



# Ultimate Sensor kit

for UNO

Sharing Perfects Innovation

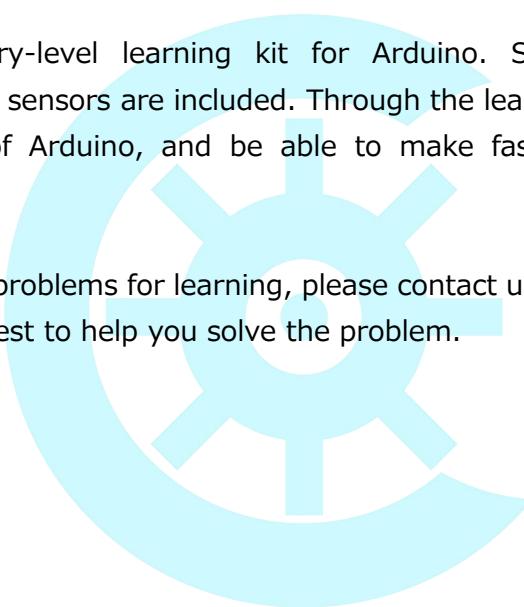


## Preface

Adeept is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim to create infinite possibilities with sharing. No matter what field you are in, we can lead you into the electronic world and bring your ideas into reality.

This is an entry-level learning kit for Arduino. Some common electronic components and sensors are included. Through the learning, you will get a better understanding of Arduino, and be able to make fascinating works based on Arduino.

If you have any problems for learning, please contact us at [support@adeept.com](mailto:support@adeept.com). We will do our best to help you solve the problem.

The logo consists of a large, stylized, light blue lowercase 'a' and 'd' followed by a lowercase 'ee' and a lowercase 'p' stacked vertically. A smaller, light blue plus sign is positioned to the right of the 'p'.

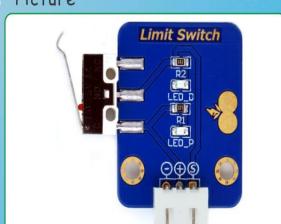
Adeept

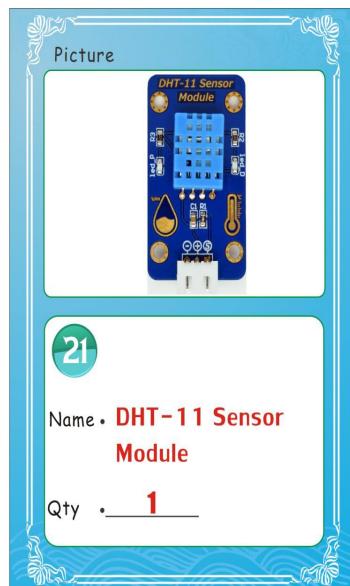
## Component List

<p>Picture</p> 	<p>Picture</p> 	<p>Picture</p> 
<p>1</p> <p>Name • <b>Arduino compatible UNO R3</b></p> <p>Qty • <u>1</u></p>	<p>2</p> <p>Name • <b>LED Module</b></p> <p>Qty • <u>4</u></p>	<p>3</p> <p>Name • <b>Button Module</b></p> <p>Qty • <u>4</u></p>
<p>Picture</p> 	<p>Picture</p> 	<p>Picture</p> 
<p>4</p> <p>Name • <b>RGB LED Module</b></p> <p>Qty • <u>1</u></p>	<p>5</p> <p>Name • <b>Potentiometer Module</b></p> <p>Qty • <u>1</u></p>	<p>6</p> <p>Name • <b>Vibration Sensor Module</b></p> <p>Qty • <u>1</u></p>
<p>Picture</p> 	<p>Picture</p> 	<p>Picture</p> 
<p>7</p> <p>Name • <b>Hall Sensor Module</b></p> <p>Qty • <u>1</u></p>	<p>8</p> <p>Name • <b>Photoresistor Module</b></p> <p>Qty • <u>1</u></p>	<p>9</p> <p>Name • <b>Thermistor Module</b></p> <p>Qty • <u>1</u></p>

<p>Picture</p>  <p>10</p> <p>Name • <b>DS18B20 Module</b></p> <p>Qty • <u>1</u></p>	<p>Picture</p>  <p>11</p> <p>Name • <b>Active Buzzer Module</b></p> <p>Qty • <u>1</u></p>	<p>Picture</p>  <p>12</p> <p>Name • <b>Passive Buzzer Module</b></p> <p>Qty • <u>1</u></p>
--	--	---

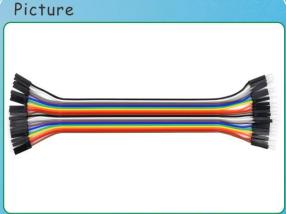
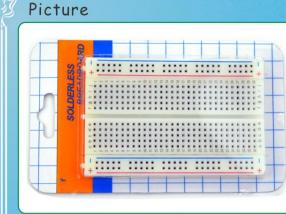
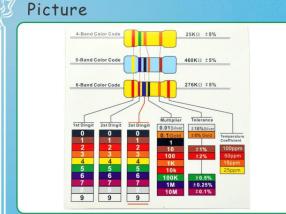
<p>Picture</p>  <p>13</p> <p>Name • <b>PIR Sensor Module</b></p> <p>Qty • <u>1</u></p>	<p>Picture</p>  <p>14</p> <p>Name • <b>LCD1602 Module</b></p> <p>Qty • <u>1</u></p>	<p>Picture</p>  <p>15</p> <p>Name • <b>IIC Interface Module</b></p> <p>Qty • <u>1</u></p>
--	---	---

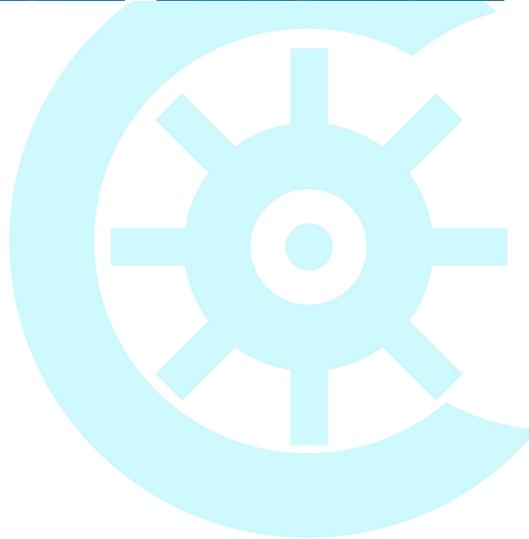
<p>Picture</p>  <p>16</p> <p>Name • <b>Touch Button Module</b></p> <p>Qty • <u>1</u></p>	<p>Picture</p>  <p>17</p> <p>Name • <b>Rotary Encoder Module</b></p> <p>Qty • <u>1</u></p>	<p>Picture</p>  <p>18</p> <p>Name • <b>Limit Switch Module</b></p> <p>Qty • <u>1</u></p>
---	---	---







<p>Picture</p> 	<p>Picture</p> 	<p>Picture</p> 
<p><b>46</b></p> <p>Name • Male to Female Jumper Wires</p> <p>Qty • <u>20</u></p>	<p><b>47</b></p> <p>Name • Breadboard</p> <p>Qty • <u>1</u></p>	<p><b>48</b></p> <p>Name • Band Resistor Card</p> <p>Qty • <u>1</u></p>



# Adeept

# Contents

<b>About Arduino.....</b>	<b>1</b>
<b>About Processing.....</b>	<b>2</b>
<b>Lesson 1 Blinking LED.....</b>	<b>3</b>
<b>Lesson 2 Controlling An LED by A Button.....</b>	<b>6</b>
<b>Lesson 3 Controlling An RGB LED by PWM.....</b>	<b>9</b>
<b>Lesson 4 How To Use Potentiometers.....</b>	<b>12</b>
<b>Lesson 5 Control An LED by Vibration.....</b>	<b>16</b>
<b>Lesson 6 How To Use The Hall Sensor.....</b>	<b>19</b>
<b>Lesson 7 How To Use The Photoresistor.....</b>	<b>23</b>
<b>Lesson 8 How To Use The Thermistor.....</b>	<b>27</b>
<b>Lesson 9 How To Use The DS18B20.....</b>	<b>31</b>
<b>Lesson 10 Alarm Prompt.....</b>	<b>37</b>
<b>Lesson 11 Playing Music.....</b>	<b>40</b>
<b>Lesson 12 Detection of Human Body Movement.....</b>	<b>43</b>
<b>Lesson 13 How To Use The LCD1602.....</b>	<b>47</b>
<b>Lesson 14 IIC Interface Application.....</b>	<b>52</b>
<b>Lesson 15 Application of Touch Button.....</b>	<b>58</b>
<b>Lesson 16 Pulse Count.....</b>	<b>62</b>
<b>Lesson 17 Impact checking.....</b>	<b>66</b>
<b>Lesson 18 Simple Laser Cannon.....</b>	<b>70</b>
<b>Lesson 19 Simple Laser Targeting.....</b>	<b>73</b>
<b>Lesson 20 Temperature And Humidity Detection.....</b>	<b>77</b>
<b>Lesson 21 How To Use The Reed.....</b>	<b>82</b>
<b>Lesson 22 Fire Detection.....</b>	<b>86</b>
<b>Lesson 23 Decetion of Flammable Gases.....</b>	<b>90</b>
<b>Lesson 24 Tracking Test.....</b>	<b>94</b>
<b>Lesson 25 How To Use Slide Potentiometer.....</b>	<b>98</b>
<b>Lesson 26 Small Fan Works.....</b>	<b>102</b>
<b>Lesson 27 How To Use The Joystick.....</b>	<b>105</b>
<b>Lesson 28 How To Use The MIC Module.....</b>	<b>109</b>
<b>Lesson 29 How To Use The Relay Module.....</b>	<b>113</b>
<b>Lesson 30 How To Use The Segment Module.....</b>	<b>117</b>
<b>Lesson 31 How To Use The 8*8 LED Matrix.....</b>	<b>121</b>
<b>Lesson 32 Indication of Signal.....</b>	<b>124</b>
<b>Lesson 33 Making A Simple Remote Control Device.....</b>	<b>129</b>
<b>Lesson 34 Detection of The Soil Moisture System.....</b>	<b>134</b>
<b>Lesson 35 Detection of The Water Height System.....</b>	<b>139</b>
<b>Lesson 36 Detection of The Distance System.....</b>	<b>144</b>
<b>Lesson 37 Control An LED by PC.....</b>	<b>148</b>
<b>Lesson 38 Upload The State of A Button to PC.....</b>	<b>151</b>
<b>Lesson 39 Simple Laser Pen.....</b>	<b>154</b>

<b>Lesson 40 Control Buzzer by Button</b> .....	157
<b>Lesson 41 A Simple Piano</b> .....	160
<b>Lesson 42 Change The Color of The RGB LED</b> .....	163
<b>Lesson 43 A Simple Light Control Lamp</b> .....	166
<b>Lesson 44 Control Segment Display by Rotary Encoder</b> .....	169
<b>Lesson 45 A Simple Temperature &amp; Humidity Monitoring and Alarm System(1)</b> .....	172
<b>Lesson 46 A Simple Temperature &amp; Humidity Monitoring and Alarm System(2)</b> .....	175
<b>Lesson 47 A Simple Flammable Gases Monitoring and Alarm System(1)</b> .....	178
<b>Lesson 48 A Simple Flammable Gases Monitoring and Alarm System(2)</b> .....	181
<b>Lesson 49 A Simple Clock</b> .....	184
<b>Lesson 50 Arduino Interacts with Processing(IED Module)</b> .....	186
<b>Lesson 51 Arduino Interacts with Processing(Button Module)</b> .....	207
<b>Lesson 52 Arduino Interacts with Processing(RGB Module)</b> .....	212
<b>Lesson 53 Arduino Interacts with Processing(Potentiometer)</b> .....	217
<b>Lesson 54 Arduino Interacts with Processing(Vibration Module)</b> .....	223
<b>Lesson 55 Arduino Interacts with Processing(Photoresistor)</b> .....	229
<b>Lesson 56 Arduino Interacts with Processing(DS18B20 Module)</b> .....	234
<b>Lesson 57 Arduino Interacts with Processing(Buzzer Module)</b> .....	239
<b>Lesson 58 Arduino Interacts with Processing(Rotary Encoder)</b> .....	243
<b>Lesson 59 Arduino Interacts with Processing(Joystick Module)</b> .....	251
<b>Lesson 60 Arduino Interacts with Processing(Ultrasonic Distance Module)</b> .....	256



# About Arduino

## *What is Arduino?*

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

## *ARDUINO BOARD*

Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

## *ARDUINO SOFTWARE*

You can tell your Arduino what to do by writing code in the Arduino programming language and using the Arduino development environment.

Before the development of Arduino program, the first thing you have to do is to install Arduino IDE software. The software provides you with the basic development environment that is required for developing Arduino program. You need the following URL to download Arduino IDE:

<http://www.arduino.cc/en/Main/Software>

For different operating system platforms, the way of using Arduino IDE is different. Please refer to the following links:

Windows User : <http://www.arduino.cc/en/Guide/Windows>

Mac OS X User : <http://www.arduino.cc/en/Guide/MacOSX>

Linux User : <http://playground.arduino.cc/Learning/Linux>

For more detailed information about Arduino IDE, please refer to the following link:

<http://www.arduino.cc/en/Guide/HomePage>

## About Processing

### *What is Processing?*

Processing is a programming language, development environment, and online community. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology. Initially created to serve as a software sketchbook and to teach computer programming fundamentals within a visual context, Processing evolved into a development tool for professionals. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

- » Free to download and open source
- » Interactive programs with 2D, 3D or PDF output
- » OpenGL integration for accelerated 3D
- » For GNU/Linux, Mac OS X, and Windows
- » Over 100 libraries extend the core software

### *PROCESSING SOFTWARE*

Download Processing:

<https://www.processing.org/download/>

For more detailed information about Processing IDE, please refer to the following link:

<https://www.processing.org/reference/environment/>

A  
d  
e  
e  
p  
t

# Lesson 1 Blinking LED

## Introduction

LED is usually used in office lighting, furniture, decoration, sign board, streetlight, etc.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* LED Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

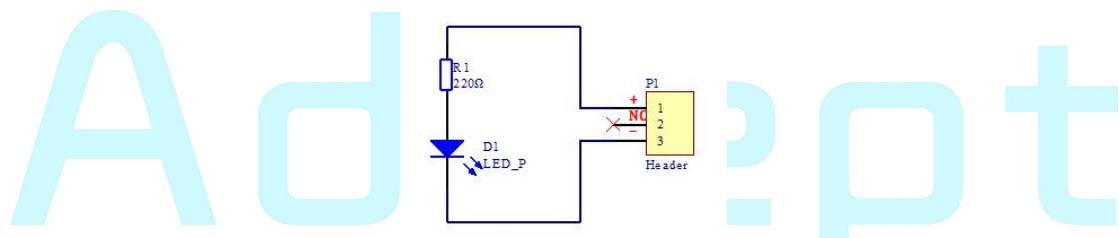
The Fritzing images:



Pin definition:

N	NULL
+	VCC
-	GND

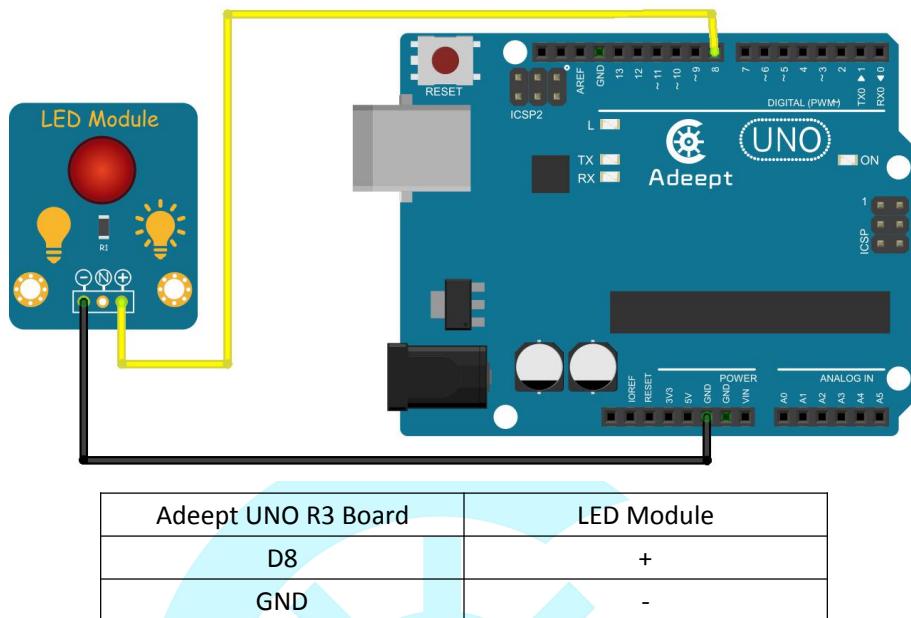
The schematic diagram:



In this experiment, we make the pin D8 of the Arduino board output High/Low by programming, to control the LED to blink in a certain frequency.

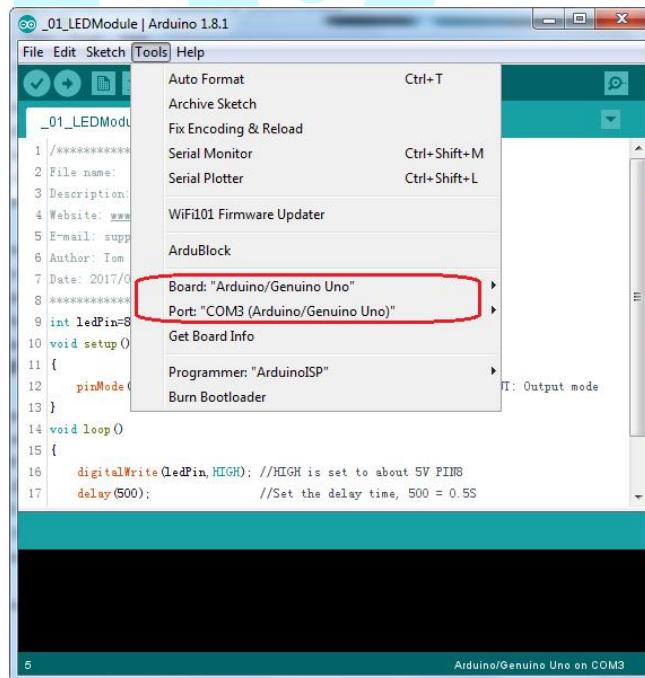
## Experimental Procedures

**Step 1:** Build the circuit



**Step 2:** Program \_01\_LEDModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



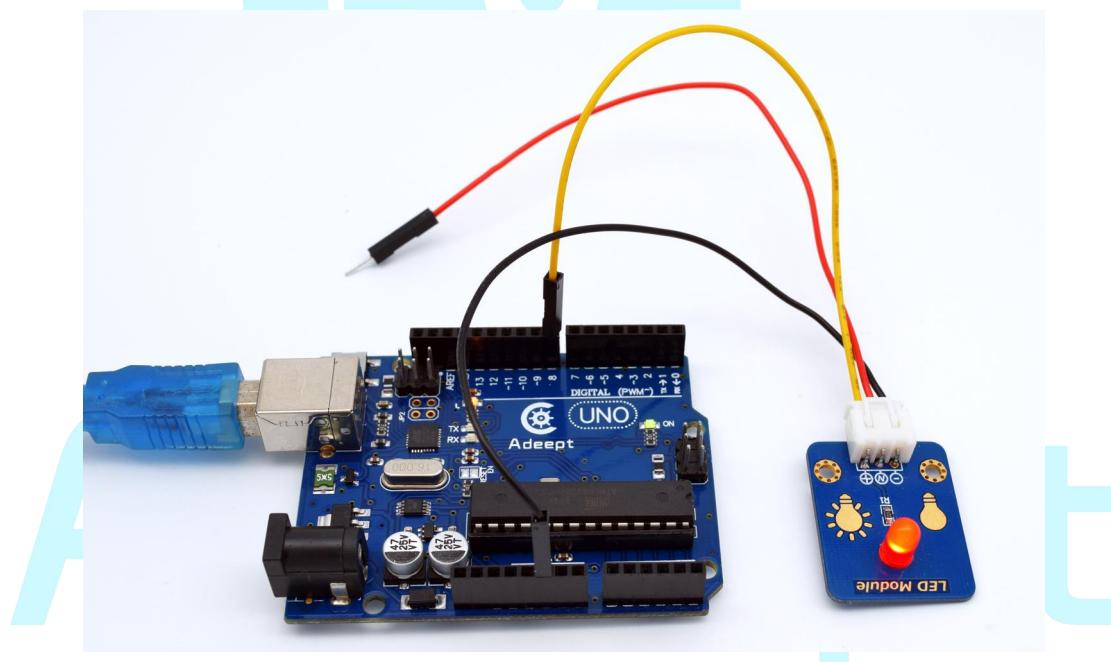


```
01_LEDModule | Arduino 1.8.1
File Edit Sketch Tools Help
01_LEDModule
1 //*****
2 File name: _01_LEDModule.ino
3 Description: Lit LED, let LED blinks.
4 Website: www.adeept.com
5 E-mail: support@adeept.com
6 Author: Tom
7 Date: 2017/07/12
8 ****
9 int ledPin=8; //definition digital 8 pin to control the LED module
10 void setup()
11 {
12     pinMode(ledPin,OUTPUT); //Set the digital 8 port mode, OUTPUT: Output mode
13 }
14 void loop()
15 {
16     digitalWrite(ledPin,HIGH); //HIGH is set to about 5V PINS
17     delay(500); //Set the delay time, 500 = 0.5S
}
Done uploading.

Sketch uses 940 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.

3
!!!!
Arduino/Genuine Uno on COM3
```

Now you can see the LED on the LED module blinks once per second.



# Lesson 2 Controlling An LED by A Button

## Introduction

Buttons, or touch switches, are light-touch button switches. A button is an electronic switch and usually used for device control.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Button Module
- 1 \* LED Module
- 1 \* USB Cable
- 2 \* 3-Pin Wires
- 2 \* Hookup Wire Set
- 1 \* Breadboard

## Experimental Principle

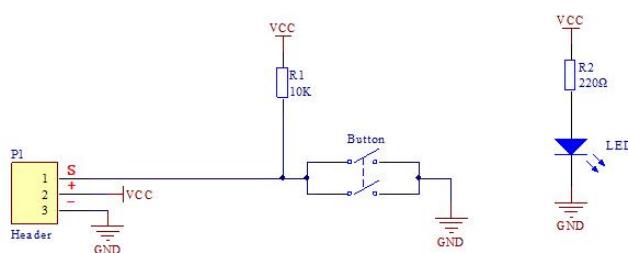
The Fritzing image:



Pin definition:

S	Digital keys output
+	VCC
-	GND

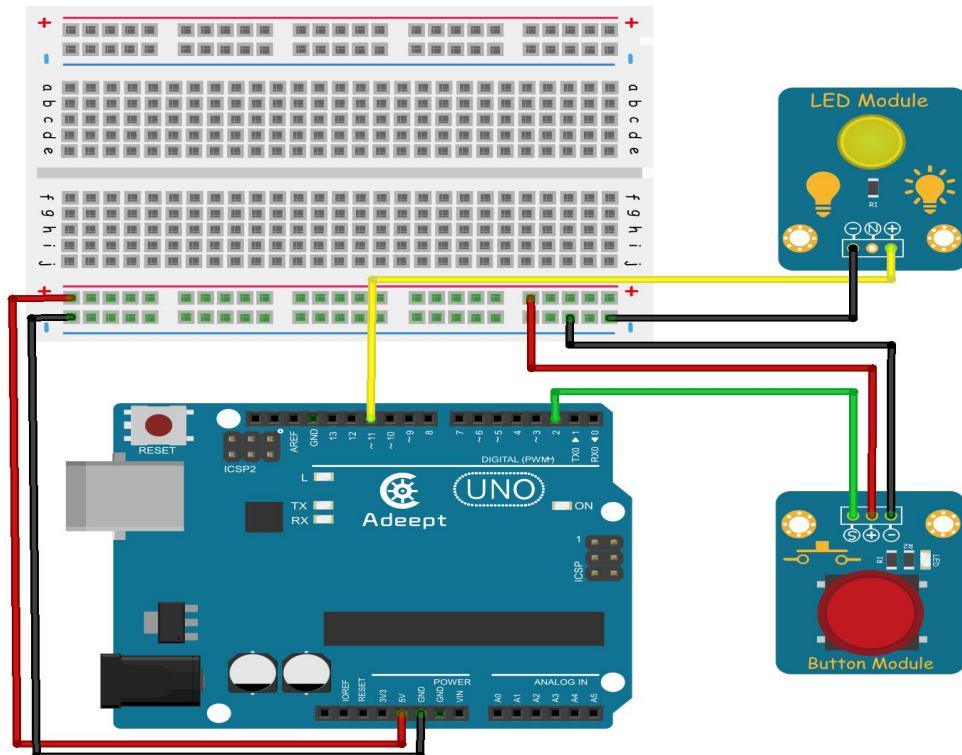
The schematic diagram:



In this experiment, we detect the High or Low level of pin D2 of the Arduino board and then control the LED connected to D8 accordingly.

## Experimental Procedures

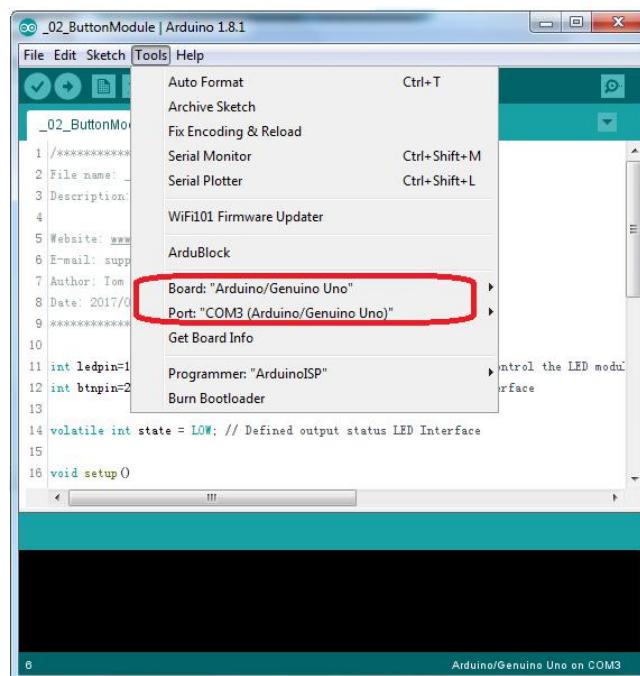
**Step 1:** Build the circuit

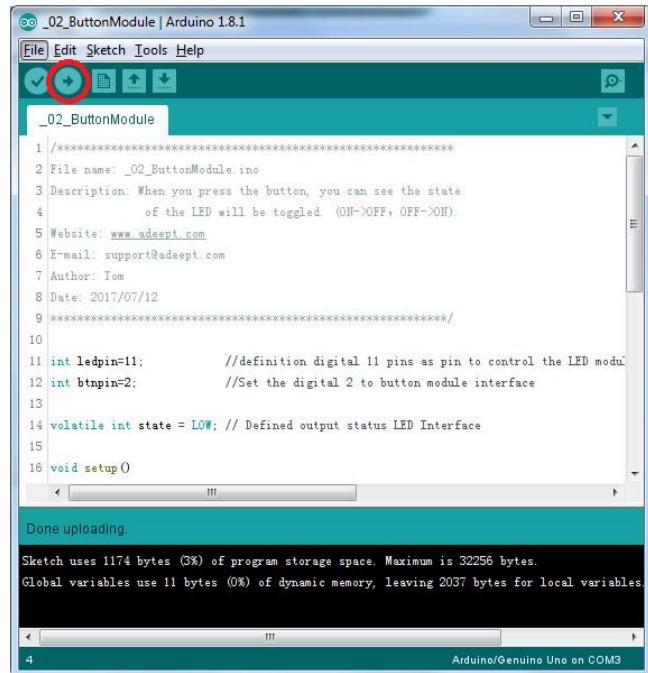


Adept UNO R3 Board	Button Module	LED Module
GND	-	-
5V	+	
D2	S	
D11		+

**Step 2:** Program \_02\_ButtonModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



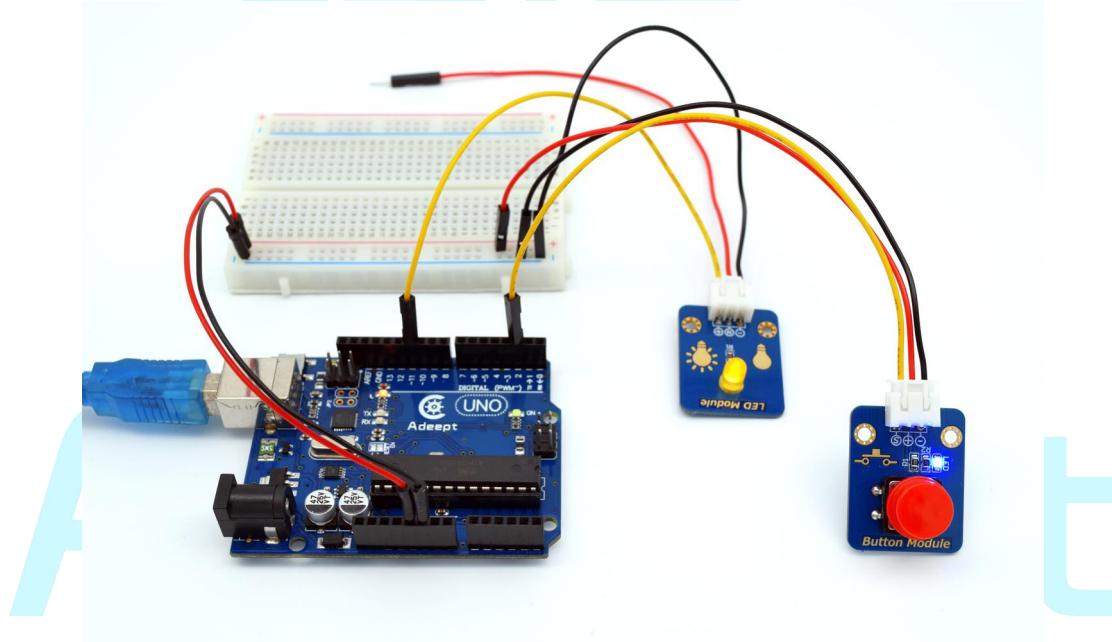


The screenshot shows the Arduino IDE interface with the title bar "00 \_02\_ButtonModule | Arduino 1.8.1". The menu bar includes File, Edit, Sketch, Tools, and Help. A red circle highlights the "Upload" button in the toolbar. The code editor contains the following sketch:

```
1 //*****  
2 File name: _02_ButtonModule.ino  
3 Description: When you press the button, you can see the state  
4          of the LED will be toggled. (ON->OFF, OFF->ON).  
5 Website: www.adeept.com  
6 E-mail: support@adeept.com  
7 Author: Tom  
8 Date: 2017/07/12  
9 *****  
10  
11 int ledpin=11;           //definition digital 11 pins as pin to control the LED module  
12 int btnpin=2;            //Set the digital 2 to button module interface  
13  
14 volatile int state = LOW; // Defined output status LED Interface  
15  
16 void setup()  
17 {  
18     // put your setup code here, to run once:  
19     // initialize your digital pin as an output.  
20     // set the digital pin to HIGH or LOW  
21     pinMode(ledpin, OUTPUT);  
22     // initialize your digital pin as an input.  
23     // set the digital pin to INPUT  
24     pinMode(btnpin, INPUT);  
25 }  
26  
27 void loop()  
28 {  
29     if (digitalRead(btnpin) == HIGH)  
30     {  
31         if (state == LOW)  
32             state = HIGH;  
33         else  
34             state = LOW;  
35         // set the LED pin to state  
36         digitalWrite(ledpin, state);  
37     }  
38 }  
39 
```

The status bar at the bottom right indicates "Arduino/Genuine Uno on COM3". Below the IDE, a message says "Done uploading." followed by memory usage details.

Press the button and you can see the LED toggle between on and off.



# Lesson 3 Controlling An RGB LED by PWM

## Introduction

RGB LED is designed based on the principle of three primary colors. In an RGB LED, three LEDs in red, green, and blue respectively are packaged together, thus by controlling the brightness of three LEDs, making the RGB LED flash multiple colors

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* RGB LED Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

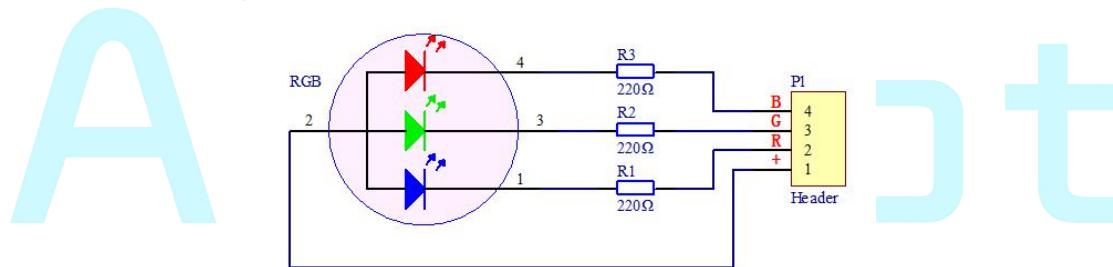
The Fritzing image:



Pin definition:

B	Blue Channel
G	Green Channel
R	Red Channel
+	VCC

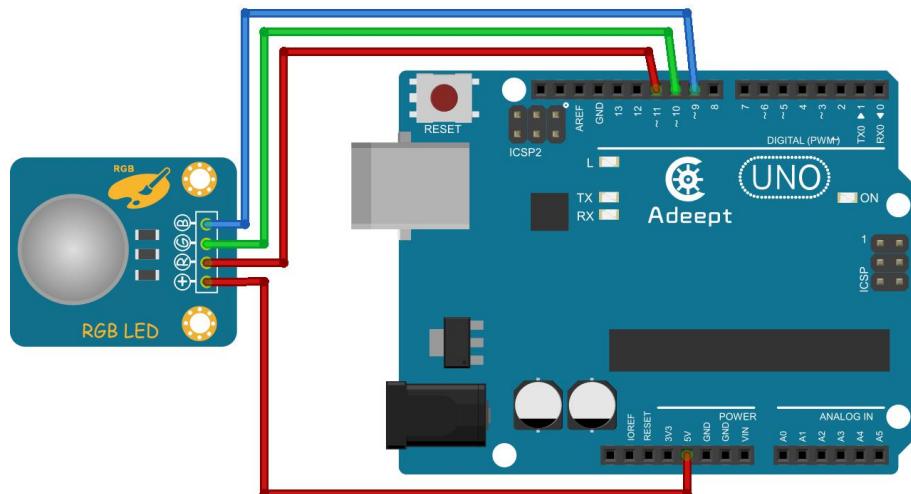
The schematic diagram:



In this experiment, we make the pin D9, D10, and D11 of the Arduino board output PWM (pulse-width modulation) signals by programming, to make the RGB LED flash different colors.

## Experimental Procedures

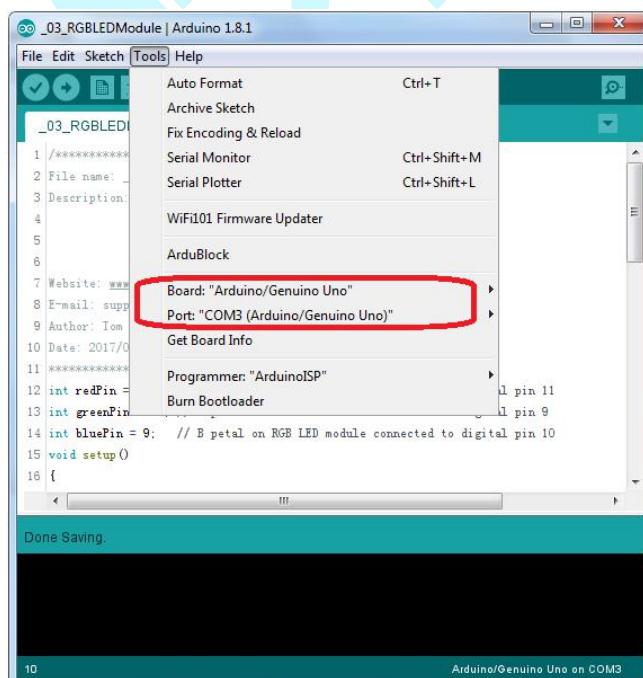
### Step 1: Build the circuit

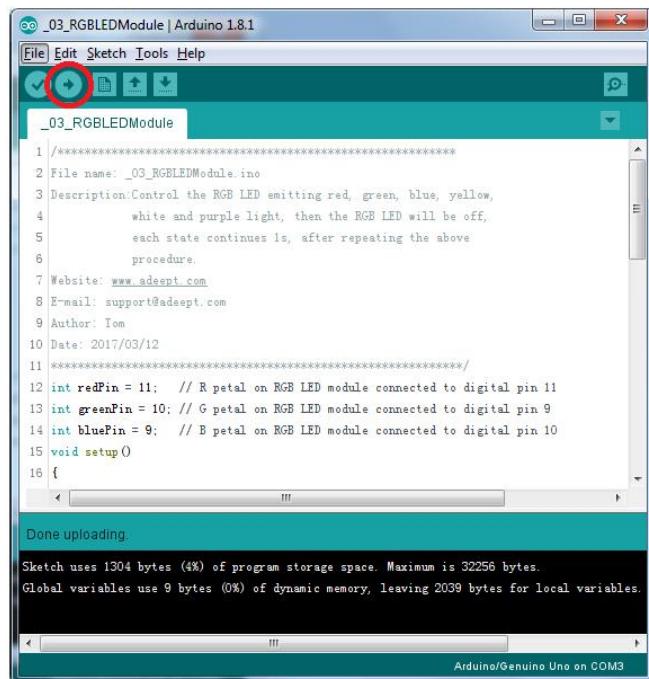


Adeept UNO R3 Board	RGB LED Module
D9	B
D10	G
D11	R
5V	+

**Step 2:** Program \_03\_RGBLEDModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



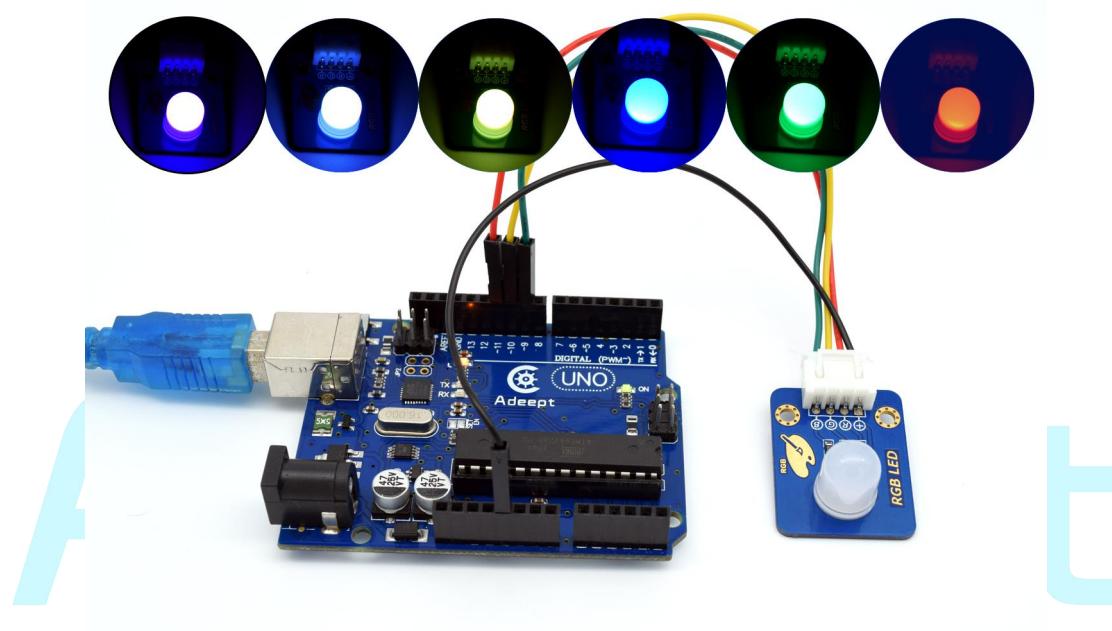


The screenshot shows the Arduino IDE interface with the title bar "03\_RGBLEDModule | Arduino 1.8.1". The code editor contains the following sketch:

```
03_RGBLEDModule
1 //*****
2 File name: _03_RGBLEDModule.ino
3 Description: Control the RGB LED emitting red, green, blue, yellow,
4           white and purple light, then the RGB LED will be off,
5           each state continues 1s, after repeating the above
6           procedure.
7 Website: www.adeept.com
8 E-mail: support@adeept.com
9 Author: Tom
10 Date: 2017/03/12
11 *****
12 int redPin = 11; // R petal on RGB LED module connected to digital pin 11
13 int greenPin = 10; // G petal on RGB LED module connected to digital pin 9
14 int bluePin = 9; // B petal on RGB LED module connected to digital pin 10
15 void setup()
16 {
```

The status bar at the bottom right indicates "Arduino/Genuine Uno on COM3".

Now you can see the RGB LED flash different colors alternately.



# Lesson 4 How To Use Potentiometers

## Introduction

Potentiometer is a resistor whose resistance can be adjusted continuously. When its shaft is turned, the moving contact (or wiper) slides along the resistor.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Potentiometer Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

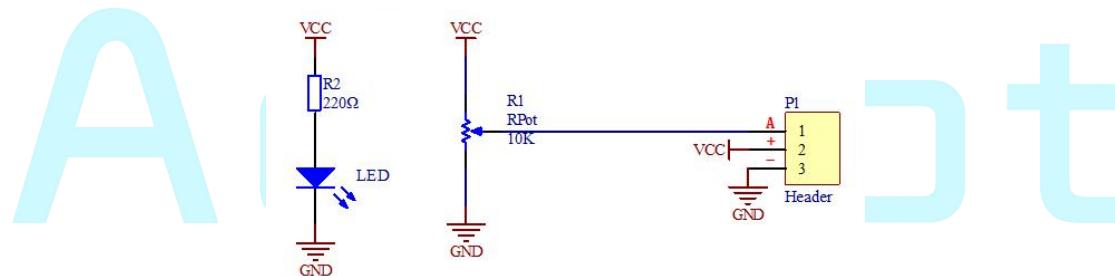
The Fritzing image:



Pin definition:

A	Analog output
+	VCC
-	GND

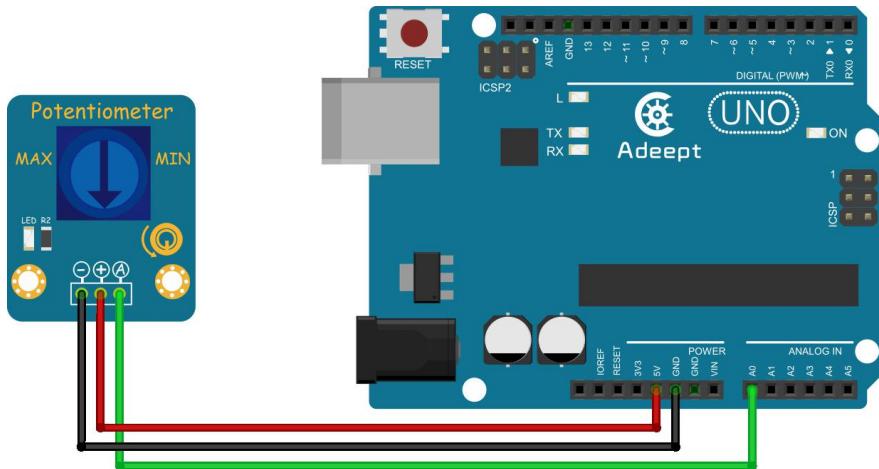
The schematic diagram:



In this experiment, by programming the Arduino, we collect the analog values output by the Potentiometer module through pin A0 of the Arduino board, convert it to digital values and display them on the computer via serial port.

## Experimental Procedures

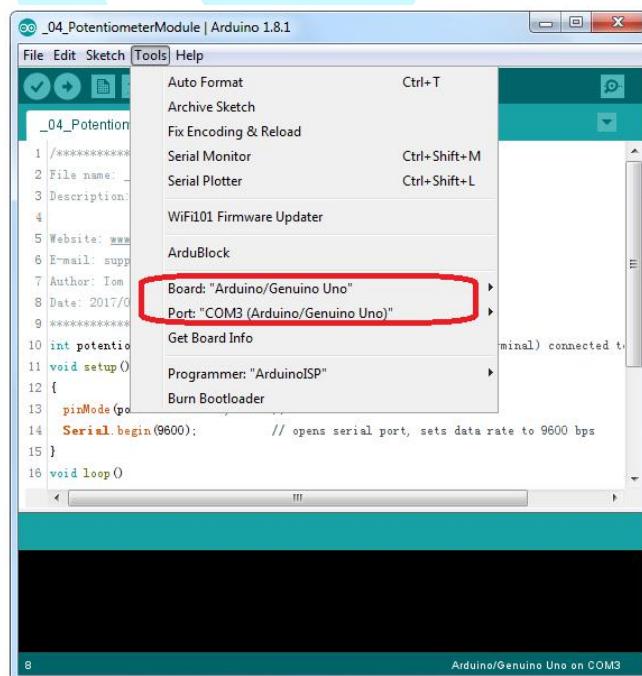
### Step 1: Build the circuit

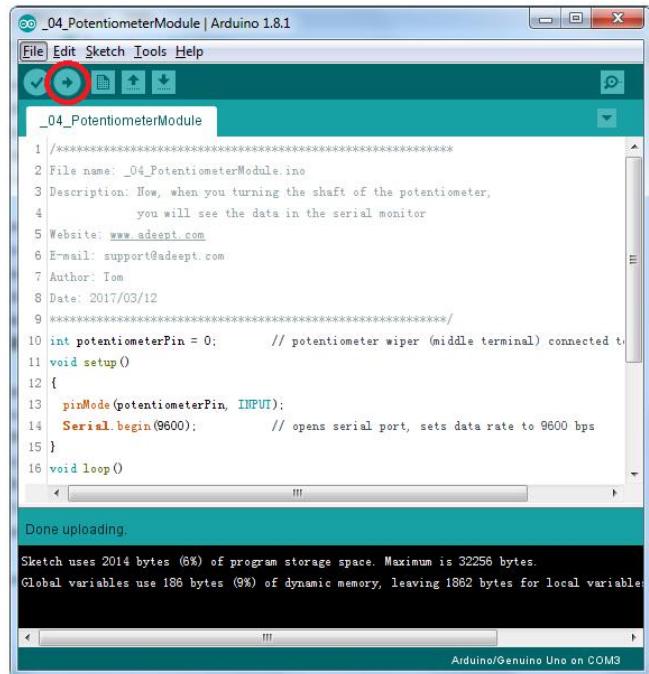


Adept UNO R3 Board	Potentiometer Module
A0	A
5V	+
GND	-

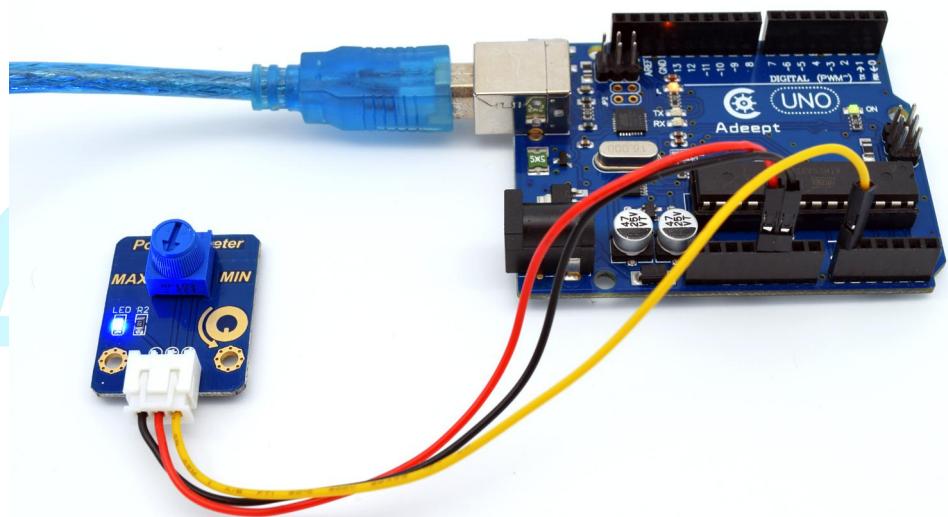
Step 2: Program \_04\_PotentiometerModule.ino

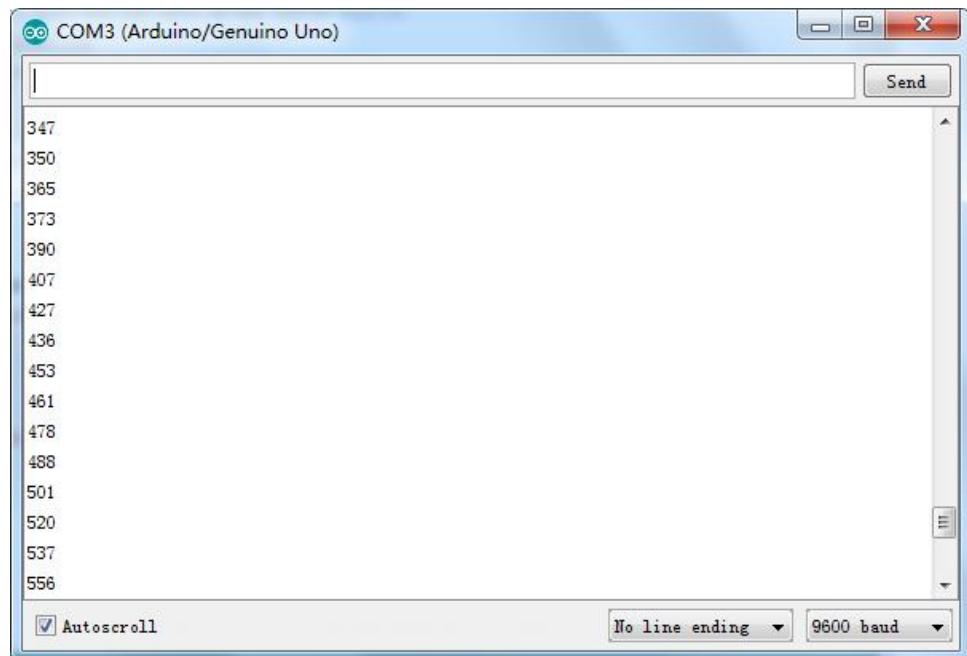
**Step 3:** Compile and download the sketch to the UNO R3 board.





Open the Serial Monitor in Arduino IDE and turn the knob on the potentiometer module. Then the UNO R3 board will upload the data collected to and display in the Serial Monitor. Turn the pot knob clockwise and the value on the window will decrease and turn counterclockwise, it will increase.





# Adeept

# Lesson 5 Control An LED by Vibration

## Introduction

The vibration digital input module can sense weak vibration signals, thus it can be used for related interaction projects. The core sensor is SW-540, a spring component of no-directional vibration sensing which can be triggered at any angle. The module stays off at any angle when it's still. But when it's hit or knocked by external force, the distorted spring contacts and connects the electrode in the middle thus connecting the two pins and turning the sensor on. When the force disappears, the circuit is back to off.

This sensor module is suitable for small-current vibration detection circuits and has been widely used in products such as toys, light shoes, burglar alarms, electronic scales, flash dance shoes, hot wheels, flash balls, etc.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Vibration Sensor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

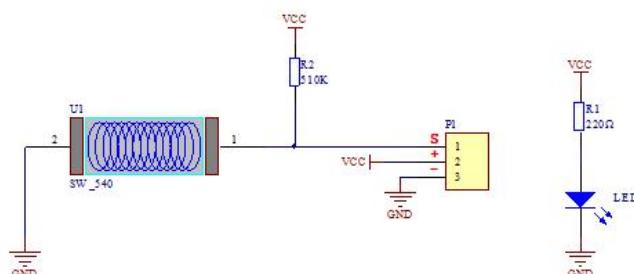
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

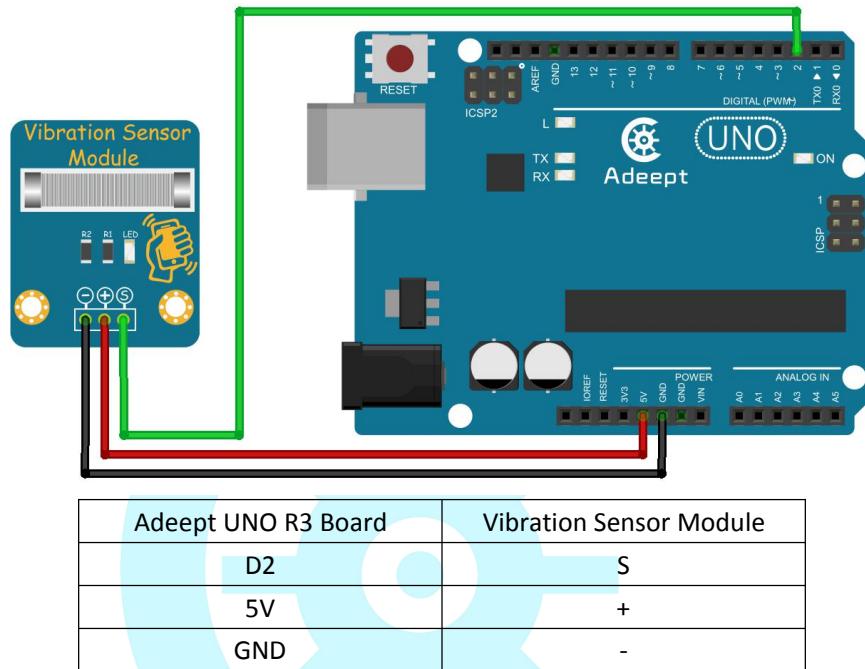
The schematic diagram:



This experiment is to make the LED on the Arduino Uno R3 board flicker with the actions of the Vibration Sensor module.

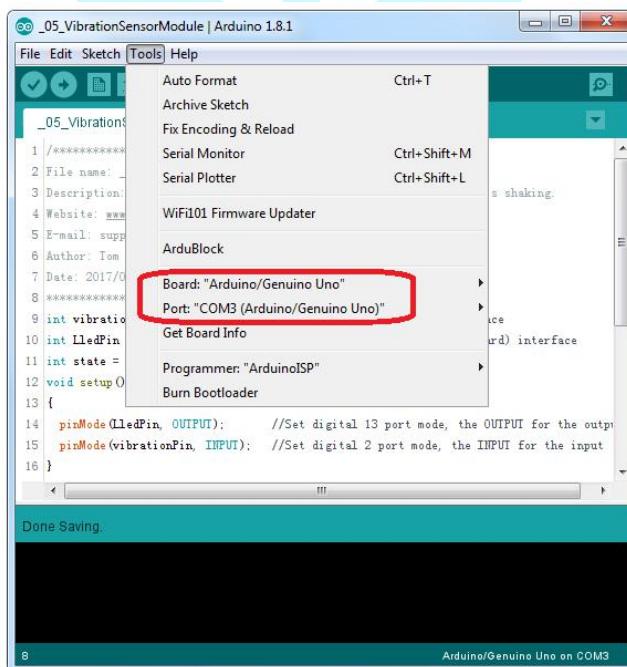
## Experimental Procedures

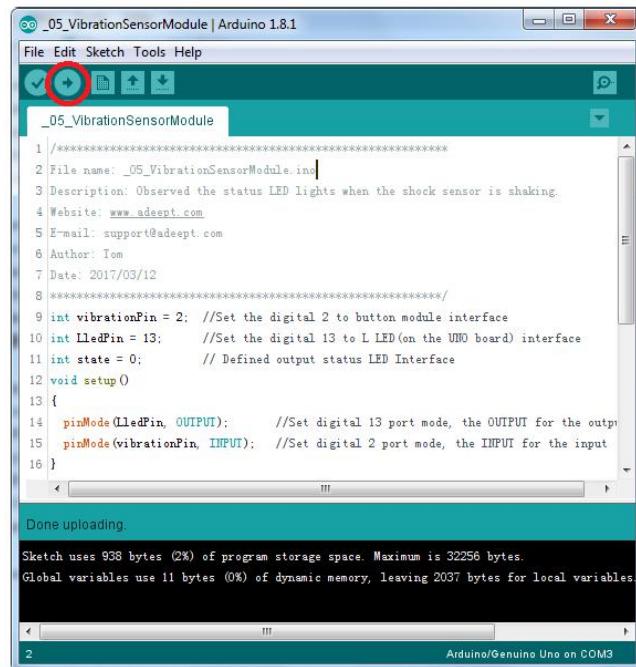
## Step 1: Build the circuit



**Step 2:** Program \_05\_VibrationSensorModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.





```
_05_VibrationSensorModule | Arduino 1.8.1
File Edit Sketch Tools Help
...
_05_VibrationSensorModule
1 /*****
2 File name: _05_VibrationSensorModule.ino
3 Description: Observed the status LED lights when the shock sensor is shaking.
4 Website: www.adeept.com
5 E-mail: support@adeept.com
6 Author: Tom
7 Date: 2017/03/12
8 *****/
9 int vibrationPin = 2; //Set the digital 2 to button module interface
10 int lledPin = 13; //Set the digital 13 to L LED(on the UNO board) interface
11 int state = 0; // Defined output status LED Interface
12 void setup()
13 {
14   pinMode(lledPin, OUTPUT); //Set digital 13 port mode, the OUTPUT for the output
15   pinMode(vibrationPin, INPUT); //Set digital 2 port mode, the INPUT for the input
16 }
...
Done uploading.
Sketch uses 938 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 11 bytes (0%) of dynamic memory, leaving 2037 bytes for local variables.
```

Knock or tap the Vibration Sensor module and you'll see the LED on the Arduino UNO R3 board brighten.



# Lesson 6 How To Use The Hall Sensor

## Introduction

Hall 3144 is a unipolar Hall switch circuit. When the N pole of a magnet approaches to its print surface, the switch outputs Low; when the N pole moves away, the switch outputs High.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Hall Sensor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

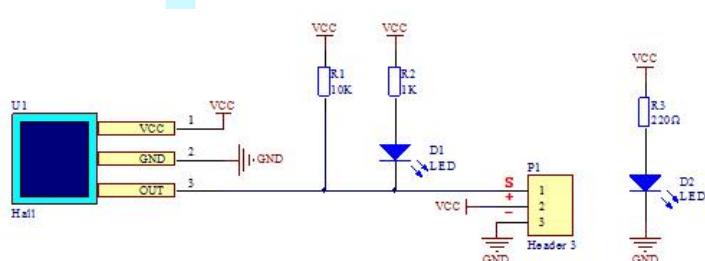
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

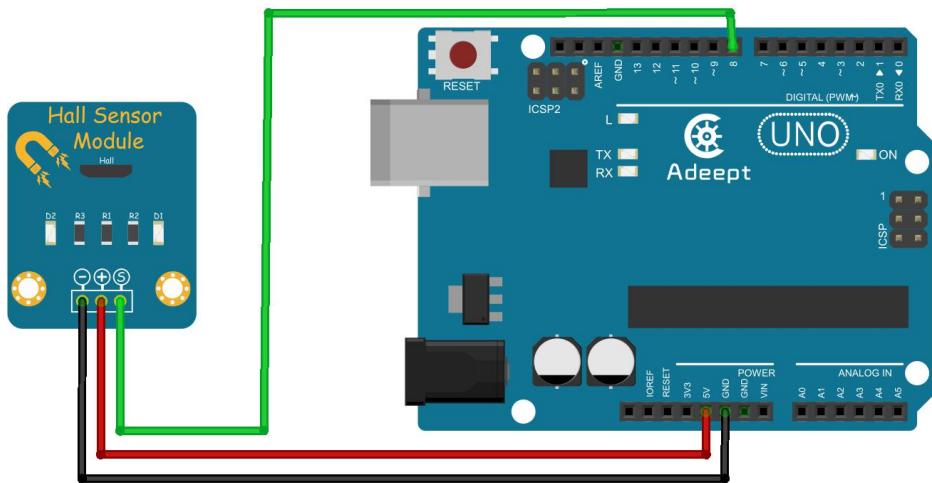
The schematic diagram:



In this experiment, by programming the Arduino, we detect the High or Low level of pin D8 of the Arduino board and then display the result on Serial Monitor.

## Experimental Procedures

### Step 1: Build the circuit



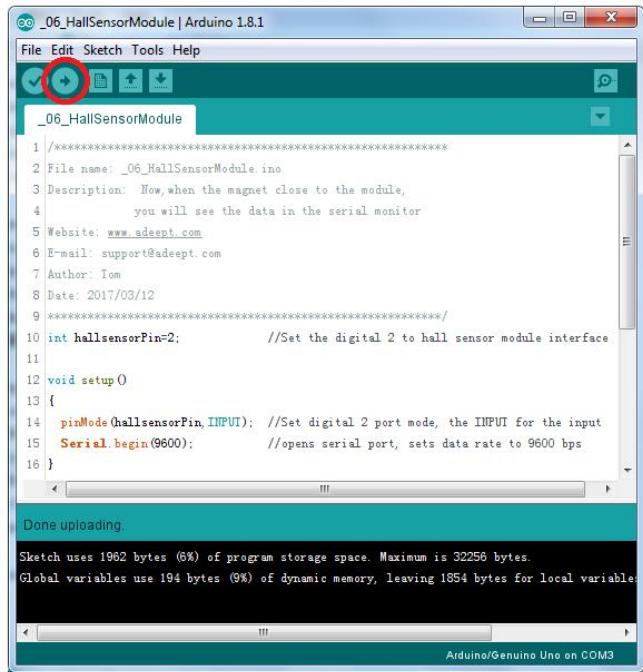
Adeept Uno R3 Board	Hall Sensor Module
D8	S
5V	+
GND	-

**Step 2:** Program `_06_HallSensorModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.

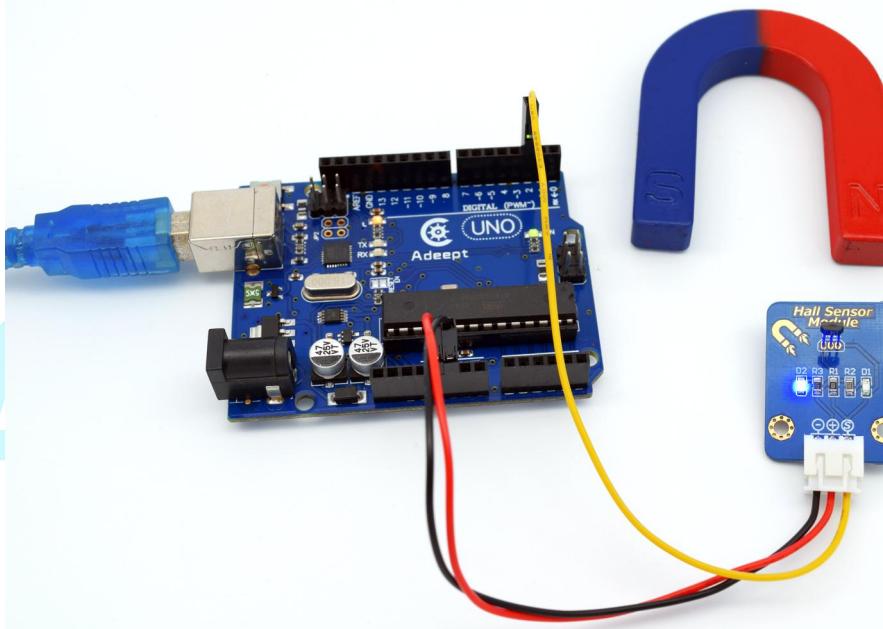
```

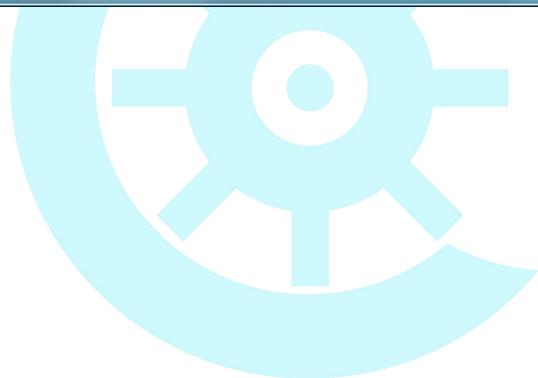
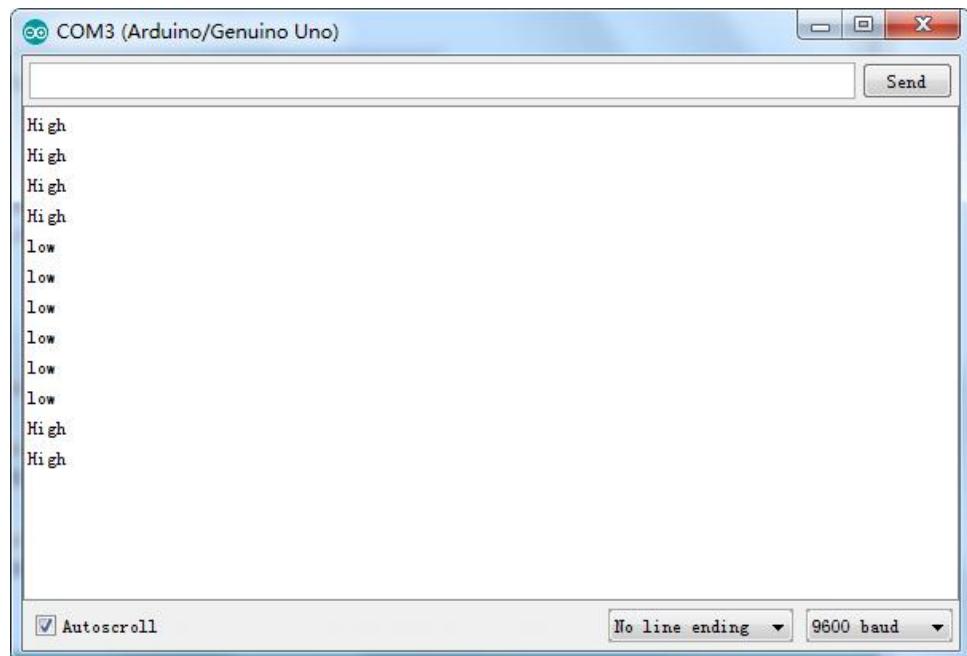
_06_HallSensorModule | Arduino 1.8.1
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
ArduBlock
Board: "Arduino/Genuino Uno"
Port: "COM3 (Arduino/Genuino Uno)" (highlighted)
Get Board Info
Programmer: "ArduinoISP"
Burn Bootloader
module interface
1 //*****
2 File name: _06_HallSensorModule.ino
3 Description:
4 Website: www.adeept.com
5 E-mail: support@adeept.com
6 Author: Tom
7 Date: 2017/01/10
8 *****
9 int hallsensorPin;
10 void setup()
11 {
12   pinMode(hallsensorPin, INPUT); //Set digital 2 port mode, the INPUT for the input
13   Serial.begin(9600); //opens serial port, sets data rate to 9600 bps
14 }
15
16 
```



```
_06_HallSensorModule | Arduino 1.8.1
File Edit Sketch Tools Help
_06_HallSensorModule
1 //*****
2 File name: _06_HallSensorModule.ino
3 Description: Now,when the magnet close to the module,
4 you will see the data in the serial monitor
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/12
9 ****
10 int hallsensorPin=2;           //Set the digital 2 to hall sensor module interface
11
12 void setup()
13 {
14   pinMode(hallsensorPin,INPUT); //Set digital 2 port mode, the INPUT for the input
15   Serial.begin(9600);         //opens serial port, sets data rate to 9600 bps
16 }
Done uploading.
Sketch uses 1982 bytes (6%) of program storage space. Maximum is 32256 bytes.
Global variables use 194 bytes (9%) of dynamic memory, leaving 1854 bytes for local variables
Arduino/Genuine Uno on COM3
```

Now open the Serial Monitor in Arduino IDE. When you place the N pole of a magnet close to or move it away from the Hall Sensor, corresponding status of pin D8 of the Arduino board, i.e. Low or High, will be displayed on the window.





# Adeept

# Lesson 7 How To Use The Photoresistor

## Introduction

The photoresistor module is a resistor module designed based on the principle of photoconductive effect of semiconductors, of which the resistance varies with the intensity of incident light. The resistance of the photoresistor we use decreases with stronger incident light and increases with weaker light. In experiments and daily light, photoresistors are usually used for detecting light intensity.

## Components

- 1 \* Adept Arduino UNO R3 Board
- 1 \* Photoresistor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

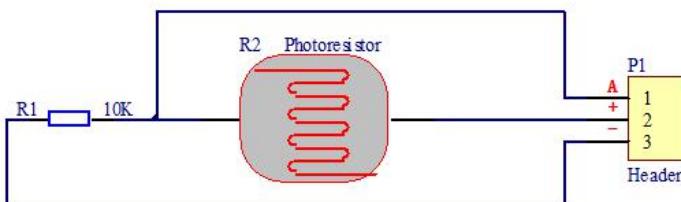
The Fritzing image:



Pin definition:

A	Analog output
+	VCC
-	GND

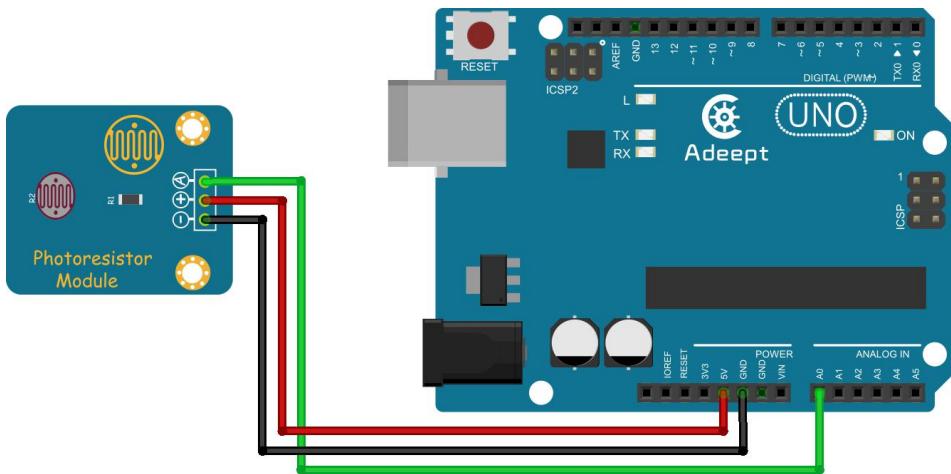
The schematic diagram:



This experiment is to collect the data of light intensity by the Photoresistor module and then display it on the computer.

