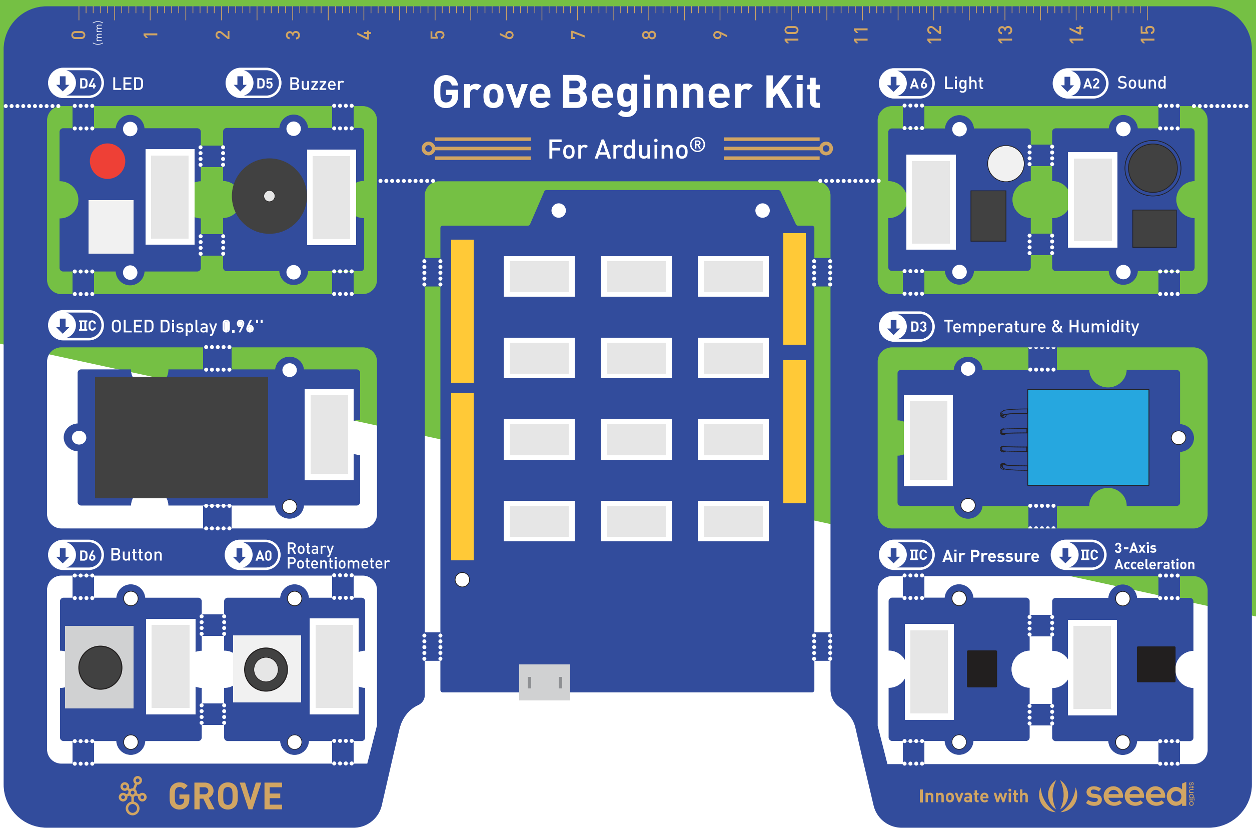
Grove Beginner Kit For Arduino®

User Manual



Arduino UNO compatible board included

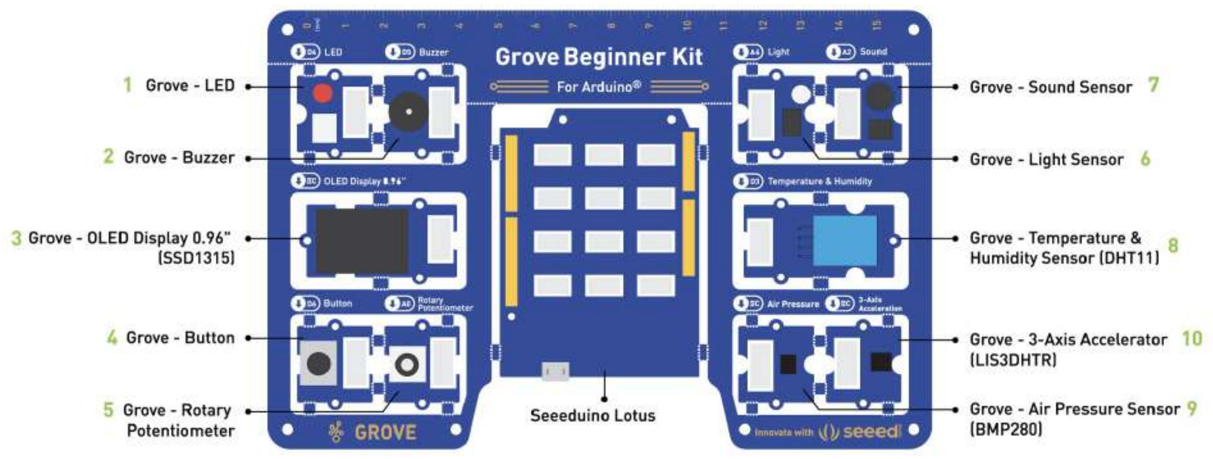
10 well-selected modules and all pre-wired

12 tutorials, from basic concept to fun projects

Grove Beginner Kit For Arduino®

Grove Beginner Kit for Arduino is one of the best Arduino Beginner Kit for beginners. It includes one Arduino compatible Board and 10 additional Arduino sensors and all in one-piece of PCB design. **All the modules have been connected to the Seeeduino through the PCB stamp holes so no Grove cables are needed to connect**. Of course, you can also take the modules out and use Grove cables to connect the modules. You can build any Arduino project you like with this Grove Beginner Kit For Arduino.

Hardware Overview


1 Grove - LED
2 Grove - Buzzer
3 Grove - OLED Display 0.96" (SSD1315)
4 Grove - Button
5 Grove - Rotary Potentiometer
Seeeduino Lotus
Grove - Sound Sensor 7
Grove - Light Sensor 6
Grove - Temperature & Humidity Sensor (DHT11) 8
Grove - 3-Axis Accelerator (LIS3DHTR) 10
Grove - Air Pressure Sensor (BMP280) 9
Grove Beginner Kit For Arduino®
GROVE

Note: Dimensions - 17.69 * 11.64 * 1.88cm

1. **Grove - LED:** Simple LED module
2. **Grove - Buzzer:** Piezo Buzzer
3. **Grove - OLED Display 0.96":** 128×64 dot resolution High brightness,self-emission and high contrast ratio Big screen on a compact design Low power consumption.
4. **Grove - Button:** Push button for human input interfaces
5. **Grove - Rotary Potentiometer:** Rotary knob for human input interfaces
6. **Grove - Light:** Detects surrounding light intensity
7. **Grove - Sound:** Detects surrounding sound intensity
8. **Grove - Temperature & Humidity Sensor:** Detects surrounding temperature and humidity values
9. **Grove - Air Pressure Sensor:** Detects surrounding atmospheric pressure
10. **Grove - 3-Axis Accelerator:** Detects object acceleration
11. **Seeeduino Lotus:** Arduino Compatible Board with Grove Ports

Index

Note:
By default, Grove modules are connected to Seeeduino via PCB stamp holes. This means you don't need to use Grove cables to connect if not broken out. The default pins are as follow:

Modules	Interface	Pins/Address
LED	Digital	D4
Buzzer	Digital	D5
OLED Display 0.96"	I2C	I2C, 0x78(default)
Button	Digital	D6
Rotary Potentiometer	Analog	A0
Light	Analog	A6
Sound	Analog	A2
Temperature & Humidity Sensor	Digital	D3
Air Pressure Sensor	I2C	I2C, 0x77(default) / 0x76(optional)
3-Axis Accelerator	I2C	I2C, 0x63(default)

Breakout Instruction

Attention:
Please be careful not to cut your hands when using a knife

If you prefer to use the modules in elsewhere then you can simply follow the procedures to break the modules out!

Step 1

Use a knife or a sharp object to cut at the stamp holes that connect the sensors together

Step 2

Shake the modules up and down and it should come out quite easily!

Part List

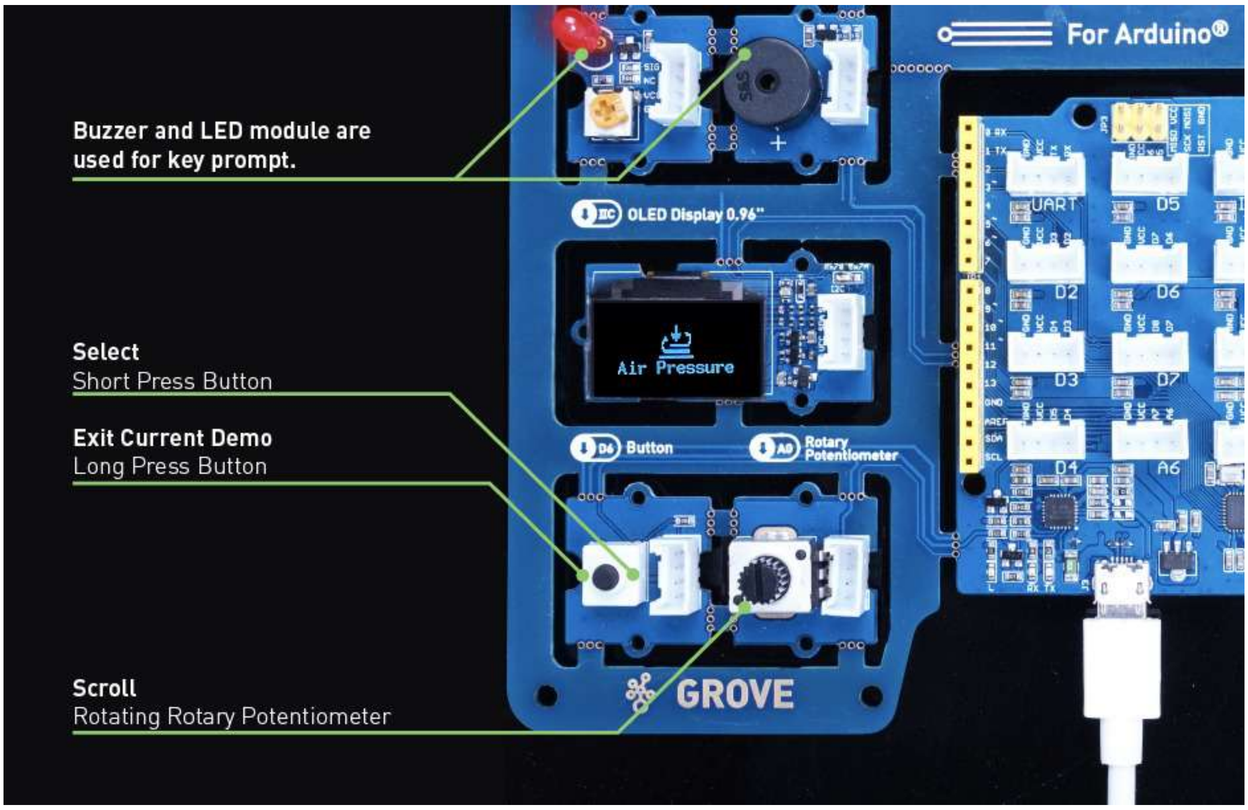
Modules	**Quantity**
Sensors	
Temperature & Humidity Sensors	x1
3-Axis Accelerometers	x1
Air Pressure	x1
Light Sensor	x1
Sound Sensor	x1
Input Modules	
Rotary Potentiometer	x1
Button	x1
Output Modules	
LED	x1
Buzzer	x1
Display Module	
OLED Display	x1
Grove Cables	x6
Micro USB Cable	x1

Learning Objectives

- Basics of Open Source Hardware Systems.
- Basic Arduino Programming.
- Communication principles and methods for sensors.
- Hands-on implementation of Open Source Hardware projects.

Plug and Play Unboxing Demo

The Grove Beginner Kit has a plug and plays unboxing demo, where you first plug in the power to the board, you get the chance to experience all the sensors in one go! Use the button and rotary potentiometer to experience each sensor demo!



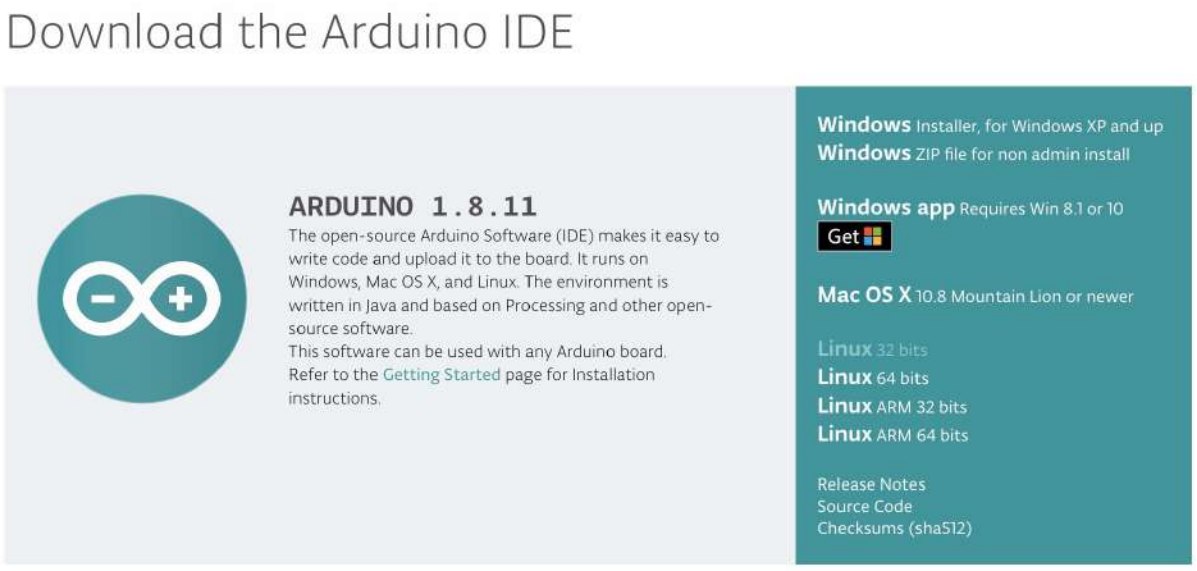
- **Scroll** -> Rotating Rotary Potentiometer
- **Select** -> Short Press Button
- **Exit Current Demo** -> Long Press Button

Buzzer and LED module are used for key prompt.

