin) | pin   * Pin number (i.e. 5, A3) | * Returns the value which the pin is reading (0 - 1023) * If the pin has been set to output, this function cannot be used as this function is used to read an input |
| Serial.begin(value) | value   * 9600 (rate at which Arduino is communicating with the computer) | * Serial.begin(value) is placed in void setup() |
| Serial.println(value) | value   * Any string placed in inverted commas (i.e. “Hello World”) * Any variable | * Specifier of the variable is not required |
| delay(value) | value   * The number of milliseconds to pause (unsigned long) | * Pauses Arduino for a specified amount of time (milliseconds) * Cannot run other tasks simultaneously |
| millis() |  | * Returns time (in milliseconds) since the board started running * The value can be stored in a variable * Overflow after 50 days |
| map(value, fromLow, fromHigh, toLow, toHigh) | Value   * the number to map   fromLow   * the lower bound of the value’s current range   fromHigh   * the upper bound of the value’s current range   toLow   * the lower bound of the value’s target range   toHigh   * the upper bound of the value’s target range |  |

1. **Arduino IDE**
   1. **Connecting Arduino to Computer**

****

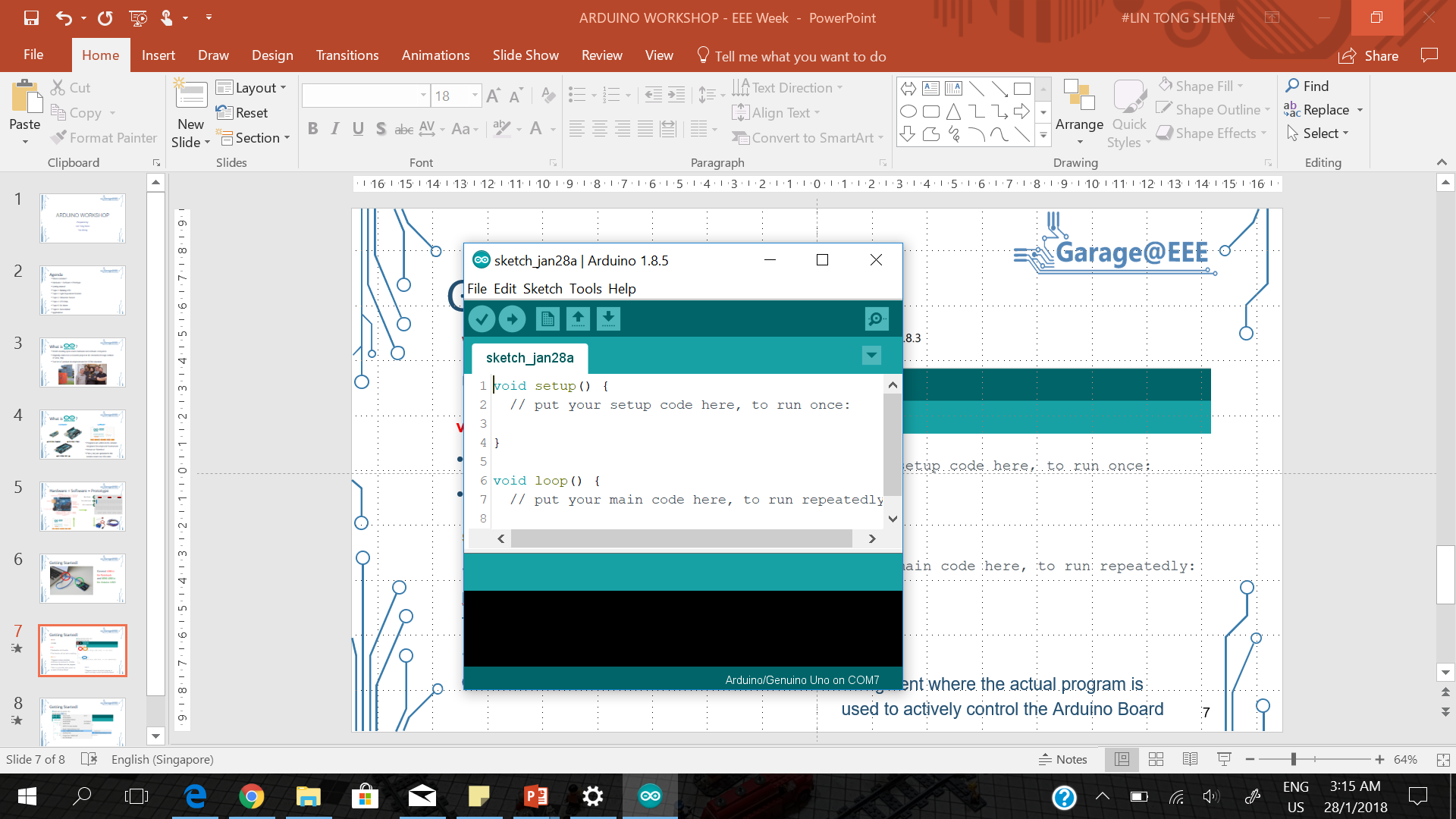
1. Connect USB to the Notebook and USB-B to the Arduino UNO