## Experimental Procedures

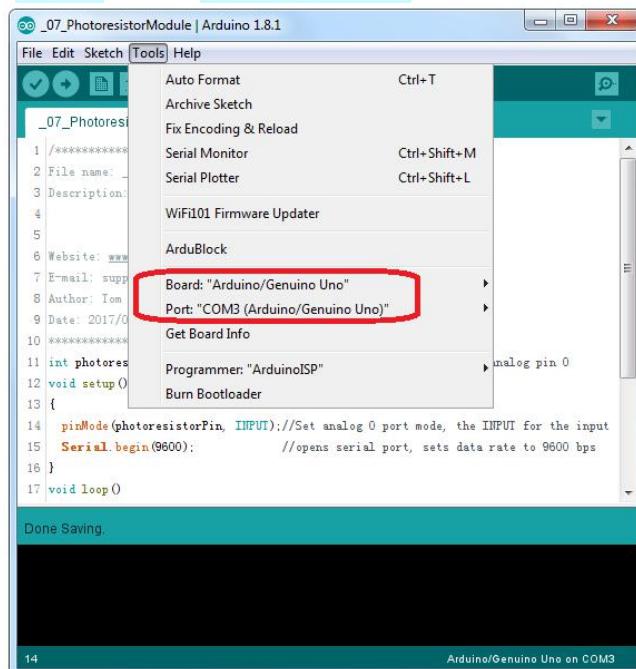
### Step 1: Build the circuit

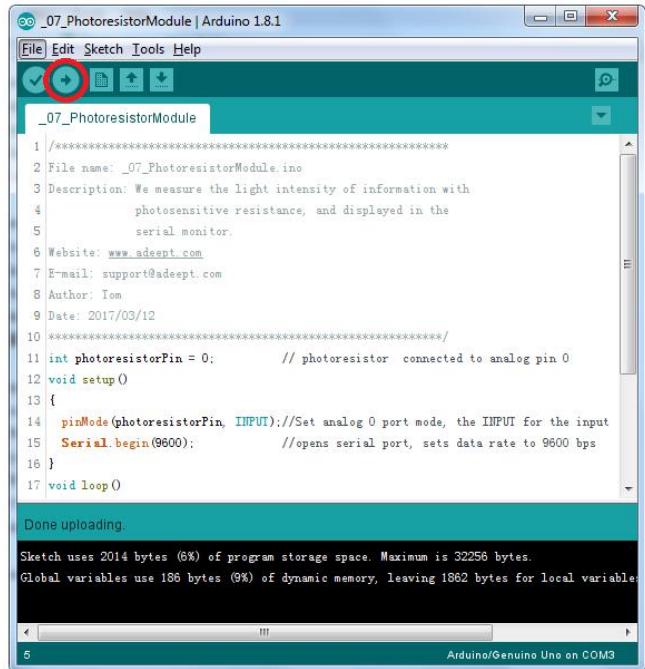


Adeept Uno R3 Board	Photoresistor Module
A0	A
5V	+
GND	-

**Step 2:** Program `_07_PhotoresistorModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



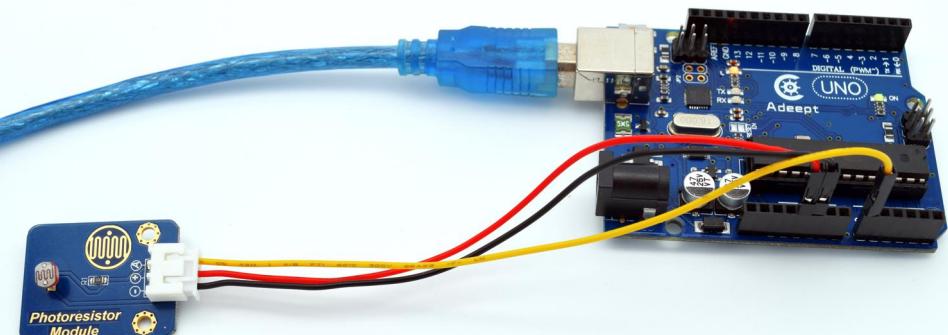


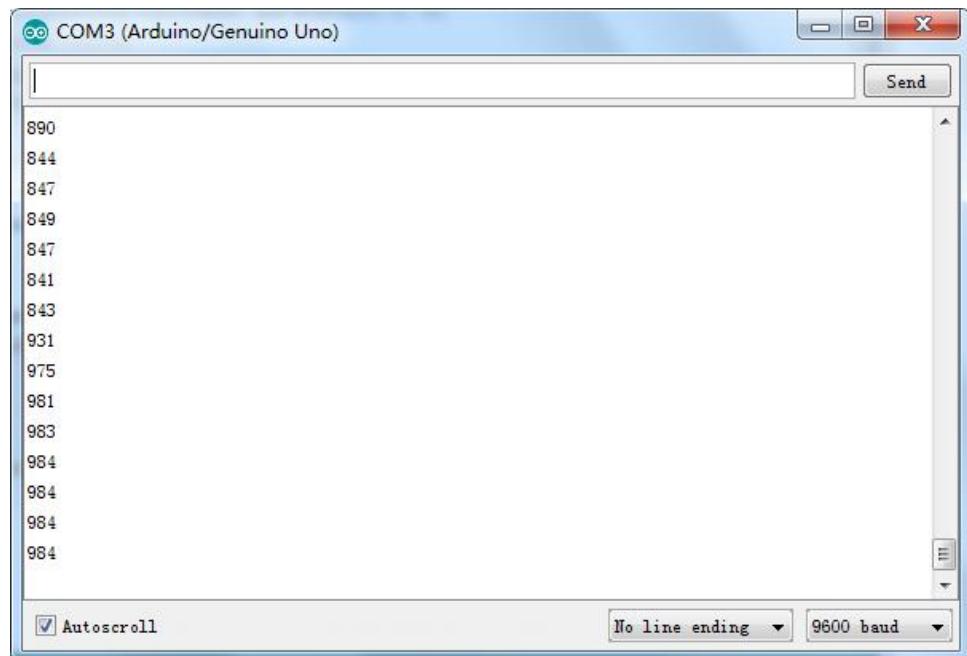
The screenshot shows the Arduino IDE interface with the title bar "07\_PhotoresistorModule | Arduino 1.8.1". A red circle highlights the "Upload" button (a blue arrow pointing right) in the toolbar. The code editor contains the following sketch:

```
1 //*****  
2 File name: _07_PhotoresistorModule.ino  
3 Description: We measure the light intensity of information with  
4 photosensitive resistance, and displayed in the  
5 serial monitor.  
6 Website: www.adeept.com  
7 E-mail: support@adeept.com  
8 Author: Tom  
9 Date: 2017/03/12  
10 *****  
11 int photoresistorPin = 0; // photoresistor connected to analog pin 0  
12 void setup ()  
13 {  
14   pinMode(photoresistorPin, INPUT); //Set analog 0 port mode, the INPUT for the input  
15   Serial.begin(9600); //opens serial port, sets data rate to 9600 bps  
16 }  
17 void loop ()  
  
Done uploading.  
Sketch uses 2014 bytes (6%) of program storage space. Maximum is 32256 bytes.  
Global variables use 186 bytes (9%) of dynamic memory, leaving 1862 bytes for local variables.
```

The status bar at the bottom right indicates "Arduino/Genuino Uno on COM3".

Open the Serial Monitor in Arduino IDE. Change the intensity of light shone on the photoresistor module and you can see the value displayed on the window changes. Basically, when you shine the stronger light on the module, the value will become greater; when the light dims slowly on it, the value decreases.





# Adeept

# Lesson 8 How To Use The Thermistor

## Introduction

Thermistors can be divided into two types by temperature coefficient: positive temperature coefficient (PTC) and negative temperature coefficient (NTC). The typical feature of a thermistor is sensitive to temperature – its resistance varies with ambient temperature changes. For PTC thermistor, when the temperature gets high, its resistance increases; for NTC thermistor, the case is the opposite. The thermistor we use is an NTC one.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Thermistor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

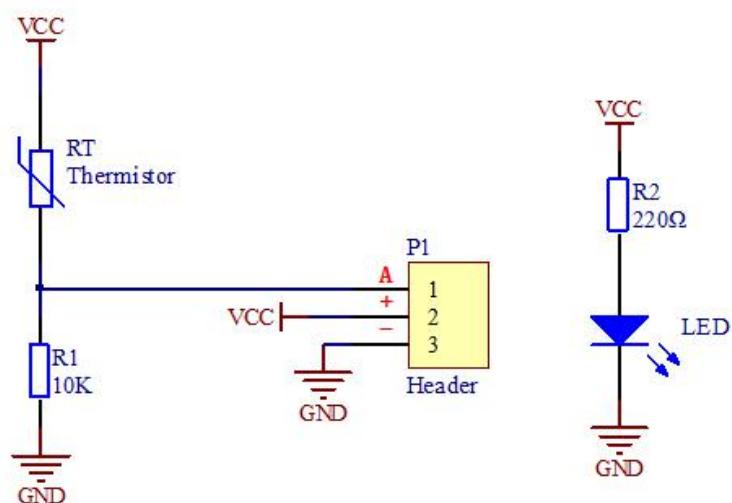
The Fritzing image:



Pin definition:

A	Analog output
+	VCC
-	GND

The schematic diagram:



The key parameters of an MF52 thermistor:

**B-parameter: 3470.**

**25°C resistance: 10kΩ.**

The relationship between the resistance of thermistor and temperature is as follows:

$$R_{thermistor} = R * e^{\left(\frac{B}{T_1} - \frac{1}{T_2}\right)}$$

**R<sub>thermistor</sub>**: the resistance of the thermistor at temperature T1

**R**: the nominal resistance of the thermistor at room temperature T2

**e**: 2.718281828459

**B**: one of the important parameters of thermistor

**T<sub>1</sub>**: the Kelvin temperature that you want to measure

**T<sub>2</sub>**: Under the condition of a 25 °C (298.15K) room temperature, the standard resistance of MF52 thermistor is 10K;

Kelvin temperature = 273.15 (absolute temperature) + degrees Celsius;

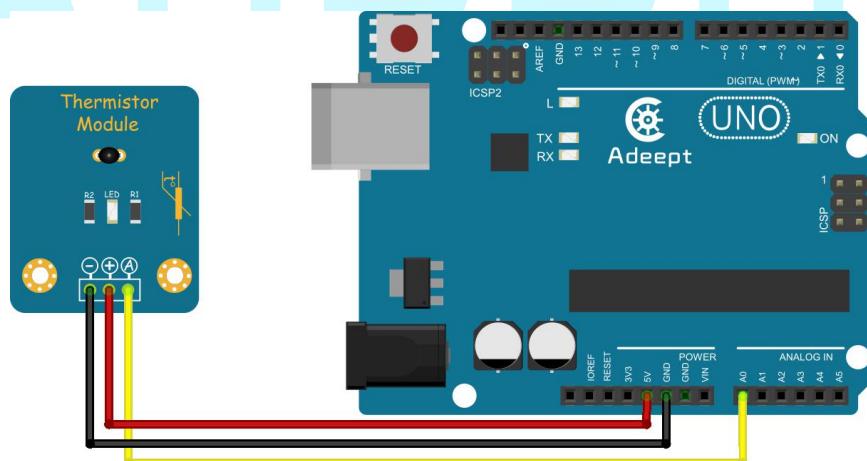
After transforming the above equation, we can get the following formula:

$$T_1 = \frac{B}{\ln\left(\frac{R_{thermistor}}{R}\right) + \frac{B}{T_2}}$$

In this experiment, by programming the Arduino, we collect the analog values output by the Thermistor module through pin A0 of the Arduino board, change it to temperature values and display them on Serial Monitor.

## Experimental Procedures

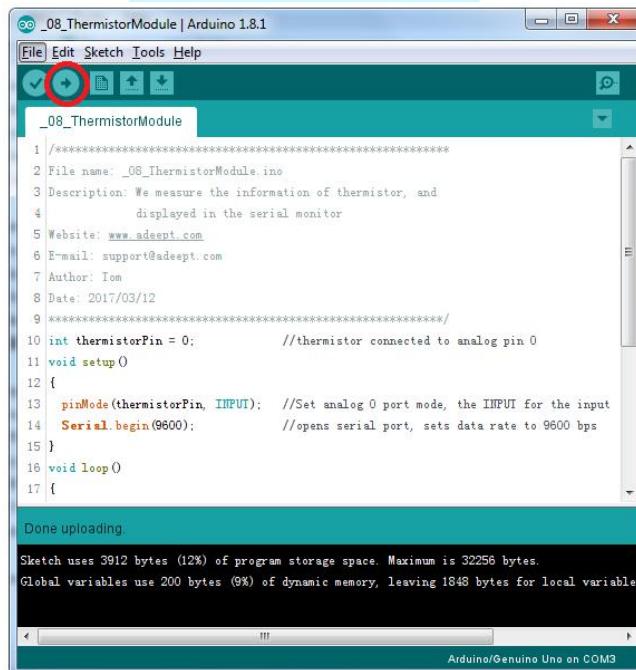
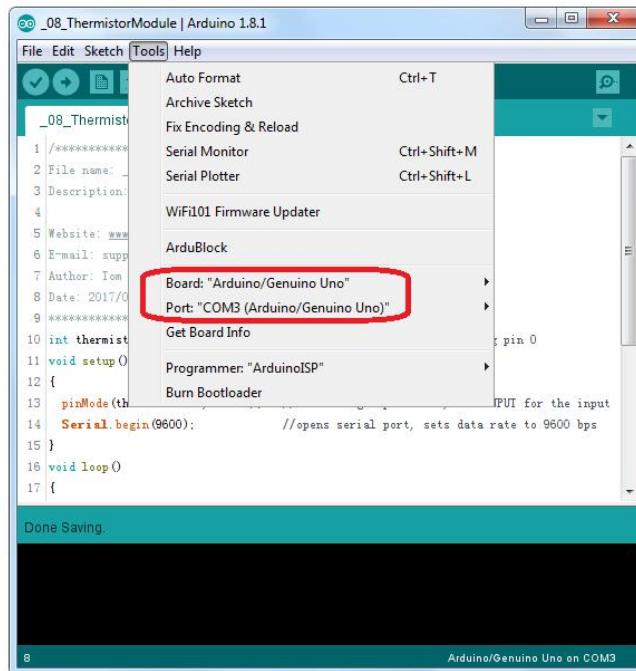
**Step 1: Build the circuit**



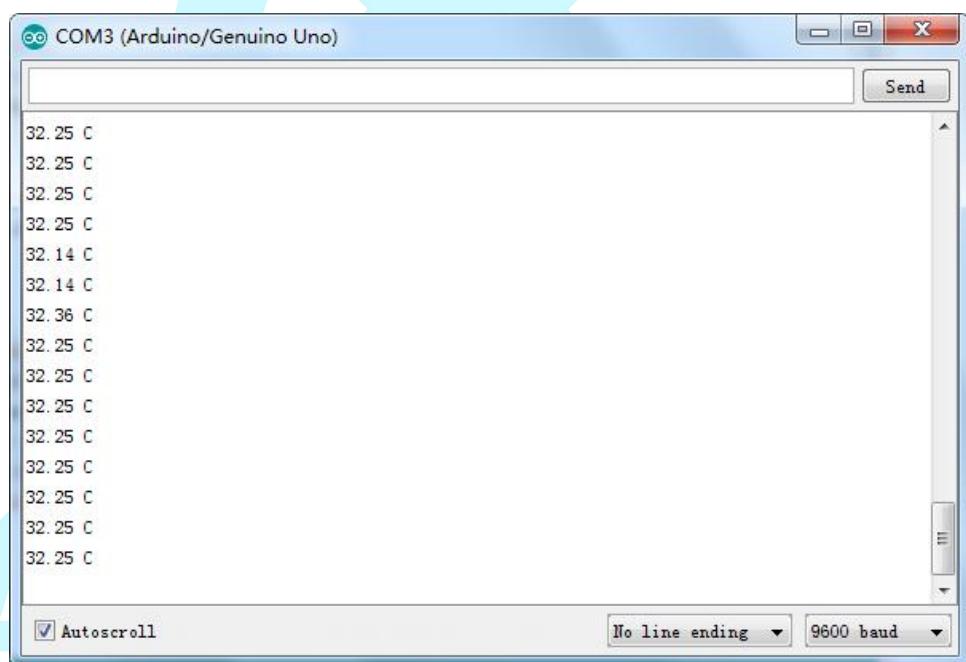
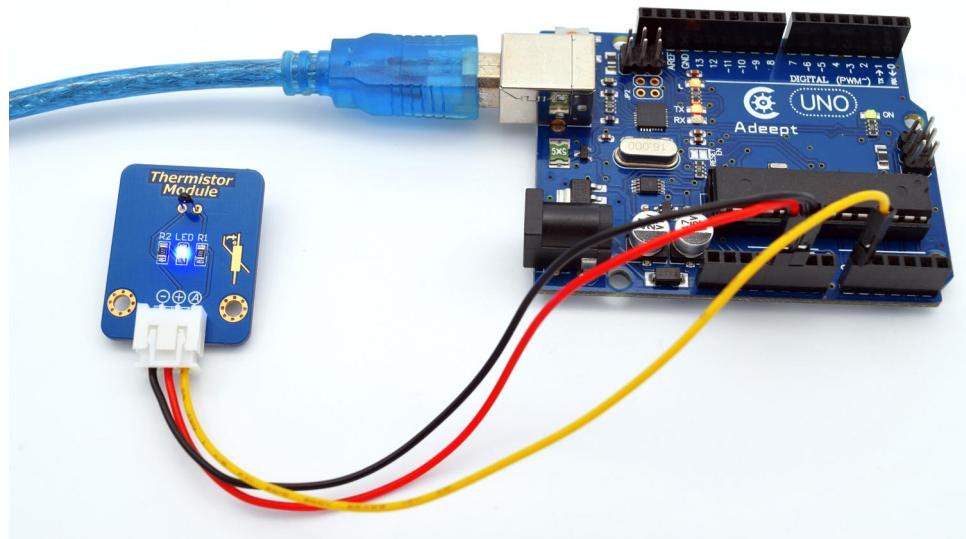
Adept UNO R3 Board	Thermistor Module
A0	A
5V	+
GND	-

**Step 2:** Program \_08\_ThermistorModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



Open the Serial Monitor in Arduino IDE and you can see the current temperature on the window.



# Lesson 9 How To Use The DS18B20

## Introduction

DS18B20 is a single-bus digital temperature sensor of high-precision. The measurement range is  $-55^{\circ}\text{C}$  -  $+125^{\circ}\text{C}$  and inherent temperature resolution is  $0.5^{\circ}\text{C}$ . The sensor support multi-point network and multi-point temperature measurement – the measured result is sent to the controller via serial port in the format of a 9-12-bit number. This digital sensor can be applied to various microcontrollers. It's simpler on Arduino boards – just call the related functions and the temperature can be measured then.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* DS18B20 Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

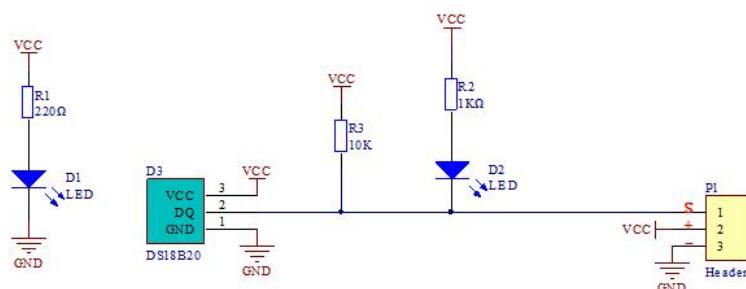
The Fritzing image:



Pin definition:

S	Digital pin
+	VCC
-	GND

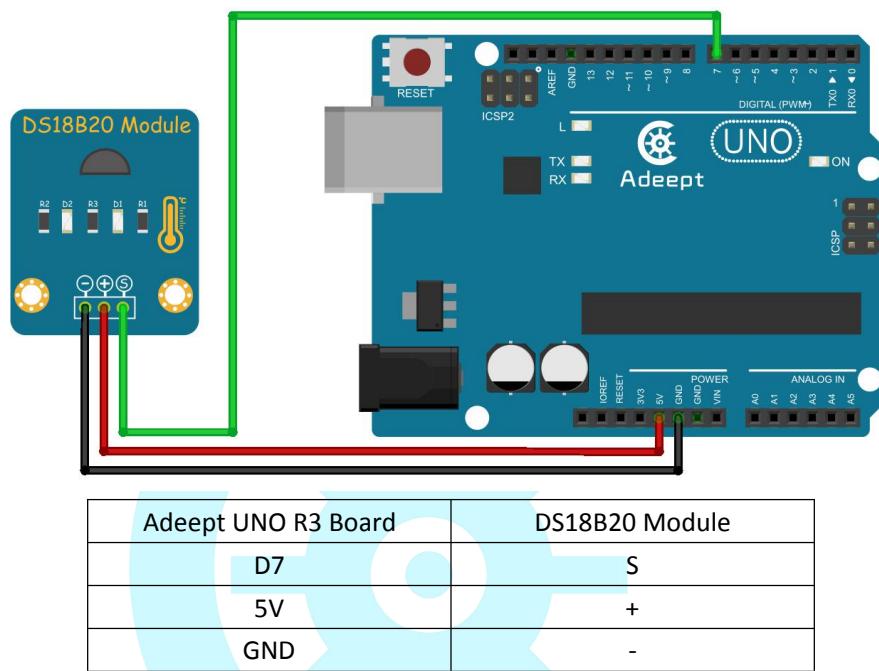
The schematic diagram:



In this experiment, by programming the Arduino, we read the temperature value collected by the DS18B20 module through pin D7 of the Arduino board, and display it on Serial Monitor.

## Experimental Procedures

### Step 1: Build the circuit



**Step 2:** Install the function library (Arduino-Temperature-Control-Library-master.zip and OneWire.zip).

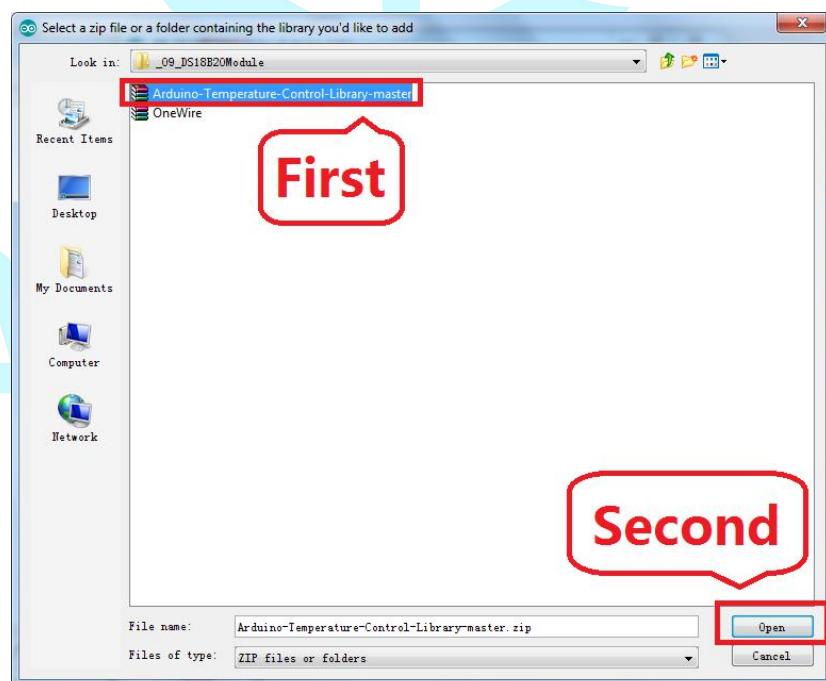
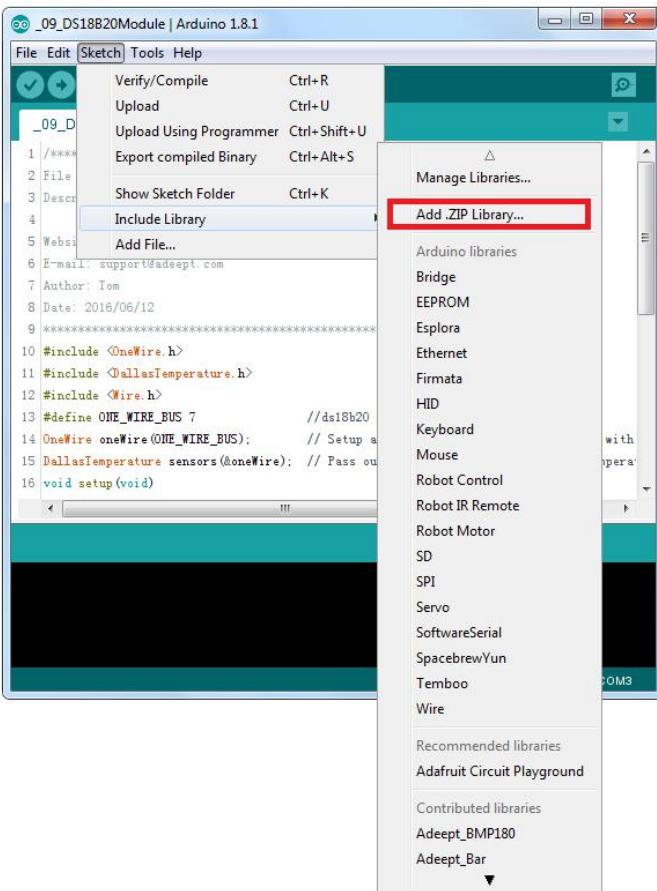


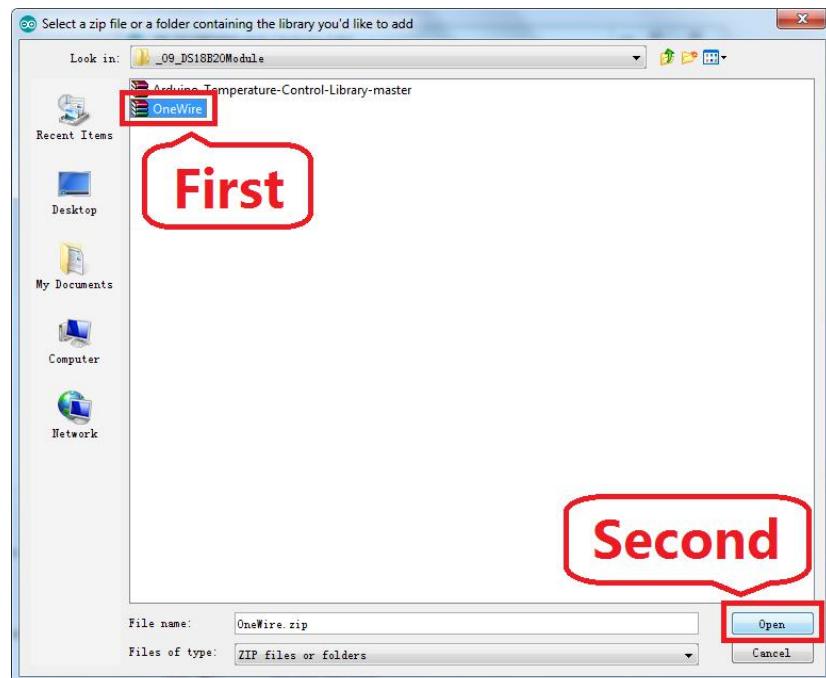
Arduino-Temperature-Control-Library-master



OneWire

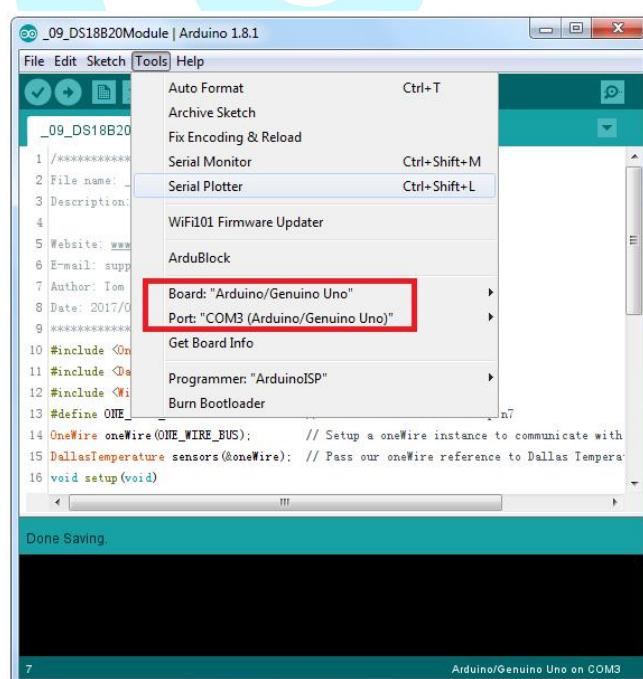
Adeept

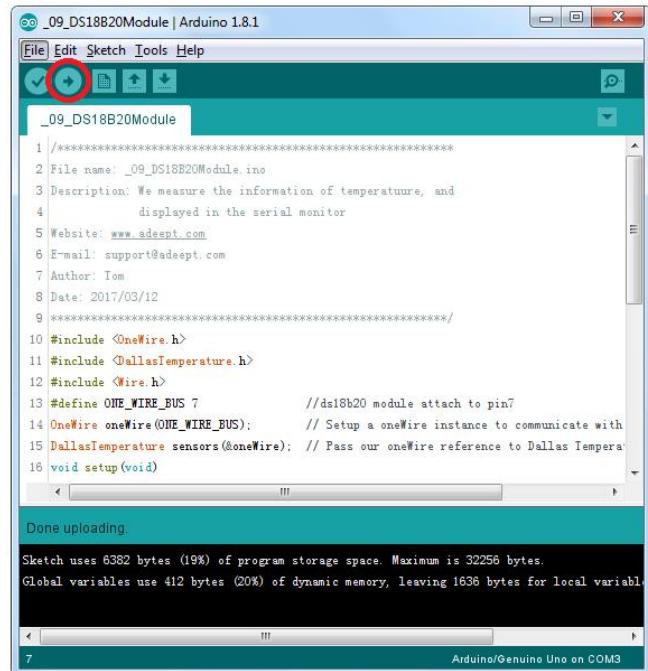




**Step 3:** Program `_09_DS18B20Module.ino`

**Step 4:** Compile and download the sketch to the UNO R3 board.



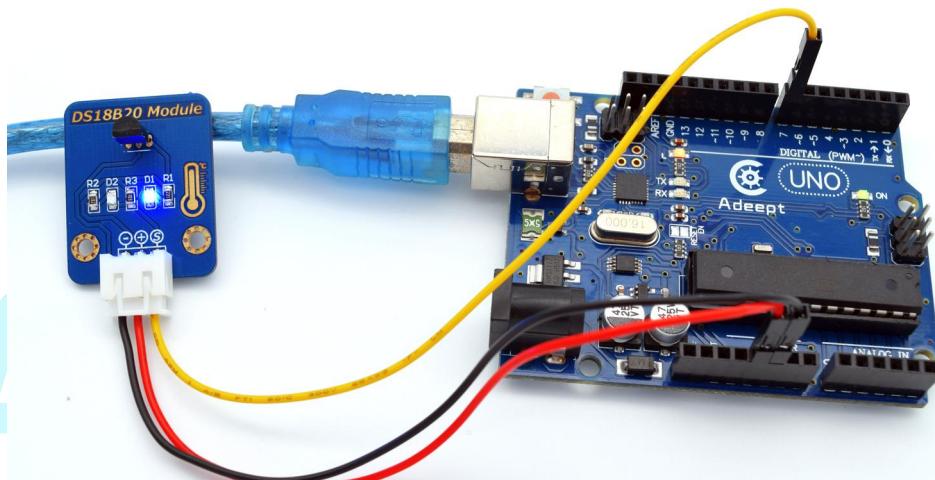


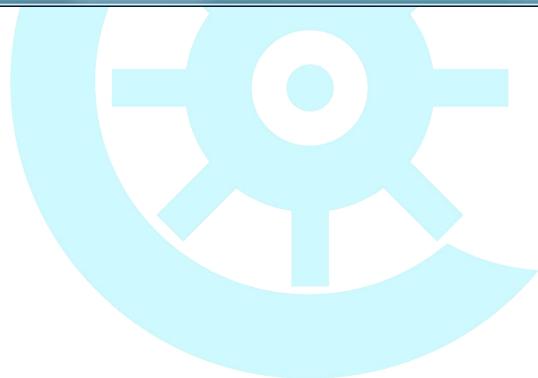
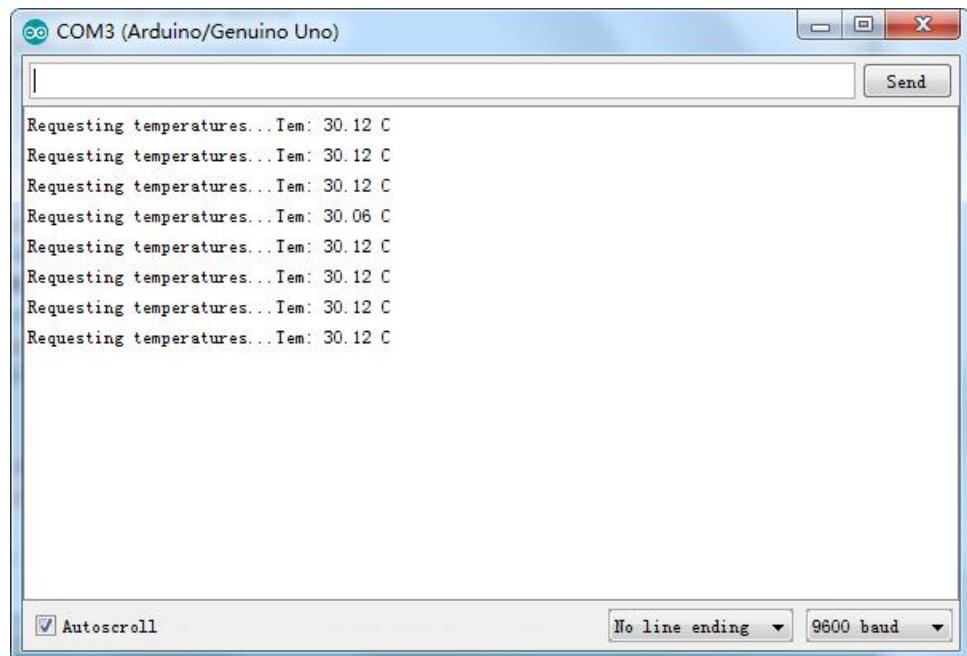
The screenshot shows the Arduino IDE interface with the title bar "09\_DS18B20Module | Arduino 1.8.1". The menu bar includes File, Edit, Sketch, Tools, and Help. A red circle highlights the "Upload" button (a blue arrow pointing right) in the toolbar. The code editor contains a sketch named "09\_DS18B20Module" with comments and code related to the DS18B20 module. The status bar at the bottom right indicates "Arduino/Genuine Uno on COM3". The message area shows "Done uploading." and details about memory usage.

```
09_DS18B20Module | Arduino 1.8.1
File Edit Sketch Tools Help
09_DS18B20Module
1 //*****
2 File name: _09_DS18B20Module.ino
3 Description: We measure the information of temperature, and
4 displayed in the serial monitor
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/12
9 ****
10 #include <OneWire.h>
11 #include <DallasTemperature.h>
12 #include <Wire.h>
13 #define ONE_WIRE_BUS 7          //ds18b20 module attach to pin7
14 OneWire oneWire(ONE_WIRE_BUS); // Setup a oneWire instance to communicate with
15 DallasTemperature sensors(&oneWire); // Pass our oneWire reference to Dallas Temperature
16 void setup(void)
17 {
18 }
19 void loop(void)
20 {
21     sensors.requestTemperatures();
22     float tempC = sensors.getTempCByIndex(0);
23     Serial.print("Temperature: ");
24     Serial.print(tempC);
25     Serial.println(" C");
26 }
```

Done uploading.  
Sketch uses 6382 bytes (19%) of program storage space. Maximum is 32256 bytes.  
Global variables use 412 bytes (20%) of dynamic memory, leaving 1636 bytes for local variables.

Open the Serial Monitor in Arduino IDE and you can see the current temperature on the window.





# Adeept

# Lesson 10 Alarm Prompt

## Introduction

Buzzers are a type of integrated electronic alarm devices and powered by DC supply. They are widely applied for sound producing in devices such as computer, printer, duplicator, alarm, electronic toy, vehicle electronic equipment, phone, and timer and so on. Active buzzers can make sounds constantly when connected with a 5V DC supply. This Active Buzzer module is connected to a digital pin of the Arduino Uno board. When the pin outputs High level, the buzzer will beep; when the pin gives Low, it stays dumb.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Active Buzzer Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

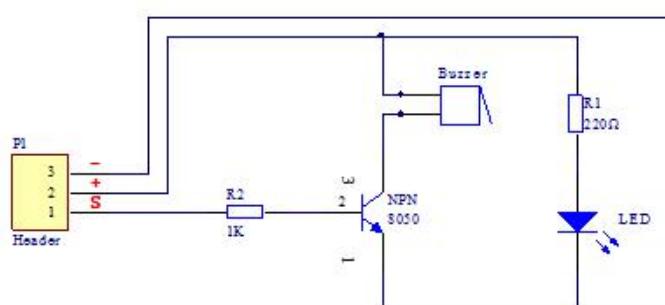
The Fritzing image:



Pin definition:

S	Digital input
+	VCC
-	GND

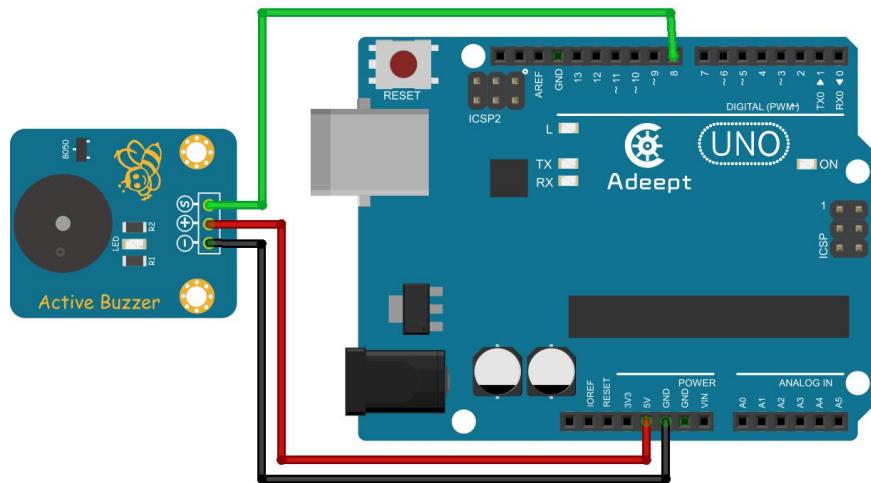
The schematic diagram:



In this experiment, by programming the Arduino, we make the pin D8 of the Arduino board output High and Low alternately, so the active buzzer makes sounds accordingly.

## Experimental Procedures

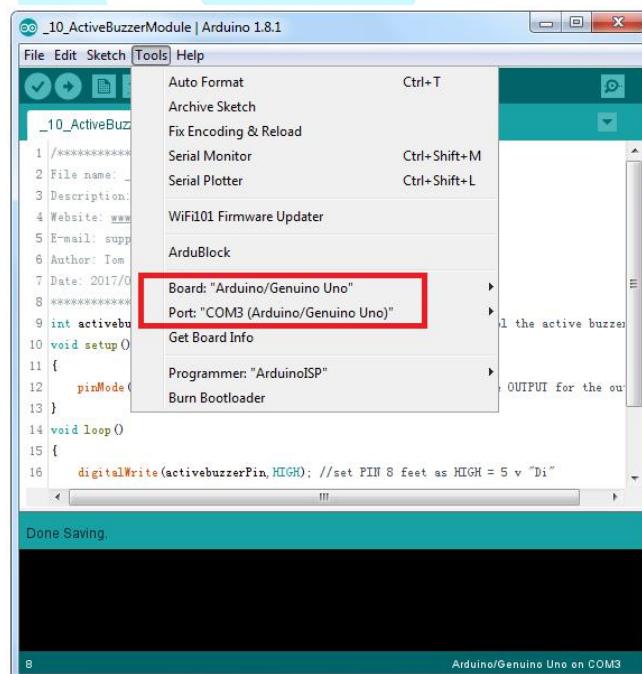
**Step 1:** Build the circuit

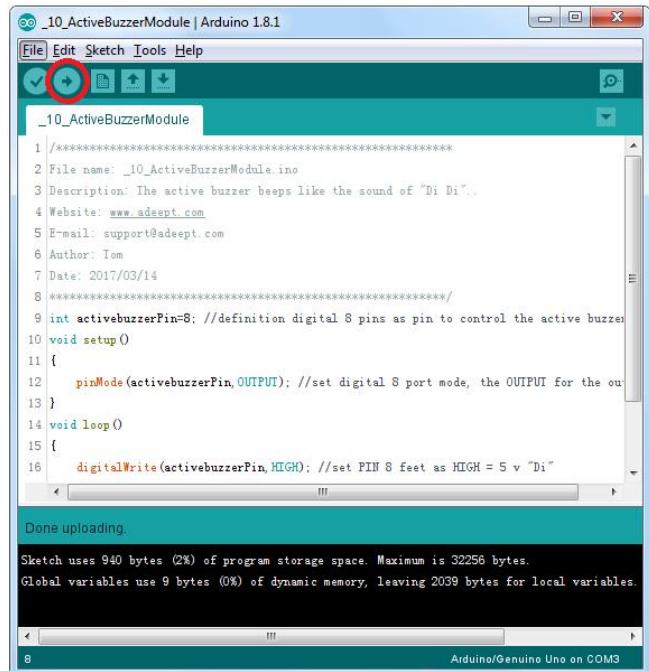


Adeept UNO R3 Board	Active Buzzer Module
D8	S
5V	+
GND	-

**Step 2:** Program `_10_ActiveBuzzerModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



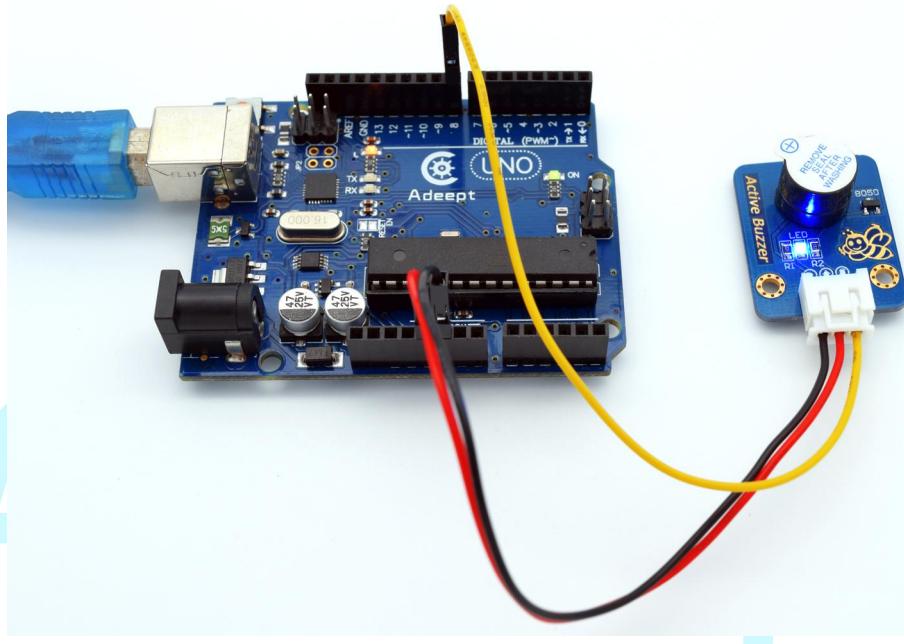


```
_10_ActiveBuzzerModule | Arduino 1.8.1
File Edit Sketch Tools Help
1 //*****
2 File name: _10_ActiveBuzzerModule.ino
3 Description: The active buzzer beeps like the sound of "Di Di".
4 Website: www.adeept.com
5 E-mail: support@adeept.com
6 Author: Iom
7 Date: 2017/03/14
8 ****
9 int activebuzzerPin=8; //definition digital 8 pins as pin to control the active buzzer
10 void setup()
11 {
12     pinMode(activebuzzerPin, OUTPUT); //set digital 8 port mode, the OUTPUT for the ou
13 }
14 void loop()
15 {
16     digitalWrite(activebuzzerPin, HIGH); //set PIN 8 feet as HIGH = 5 v "Di"
}
Done uploading.

Sketch uses 940 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.

8 Arduino/Genuino Uno on COM3
```

Now you can hear the active buzzer beeps like the sound of "Di Di".



# Lesson 11 Playing Music

## Introduction

The difference between an active buzzer and a passive one radically lies in the requirement for input signals. The ideal signals for active buzzers are direct currents, usually marked with VCC or VDD. Inside them there are a simple oscillation circuit that can convert constant direct currents into pulse signal of a certain frequency, causing magnetic fields alternation and then Mo sheet vibrating and making sounds. On the other hand, there is no driving circuit in a passive buzzer. So the ideal signal for passive buzzer is square wave. If DC is given, it will not respond since the magnetic field is unchanged, the Mo sheet cannot vibrate and produce sounds.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Passive Buzzer Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

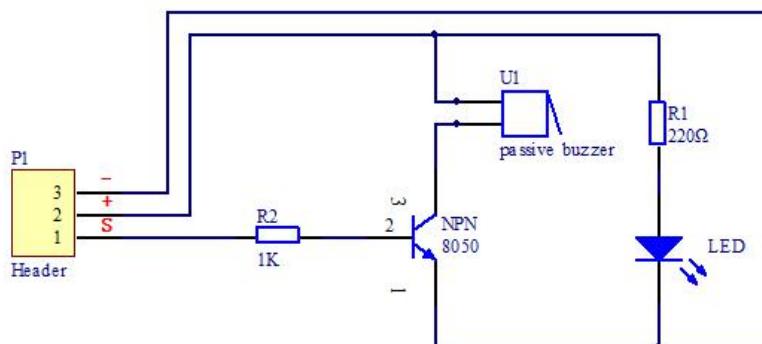
The Fritzing image:



Pin definition:

S	Digital input
+	VCC
-	GND

The schematic diagram:

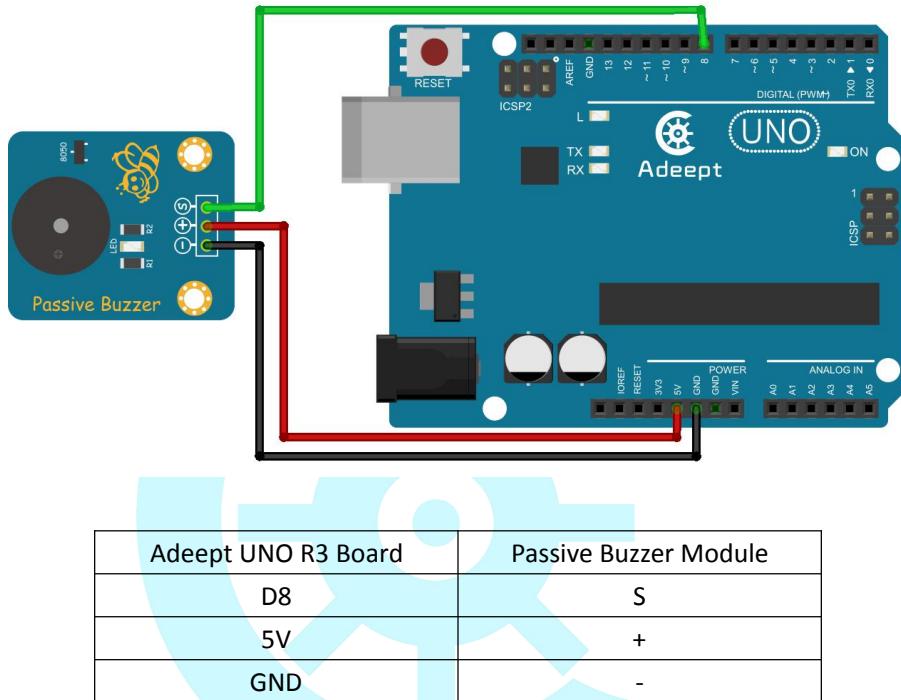


In this experiment, by programming the Arduino, we make the pin D8 of the Arduino board

output square waves of different frequencies alternately, thus driving the passive buzzer to play music.

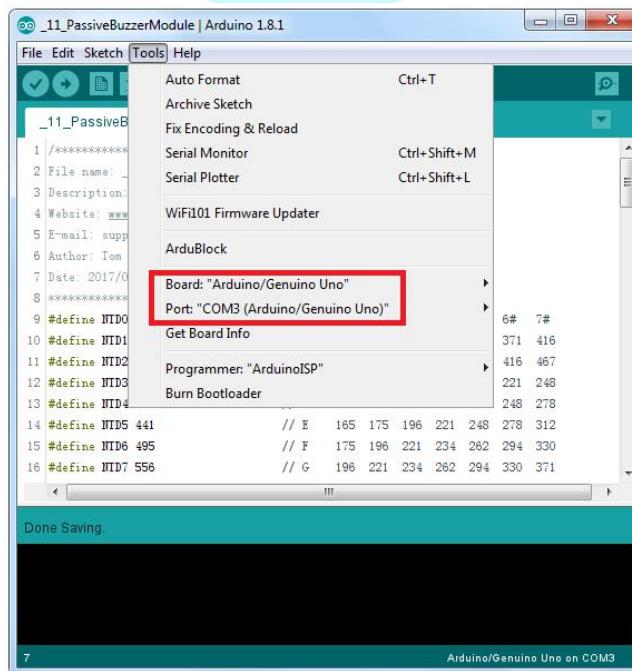
## Experimental Procedures

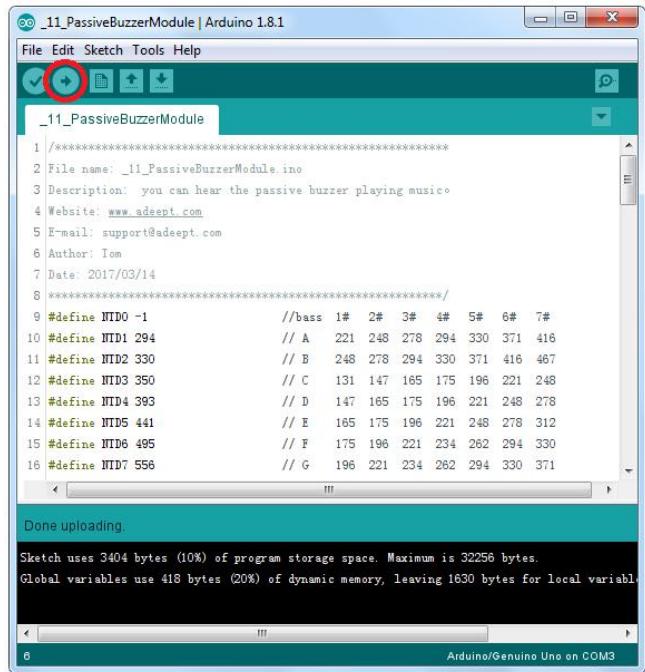
### Step 1: Build the circuit



**Step 2:** Program `_11_PassiveBuzzerModule.ino`

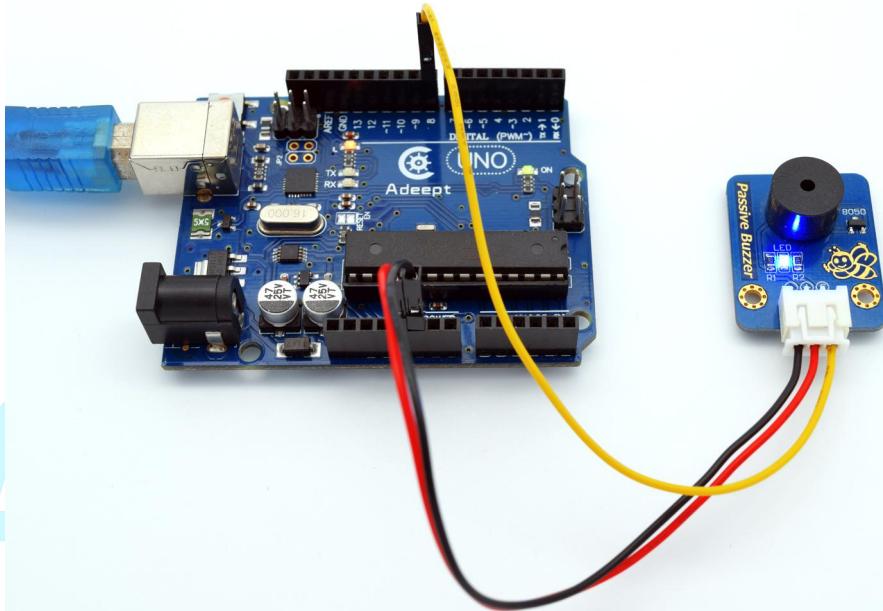
**Step 3:** Compile and download the sketch to the UNO R3 board.





The screenshot shows the Arduino IDE interface with the title bar "11\_PassiveBuzzerModule | Arduino 1.8.1". The menu bar includes File, Edit, Sketch, Tools, and Help. A red circle highlights the "Upload" button (a blue arrow pointing right) in the toolbar. The code editor contains the sketch code, which defines frequency values for various notes (A, B, C, D, E, F, G) in Hz. Below the code, a message says "Done uploading." and provides memory usage details: "Sketch uses 3404 bytes (10%) of program storage space. Maximum is 32256 bytes. Global variables use 418 bytes (20%) of dynamic memory, leaving 1630 bytes for local variables." The status bar at the bottom indicates "Arduino/Genuino Uno on COM3".

Now you can hear the passive buzzer play music.



# Lesson 12 Detection of Human Body Movement

## Introduction

The PIR (passive Infrared) sensor can detect the Infrared rays emitted by human or animals and then output On/Off signals. Traditional pyroelectric PIR sensor needs pyroelectric Infrared probe, special chip and complex circuits to make the effect. In this case, the sensor is a large one with complicated circuits and comparatively less credible. But this Infrared pyroelectric motion sensor adopts the digital integration of the probe, so it boasts high credibility, low power consumption and simple outside circuit with a small size. It can be applied to any cases in detecting moving human or animals.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* PIR Sensor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

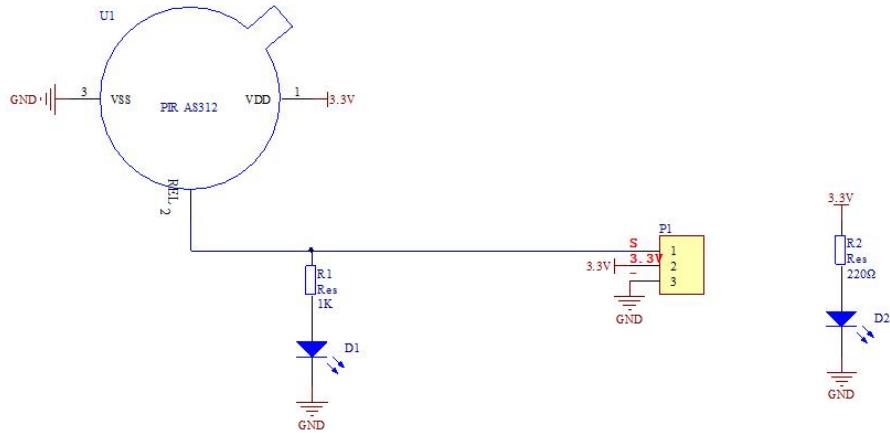
The Fritzing image:



Pin definition:

S	Digital output
3.3V	3.3V
-	GND

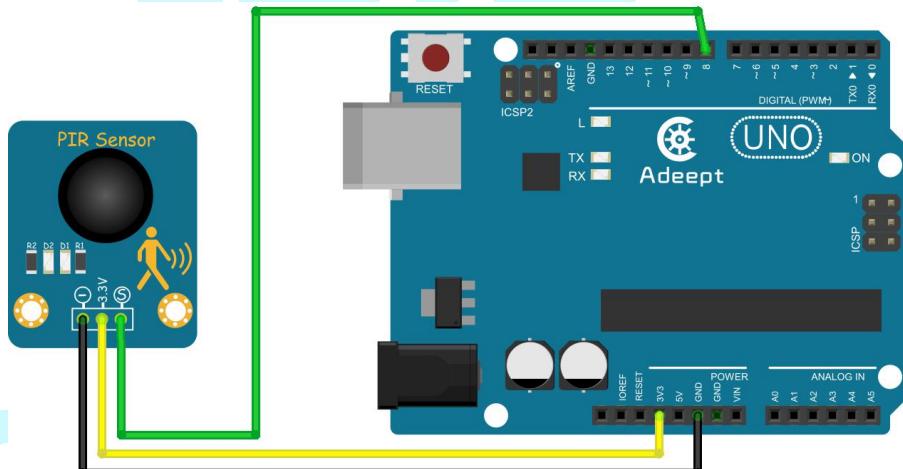
The schematic diagram:



This experiment is to use the PIR Sensor Module to detect whether there is any human activity and send the data to computer via serial port, and then display it on Serial Monitor.

## Experimental Procedures

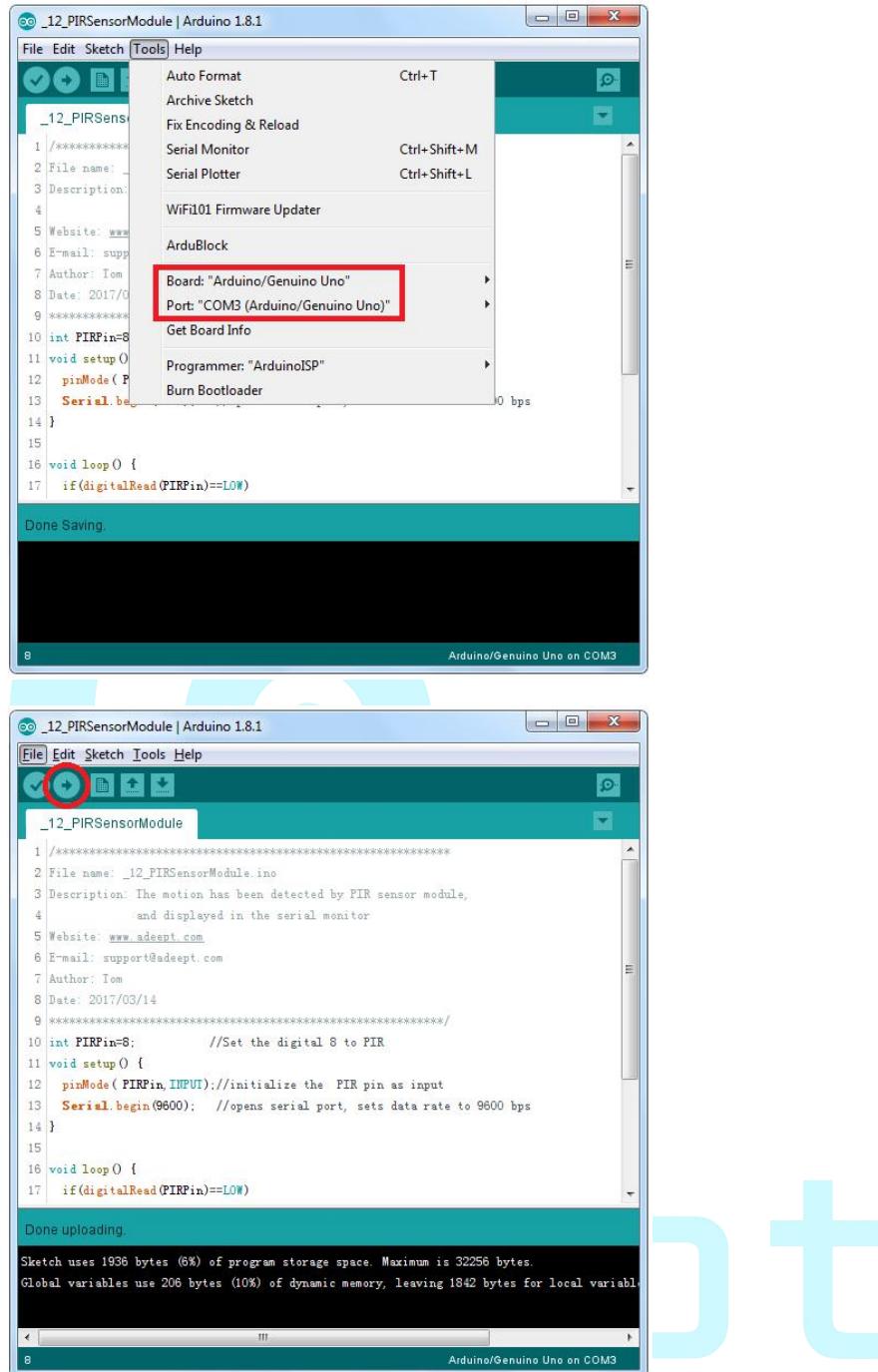
### Step 1: Build the circuit



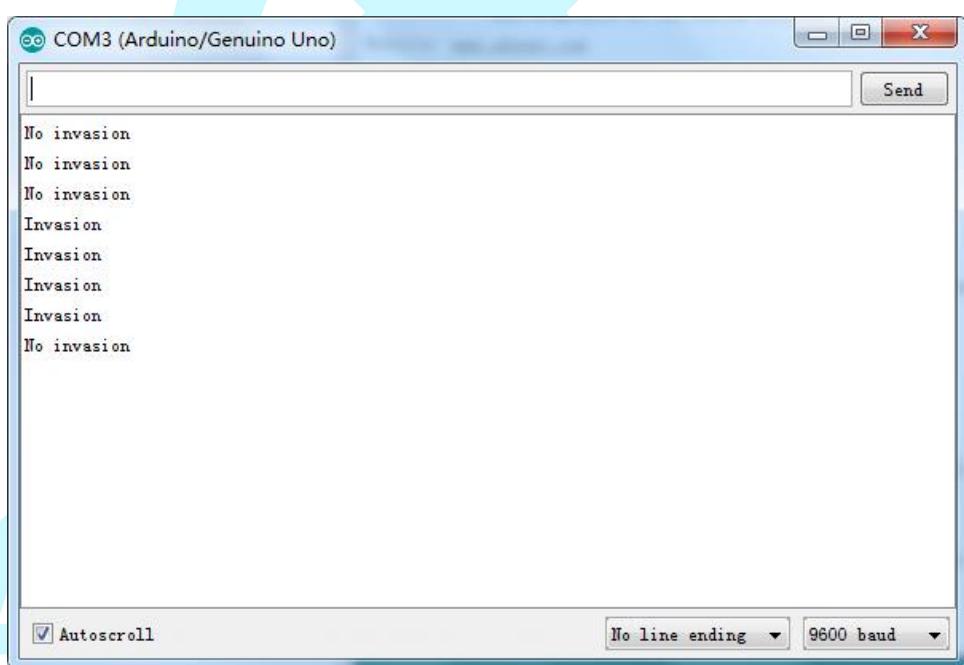
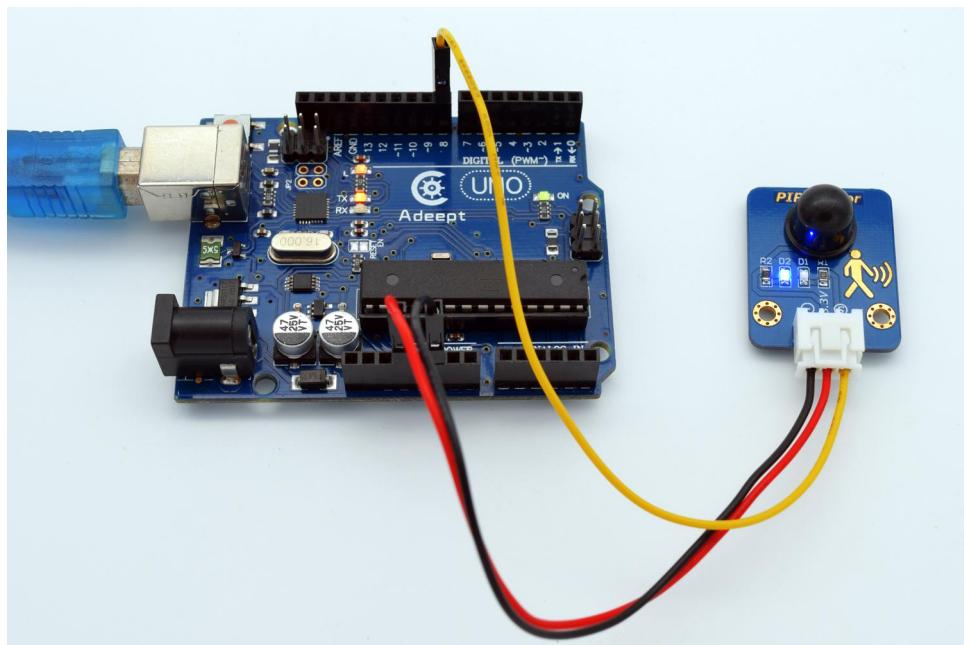
Adeept UNO R3 Board	PIR Sensor Module
D8	S
3.3V	3.3V
GND	-

**Step 2:** Program `_12_PIRSensorModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



Open the Serial Monitor in Arduino IDE. When the PIR module detects human movement, "Invasion!" will be displayed on the window; otherwise, "No invasion".



# Lesson 13 How To Use The LCD1602

## Introduction

1602 crystal, or 1602 character crystal, is a dot-matrix crystal module used specially to display letters, numbers, symbol, etc. It's composed of several dot-matrix character bits, each of which can display one character. The character bits are separated by one dot pitch and there's a gap between each line. Therefore, characters are spaced within and between lines.

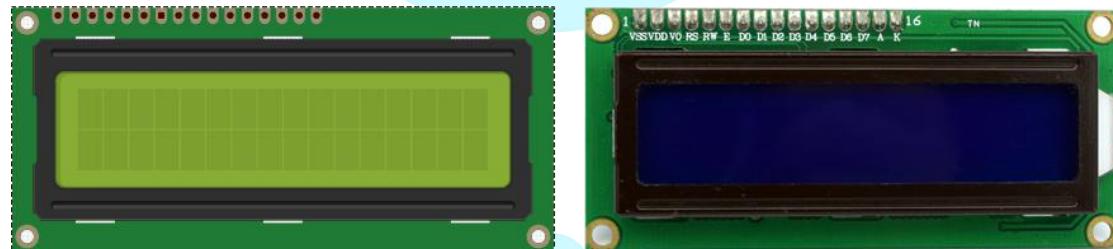
The name LCD 1602 indicates that the display is 16x2, that is, two lines with 16 characters in each.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* LCD1602 Display Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires
- 2 \* Hookup Wire Set
- 12 \* Male to Female Jumper Wires
- 1 \* Breadboard

## Experimental Principle

The Fritzing image:



Pin definition:

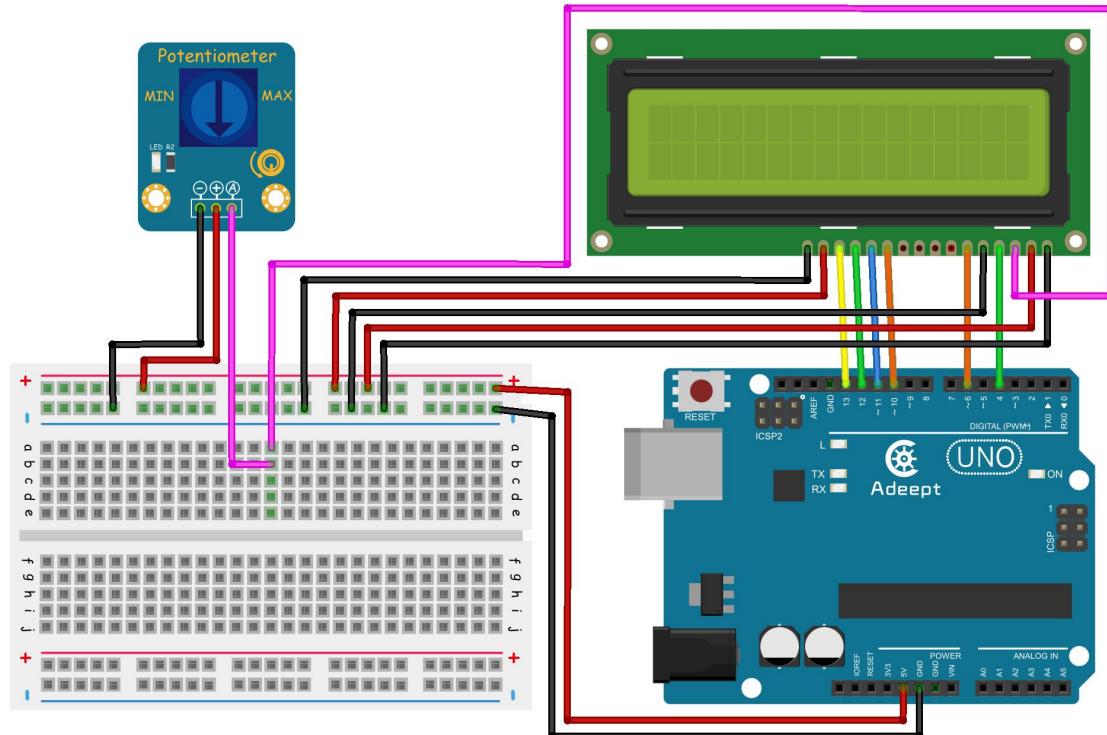
VSS	GND
VDD	VCC
VO	Analog input
RS	Digital input
RW	Digital input
E	Digital input
D0	Digital input
D1	Digital input
D2	Digital input
D3	Digital input
D4	Digital input
D5	Digital input
D6	Digital input

D7	Digital input
A	VCC
K	GND

In this experiment, by programming the Arduino, we make the LCD1602 display scrolling "Hello Geeks!" first and then "Adeept www.adeept.com" steadily.