How to Get Started With Arduino

Install the Arduino IDE

- **Arduino IDE** is an integrated development environment for Arduino, which is used for single-chip microcomputer software programming, downloading, testing and so on.
- Download and Install Arduino IDE for your desired operating system here.


Download the Arduino IDE

ARDUINO 1.8.11
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
Get

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

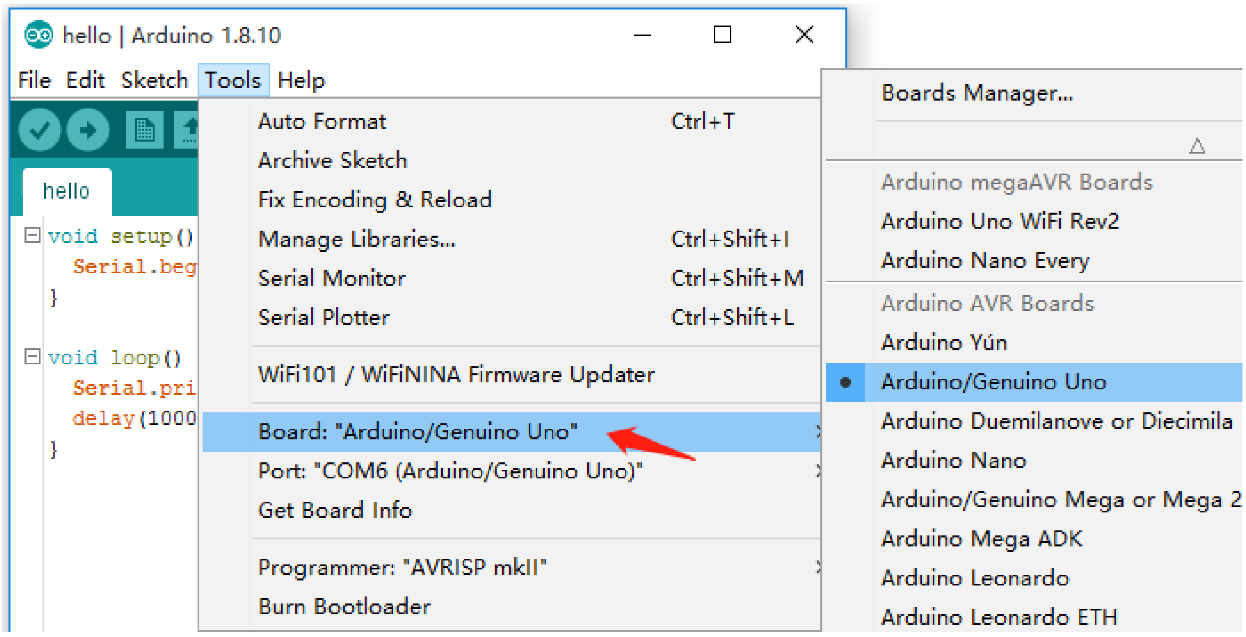
Release Notes
Source Code
Checksums (sha512)

Install the USB driver

- Arduino connects to the PC via a USB cable. The USB driver depends on the type of USB chip you're using on your Arduino. *Note: USB chips are usually printed on the back of the development board.*
  - Download the CP2102 USB Driver. **Note:** Download according to your OS.
  - After the driver installation is completed, connect Arduino to the USB port of PC with a USB cable.
  - **For Windows users:** You can see it in `My Computer` -> `Properties` -> `Hardware` -> `Device Management`. A `COM` will appear.
  - **For Mac OS users:** You can navigate to on the top left corner, and choose `About this Mac` -> `System Report...` -> `USB`. A CP2102 USB Driver should appear.
  - If the driver is not installed, or if the driver is installed incorrectly (not matching the chip model), it will appear as an "unknown device" in the device manager. At this point, the driver should be reinstalled.

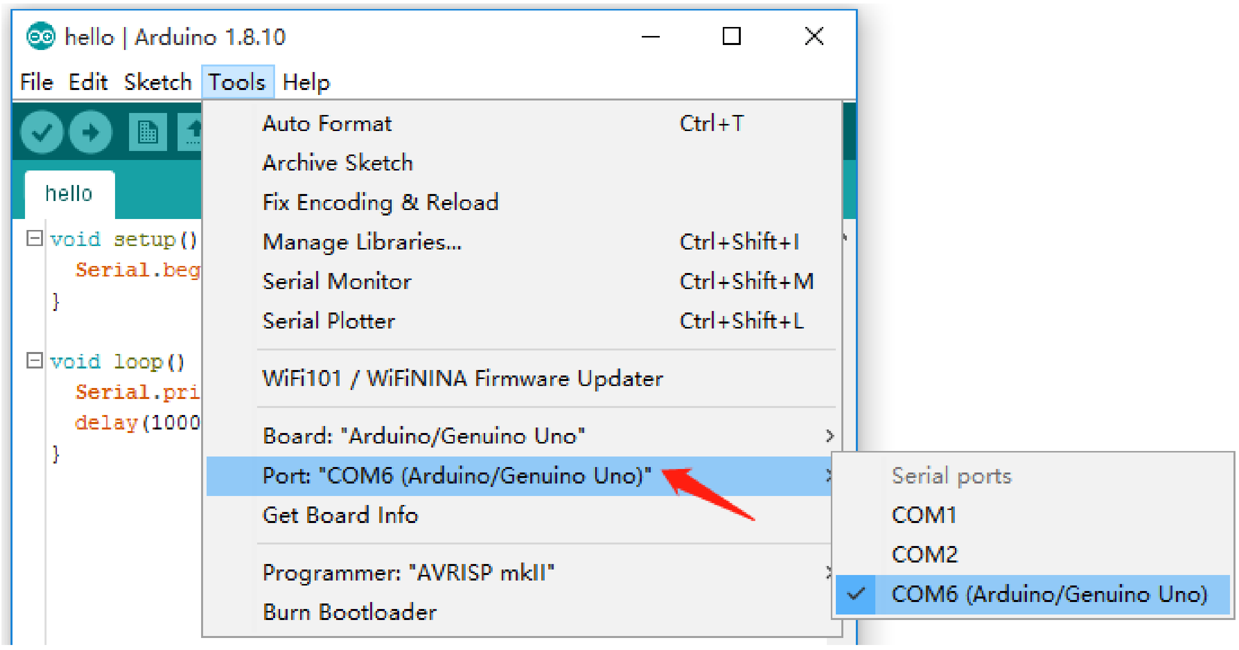
Start the Arduino IDE

1. Open the **Arduino IDE** on your PC.
2. Click on `Tools` -> `Board` to select the correct Development Board Model. Select **Arduino/Genuino Uno** as Board.



```
void setup()
  Serial.beg
}

void loop()
  Serial.pri
  delay(1000
}
```

3. Click `Tools` -> `Port` to select the correct Port (the Serial Port showed in Device Manager in the previous step). In this case, `COM6` is selected. **For Mac OS users**, it should be `/dev/cu.SLAB_USBtoUART`.



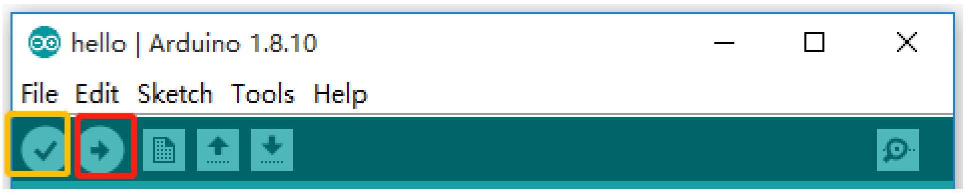
```
void setup()
  Serial.beg
}

void loop()
  Serial.pri
  delay(1000
}
```

hello | Arduino 1.8.10
File Edit Sketch Tools Help
hello
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM6 (Arduino/Genuino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
COM2
✓ COM6 (Arduino/Genuino Uno)

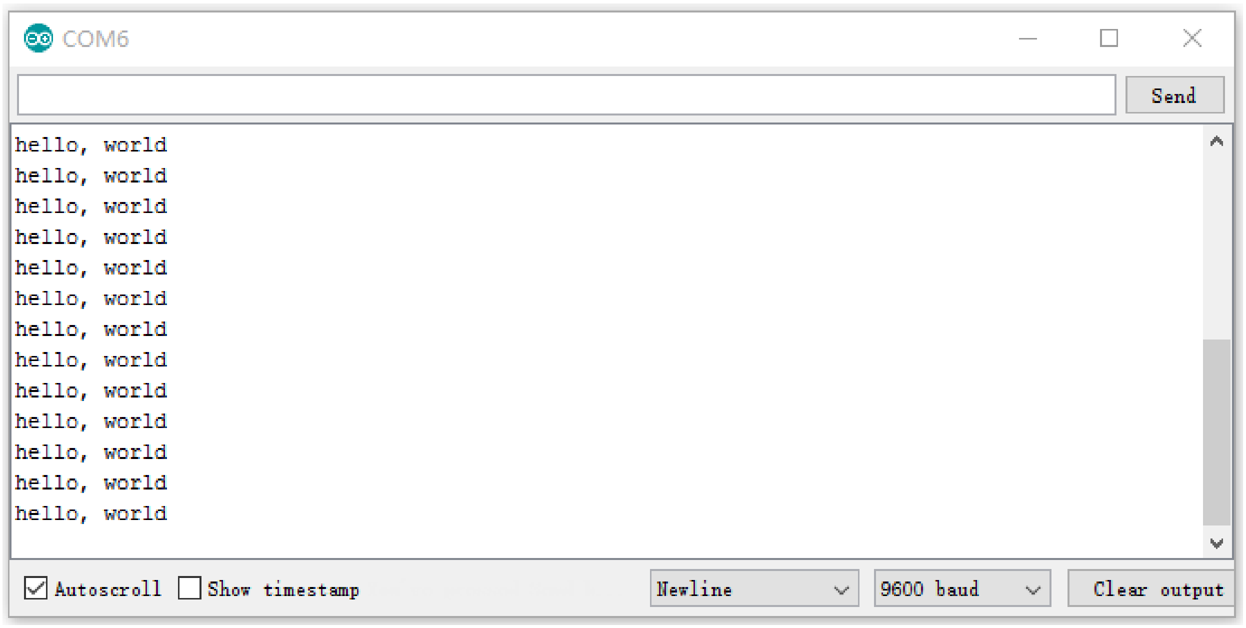
1. Create a new Arduino file and name it `Hello.ino`, then copy the following code into it:

```
void setup() {
  Serial.begin(9600); // initializes the serial port with a baud rate of
 9600
}
void loop() {
  Serial.println("hello, world"); // prints a string to a serial port
  delay(1000); //delay of 1 second
}
```

5. In the upper left corner of the Arduino IDE, there are two buttons, **Verify and Upload**. First, press the Verify button(✓) to compile. After the compilation is successful, press the upload button(→).

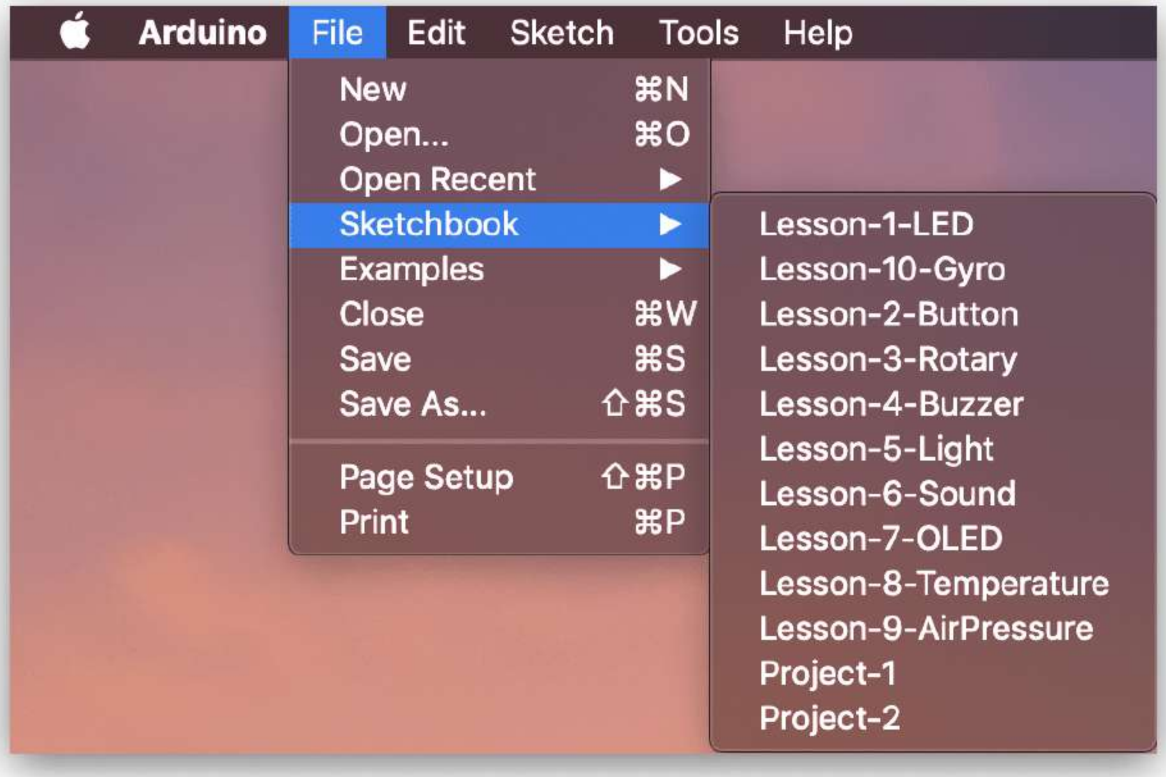


6. Navigate to `Tools` -> `Serial Monitor`, or click the **Serial Monitor** in the upper right corner(Magnifier Symbol), you can see the program running results:



```
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
hello, world
```

Note: If you installed the portable Arduino IDE from our USB Drive, you can find all the module demos in the **Files** -> **Sketch Book**, as well as all the module libraries, are pre-installed with Arduino IDE!



Note:
All modules are pre-wired on a single circuit board, so no cables and soldering are needed. However, if you break out the modules and want to connect them with Grove cables, please kindly check the Breakout Guide.