1. Ensure that “Arduino/Genuino Uno” is selected as the board
2. Next, ensure the following are selected:
   * Tools > Board > Arduino/Genuino Uno
   * Tools > Port > COM[X] (Arduino/Genuino Uno)

Note: the COM value is different on every computer

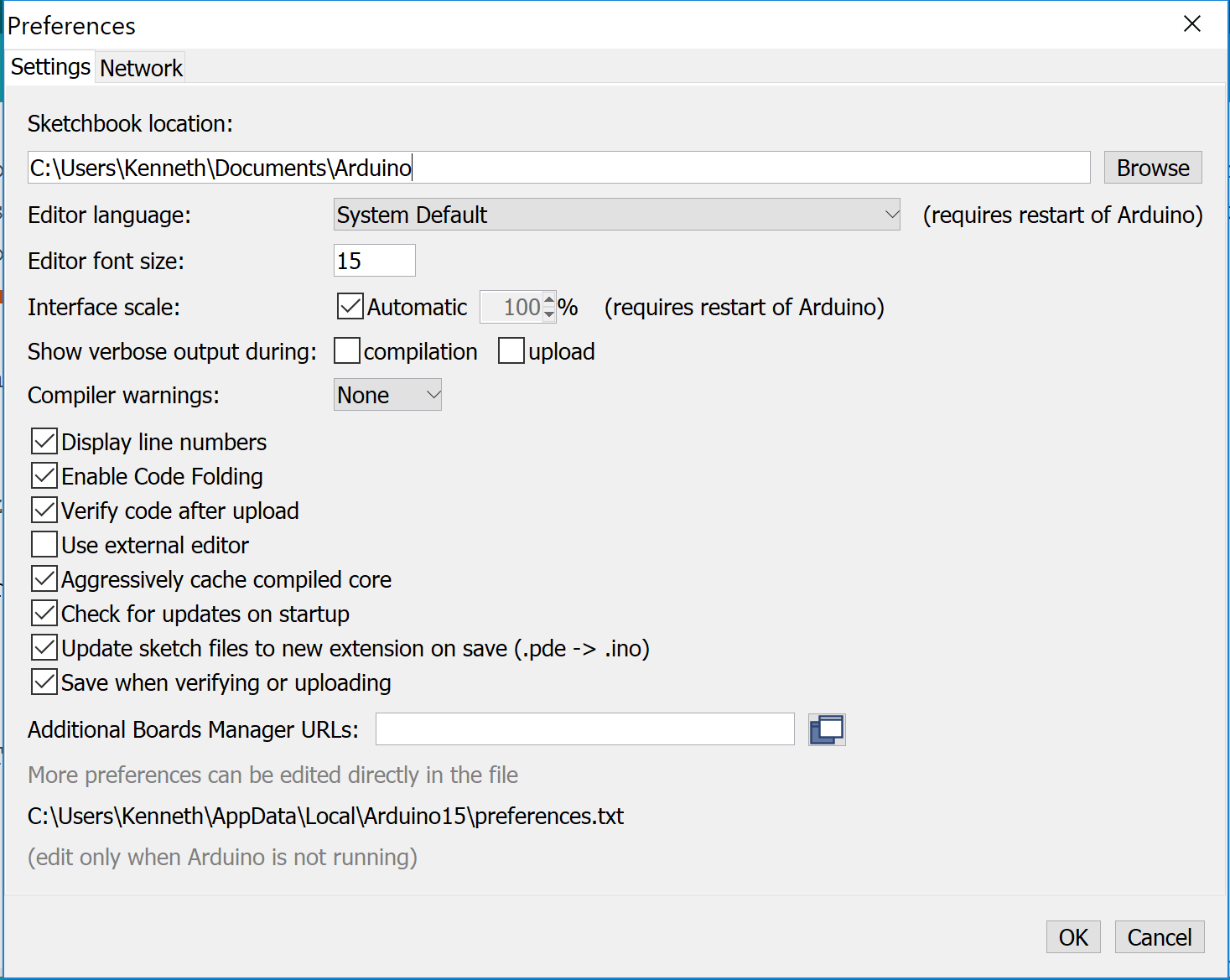
* 1. **Basic Functions**



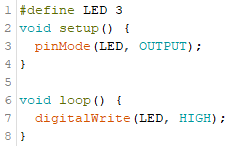
|  |  |  |
| --- | --- | --- |
|  | **Functionalities (colour coded)** | **Remarks** |
| **IDE** | **Verify** | Compile the program and check for errors |
| **Upload** | Upload program to the Arduino Board |
| **Serial Monitor** | Arduino can communicate with the computer board |
| **Board Information** | Provides information about the Arduino board |