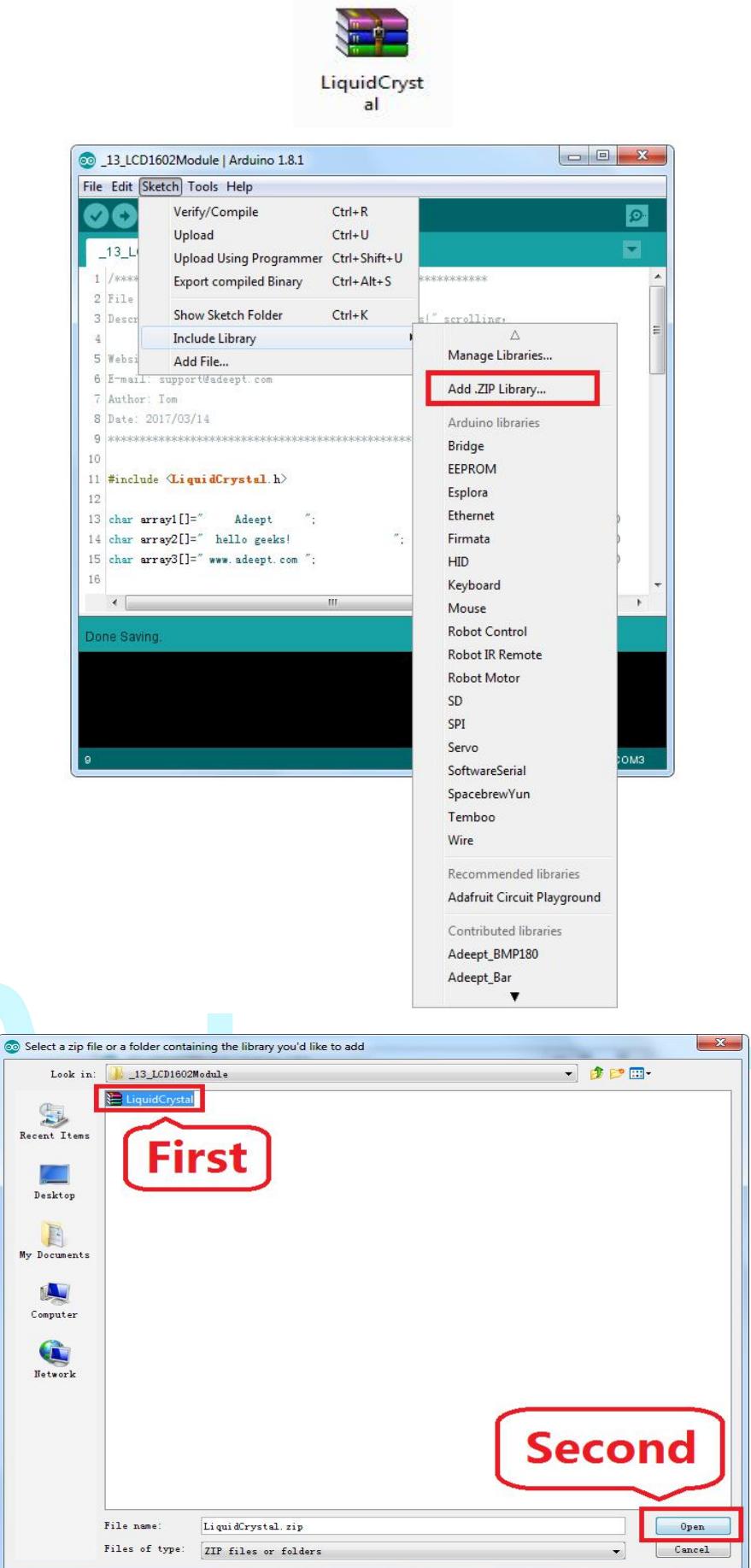
## Experimental Procedures

### Step 1: Build the circuit



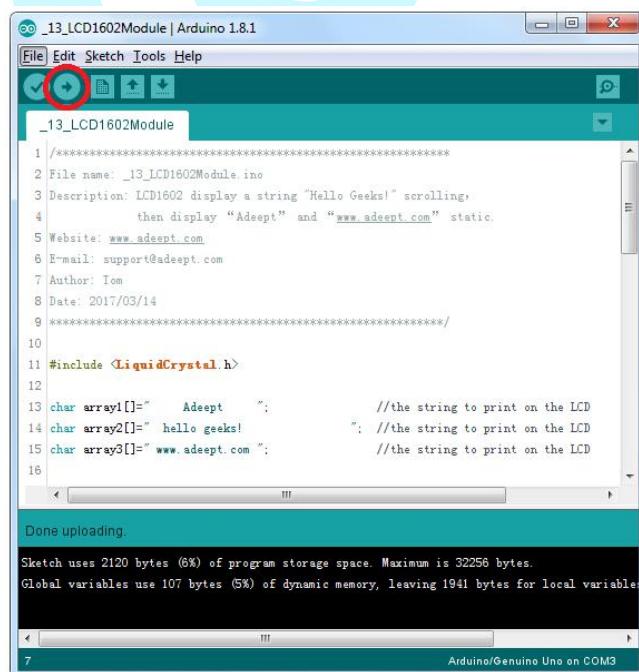
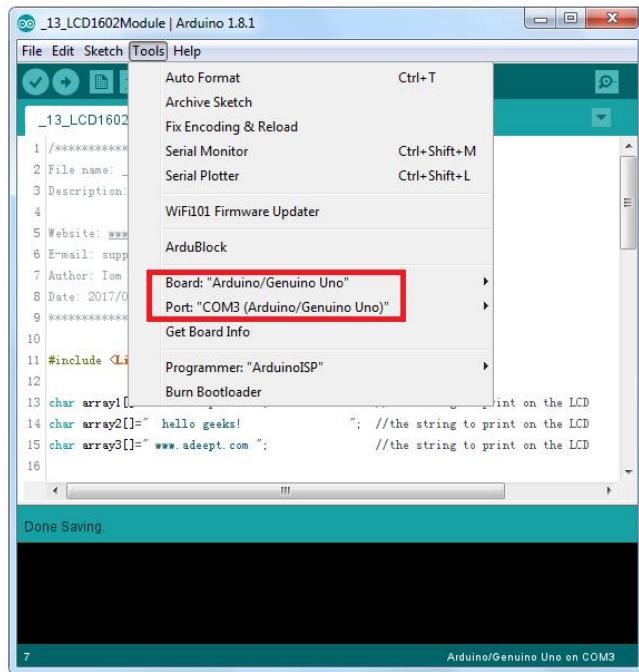
Adeept UNO R3 Board	LCD1602 Module	Potentiometer Module
GND	VSS	-
5V	VDD	+
	VO	A
D4	RS	
GND	RW	
D6	E	
D10	D4	
D11	D5	
D12	D6	
D13	D7	
5V	A	
GND	K	

**Step 2:** Install the function library (LiquidCrystal.zip).

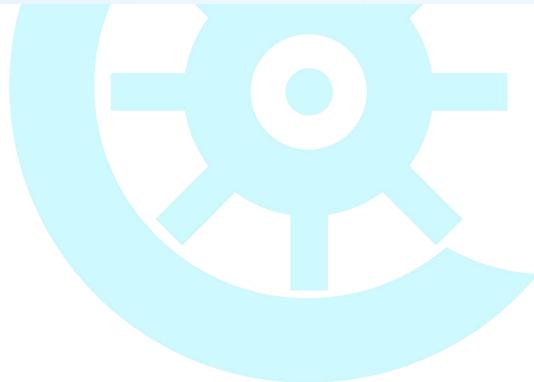
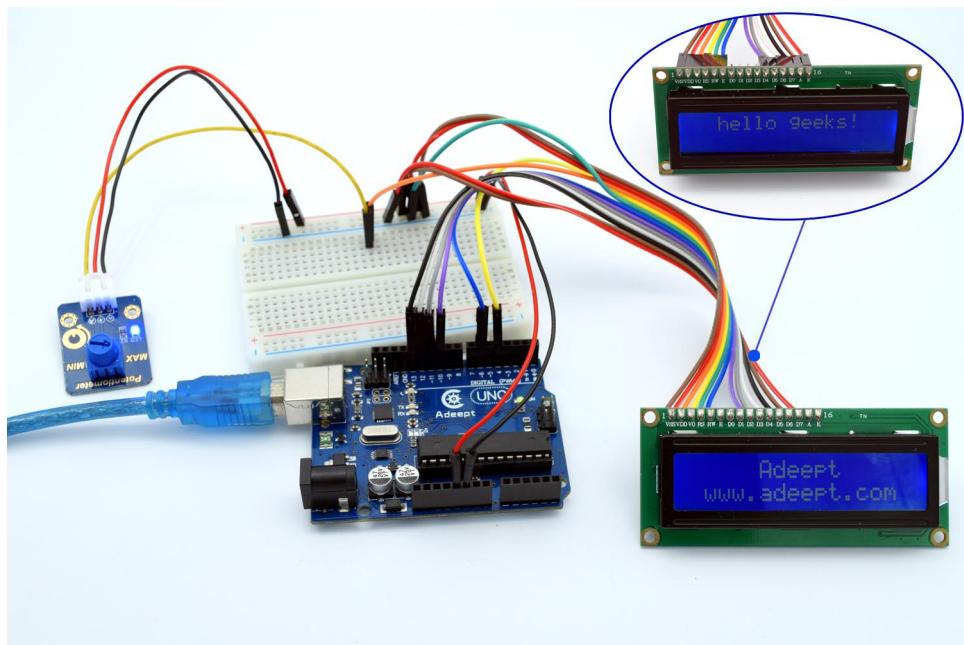


**Step 3:** Program \_13\_LCD1602Module.ino

**Step 4:** Compile and download the sketch to the UNO R3 board.



Now you can see on the LCD1602 display, "Hello Geeks!" is scrolling first, and then "Adeept www.adeept.com" appears and stays unchanged.



# Adeept

# Lesson 14 IIC Interface Application

## Introduction

Since there are limited I/O ports on the Arduino UNO R3 board, if you use them to drive the LCD1602, it needs many of the ports and there may be insufficient to connect other devices. To solve this problem, an IIC (or I2C) Interface Module based on PCF8574 is designed to extend the I/O ports of the Arduino board. You only need two wires (SDA and SCL) to control the LCD1602 and save many ports for the board to connect more sensors.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* IIC Interface Module
- 1 \* LCD1602 Display Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

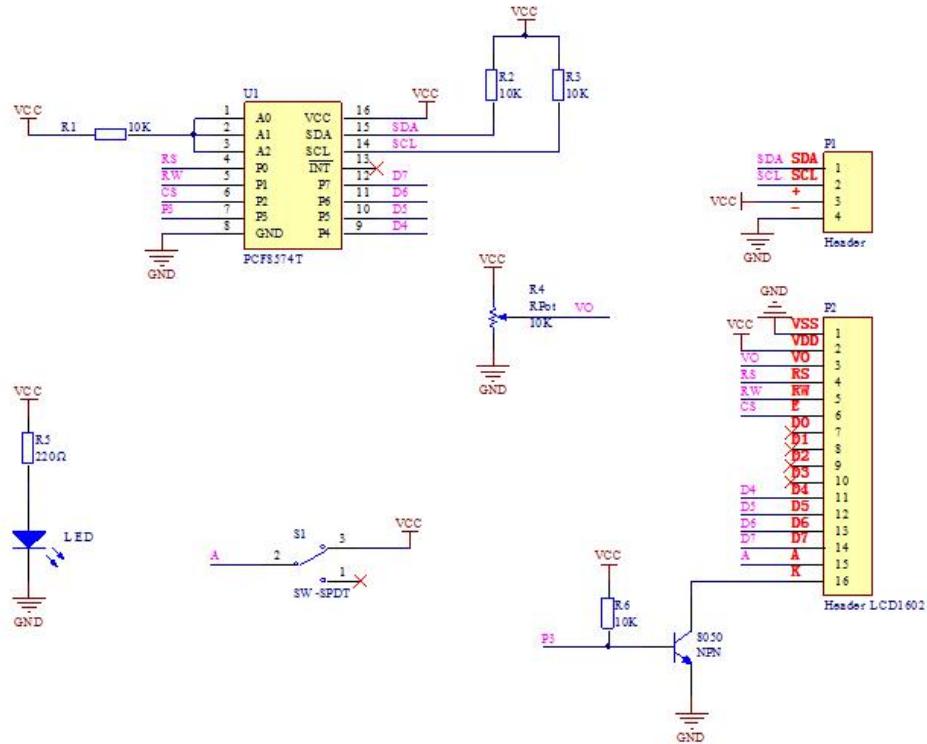
The Fritzing image:



Pin definition:

SDA	Data Pin
SCL	Clock Pin
+	VCC
-	GND

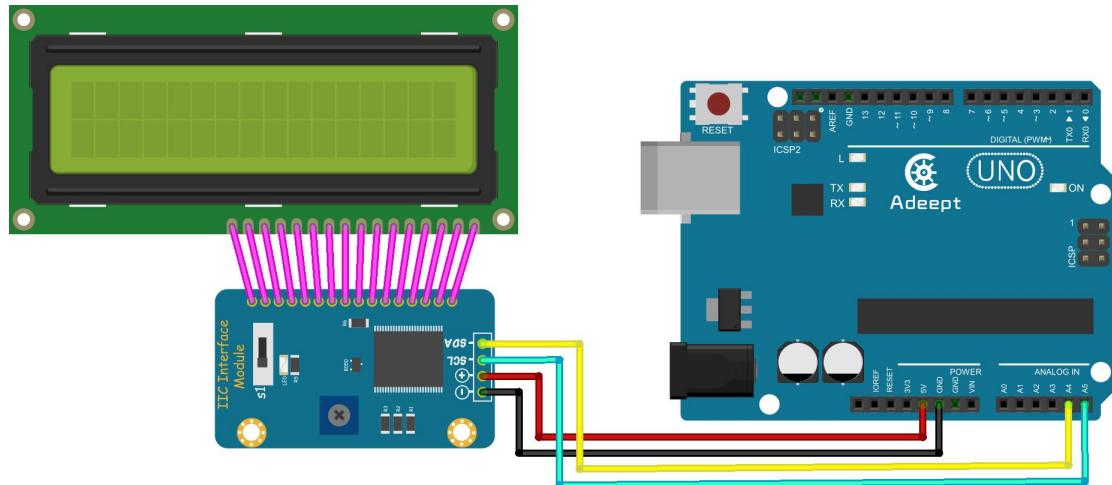
The schematic diagram:



In this experiment, by programming the Arduino, we write data to the I2C Interface Module via the I2C interface on the Arduino board so as to control the LCD1602 to display data as we want.

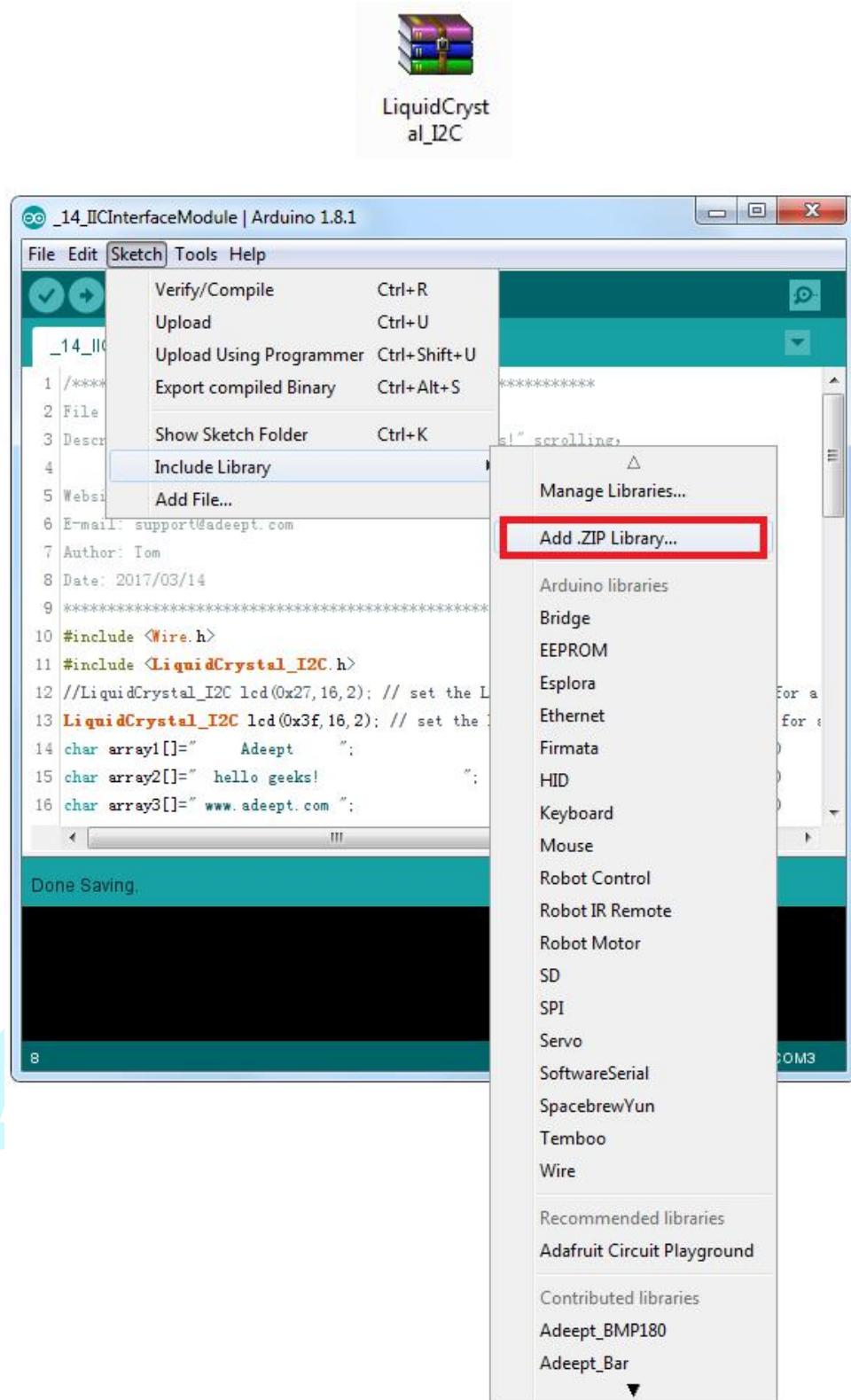
## Experimental Procedures

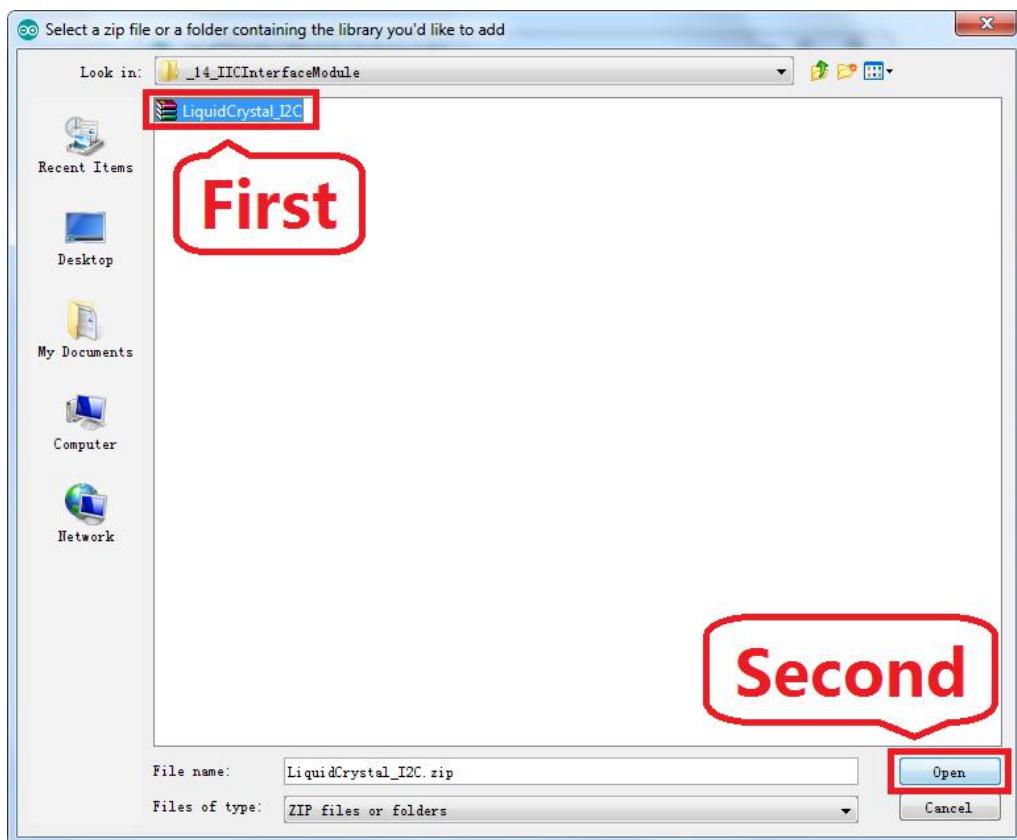
### Step 1: Build the circuit



Adeept UNO R3 Board	I2C Interface Module
A4	SDA
A5	SCL
5V	+
GND	-

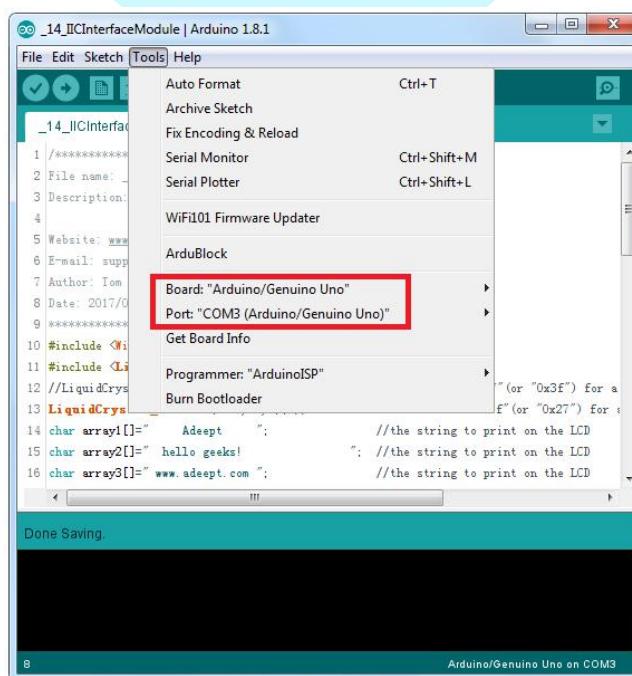
### Step 2: Install the function library (LiquidCrystal.zip).

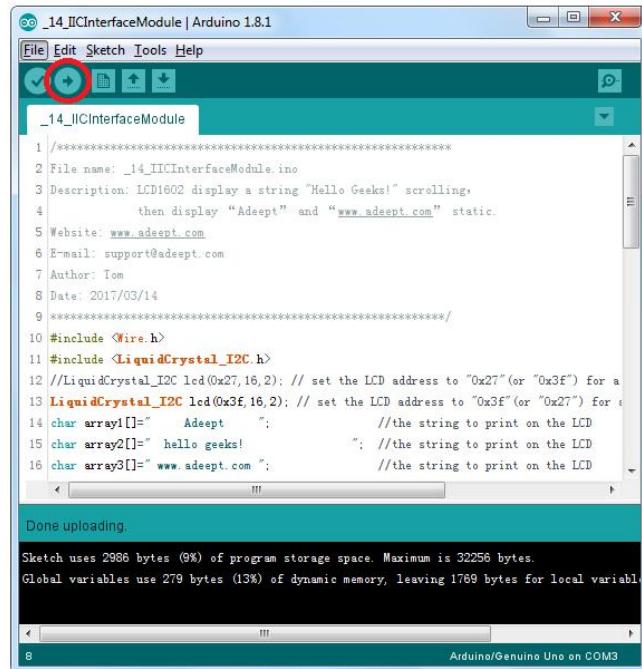




**Step 3:** Program `_14_IICInterfaceModule.ino`

**Step 4:** Compile and download the sketch to the UNO R3 board.

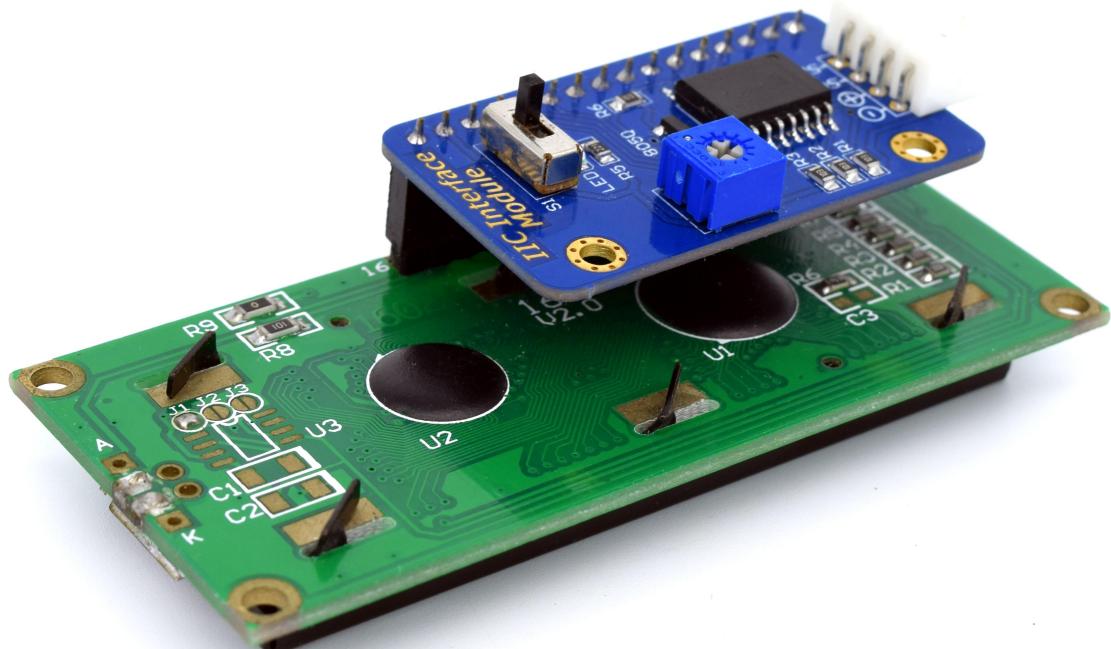


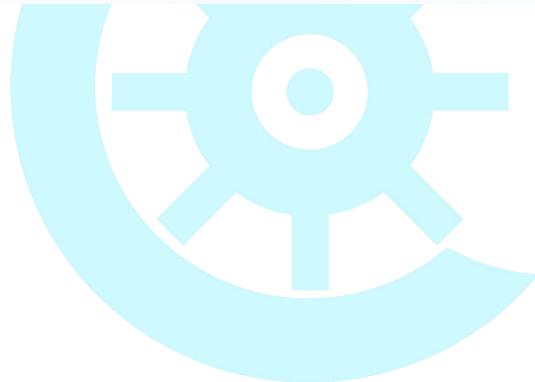
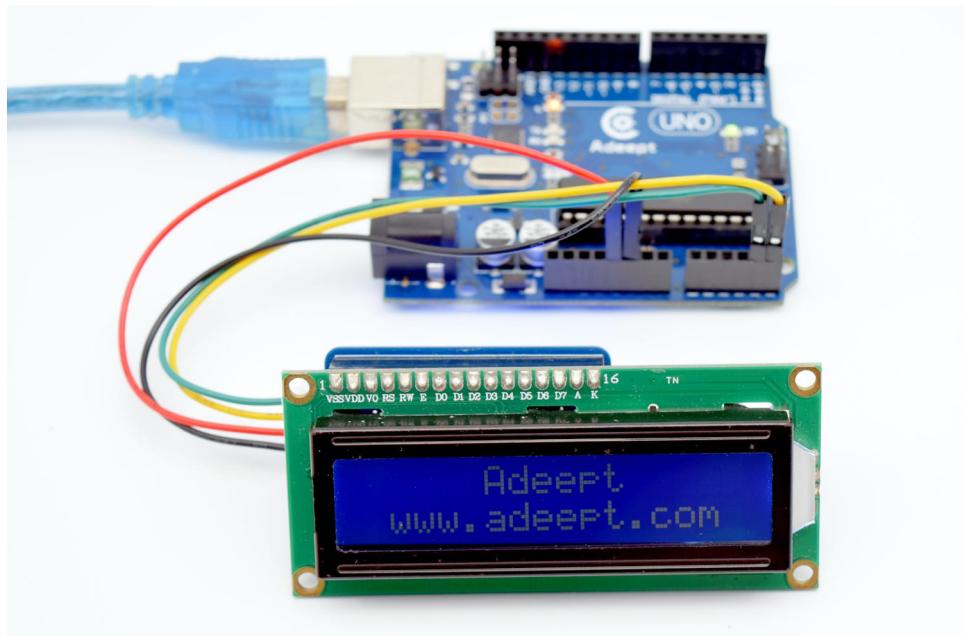


The screenshot shows the Arduino IDE interface with the title bar "14\_I2CInterfaceModule | Arduino 1.8.1". The menu bar includes File, Edit, Sketch, Tools, and Help. A red circle highlights the "Upload" button (a blue triangle icon) in the toolbar. The main window displays the code for "14\_I2CInterfaceModule.ino". The code includes comments about the LCD1602 display, the website "www.adeept.com", and the date "2017/03/14". It also includes #include statements for "Wire.h" and "LiquidCrystal\_I2C.h", and defines three character arrays: "Adeept", "hello geeks!", and "www.adeept.com". Below the code, a message says "Done uploading." and provides memory usage details: "Sketch uses 2986 bytes (9%) of program storage space. Maximum is 32256 bytes. Global variables use 279 bytes (13%) of dynamic memory, leaving 1769 bytes for local variables". At the bottom right, it says "Arduino/Genuino Uno on COM3".

Now you can see on the LCD1602 display, "Hello Geeks!" is scrolling first, and then "Adept www.adeept.com" appears and stays unchanged.

If characters are not displayed on the LCD, try to turn the potentiometer (a blue knob) for contrast adjustment on the I2C Interface Module. Or, you can toggle the switch on the module to see whether the backlight is on.





# Adeept

# Lesson 15 Application of Touch Button

## Introduction

The Touch Button Module is a touch switch module developed based on the principle of capacitive sensing. Touch of human or metal onto the gilded touch surface can be sensed. Besides, it can also detect other such touch with certain materials like plastic and glass between. The sensitivity in these cases depends on the touched area and thickness of the material between the touch pad and human or metal. The module can be conveniently used to replace physical buttons.

**Note:** During the application, remember to leave some space in height between the module and fixed surface in case of signal errors.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Touch Button Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

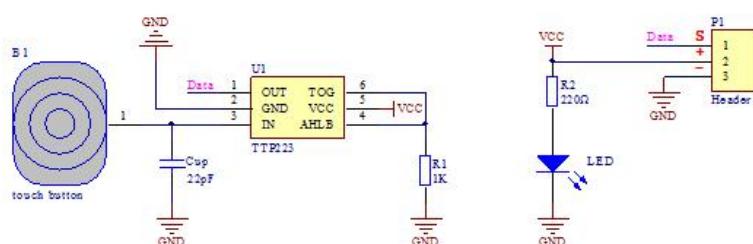
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

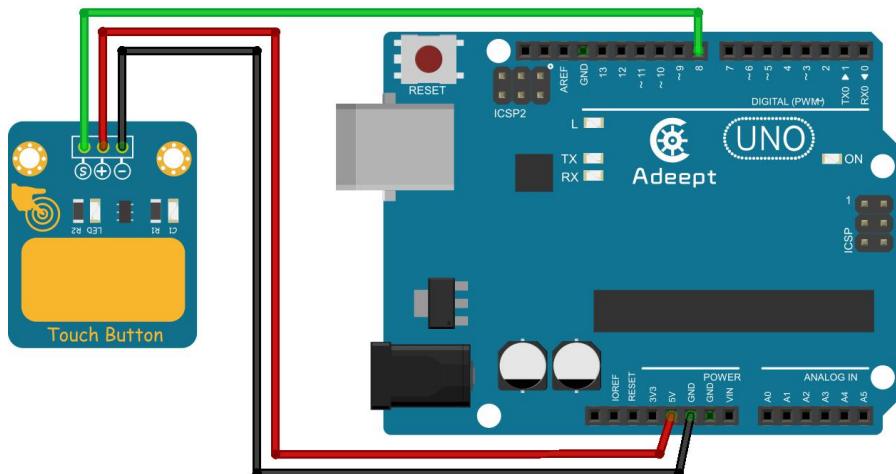
The schematic diagram:



In this experiment, by programming the Arduino, we detect the High or Low of the output terminal of the Touch Button Module by pin D8 of the Arduino board, so as to tell whether fingers touched the touch button or not.

## Experimental Procedures

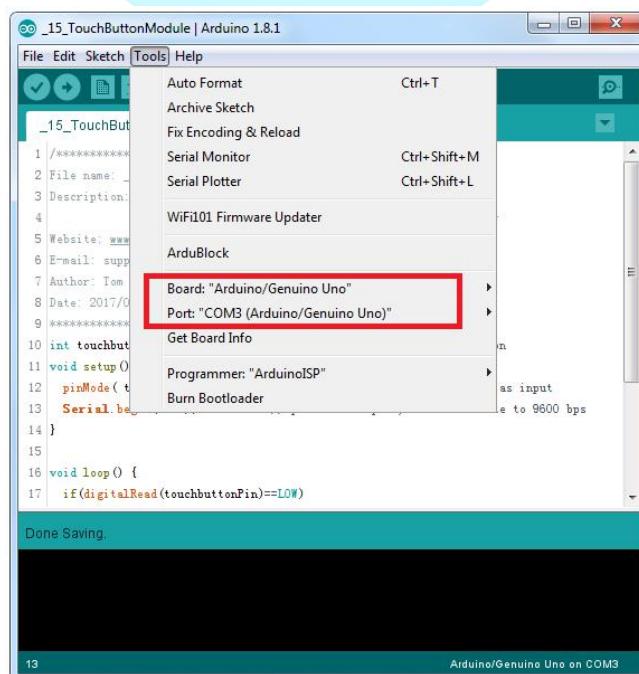
### Step 1: Build the circuit

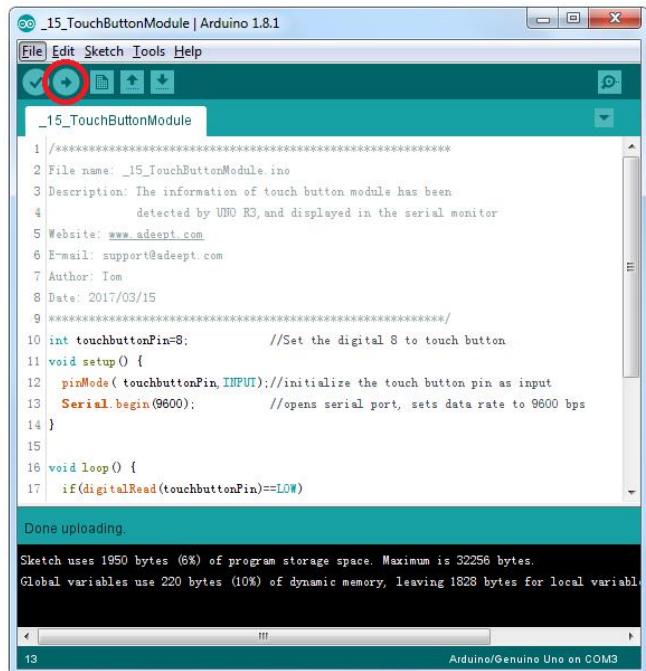


Adeept UNO R3 Board	Touch Button Module
D8	S
5V	+
GND	-

**Step 2:** Program `_15_TouchButtonModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



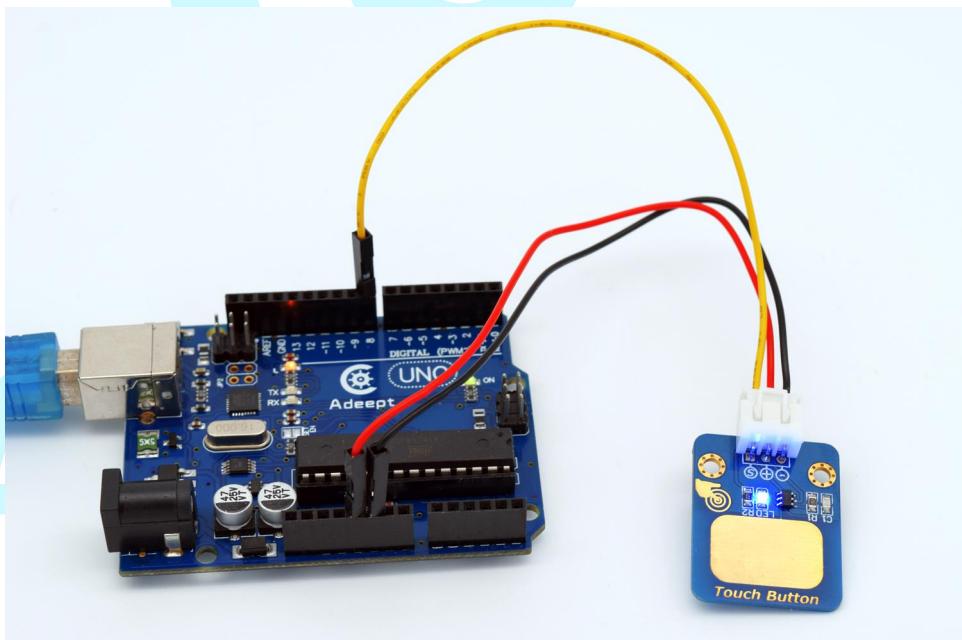


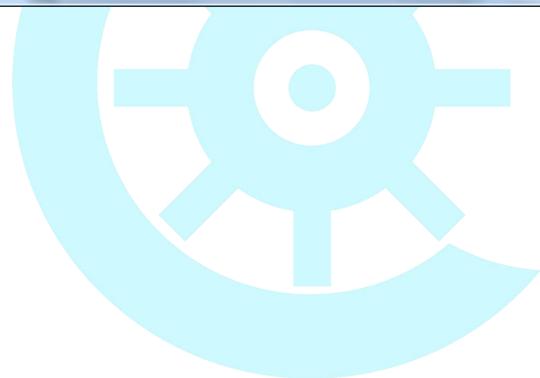
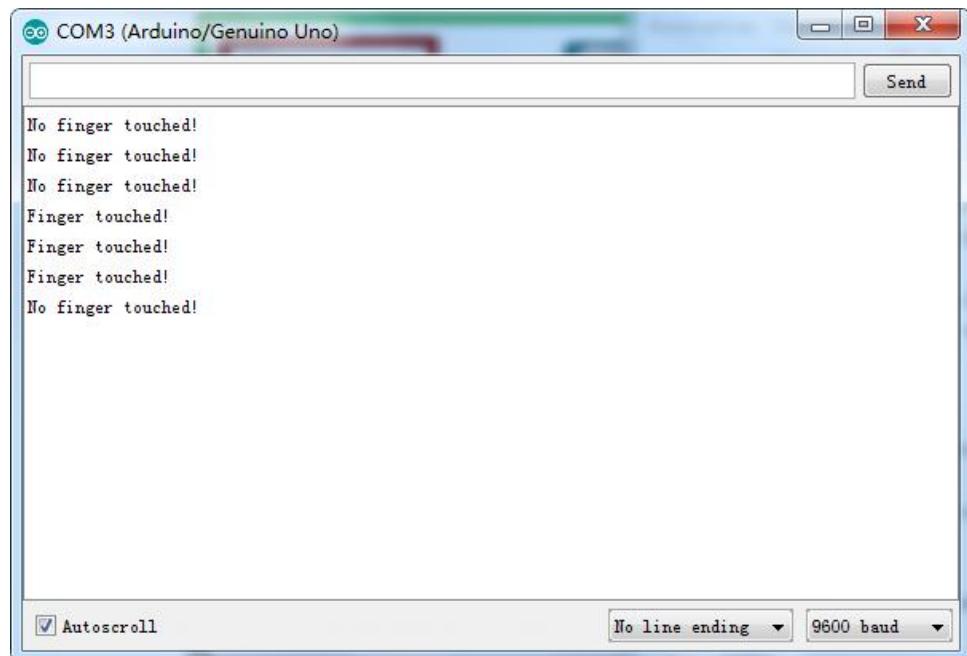
The screenshot shows the Arduino IDE interface with the title bar '\_15\_TouchButtonModule | Arduino 1.8.1'. The menu bar includes File, Edit, Sketch, Tools, Help. A red circle highlights the 'Upload' button (a blue arrow pointing right) in the toolbar. The code editor contains the '\_15\_TouchButtonModule' sketch. The serial monitor at the bottom shows the message 'Done uploading.' followed by the program statistics: 'Sketch uses 1950 bytes (6%) of program storage space. Maximum is 32256 bytes. Global variables use 220 bytes (10%) of dynamic memory, leaving 1828 bytes for local variables.'

```
1 //*****
2 File name: _15_TouchButtonModule.ino
3 Description: The information of touch button module has been
4 detected by UNO R3, and displayed in the serial monitor
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/15
9 *****/
10 int touchbuttonPin=8;           //Set the digital 8 to touch button
11 void setup() {
12   pinMode( touchbuttonPin, INPUT); //initialize the touch button pin as input
13   Serial.begin(9600);           //opens serial port, sets data rate to 9600 bps
14 }
15
16 void loop() {
17   if(digitalRead(touchbuttonPin)==LOW)

```

Now open the Serial Monitor in Arduino IDE. Touch the Touch Button Module and you'll see "Finger touched!" on the window; move away and "No finger touched!" will appear.





# Adeept

# Lesson 16 Pulse Count

## Introduction

Rotary encoder switch, or small rotary encoder, is a switch electronic component that has a set of regular and strictly-sequenced pulses. The module supports functions such as increase, decrease, turn pages, etc., by collaboration with a microcontroller. For example, in daily life you can see page turning of the mouse, menu selection, volume adjustment of speakers, temperature adjustment of toaster, frequency adjustment of medical equipment, etc.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Rotary Encoder Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

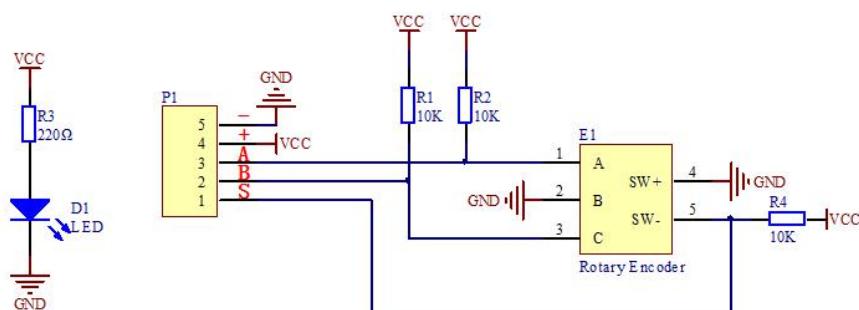
The Fritzing image:



Pin definition:

S	Digital output
B	Digital output
A	Digital output
+	VCC
-	GND

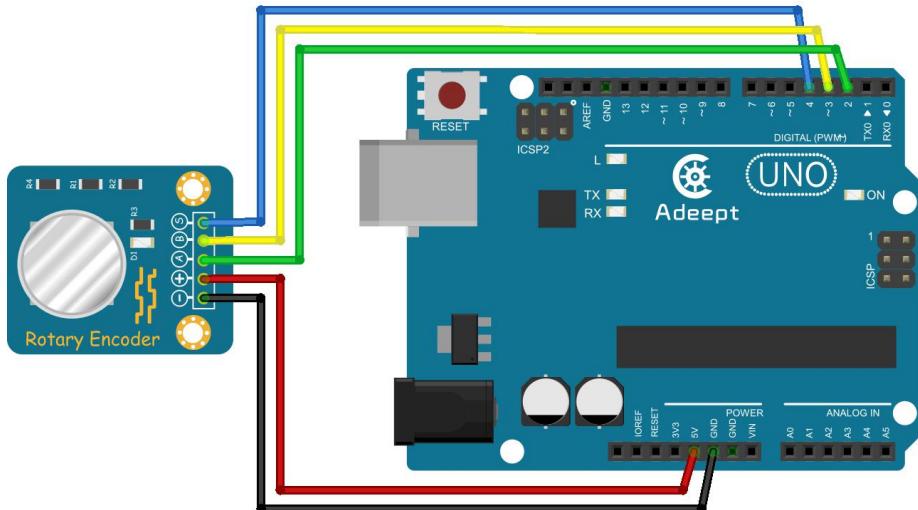
The schematic diagram:



In this experiment, by programming the Arduino, we change a value by reading the status of the Rotary Encoder. When we turn the knob of the Rotary Encoder clockwise, the value on the window will increase; when we turn the knob counterclockwise, the value will decrease. When we press down the switch, the value will be zeroed out.

## Experimental Procedures

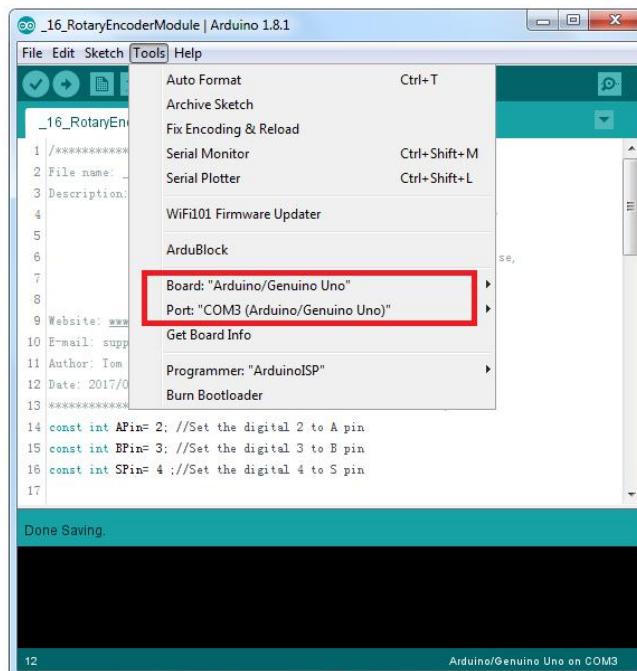
## Step 1: Build the circuit

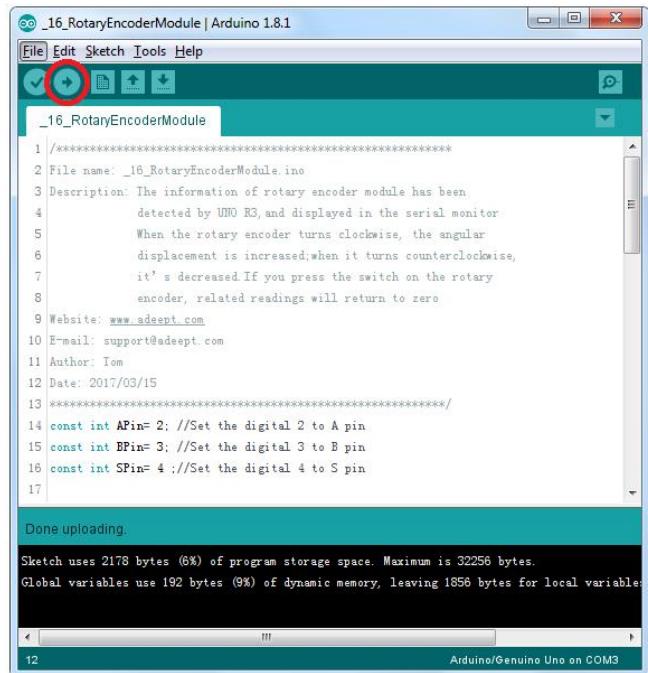


Adept UNO R3 Board	Rotary Encoder Module
D4	S
D3	B
D2	A
5V	+
GND	-

**Step 2:** Program \_16\_RotaryEncoderModule.ino

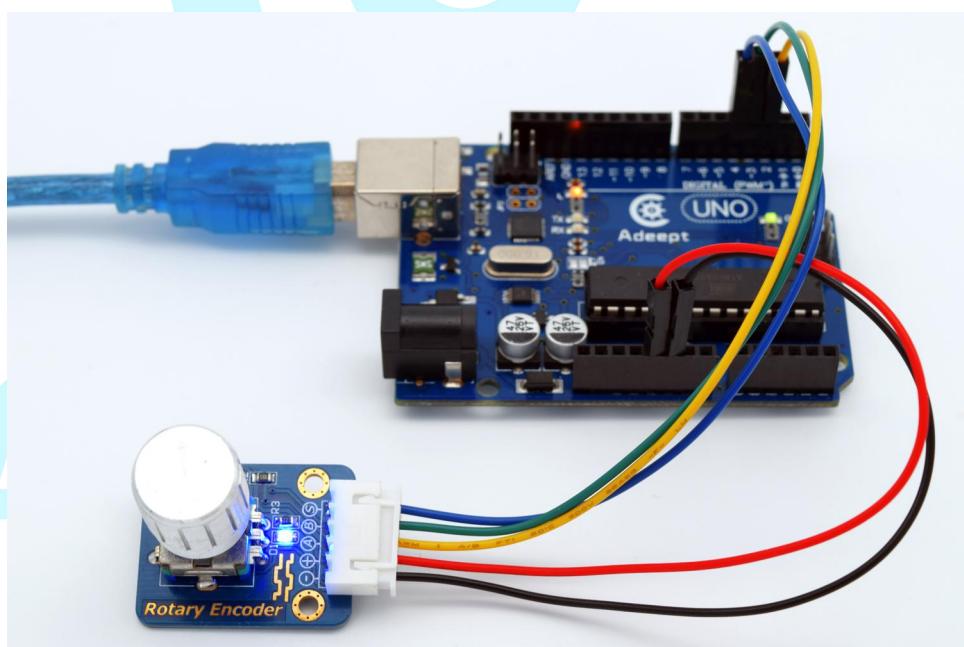
**Step 3:** Compile and download the sketch to the UNO R3 board.

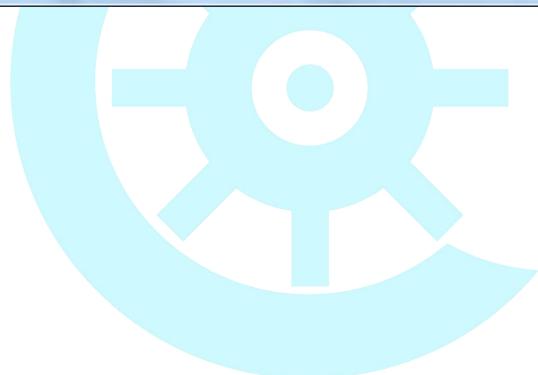
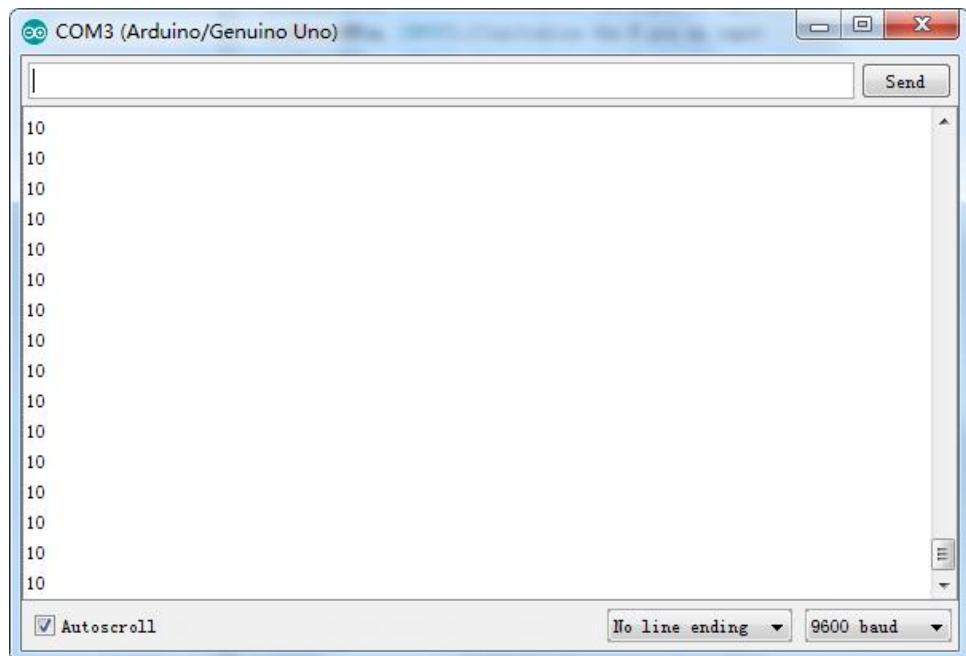




The screenshot shows the Arduino IDE interface with the title bar '16\_RotaryEncoderModule | Arduino 1.8.1'. The menu bar includes File, Edit, Sketch, Tools, and Help. A circular progress bar at the top left indicates the upload status. The code editor contains the '\_16\_RotaryEncoderModule.ino' sketch. The code is a simple program for reading values from a rotary encoder connected to pins A, B, and S. The serial monitor window at the bottom shows the message 'Done uploading.' and provides memory usage details: 'Sketch uses 2178 bytes (6%) of program storage space. Maximum is 32256 bytes. Global variables use 192 bytes (9%) of dynamic memory, leaving 1856 bytes for local variables.'

Open the Serial Monitor in Arduino IDE. Turn or press down the knob of the Rotary Encoder, the value on the window will increase, decrease, or be cleared.





# Adeept

# Lesson 17 Impact checking

## Introduction

Limit Switch, or travel switch, can be installed on relatively stationary objects such as mounting bracket and door frame, or moving objects like car and door. When the moving object approaches the stationary one, the switch is closed; when the moving one moves away from the static one, the switch is open.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Limit Switch Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

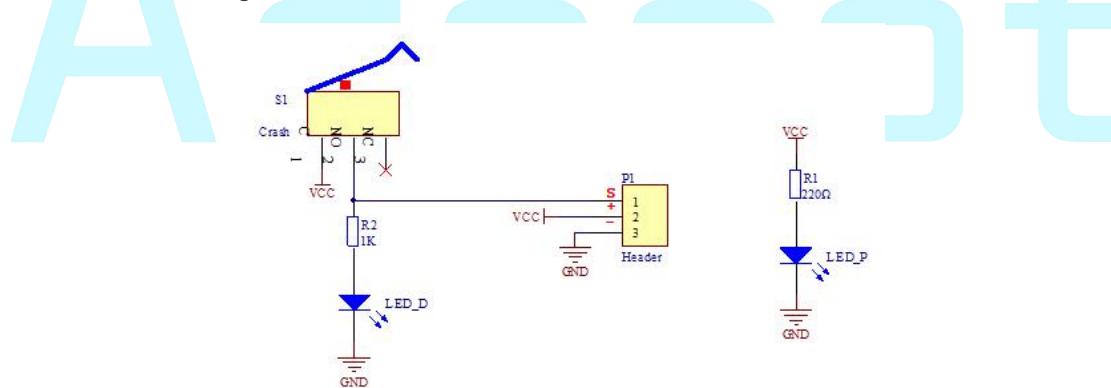
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

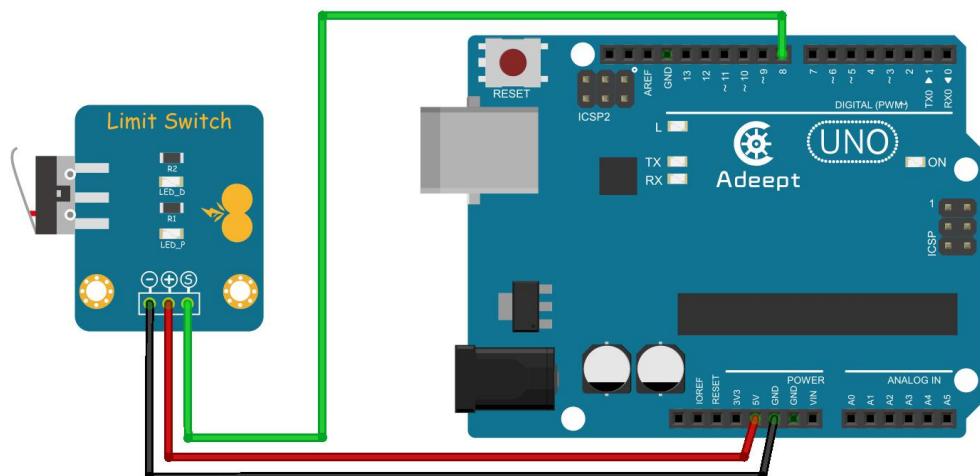
The schematic diagram:



In this experiment, by programming the Arduino, we detect the status of the Limit Switch module through pin D8 of the Arduino board and display it on Serial Monitor via the serial port.

## Experimental Procedures

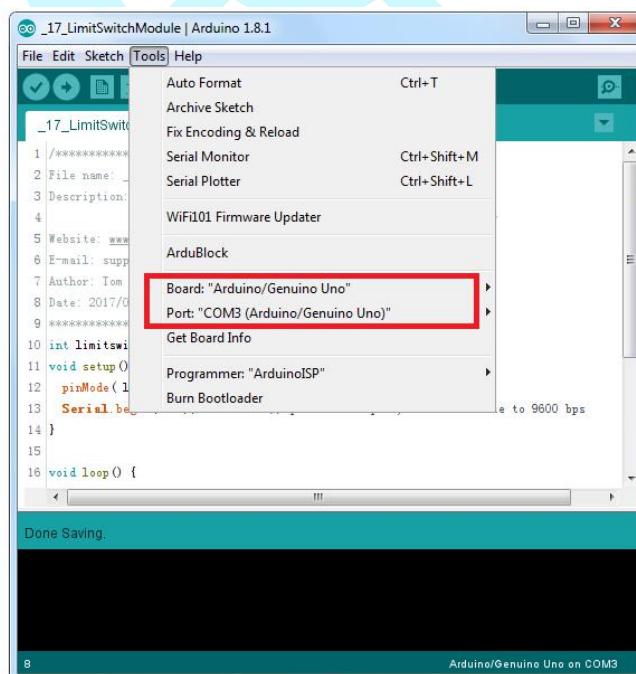
**Step 1: Build the circuit**

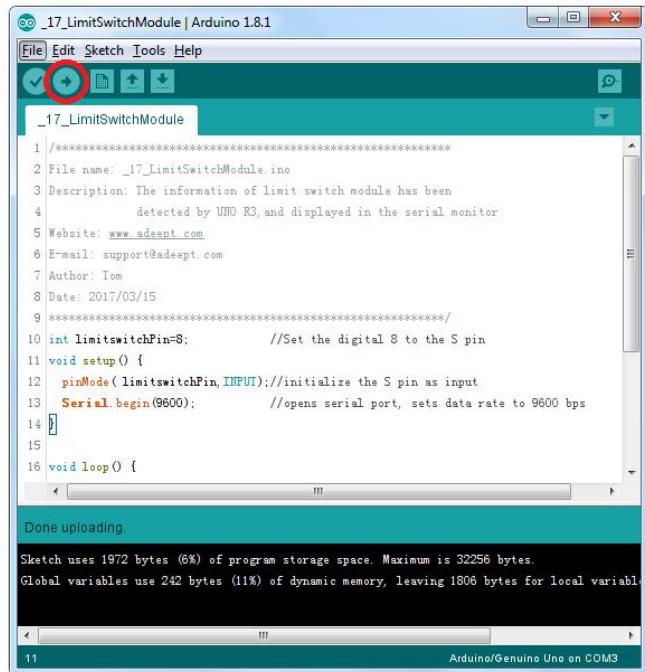


Adeept UNO R3 Board	Limit Switch Module
D8	S
5V	+
GND	-

**Step 2:** Program `_17_LimitSwitchModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



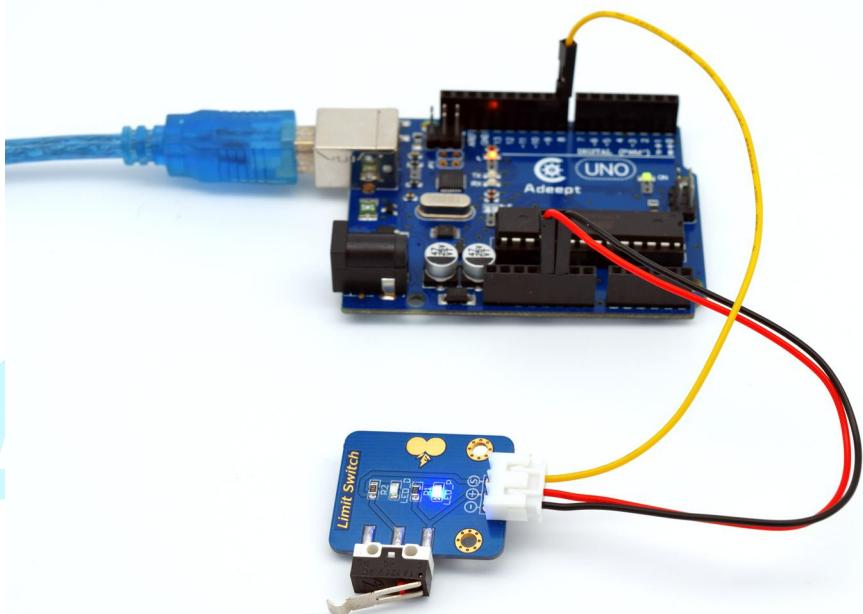


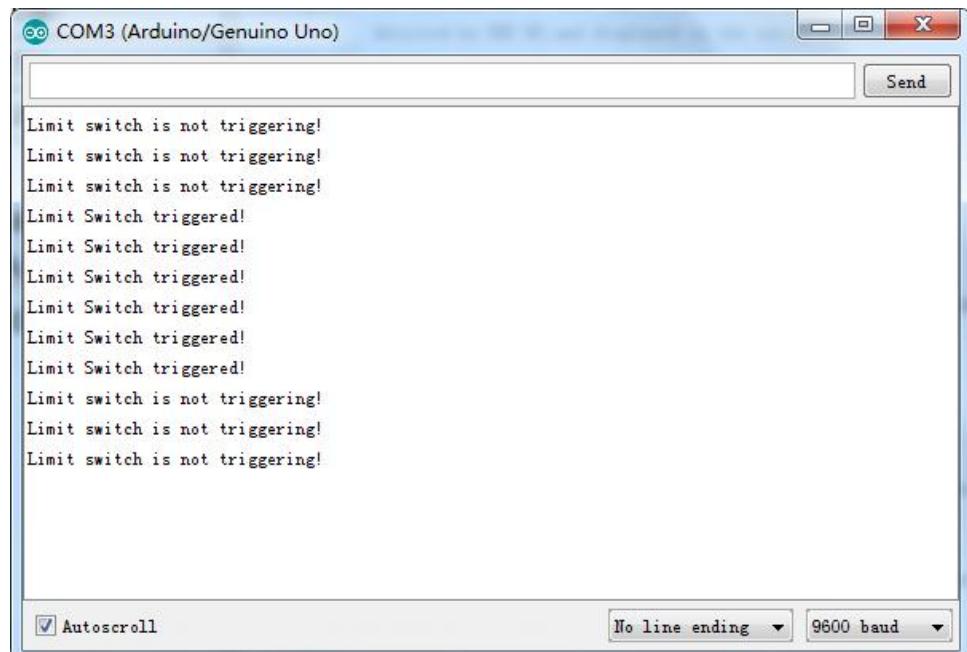
```
_17_LimitSwitchModule | Arduino 1.8.1
File Edit Sketch Tools Help
1 //*****
2 File name: _17_LimitswitchModule.ino
3 Description: The information of limit switch module has been
4 detected by UNO R3, and displayed in the serial monitor
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/15
9 ****
10 int limitswitchPin=8;           //Set the digital 8 to the S pin
11 void setup() {
12   pinMode( limitswitchPin,INPUT); //initialize the S pin as input
13   Serial.begin(9600);          //opens serial port, sets data rate to 9600 bps
14 }
15
16 void loop() {
}
Done uploading.

Sketch uses 1972 bytes (6%) of program storage space. Maximum is 32256 bytes.
Global variables use 242 bytes (1%) of dynamic memory, leaving 1806 bytes for local variables.

11 Arduino/Genuino Uno on COM3
```

Open the Serial Monitor in Arduino IDE. Press the Limit Switch and you can see "Limit Switch triggered!" on the window.





# Adeept

# Lesson 18 Simple Laser Cannon

## Introduction

Semiconductor laser modules are widely used in laser communication, ranging, ladar, ignition and blasting, and testing instruments.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Laser Transmitter Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

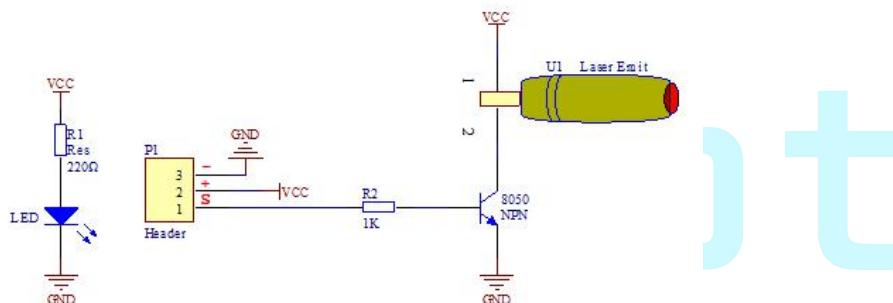
The Fritzing image:



Pin definition:

S	Digital Input
+	VCC
-	GND

The schematic diagram:

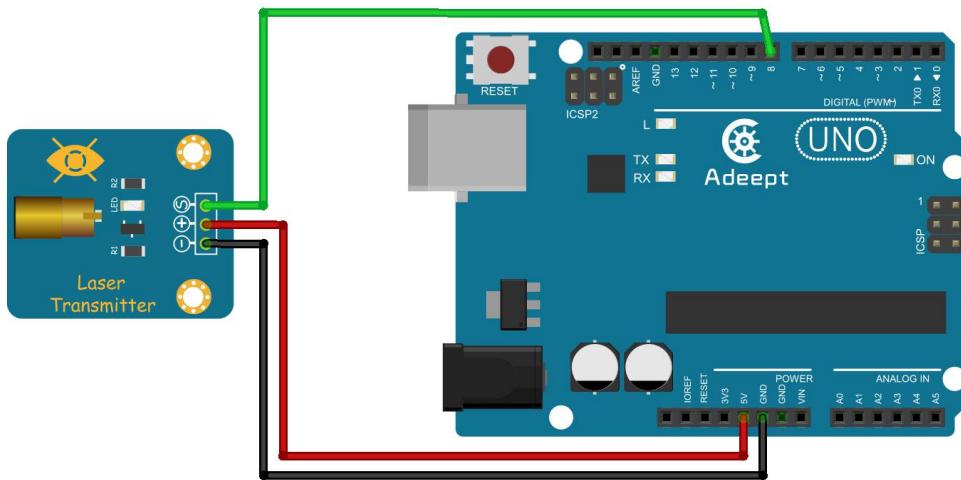


In this experiment, by programming the Arduino, we control the Laser Transmitter Module to emit laser by pin A0 of the Arduino board.