Lesson Guide

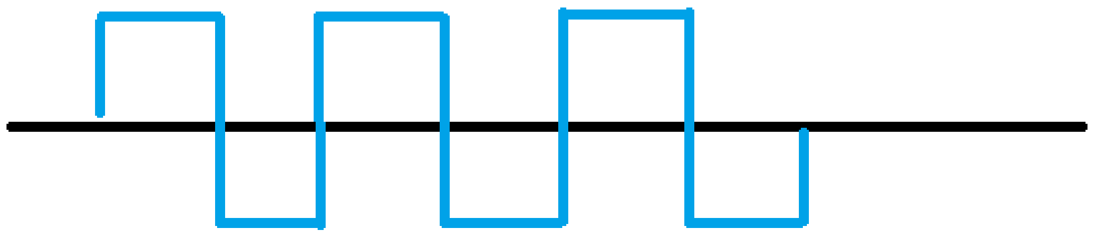
Lesson 1: Blinking with the LED

We have completed the output "Hello world" program. Now let's learn how to light the LED module. We know the three basic components of a control system: Input, Control, and Output. But lighting up LED uses only the output, not the input. Seeeduino is the control unit, the LED module is the output unit and the output signal is a digital signal.

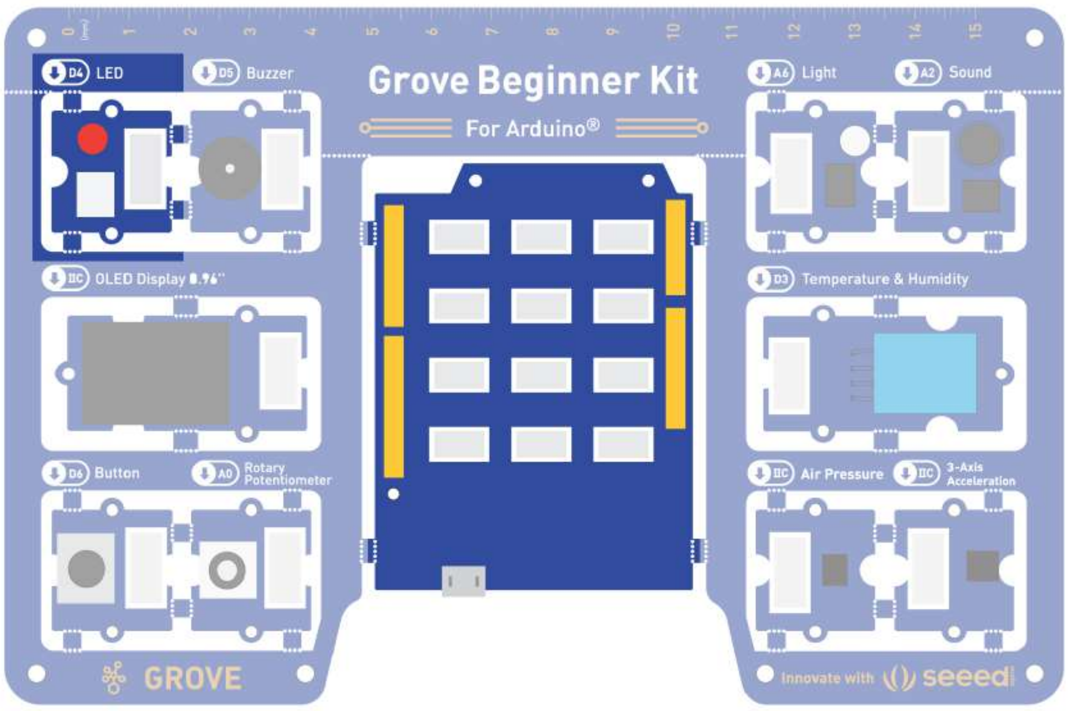
Background Information:

- **What is Digital Signal**

Digital signal: Digital signal refers to the value of the amplitude is discrete, the amplitude is limited to a finite number of values. In our controller, the digital signal has two states: LOW(0V) for 0; HIGH(5V) for 1. So sending a HIGH signal to LED can light it up.



- ### Components Involved
    1. Seeeduino Lotus
    2. Grove LED
    3. Grove Cable(If Broken out)



- ## Hardware connection
  - **Module connection**
  - Default connection by PCB stamp hole.
  - Connect the Seeeduino to the computer through the USB cable.

- ## Software Code
  - Open Arduino IDE.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//LED Blink
//The LED will turn on for one second and then turn off for one second
int ledPin = 4;
void setup() {
    pinMode(ledPin, OUTPUT);
}
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

- ## Code Analysis

```
setup(){
}
```

The `setup()` function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The `setup()` function will only run once, after each powerup or reset of the Arduino board.

```
loop(){
}
```

After creating a `setup()` function, which initializes and sets the initial values, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

```
int ledPin = 4;
```

Description:

Converts a value to the int data type.

Syntax:

int(**x**) or (int)**x** (C-style type conversion)

Parameters:

x: a value. Allowed data types: any type.

Assigned an `int` type 4 to variable named ledPin.

```
pinMode(ledPin, OUTPUT);
```

Description:

Configures the specified pin to behave either as an input or an output. See the Digital Pins page for details on the functionality of the pins.

As of Arduino 1.0.1, it is possible to enable the internal pull-up resistors with the mode `INPUT_PULLUP`. Additionally, the `INPUT` mode explicitly disables the internal pullups.

Syntax:

pinMode(**pin, mode**)

Parameters:

pin: the Arduino pin number to set the mode of.

mode: `INPUT`, `OUTPUT`, or `INPUT_PULLUP`.

Setting ledPin to the output mode.

```
digitalWrite(ledPin, HIGH);
```

Description:

Write a `HIGH` or a `LOW` value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for `HIGH`, 0V (ground) for `LOW`.

If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to `INPUT_PULLUP` to enable the internal pull-up resistor. See the Digital Pins tutorial for more information.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts as a large current-limiting resistor.

Syntax:

digitalWrite(**pin, value**)

Parameters:

pin: the Arduino pin number.

value: `HIGH` or `LOW`.

When we set the ledPin as output, HIGH means sending high level to the pin, LED turns on.

```
digitalWrite(ledPin, LOW);
```

When we set the led as output, low stands for sending low level to the pin, LED turns off.

```
delay(1000);
```

Description:

Pauses the program for the amount of time (in milliseconds) specified as a parameter. (There are 1000 milliseconds in a second.)

Syntax:

delay(**ms**)

Parameters:

ms: the number of milliseconds to pause. Allowed data types: unsigned long.

Delay the program by 1000ms(1s).

Demo Effect and Serial Print Result:

The LED module will be 1 second on and 1 second off.

- ## Breakout Guide

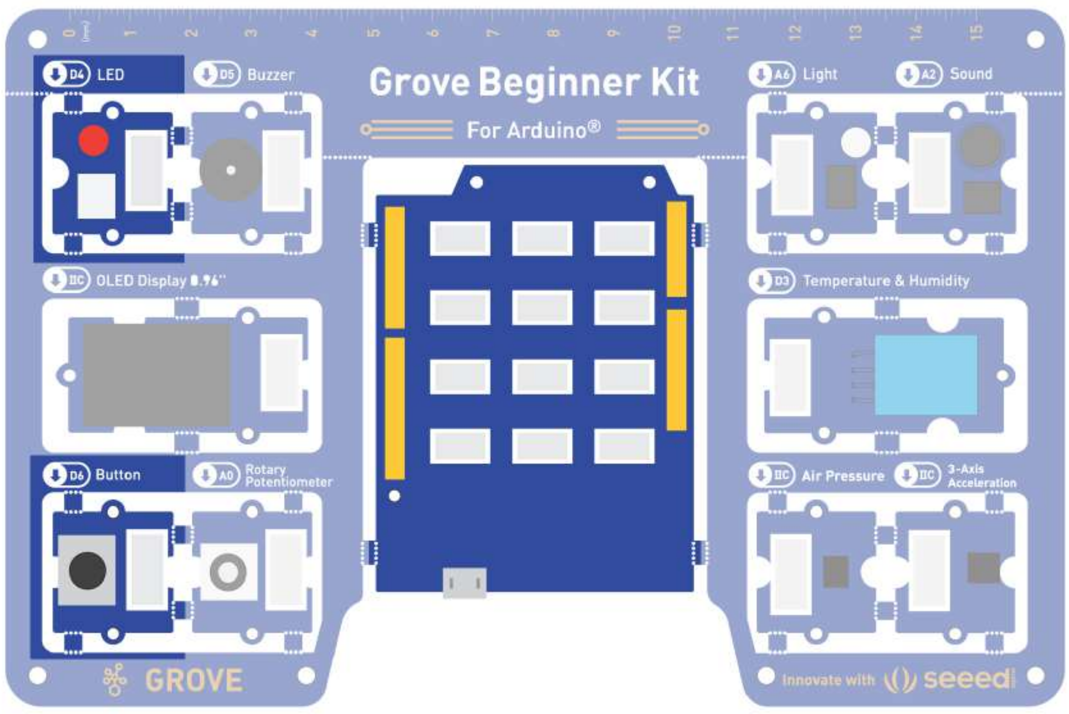
If modules are broken out from the board. Use a Grove cable to connect the **Grove LED** to Seeeduino Lotus's digital interface **D4.**

Lesson 2: Pressing Button to Light Up LED

The first thing we need to know is that the input of the button is a digital signal, and there are only two states, 0 or 1, so we can control the output based on those two states.

- **Practice:** Use button to turn ON and OFF the LED module

- ### Components Involved
  1. Seeeduino Lotus
  2. Grove LED
  3. Grove Button
  4. Grove Cables(If broken out)



- ### Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.
- **Hardware analysis**:
  - Input: Button
  - Control: Seeeduino

- Output: LED module

Both the sensor and the LED use digital signals, so they should be connected to digital interfaces.

- **Software code**:
  - Open Arduino IDE.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//Button to turn ON/OFF LED
//Constants won't change. They're used here to set pin numbers:
const int buttonPin = 6;     // the number of the pushbutton pin
const int ledPin =  4;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton statu
s

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIG
H:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

- ## Code Analysis

```
pinMode(ledPin, OUTPUT);
```

Define LED as the output unit.

```
pinMode(buttonPin, INPUT);
```

Define button as the input unit.

```
buttonState = digitalRead(buttonPin);
```

Description:

Reads the value from a specified digital pin, either `HIGH` or `LOW`.

Syntax:

digitalRead(**pin**)

Parameters:

pin: the Arduino `pin` number you want to read

This function is used to read the states of digital pins, either HIGH or LOW. When the button is pressed, the state is HIGH, otherwise is LOW.

```
    if (buttonState == HIGH) {
      digitalWrite(ledPin, HIGH);
    } else {
      digitalWrite(ledPin, LOW);
    }
  }
```

Description:

The if…else allows greater control over the flow of code than the basic if statement, by allowing multiple tests to be grouped. An else clause (if at all exists) will be executed if the condition in the if statement results in false. The else can proceed another if test, so that multiple, mutually exclusive tests can be run at the same time.

Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default else block is executed, if one is present, and sets the default behaviour.

Note that an else if block may be used with or without a terminating else block and vice versa. An unlimited number of such else if branches are allowed.

Syntax:

```
if (condition1) {
  // do Thing A
}
else if (condition2) {
  // do Thing B
}
else {
  // do Thing C
}
```

The usage of the statement is: if the logical expression in parentheses is true, execute the statement in curly braces after **if**, if not, execute the statement in curly braces after the **else**. If the state of the button is high, the LED pin outputs a high level and turn the LED on, else turn LED off.

Demo Effect and Serial Print Result:

Pressing the button will turn the LED module on.

- ## Breakout Guide

Use a Grove cable to connect the Grove LED to Seeeduino Lotus's digital interface **D4**. Connect the Grove Button to digital interface **D6**.

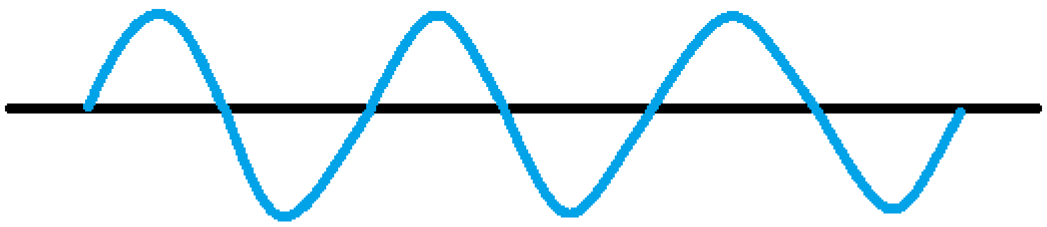
Lesson 3: Controlling the Speed of the Blink

In the last section, we studied that button only has two states, ON/OFF state corresponding 0V and 5V, but in practice, we often counter the need for many states, not just 0V and 5V. Then you need to use Analog Signal! Rotary Potentiometer is a classic example that uses an analog signal.

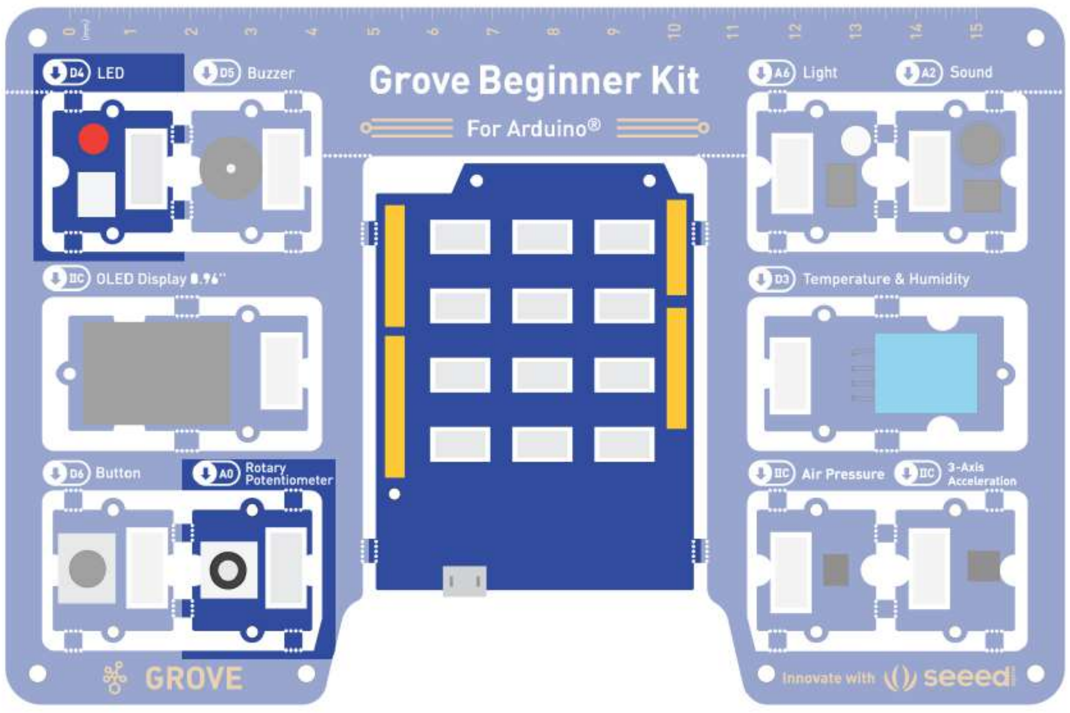
Background Information:

- **What is Analog Signal**

Analog signals: Signals vary continuously in time and value, and the amplitude, frequency, or phase of the signal changes continuously at any time, such as the current broadcast sound signal, or image signal, etc. The analog signal has sine wave and triangle wave and so on. The analog pins of your microcontroller can have between 0V and 5V is mapped to a range between 0 and 1023 where 1023 is mapped as 5V and 512 is mapped as 2.5v and etc.