|  | | |
| **Code** | **setup()** | Segment where variables, pinModes are declared to initialise the Arduino Board and the program  Will run once after each power up or reset of Arduino Board |
| **loop()** | Segment where the actual program is used to actively control the Arduino Board  Code in this segment will be run continuously |

**Enable Line numbers**

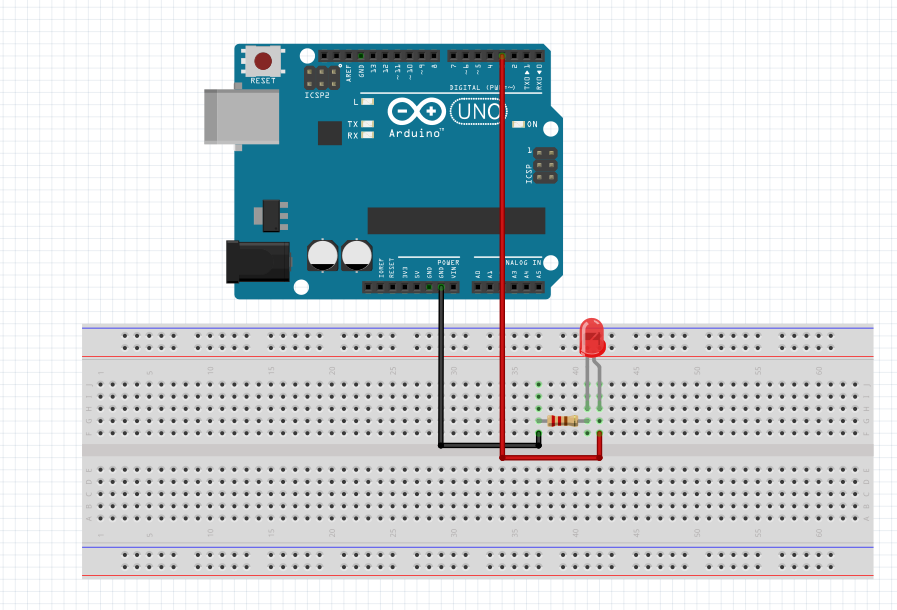
1. Go to: File > Preferences
2. Check the “Display line numbers” check box

****

1. **LED**
   1. **Turn on LED using a digital pin**

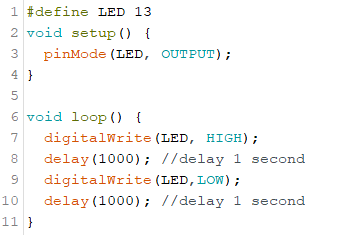
****

Pin 3 is told to output a voltage and subsequently, it is used to output HIGH (i.e. 5V).