**Note: DO NOT look directly into the laser!**

## Experimental Procedures

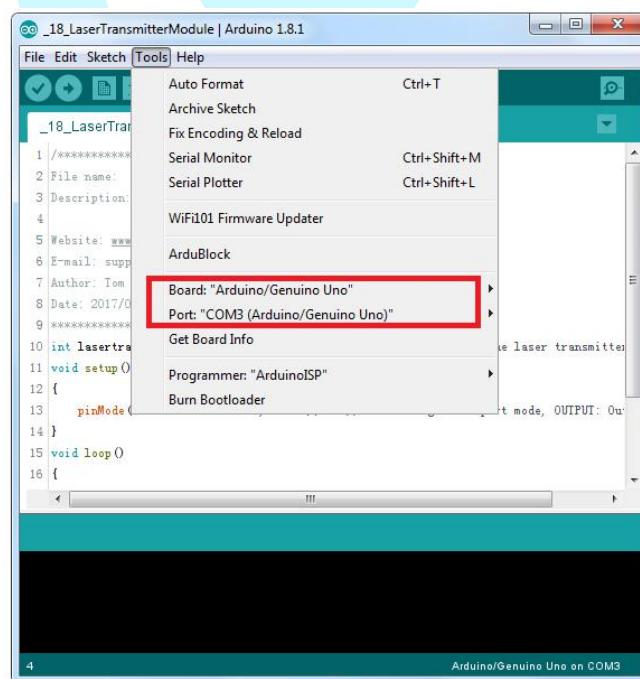
**Step 1:** Build the circuit

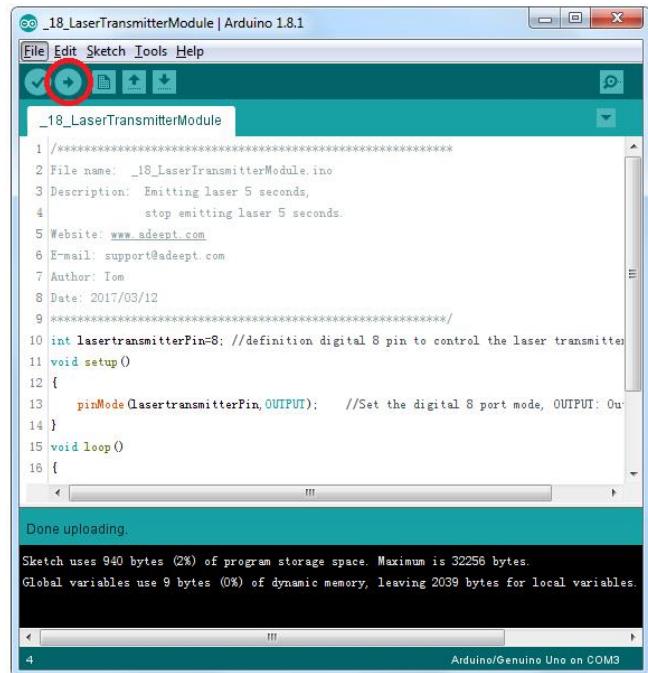


Adeept Uno R3 Board	Laser Transmitter Module
D8	S
5V	+
GND	-

**Step 2:** Program `_18_LaserTransmitterModule.ino`

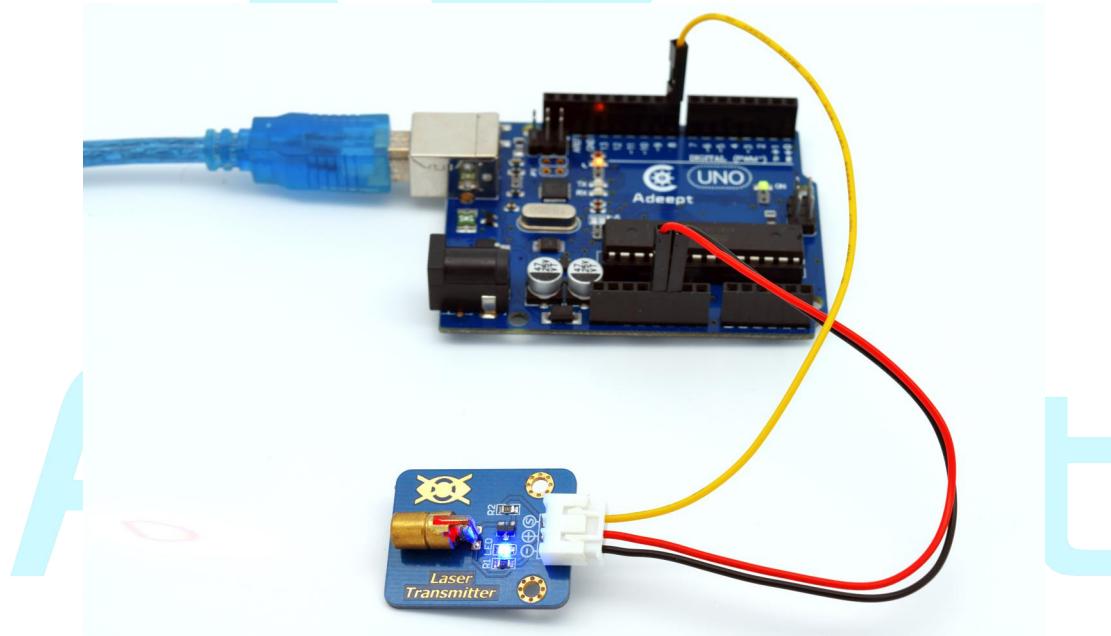
**Step 3:** Compile and download the sketch to the Uno R3 board.





The screenshot shows the Arduino IDE interface with the title bar '00 \_18\_LaserTransmitterModule | Arduino 1.8.1'. The menu bar includes File, Edit, Sketch, Tools, and Help. A red circle highlights the 'Upload' button (a blue arrow pointing right) in the toolbar. The code editor displays the '\_18\_LaserTransmitterModule' sketch, which includes comments about emitting a laser for 5 seconds and stopping for 5 seconds, along with setup and loop functions. Below the code, a message says 'Done uploading.' and provides memory usage details: 'Sketch uses 940 bytes (2%) of program storage space. Maximum is 32256 bytes. Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.' The status bar at the bottom right indicates 'Arduino/Genuino Uno on COM3'.

Now you can see the Laser Transmitter Module emit laser and the emission lasts for 5 seconds, and then it stops. After 5s, the cycle repeats.



# Lesson 19 Simple Laser Targeting

## Introduction

The principle for many laser receiving devices is the same. The laser ray goes through the optical lens and then is received by the photosensitive device, i.e. the photodiode. After receiving the rays, the photodiode will generate currents accordingly (based on the light intensity) which then output electrical signal after running through the amplifier. Then use an I/O port of the Arduino Uno board to detect the output terminal of the Laser Receiver module, and thus we can tell whether there are rays shone on the module.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Laser Receiver Module
- 1 \* Laser Transmitter Module
- 1 \* USB Cable
- 2 \* 3-Pin Wires
- 2 \* Hookup Wire Set
- 1 \* Breadboard

## Experimental Principle

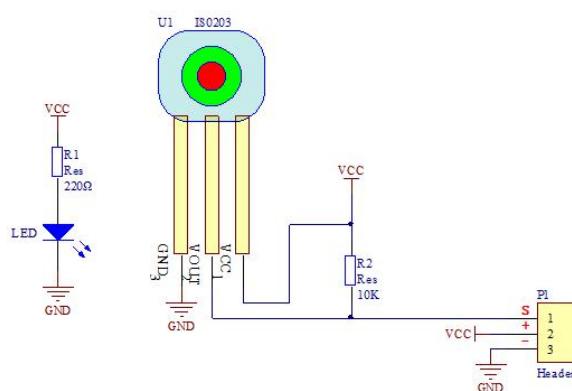
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

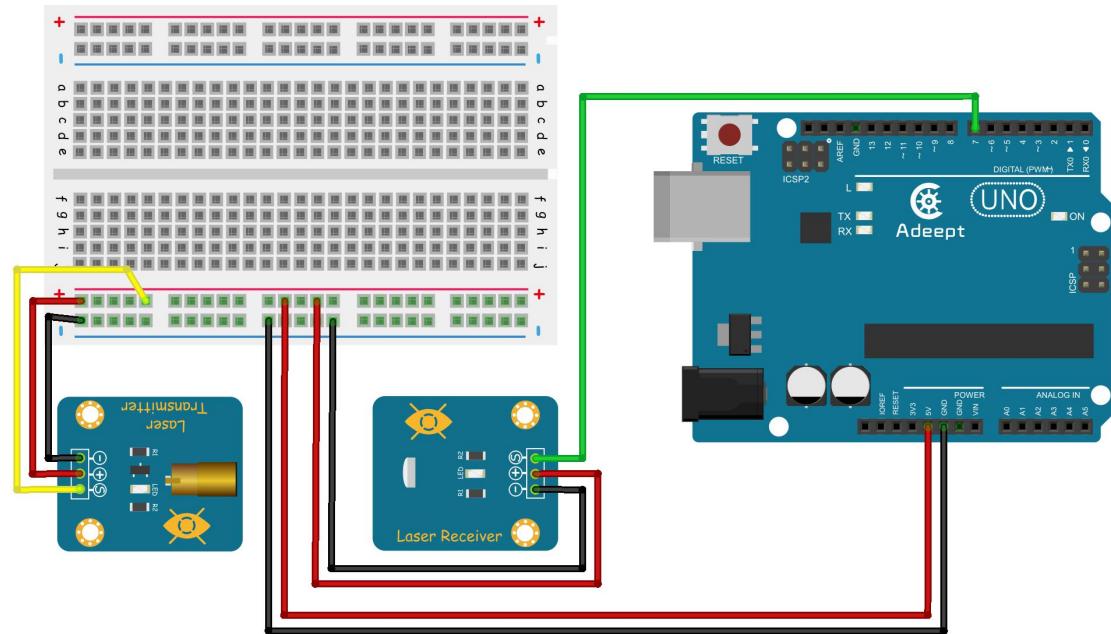
The schematic diagram:



In this experiment, we use the Laser Receiver module to detect whether there is laser ray shining on the module. If yes, the output pin (S) of the module will output Low.

## Experimental Procedures

### Step 1: Build the circuit



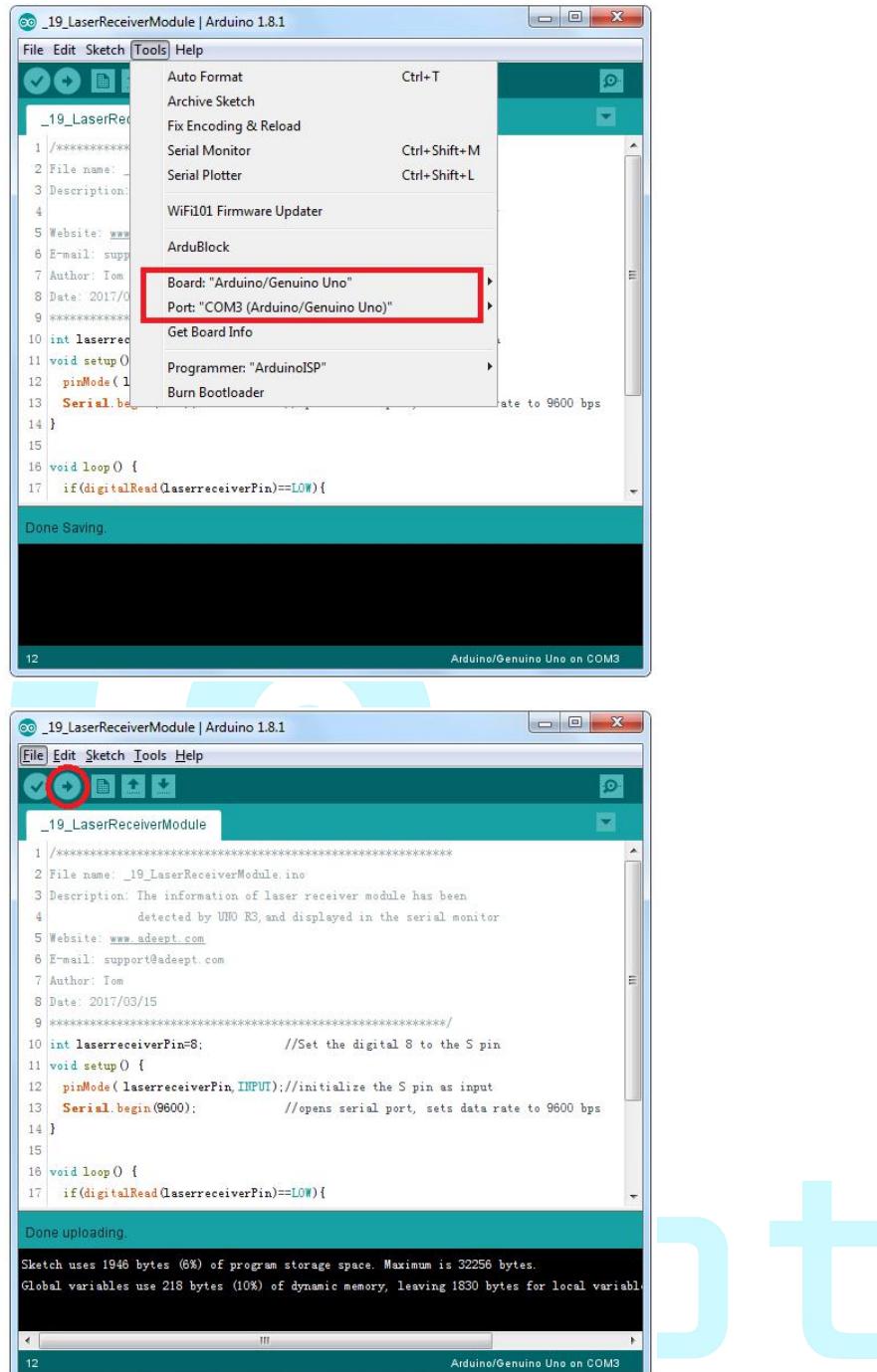
Adeept UNO R3 Board	Laser receiver Module
D7	S
5V	+
GND	-

Note: DO NOT look directly into the laser!

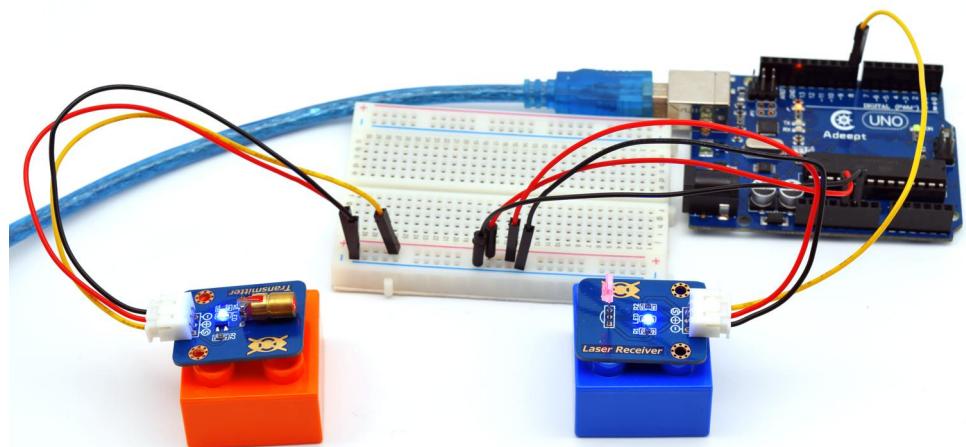
Step 2: Program \_19\_LaserReceiverModule.ino

Step 3: Compile and download the sketch to the UNO R3 board.

Adeept



Open the Serial Monitor in Arduino IDE. Make the Laser Transmitter module to shoot laser ray onto the Laser Receiver module. Then "Received laser" will be displayed on the window. Remove the transmitter module and "No laser received" will be shown.



```
COM3 (Arduino/Genuino Uno)
Send
No laser received
Received laser
Autoscroll
No line ending 9600 baud
```

A screenshot of the Arduino Serial Monitor window titled "COM3 (Arduino/Genuino Uno)". The window displays a series of text messages. It starts with six "No laser received" messages, followed by a single "Received laser" message, then a continuous loop of "Received laser" messages. At the bottom of the window, there are two status bars: "Autoscroll" and "No line ending" and "9600 baud".

# Lesson 20 Temperature And Humidity Detection

## Introduction

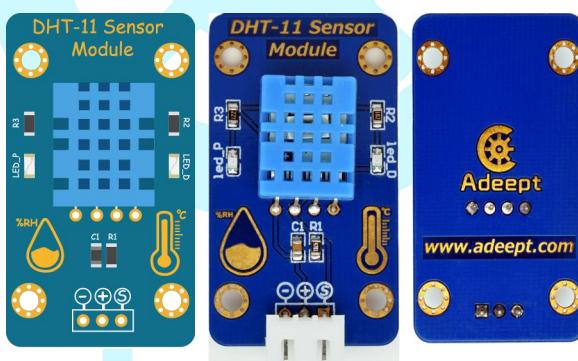
DHT11 is a composite digital thermal sensor that integrates temperature and humidity detection. It can convert the temperature and humidity analog values into digital values via corresponding sensitive components and built-in circuits, which can be directly read by computer or other data collecting devices.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* DHT-11 Sensor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

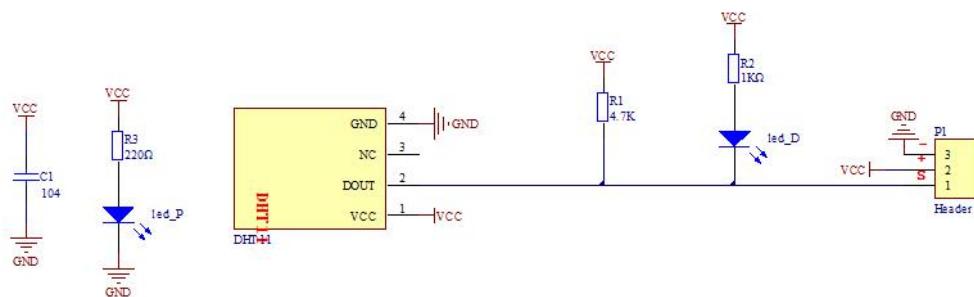
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

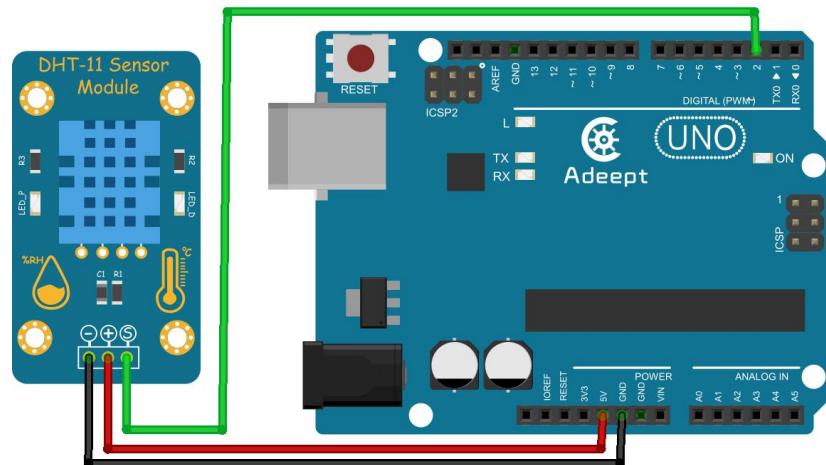
The schematic diagram:



In this experiment, by programming the Arduino, we read the temperature and humidity data collected by the DHT11 module by pin D2 of the Arduino board and display it on Serial Monitor via the serial port.

## Experimental Procedures

### Step 1: Build the circuit

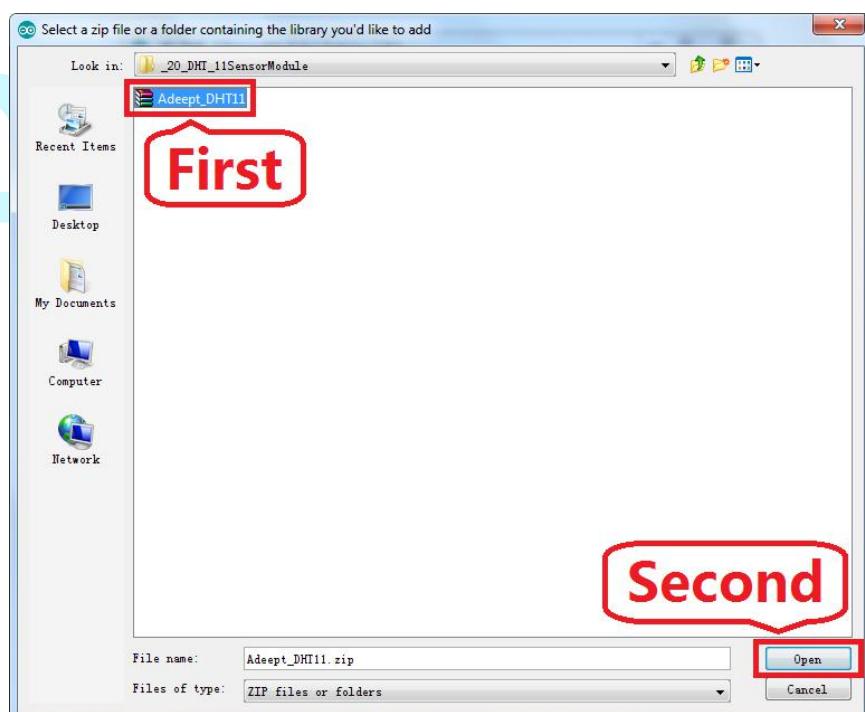
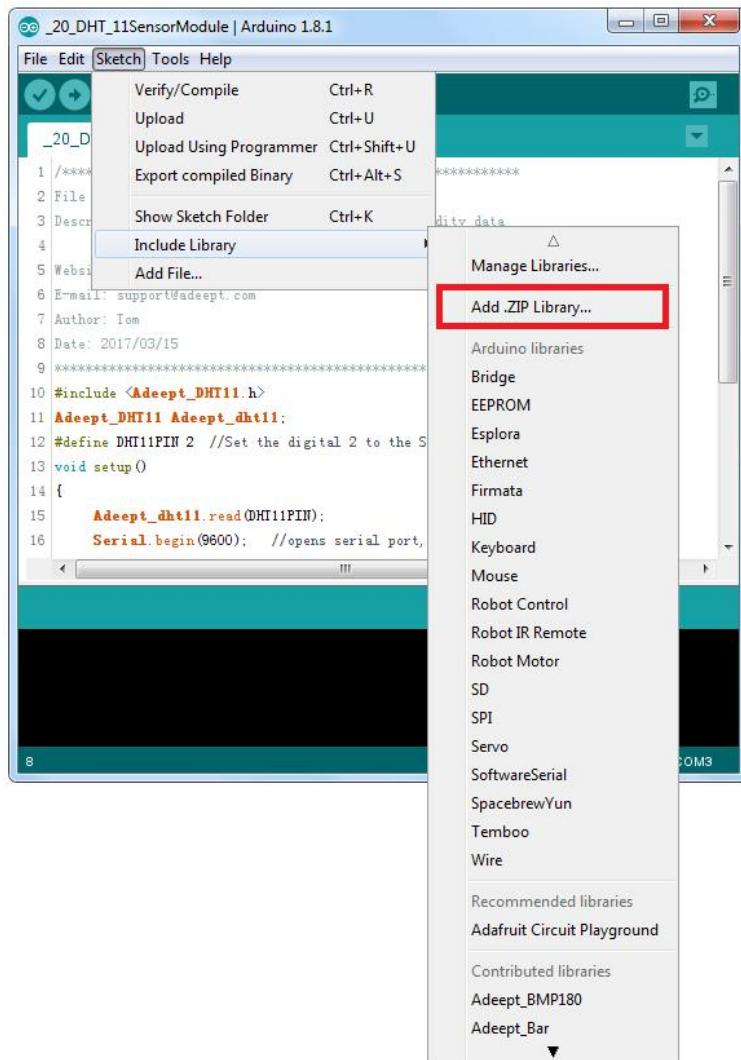


Adeept UNO R3 Board	DHT-11 Sensor Module
D2	S
5V	+
GND	-

Step 2: Install the function library (Adeept\_DHT11.zip).

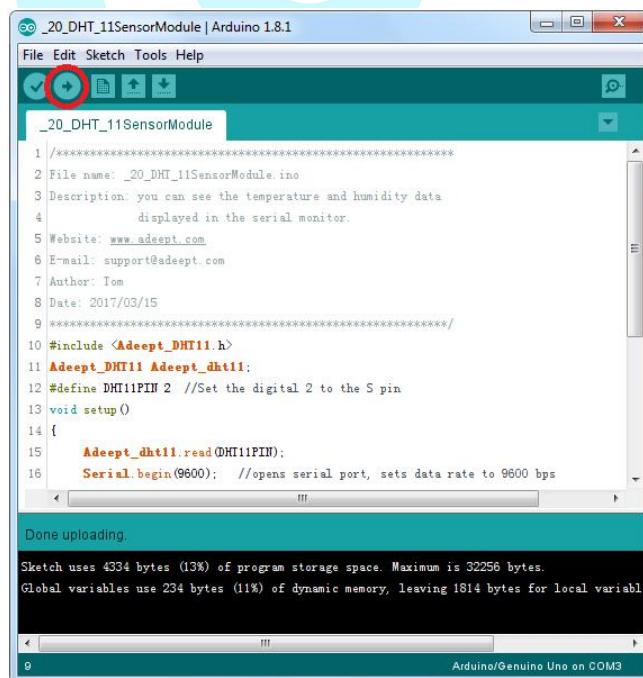
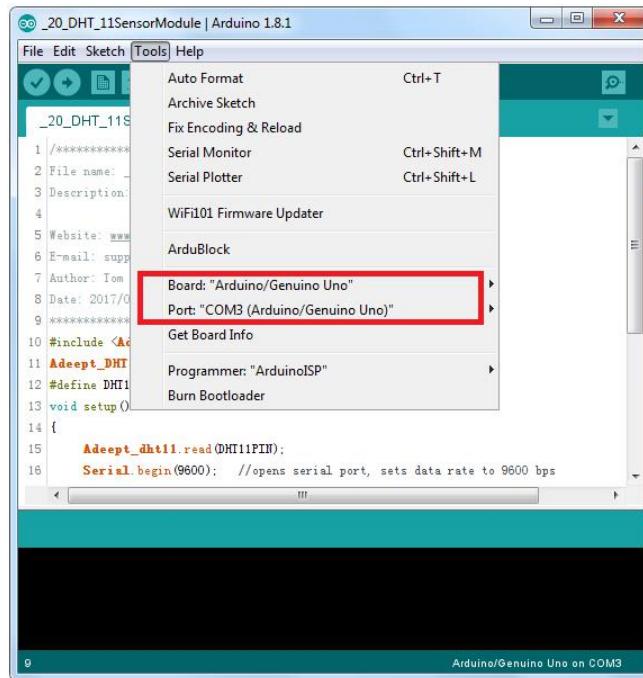


Adeept

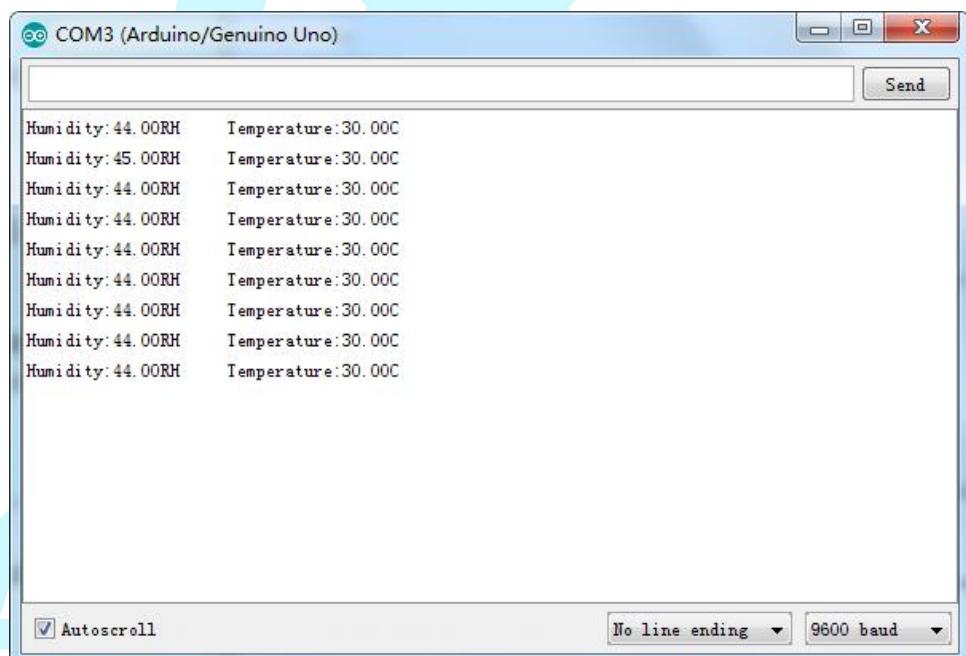
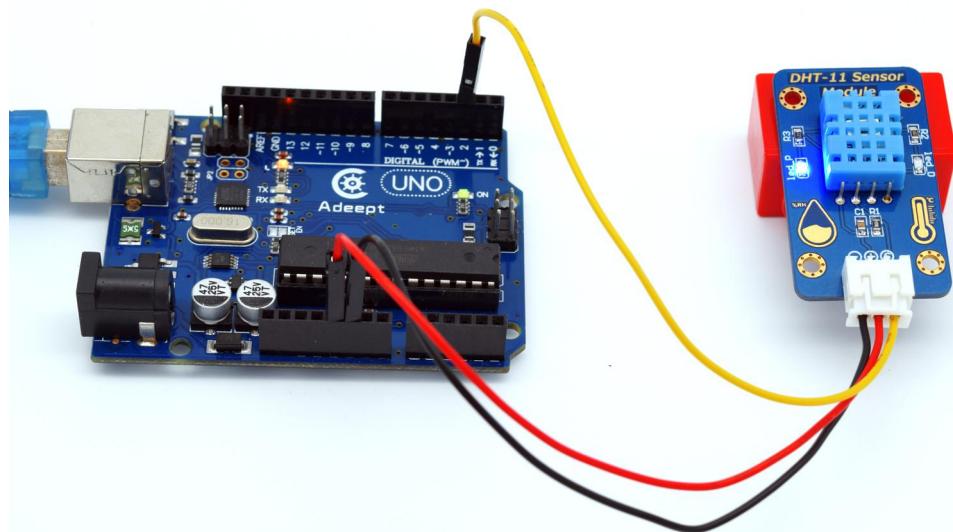


**Step 3:** Program \_20\_DHT\_11SensorModule.ino

**Step 4:** Compile and download the sketch to the UNO R3 board.



Open the Serial Monitor in Arduino IDE and you will see the data of current temperature and humidity displayed on the window.



# Lesson 21 How To Use The Reed

## Introduction

Reed switch is a special magnet-sensitive switch. Inside the glass tube, the reed sheets placed in parallel with a gap between compose the normally-open contact. When a magnet approaches the reed switch, or after the coil wrapped on the reed is electrified and a magnet field comes into existence thus magnetizing the reed, the contact of the reed will sense it and become the opposite pole. Due to the principle that different poles attract each other, when the magnetic force is larger than resistance of the reed, the open contacts (sheets) are closed; when the magnetic force decreases to a certain degree, the resistance takes charge again and the reed will back to the original state.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Reed Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

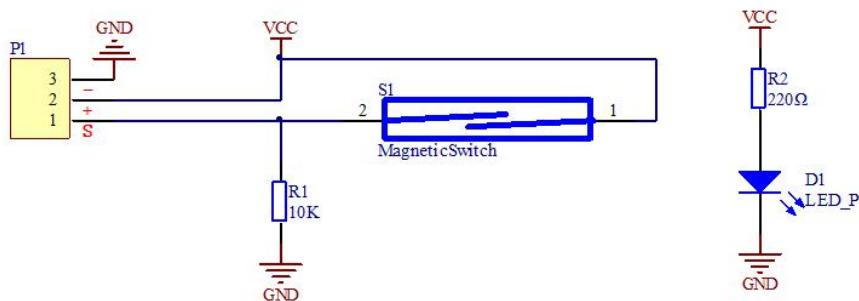
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

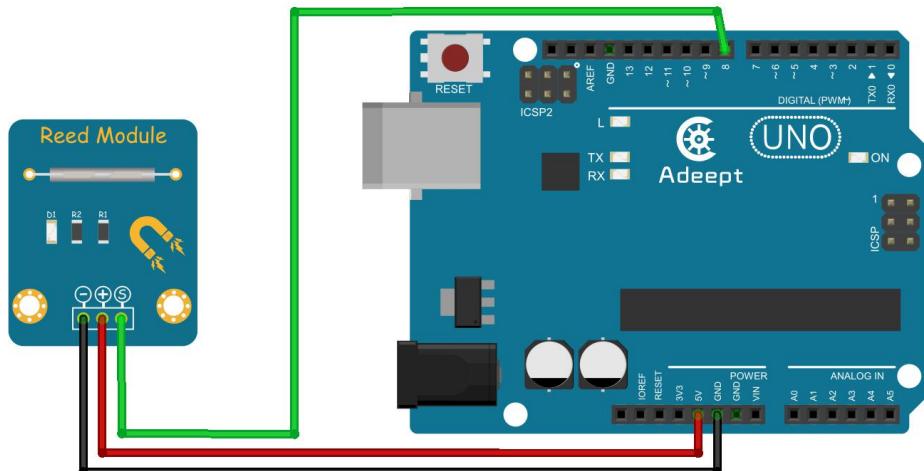
The schematic diagram:



In this experiment, by programming the Arduino, we detect pin D8 of the Arduino board and display the High or Low at D8 on the Serial Monitor.

## Experimental Procedures

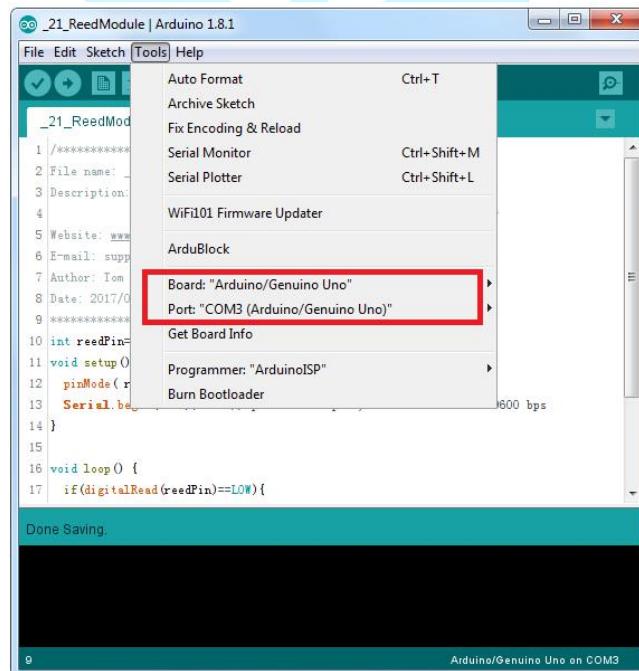
### Step 1: Build the circuit

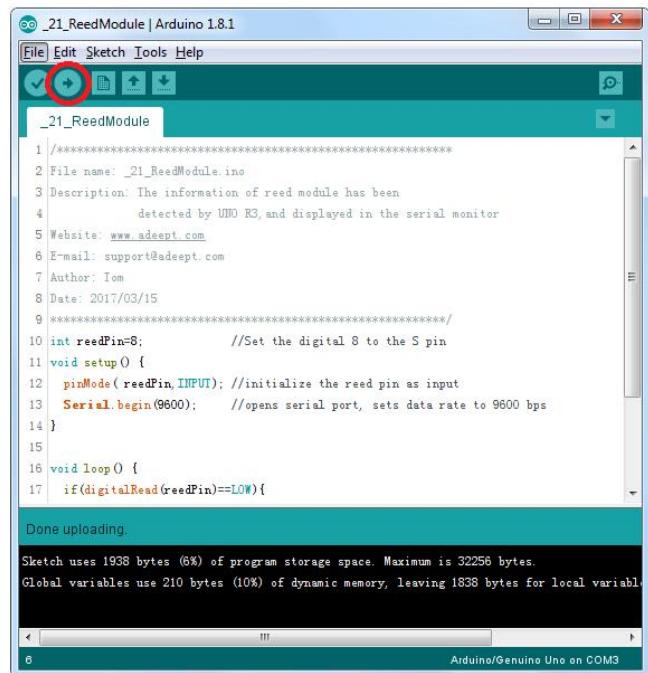


Adeept UNO R3 Board	Reed Module
D8	S
5V	+
GND	-

**Step 2:** Program `_21_ReedModule.ino`

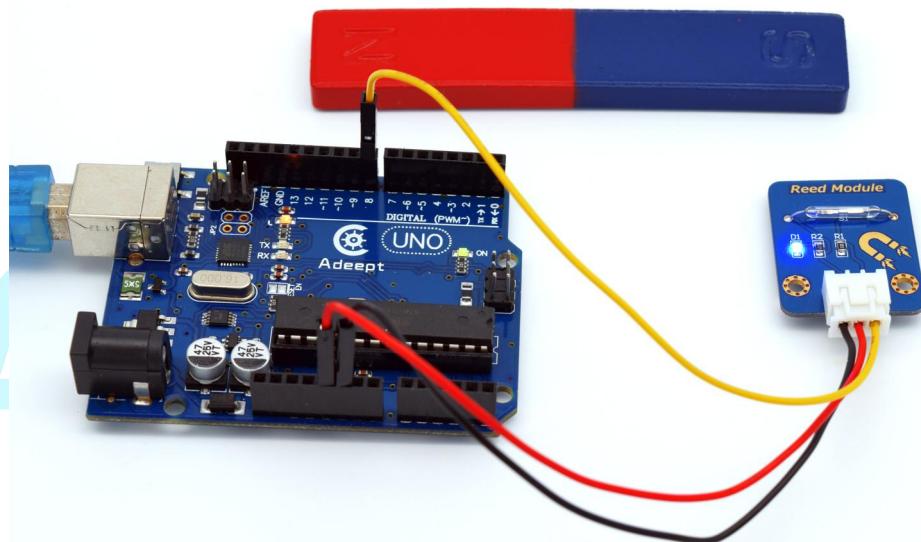
**Step 3:** Compile and download the sketch to the UNO R3 board.

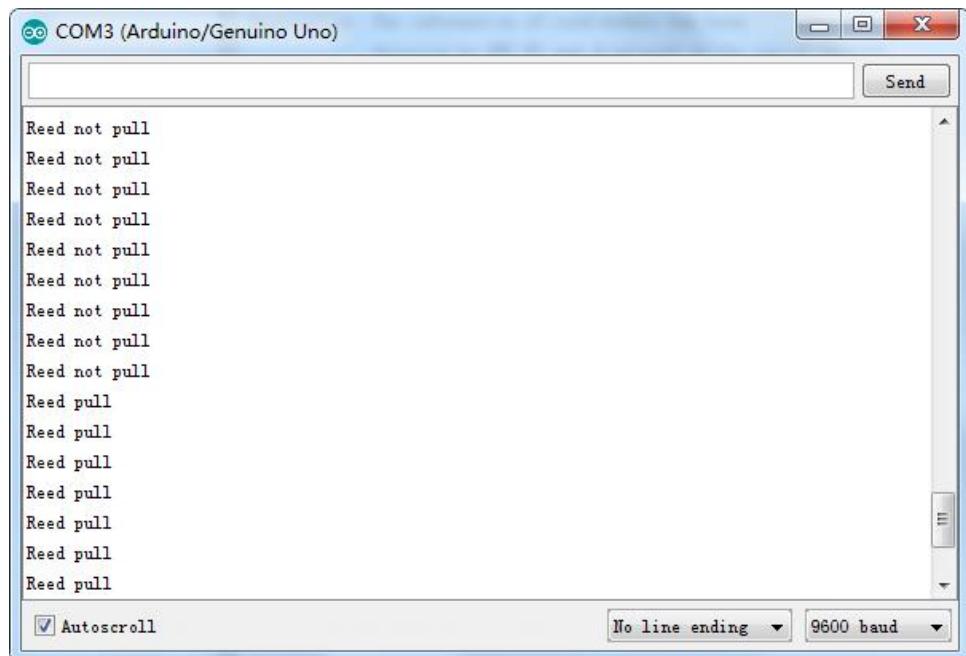




```
_21_ReedModule | Arduino 1.8.1
File Edit Sketch Tools Help
[Upload] [Save] [Upload] [Save] [Upload] [Save]
_21_ReedModule
1 //*****
2 File name: _21_ReedModule.ino
3 Description: The information of reed module has been
4 detected by UNO R3, and displayed in the serial monitor
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/15
9 ****
10 int reedPin=8;           //Set the digital 8 to the S pin
11 void setup() {
12   pinMode( reedPin,INPUT); //initialize the reed pin as input
13   Serial.begin(9600);    //opens serial port, sets data rate to 9600 bps
14 }
15
16 void loop() {
17   if(digitalRead(reedPin)==LOW){
18
Done uploading.
Sketch uses 1938 bytes (6%) of program storage space. Maximum is 32256 bytes.
Global variables use 210 bytes (10%) of dynamic memory, leaving 1838 bytes for local variables.
```

Open the Serial Monitor in Arduino IDE. Place the magnet near or away from the Reed Module and you will see the High ("Reed pull") and Low ("Reed not pull") change at D8 of the Arduino board on Serial Monitor.





# Lesson 22 Fire Detection

## Introduction

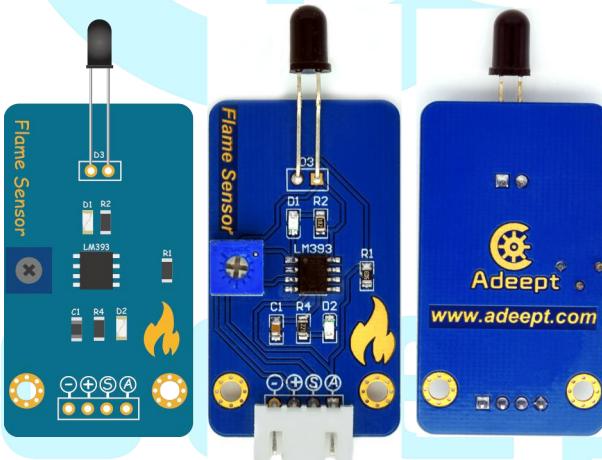
The Flame Sensor detects flames by the special infrared receiver to capture the infrared rays of a specific wavelength in the flames. It supports a detection angle of as high as 60 degrees and works within -25 - 85°C. When in use, you need to pay attention and do not place the probe of the sensor too close to the flames in case of damages. Besides, the sensor can be used to detect light intensity. It can detect a light source with a wavelength of 760 - 1100nm. Pin A outputs the analog data collected by the sensor. You can adjust the potentiometer on the module to set the alarm threshold of the sensor. So when the value reaches the threshold, pin S will switch between High and Low.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Flame Sensor Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

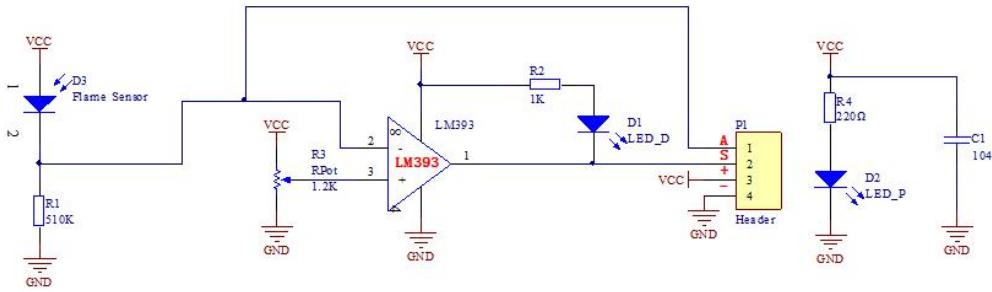
The Fritzing image:



Pin definition:

S	Digital output
A	Analog output
+	VCC
-	GND

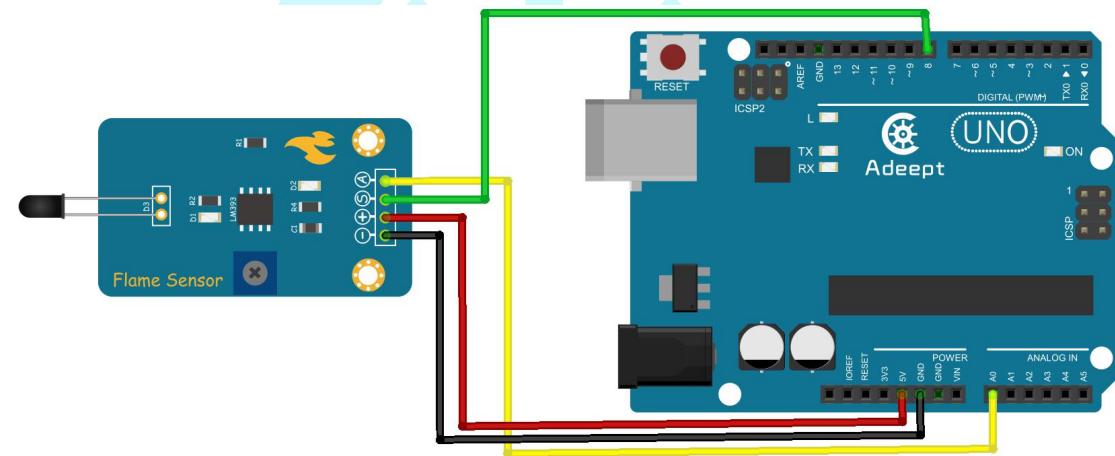
The schematic diagram:



In this experiment, by programming the Arduino, we collect the analog values and switch output from the Flame Sensor module through pin A0 and D8 of the Arduino board, and display them on the Serial Monitor via serial port.

## Experimental Procedures

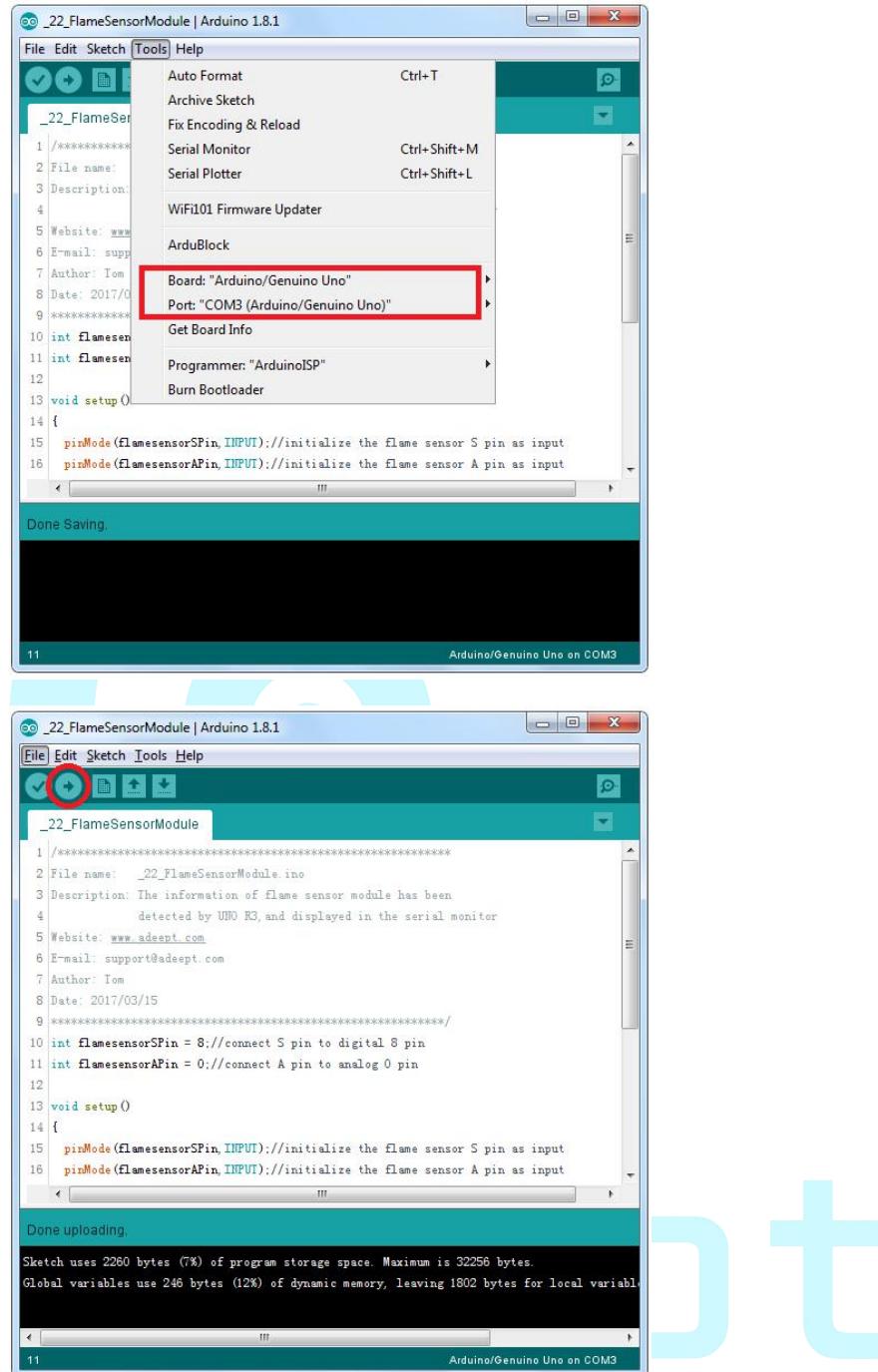
### Step 1: Build the circuit



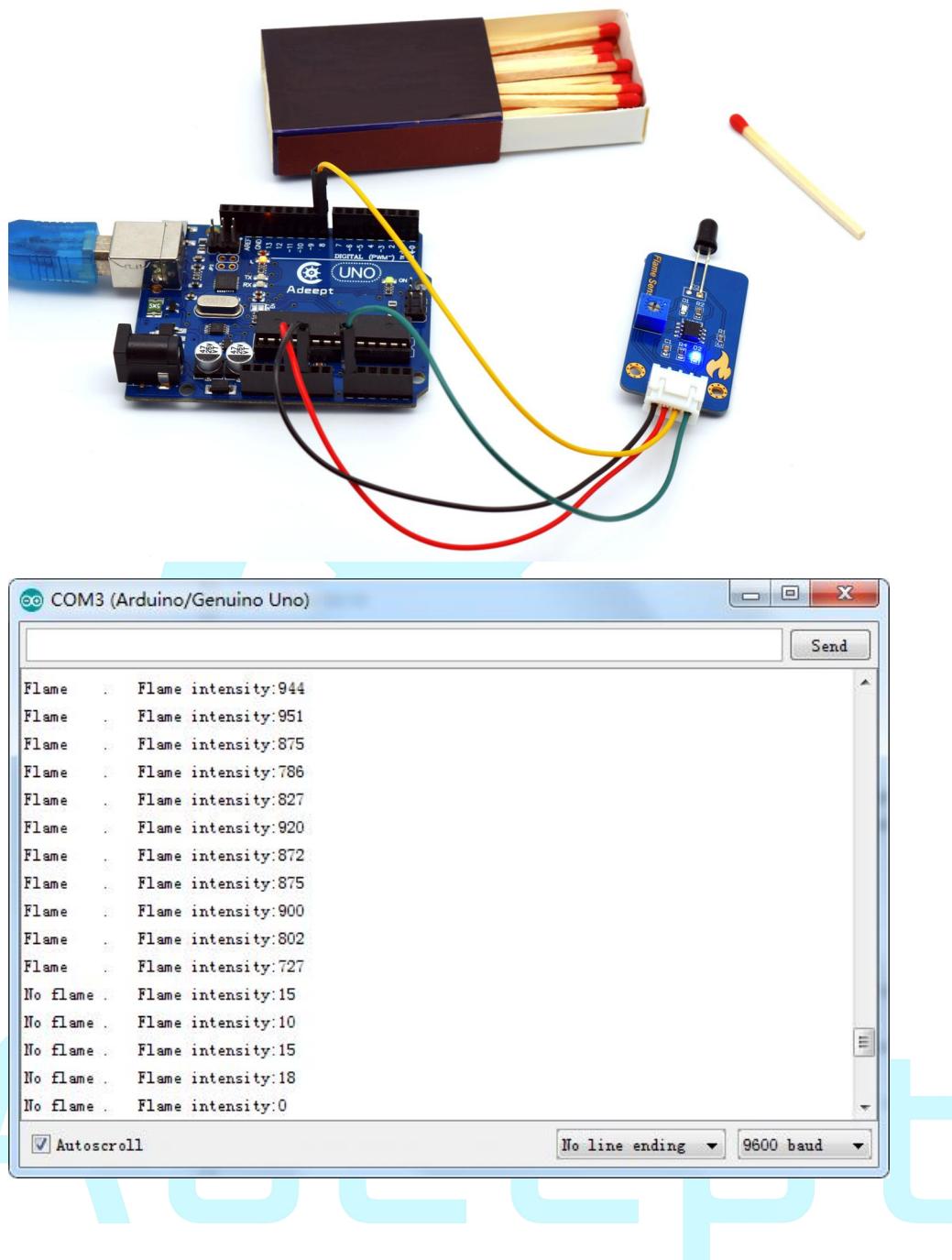
Adeebt UNO R3 Board	Flame Sensor Module
D8	S
A0	A
5V	+
GND	-

Step 2: Program `_22_FlameSensorModule.ino`

Step 3: Compile and download the sketch to the UNO R3 board.



Open the Serial Monitor in Arduino IDE. Then you'll see the data detected by the Flame Sensor module on the window. The data includes two parts: "Flame"/"No flame" and "Flame intensity".



# Lesson 23 Decetion of Flammable Gases

## Introduction

MQ-2 is a sensor that can detect flammable gases such as methane, hydrogen, and propane and so on. It adopts the low conductivity stannic oxide for the basic material. When there are flammable gases in the ambient environment, the conductivity of the sensor will increase as the gases become denser. This type can detect a wide range of gases thus making it a low cost multifunctional sensor. The sensor can be used for methane leak alarm and automatic smoke exhaust fan. With the features, it boasts a perfect sensor for indoor air regulation that meets the environmental standards.

### Notes:

1. This sensor is equipped with an adjustment potentiometer for the alarm threshold. Spin the knob of the pot clockwise, and the threshold will be increased; spin it counterclockwise, the threshold will be reduced.
2. The sensor may not output a steady and accurate data immediately; it needs to be preheated for about 1 minute to collect data steadily.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* MQ-2 Gas Sensor Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

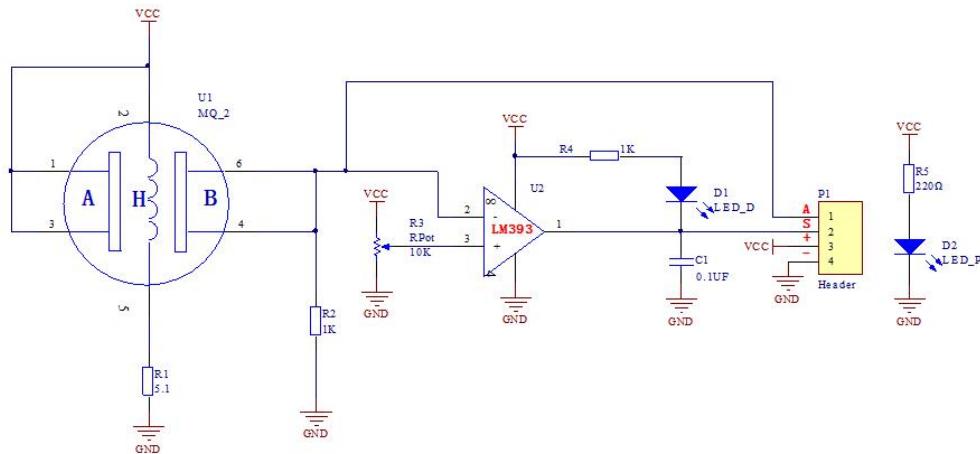
The Fritzing image:



Pin definition:

S	Digital output
A	Analog output
+	VCC
-	GND

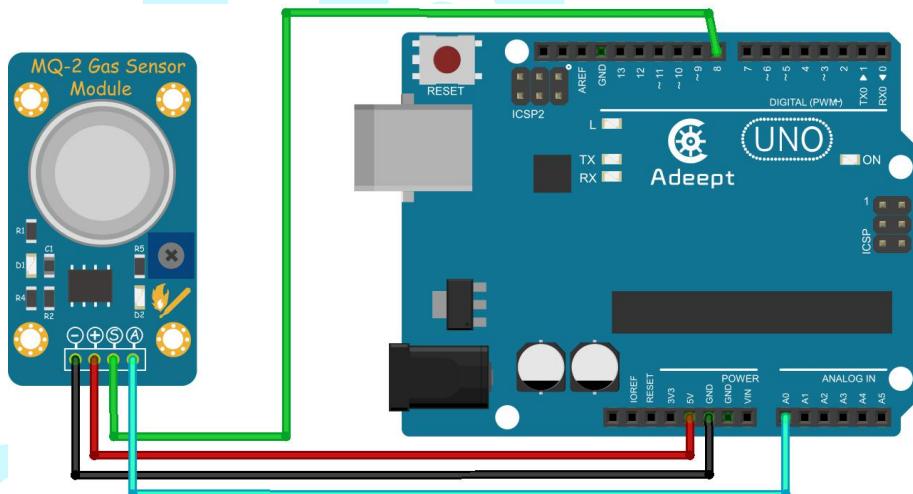
The schematic diagram:



In this experiment, by programming the Arduino, we read the analog and switch values collected by the MQ-2 Gas Sensor and display them on the Serial Monitor via serial port.

## Experimental Procedures

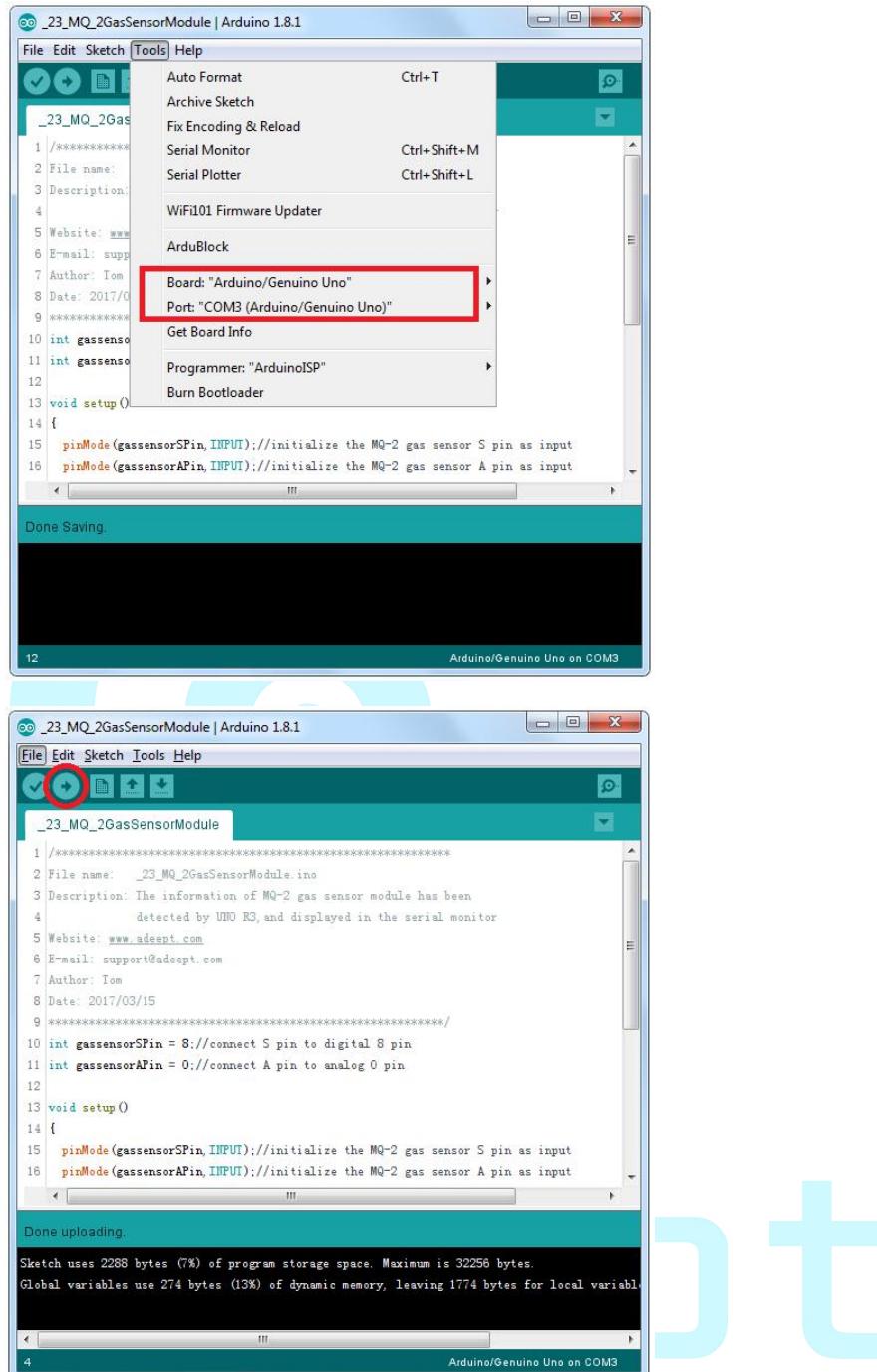
### Step 1: Build the circuit



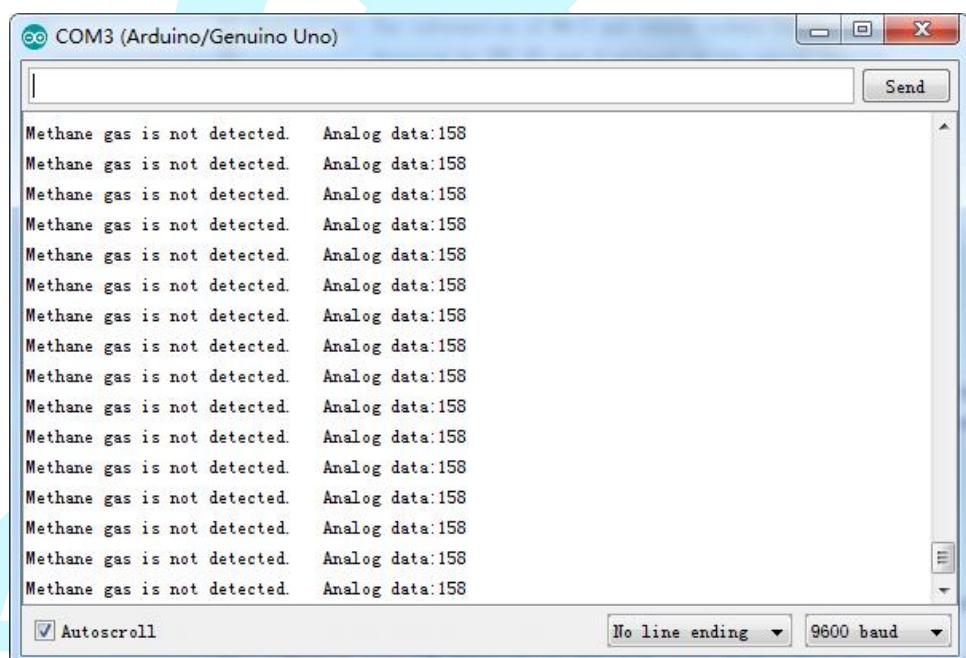
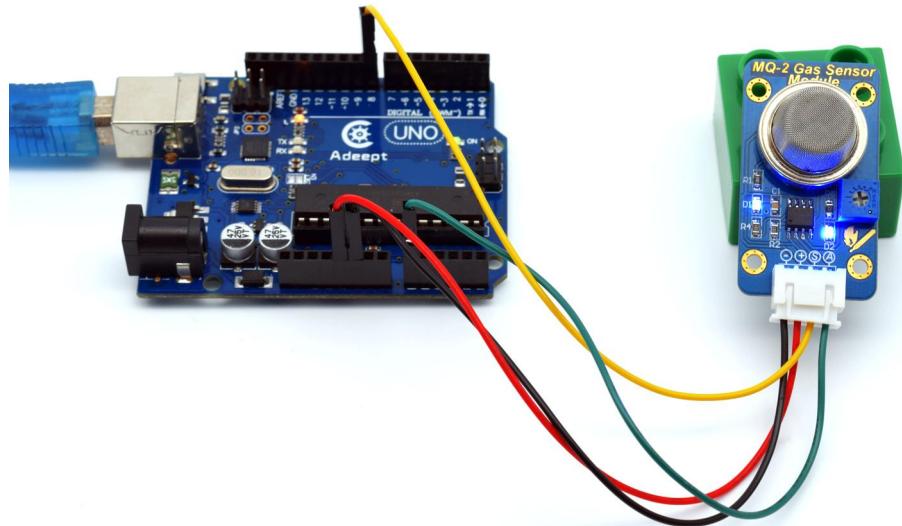
Adeept UNO R3 Board	MQ-2 Gas Sensor Module
A0	A
D8	S
5V	+
GND	-

**Step 2:** Program `_23_MQ_2GasSensorModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



Open the Serial Monitor in Arduino IDE. Release some methane near the module. And you will see the corresponding message on the window indicating flammable gases. Also the value output by the analog pin of the module will be printed.



# Lesson 24 Tracking Test

## Introduction

The Line Finder Module applies the principle that infrared rays reflect differently on surfaces of different colors. After electrified, the infrared diode on the module sends out infrared rays constantly. When they encounter a white surface, the diffused reflection happens and the reflected rays are received by the receiver on the module. On the other hand, when they come across a black one, the receiver cannot get any infrared.

Thus, the processor can tell whether it is a white or black detected surface by receiving the reflected infrared rays or not. Based on this, the module is usually used in line finding on a smart car.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Line Finder Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

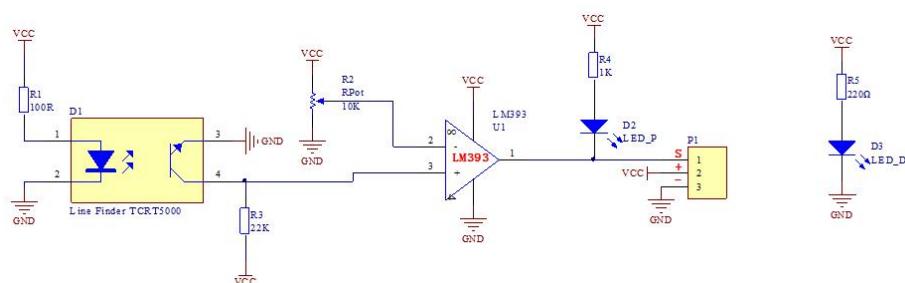
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

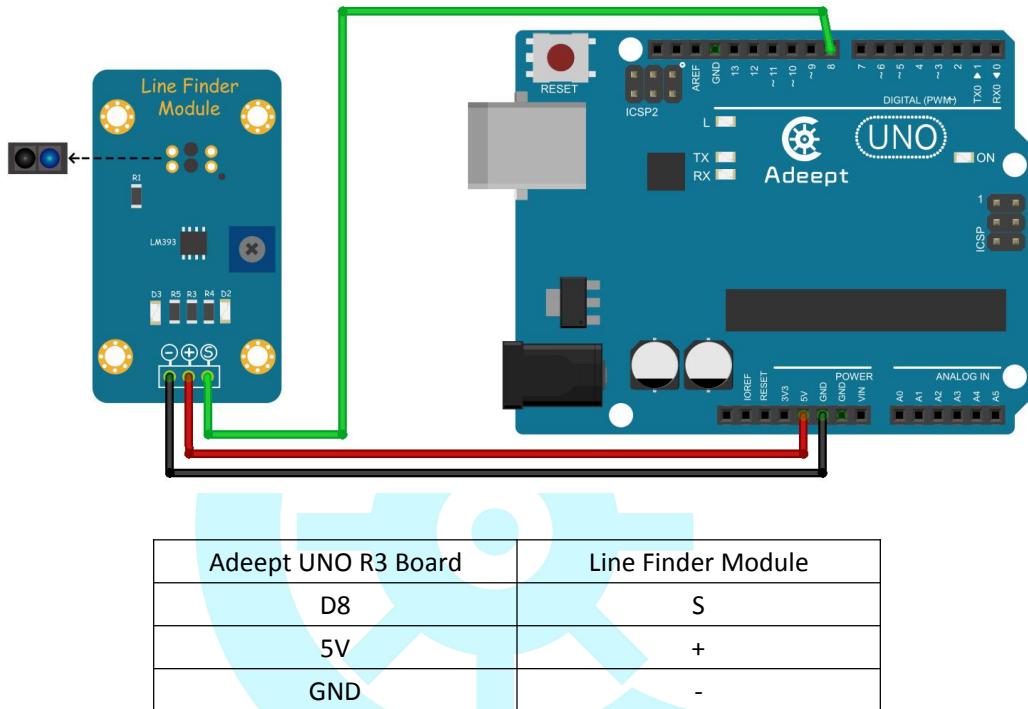
The schematic diagram:



In this experiment, we detect a piece of white and another of black paper via the Line Finder Module.

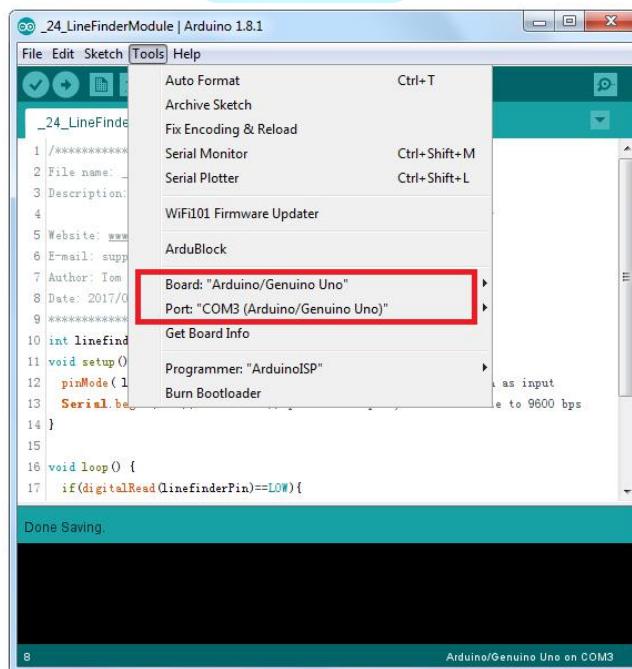
## Experimental Procedures

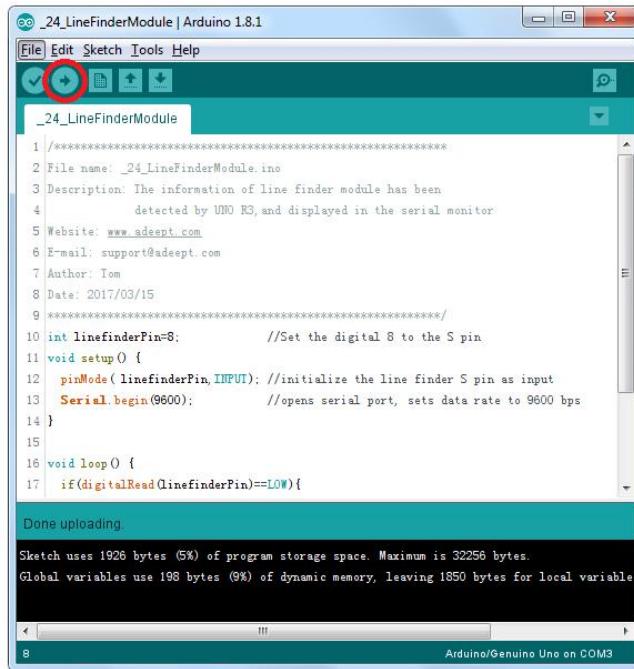
### Step 1: Build the circuit



**Step 2:** Program `_24_LineFinderModule.ino`

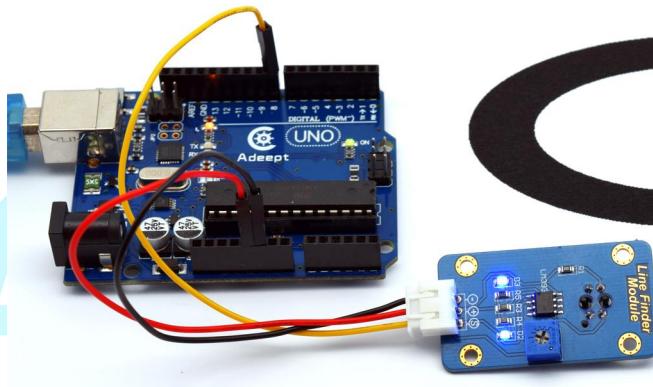
**Step 3:** Compile and download the sketch to the UNO R3 board.