- ## Components Involved
  1. Seeeduino Lotus
  2. Grove LED
  3. Grove Rotary Switch
  4. Grove Cables(If broken out)



- ## Hardware connection
    - **Module connection:**
    - Default connection by PCB stamp hole.
    - The Seeeduino is then connected to the computer via a USB cable.
- **Hardware analysis:**
    - Input: Rotary Potentiometer
    - Control: Seeeduino Lotus
    - Output: LED module

The input is an analog signal, so it is connected to the analog signal interface, the LED module is connected to the digital signal interface.

- ## Software Code
  - Open Arduino IDE.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//Rotary controls LED
int rotaryPin = A0;    // select the input pin for the rotary
int ledPin = 4;      // select the pin for the LED
int rotaryValue = 0;  // variable to store the value coming from the rota
ry

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  rotaryValue = analogRead(rotaryPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(rotaryValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(rotaryValue);
}
```

Code Analysis

```
rotaryValue = analogRead(rotaryPin);
```

Description:

Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023. On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit.

Syntax:

analogRead(**pin**)

Parameters:

pin: the name of the analog input pin to read from (A0 to A5 on most boards).

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.

This function is used to read the value of Analog pins(the rotary sensor position), the range of values is: 0 ~ 1023.

```
delay(rotaryValue);
```

Delay function, The millisecond duration of the delay is the value in parentheses. Because the value is the value of the analog signal of the knob pin being read, so the delay time can be controlled by the knob.

Demo Effect and Serial Print Result:

Turning the Potentiometer will change the frequency of LED flickering.

Breakout Guide

Use a Grove cable to connect LED to Seeeduino Lotus's digital interface **D4**, and a Grove cable to connect the Grove Rotary Switch to analog signal interface **A0**.

Lesson 4: Making the Buzzer go BEEP

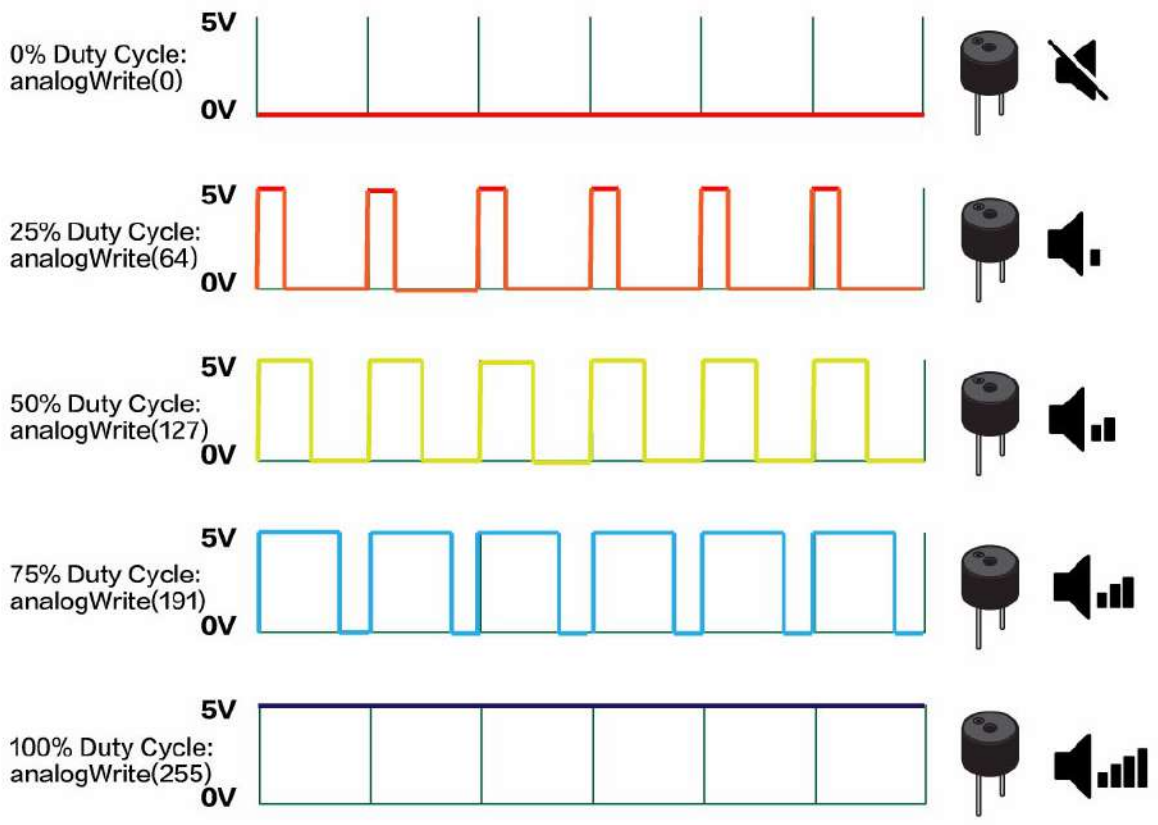
Just like the LED module, Buzzer is also an output module, instead of lighting up it produces a beep sound. This can be used for many situations for indication purposes. We studied the use of potentiometer in the last section, so how do we use the potentiometer to control the volume of the buzzer? This requires the used of PWM control!

Background Information:

- **What is PWM**

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED. *Reference: Arduino*

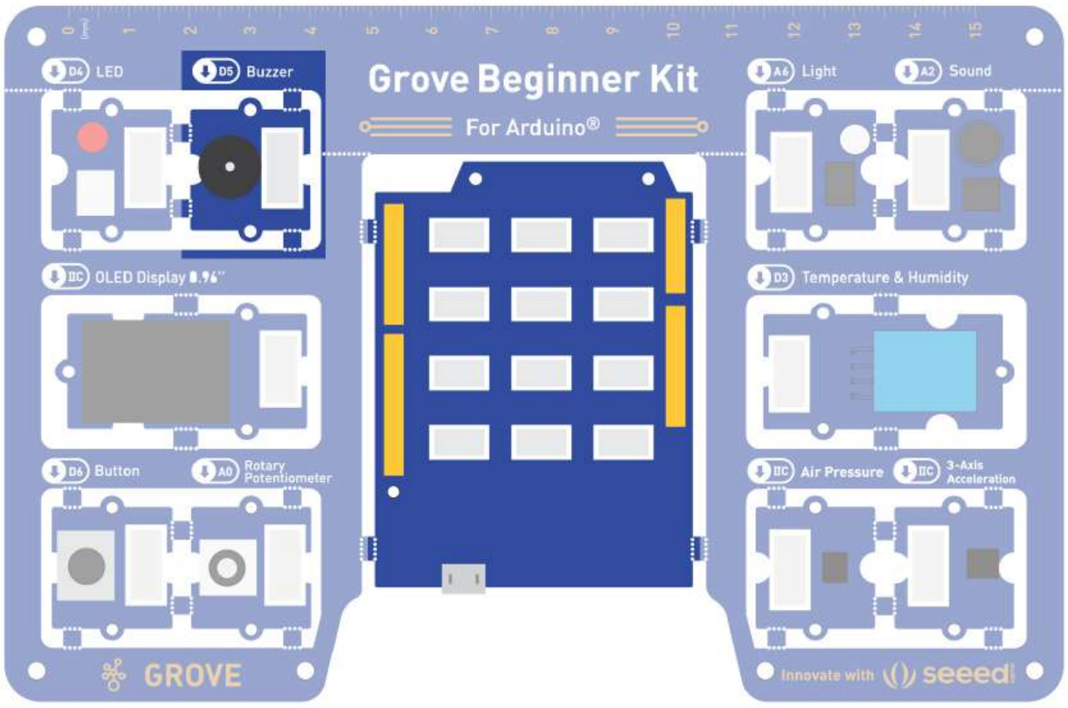
As the diagram indicates below, use `analogWrite()` to generate PWM waves, the higher the percentage of Duty Cycle, the louder the buzzer.



There are six digital pins on your Seeeduino that are marked with the symbol "~", which means they can send out a PWM signal : 3,5,6,9,10,11. They are celled PWM pins.

- ## Components Involved

  1. Seeeduino Lotus
  2. Grove Buzzer
  3. Grove Cable(If Broken out)



- ## Hardware connection

  - **Module connection**
  - Default connection by PCB stamp hole.
  - Connect the Seeeduino to the computer through the USB cable.

- ## Software Code
    - Open Arduino IDE.
    - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
int BuzzerPin = 5;
int Potentiometer = A0;

void setup() {
  pinMode(BuzzerPin, OUTPUT);
}

void loop() {
  int potentioValue, Value;
  potentioValue = analogRead(Potentiometer);
  Value = map(potentioValue, 0, 1023, 0, 255); //Mapping potentiometer va
lue to PWM signal value
  analogWrite(BuzzerPin, Value);
}
```

• Code Analysis

```
analogWrite(BuzzerPin, Value);
```

Description:

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

Syntax:

analogWrite(**pin, value**)

Parameters:

pin: the Arduino `pin` to write to. Allowed data types: int.

value: the duty cycle: between `0` (always off) and `255` (always on). Allowed data types: int.

Writes an analog value (PWM wave) to the Buzzer.

Demo Effect and Serial Print Result:

The buzzer beeps 3 times fast at startup, waits a second then beeps continuously at a slower pace.

• Breakout Guide

Use a Grove cable to connect the Grove Buzzer to Seeeduino Lotus's digital interface **D5**.

Lesson 5: Making an Light Induct LED

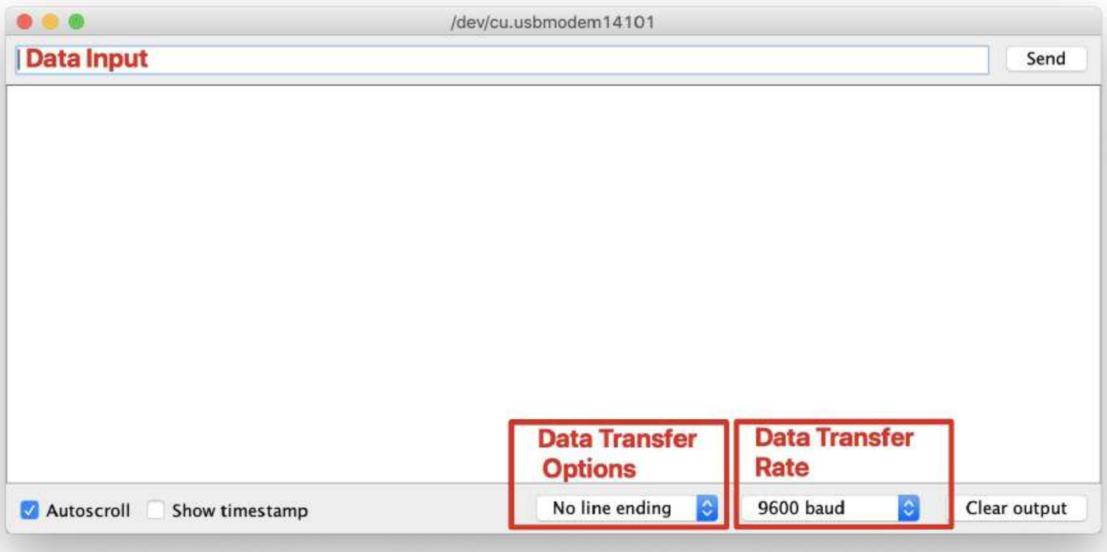
The light sensor contains a photosensitive resistor to measure the intensity of light. The resistance of the photosensitive resistor decreases with the increase of light intensity. The output signal is the analog value, the brighter the light source, the larger the analog value. Based on this property, you can use it to make a light switch.

In the following sections, we will use Serial Monitor to observe results from our sensors so here comes the brief introduction!

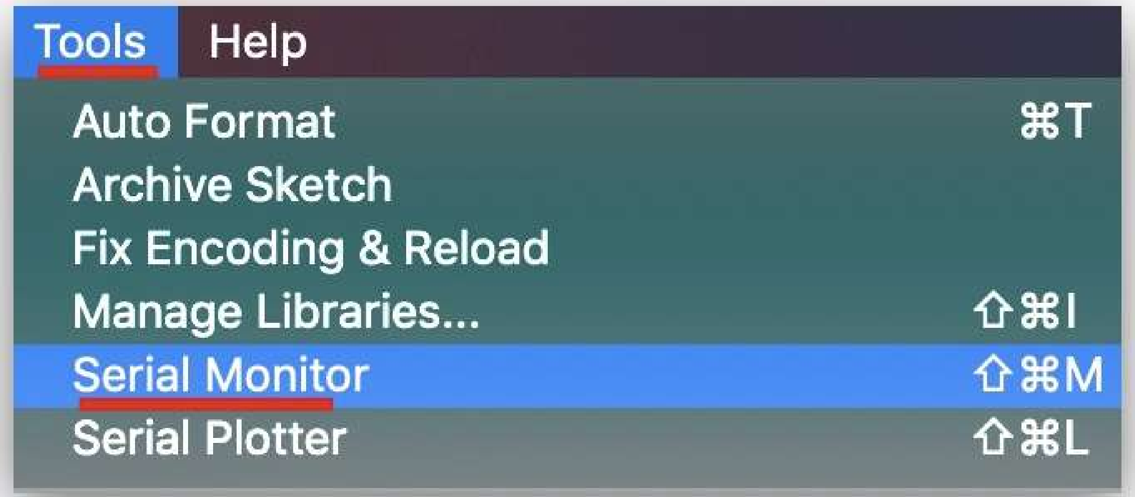
Background Information:

- **What is Serial Monitor**

Serial Monitor is a useful tool to observe results on Arduino, it can be very useful in terms of printing results from the sensors or debugging in general. You can also send data back to the controller via the serial monitor to do certain tasks! Note: Make sure the Serial data transfer match with the code.