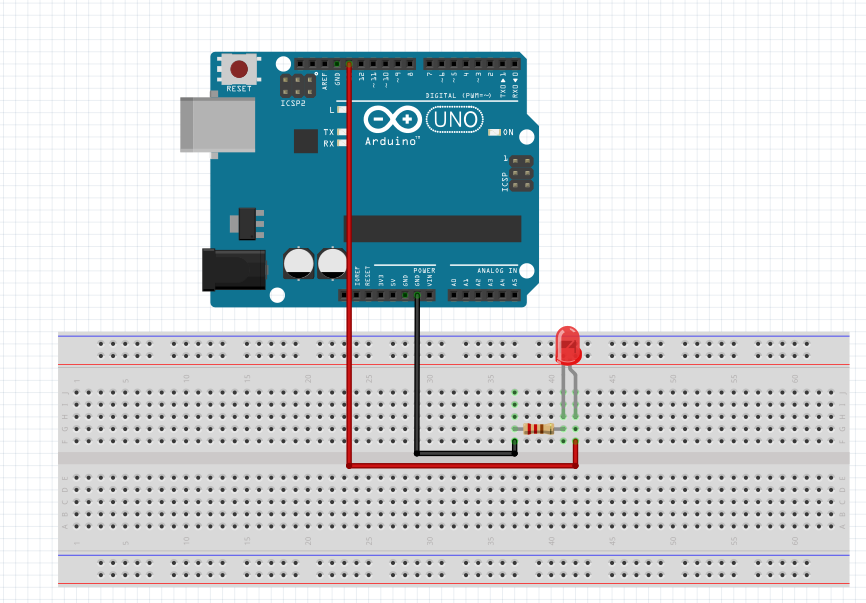


Note: As the LED is a diode, the longer end on the LED must be connected to the pin with higher voltage for current to flow.

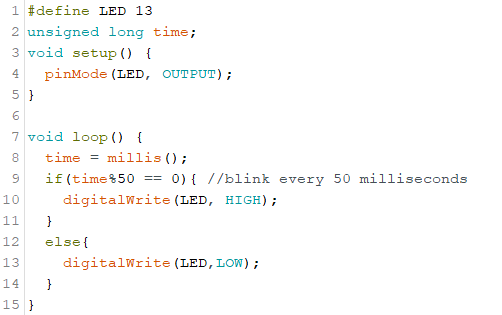
* 1. **Make an LED blink**

****

In the code, pin 13 is told to output a voltage and it is made to alternate between HIGH (i.e. 5V) and LOW (i.e. 0V) output every 1000 millisecond (i.e. 1 second) using the delay function. When it outputs 5V, current flows and the LED turns on. When the pin outputs 0V, no current flows and the LED turns off. Hence, by alternating the output of the digital pin, and using the delay function, the LED will blink every second.



* 1. **Make an LED blink without delay**

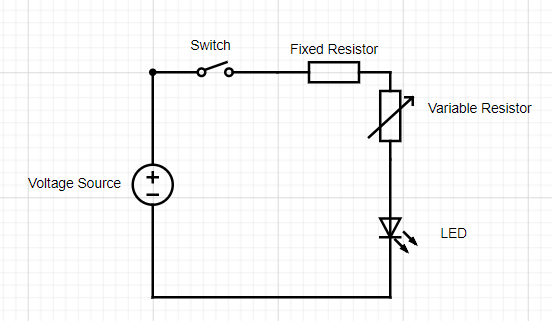
****

A major disadvantage of implementing the delay function is that the Arduino pauses every time the delay function is used. As a result, it is unable to run other tasks concurrently. Hence, we would want to avoid the use of delay by simulating the delay function using millis(). millis() returns the amount of time in milliseconds which the programme has been running and this value is stored in a variable labelled time. The variable is initialised as an unsigned datatype as time is always positive. It is also initialised as long datatype to accommodate large values returned from millis() which exceeds the range of the integer data type.

In the above code, the pin will be set to output 5V every time the amount of time the programme has been running is a multiple of 50. This means that the circuit will turn on every 50 milliseconds, hence simulating the blinking effect.

* 1. **Make an LED dim**

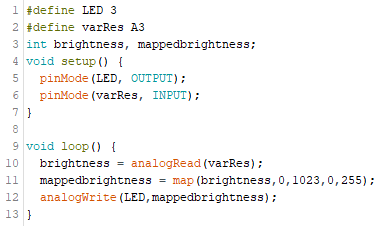
To change the LED’s brightness, the direct method to do so is to change the amount of current in the circuit by changing the resistance in the circuit. If less current is supplied, the LED will dim.



However, we can change the brightness of the LED by changing how quickly it blinks.