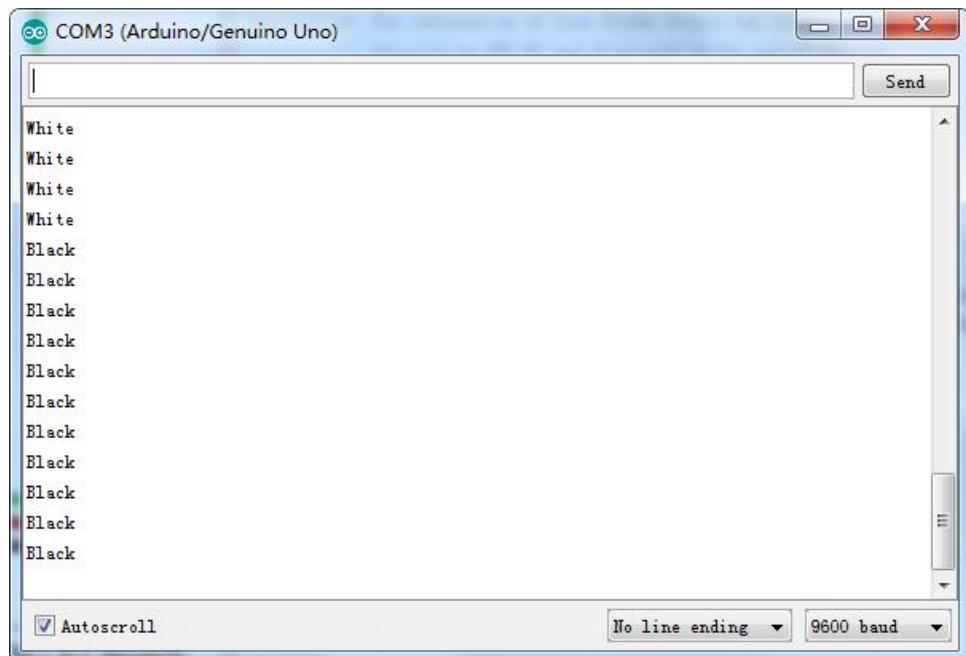




```
_24_LineFinderModule | Arduino 1.8.1
File Edit Sketch Tools Help
_24_LineFinderModule
1 //*****
2 File name: _24_LineFinderModule.ino
3 Description: The information of line finder module has been
4 detected by UNO R3, and displayed in the serial monitor
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/15
9 *****/
10 int linefinderPin=8;           //Set the digital 8 to the S pin
11 void setup() {
12   pinMode( linefinderPin, INPUT); //initialize the line finder S pin as input
13   Serial.begin(9600);          //opens serial port, sets data rate to 9600 bps
14 }
15
16 void loop() {
17   if(digitalRead(linefinderPin)==LOW){
18
Done uploading.
Sketch uses 1926 bytes (5%) of program storage space. Maximum is 32256 bytes.
Global variables use 198 bytes (9%) of dynamic memory, leaving 1850 bytes for local variables.
```

Open the Serial Monitor in Arduino IDE. Place the sensor module over a piece of white paper and another of black and you will see the data detected on the window. You can adjust the blue potentiometer on the module to change the sensitivity.





# Lesson 25 How To Use Slide Potentiometer

## Introduction

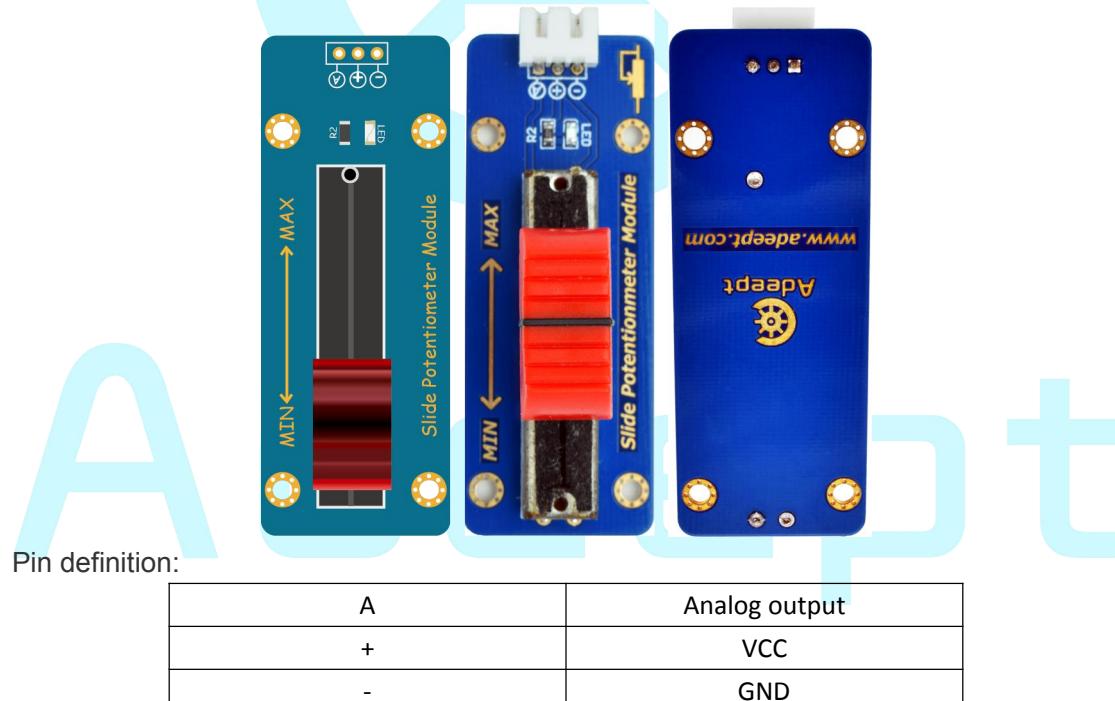
The similarity of the Slide Potentiometer Module and Potentiometer Module lies in: holding three terminals; changing the resistance between the changeable terminal and one end by changing the position of the slider. When the difference is: the Slide Potentiometer usually has a larger power (and size) and can be used directly as a load or connected in serial in the circuit of the load for current limiting. The potentiometer has a smaller power and size, and generally used for voltage sampling in signal circuit.

## Components

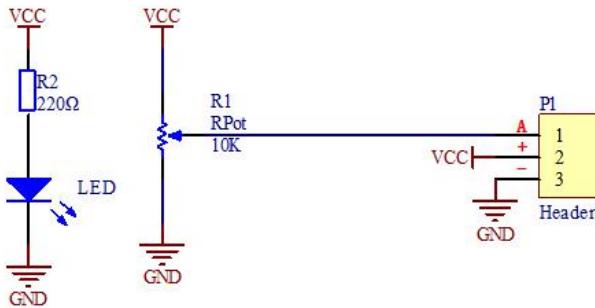
- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Slide Potentiometer Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Experimental Principle

The Fritzing image:



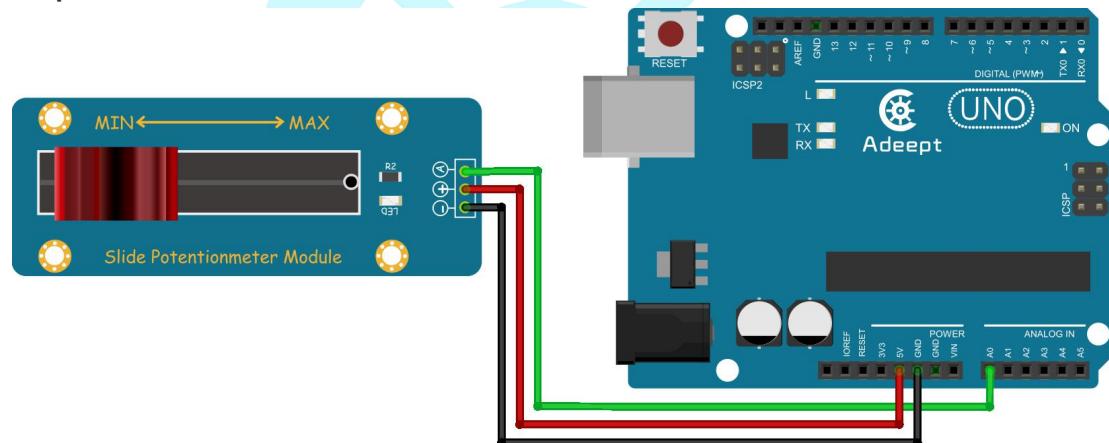
The schematic diagrams:



This experiment programs the Arduino board and collects analog quantities output from the Slide Potentiometer module via pin A0 of the board, and converts them into digital ones and display the value on the computer by serial port.

## Experimental Procedures

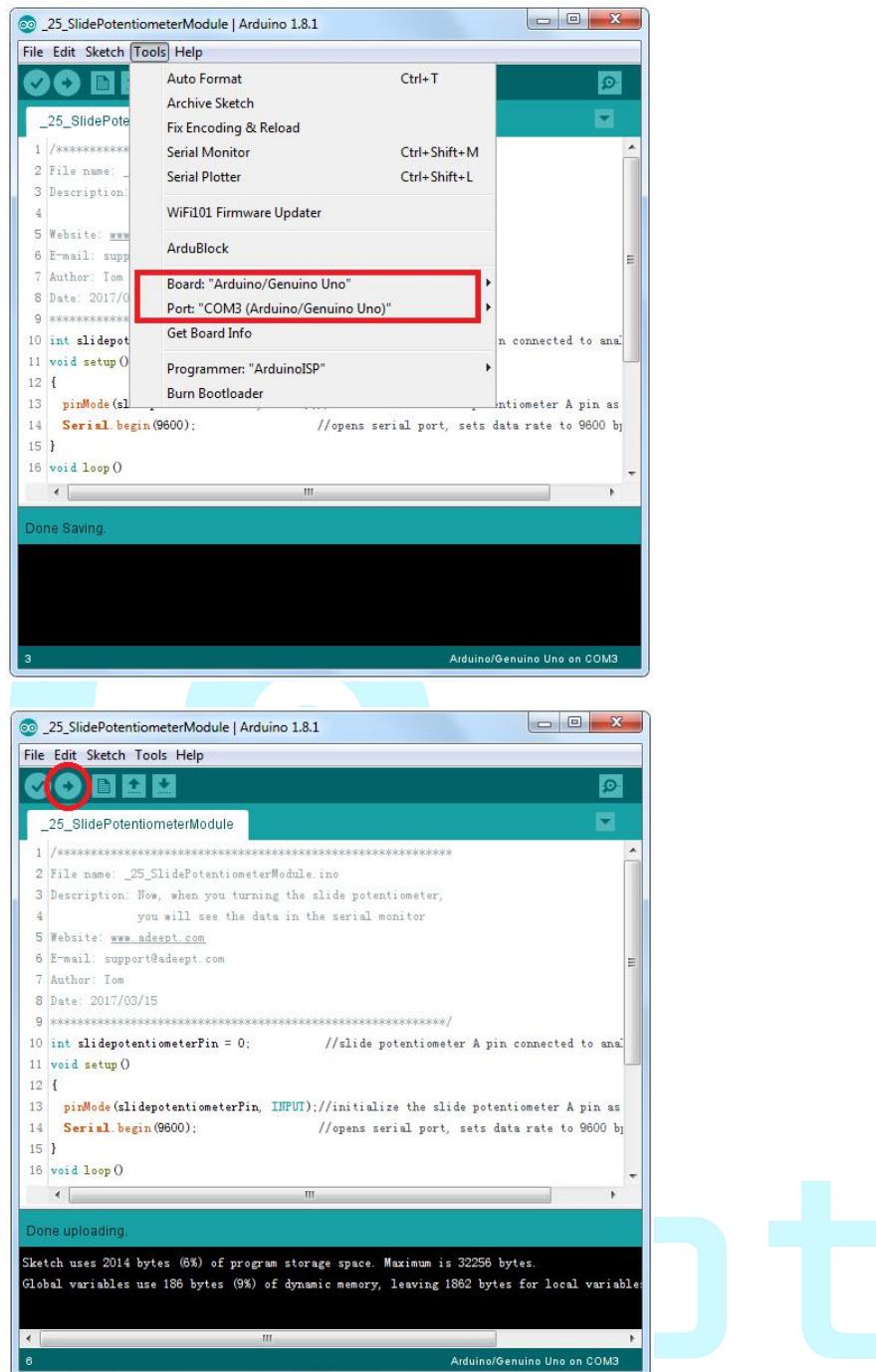
**Step 1:** Build the circuit



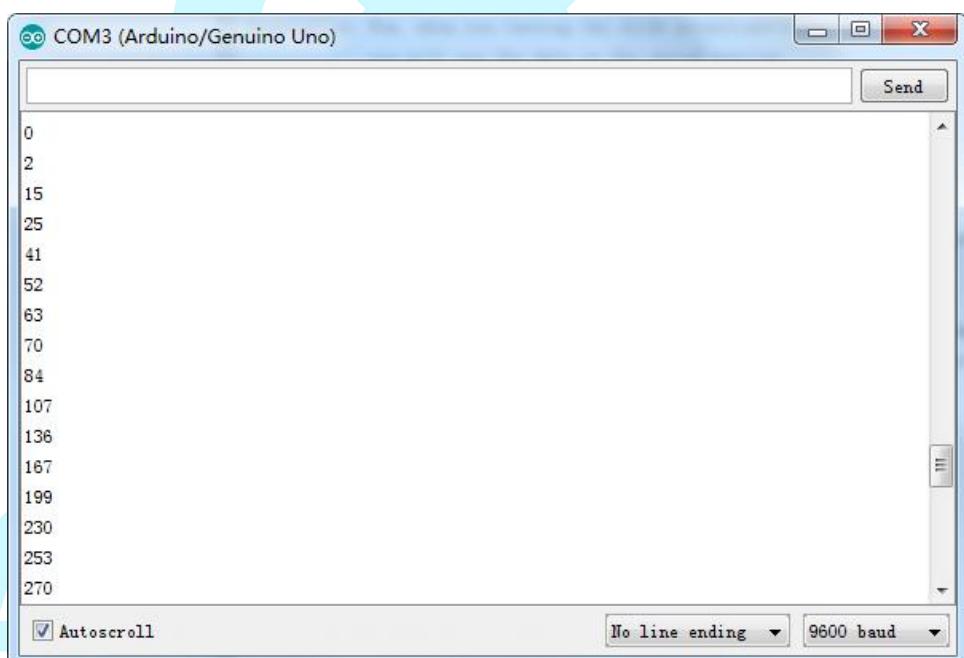
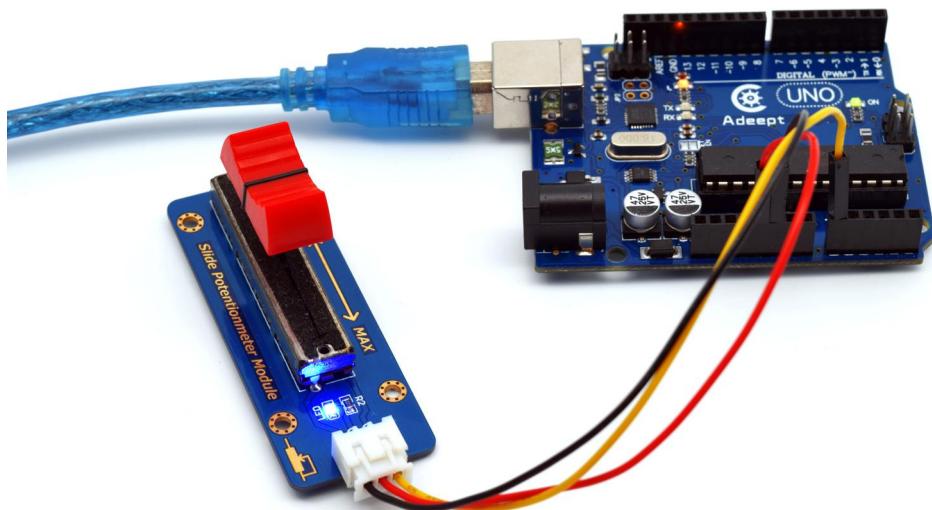
Adeept Uno R3 Board	Slide Potentiometer Module
A0	A
5V	+
GND	-

**Step 2:** Program `_25_SlidePotentiometerModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



Open the Serial Monitor in Arduino IDE. Move the slide of the Slide Potentiometer module. Then the value output by port A of the module will be displayed on the Serial Monitor. Slide it toward MIN and the value on the window will decrease; slide it toward MAX, it will increase.



# Lesson 26 Small Fan Works

## Introduction

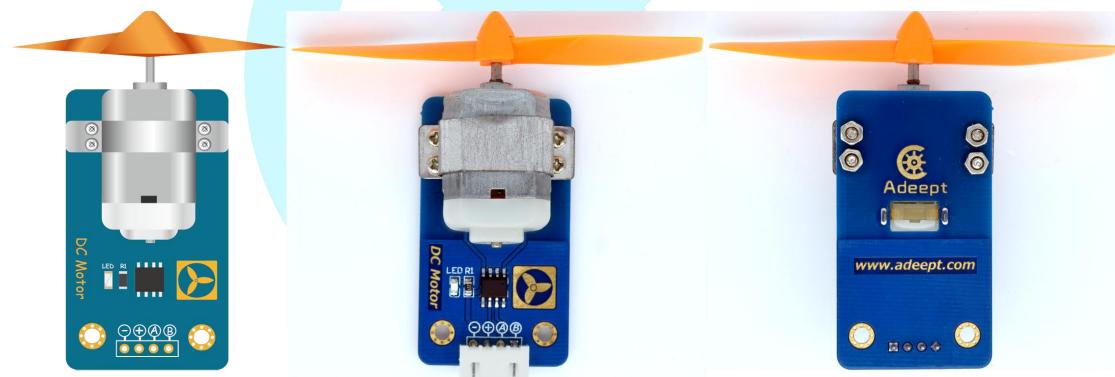
DC motor is a device that converts electrical energy into mechanical energy. Due to the ease of control, it is usually used in fan, electronic toy, shaver, etc.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* DC Motor Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

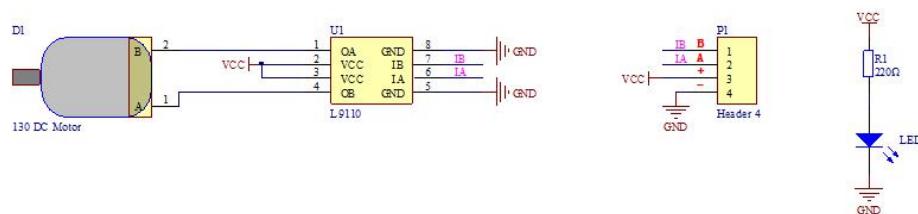
The Fritzing image:



Pin definition:

B	Digital output
A	Digital output
+	VCC
-	GND

The schematic diagram:



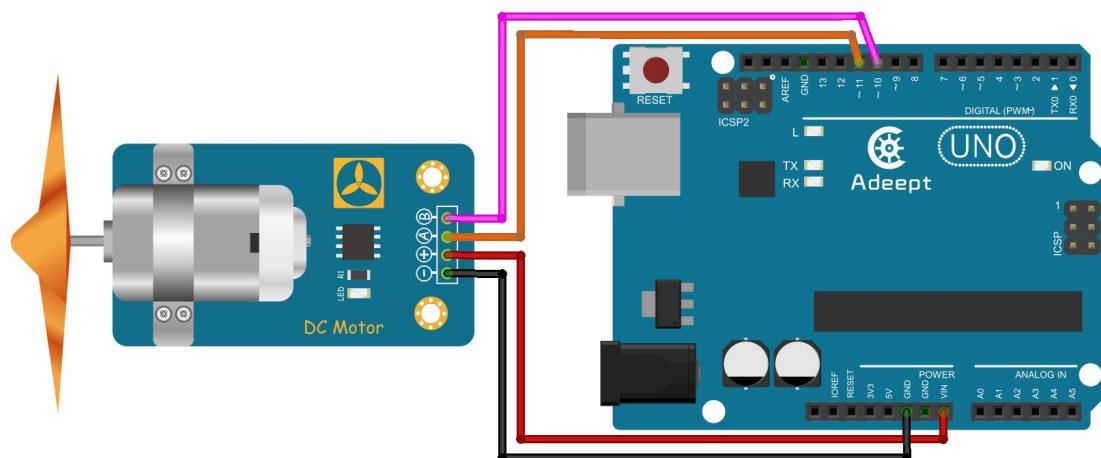
This experiment is to control the status of the DC motor via the Arduino board. The statuses include running, stop, forward, reversing, acceleration, and deceleration.

**note:**

We need to connect the power first. After powering on the module, we can connect the module signal pins. Otherwise the motor module will be hot and may not work.

## Experimental Procedures

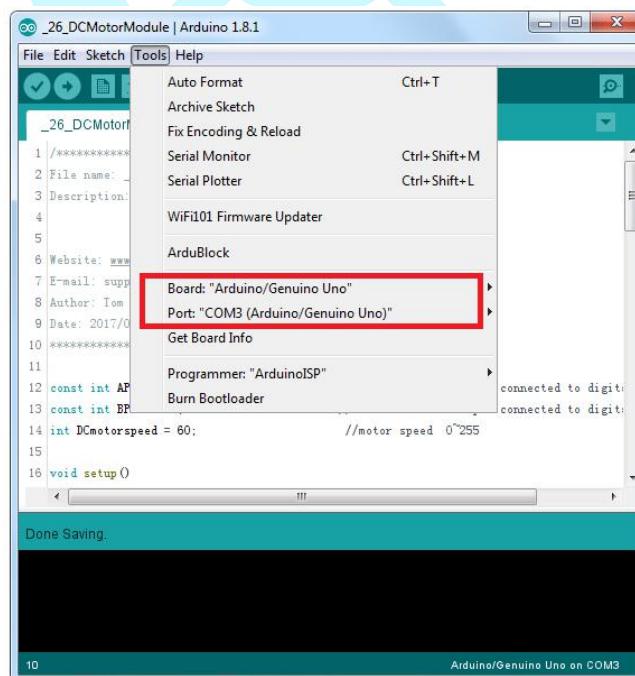
### Step 1: Build the circuit

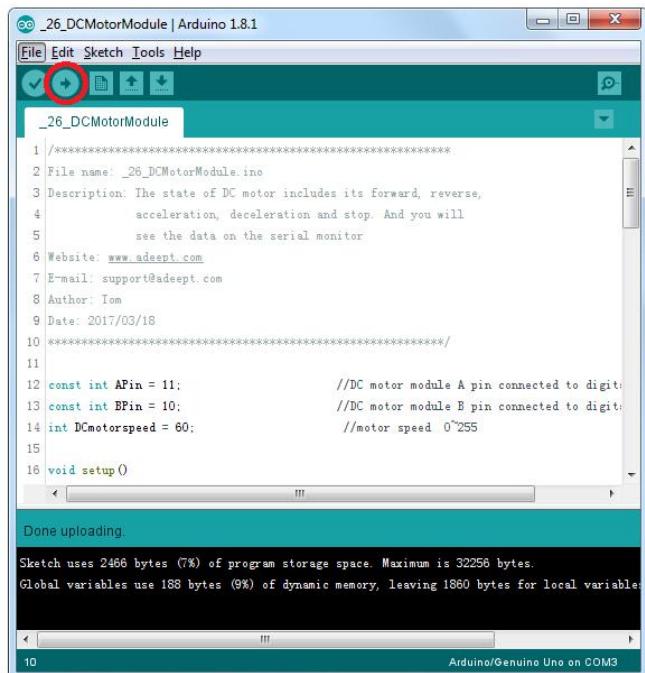


Adeept Uno R3 Board	DC Motor Module
D10	B
D11	A
Vin(+5V)	+
GND	-

Step 2: Program \_26\_DCMotorModule.ino

Step 3: Compile and download the sketch to the UNO R3 board.



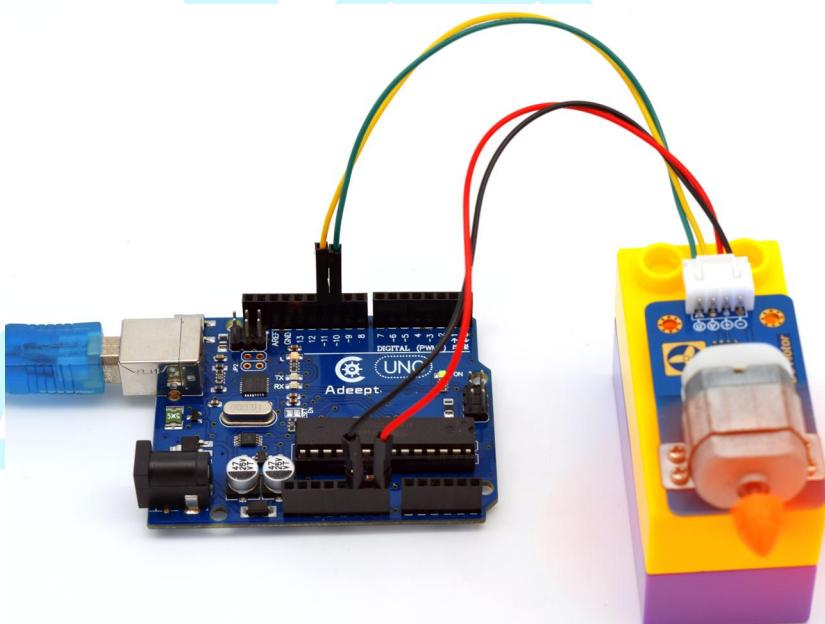


```
_26_DCMotorModule | Arduino 1.8.1
File Edit Sketch Tools Help
_26_DCMotorModule
1 //*****
2 File name: _26_DCMotorModule.ino
3 Description: The state of DC motor includes its forward, reverse,
4           acceleration, deceleration and stop. And you will
5           see the data on the serial monitor
6 Website: www.adeept.com
7 E-mail: support@adeept.com
8 Author: Tom
9 Date: 2017/03/18
10 *****/
11
12 const int APin = 11;           //DC motor module A pin connected to digital pin 11
13 const int BPin = 10;           //DC motor module B pin connected to digital pin 10
14 int DCmotorspeed = 60;        //motor speed 0~255
15
16 void setup()
{
}
Done uploading.

Sketch uses 2466 bytes (7%) of program storage space. Maximum is 32256 bytes.
Global variables use 188 bytes (9%) of dynamic memory, leaving 1860 bytes for local variables.

10
Arduino/Genuino Uno on COM3
```

Now you can see the fan rotates forward, accelerate, speed down, stop, start and accelerate, speed down again, and stop again. At the same time the value of the speed set will be sent to and displayed on Serial Monitor.



# Lesson 27 How To Use The Joystick

## Introduction

The PS2 Joystick Module is an input device. It consists of a station and the control knob on side. It functions by sending angle or direction signals to the device controlled. The button on the module can also be recognized by the microcontroller. The module supports two-channel analog output, namely, x- and y-axis offset, and one-channel digital output which indicates whether the user has pressed the button at z-axis or not. The Joystick Module can be used to easily control the object to move in a three-dimensional space. For example, it can be applied to control crane, truck, electronic games, robots, etc.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Joystick Module
- 1 \* USB Cable
- 1 \* 5-Pin Wires

## Experimental Principle

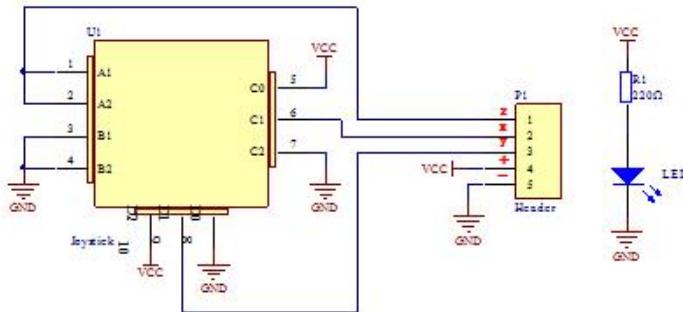
The Fritzing image:



Pin definition:

z	Digital key output
x	Analog output(X)
y	Analog output(Y)
+	VCC
-	GND

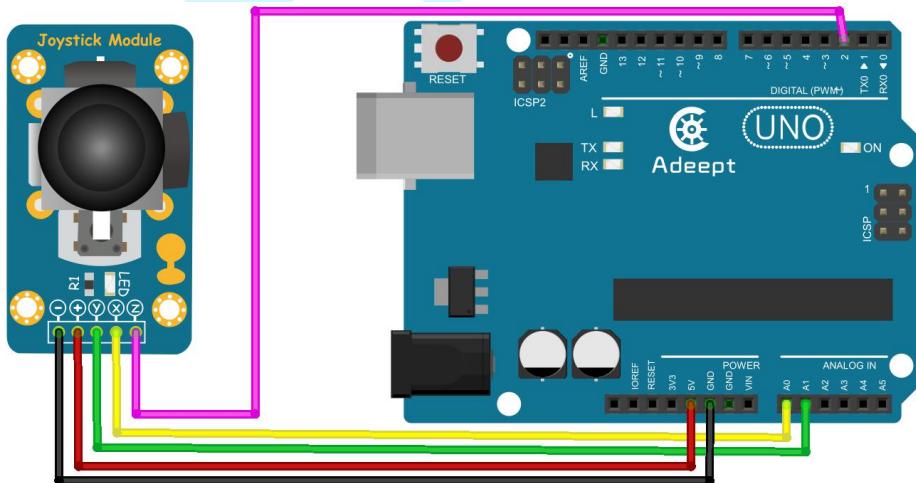
The schematic diagram:



The experiment reads the status of the PS2 Joystick Module, send the data to and display it on Serial Monitor.

## Experimental Procedures

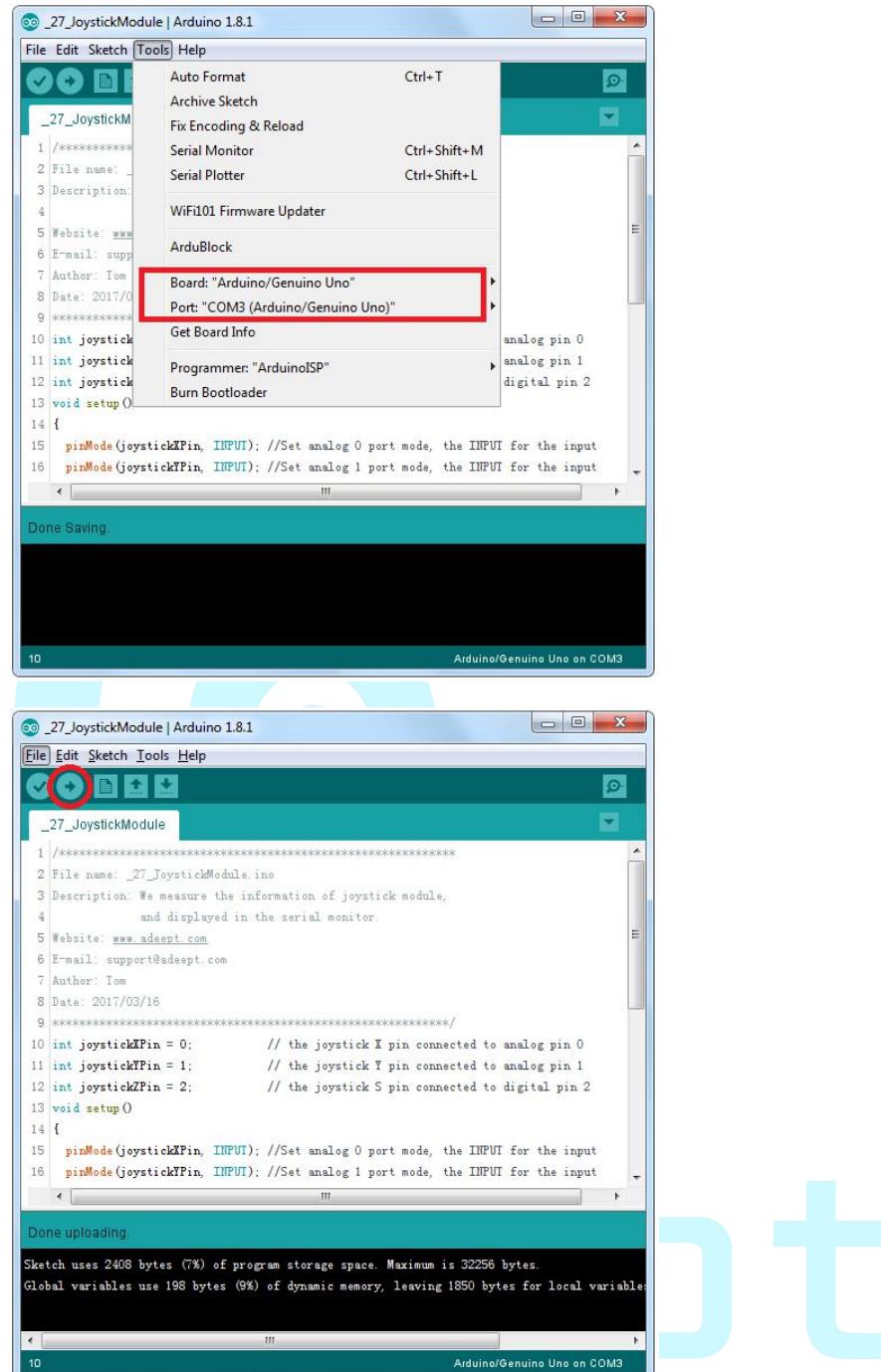
### Step 1: Build the circuit



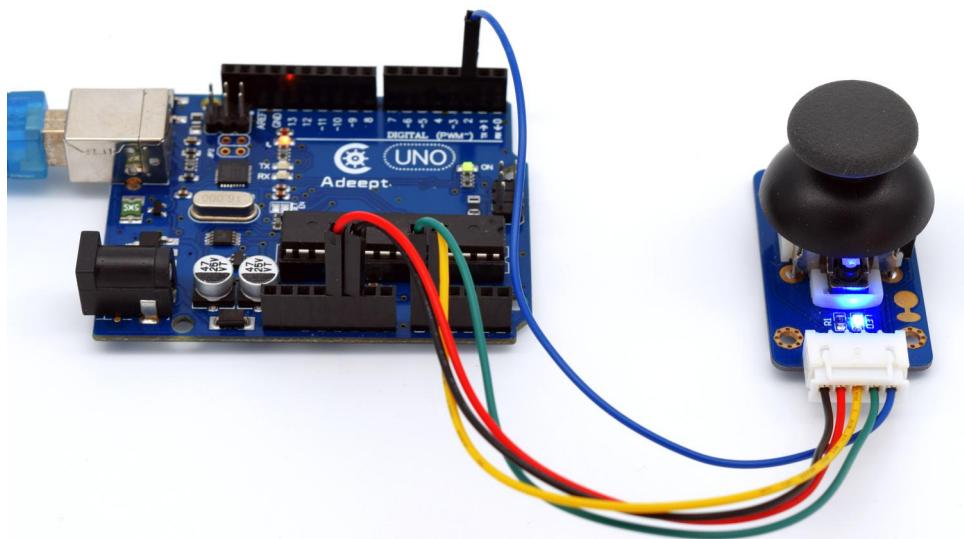
Adeebt UNO R3 Board	Joystick Module
D2	z
A0	x
A1	y
5V	+
GND	-

**Step 2:** Program `_27_JoystickModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



Open Serial Monitor of the Arduino IDE. Press or pull the knob and you will see the value of current status displayed on the window.



COM3 (Arduino/Genuino Uno)

```
X:505 Y:516 Z:1
X:506 Y:516 Z:1
X:254 Y:516 Z:1
X:0 Y:517 Z:1
X:0 Y:516 Z:1
X:0 Y:516 Z:1
X:0 Y:515 Z:1
X:8 Y:657 Z:1
X:70 Y:764 Z:0
X:130 Y:788 Z:0
X:229 Y:790 Z:0
X:359 Y:825 Z:0
X:471 Y:826 Z:0
X:488 Y:824 Z:0
X:501 Y:822 Z:0
X:502 Y:813 Z:0
```

Autoscroll      No line ending      9600 baud

# Lesson 28 How To Use The MIC Module

## Introduction

The MIC Module is composed of a small microphone and an LM393 voltage comparator. It can capture minor sound signals and convert them into electric ones. The threshold of the comparator can be adjusted by the blue potentiometer on the module. This module can be applied in sound alarm system.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* MIC Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

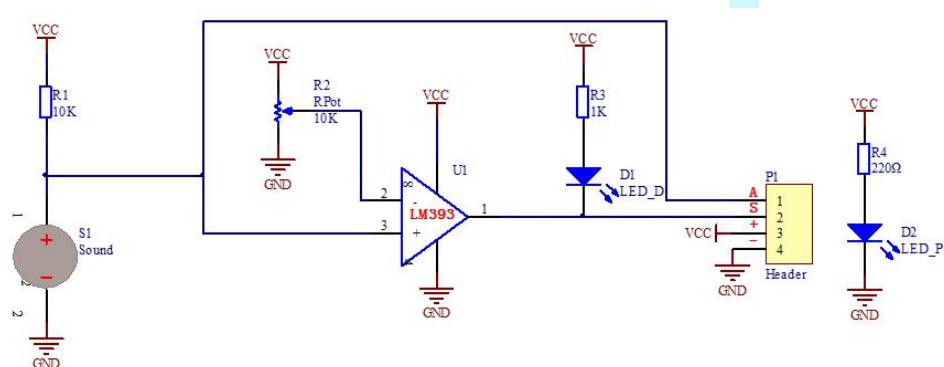
The Fritzing image:



Pin definition:

S	Digital output
A	Analog output
+	VCC
-	GND

The schematic diagram:

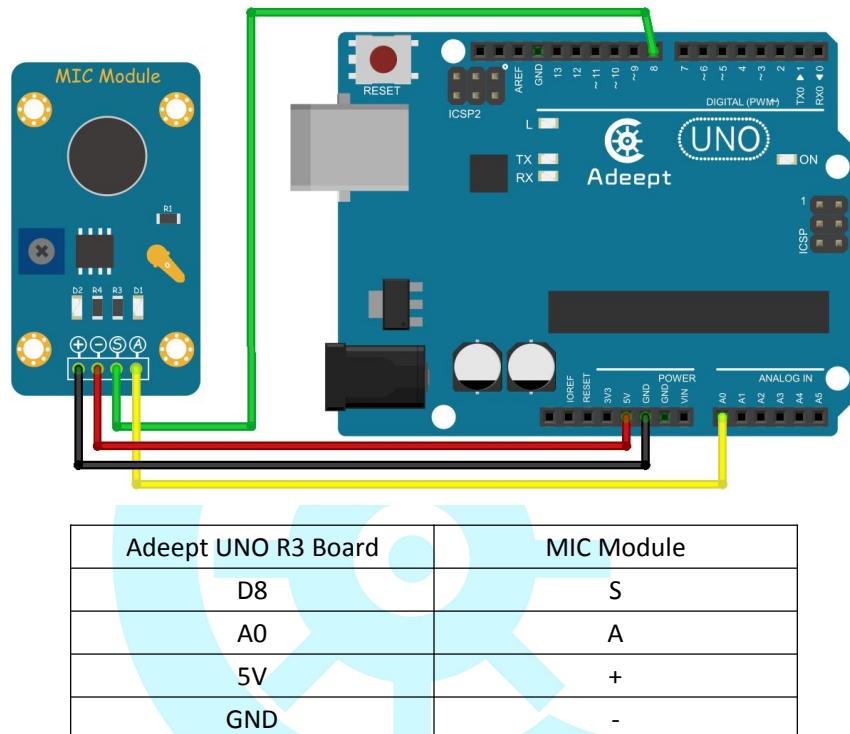


This experiment uses the MIC Module to detect the sound and display the data on Serial

Monitor via the serial port.

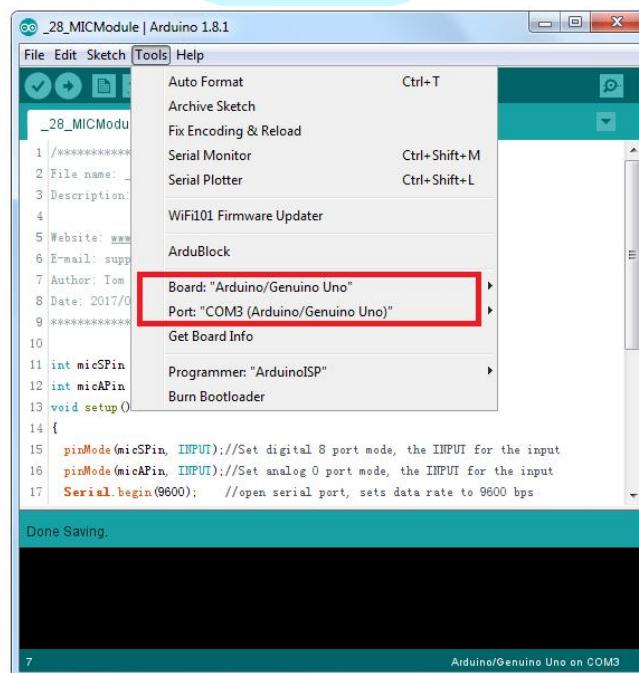
## Experimental Procedures

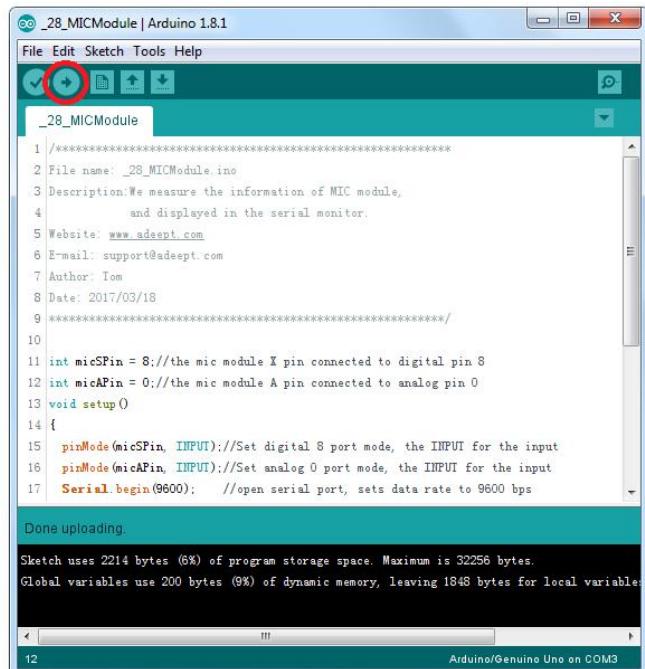
### Step 1: Build the circuit



Step 2: Program \_28\_MICModule.ino

Step 3: Compile and download the sketch to the UNO R3 board.



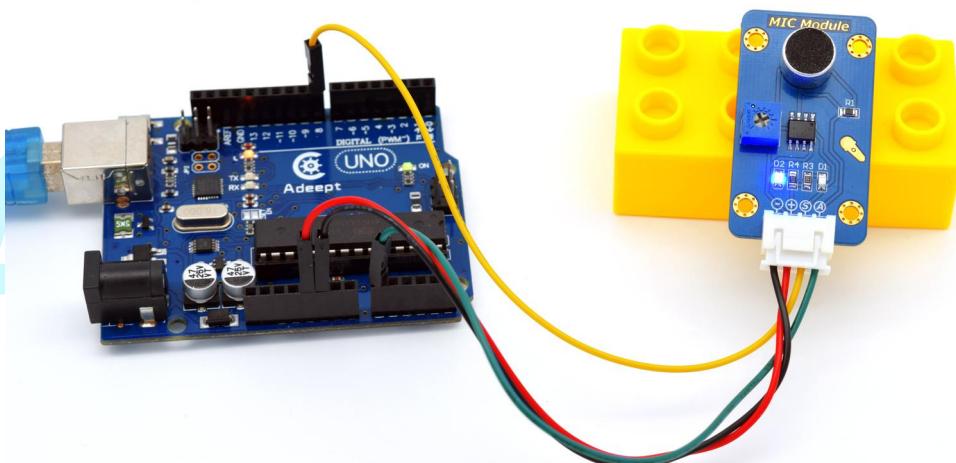


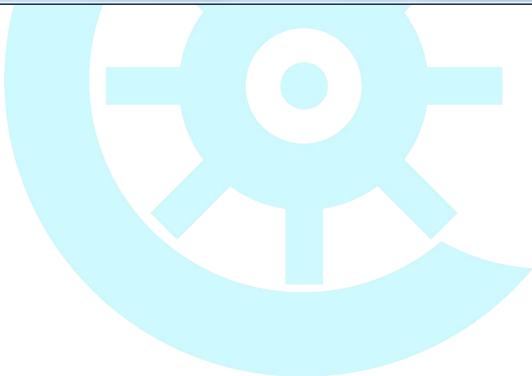
```
_28_MICModule | Arduino 1.8.1
File Edit Sketch Tools Help
_28_MICModule
1 //*****
2 File name: _28_MICModule.ino
3 Description:# measure the information of MIC module,
4         and displayed in the serial monitor.
5 Website: www.adeept.com
6 E-mail: support@adeept.com
7 Author: Tom
8 Date: 2017/03/18
9 *****/
10
11 int micSPin = 8;//the mic module X pin connected to digital pin 8
12 int micAPin = 0;//the mic module A pin connected to analog pin 0
13 void setup()
14 {
15   pinMode(micSPin, INPUT);//Set digital 8 port mode, the INPUT for the input
16   pinMode(micAPin, INPUT);//Set analog 0 port mode, the INPUT for the input
17   Serial.begin(9600);    //open serial port, sets data rate to 9600 bps
}
Done uploading.

Sketch uses 2214 bytes (6%) of program storage space. Maximum is 32256 bytes.
Global variables use 200 bytes (9%) of dynamic memory, leaving 1848 bytes for local variables

12
Arduino/Genuino Uno on COM3
```

Open Serial Monitor of the Arduino IDE. Blow at the MIC Module or make some other sounds near it, and you can see the value on the window indicating the sound intensity. The higher the sound volume, the larger value on the window; the lower volume, the smaller value.





```
COM3 (Arduino/Genuino Uno)
Send
NO sound458
NO sound451
NO sound462
NO sound447
NO sound456
NO sound456
NO sound447
NO sound475
NO sound460
NO sound460
NO sound457
NO sound438
Sound422
NO sound474
NO sound467
NO sound456
Autoscroll No line ending 9600 baud
```

Adeept

# Lesson 29 How To Use The Relay Module

## Introduction

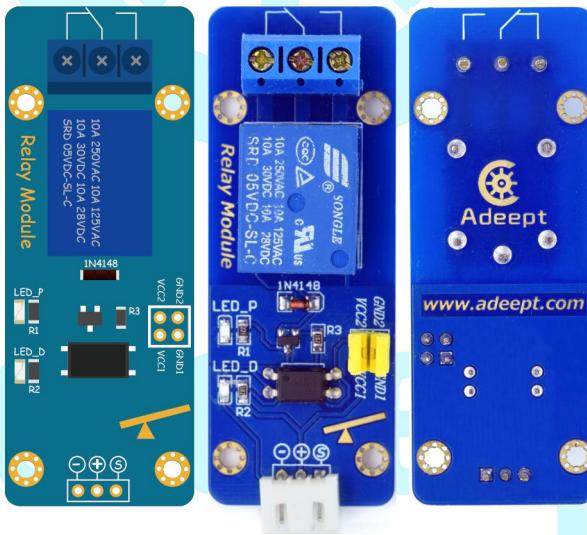
The relay is an electronic and electrical component that controls large currents by small currents. In the course of building an Arduino project, generally many large current or high volume devices like solenoid valve, lamp and motor cannot be connected directly to digital I/Os of the Arduino board. At this moment, a relay can save your project.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Relay Module
- 1 \* LED Module
- 1 \* USB Cable
- 2 \* 3-Pin Wires
- 3 \* Hookup Wire Set
- 1 \* Breadboard

## Experimental Principle

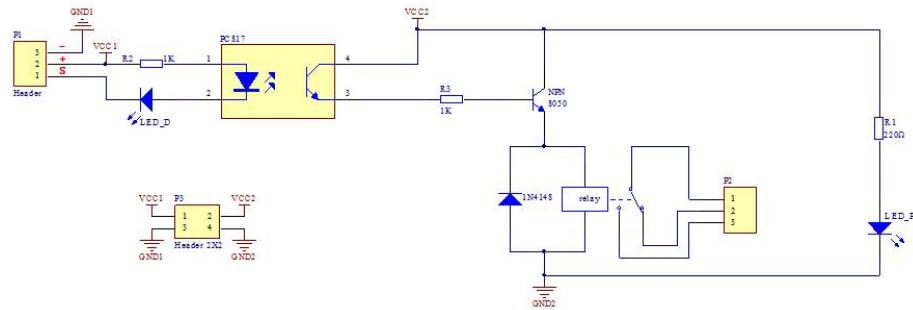
The Fritzing image:



Pin definition:

S	Digital Data Input
+	VCC1
-	GND1
VCC1	VCC1
GND1	GND1
VCC2	VCC1
GND2	GND2

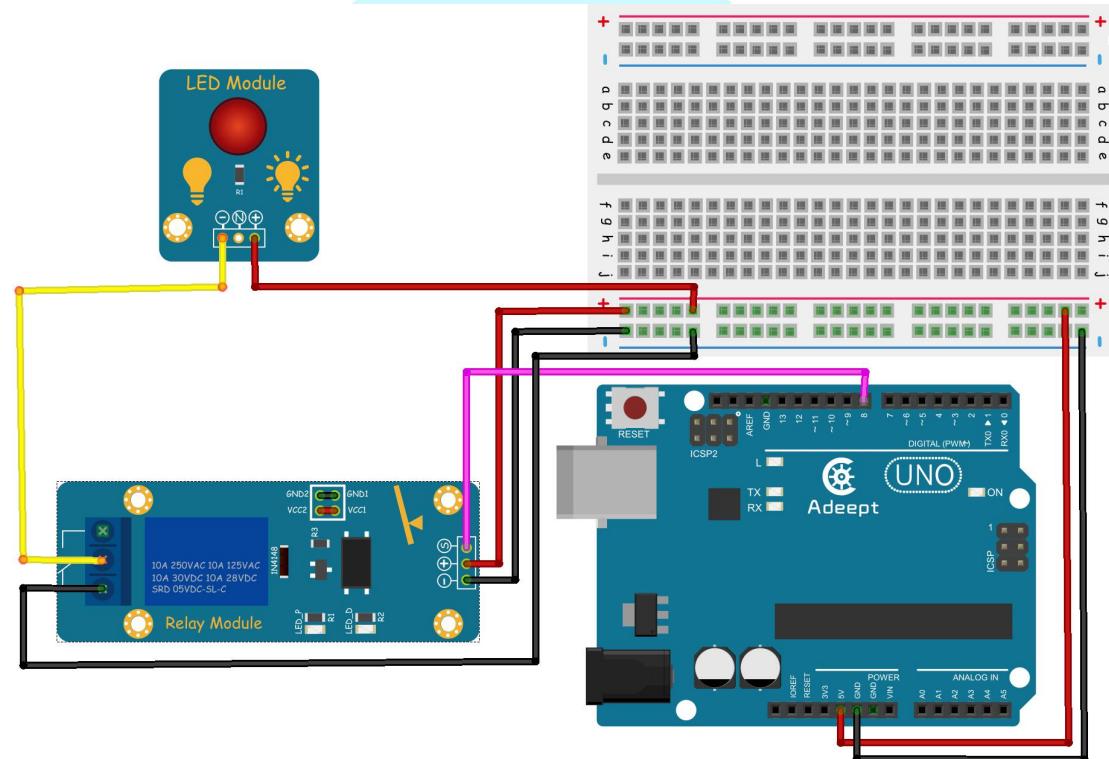
The schematic diagram:



This experiment is to control an LED to brighten and dim by a relay.

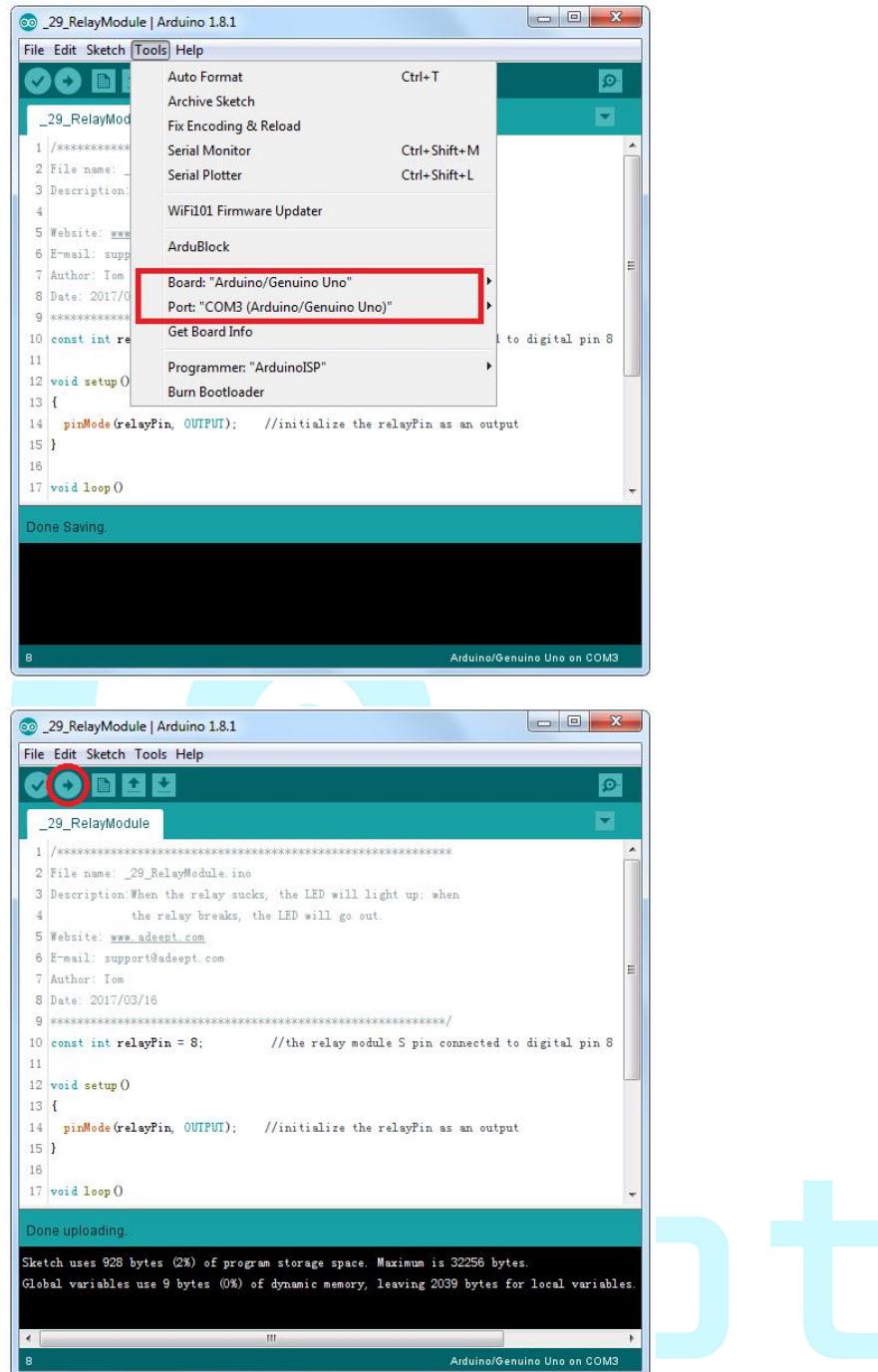
## Experimental Procedures

### Step 1: Build the circuit

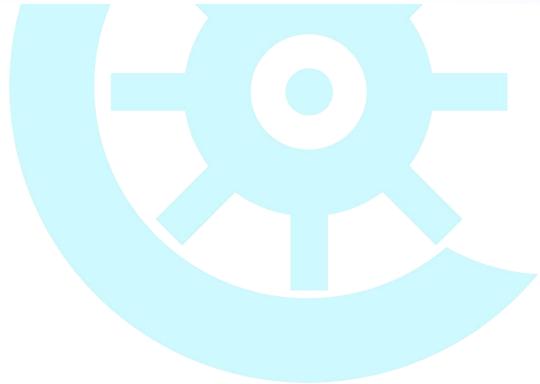
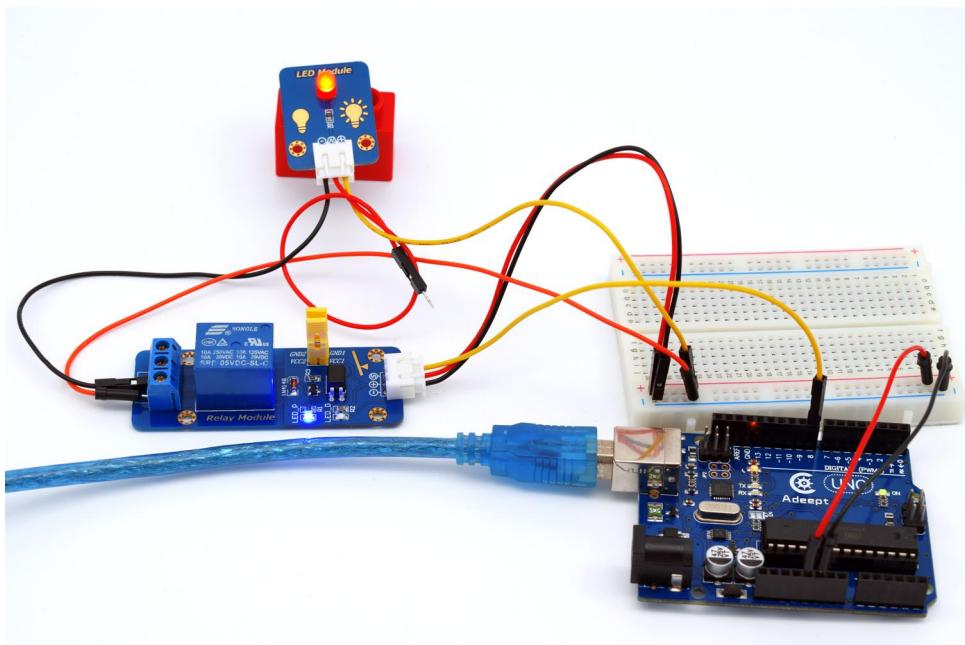


**Step 2:** Program `_29_RelayModule.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.



Now you can see the LED on the LED Module flickers every 2s and can hear the sound of relay closing and opening.



# Adeept

# Lesson 30 How To Use The Segment Module

## Introduction

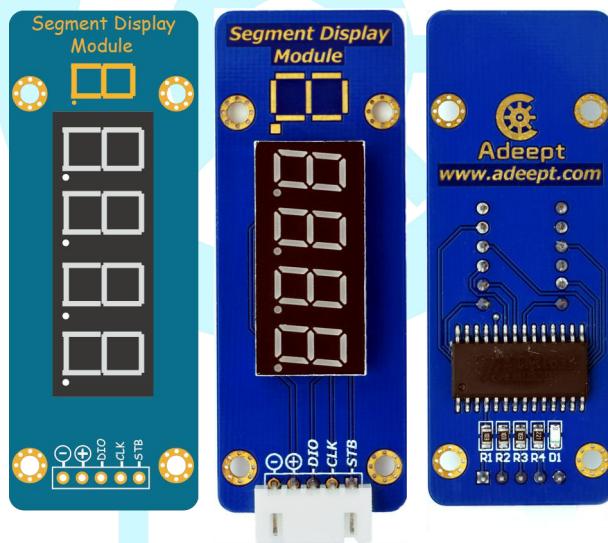
The module consists of a 4-bit 7-segment common-cathode (CC) diode and a control chip TM1638. It communicates with the Arduino board via three wires and can show numbers and simple characters.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Segment Display Module
- 1 \* USB Cable
- 1 \* 5-Pin Wires

## Experimental Principle

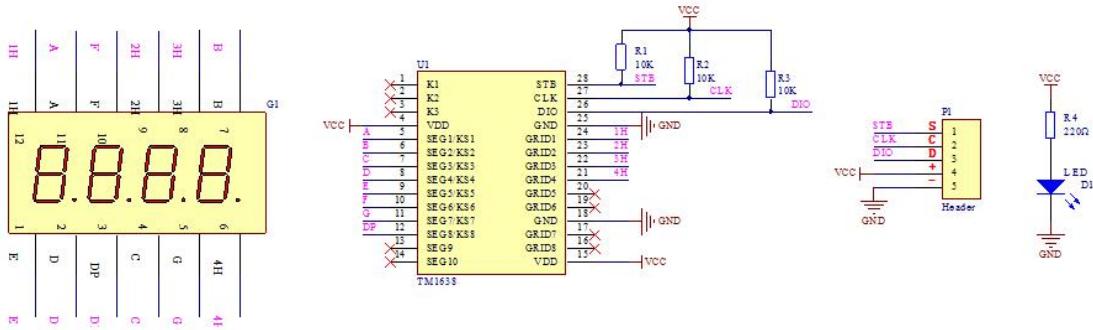
The Fritzing image:



Pin definition:

STB	Digital input
CLK	Digital input
DIO	Digital input
+	VCC
-	GND

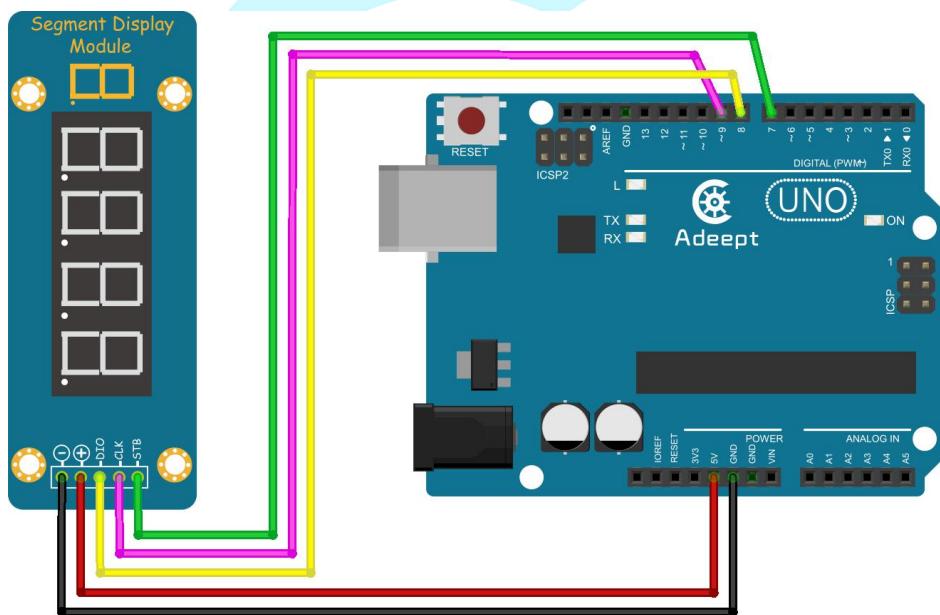
The schematic diagram:



Through programming the Arduino board, make the module display 0000~9999.

## Experimental Procedures

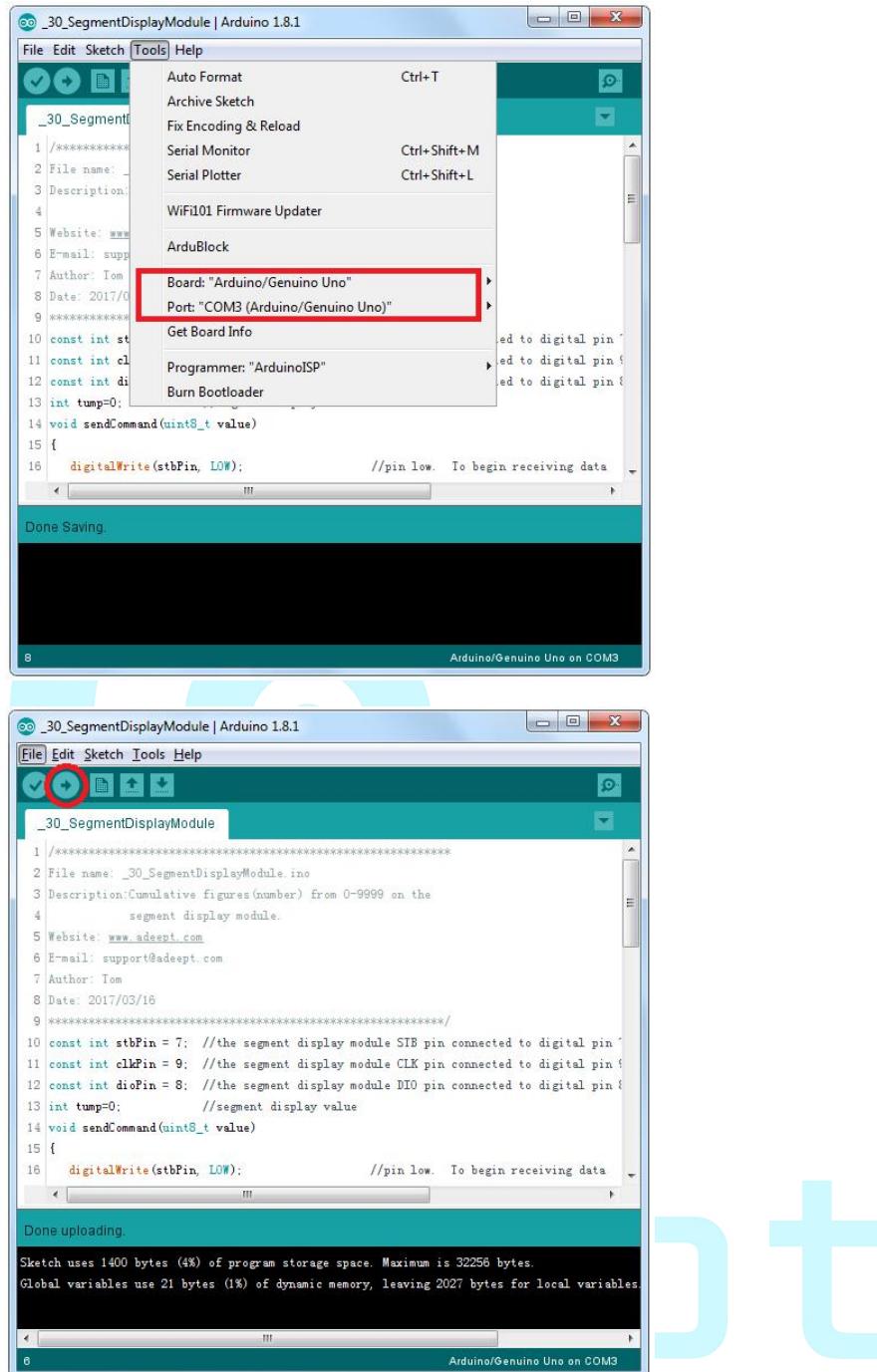
### Step 1: Build the circuit



Adeept Uno R3 Board	Segment Display Module
D7	STB
D9	CLK
D8	DIO
5V	+
GND	-

**Step 2:** Program \_30\_SegmentDisplayModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



Now you can see the number 0~9999 shown repeatedly on the digital display.



# Adeept

# Lesson 31 How To Use The 8\*8 LED Matrix

## Introduction

The module drives the 8\*8 LED Matrix Module by cascading two 74HC595 chips. The module communicates with the microcontroller through SPI. It only occupies three I/Os of the Arduino board and save precious ones for connecting other devices.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* 8\*8 LED Matrix Module
- 1 \* USB Cable
- 1 \* 5-Pin Wires

## Experimental Principle

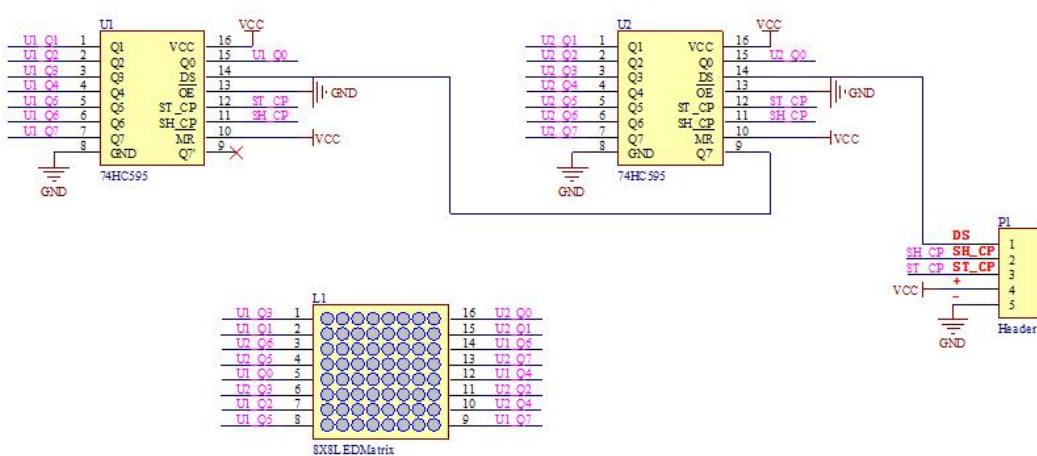
The Fritzing image:



Pin definition:

DS	Digital input
SH_CP	Digital input
ST_CP	Digital input
+	VCC
-	GND

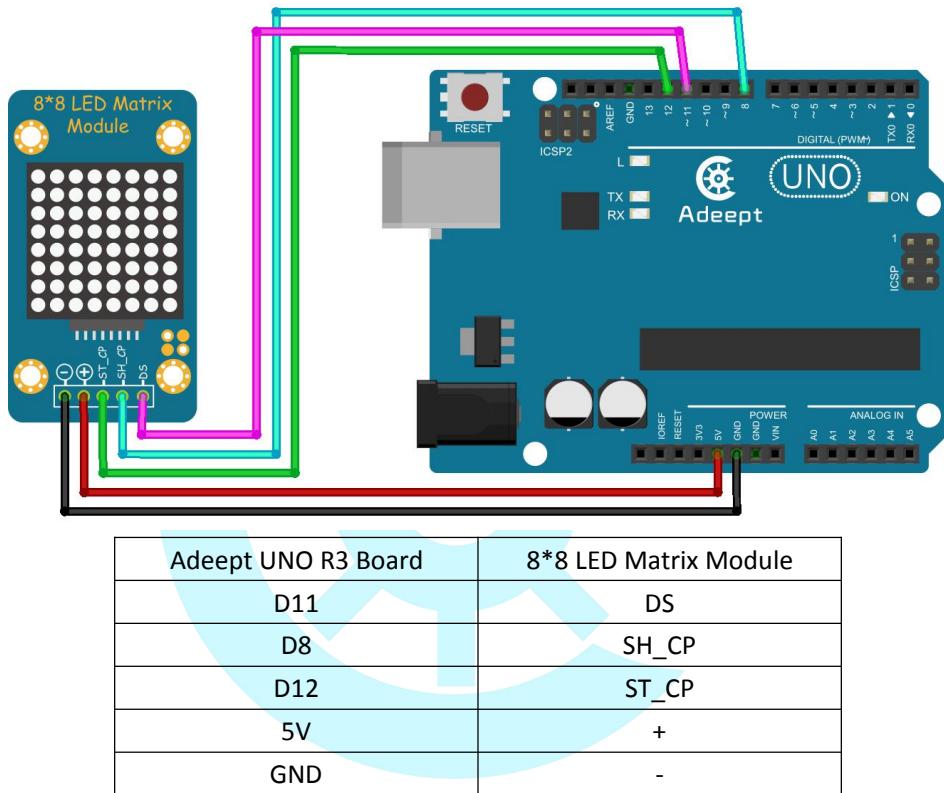
The schematic diagram:



In this experiment, by programming the Arduino board, we send the data to the dot matrix module via the SPI interface and make the display scroll the characters “Adeept”.