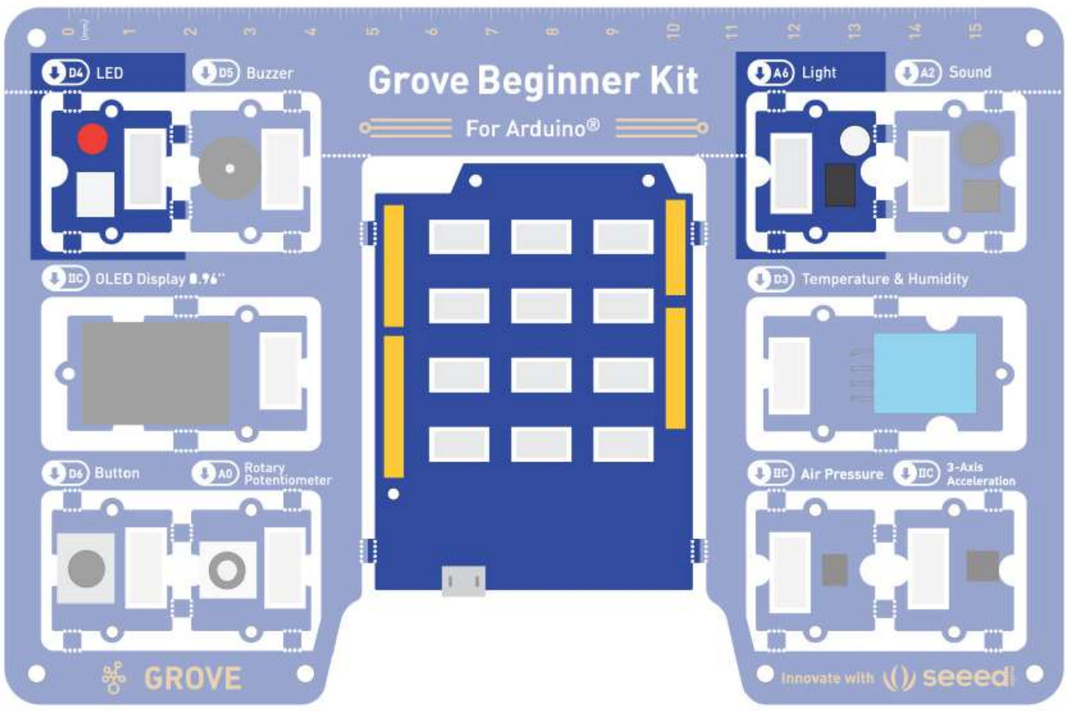


You can open the Serial Plotter by clicking **Tools** -> **Serial Monitor.**



- **Exercise:** As the environment slowly brightens, the LED lights will lighten. As the light slowly dimmed, the LED dimmed. The LED will go from dark to light or from light to dark. To achieve this, we will use pulse width modulation(PWM).

- ## Components Involved
  1. Seeeduino Lotus
  2. Grove LED
  3. Grove Light Sensor
  4. Grove Cable(If broken out)



- ## Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.
- **Hardware analysis:**
  - Input: Light Sensor
  - Control: Seeeduino Lotus
  - Output: LED module

- ## Software Code
    - Open Arduino IDE.
    - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
// Light Switch
int sensorpin = A6; // Analog input pin that the sensor is attached to
int ledPin = 4;    // LED port
int sensorValue = 0;        // value read from the port
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  pinMode(ledPin,OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // read the analog in value:
  sensorValue = analogRead(sensorpin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  Serial.println(sensorValue);
  // change the analog out value:
  analogWrite(ledPin, outputValue);
  delay(30);
}
```

You can also see the light intensity readings from the **Serial Monitor**, navigate to **Tools** -> **Serial Monitor**.

• Code Analysis

```
Serial.begin(9600);
```

Description:

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with Serial Monitor, make sure to use one of the baud rates listed in the menu at the bottom right corner of its screen. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.

The software running on the computer communicates with the development board, and the baud rate is 9600.

Syntax:

Serial.begin(**speed**)

Parameters:

speed: Speed of Serial communication. i.e `9600` , `115200` and etc.

Set the Serial baud rate to 9600.

```
Serial.println(sensorValue);
```

Description:

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or ‘\r’) and a newline character (ASCII 10, or ‘\n’). This command takes the same forms as Serial.print().

Syntax:

Serial.println(**val**) or Serial.println(**val**, **format**)

Parameters:

val: the value to print. Allowed data types: any data type.

format: specifies the number base (for integral data types) or the number of decimal places (for floating point types).

Serial port print the Light sensor’s value. So you open the **serial monitor** on the IED interface, and you see the value of the output sensor.

```
outputValue = map(sensorValue, 0, 1023, 0, 255);
```

Description:

Re-maps a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The `constrain()` function may be used either before or after this function, if limits to the ranges are desired.

Note that the "lower bounds" of either range may be larger or smaller than the "upper bounds" so the `map()` function may be used to reverse a range of numbers, for example

y = map(x, 1, 50, 50, 1);

The function also handles negative numbers well, so that this example

y = map(x, 1, 50, 50, -100);

is also valid and works well.

The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated and are not rounded or averaged.

Syntax:

map(**value, fromLow, fromHigh, toLow, toHigh**)

Parameters:

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.

Mapping light sensor analog signal(0 to 1023)to the brightness value of LED(0 to 255).

Keep equal and potentiometer after mapping the value of the time. The Map has five parameters, which in turn is: to map the original value, the original value of the minimum value, original value maximum, minimum value after the mapping, mapping the maximum. In this way, the data returned by the sensor can be mapped from its original value of 0-1023 to 0-255.

```
analogWrite(ledPin,255-outputvalue);
```

The function is used to write an analog value between 0 - 255 a PWM pin. analogWrite() can only be used for PWM pins. The new mapping data in the previous statement can be output to ledPin to lighten/dim the LED.

Demo Effect and Serial Print Result:

The LED module will change its intensity according to the light intensity of the surrounding. The darker the surrounding, the lighter it gets.

- ## Breakout Guide

Use Grove Cable to connect the Grove LED to Seeeduino Lotus's digital signal interface **D4**,connect the Grove Light Sensor to Seeeduino Lotus's analog signal interface **A1**.

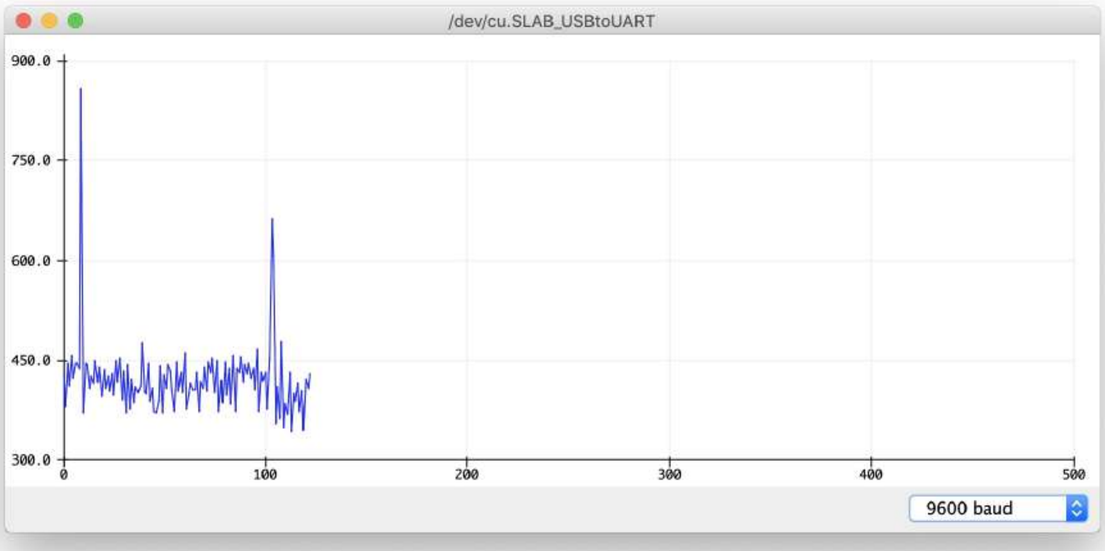
Lesson 6: Sound Sensitive LED Light

The sound sensor can detect the sound intensity of the environment, and its output is also simulated. I'm sure you've all been exposed to the sound control lights, but now we can do one ourselves, and with the basics, this experiment will be easy for you. Here used Serial Plotter to visualize results.

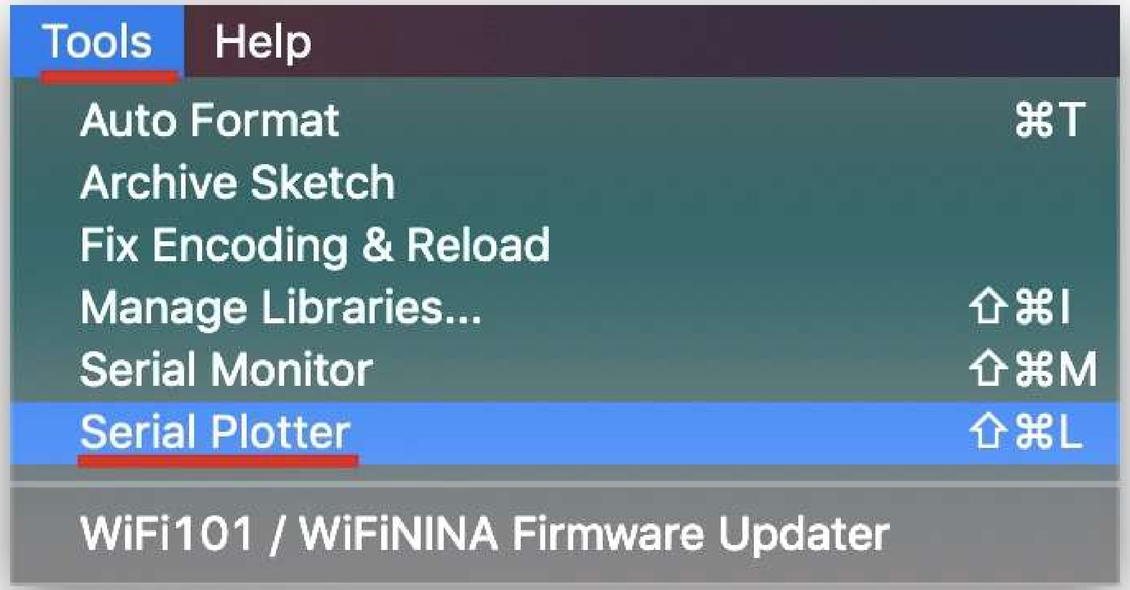
Background Information:

- **What is Serial Plotter**

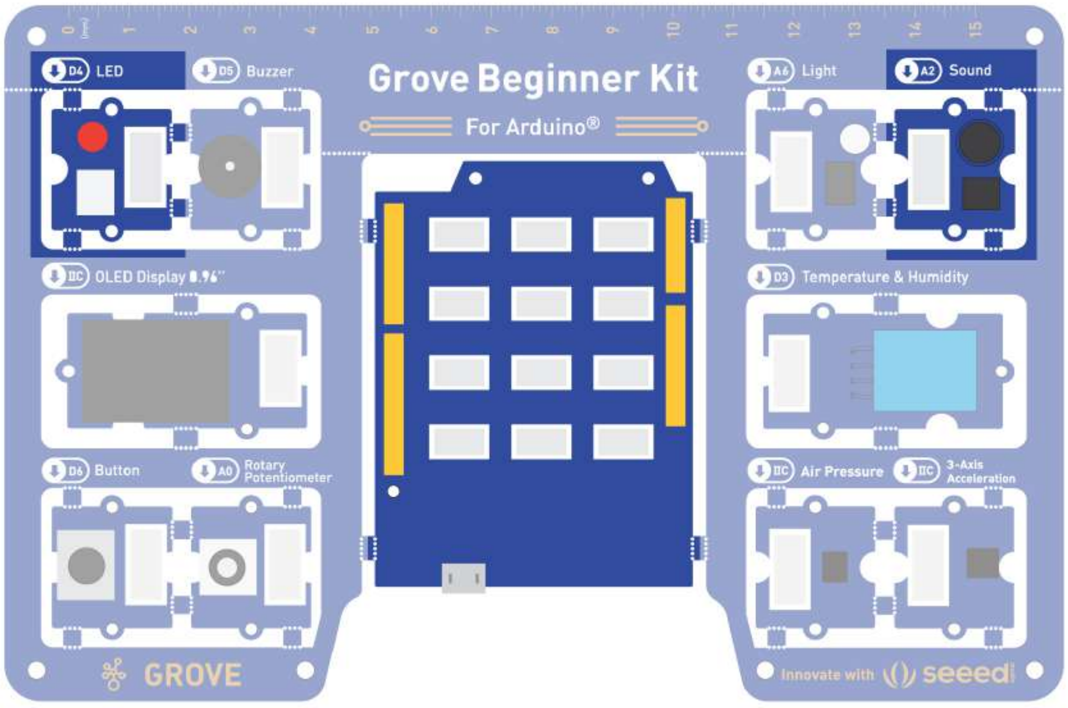
Serial Plotter is similar to Serial Monitor, allowing you to natively graph serial data from your Arduino to your computer in real-time. This is very useful when data needs to be visualized.



You can open the Serial Plotter by clicking **Tools** -> **Serial Plotter**.



- **Practice:** The LED lights light up when the sound is made. When there is no sound and it is very quiet, the LED lights go off.
- ## Components Involved
    1. Seeeduino Lotus
    2. Grove LED
    3. Grove Sound Sensor
    4. Grove cable(If broken out)



- ## Hardware connection
    - **Module connection:**
    - Default connection by PCB stamp hole.
    - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
  - Open Arduino IDE.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//Sound Control Light
int soundPin = A2; // Analog sound sensor is to be attached to analog
int ledPin = 4; // Digital LED is to be attached to digital
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop(){
  int soundState = analogRead(soundPin); // Read sound sensor’s value
  Serial.println(soundState);
  // if the sound sensor’s value is greater than 200, the light will be o
n for 5 seconds.
  //Otherwise, the light will be turned off
  if (soundState > 200) {
    digitalWrite(ledPin, HIGH);
    delay(100);
  }else{
    digitalWrite(ledPin, LOW);
  }
}
```

You can also see the light intensity readings from the **Serial Monitor**, navigate to **Tools** -> **Serial Plotter**.

Code Analysis

```
Serial.begin(9600);
```

The software running on the computer communicates with the development board, and the baud rate is 9600.

```
Serial.print(" ");
```

This function is used to output data from the serial port, the output is what is contained in the double quotation marks.

```
Serial.println( );
```

This statement is similar to the one above, except that **serial.println** has a newline return.

```
  Serial.println(soundState);
```

Serial port print the sound sensor's value. So you open the **serial monitor** on the IED interface, and you see the value of the output sensor.

Demo Effect and Serial Print Result:

The LED module will light up if the surrounding is loud enough.