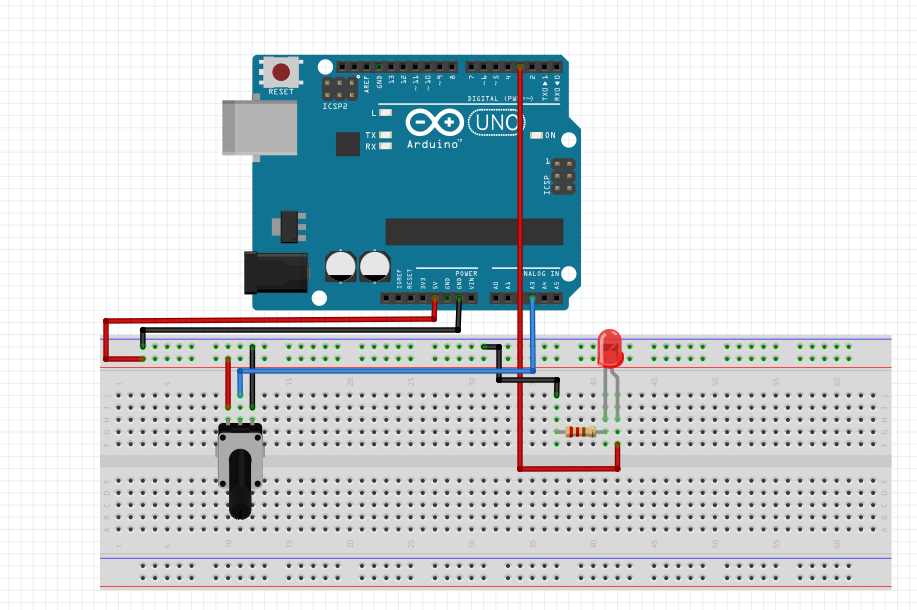
If it blinks very rapidly, our eyes are unable to detect the point in times when the LED turns on and off and perceive as if no blinking is taking place. By changing the ratio of the amount of time it is turned on to the amount of time it is turned off, we will perceive a change in brightness.

Another way to change the brightness of the LED is to change the voltage in the circuit, which causes the current in the circuit to change. Hence, we will use analogWrite() to output a range of voltages. To determine the voltage which the LED pin needs to ouput, we will use a variable resistor to control the output voltage. A pin will be used to read the voltage at the variable resistor and the value will be stored in a variable. Subsequently, the value stored in the variable is used to determine the voltage in the LED circuit.



In the code, the Arduino will read the voltage at the variable resistor using pin A3, store it in a variable, map the value to a range of (0-255), and output the same voltage using a PWM digital pin, pin 3.

The map function is required as analogRead returns a value from 0 to 1023, however, analogWrite can only output values from 0 to 255. Hence, we need to limit the values to 255 using the map function. If we do not, an analogRead value of 256 will overflow and take on a value of 1. This will cause the output voltage to be very low, which is not what we intended. Without the map function, the LED will go from dark to bright 3 times when the variable resistor is turned from one end to the other.



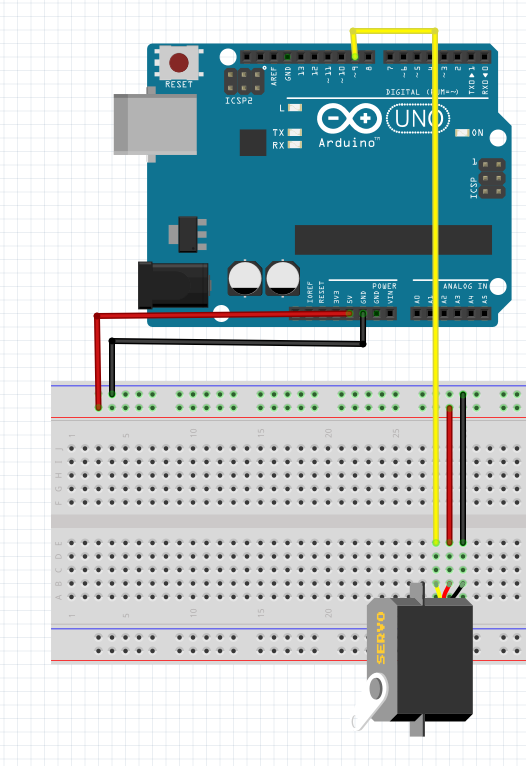
1. **Servo**
   1. **Introduction to Servo**

A servo is a type of motor which can be controlled very precisely using a feedback mechanism. In the context of Arduino, the servo shaft can be positioned between 0 and 180 degrees.