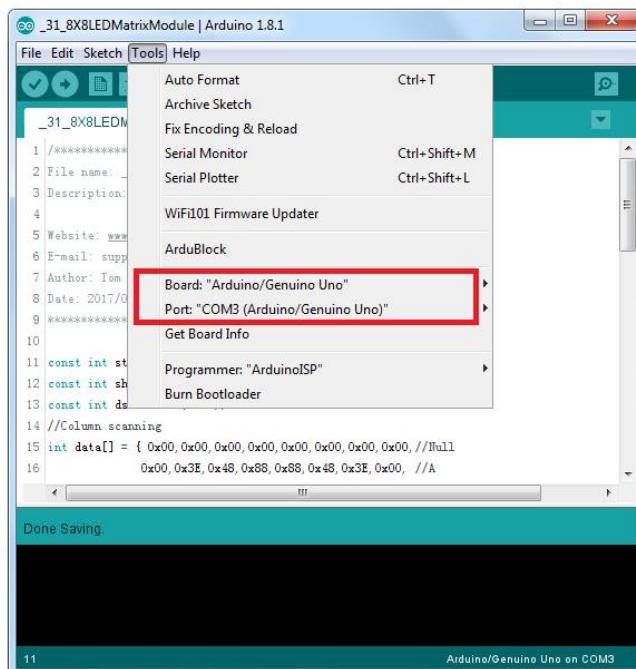
## Experimental Procedures

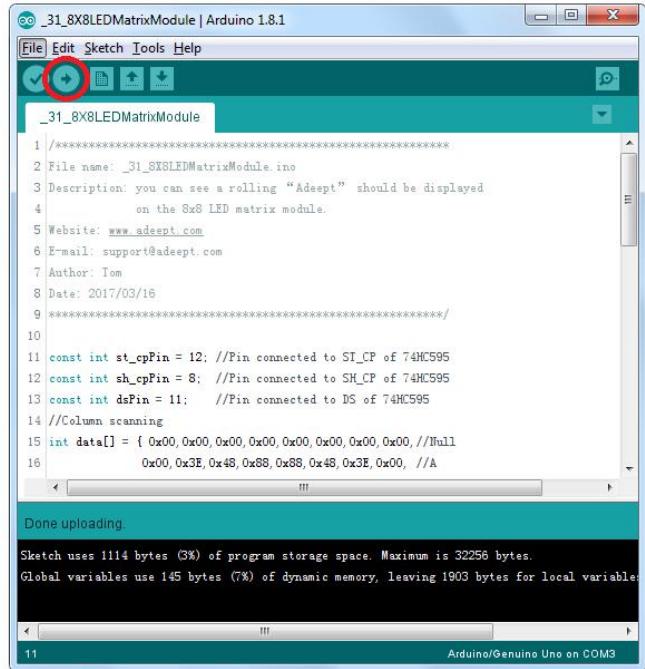
## Step 1: Build the circuit



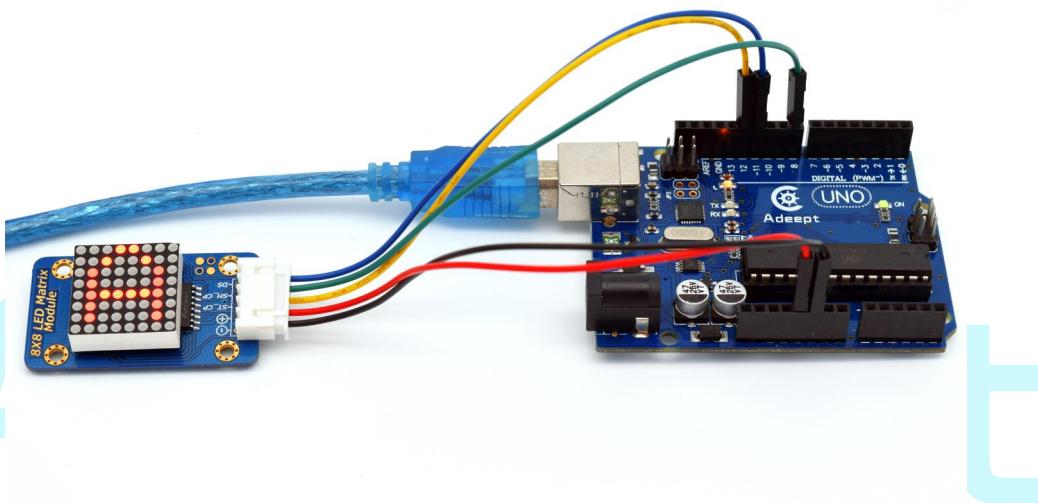
## **Step 2: Program \_31\_8X8LEDMatrixModule.ino**

**Step 3:** Compile and download the sketch to the UNO R3 board.





Now you can see on the dot matrix module, "Adept" is displayed in the way of scrolling.



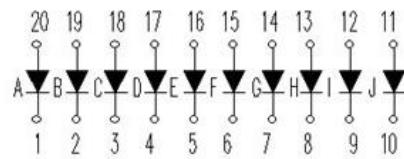
## Lesson 32 Indication of Signal

### Introduction

The LED bar is an analog indicating component usually used for volume indication.



The internal schematic diagram for the LED bar graph is as shown below:

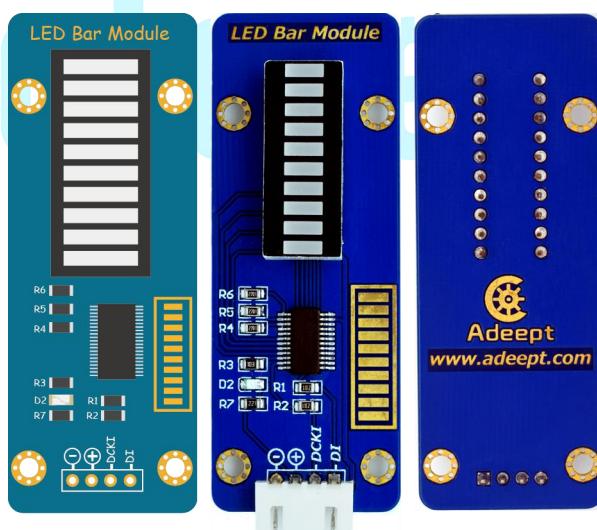


### Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* LED Bar Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

### Experimental Principle

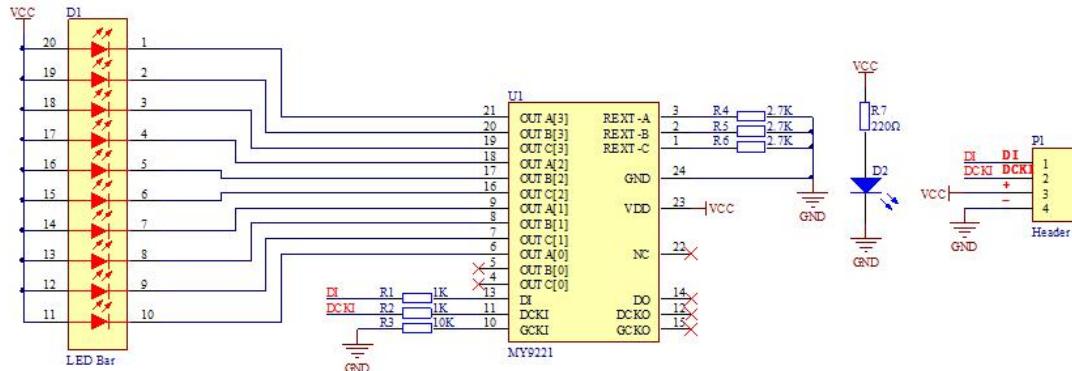
The Fritzing image:



## Pin definition:

DI	Digital input
DCLK	Digital input
+	VCC
-	GND

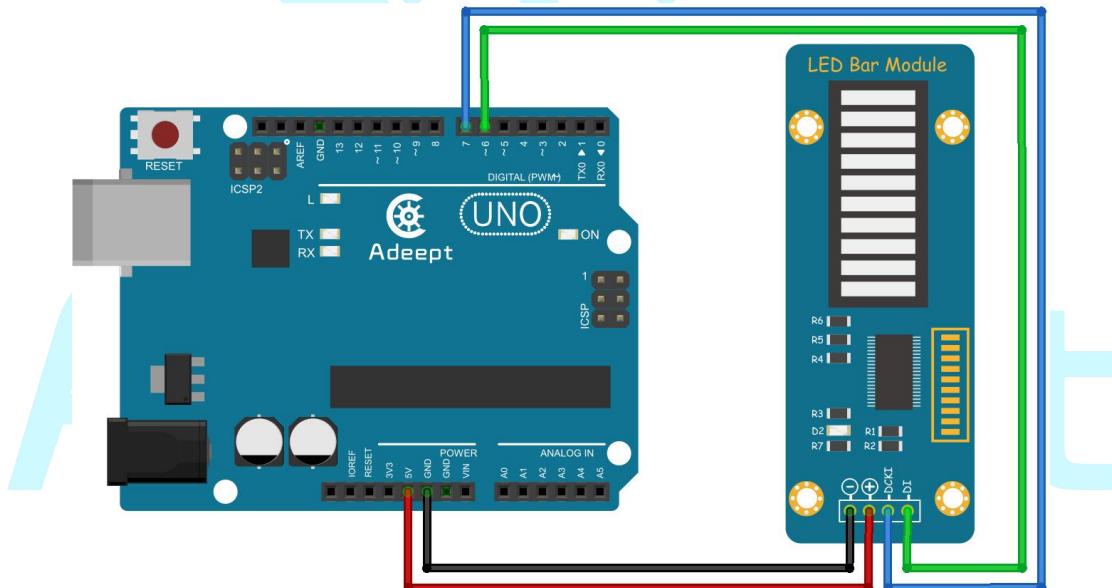
The schematic diagram:



The experiment is to control the number of LEDs brightened on the LED bar graph by programming the Arduino.

## Experimental Procedures

## Step 1: Build the circuit

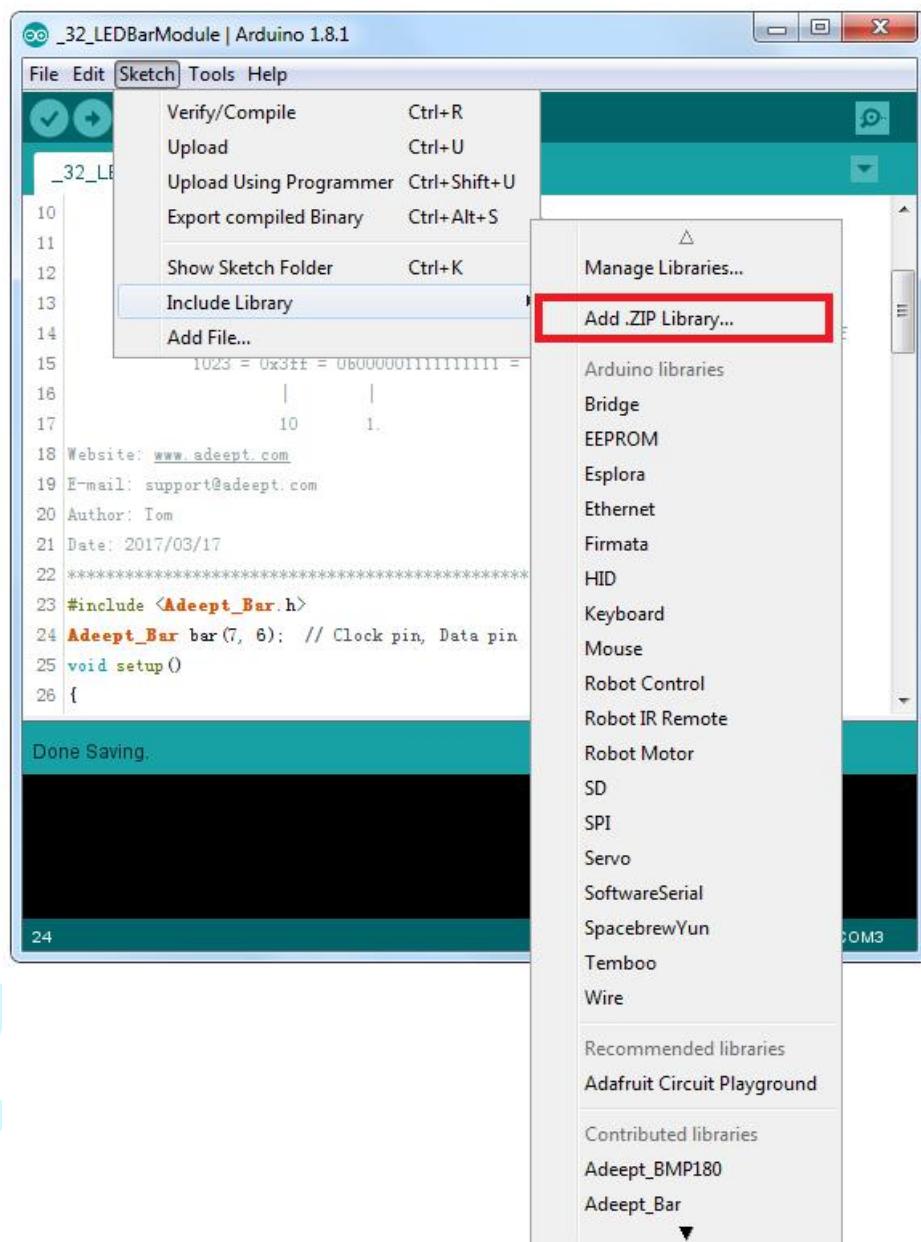


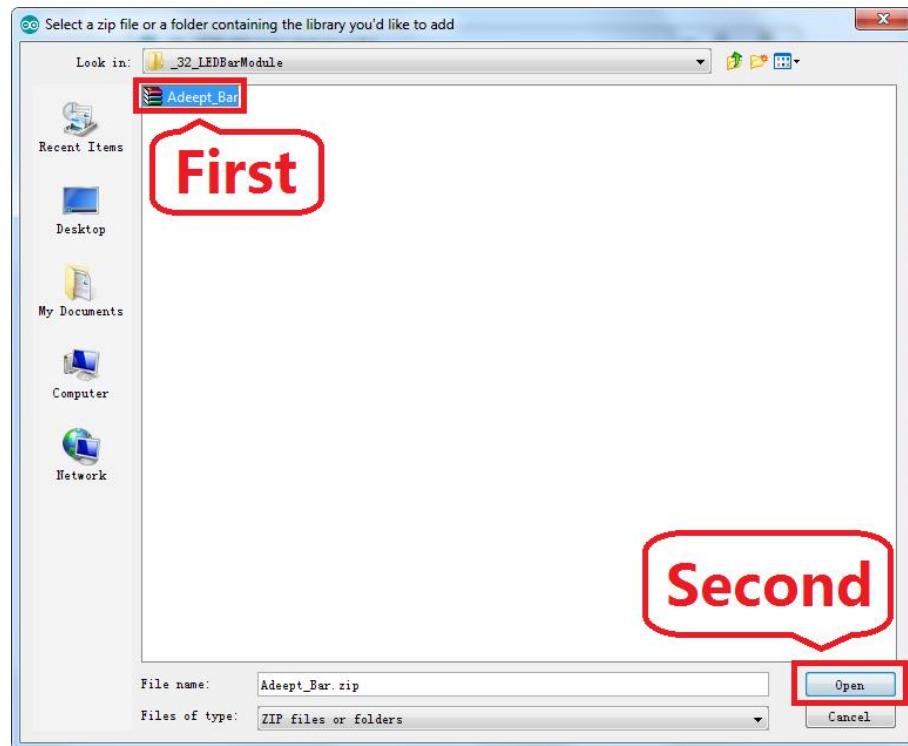
Adept UNO R3 Board	LED Bar Module
D7	DCLK
D6	DI
5V	+
GND	-

## **Step 2: Install the function library (Adeept\_Bar.zip).**



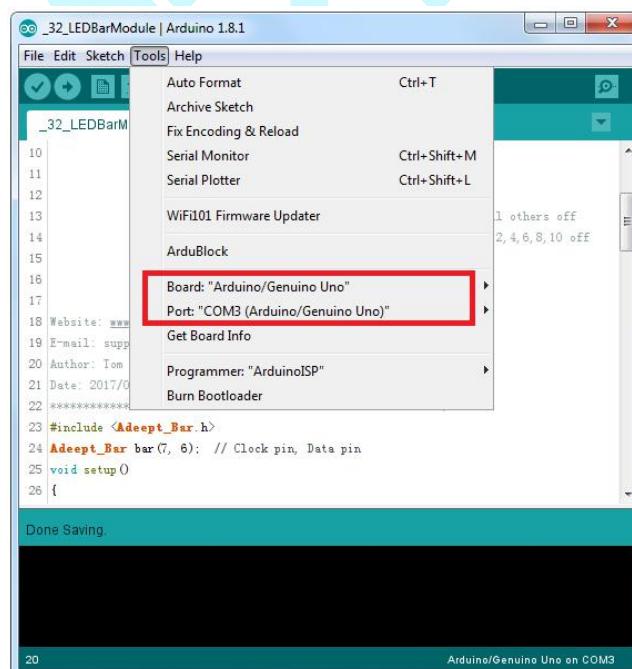
Adept\_Bar

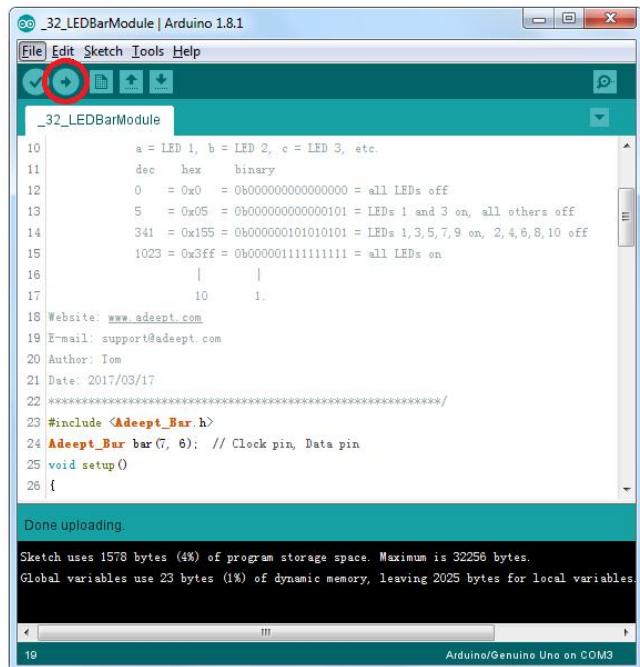




**Step 3:** Program `_32_LEDBarModule.ino`

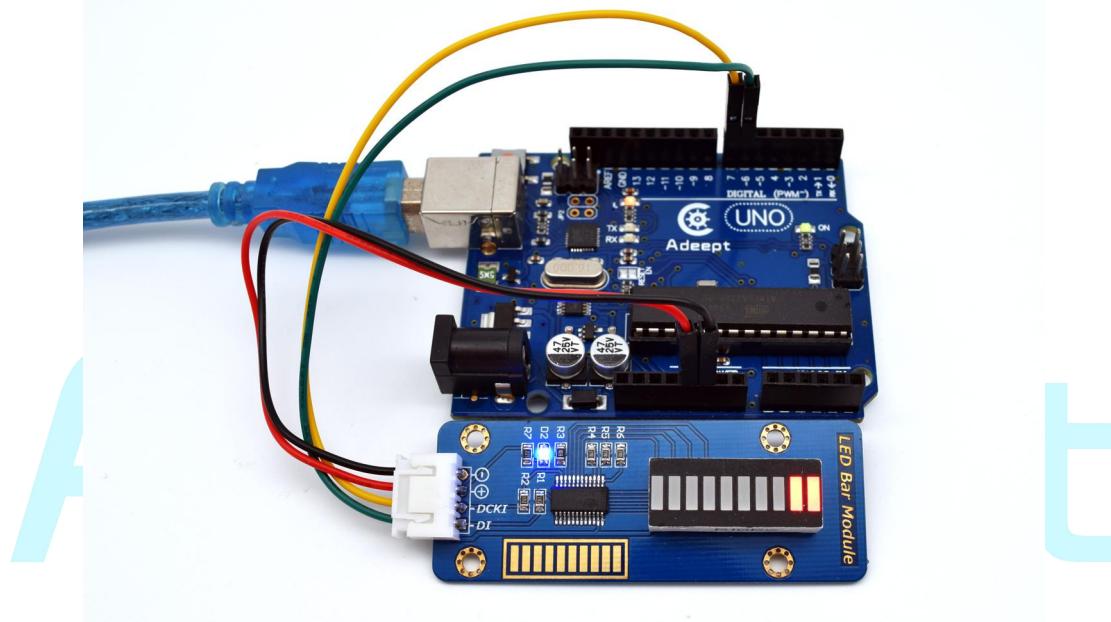
**Step 4:** Compile and download the sketch to the UNO R3 board.





The screenshot shows the Arduino IDE interface with the title bar '\_32\_LEDBarModule | Arduino 1.8.1'. A red circle highlights the 'Upload' button (a blue arrow pointing right) in the toolbar. The code editor contains the '\_32\_LEDBarModule' sketch, which includes comments about LED addresses and a copyright notice for Adeept. Below the code, a message says 'Done uploading.' and provides memory usage details: 'Sketch uses 1578 bytes (4%) of program storage space. Maximum is 32256 bytes. Global variables use 23 bytes (1%) of dynamic memory, leaving 2025 bytes for local variables.' The status bar at the bottom right indicates 'Arduino/Genuino Uno on COM3'.

Now you can see the LEDs on the LED Bar Graph module light up and dim one by one repeatedly.



# Lesson 33 Making A Simple Remote Control Device

## Introduction

The 1838B IR Receiver is a 38KHz IR receiver that can receive signals modulated by a standard 38KHz remote control. By programming the Arduino UNO R3 board, we can decode the signals.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* IR Receiver Module
- 1 \* Remote Controller Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

## Experimental Principle

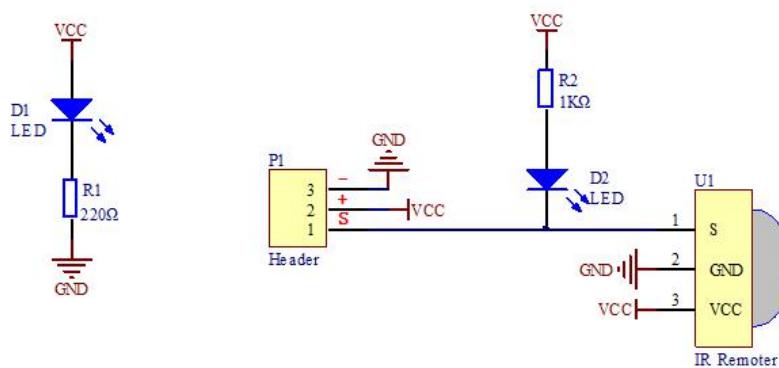
The Fritzing image:



Pin definition:

S	Digital output
+	VCC
-	GND

The schematic diagram:



In this experiment, by programming the Arduino board, we encode the data received by the IR Receiver that is sent by the remote control, and display the deciphered data on

Serial Monitor via the serial port. In the program, we use the Arduino-IRremote-master library (provided).

**Note:**

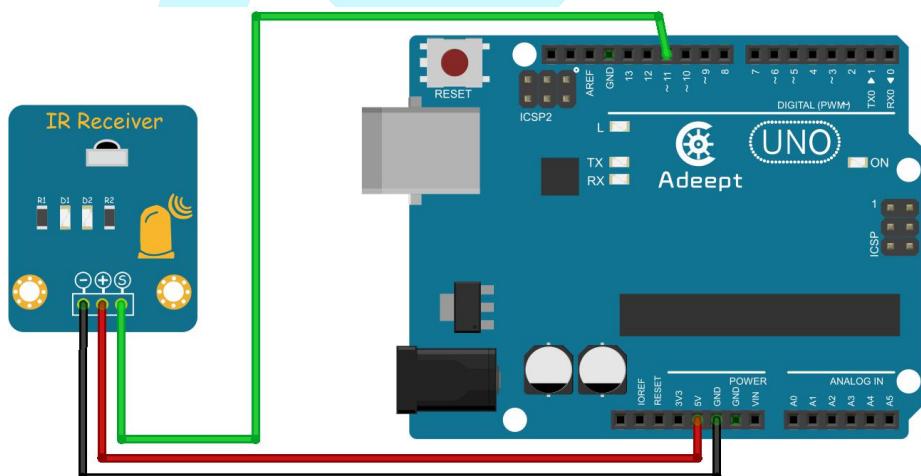
Before using this library, you have to delete the **RobotIRremote** directory in your Arduino IDE directory (check in IDE by File->Preference, and see the path in the Browse dialog box), and delete the **RobotIRremote** directory in the system Documents folder. For example, if your computer is running on Windows 7, you need to delete the **RobotIRremote** Directory in

C:\ProgramFiles(x86)\Arduino\libraries and C:\Users\SGJ\Documents\Arduino\libraries.

Otherwise, when you compile the program, errors will be prompted.

## Experimental Procedures

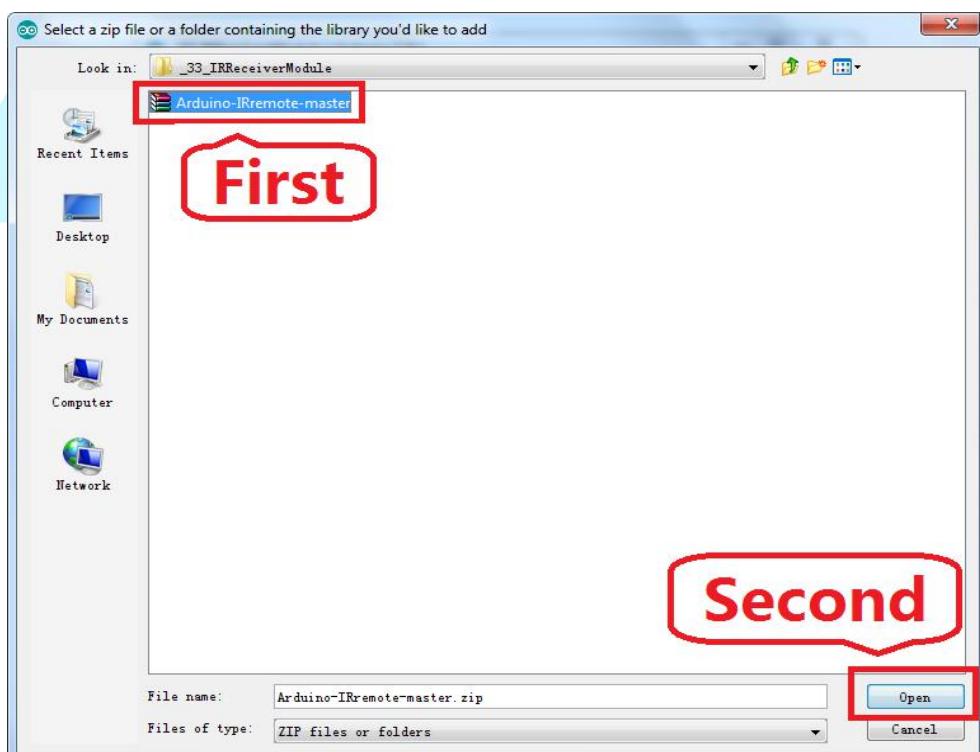
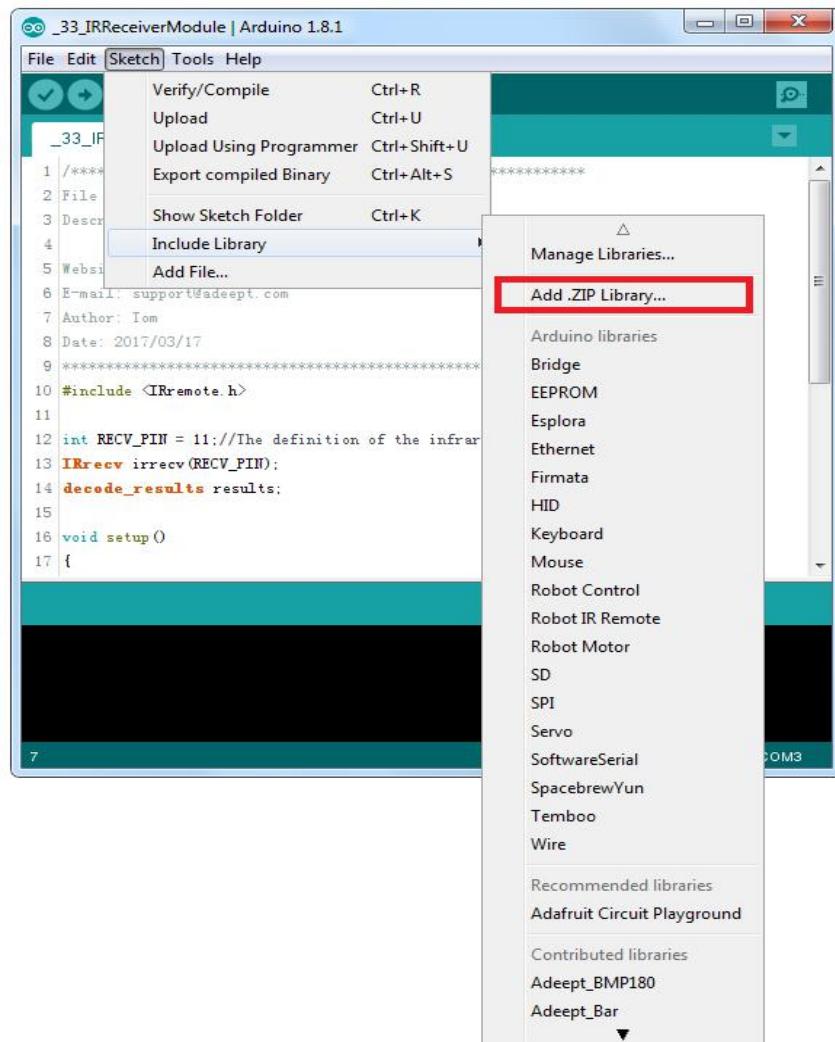
### Step 1: Build the circuit



Adeebt UNO R3 Board	IR Receiver Module
D11	S
5V	+
GND	-

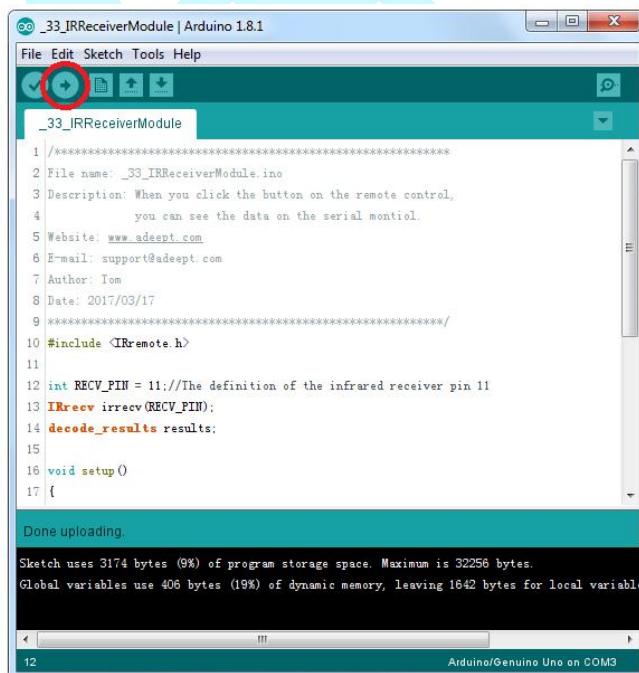
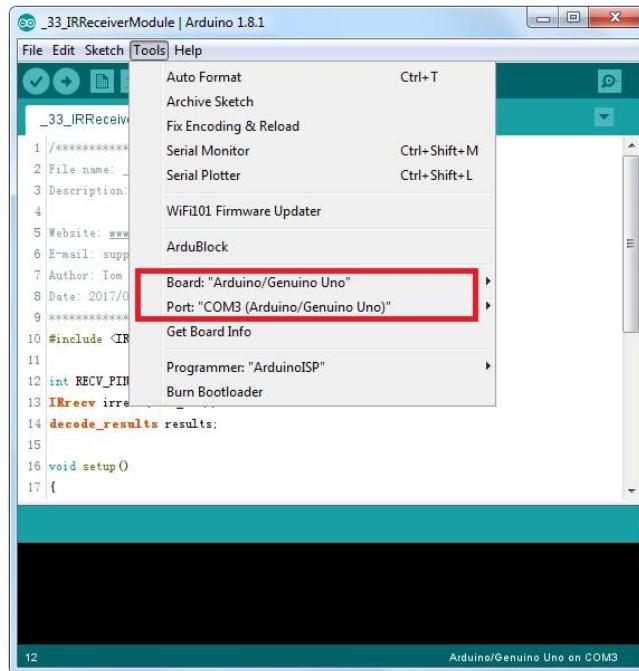
### Step 2: Install the function library (Arduino-IRremote-master.zip).



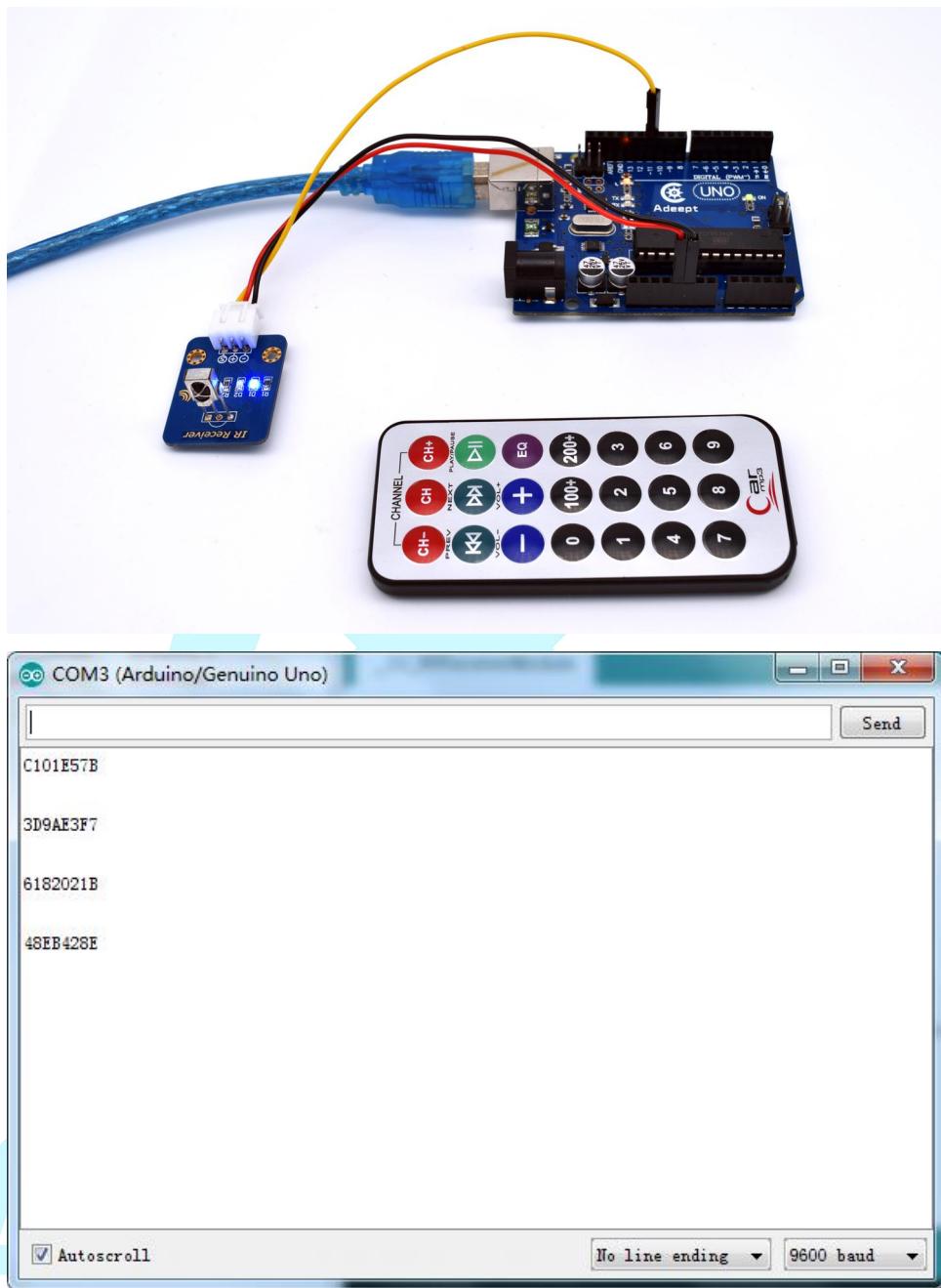


**Step 3:** Program \_33\_IRReceiverModule.ino

**Step 4:** Compile and download the sketch to the UNO R3 board.



Now press any key on the remote control, and you will see the corresponding code displayed on Serial Monitor.



# Lesson 34 Detection of The Soil Moisture System

## Introduction

The Soil Moisture Sensor module is a simple sensor that measures the soil moisture. When the soil moisture is insufficient, the output value of the sensor will decrease; on the other hand, the value will increase when there's enough water. The surface of the sensor is gilded to prolong its life.

The CM Module consists of a comparator LM393 and extremely simple external circuits. When using the module, you can set a threshold via the blue potentiometer beforehand. When the input analog value reaches the threshold, the digital pin S will output a Low level.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Soil Moisture Sensor Module
- 1 \* CM Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires
- 1 \* 2-Pin Female to Female Wires

## Experimental Principle

The Fritzing images:

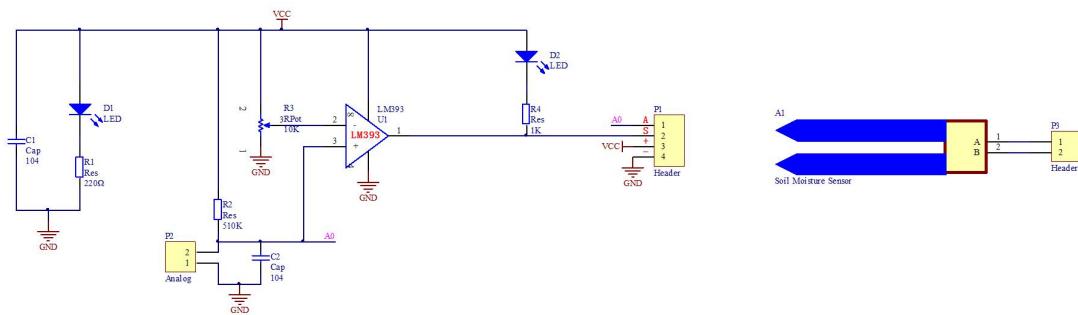




Pin definition:

Soil Moisture Sensor Module	
1	Analog output
2	Analog output
CM Module	
1	Analog output
2	Analog output
S	Digital output
A	Analog output
+	VCC
-	GND

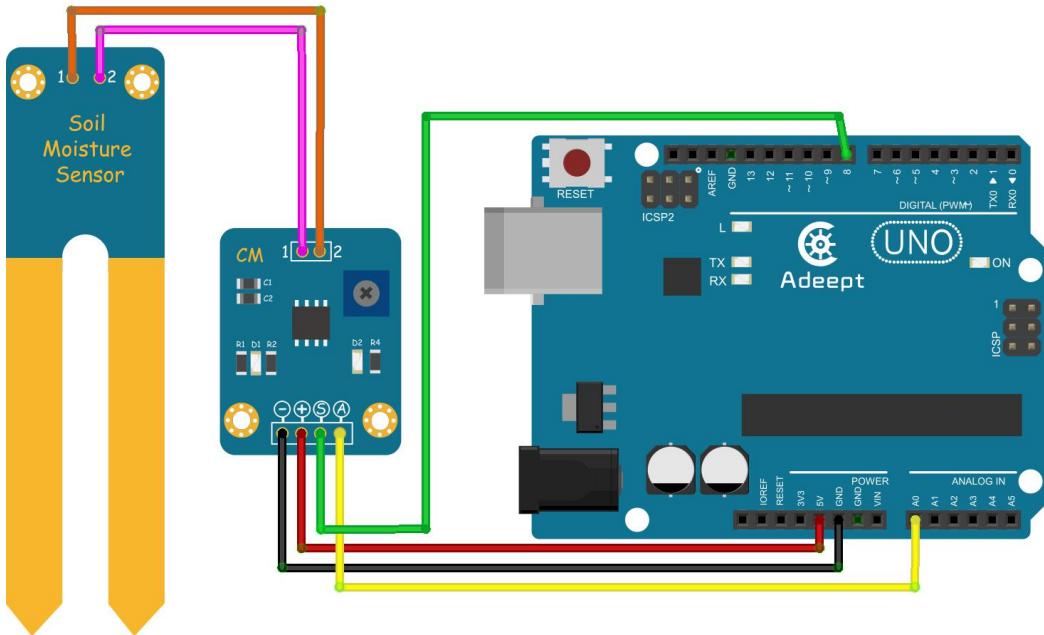
The schematic diagram:



The experiment uses the Soil Moisture Sensor module to collect data of soil moisture and display it on Serial Monitor.

## Experimental Procedures

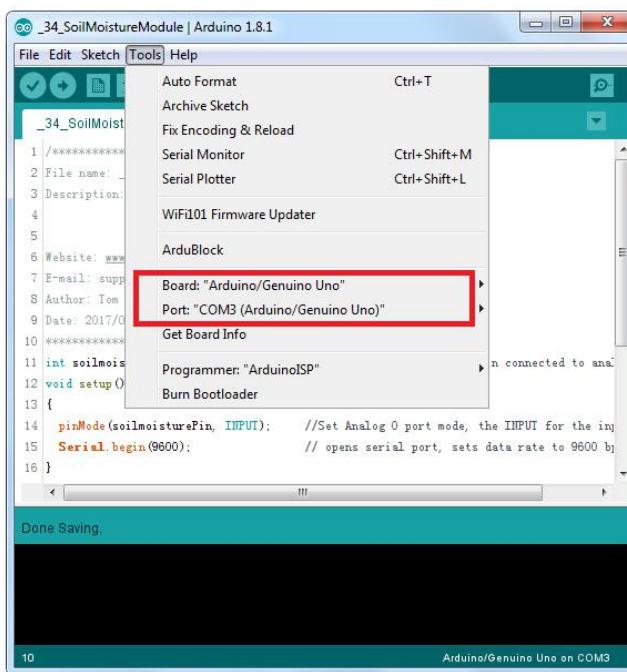
**Step 1:** Build the circuit

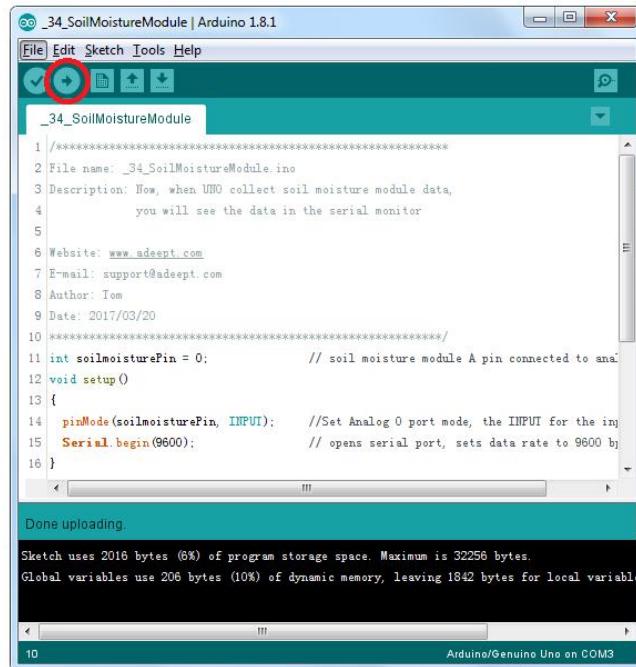


Adeept UNO R3 Board	CM Module	Soil Moisture Sensor Module
D8	S	
A0	A	
5V	+	
GND	-	
	1	2
	2	1

**Step 2:** Program \_34\_SoilMoistureModule.ino

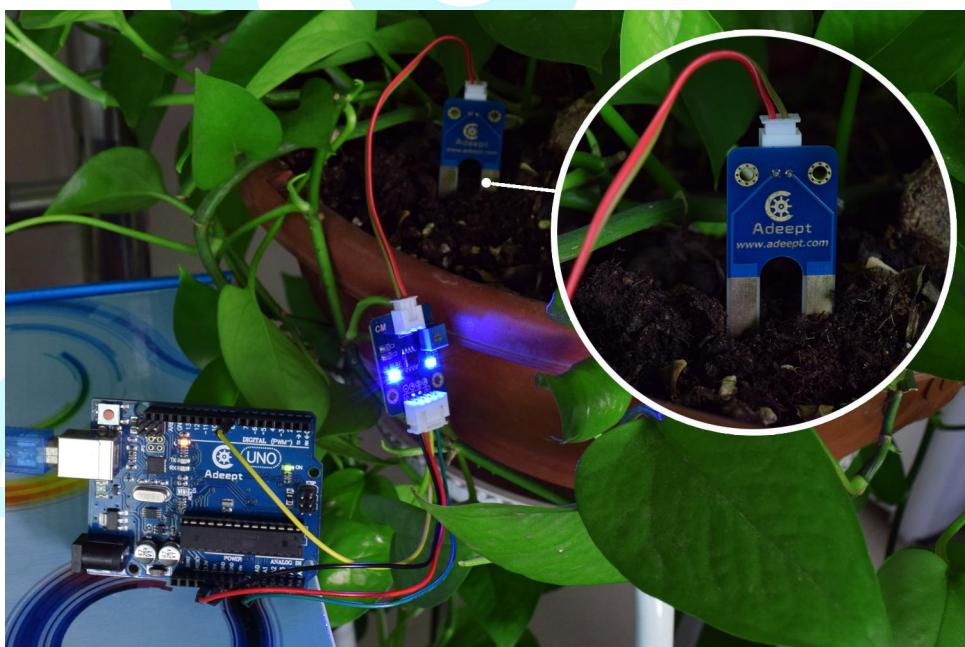
**Step 3:** Compile and download the sketch to the UNO R3 board.





The screenshot shows the Arduino IDE interface with the title bar '\_34\_SoilMoistureModule | Arduino 1.8.1'. The menu bar includes File, Edit, Sketch, Tools, Help. A red circle highlights the 'Upload' button (a blue arrow pointing right) in the toolbar. The code editor contains the '\_34\_SoilMoistureModule' sketch. Below the code, a message says 'Done uploading.' and provides memory usage details: 'Sketch uses 2016 bytes (6%) of program storage space. Maximum is 32256 bytes. Global variables use 206 bytes (10%) of dynamic memory, leaving 1842 bytes for local variables.' The status bar at the bottom right shows 'Arduino/Genuine Uno on COM3'.

Open Serial Monitor of the Arduino IDE. You will see the value of soil moisture collected by the module displayed on the window.





```
COM3 (Arduino/Genuino Uno)
Soil moisture data: 671
Soil moisture data: 676
Soil moisture data: 681
Soil moisture data: 688
Soil moisture data: 691
Soil moisture data: 694
Soil moisture data: 694
Soil moisture data: 696
Soil moisture data: 692
Soil moisture data: 686
Soil moisture data: 683
Soil moisture data: 678
Soil moisture data: 673
Soil moisture data: 680
Soil moisture data: 680
```

Autoscroll      No line ending      9600 baud

# Adeept

# Lesson 35 Detection of The Water Height System

## Introduction

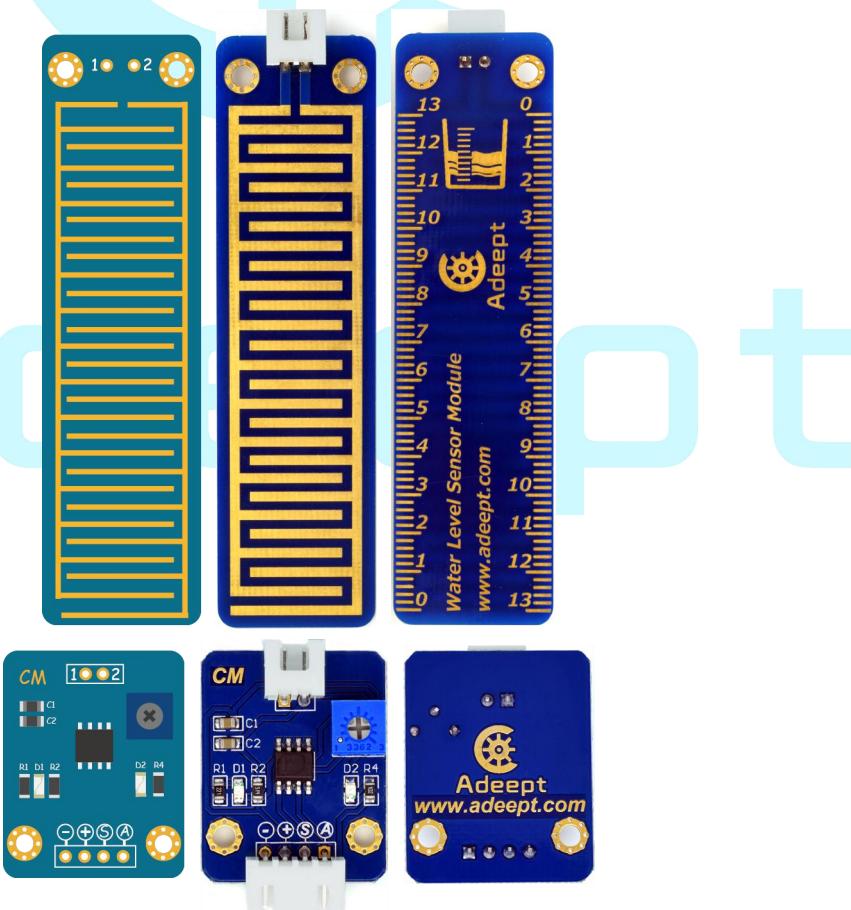
The module is a simple water level sensor. It measures the water volume by the printed wires exposed to the air on the module. The more water on the surface, more wires connected. Thus, the area of electrified wires gets larger, so the output voltage will increase. The surface of the sensor is gilded to prolong its life.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Water Level Sensor Module
- 1 \* CM Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires
- 1 \* 2-Pin Female to Female Wires

## Experimental Principle

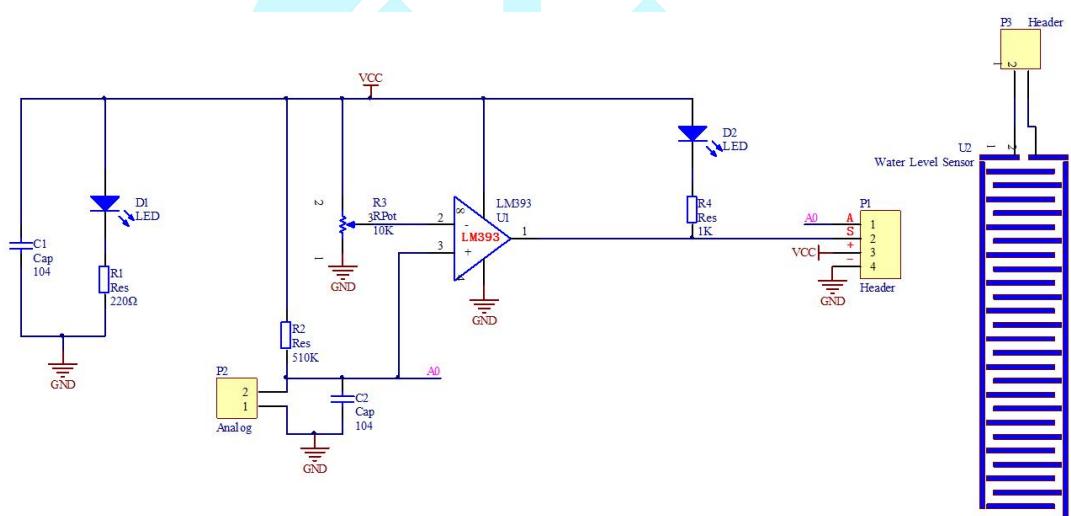
The Fritzing images:



Pin definition:

Water Level Sensor Module	
1	Analog output
2	Analog output
CM Module	
1	Analog output
2	Analog output
S	Digital output
A	Analog output
+	VCC
-	GND

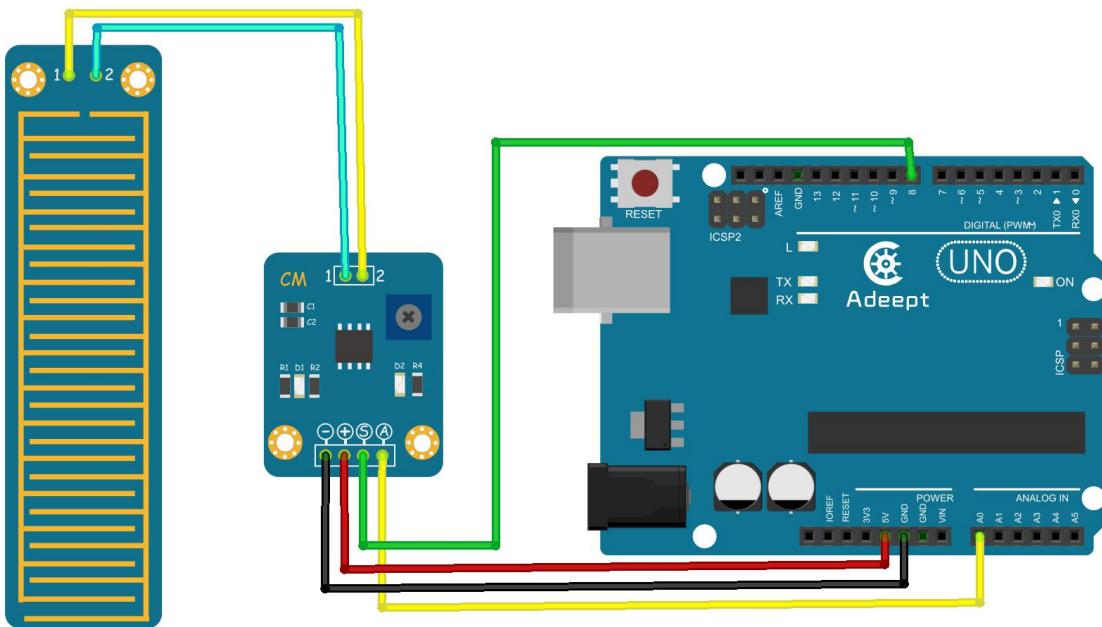
The schematic diagram:



This experiment collects the data of water level by the Water Level Sensor module and displays it on Serial Monitor.

## Experimental Procedures

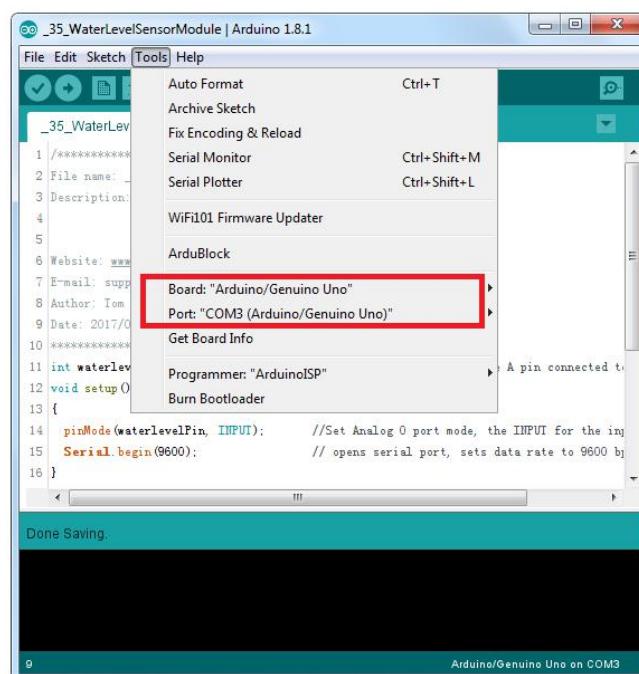
**Step 1:** Build the circuit

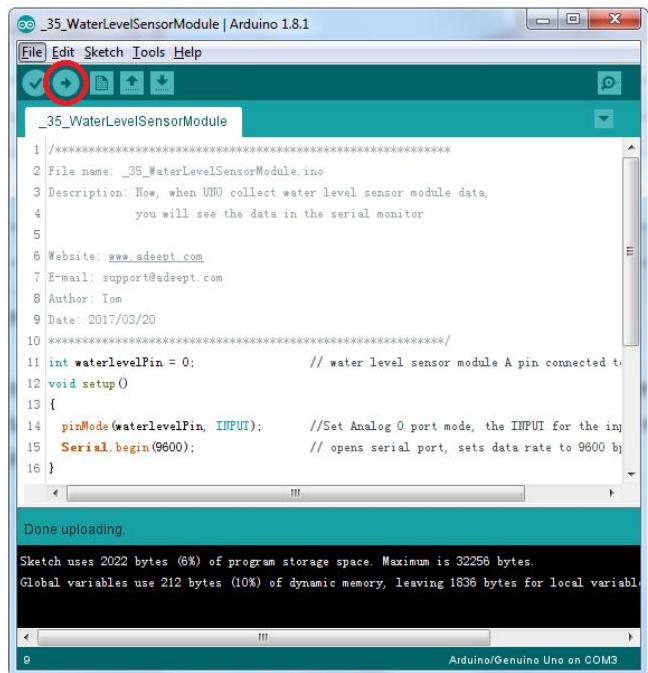


Adeept UNO R3 Board	CM Module	Water Level Sensor Module
D8	S	
A0	A	
5V	+	
GND	-	
	1	2
	2	1

**Step 2:** Program `_35_WaterLevelSensorModule.ino`

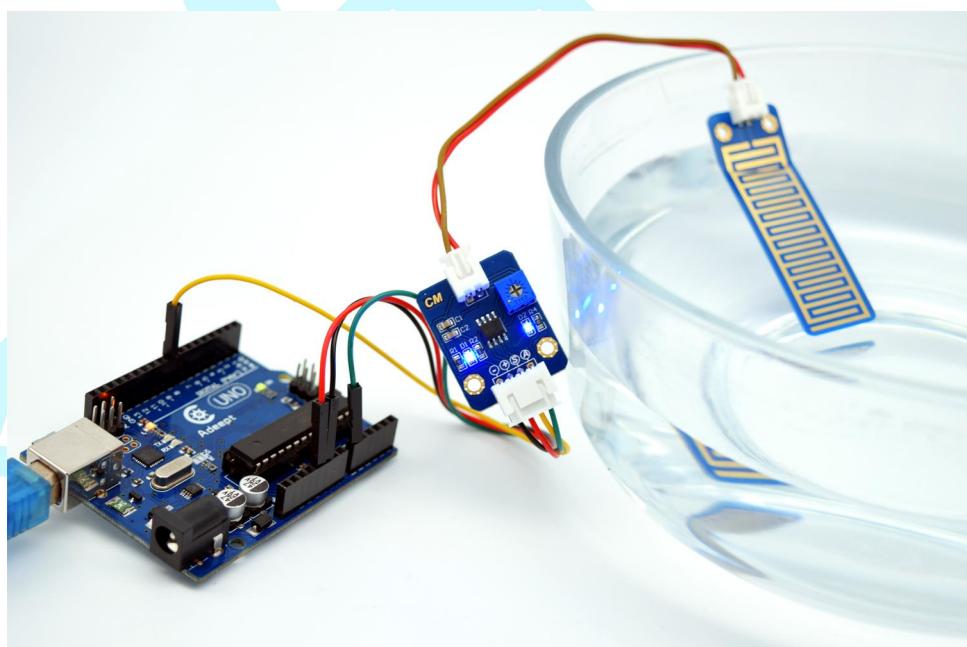
**Step 3:** Compile and download the sketch to the UNO R3 board.





The screenshot shows the Arduino IDE interface with the title bar "35\_WaterLevelSensorModule | Arduino 1.8.1". The menu bar includes File, Edit, Sketch, Tools, and Help. A red circle highlights the "Upload" button (a blue arrow pointing right) in the toolbar. The code editor contains a sketch named "35\_WaterLevelSensorModule.ino" which reads data from an analog pin connected to a water level sensor module. The serial monitor at the bottom shows the message "Done uploading." and provides memory usage details: "Sketch uses 2022 bytes (6%) of program storage space. Maximum is 32256 bytes. Global variables use 212 bytes (10%) of dynamic memory, leaving 1836 bytes for local variables." The status bar at the bottom right indicates "Arduino/Genuino Uno on COM3".

Open Serial Monitor of the Arduino IDE. Take a bottle with some water. Place the Water Level Sensor module in the water at end and add water into the bottle. You will see the value of water level displayed on the window and changes as you add water.





```
COM3 (Arduino/Genuino Uno)
Send
Water level sensor data: 123
Water level sensor data: 124
Water level sensor data: 129
Water level sensor data: 154
Water level sensor data: 189
Water level sensor data: 195
Water level sensor data: 151
Water level sensor data: 151
Water level sensor data: 135
Water level sensor data: 122
Water level sensor data: 120
Water level sensor data: 123
Water level sensor data: 124
Water level sensor data: 123
Water level sensor data: 135

Autoscroll No line ending 9600 baud
```

# Adeept

# Lesson 36 Detection of The Distance System

## Introduction

Ultrasonic Distance Sensor module supports a contactless detection within a distance of 2cm-400cm. It contains an ultrasonic emitter, receiver and control circuits.

### Notes:

1. The module is not suggested to connect wires when power is on. If you have to do so, please first connect the GND and then other pins; otherwise, the module may not work.
2. During the ranging, the area of the targeted object should be no less than 0.5mm and the surface facing the module should be as flat as possible; otherwise the result may be inaccurate.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Ultrasonic Distance Sensor Module
- 1 \* USB Cable
- 4 \* Male to Female Wires

## Experimental Principle

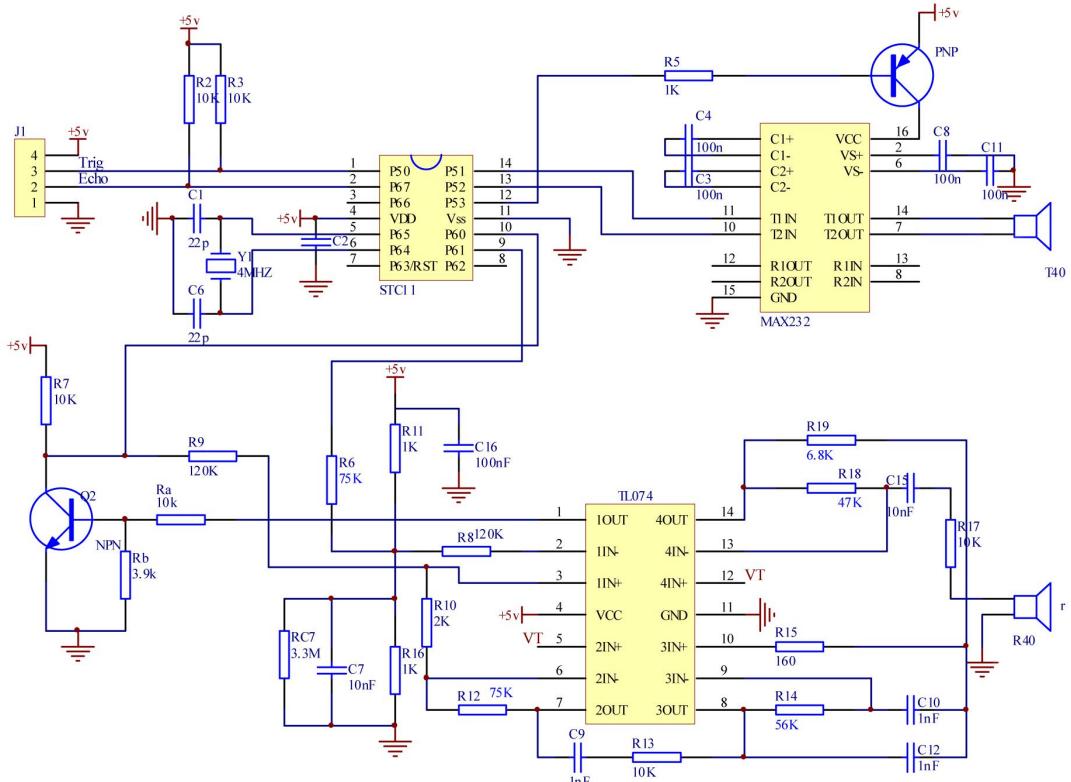
The Fritzing image:



Pin definition:

Trig	Digital output
Echo	Digital output
Vcc	VCC
Gnd	GND

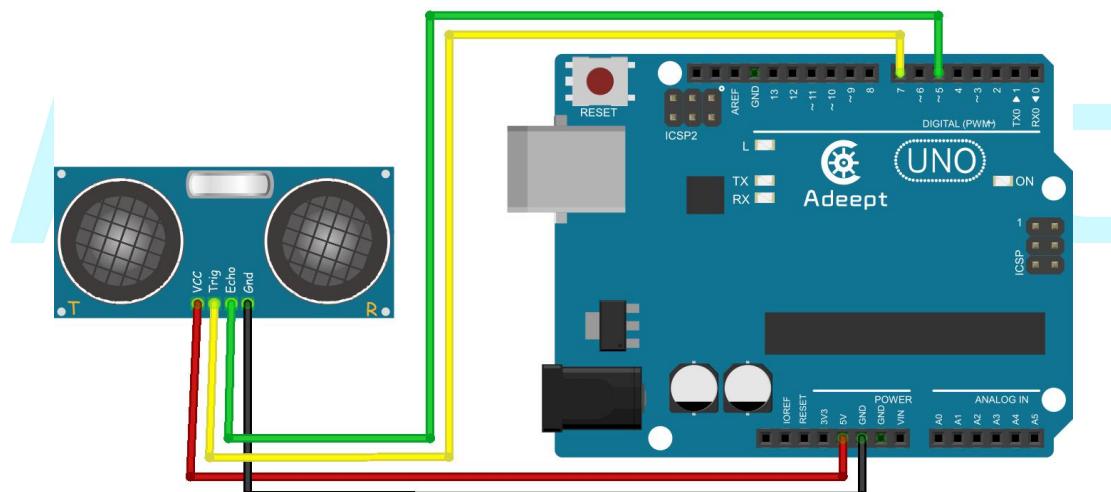
The schematic diagram:



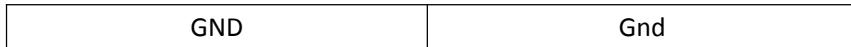
This experiment uses the Ultrasonic Distance Sensor module to detect the distance between the obstacle and module and show the data sensed on Serial Monitor via the serial port.