- ## Breakout Guide

Use Grove cables to connect the Grove LED to Seeeduino Lotus's digital signal interface **D4**, Connect the Grove Sound Sensor to Seeeduino Lotus's analog signal interface **A2.**

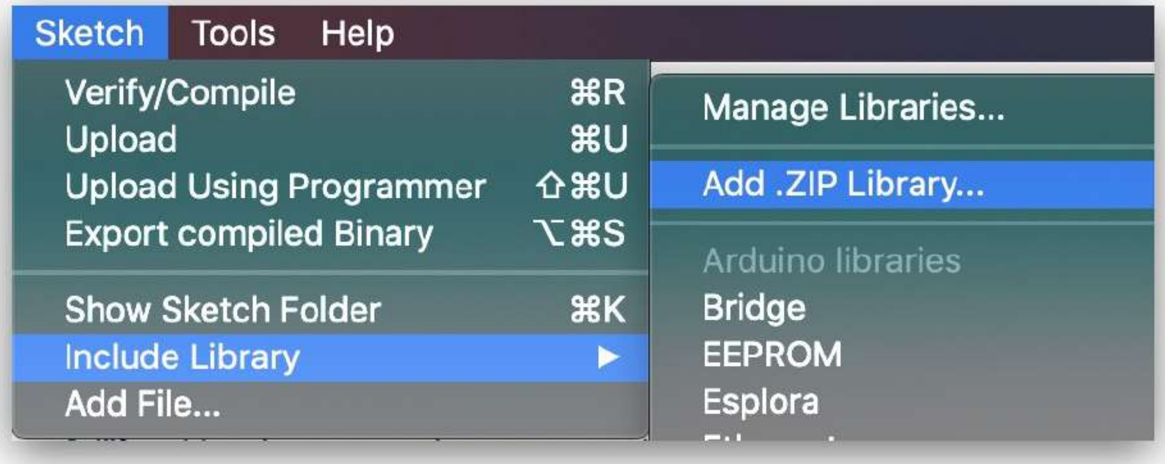
Lesson 7: Displaying Data on OLED

OLED Display can be used for many situations, where you could use it to visualize sensor readings!

Background Information:

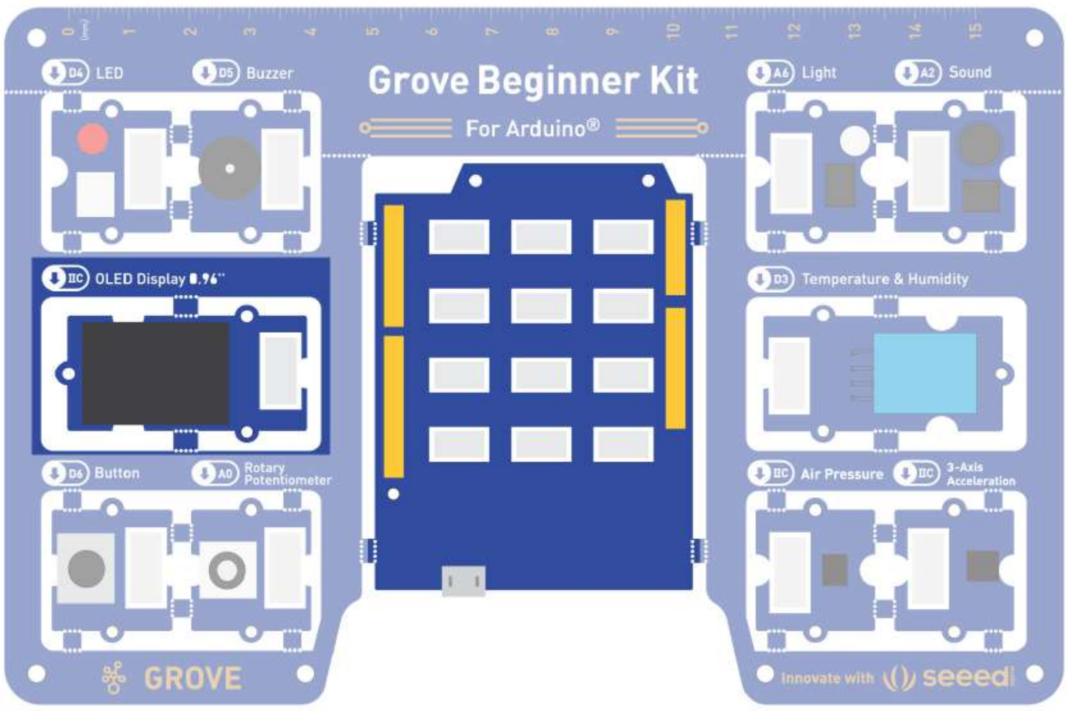
- **What is Arduino Libraries**

The Arduino environment can be extended through the use of libraries, just like most other programming platforms. Libraries provide extra functionalities for use in sketches, i.e. working with specific hardware or manipulating data. To use a library in a sketch, select it from **Sketch** ->**Include Library**.



For more information, please also visit How to install Arduino Libraries.

- ## Components Involved
  1. Seeeduino Lotus
  2. Grove OLED
  3. Grove cable(If broken out)



- ## Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
    - Open Arduino IDE.
    - Install the **U8g2 library**: Navigate to **Sketch** -> **Include Library** -> **Manage Libraries…** and Search for the keyword “**U8g2**” in the **Library Manager**, then install.
    - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
#include <Arduino.h>
#include <U8x8lib.h>

U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);

void setup(void) {
  u8x8.begin();
  u8x8.setFlipMode(1);
}

void loop(void) {
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.setCursor(0, 0);
  u8x8.print("Hello World!");
}
```

- **Code analysis**

```
#include <>
```

Description:

`#include` is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

Note that `#include`, similar to `#define`, has no semicolon terminator, and the compiler will yield cryptic error messages if you add one.

#include is an instruction that introduces a header file. Here we use the DHT.h, <Arduino.h>, <U8g2lib.h>, <SPI.h>, <Wire.h> library, these library are included in Arduino IDE.

```
#define
```

Description:

`#define` is a useful C++ component that allows the programmer to give a name to a constant value before the program is compiled. Defined constants in Arduino don't take up any program memory space on the chip. The compiler will replace references to these constants with the defined value at compile time.

Force a variable to be the value you want.

```
U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);
```

Once the object is declared, you can use functions from the library.

```
u8x8.begin();
```

Description:

Simplified setup procedure of the display for the Arduino environment. See the setup guide for the selection of a suitable U8g2 constructor.

Syntax:

u8x8.begin()

Initialize the u8g2 library

```
u8x8.setFlipMode(1);
```

Description:

Some displays support a 180-degree rotation of the internal frame buffer. This hardware feature can be controlled with this procedure. Important: Redraw the complete display after changing the flip mode. Best is to clear the display first, then change the flip mode and finally redraw the content. Results will be undefined for any existing content on the screen.

Syntax:

u8x8.setFlipMode(**mode**)

Parameters:

mode: `0` or `1`

Flips the display 180 degrees.

```
u8x8.setCursor();
```

Description:

Define the cursor for the print function. Any output of the print function will start at this position.

Syntax:

u8x8.setCursor(**x, y**)

Parameters:

x, y: Column/row position for the cursor of the print function.

Sets the draw cursor position.

```
u8x8.setFont()
```

Description:

Define a u8x8 font for the glyph and string drawing functions.

Syntax:

u8x8.setFont(***font_8x8**)

Set the font for display.

```
u8x8.print();
```

Draw the content on the OLED.

Demo Effect and Serial Print Result:

Prints Hello World onto the OLED Display.

- ## Breakout Guide

Use Grove cable to connect the OLED to Seeeduino Lotus's **I2C** interface (Note: I2C's default address is 0x78).

Lesson 8: Detecting Surrounding Temperature & Humidity

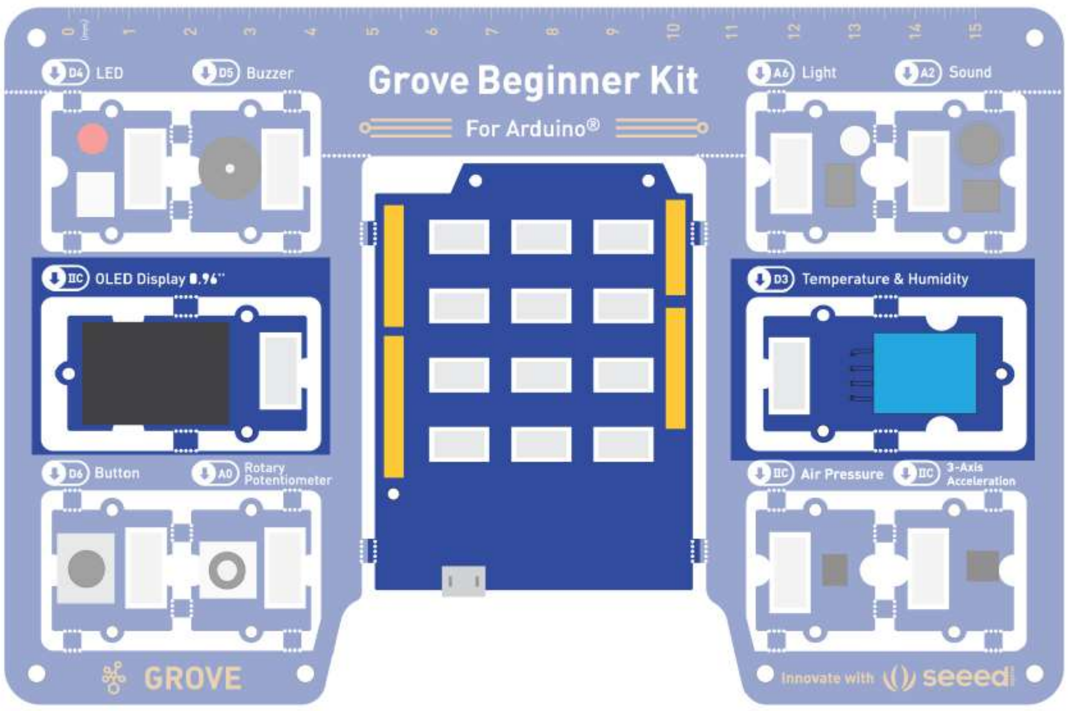
Have you ever wondered about the temperature and humidity of your surroundings? Want to know the exact number? Want to wear a skirt or coat today depending on the temperature? Let's make a temperature meter!

Background Information:

- **What is Protocol Signal (I2C)**

Protocol signal: the protocol signal we use is I2C, so here is a brief introduction to I2C. I2C bus just needs two wires in the transmission of information connection between the devices: the SDA (Serial Data Line) and SCL (Serial Clock Line). These two lines are bidirectional I/O lines, the main component used to start the bus transfer data, and generate the clock to open transmission device, any devices that are addressing at this time is considered from the device. The relationship between master and slave, sender and receiver on the bus is not constant but depends on the direction of data transmission. If the host wants to send data to the slave device, the host first addresses the slave device, then actively sends data to the slave device, and finally terminates the data transmission by the host. If the host is to receive data from the slave, the slave is first addressed by the master. The host then receives the data sent from the device, and the host terminates the receiving process. In this case. The host is responsible for generating the timing clock and terminating the data transfer.

- **Practice:** Let your OLED Display display the current ambient temperature and humidity.
- ## Components Involved
  1. Seeeduino Lotus
  2. Grove OLED
  3. Grove Temperature and Temperature Sensor
  4. Grove cable(If broken out)



- ## Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
    - Open Arduino IDE.
    - Install the **Grove Temperature and Humidity Sensor(DHT11) library**: Navigate to **Sketch** -> **Include Library** -> **Manage Libraries…** and Search for the keyword "**Grove Temperature and Humidity Sensor(DHT11)**" in the **Library Manager**, then install.
    - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//Temperature and Humidity Sensor
//Temperature and Humidity Sensor
#include "DHT.h"
#include <Arduino.h>
#include <U8x8lib.h>

#define DHTPIN 3     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);

U8X8_SSD1306_128X64_ALT0_HW_I2C u8x8(/* reset=*/ U8X8_PIN_NONE);

void setup(void) {
  Serial.begin(9600);
  Serial.println("DHTxx test!");
  dht.begin();
  u8x8.begin();
  u8x8.setPowerSave(0);
  u8x8.setFlipMode(1);
}

void loop(void) {

  float temp, humi;
  temp = dht.readTemperature();
  humi = dht.readHumidity();

  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.setCursor(0, 33);
  u8x8.print("Temp:");
  u8x8.print(temp);
  u8x8.print("C");
  u8x8.setCursor(0,50);
  u8x8.print("Humidity:");
  u8x8.print(humi);
  u8x8.print("%");
  u8x8.refreshDisplay();
  delay(200);
}
```

- ## Code Analysis

```
float temp, humi;
```

Defines variables to store readings.

```
temp = dht.readTemperature();
humi = dht.readHumidity();
```

Description:

Functions to be used to read temperature and humidity values from the sensor.

Syntax:

dht.readTemperature() and **dht.readHumidity()**. Return type: float.

Call these functions to read the temperature and humidity and store them in defined variables.

Demo Effect and Serial Print Result:

The surrounding temperature and humidity appear on the OLED screen.

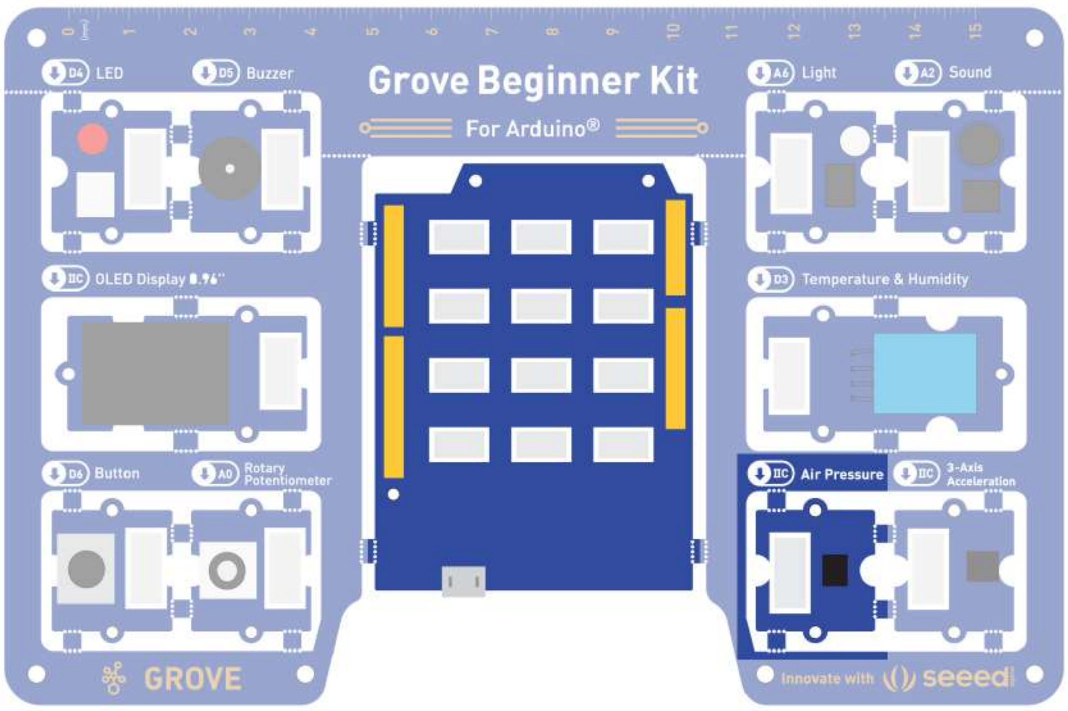
- ## Breakout Guide

Use Grove cable to connect the OLED to Seeeduino Lotus's **I2C** interface (Note: I2C's default address is 0x78). Connect the Grove Temperature and Humidity Sensor to Seeeduino Lotus's digital signal interface **D3**.