It is different from a dc motor which can rotate 360 degrees, hence, a servo cannot be used to control wheels.

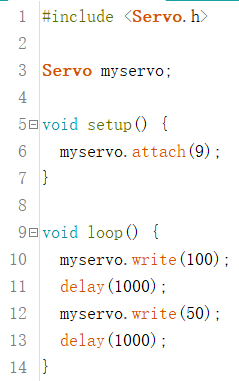
A servo has three connections:

* + 1 Black/Brown ground wire.
  + 1 Red power wire (around 5V).
  + 1 Yellow or White PWM wire.



**Connect:**

1. Yellow wire -> pin 9
2. Red wire -> 5V
3. Brown wire -> Ground



} Initialization

} Using pin 9 to

connect servo motor

} Rotate to 100 degrees

} Include library

} Wait for 1 second

} Rotate back to 50°

} Wait for 1 second

The code first calls the Servo library, this allows the user to control the servo easily using functions written in the library. Next, a Servo object called myservo is initialised and pin 9 is used to control the servo. Subsequently, the servo is made to turn to 100 degrees and 50 degrees every second. Note that a waiting time **must** to be included before the next function is called as the servo needs some time to turn.

* 1. **Controlling a Servo using a variable resistor**

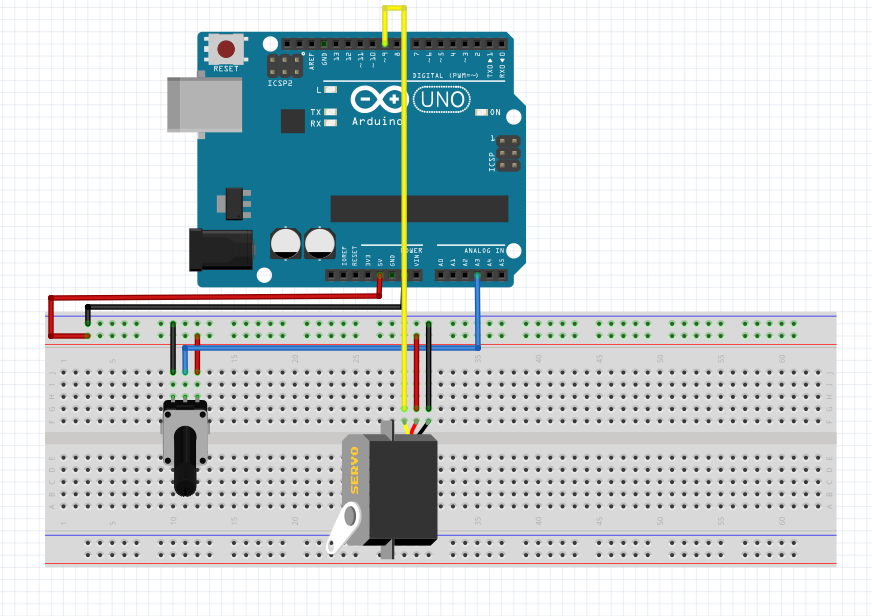
**Goal:** Make the variable resistor into a “joy stick”. When the variable resistor is turned to the left beyond a certain point, the servo turns left. When the variable resistor is turned to the right beyond a certain point, the variable resistor turns right.



Similar to the previous example, the servo library is called, and a Servo object is initialised. As a variable resistor is used, another pin, A2, is used to read the voltage at the variable resistor.

Facing the centre pin away from you, when the variable resistor is turned to the left, the resistance increases, and the voltage reading at the variable resistor decreases. The opposite occurs when the variable is turned to the right.

Hence, when we turn the variable resistor to the left and detect that the analogRead value from the variable resistor is low (i.e. 200), the position of the servo is decremented (i.e. turned to the left). Similarly, when we turn the variable resistor to the right and detect that the analogRead value from the variable resistor is high (i.e. 800), the position of the servo is incremented (i.e. turned to the right). Note that the values 200 and 800 are arbitrary and it is obtained after calibration.



1. Modules are hardware components attached to the Arduino externally to provide additional functionalities. [↑](#footnote-ref-1)