## Experimental Procedures

### Step 1: Build the circuit

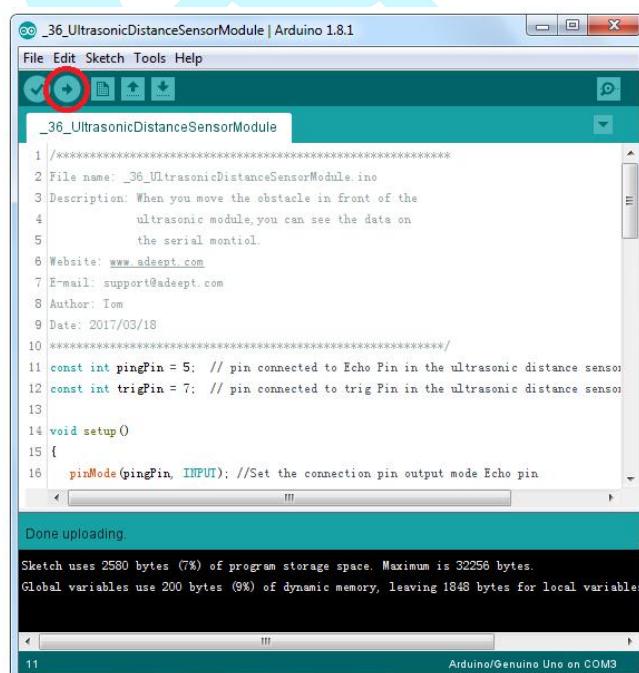
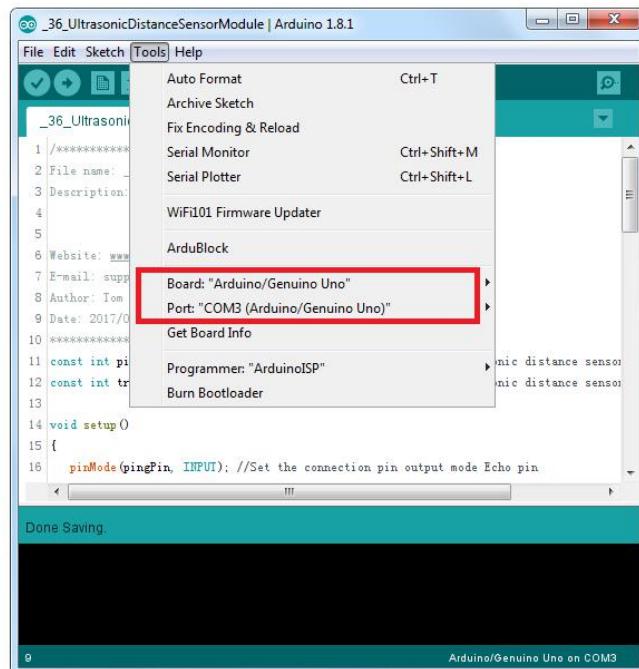


Adeept Uno R3 Board	Ultrasonic Distance Sensor Module
D7	Trig
D5	Echo
5V	Vcc

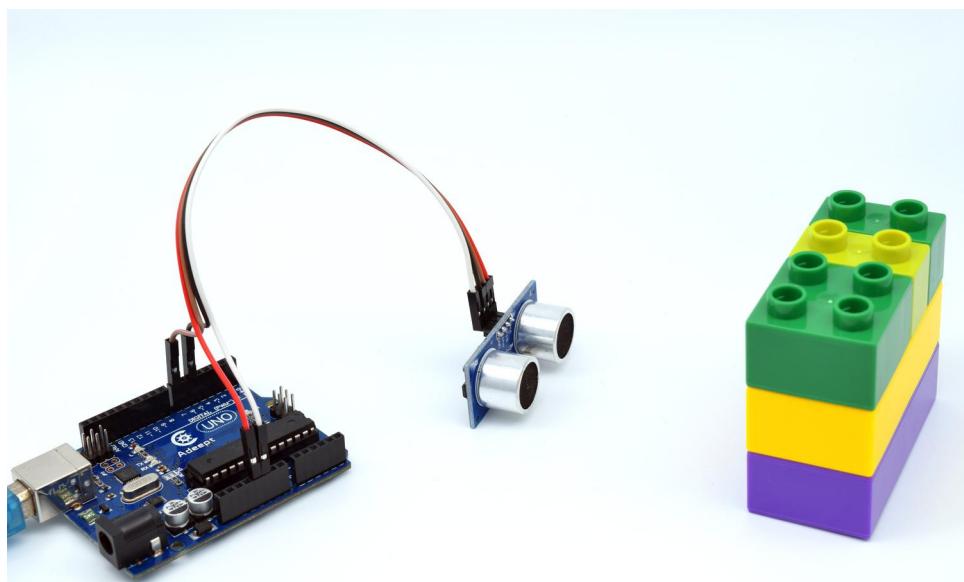


**Step 2:** Program \_36\_UltrasonicDistanceSensorModule.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



Open Serial Monitor of the Arduino IDE. You will see the distance to the obstacle at front of the Ultrasonic Distance Sensor module displayed on the window.



COM3 (Arduino/Genuino Uno)

```
distance: 5 cm
distance: 6 cm
distance: 5 cm
```

Autoscroll      No line ending ▾      9600 baud ▾

# Lesson 37 Control An LED by PC

## Introduction

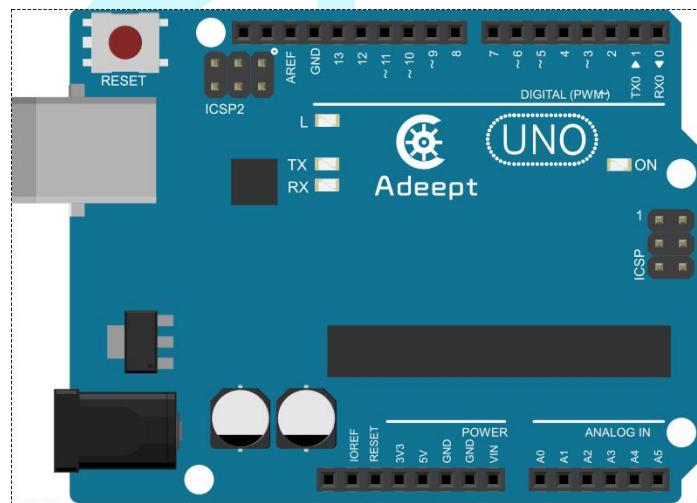
In this lesson, we will control an LED by PC. We send '0' or '1' to Arduino UNO via serial port, the state of the LED on the Arduino UNO board will be toggled.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* USB Cable

## Experimental Procedures

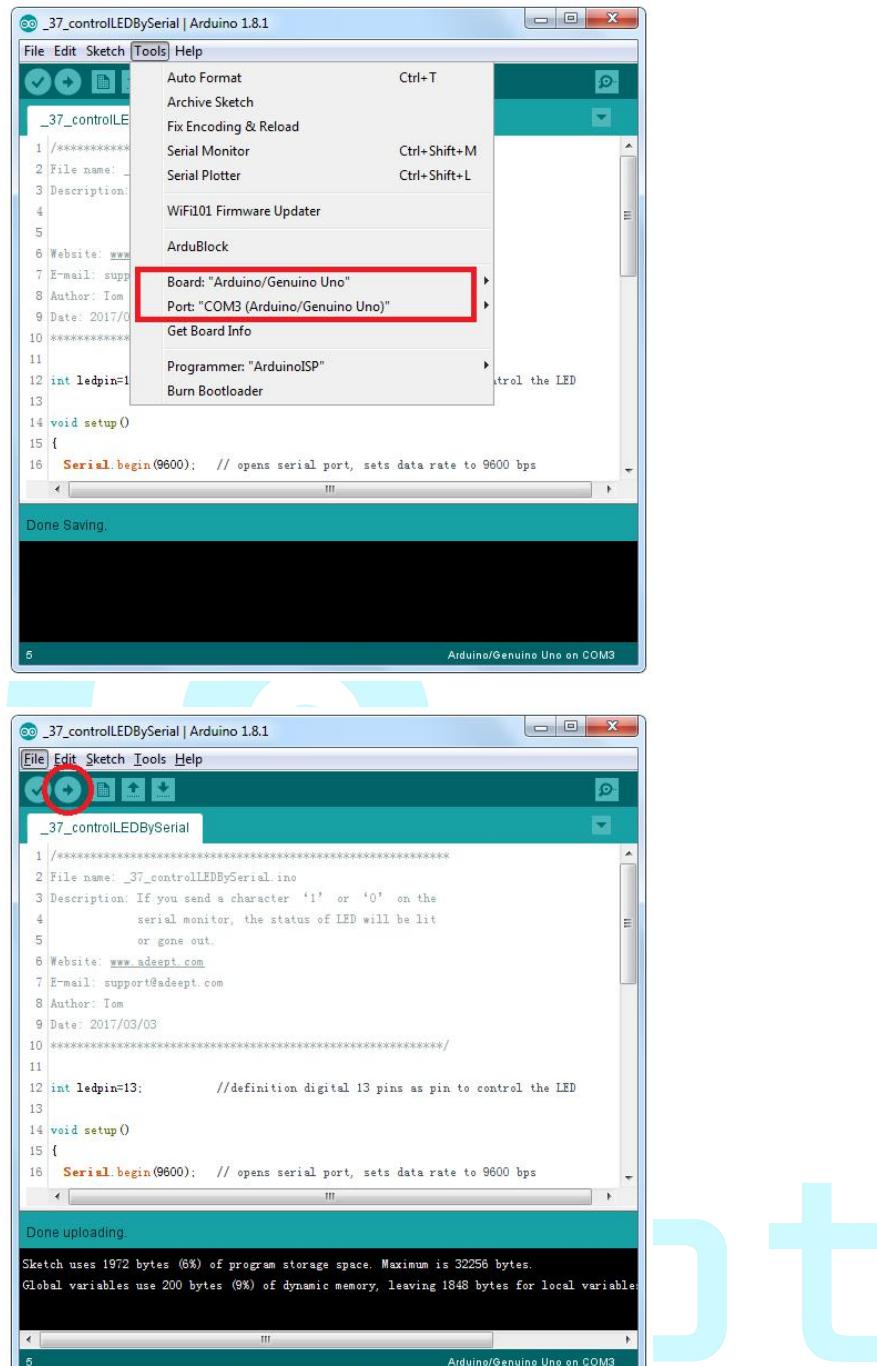
### Step 1: Build the circuit



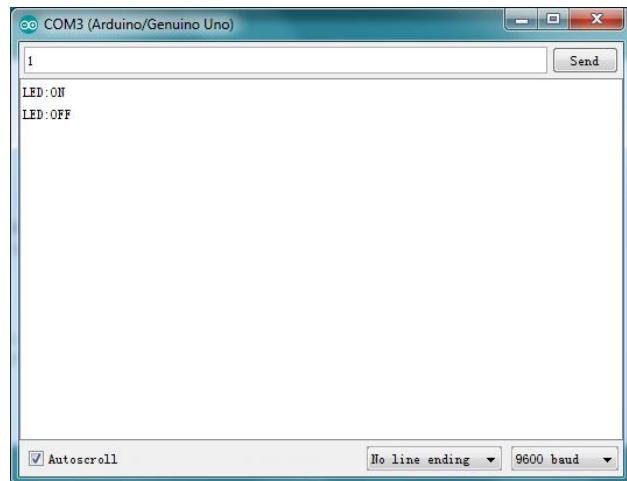
**Step 2:** Program \_37\_controlLEDBySerial.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.

A  
d  
e  
e  
p  
t



Open Serial Monitor of the Arduino IDE. Send 1 or 0, you can control the LED light or off.



# Adeept

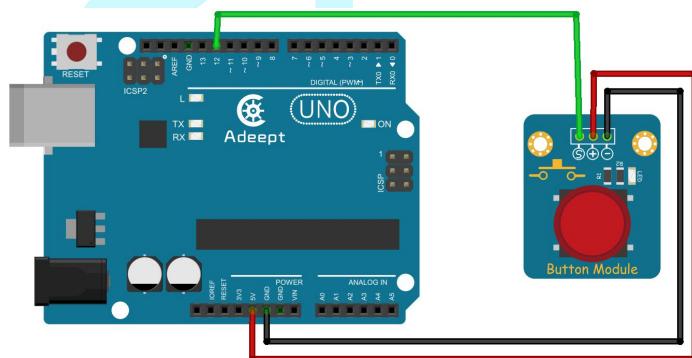
# Lesson 38 Upload The State of A Button to PC

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* USB Cable
- 1 \* Button Module
- 1 \* 3-Pin Wires

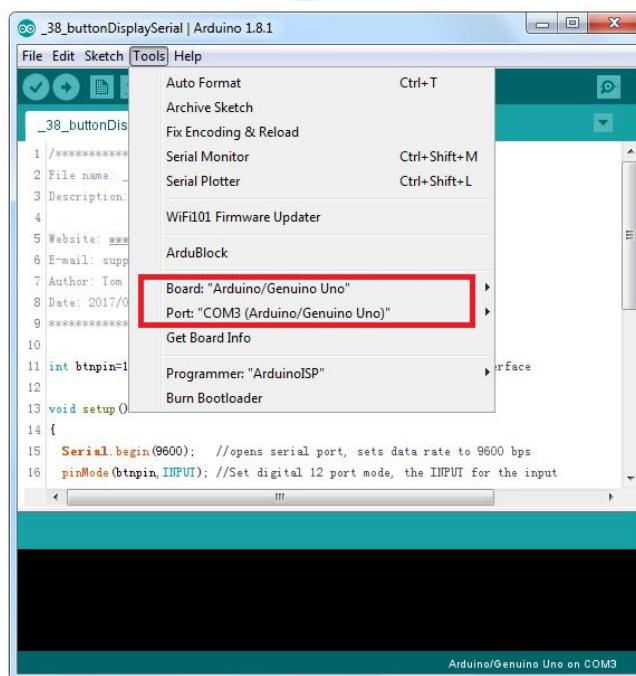
## Experimental Procedures

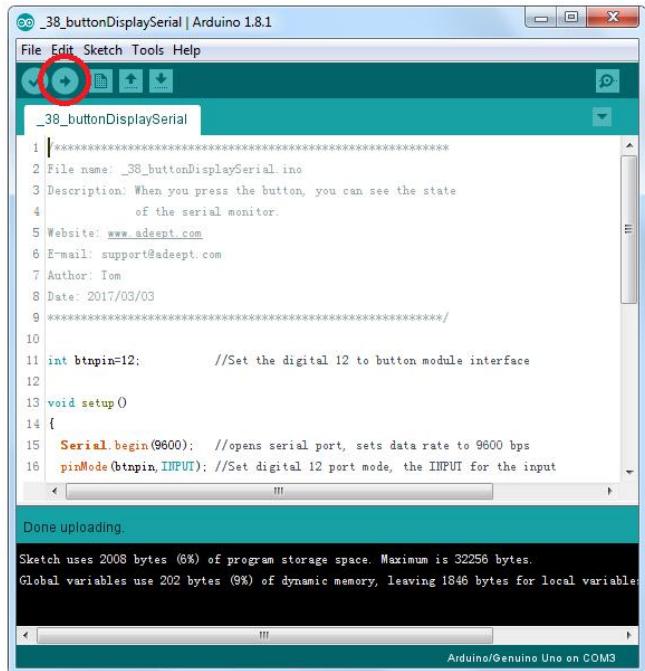
Step 1: Build the circuit



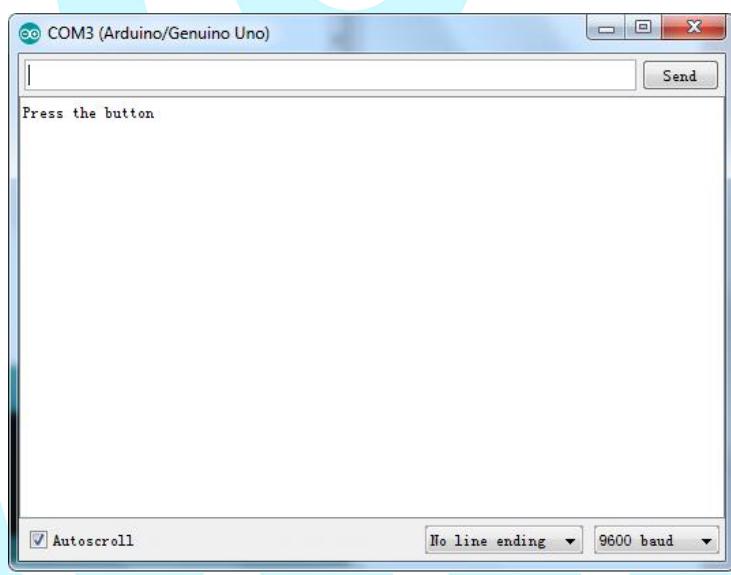
Step 2: Program \_38\_buttonDisplaySerial.ino

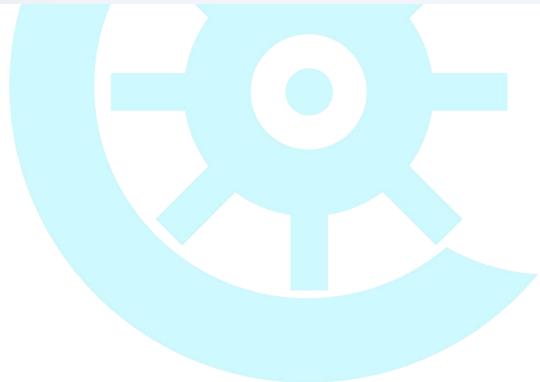
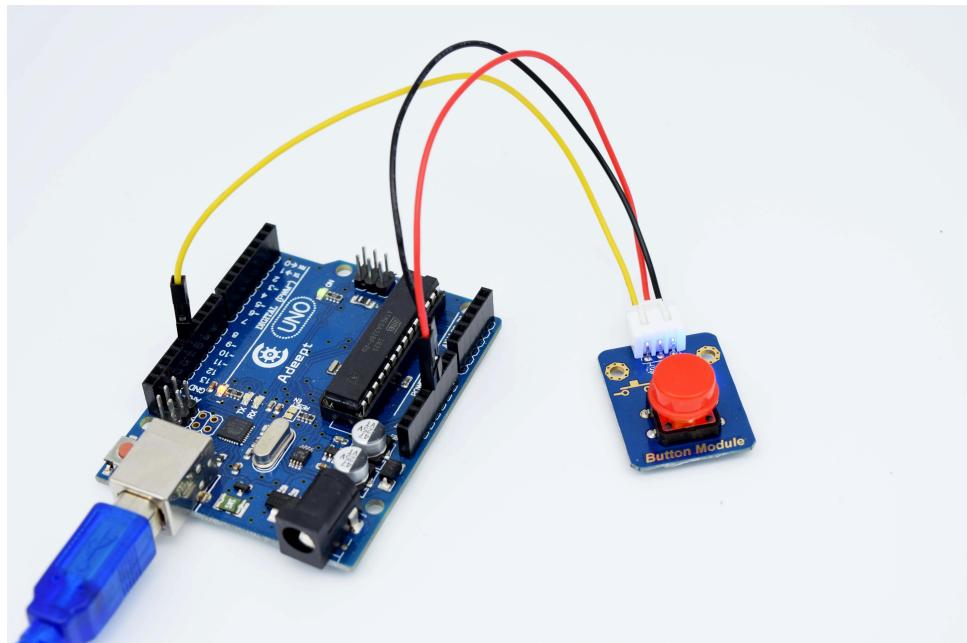
Step 3: Compile and download the sketch to the UNO R3 board.





Open the Serial Monitor in Arduino IDE and press the button module. Then the UNO R3 board will upload the data collected to and display in the Serial Monitor.





# Adeept

# Lesson 39 Simple Laser Pen

## Introduction

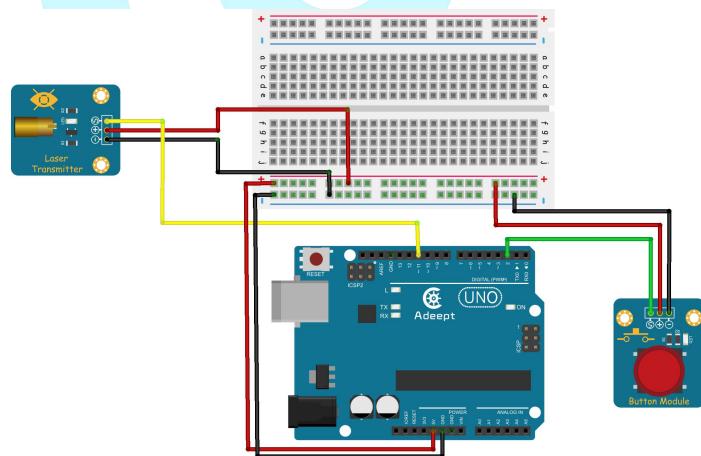
In this lesson, we will make a simple laser pen. The laser pen can be used for teaching or shooting.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Laser Transmitter Module
- 1 \* Button Module
- 2 \* 3-Pin Wires
- 1 \* USB Cable
- 1 \* Breadboard
- 2 \* Hookup Wire Set

## Experimental Procedures

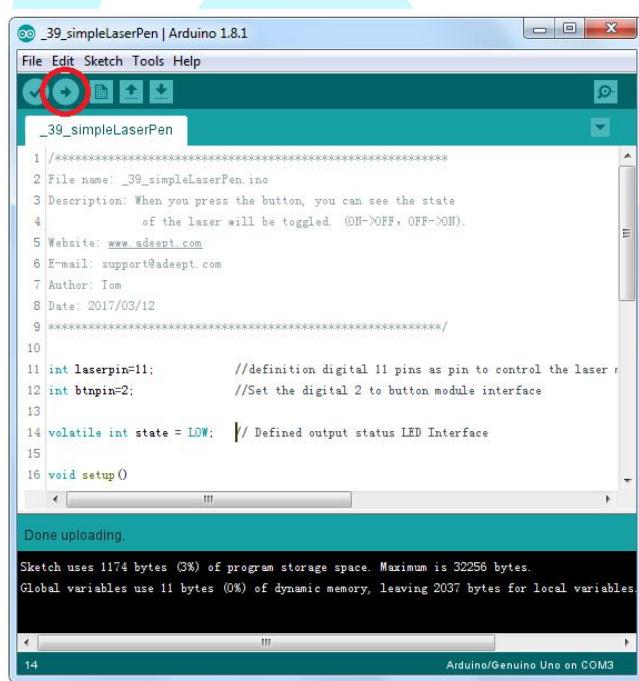
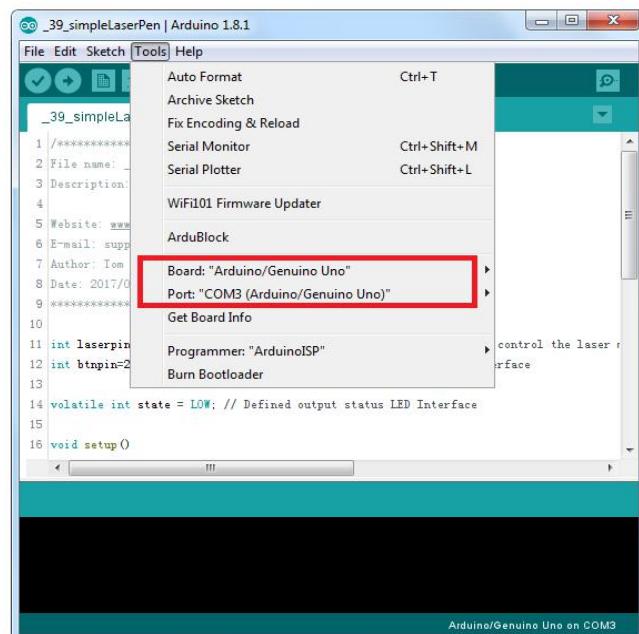
### Step 1: Build the circuit



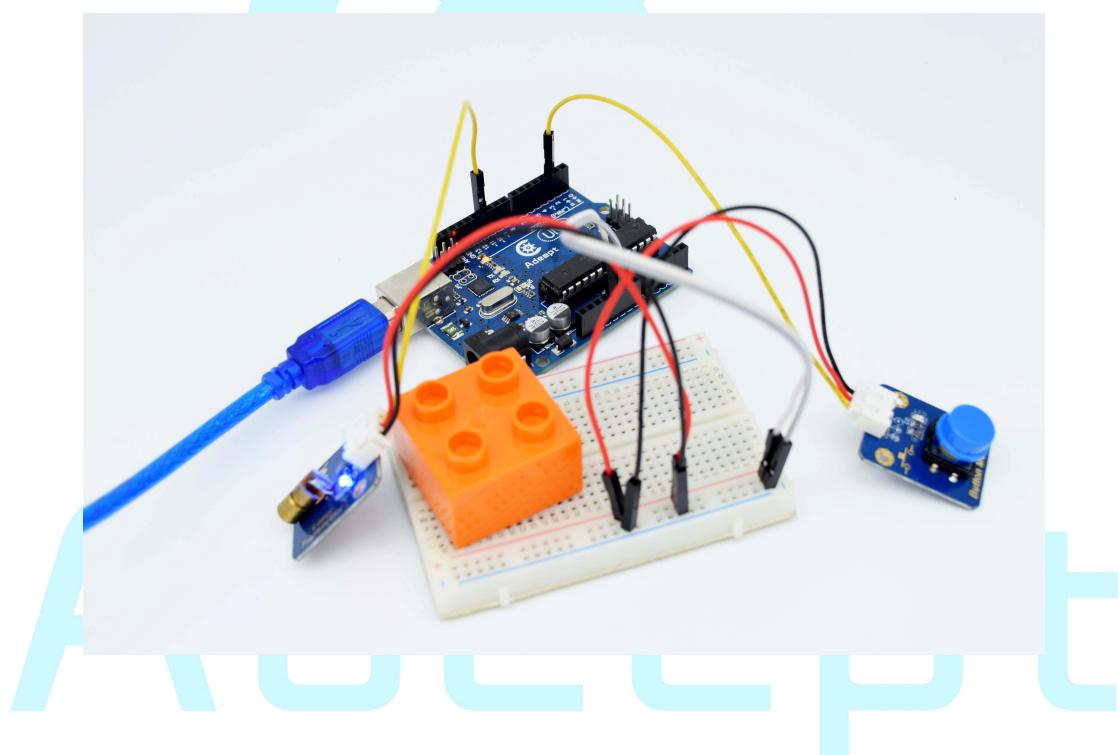
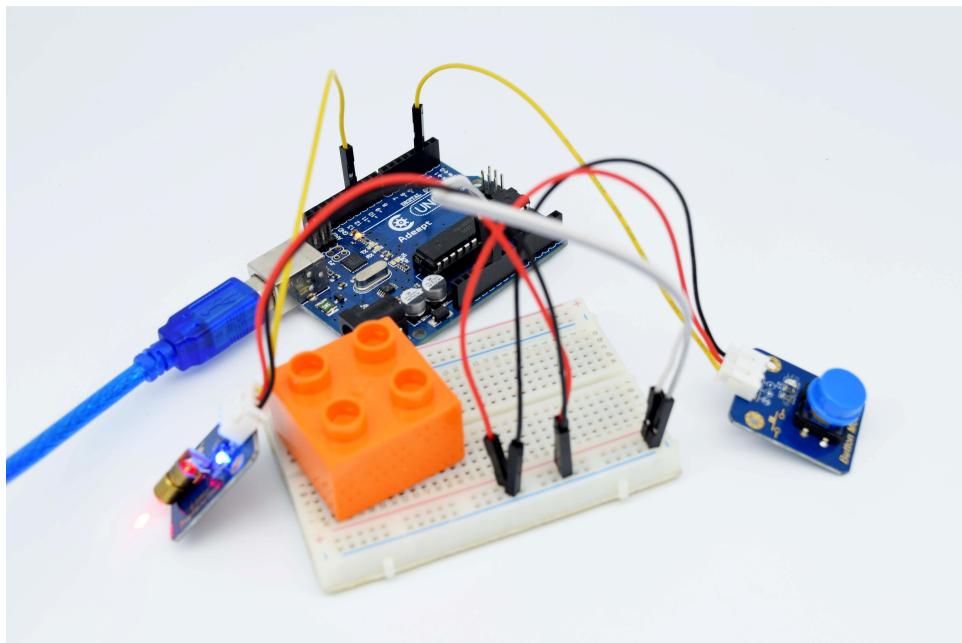
Note: DO NOT look directly into the laser!

Step 2: Program \_39\_simpleLaserPen.ino

Step 3: Compile and download the sketch to the UNO R3 board.



Use the button to toggle the laser module on and off.



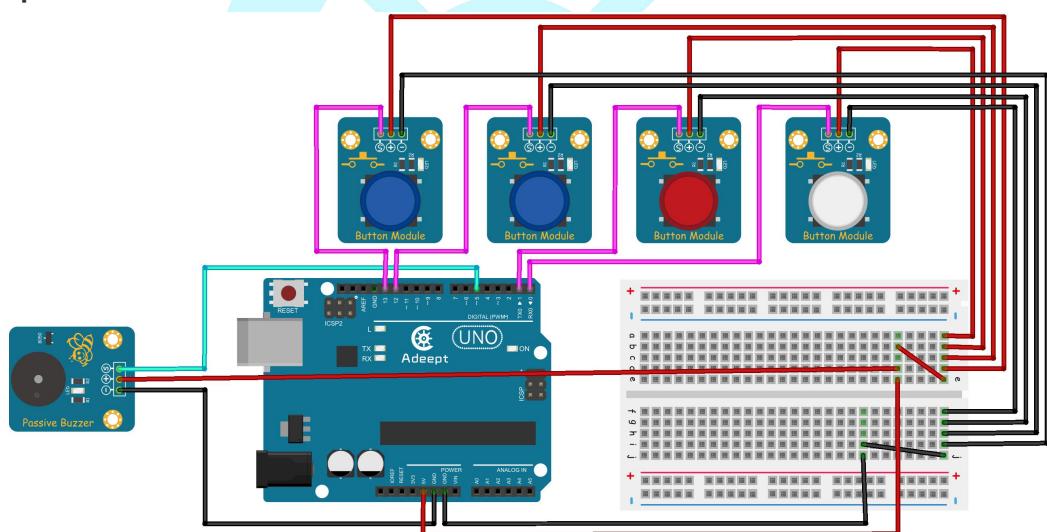
# Lesson 40 Control Buzzer by Button

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 4 \* Button Module
- 1 \* Passive Buzzer Module
- 1 \* USB Cable
- 5 \* 3-Pin Wires
- 4 \* Hookup Wire Set
- 1 \* Breadboard

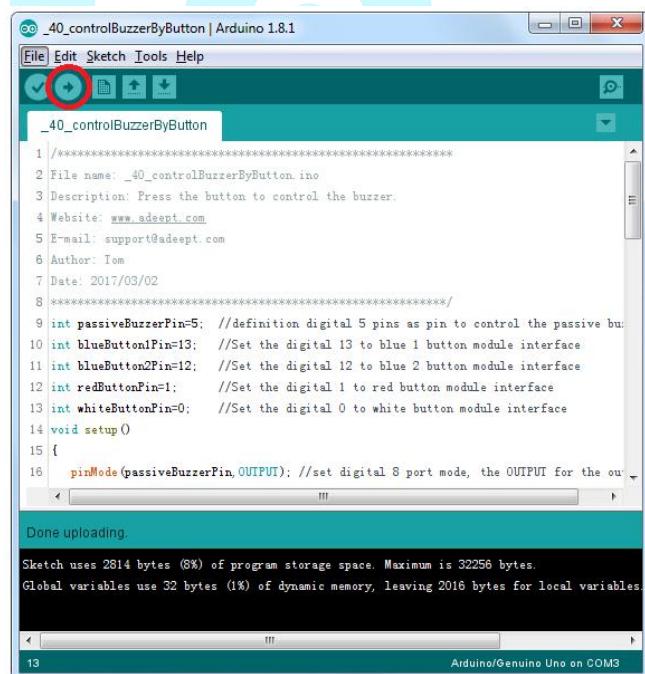
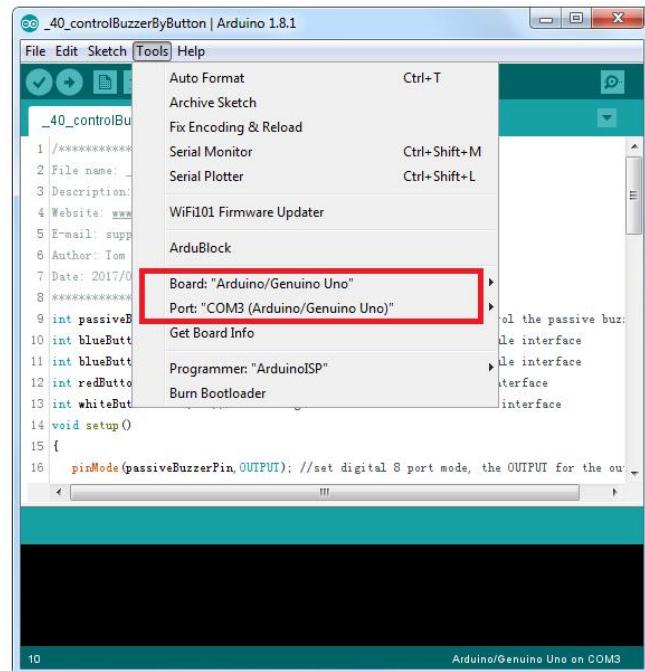
## Experimental Procedures

Step 1: Build the circuit

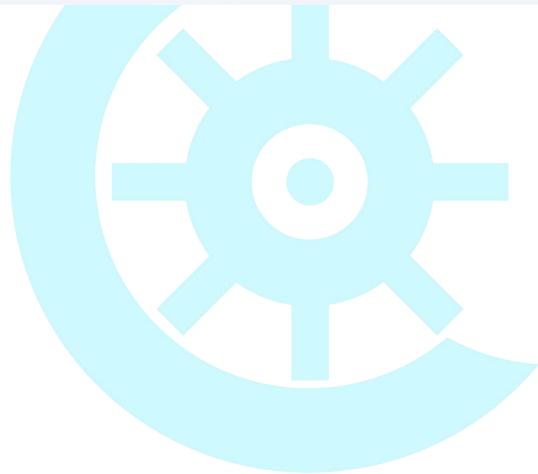
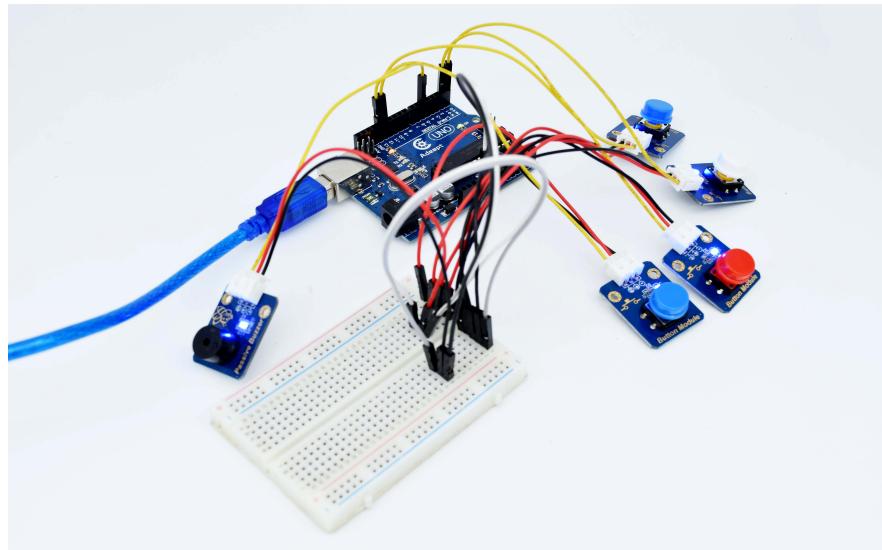


Step 2: Program \_40\_controlBuzzerByButton.ino

Step 3: Compile and download the sketch to the UNO R3 board.



When you press different buttons, the buzzer will make different sounds



# Adeept

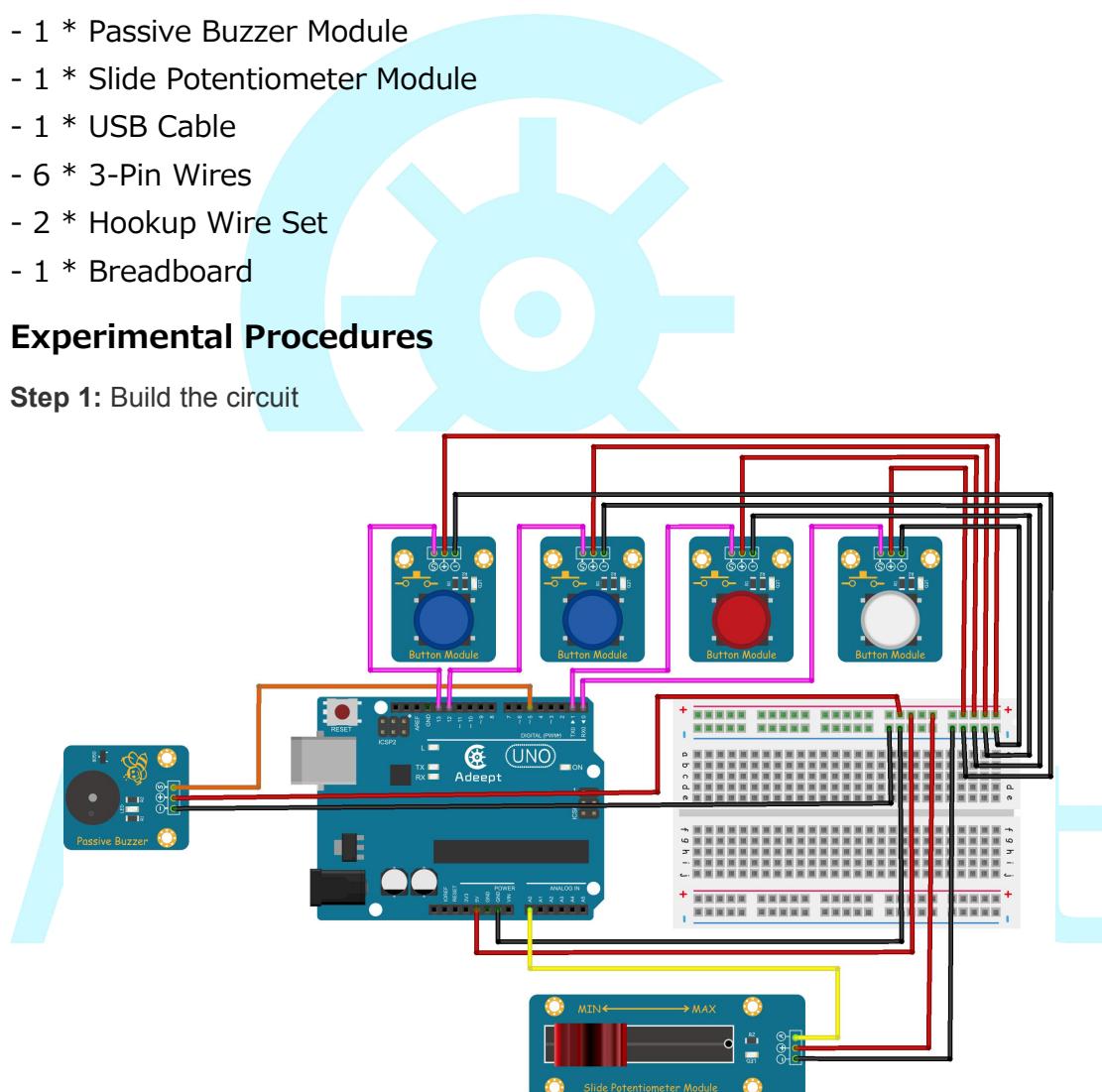
# **Lesson 41 A Simple Piano**

## Introduction

In this lesson, we use the potentiometer and four buttons to make the buzzer with a variety of sounds.

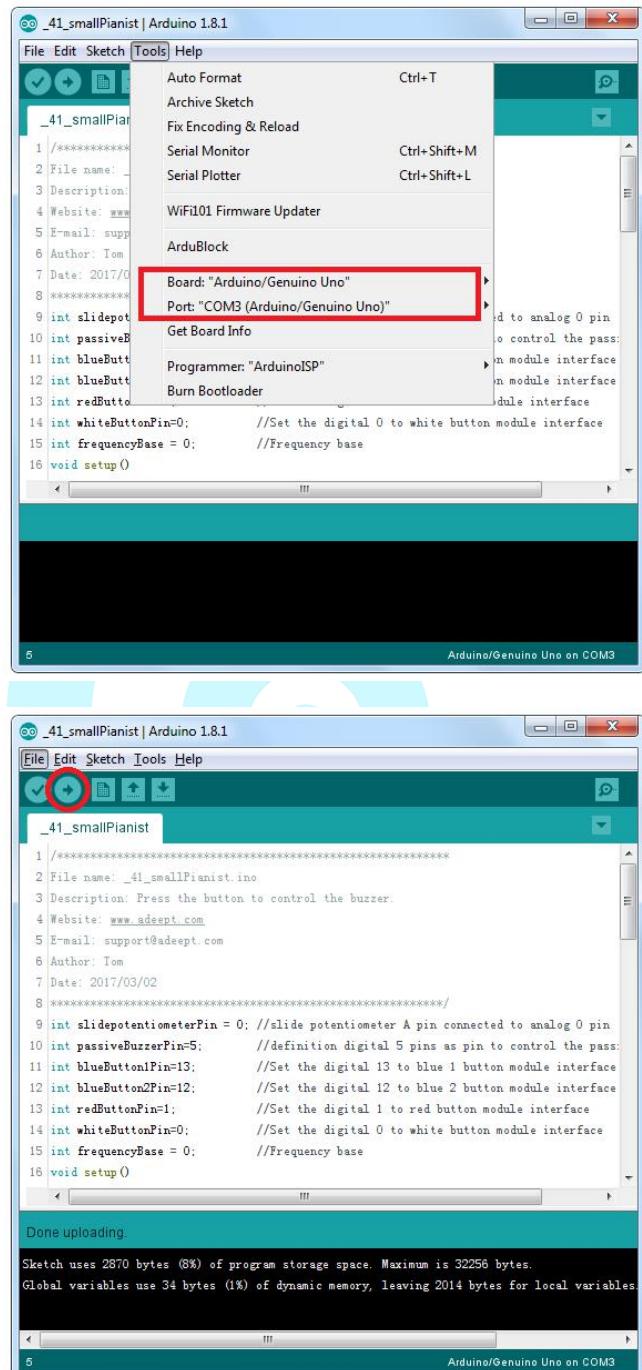
## Components

- 1 \* Adeept Arduino UNO R3 Board
  - 4 \* Button Module
  - 1 \* Passive Buzzer Module
  - 1 \* Slide Potentiometer Module
  - 1 \* USB Cable
  - 6 \* 3-Pin Wires
  - 2 \* Hookup Wire Set
  - 1 \* Breadboard

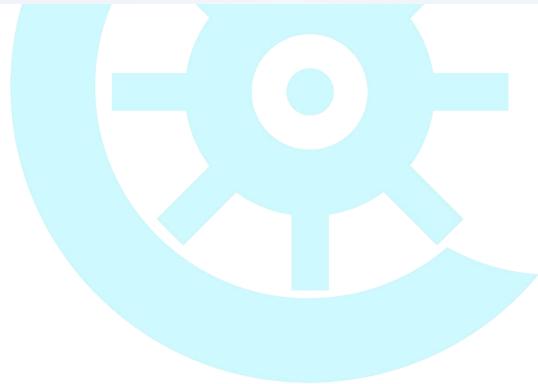
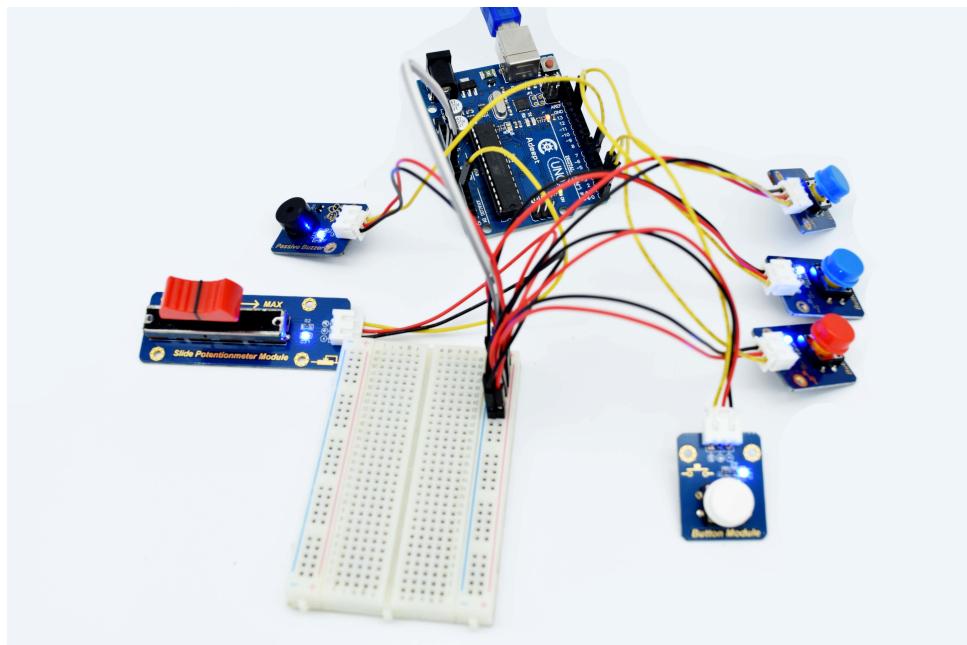


## Step 2: Program 41 smallPianist.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



When you press different buttons, the buzzer makes different sounds. Slide potentiometer module allows the buzzer to make more sounds.



# Adeept

# Lesson 42 Change The Color of The RGB LED

## Introduction

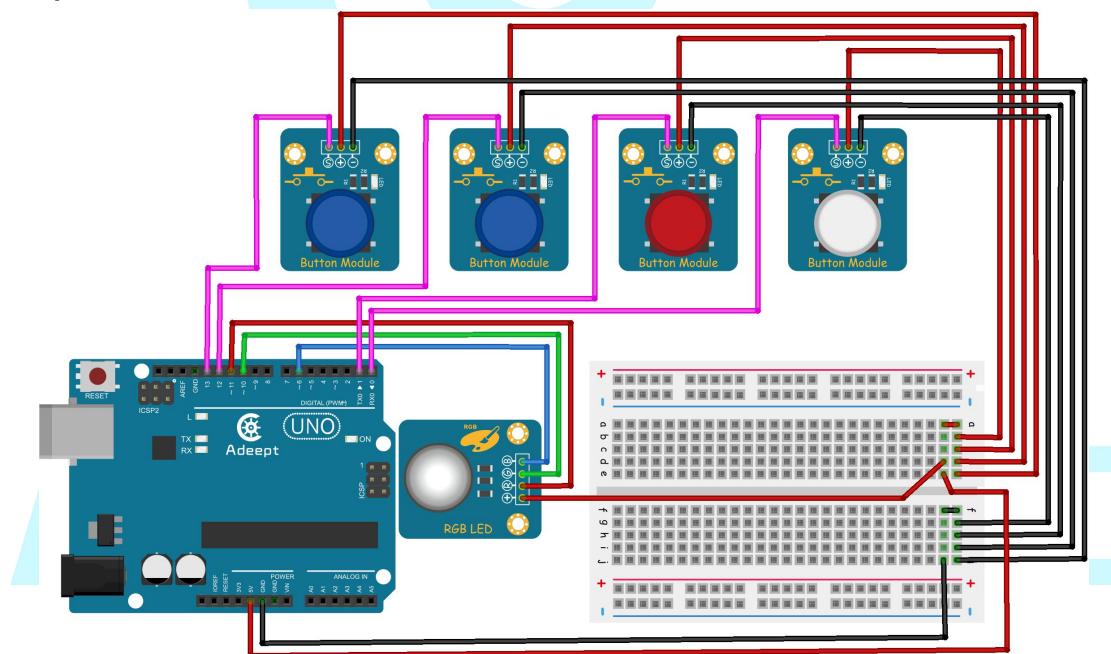
In this lesson, we will control an RGB LED through four buttons to change the color of the LED.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 4 \* Button Module
- 1 \* RGB Module
- 1 \* USB Cable
- 5 \* 3-Pin Wires
- 4 \* Hookup Wire Set
- 1 \* Breadboard

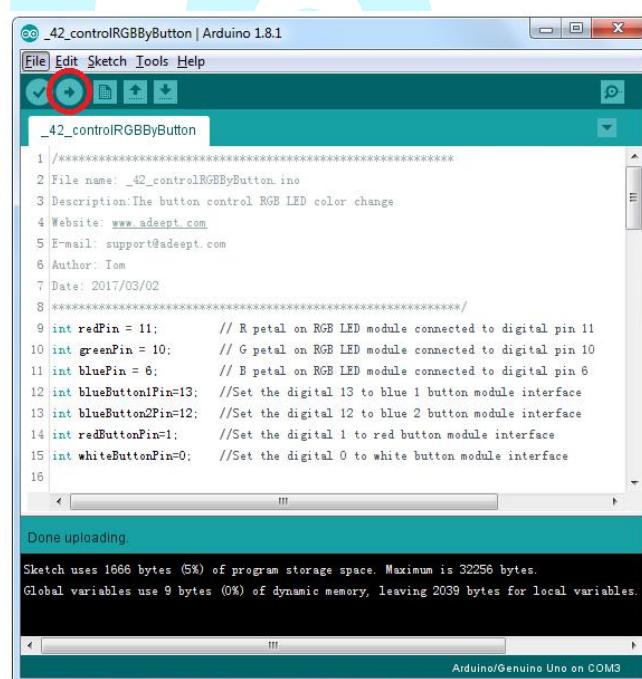
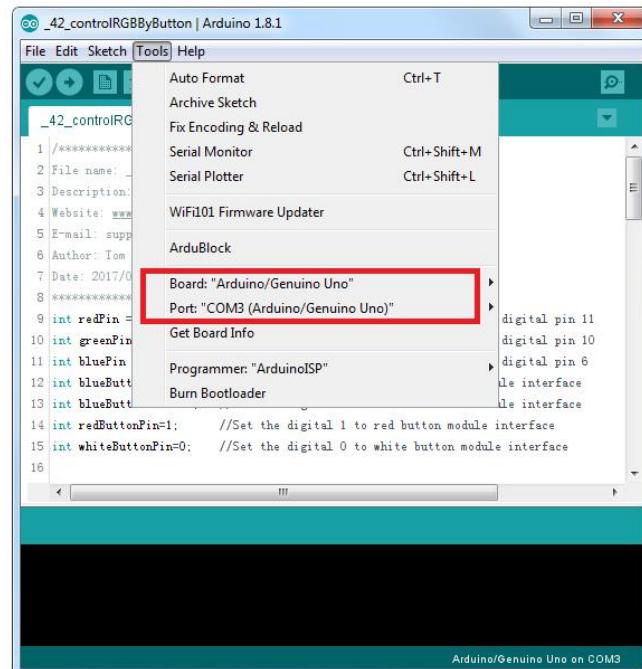
## Experimental Procedures

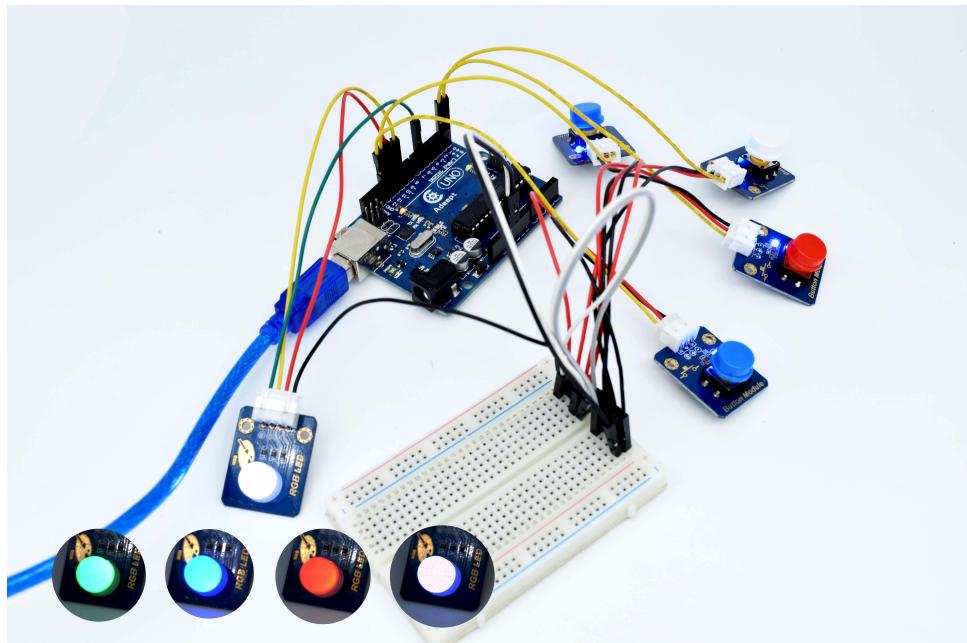
### Step 1: Build the circuit



**Step 2:** Program `_42_controlRGBByButton.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





# Adeept

# Lesson 43 A Simple Light Control Lamp

## Introduction

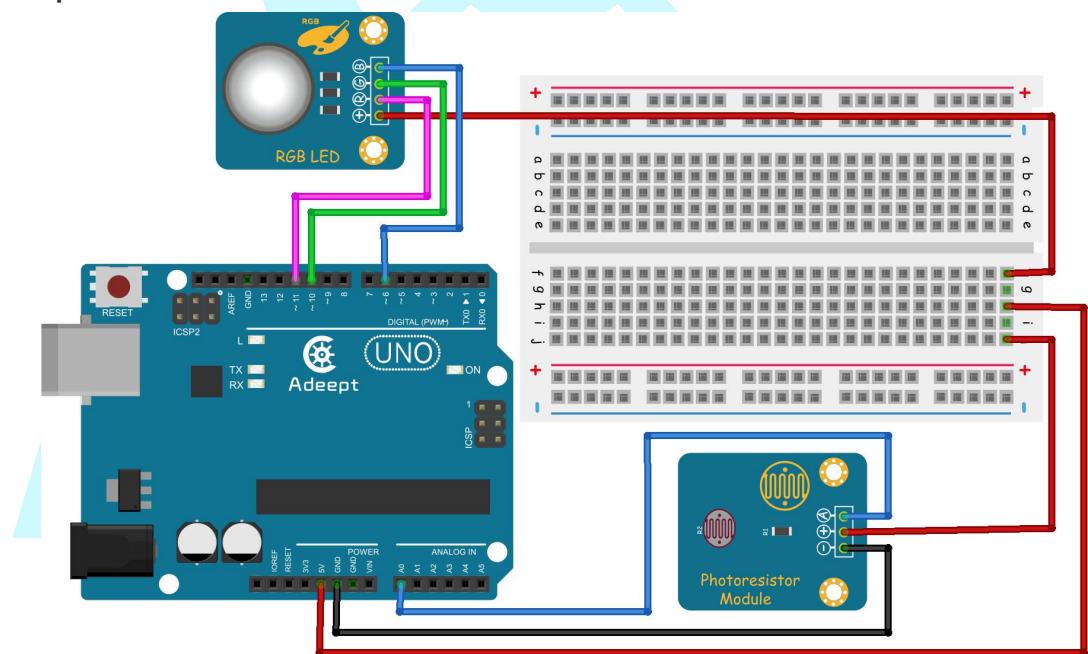
In this lesson, we will use photoresistor module and RGB LED to build a simple light control street lamp model. The lamp will automatically open at night and automatically closed at day.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Photoresistor Module
- 1 \* RGB Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires
- 1 \* 3-Pin Wires
- 1 \* Hookup Wire Set
- 1 \* Breadboard

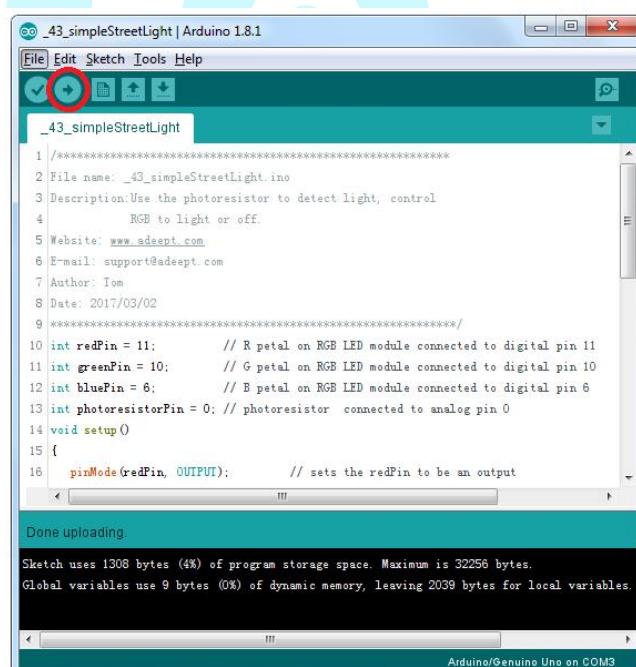
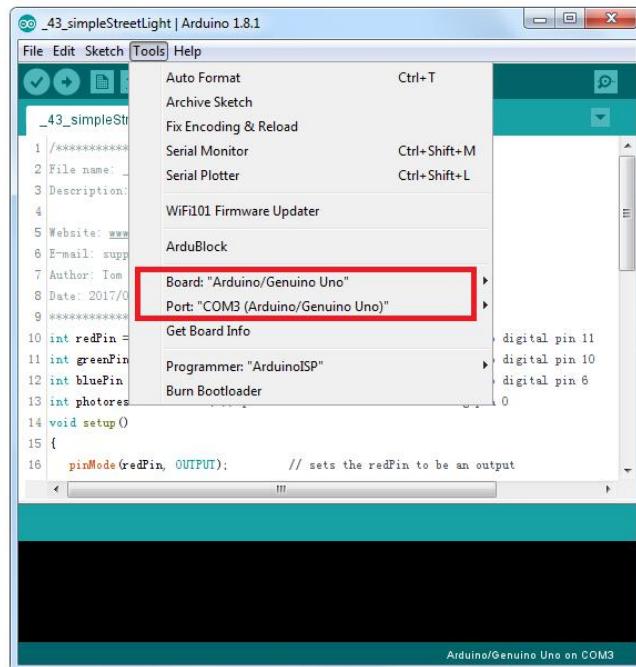
## Experimental Procedures

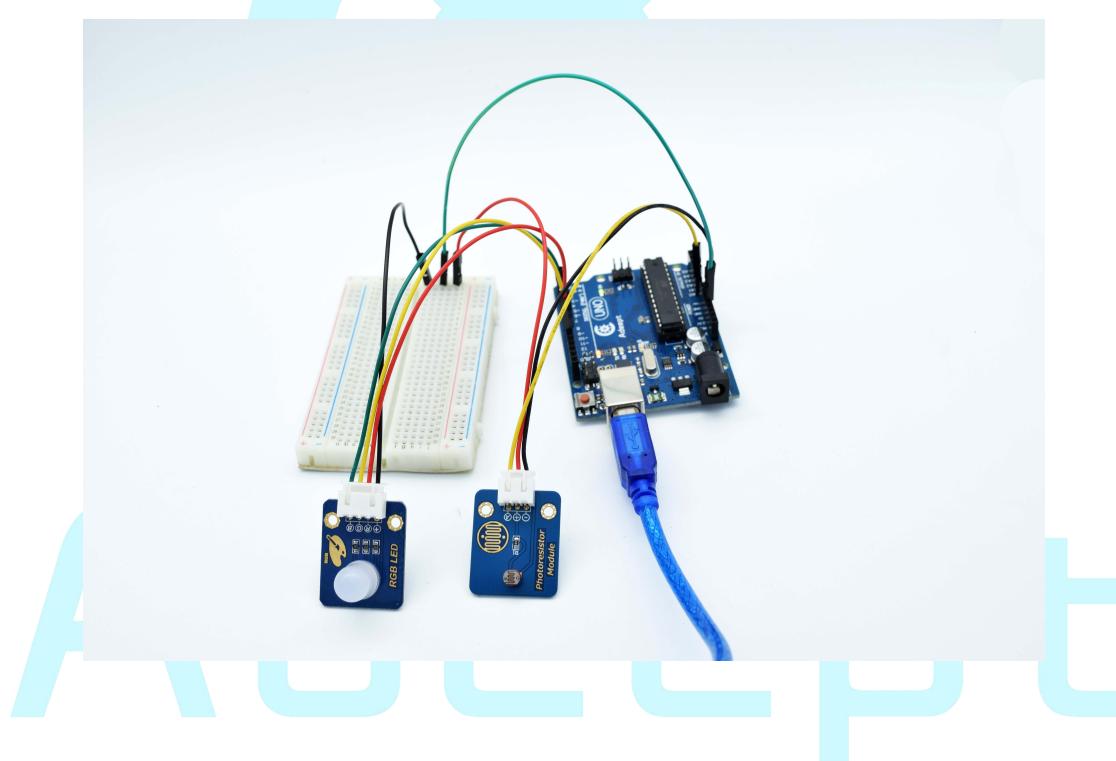
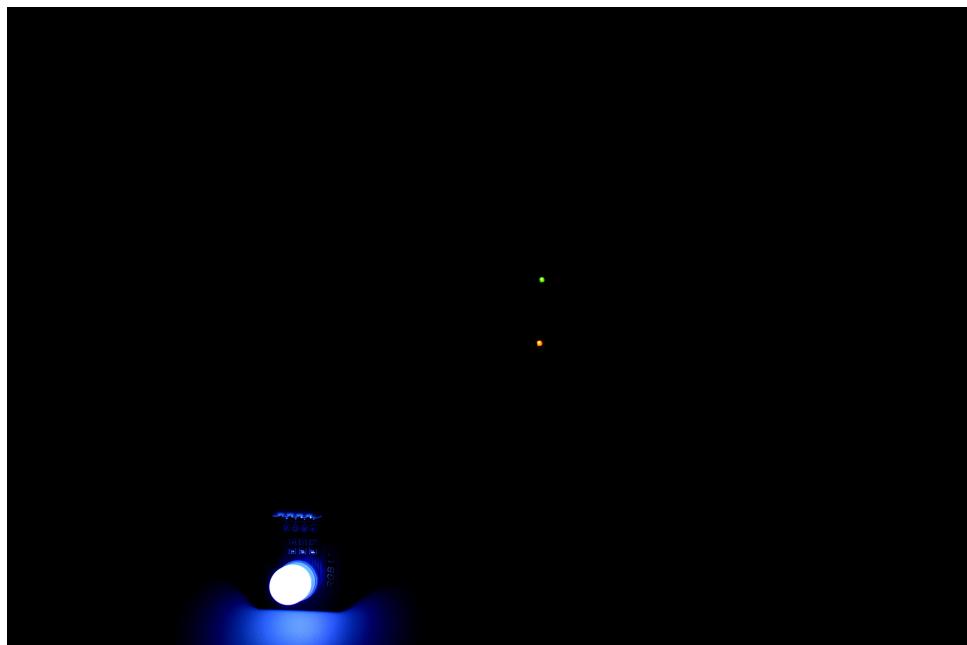
### Step 1: Build the circuit



**Step 2:** Program `_43_simpleStreetLight.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





# Lesson 44 Control Segment Display by Rotary Encoder

## Introduction

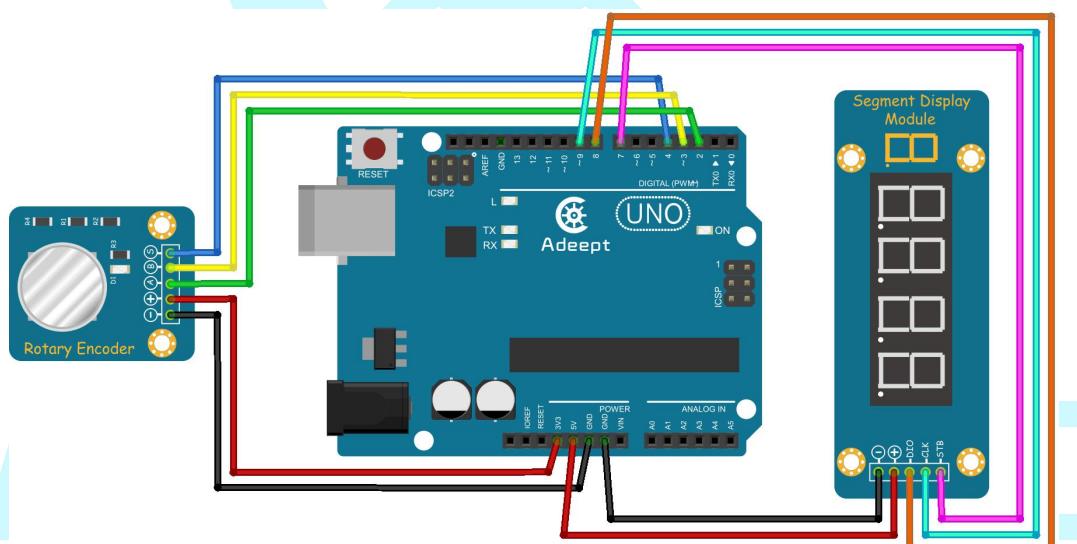
In this lesson, we will rotate the rotary knob to change the data displayed on the segment display.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Segment Display Module
- 1 \* Rotary Encoder Module
- 1 \* USB Cable
- 2 \* 5-Pin Wires
- 1 \* Breadboard

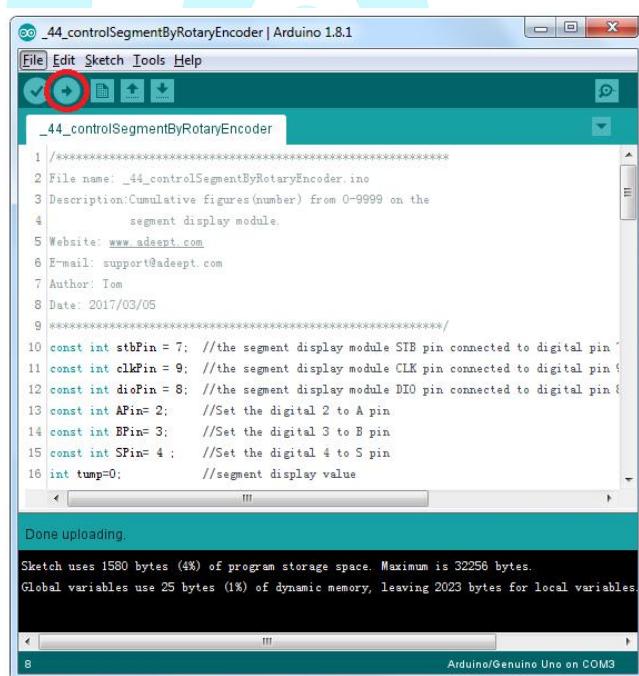
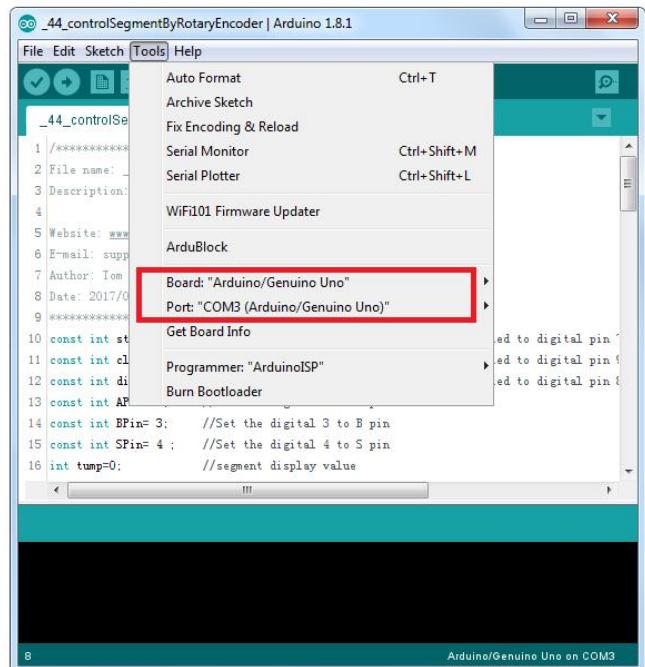
## Experimental Procedures

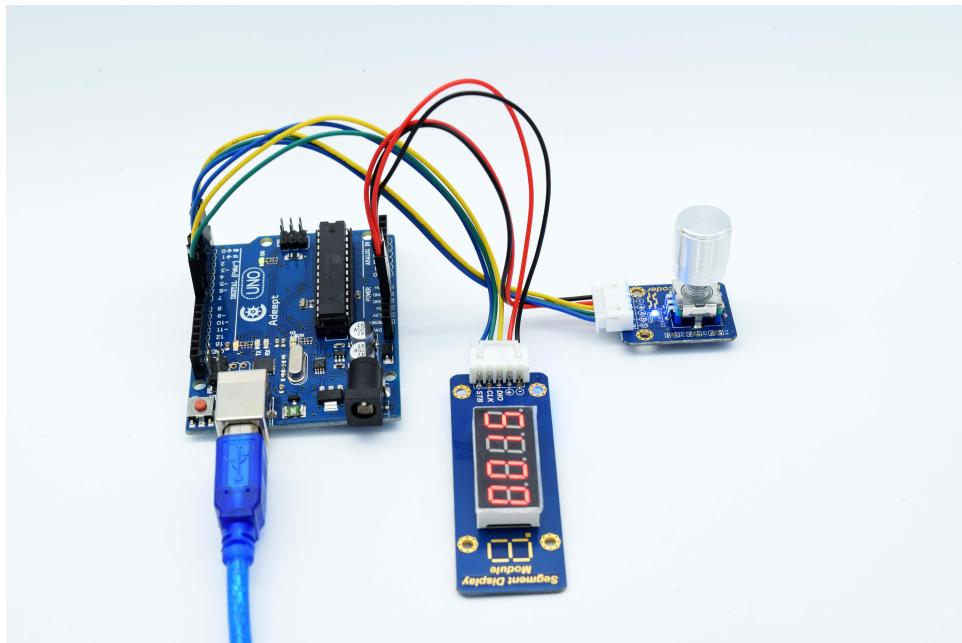
### Step 1: Build the circuit



**Step 2:** Program `_44_controlSegmentByRotaryEncoder.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





Turning the knob clockwise, the number displayed on the segment display will be increased, otherwise, it will be decreased. Press the knob, the data will return to zero.