Lesson 9: Measuring Surrounding Air Pressure

Grove Air Pressure Sensor(BMP280) is a breakout board for Bosch BMP280 high-precision and low-power digital barometer. This module can be used to measure temperature and atmospheric pressure accurately. As the atmospheric pressure changes with altitude, it can also measure the approximate altitude of a place.

- ## Components Involved
  1. Seeeduino Lotus
  2. Grove Air Pressure Sensor
  3. Grove cable(if broken out)



- ## Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
  - Open Arduino IDE.
  - Install the **Grove Barometer Sensor library**: Navigate to **Sketch** -> **Include Library** -> **Manage Libraries...** and Search for the keyword “**Grove BMP280**” in the **Library Manager**, then install.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.
  - In this program, acceleration information is sent from the sensor to Seeeduino via I2C bus and then Seeeduino printed them onto the serial monitor. Open the **serial monitor** to check the result.

```
//Air pressure detection
#include "Seeed_BMP280.h"
#include <Wire.h>

BMP280 bmp280;

void setup() {
    Serial.begin(9600);
    if (!bmp280.init()) {
        Serial.println("Device not connected or broken!");
    }
}

void loop() {

    float pressure;

    //get and print temperatures
    Serial.print("Temp: ");
    Serial.print(bmp280.getTemperature());
    Serial.println("C"); // The unit for  Celsius because original arduin
o don't support speical symbols

    //get and print atmospheric pressure data
    Serial.print("Pressure: ");
    Serial.print(pressure = bmp280.getPressure());
    Serial.println("Pa");

    //get and print altitude data
    Serial.print("Altitude: ");
    Serial.print(bmp280.calcAltitude(pressure));
    Serial.println("m");

    Serial.println("\n");//add a line between output of different times.

    delay(1000);
}
```

Code Analysis

```
#include <Wire.h>
```

#include is an instruction that introduces a header file. Here we use the <Wire.h> library, this library is included in Arduino IDE.

```
#include "Seeed_BMP280.h"
```

Represents the Seeed_BMP280.h header file that introduces the current path.

```
if (!bmp280.init()) {
    Serial.println("Device not connected or broken!");
}
```

Description:

Initialize the air pressure sensor.

Syntax:

bmp280.init()

if the Air pressure sensor did not start properly, then prints out an error to the serial monitor.

```
Serial.print(bmp280.getTemperature());
```

Description:

Functions to be used to read temperature value from the sensor.

Syntax:

bmp280.getTemperature(). Return type: float

Prints the temperature data to the serial monitor.

```
Serial.print(pressure = bmp280.getPressure());
```

Description:

Functions to be used to read air pressure value from the sensor.

Syntax:

bmp280.getPressure(). Return type: float

Prints the current air pressure.

```
Serial.print(bmp280.calcAltitude(pressure));
```

Description:

Takes the pressure value can convert to altitude.

Syntax:

bmp280.calcAltitude(**float**). Return type: float

Parameter:

float: Pressure value.

Prints the amplitude.

Demo Effect and Serial Print Result:

The Air pressure readings are display on the Serial Monitor.

- ## Breakout Guide

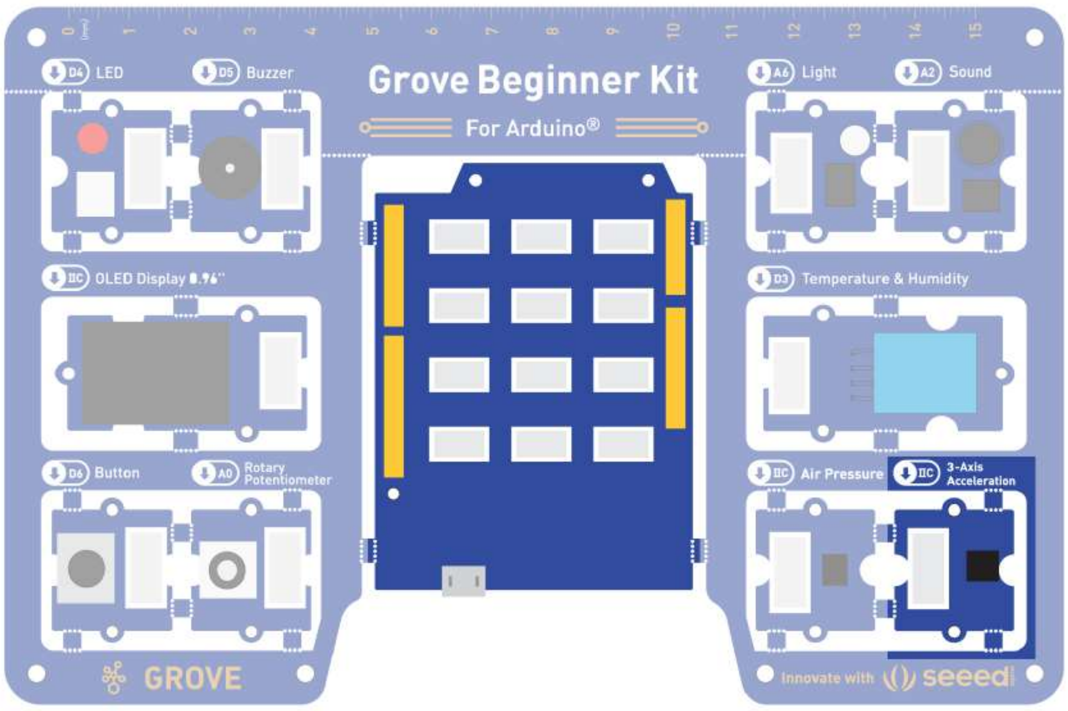
Use Grove cable to connect Grove 3-axis Accelerometer to Seeeduino Lotus's **I2C** interface using a Grove cable (note: I2C default address is 0x77 or 0x76).

Lesson 10: Sensing Movement

This is the last sensor, the triaxial accelerometer, and with this module, you can easily add motion monitoring to your design. So we can do a lot of interesting little experiments on the basis of the motion.

- **Practice:** when motion is detected, the buzzer gives an alarm indicating that the object is in motion.

- ### Components Involved
  1. Seeeduino Lotus
  2. Grove 3-axis Accelerometer
  3. Grove cable(if broken out)



- ### Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
  - Open Arduino IDE.
  - Download the 3-Axis Digital Accelerometer(±2g to 16g) from Github. Click on **Sketch** > **Include library** > **Add .ZIP library**, import the library into the IED.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.
  - In this program, acceleration information is sent from the sensor to Seeeduino via I2C bus and then Seeeduino printed them onto the serial monitor. Open the **serial monitor** to check the result.

```
//Gravity Acceleration
#include "LIS3DHTR.h"
#ifdef SOFTWAREWIRE
    #include <SoftwareWire.h>
    SoftwareWire myWire(3, 2);
    LIS3DHTR<SoftwareWire> LIS(I2C_MODE);//IIC
    #define WIRE myWire
#else
    #include <Wire.h>
    LIS3DHTR<TwoWire> LIS(I2C_MODE);//IIC
    #define WIRE Wire
#endif

void setup() {
    Serial.begin(9600);
    while (!Serial) {};
    LIS.begin(WIRE); //IIC init
    delay(100);
    LIS.setOutputDataRate(LIS3DHTR_DATARATE_50HZ);
}
void loop() {
    if (!LIS) {
        Serial.println("LIS3DHTR didn't connect.");
        while (1);
        return;
    }
    //3 axis
    Serial.print("x:"); Serial.print(LIS.getAccelerationX()); Serial.prin
t("  ");
    Serial.print("y:"); Serial.print(LIS.getAccelerationY()); Serial.prin
t("  ");
    Serial.print("z:"); Serial.println(LIS.getAccelerationZ());

    delay(500);
}
```

- ## Code Analysis

```
#include "LIS3DHTR.h"
#ifdef SOFTWAREWIRE
    #include <SoftwareWire.h>
    SoftwareWire myWire(3, 2);
    LIS3DHTR<SoftwareWire> LIS(I2C_MODE);//IIC
    #define WIRE myWire
#else
    #include <Wire.h>
    LIS3DHTR<TwoWire> LIS(I2C_MODE);//IIC
    #define WIRE Wire
#endif
```

Initializing the module using software I2C or hardware I2C.

```
while (!Serial) {};
```

Code stops here if don’t open the serial monitor, so open serial monitor.

```
LIS.begin(WIRE);
LIS.setOutputDataRate(LIS3DHTR_DATARATE_50HZ);
```

Description: Initialize the accelerator.

Syntax: `LIS.begin(Wire)`.

Description: Sets the output data rate of the accelerator.

Syntax: `LIS.setOutputDataRate(odr_type_t odr)`.

Initialize the accelerator and set the output rate to 50Hz.

```
Serial.print("x:"); Serial.print(LIS.getAccelerationX()); Serial.print("
 ");
Serial.print("y:"); Serial.print(LIS.getAccelerationY()); Serial.print("
 ");
Serial.print("z:"); Serial.println(LIS.getAccelerationZ());
```

Description:

Functions to be used to read X-axis value from the sensor.

Syntax:

LIS.getAccelerationX(). Return type: float.

Description:

Functions to be used to read Y-axis value from the sensor.

Syntax:

LIS.getAccelerationY(). Return type: float.

Description:

Functions to be used to read Z-axis value from the sensor.

Syntax:

LIS.getAccelerationZ(). Return type: float.

Prints the 3 axis data to the serial monitor.

Demo Effect and Serial Print Result:

The 3-axis accelerator readings are displayed on the Serial Monitor.

- ## Breakout Guide

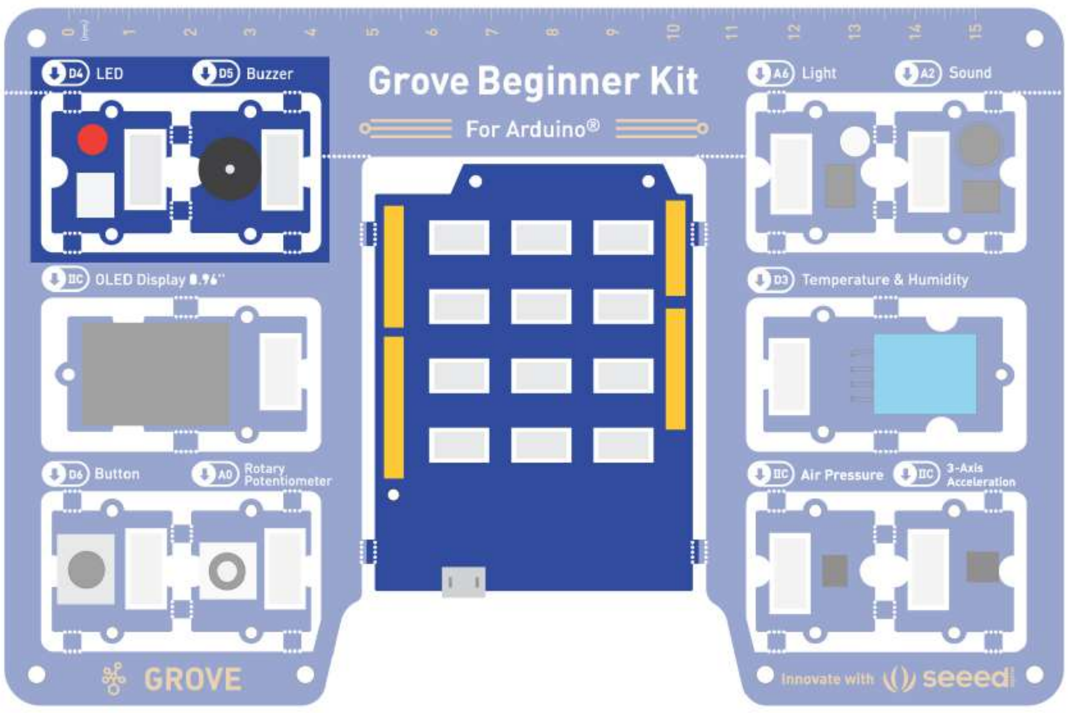
Use Grove cable to connect Grove 3-axis Accelerometer to Seeeduino Lotus's **I2C** interface using a Grove cable (note: I2C default address is 0x4c).

Bonus Projects

Project 1: Music dynamic rhythm lamp

- **Project description:** In this experiment, we will make the buzzer play pleasant music and the led lights flash according to the music frequency and beat.

- ### Components Involved
  1. Seeeduino Lotus
  2. Grove LED
  3. Buzzer
  4. Grove Cables(if broken out)



- ### Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
  - Open Arduino IDE.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//Music Dynamic Rhythm Lamp
#define NTD0 -1
#define NTD1 294
#define NTD2 330
#define NTD3 350
#define NTD4 393
#define NTD5 441
#define NTD6 495
#define NTD7 556

#define NTDL1 147
#define NTDL2 165
#define NTDL3 175
#define NTDL4 196
#define NTDL5 221
#define NTDL6 248
#define NTDL7 278

#define NTDH1 589
#define NTDH2 661
#define NTDH3 700
#define NTDH4 786
#define NTDH5 882
#define NTDH6 990
#define NTDH7 112

#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.25
#define SIXTEENTH 0.625

int tune[]=
{
NTD3,NTD3,NTD4,NTD5,
NTD5,NTD4,NTD3,NTD2,
NTD1,NTD1,NTD2,NTD3,
NTD3,NTD2,NTD2,
NTD3,NTD3,NTD4,NTD5,
NTD5,NTD4,NTD3,NTD2,
NTD1,NTD1,NTD2,NTD3,
NTD2,NTD1,NTD1,
NTD2,NTD2,NTD3,NTD1,
NTD2,NTD3,NTD4,NTD3,NTD1,
NTD2,NTD3,NTD4,NTD3,NTD2,
NTD1,NTD2,NTDL5,NTD0,
NTD3,NTD3,NTD4,NTD5,
NTD5,NTD4,NTD3,NTD4,NTD2,
NTD1,NTD1,NTD2,NTD3,
NTD2,NTD1,NTD1
};

float durt[]=
{
1,1,1,1,
1,1,1,1,
1,1,1,1,
1+0.5,0.5,1+1,
1,1,1,1,
1,1,1,1,
1,1,1,1,
1+0.5,0.5,1+1,
1,1,1,1,
1,0.5,0.5,1,1,
1,0.5,0.5,1,1,
1,1,1,1,
1,1,1,1,
1,1,1,0.5,0.5,
1,1,1,1,
1+0.5,0.5,1+1,
};

int length;
int tonepin=5;
int ledp=4;

void setup()
{
  pinMode(tonepin,OUTPUT);
  pinMode(ledp,OUTPUT);
  length=sizeof(tune)/sizeof(tune[0]);
}

void loop()
{
  for(int x=0;x<length;x++)
  {
    tone(tonepin,tune[x]);
    digitalWrite(ledp, HIGH);
    delay(400*durt[x]);
    digitalWrite(ledp, LOW);
    delay(100*durt[x]);
    noTone(tonepin);

  }
  delay(4000);
}
```

• Code Analysis

```
#define NTD
```

Here is the definition of the frequency of the D key, which is divided into bass, alto, and treble.

```
#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.25
#define SIXTEENTH 0.625
```

Note: rhythm is divided into one beat, half beat, 1/4 beat, 1/8 beat, we specify a beat note time is 1;Half beat is 0.5;1/4 beat is 0.25;1/8 of 0.125.

```
int tune[]=...
```

List the frequencies according to the spectrum.

```
float durt[]=...
```

List the beats according to the spectrum.

```
delay(100*durt[x]);
```

Control LED lights on and off respectively.

Demo Effect and Serial Print Result:

The buzzer will beep a tune while the LED module will flicker with same frequency.

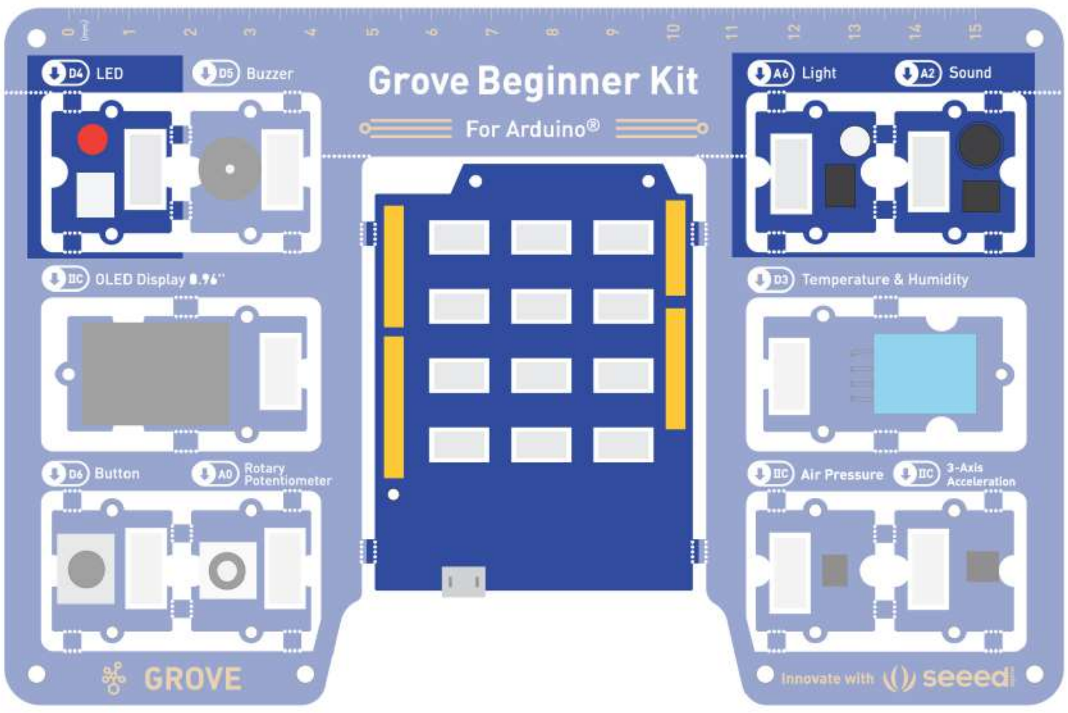
• Breakout Guide

Connect Grove LED to Seeeduino Lotus's digital signal interface **D4**, connect Buzzer to Seeeduino Lotus's digital signal interface **D5**.

Project 2: Make an intelligent sound-light induction desk lamp

- **Project description:** as the name implies, this project is to make a small lamp controlled by Sound and Light. We need to use the LED module. Of course, Light Sensor and Sound Sensor are also indispensable. In this way, you can achieve the function of the smart desk lamp: when the sound, the lamp will light up; If the environment turns dark, the lamp will automatically turn brighter.

- ### Components Involved
  1. Seeeduino Lotus
  2. Grove LED
  3. Light Sensor
  4. Sound Sensor
  5. Grove cable(If broken out)



- ### Hardware connection
  - **Module connection:**
  - Default connection by PCB stamp hole.
  - The Seeeduino is then connected to the computer via a USB cable.

- ## Software Code
  - Open Arduino IDE.
  - Copy the following code, click Verify to check for syntax errors. Verify that there are no errors, and you can upload the code.

```
//light Induction Desk Lamp
int soundPin = A2; // Analog sound sensor is to be attached to analog
int lightPin = A6; //Analog light sensor is to be attached to analog
int ledPin = 4; // Digital LED is to be attached to digital

void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop(){
  int soundState = analogRead(soundPin); // Read sound sensor’s value
  int lightState = analogRead(lightPin); // Read light sensor’s value
  // if the sound sensor's value is greater than 50 or the sound sensor's
 is less than 10, the light will be on.
  //Otherwise, the light will be turned off
if (soundState > 50 || lightState < 10) {
  digitalWrite(ledPin, HIGH);
  //delay(5000); //You can delete the "//" to make the LED on for five se
conds
}else{
  digitalWrite(ledPin, LOW);
}
}
```

- ## Code Analysis

```
if (soundState > 50 || lightState < 10) {
  ...
}
```

In parentheses is a logical expression. Both **&&** and **||** are commonly used in logical expressions. The common usage is **if (expression 1 || expression 2)** and **if (expression 1 && expression 2)**.

|| represents “**or**“, satisfies one of them, the whole expression is true, and satisfies the condition of the if judgment.

&& means “**and**“, the statement in if{} is executed only if all expressions in parentheses are true.

Demo Effect and Serial Print Result:

If the surrounding sound is loud enough or light intensity is low, the LED module will light up more intensity.

- ## Breakout Guide

Connect the Grove LED to Seeeduino Lotus’s digital signal interface **D4**, Connect the Light Sensor to Seeeduino Lotus’s analog signal interface **A1**. Connect the Sound Sensor to Seeeduino Lotus’s analog signal interface **A2** using a Grove cable.

Make Your Own Modules & Boards

After this period of study, you already have a systematic understanding of Arduino and open-source hardware, so why not go further and try to make your own module or development board?

EDA

To design your own board, you will need to design your own module's schematics, which requires an EDA tool to do so. Here recommends an open-source EDA software.

- **KiCAD**

KiCad is a free software suite for electronic design automation. It facilitates the design of schematics for electronic circuits and their conversion to PCB designs. It features an integrated environment for schematic capture and PCB layout design. The programs handle Schematic Capture and PCB Layout with Gerber output. The suite runs on Windows, Linux, and macOS and is licensed under GNU GPL v3.

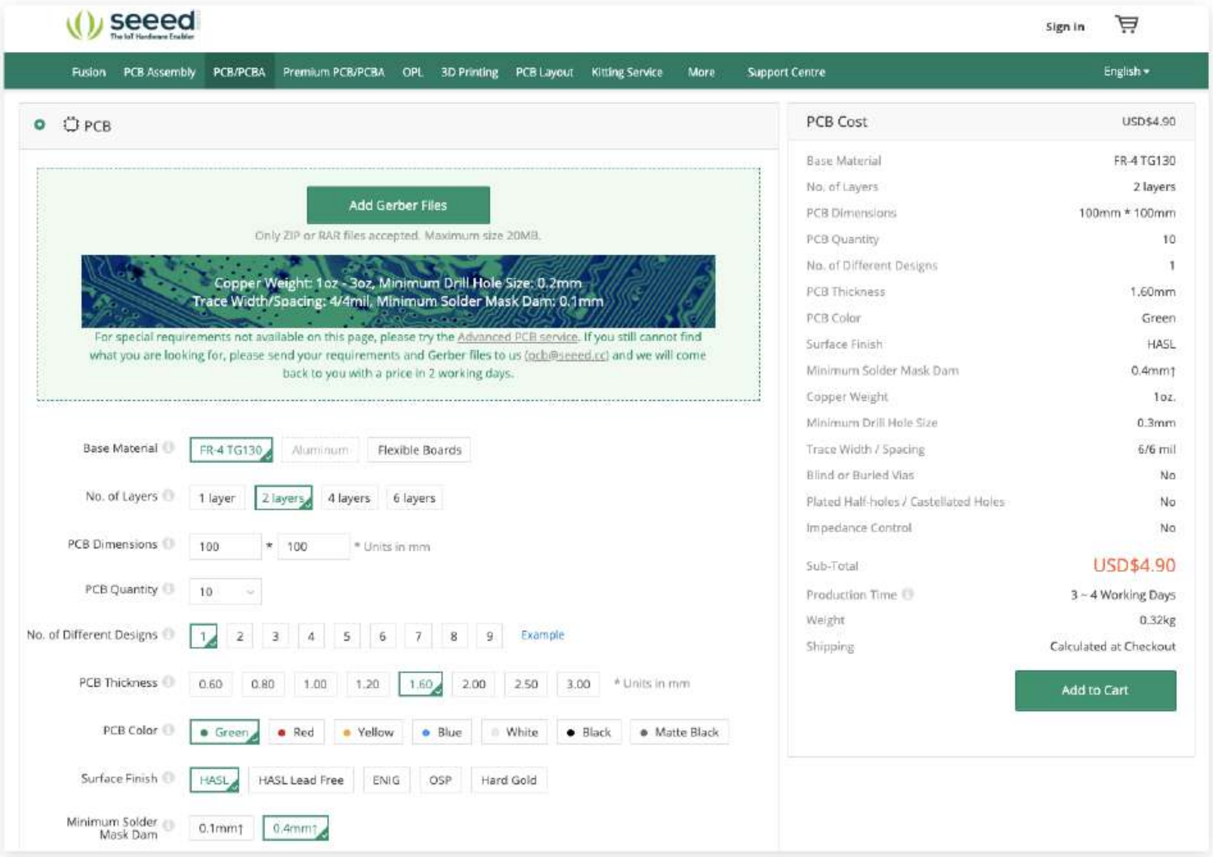
- **Geppetto**

If you don't want to work on schematic or layout yourself, but you want to convert your prototype based on Seeed's modules into an integrated product, we highly recommend you to try Geppetto.

Geppetto is, by far, the easiest and least expensive way to produce production-quality electronics. You don't need to know about resistors, capacitors, inductors, routing paths or anything EE to design your own module. Geppetto is drag & drop simple… anyone can create professional IoT devices. To make it more convenient for you, Geppeto has a library for Seeed, so you can design your own Grove modules with ease!

PCB Services

After you're done with your design, check out the Seeed Fusion PCBA service, which can help translate your designs into an actual functioning device.



Only ZIP or RAR files accepted. Maximum size 20MB.

Copper Weight: 1oz - 3oz, Minimum Drill Hole Size: 0.2mm
Trace Width/Spacing: 4/4mil, Minimum Solder Mask Dam: 0.1mm

For special requirements not available on this page, please try the Advanced PCB service. If you still cannot find what you are looking for, please send your requirements and Gerber files to us (pcb@seeed.cc) and we will come back to you with a price in 2 working days.

Base Material	[✓ FR-4 TG130] Aluminum Flexible Boards
No. of Layers	1 layer [✓ 2 layers] 4 layers 6 layers
PCB Dimensions	100 * 100 * Units in mm
PCB Quantity	10
No. of Different Designs	[✓ 1] 2 3 4 5 6 7 8 9
PCB Thickness	0.60 0.80 1.00 1.20 [✓ 1.60] 2.00 2.50 3.00 * Units in mm
PCB Color	[✓ Green] Red Yellow Blue White Black Matte Black
Surface Finish	[✓ HASL] HASL Lead Free ENIG OSP Hard Gold
Minimum Solder Mask Dam	0.1mm↑ [✓ 0.4mm↑]

PCB Cost	USD$4.90
Base Material	FR-4 TG130
No. of Layers	2 layers
PCB Dimensions	100mm * 100mm
PCB Quantity	10
No. of Different Designs	1
PCB Thickness	1.60mm
PCB Color	Green
Surface Finish	HASL
Minimum Solder Mask Dam	0.4mm↑
Copper Weight	1oz.
Minimum Drill Hole Size	0.3mm
Trace Width / Spacing	6/6 mil
Blind or Buried Vias	No
Plated Half-holes / Castellated Holes	No
Impedance Control	No
Sub-Total	USD$4.90
Production Time	3 ~ 4 Working Days
Weight	0.32kg
Shipping	Calculated at Checkout

Seeed Studio has its very own Open Parts Library (OPL) which is a collection of over 10,000 commonly used components specifically sourced for the Seeed Fusion PCBA Service. To speed up the process of PCB design, Seeed is building the component libraries for KiCad and Eagle. When all components are sourced from Seeed's PCBA OPL and used with the Seeed Fusion PCB Assembly (PCBA) service, the entire PCBA production time can be reduced from 20 working days to a mere 7 days.

Resources

1. **Grove Beginner Kit for Arduino Wiki [PDF]**
2. **Grove Beginner Kit for Arduino Schematic Design Files**
3. **Modules Libraries on Github:**
    - OLED Display
    - Temperature & Humidity Sensor
    - Air Pressure Sensor
    - 3-Axis Accelerator
4. **Sensor Datasheet**
5. **Initial Arduino Firmware Demo**

More Learning

- LSTM for live IoT data prediction

Tech Support

Please submit any technical issue into our forum