# Adeept

# Lesson 45 A Simple Temperature & Humidity Monitoring and Alarm System(1)

## Introduction

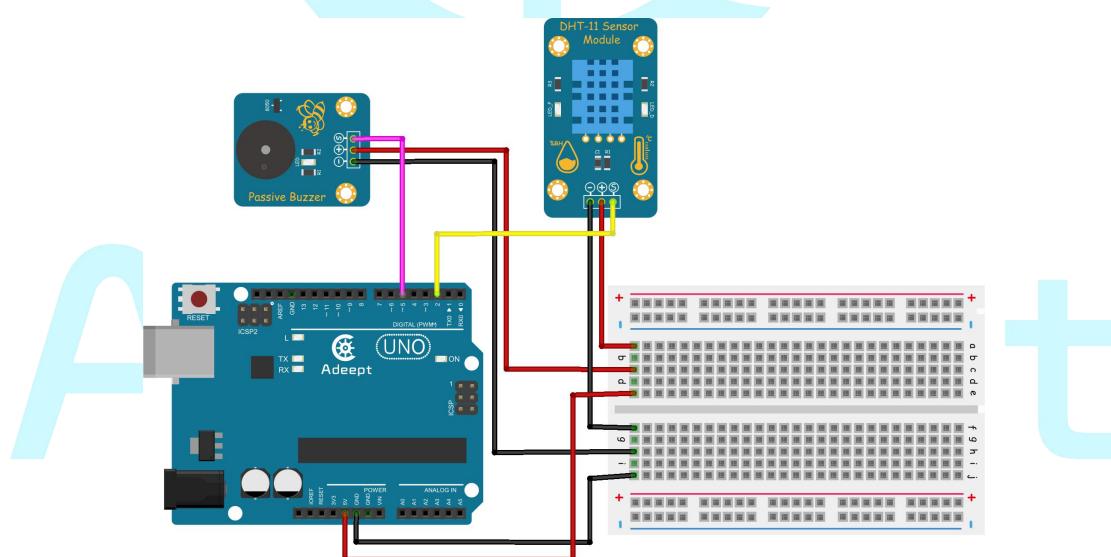
In this lesson, we will build a temperature & humidity monitoring and alarm system based on a passive buzzer and a DHT-11 sensor module.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* DHT-11 Sensor Module
- 1 \* Passive Buzzer Module
- 1 \* USB Cable
- 2 \* 3-Pin Wires
- 1 \* Breadboard
- 2 \* Hookup Wire Set

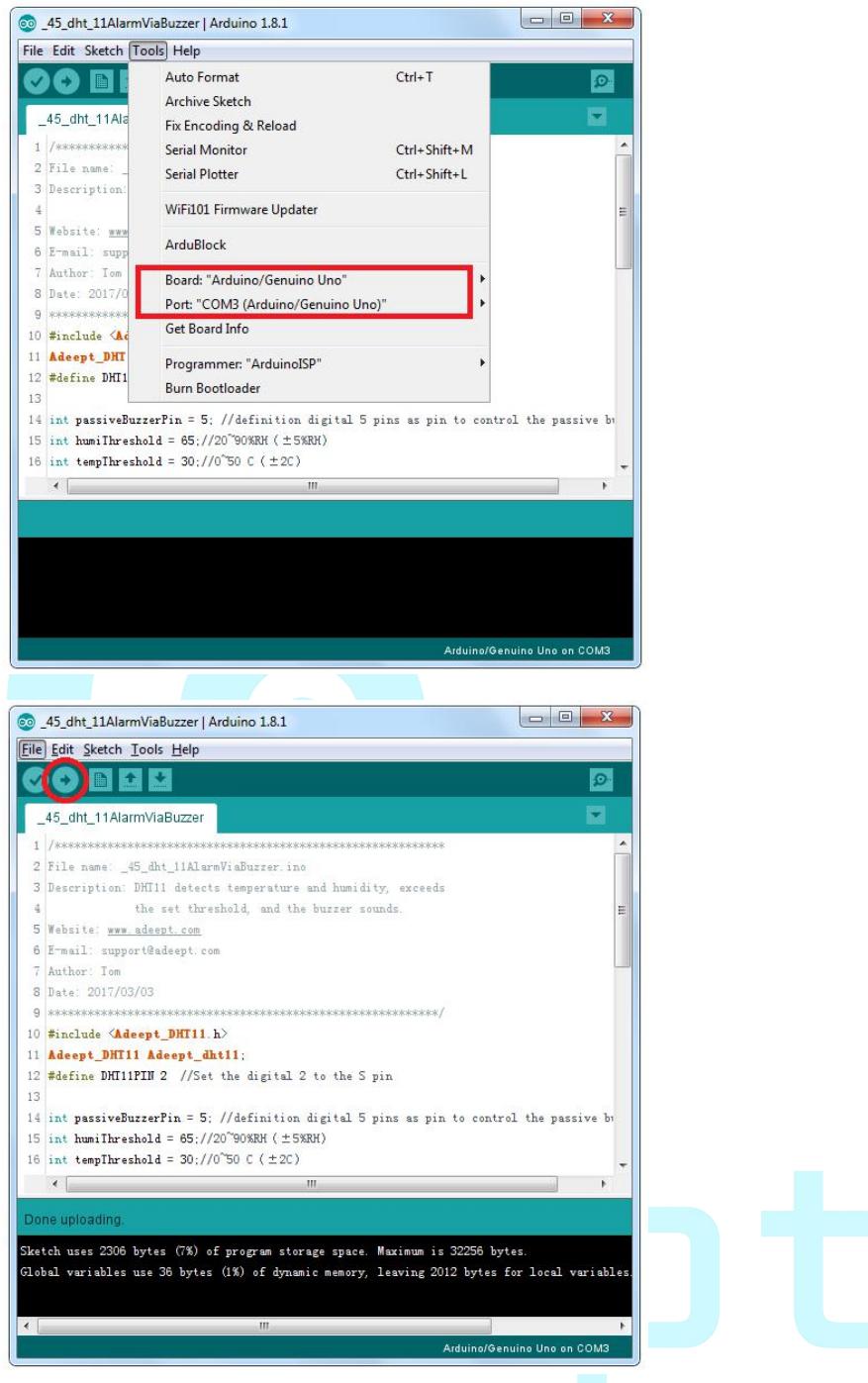
## Experimental Procedures

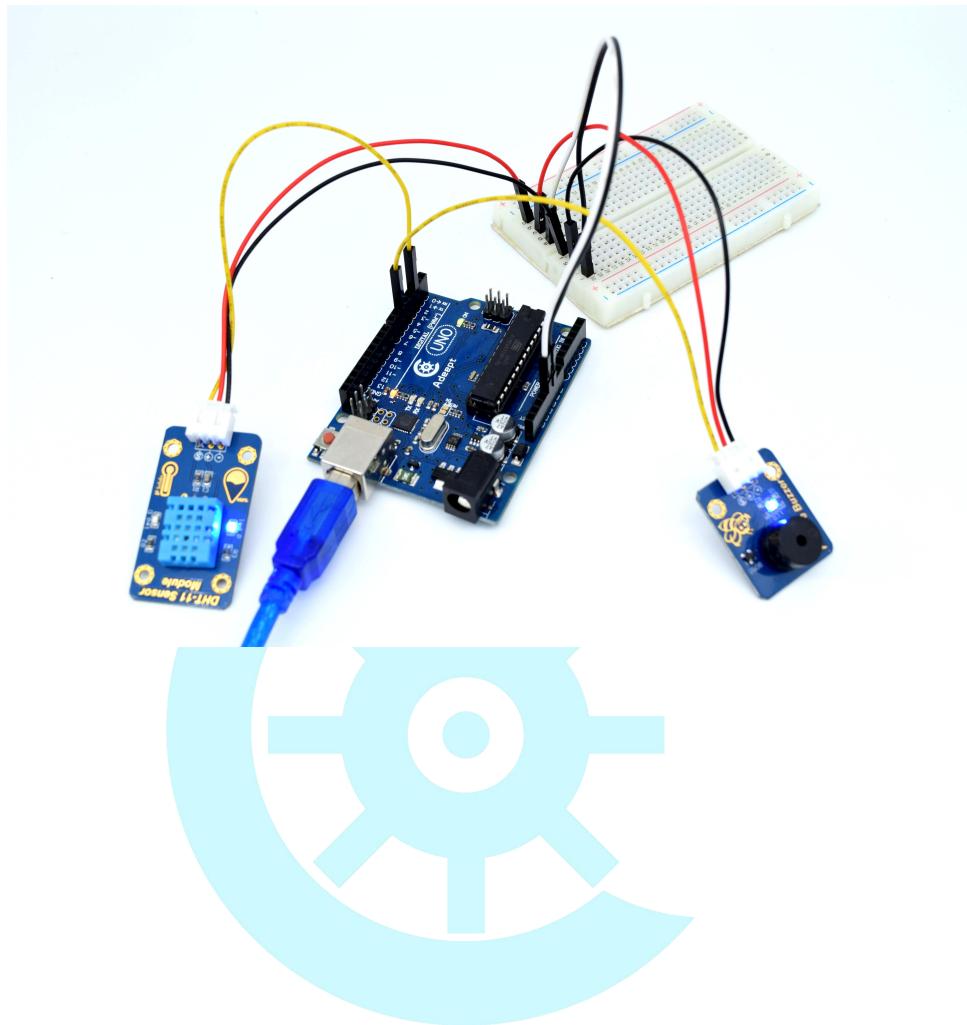
### Step 1: Build the circuit



**Step 2:** Program `_45_dht_11AlarmViaBuzzer.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





# Adeept

# Lesson 46 A Simple Temperature & Humidity Monitoring and Alarm System(2)

## Introduction

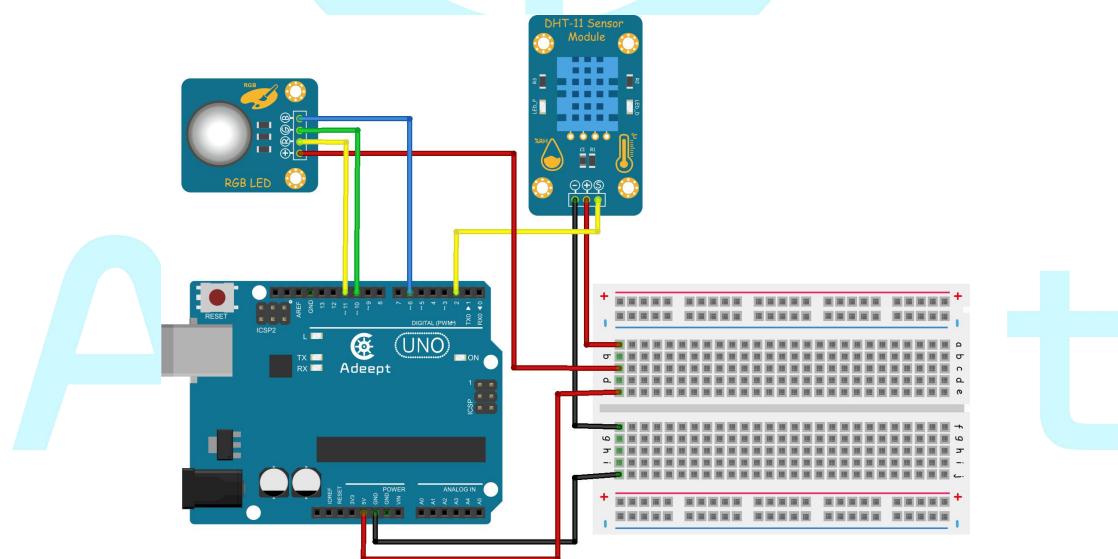
In this lesson, we use the RGB module and DHT-11 sensor module to do a temperature and humidity alarm system.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* DHT-11 Sensor Module
- 1 \* RGB Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires
- 1 \* 4-Pin Wires
- 1 \* Breadboard
- 2 \* Hookup Wire Set

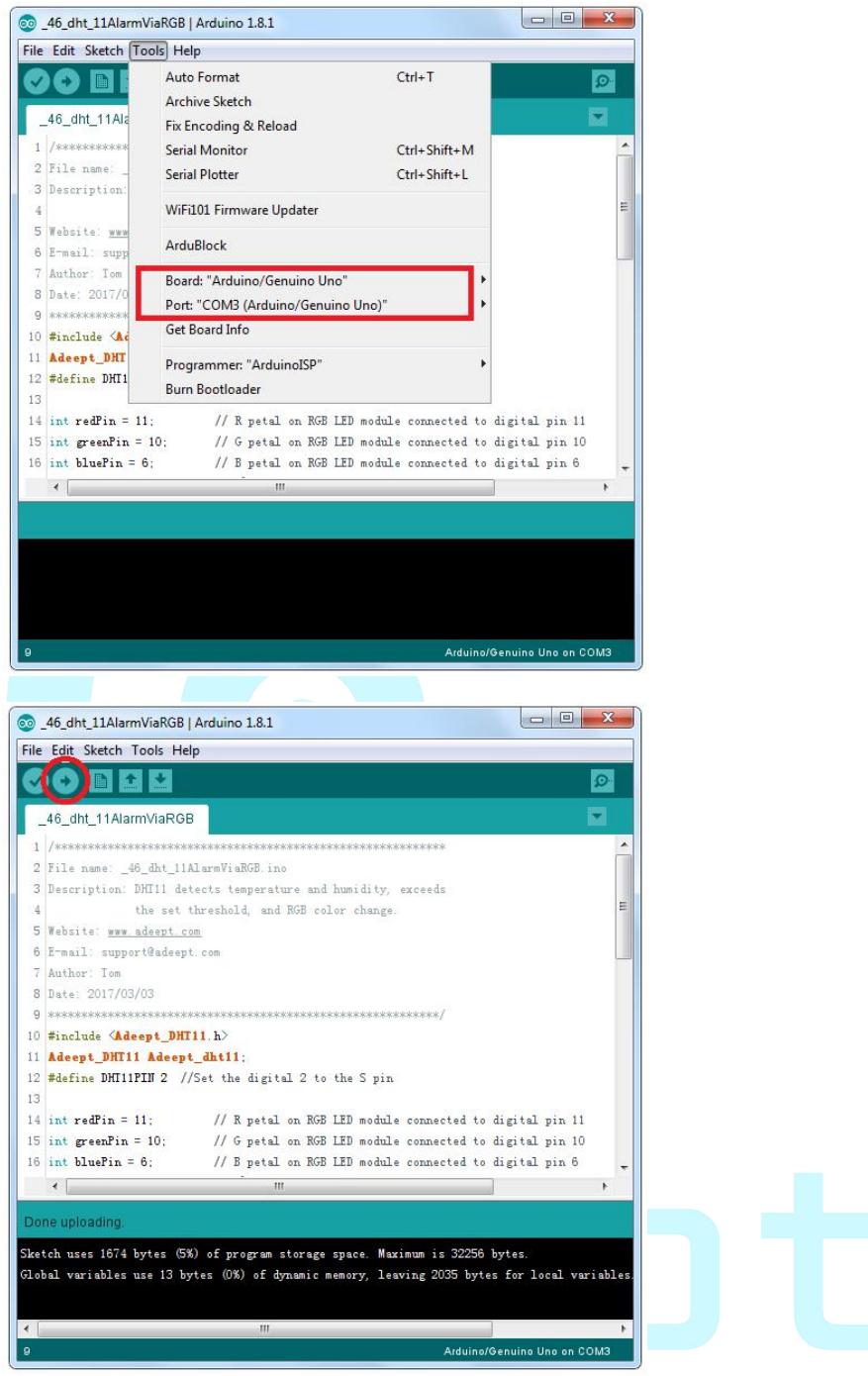
## Experimental Procedures

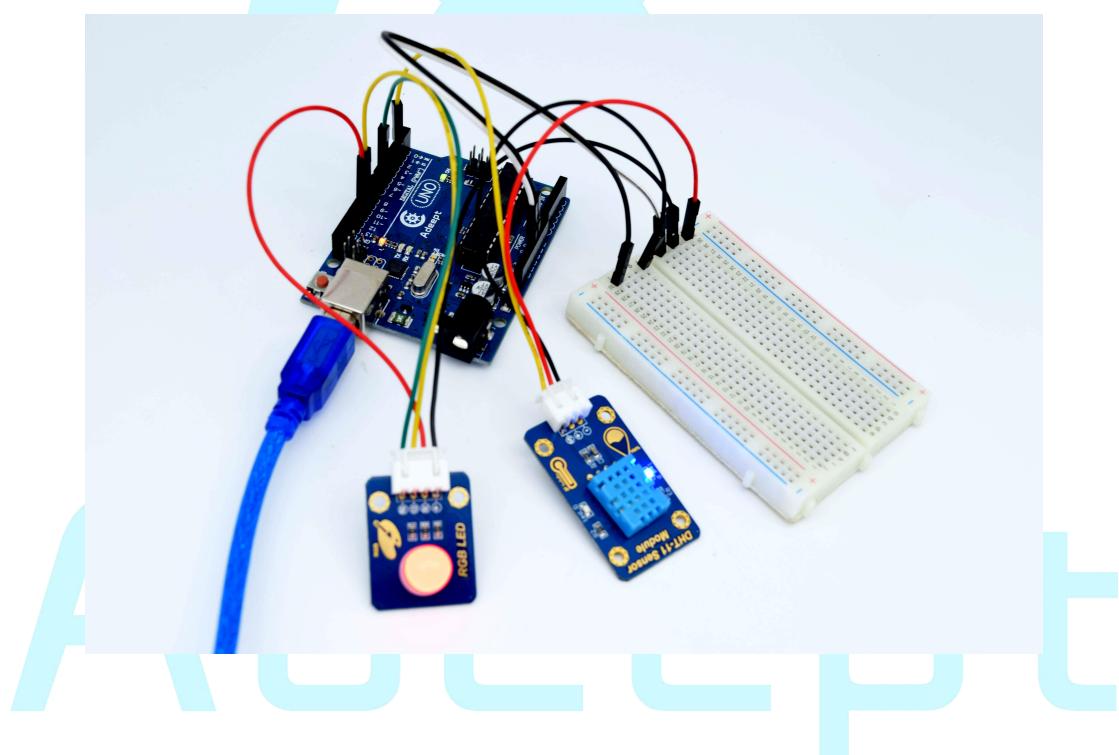
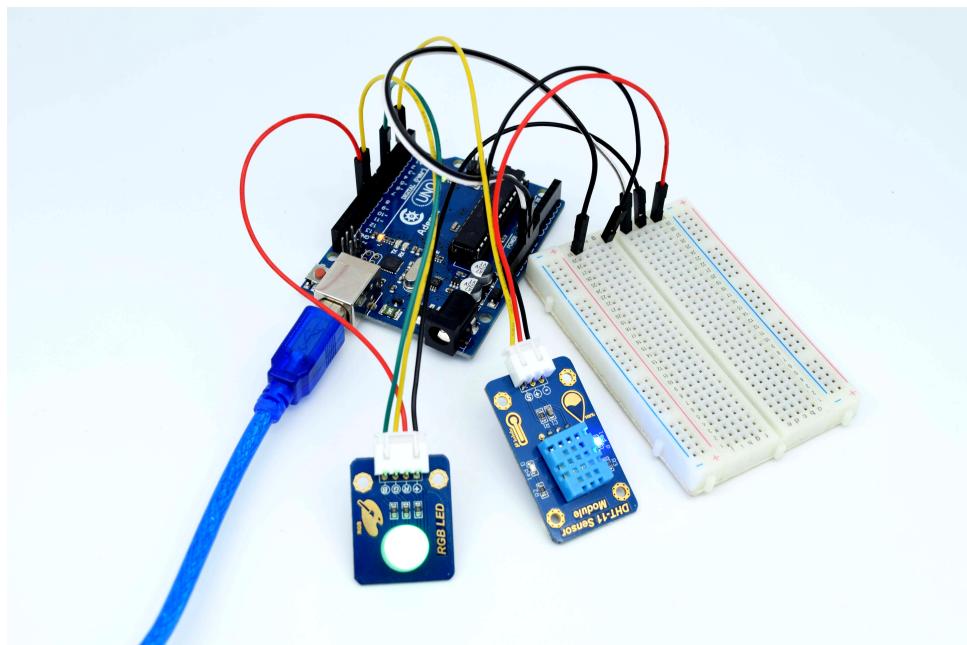
### Step 1: Build the circuit



**Step 2:** Program `_46_dht_11AlarmViaRGB.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





# **Lesson 47 A Simple Flammable Gases Monitoring and Alarm System(1)**

## Introduction

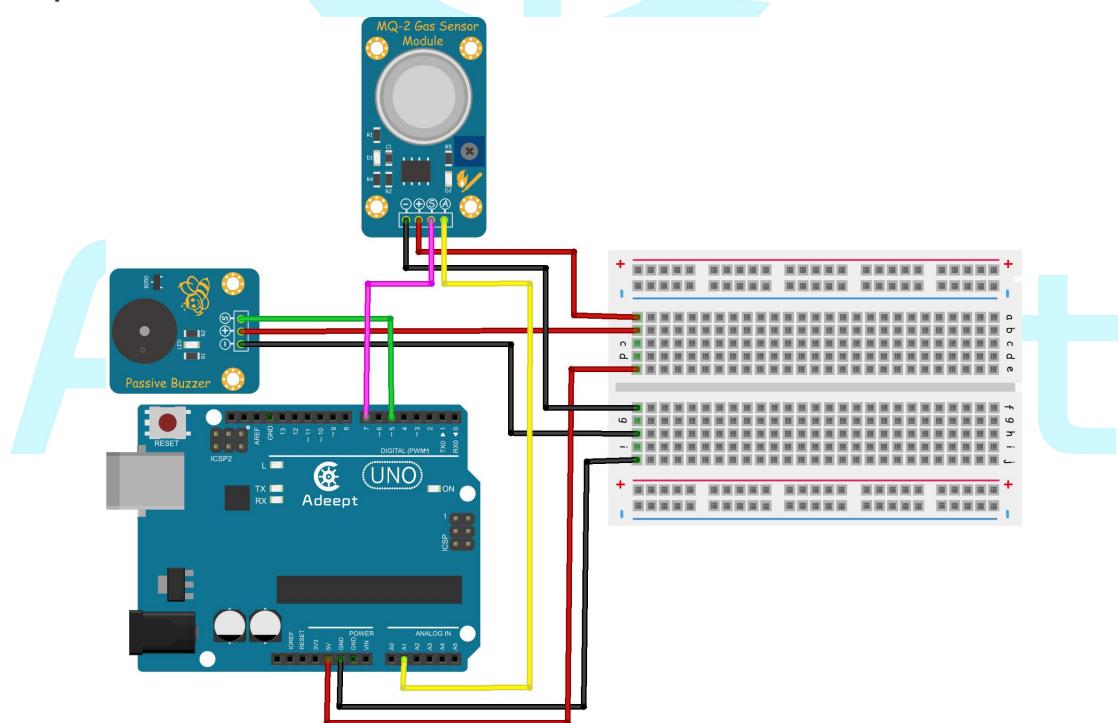
In this lesson, we use the passive buzzer module and MQ-2 gas sensor module to do a MO-2 Flammable gas alarm system.

## Components

- 1 \* Adeept Arduino UNO R3 Board
  - 1 \* MQ-2 Gas Sensor Module
  - 1 \* Passive Buzzer Module
  - 1 \* USB Cable
  - 1 \* 3-Pin Wires
  - 1 \* 4-Pin Wires
  - 1 \* Breadboard
  - 2 \* Hookup Wire Set

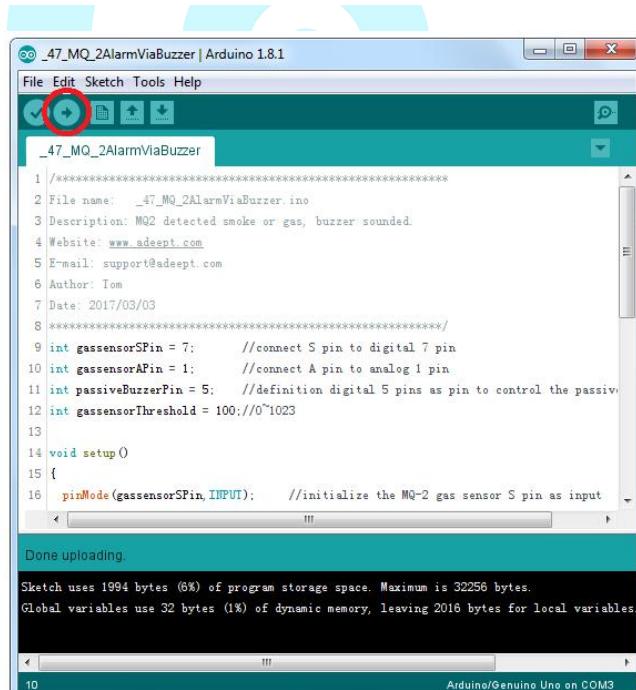
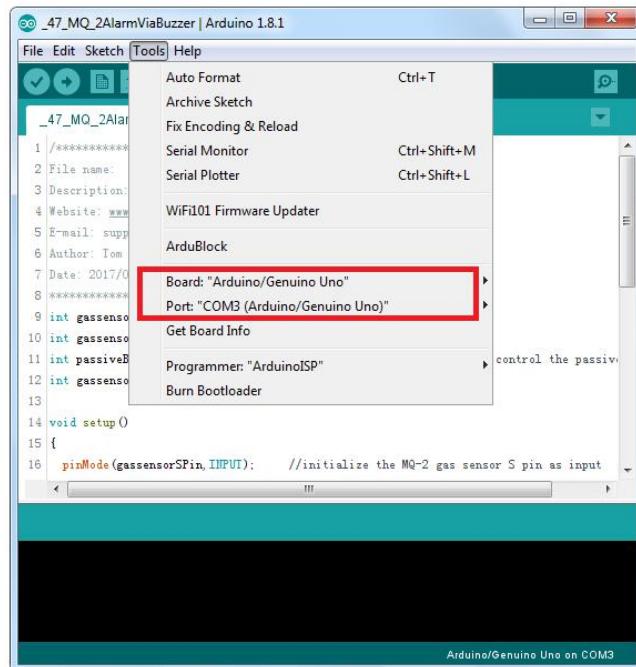
## Experimental Procedures

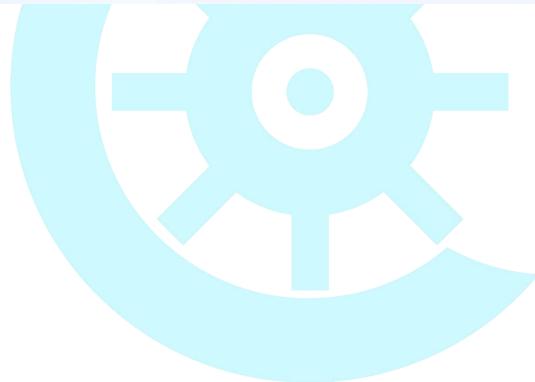
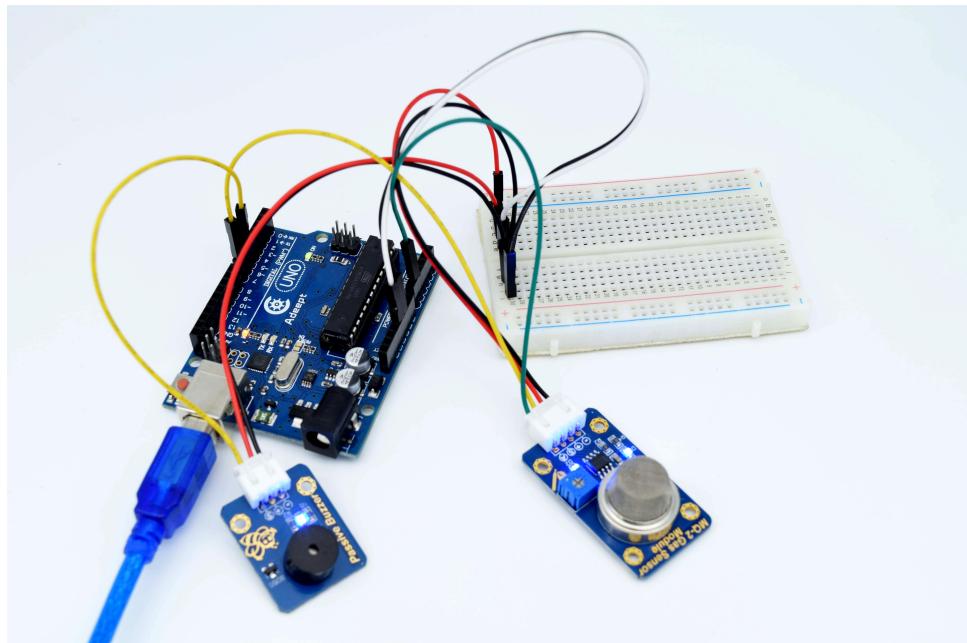
## Step 1: Build the circuit



**Step 2:** Program \_47\_MQ\_2AlarmViaBuzzer.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.





# Adeept

# Lesson 48 A Simple Flammable Gases Monitoring and Alarm System(2)

## Introduction

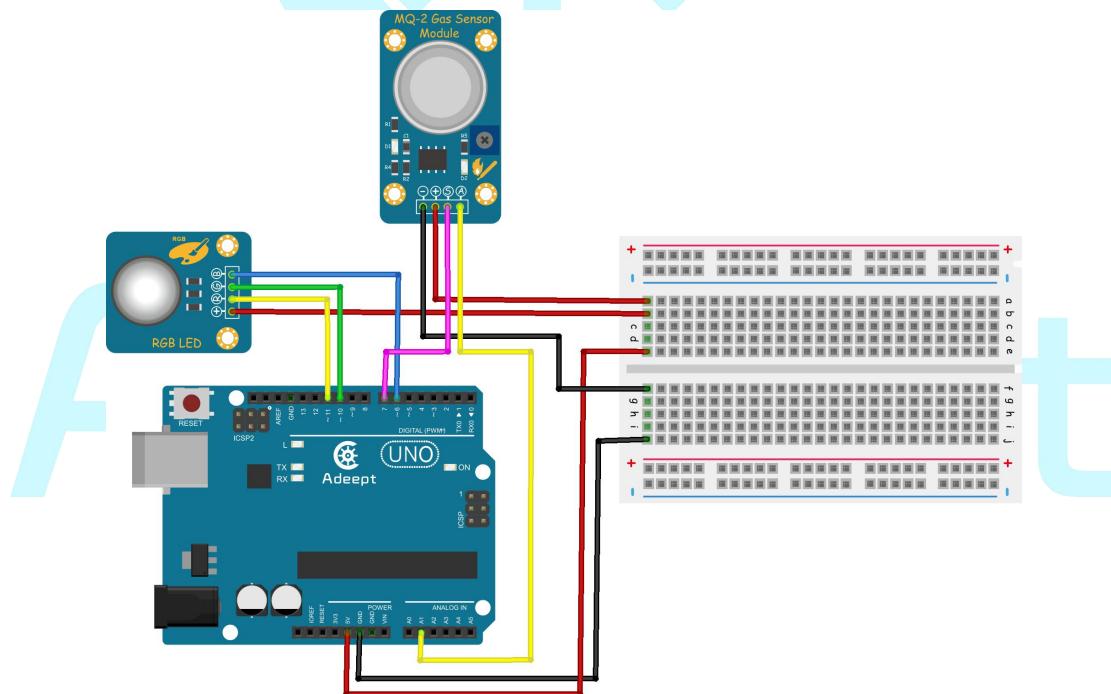
In this lesson, we use the RGB module and MQ-2 gas sensor module to do a MO-2 Flammable gas alarm system.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* MQ-2 Gas Sensor Module
- 1 \* RGB Module
- 1 \* USB Cable
- 2 \* 4-Pin Wires
- 1 \* Breadboard
- 2 \* Hookup Wire Set

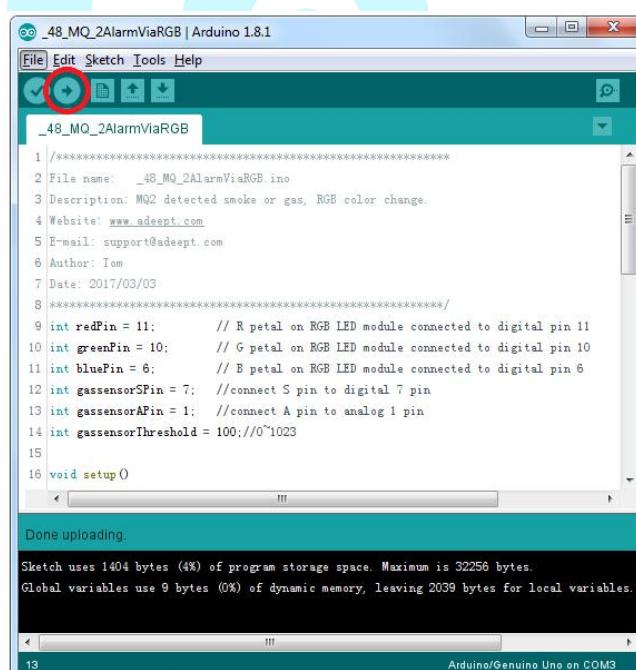
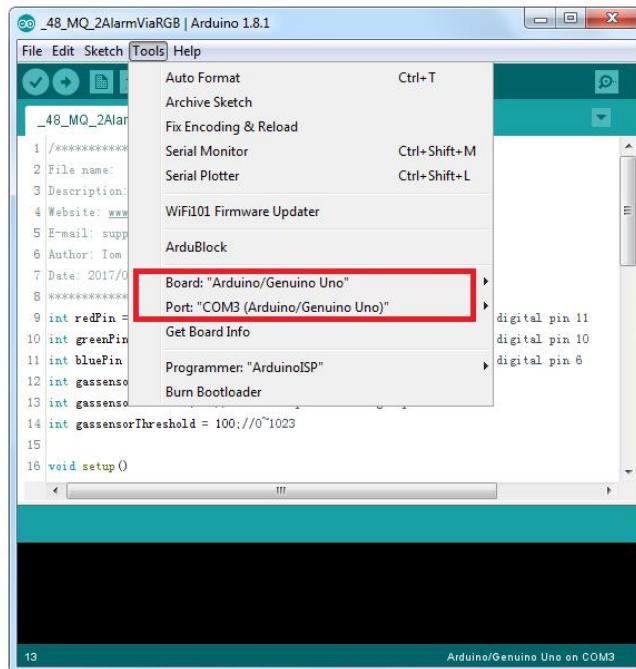
## Experimental Procedures

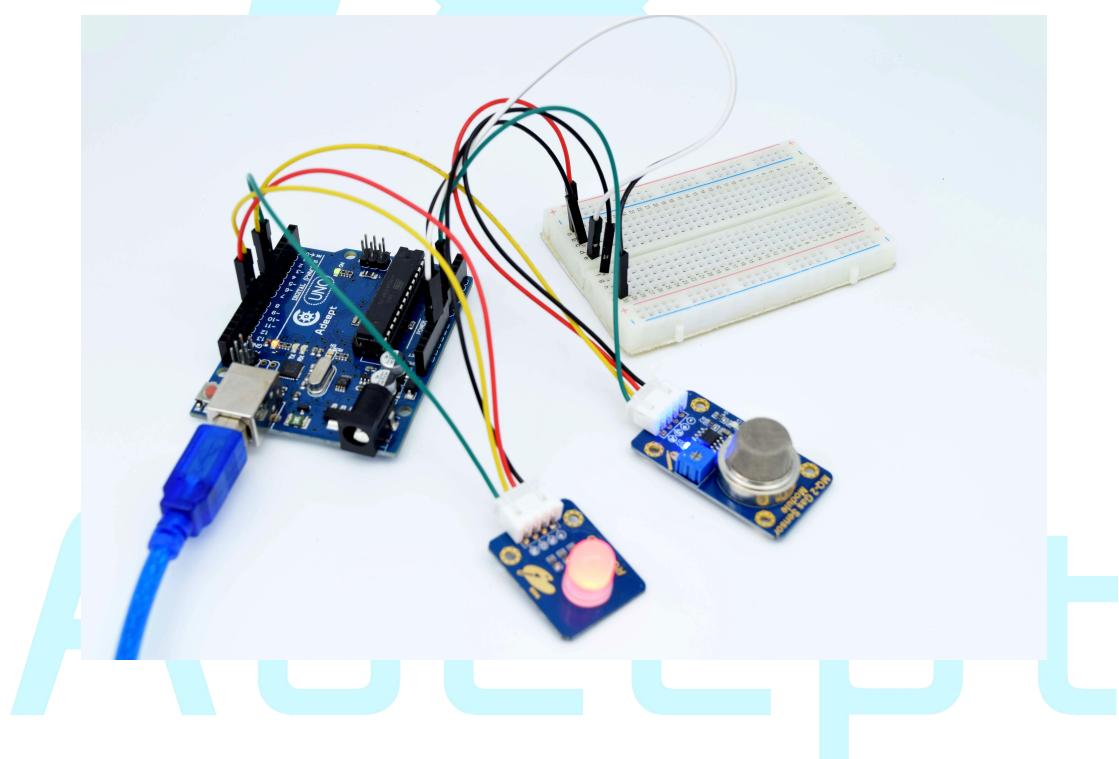
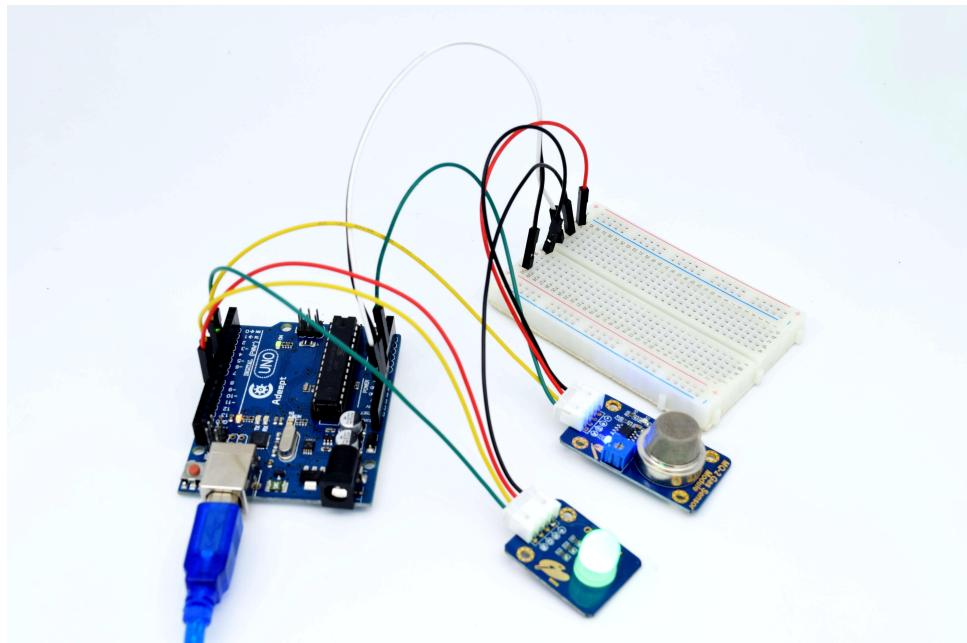
### Step 1: Build the circuit



**Step 2:** Program `_48_MQ_2AlarmViaRGB.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





# Lesson 49 A Simple Clock

## Introduction

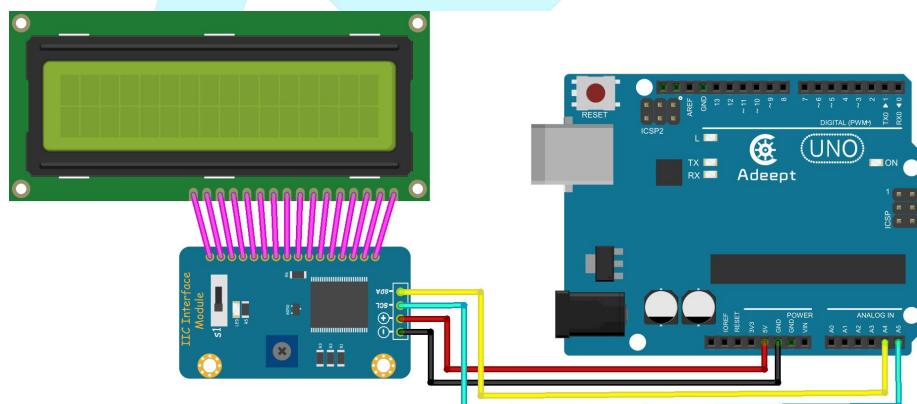
In this lesson, we will build a simple clock based on a LCD1602 and Arduino UNO.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* LCD1602 Module
- 1 \* IIC Interface Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

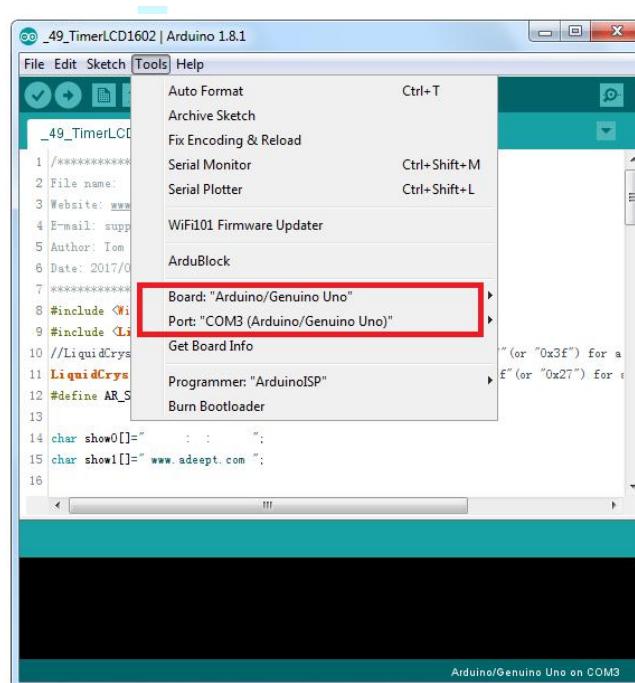
## Experimental Procedures

### Step 1: Build the circuit



### Step 2: Program \_49\_TimerLCD1602.ino

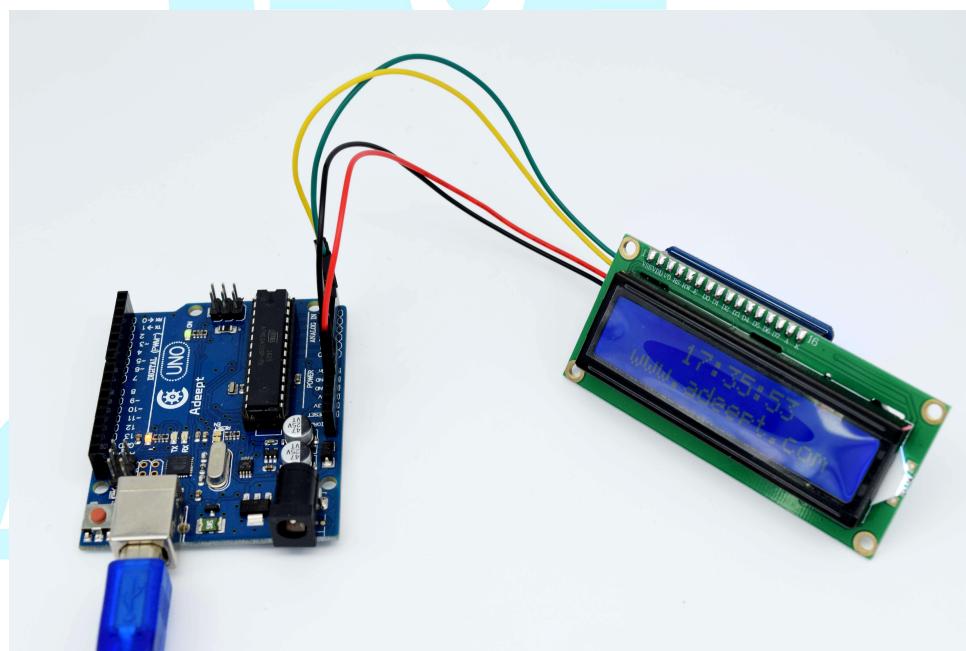
### Step 3: Compile and download the sketch to the UNO R3 board.



The screenshot shows the Arduino IDE interface with the title bar "49\_TimerLCD1602 | Arduino 1.8.1". A red circle highlights the "Upload" button (a blue arrow pointing right) in the toolbar. The main window displays the code for "49\_TimerLCD1602.ino". The code includes comments and includes for Wire.h and LiquidCrystal\_I2C.h, followed by I2C LCD initialization code. Below the code, a message says "Done uploading." and provides memory usage details: "Sketch uses 5106 bytes (15%) of program storage space. Maximum is 32256 bytes. Global variables use 265 bytes (12%) of dynamic memory, leaving 1783 bytes for local variables". At the bottom, it says "Arduino/Genuino Uno on COM3".

```
1 //*****  
2 File name: _49_TimerLCD1602.ino  
3 Website: www.adeept.com  
4 E-mail: support@adeept.com  
5 Author: Tom  
6 Date: 2017/03/02  
7 *****  
8 #include <Wire.h>  
9 #include <LiquidCrystal_I2C.h>  
10 //LiquidCrystal_I2C lcd(0x27, 16, 2); // set the LCD address to "0x27" (or "0x3E") for a  
11 LiquidCrystal_I2C lcd(0x3F, 16, 2); // set the LCD address to "0x3F" (or "0x27") for a  
12 #define AR_SIZE(a) sizeof(a) / sizeof(s[0])  
13  
14 char show0[]=" : : : ";  
15 char show1[]=" www.adeept.com ";  
16
```

Done uploading.  
Sketch uses 5106 bytes (15%) of program storage space. Maximum is 32256 bytes.  
Global variables use 265 bytes (12%) of dynamic memory, leaving 1783 bytes for local variables  
6 Arduino/Genuino Uno on COM3



# Lesson 50 Arduino Interacts with Processing(IEC Module)

## Introduction

Sarting from this lesson, we will start experimenting with the interaction between Arduino and Prosssing. we will create a simple human-computer interaction interface based on Processing, which contains four different colors of the virtual button(LED shape), when the mouse pointer points to a virtual button, the corresponding LED will be turned on.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 4 \* LED Module
- 1 \* USB Cable
- 4 \* 3-Pin Wires
- 1 \* Breadboard
- 1 \* Hookup Wire Set



## Principle

**Processing key function:**

**setup()**

The setup() function is run once, when the program starts. It's used to define initial enviroment properties such as screen size and to load media such as images and fonts as the program starts. There can only be one setup() function for each program and it shouldn't be called again after its initial execution.

If the sketch is a different dimension than the default, the size() function or fullScreen() function must be the first line in setup().

Note: Variables declared within setup() are not accessible within other functions, including draw().

**draw()**

Called directly after setup(), the draw() function continuously executes the lines of code contained inside its block until the program is stopped or noLoop() is called. draw() is called automatically and should never be called explicitly. All Processing programs update the screen at the end of draw(), never earlier.

To stop the code inside of draw() from running continuously, use noLoop(), redraw() and loop(). If noLoop() is used to stop the code in draw() from running, then redraw() will cause the code inside draw() to run a single time, and loop() will cause the code inside draw() to resume running continuously.

The number of times draw() executes in each second may be controlled with the frameRate() function.

It is common to call background() near the beginning of the draw() loop to clear the contents of the window, as shown in the first example above. Since pixels drawn to the

window are cumulative, omitting background() may result in unintended results. There can only be one draw() function for each sketch, and draw() must exist if you want the code to run continuously, or to process events such as mousePressed(). Sometimes, you might have an empty call to draw() in your program, as shown in the second example above.

The Serial library reads and writes data to and from external devices one byte at a time. It allows two computers to send and receive data. This library has the flexibility to communicate with custom microcontroller devices and to use them as the input or output to Processing programs. The serial port is a nine pin I/O port that exists on many PCs and can be emulated through USB.

Name

**PImage**

Examples



```
PImage photo;  
void setup() {  
    size(100, 100);  
    photo = loadImage("laDefense.jpg");  
}  
void draw() {  
    image(photo, 0, 0);  
}
```

Description Datatype for storing images. Processing can display .gif, .jpg, .tga, and .png images. Images may be displayed in 2D and 3D space. Before an image is used, it must be loaded with the loadImage() function. The PImage class contains fields for the width and height of the image, as well as an array called pixels[] that contains the values for every pixel in the image. The methods described below allow easy access to the image's pixels and alpha channel and simplify the process of compositing.

Before using the pixels[] array, be sure to use the loadPixels() method on the image to make sure that the pixel data is properly loaded.

To create a new image, use the createImage() function. Do not use the syntax new PImage().

Fields

pixels[] Array containing the color of every pixel in the image

width Image width

height Image height

Methods

**loadPixels()** Loads the pixel data for the image into its pixels[] array  
**updatePixels()** Updates the image with the data in its pixels[] array  
**resize()** Changes the size of an image to a new width and height  
**get()** Reads the color of any pixel or grabs a rectangle of pixels  
**set()** writes a color to any pixel or writes an image into another  
**mask()** Masks part of an image with another image as an alpha channel  
**filter()** Converts the image to grayscale or black and white  
**copy()** Copies the entire image  
**blend()** Copies a pixel or rectangle of pixels using different blending modes  
**save()** Saves the image to a TIFF, TARGA, PNG, or JPEG file

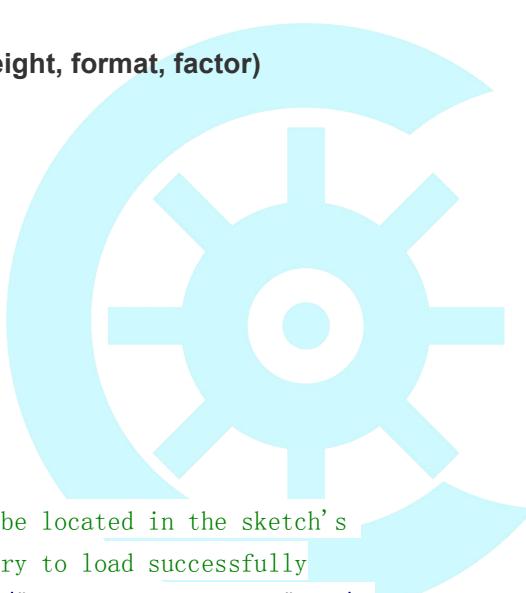
Constructor

**PImage(width, height, format, factor)**

Name

**PFont**

Examples



```
PFont font;  
// The font must be located in the sketch's  
// "data" directory to load successfully  
font = createFont("LetterGothicStd.ttf", 32);  
textFont(font);  
text("word", 10, 50);
```

Description PFont is the font class for Processing. To create a font to use with Processing, select "Create Font..." from the Tools menu. This will create a font in the format Processing requires and also adds it to the current sketch's data directory. Processing displays fonts using the .vlw font format, which uses images for each letter, rather than defining them through vector data. The loadFont() function constructs a new font and textFont() makes a font active. The list() method creates a list of the fonts installed on the computer, which is useful information to use with the createFont() function for dynamically converting fonts into a format to use with Processing.

To create a new font dynamically, use the createFont() function. Do not use the syntax new PFont().

Methods

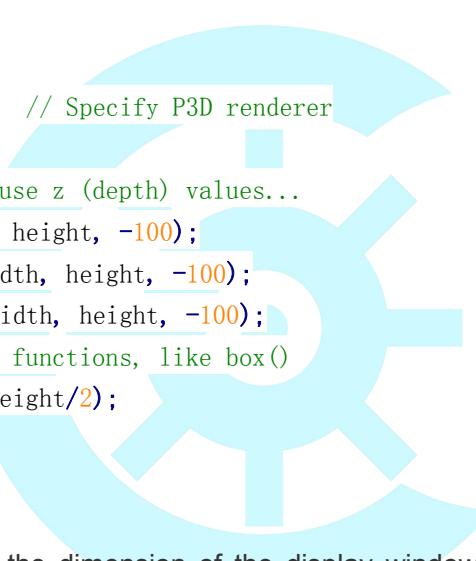
**list()** Gets a list of the fonts installed on the system

Name

**size()**

Examples

```
size(200, 100);
background(153);
line(0, 0, width, height);
void setup() {
    size(320, 240);
}
void draw() {
    background(153);
    line(0, 0, width, height);
}
```



```
size(150, 200, P3D); // Specify P3D renderer
background(153);
// With P3D, we can use z (depth) values...
line(0, 0, 0, width, height, -100);
line(width, 0, 0, width, height, -100);
line(0, height, 0, width, height, -100);
//...and 3D-specific functions, like box()
translate(width/2, height/2);
rotateX(PI/6);
rotateY(PI/6);
box(35);
```

**Description** Defines the dimension of the display window width and height in units of pixels. In a program that has the `setup()` function, the `size()` function must be the first line of code inside `setup()`.

The built-in variables `width` and `height` are set by the parameters passed to this function. For example, running `size(640, 480)` will assign 640 to the `width` variable and 480 to the `height` variable. If `size()` is not used, the window will be given a default size of 100 x 100 pixels.

The `size()` function can only be used once inside a sketch, and it cannot be used for resizing.

As of Processing 3, to run a sketch at the full dimensions of a screen, use the `fullScreen()` function, rather than the older way of using `size(displayWidth, displayHeight)`.

The maximum width and height is limited by your operating system, and is usually the width and height of your actual screen. On some machines it may simply be the number of pixels on your current screen, meaning that a screen of 800 x 600 could support `size(1600, 300)`, since that is the same number of pixels. This varies widely, so you'll have to try different rendering modes and sizes until you get what you're looking for. If you need something larger, use `createGraphics` to create a non-visible drawing surface.

The minimum width and height is around 100 pixels in each direction. This is the smallest that is supported across Windows, macOS, and Linux. We enforce the minimum size so that sketches will run identically on different machines.

The renderer parameter selects which rendering engine to use. For example, if you will be drawing 3D shapes, use P3D. In addition to the default renderer, other renderers are:

P2D (Processing 2D): 2D graphics renderer that makes use of OpenGL-compatible graphics hardware.

P3D (Processing 3D): 3D graphics renderer that makes use of OpenGL-compatible graphics hardware.

FX2D (JavaFX 2D): A 2D renderer that uses JavaFX, which may be faster for some applications, but has some compatibility quirks.

PDF: The PDF renderer draws 2D graphics directly to an Acrobat PDF file. This produces excellent results when you need vector shapes for high-resolution output or printing. You must first use Import Library → PDF to make use of the library. More information can be found in the PDF library reference.

As of Processing 3.0, to use variables as the parameters to size() function, place the size() function within the settings() function (instead of setup()). There is more information about this on the settings() reference page.

#### Syntax

**size(width, height)**  
**size(width, height, renderer)**

#### Parameters

width int: width of the display window in units of pixels

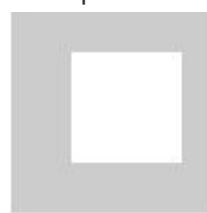
height int: height of the display window in units of pixels

Returns void

#### Name

**noStroke()**

#### Examples



```
noStroke();  
rect(30, 20, 55, 55);
```

Description Disables drawing the stroke (outline). If both noStroke() and noFill() are called, nothing will be drawn to the screen.

## Syntax

**noStroke()**

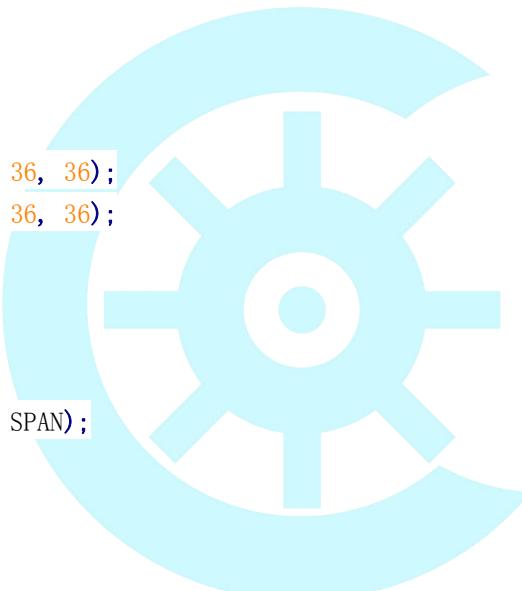
Returns void

## Name

**smooth()**

## Examples

```
void setup() {  
    size(100, 100);  
    smooth(2);  
    noStroke();  
}  
  
void draw() {  
    background(0);  
    ellipse(30, 48, 36, 36);  
    ellipse(70, 48, 36, 36);  
}  
  
void setup() {  
    fullScreen(P2D, SPAN);  
    smooth(4);  
}  
  
void draw() {  
    background(0);  
    ellipse(x, height/2, height/4, height/4);  
    x += 1;  
}
```



**Description** Draws all geometry with smooth (anti-aliased) edges. This behavior is the default, so smooth() only needs to be used when a program needs to set the smoothing in a different way. The level parameter increases the level of smoothness. This is the level of over sampling applied to the graphics buffer.

With the P2D and P3D renderers, smooth(2) is the default, this is called "2x anti-aliasing." The code smooth(4) is used for 4x anti-aliasing and smooth(8) is specified for 8x anti-aliasing. The maximum anti-aliasing level is determined by the hardware of the machine that is running the software, so smooth(4) and smooth(8) will not work with every computer.

The default renderer uses smooth(3) by default. This is bicubic smoothing. The other option for the default renderer is smooth(2), which is bilinear smoothing.

With Processing 3.0, smooth() is different than before. It was common to use smooth() and noSmooth() to turn on and off antialiasing within a sketch. Now, because of how the

software has changed, smooth() can only be set once within a sketch. It can be used either at the top of a sketch without a setup(), or after the size() function when used in a sketch with setup(). The noSmooth() function also follows the same rules.

Syntax

**smooth(level)**

Parameters

level int: either 2, 3, 4, or 8 depending on the renderer

Returns void

Name

**loadImage()**

Examples



```
PIImage img;  
img = loadImage("laDefense.jpg");  
image(img, 0, 0);
```



```
PIImage img;  
void setup() {  
    img = loadImage("laDefense.jpg");  
}  
void draw() {  
    image(img, 0, 0);  
}
```



```
PIImage webImg;  
void setup() {  
    String url = "https://processing.org/img/processing-web.png";  
    // Load image from a web server  
    webImg = loadImage(url, "png");
```



deept

```
}
```

```
void draw() {
```

```
    background(0);
```

```
    image(webImg, 0, 0);
```

```
}
```

Description Loads an image into a variable of type PImage. Four types of images (.gif, .jpg, .tga, .png) images may be loaded. To load correctly, images must be located in the data directory of the current sketch.

In most cases, load all images in setup() to preload them at the start of the program. Loading images inside draw() will reduce the speed of a program. Images cannot be loaded outside setup() unless they're inside a function that's called after setup() has already run.

Alternatively, the file maybe be loaded from anywhere on the local computer using an absolute path (something that starts with / on Unix and Linux, or a drive letter on Windows), or the filename parameter can be a URL for a file found on a network.

If the file is not available or an error occurs, null will be returned and an error message will be printed to the console. The error message does not halt the program, however the null value may cause a NullPointerException if your code does not check whether the value returned is null.

The extension parameter is used to determine the image type in cases where the image filename does not end with a proper extension. Specify the extension as the second parameter to loadImage(), as shown in the third example on this page. Note that CMYK images are not supported.

Depending on the type of error, a PImage object may still be returned, but the width and height of the image will be set to -1. This happens if bad image data is returned or cannot be decoded properly. Sometimes this happens with image URLs that produce a 403 error or that redirect to a password prompt, because loadImage() will attempt to interpret the HTML as image data.

Syntax

**loadImage(filename)**

**loadImage(filename, extension)**

Parameters

filename String: name of file to load, can be .gif, .jpg, .tga, or a handful of other image types depending on your platform

extension String: type of image to load, for example "png", "gif", "jpg"

Returns PImage

Name

**createFont()**

## Examples

```
PFont myFont;  
void setup() {  
    size(200, 200);  
    // Uncomment the following two lines to see the available fonts  
    //String[] fontList = PFont.list();  
    //printArray(fontList);  
    myFont = createFont("Georgia", 32);  
    textAlign(CENTER, CENTER);  
    text("!@#$%", width/2, height/2);  
}
```

**Description** Dynamically converts a font to the format used by Processing from a .ttf or .otf file inside the sketch's "data" folder or a font that's installed elsewhere on the computer. If you want to use a font installed on your computer, use the PFont.list() method to first determine the names for the fonts recognized by the computer and are compatible with this function. Not all fonts can be used and some might work with one operating system and not others. When sharing a sketch with other people or posting it on the web, you may need to include a .ttf or .otf version of your font in the data directory of the sketch because other people might not have the font installed on their computer. Only fonts that can legally be distributed should be included with a sketch.

The size parameter states the font size you want to generate. The smooth parameter specifies if the font should be antialiased or not. The charset parameter is an array of chars that specifies the characters to generate.

This function allows Processing to work with the font natively in the default renderer, so the letters are defined by vector geometry and are rendered quickly. In the P2D and P3D renderers, the function sets the project to render the font as a series of small textures. For instance, when using the default renderer, the actual native version of the font will be employed by the sketch, improving drawing quality and performance. With the P2D and P3D renderers, the bitmapped version will be used to improve speed and appearance, but the results are poor when exporting if the sketch does not include the .otf or .ttf file, and the requested font is not available on the machine running the sketch.

## Syntax

```
createFont(name, size)  
createFont(name, size, smooth)  
createFont(name, size, smooth, charset)
```

## Parameters

**name** String: name of the font to load

**size** float: point size of the font

**smooth** boolean: true for an antialiased font, false for aliased

**charset** char[]: array containing characters to be generated

**Returns** PFont

Name

**background()**

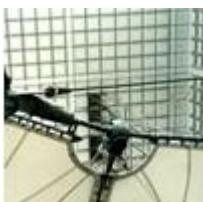
Examples



`background(51);`



`background(255, 204, 0);`



```
PIImage img;  
img = loadImage("laDefense.jpg");  
background(img);
```

Description The `background()` function sets the color used for the background of the Processing window. The default background is light gray. This function is typically used within `draw()` to clear the display window at the beginning of each frame, but it can be used inside `setup()` to set the background on the first frame of animation or if the background need only be set once.

An image can also be used as the background for a sketch, although the image's width and height must match that of the sketch window. Images used with `background()` will ignore the current `tint()` setting. To resize an image to the size of the sketch window, use `image.resize(width, height)`.

It is not possible to use the transparency alpha parameter with background colors on the main drawing surface. It can only be used along with a PGraphics object and `createGraphics()`.

Syntax

**background(rgb)**

**background(rgb, alpha)**

**background(gray)**

**background(gray, alpha)**  
**background(v1, v2, v3)**  
**background(v1, v2, v3, alpha)**  
**background(image)**

Parameters

rgb int: any value of the color datatype

alpha float: opacity of the background

grayfloat: specifies a value between white and black

v1 float: red or hue value (depending on the current color mode)

v2 float: green or saturation value (depending on the current color mode)

v3 float: blue or brightness value (depending on the current color mode)

image PImage: PImage to set as background (must be same size as the sketch window)

Returns void

Name

**image()**

Examples



```
PIImage img;  
void setup() {  
    // Images must be in the "data" directory to load correctly  
    img = loadImage("laDefense.jpg");  
}  
void draw() {
```

```
    image(img, 0, 0);  
}
```



```
PIImage img;  
void setup() {  
    // Images must be in the "data" directory to load correctly  
    img = loadImage("laDefense.jpg");  
}  
void draw() {
```

```
    image(img, 0, 0);
    image(img, 0, 0, width/2, height/2);
}
```

Description The `image()` function draws an image to the display window. Images must be in the sketch's "data" directory to load correctly. Select "Add file..." from the "Sketch" menu to add the image to the data directory, or just drag the image file onto the sketch window. Processing currently works with GIF, JPEG, and PNG images.

The `img` parameter specifies the image to display and by default the `a` and `b` parameters define the location of its upper-left corner. The image is displayed at its original size unless the `c` and `d` parameters specify a different size. The `imageMode()` function can be used to change the way these parameters draw the image.

The color of an image may be modified with the `tint()` function. This function will maintain transparency for GIF and PNG images.

Syntax

```
image(img, a, b)
image(img, a, b, c, d)
```

Parameters

`img` PImage: the image to display  
`a` float: x-coordinate of the image  
`b` float: y-coordinate of the image  
`c` float: width to display the image  
`d` float: height to display the image

Returns void

Name

**resize()**

Examples



```
PImage jelly = loadImage("jelly.jpg");
image(jelly, 0, 0);
jelly.resize(100, 50);
image(jelly, 0, 0);
```



```
PImage jelly = loadImage("jelly.jpg");
image(jelly, 0, 0);
jelly.resize(0, 50);
image(jelly, 0, 0);
```

Description Resize the image to a new width and height. To make the image scale proportionally, use 0 as the value for the wide or high parameter. For instance, to make the width of an image 150 pixels, and change the height using the same proportion, use `resize(150, 0)`.

Even though a PGraphics is technically a PImage, it is not possible to rescale the image data found in a PGraphics. (It's simply not possible to do this consistently across renderers: technically infeasible with P3D, or what would it even do with PDF?) If you want to resize PGraphics content, first get a copy of its image data using the `get()` method, and call `resize()` on the PImage that is returned.

Syntax

**pimg.resize(w, h)**

Parameters

pimg PImage: any object of type PImage

w int: the resized image width

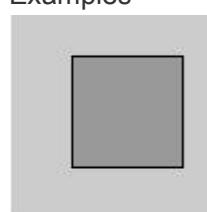
h int: the resized image height

Returns void

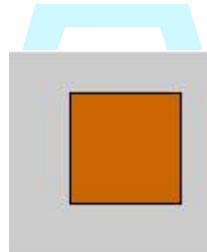
Name

**fill()**

Examples



```
fill(153);
rect(30, 20, 55, 55);
```



```
fill(204, 102, 0);
rect(30, 20, 55, 55);
```

Description Sets the color used to fill shapes. For example, if you run `fill(204, 102, 0)`, all subsequent shapes will be filled with orange. This color is either specified in terms of the RGB or HSB color depending on the current `colorMode()`. The default color space is RGB, with each value in the range from 0 to 255.

When using hexadecimal notation to specify a color, use "#" or "0x" before the values (e.g., #CCFFAA or 0xFFCCFFAA). The # syntax uses six digits to specify a color (just as colors are typically specified in HTML and CSS). When using the hexadecimal notation starting with "0x", the hexadecimal value must be specified with eight characters; the first two characters define the alpha component, and the remainder define the red, green, and blue components.

The value for the "gray" parameter must be less than or equal to the current maximum value as specified by colorMode(). The default maximum value is 255.

To change the color of an image or a texture, use tint().

Syntax

**fill(rgb)**

**fill(rgb, alpha)**

**fill(gray)**

**fill(gray, alpha)**

**fill(v1, v2, v3)**

**fill(v1, v2, v3, alpha)**

Parameters

rgb int: color variable or hex value

alpha float: opacity of the fill

grayfloat: number specifying value between white and black

v1 float: red or hue value (depending on current color mode)

v2 float: green or saturation value (depending on current color mode)

v3 float: blue or brightness value (depending on current color mode)

Returns void

Name

**textFont()**

Examples



a deept

PFont mono;

```
// The font "andalemono.ttf" must be located in the  
// current sketch's "data" directory to load successfully
```

```
mono = loadFont("andalemono.ttf", 32);
```

```
background(0);
```

```
textFont(mono);
```

```
text("word", 12, 60);
```

Description Sets the current font that will be drawn with the text() function. Fonts must

be created for Processing with `createFont()` or loaded with `loadFont()` before they can be used. The font set through `textFont()` will be used in all subsequent calls to the `text()` function. If no size parameter is specified, the font size defaults to the original size (the size in which it was created with the "Create Font..." tool) overriding any previous calls to `textFont()` or `textSize()`.

When fonts are rendered as an image texture (as is the case with the P2D and P3D renderers as well as with `loadFont()` and `vlw` files), you should create fonts at the sizes that will be used most commonly. Using `textFont()` without the size parameter will result in the cleanest type.

#### Syntax

**`textFont(which)`**  
**`textFont(which, size)`**

#### Parameters

`which` PFont: any variable of the type PFont  
`size` float: the size of the letters in units of pixels

Returns void

#### Name

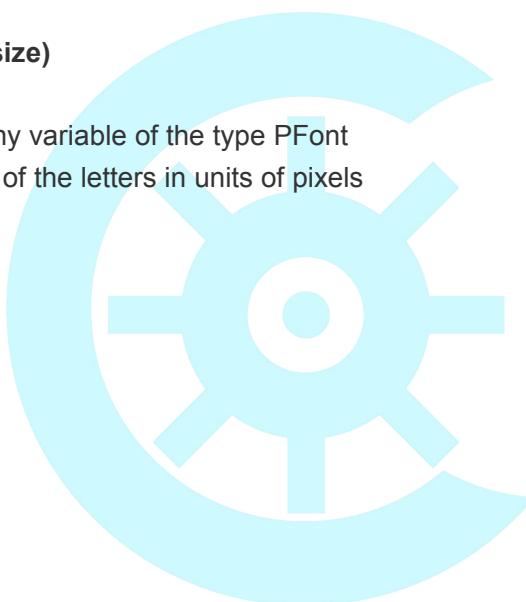
**`text()`**

#### Examples



```
textSize(32);
text("word", 10, 30);
fill(0, 102, 153);
text("word", 10, 60);
fill(0, 102, 153, 51);
text("word", 10, 90);

size(100, 100, P3D);
textSize(32);
fill(0, 102, 153, 204);
text("word", 12, 45, -30); // Specify a z-axis value
text("word", 12, 60); // Default depth, no z-value specified
```



```
The quick  
brown fox  
jumped  
over the  
lazy dog.
```

```
String s = "The quick brown fox jumped over the lazy dog.";  
fill(50);  
text(s, 10, 10, 70, 80); // Text wraps within text box
```

Description Draws text to the screen. Displays the information specified in the first parameter on the screen in the position specified by the additional parameters. A default font will be used unless a font is set with the `textFont()` function and a default size will be used unless a font is set with `textSize()`. Change the color of the text with the `fill()` function. The text displays in relation to the `textAlign()` function, which gives the option to draw to the left, right, and center of the coordinates.

The `x2` and `y2` parameters define a rectangular area to display within and may only be used with string data. When these parameters are specified, they are interpreted based on the current `rectMode()` setting. Text that does not fit completely within the rectangle specified will not be drawn to the screen.

Note that Processing now lets you call `text()` without first specifying a PFont with `textFont()`. In that case, a generic sans-serif font will be used instead. (See the third example above.)

#### Syntax

```
text(c, x, y)  
text(c, x, y, z)  
text(str, x, y)  
text(chars, start, stop, x, y)  
text(str, x, y, z)  
text(chars, start, stop, x, y, z)  
text(str, x1, y1, x2, y2)  
text(num, x, y)  
text(num, x, y, z)
```

#### Parameters

c char: the alphanumeric character to be displayed  
x float: x-coordinate of text  
y float: y-coordinate of text  
z float: z-coordinate of text  
chars char[]: the alphanumeric symbols to be displayed  
startint: array index at which to start writing characters  
stop int: array index at which to stop writing characters  
x1 float: by default, the x-coordinate of text, see `rectMode()` for more info  
y1 float: by default, the y-coordinate of text, see `rectMode()` for more info  
x2 float: by default, the width of the text box, see `rectMode()` for more info  
y2 float: by default, the height of the text box, see `rectMode()` for more info

numint, or float: the numeric value to be displayed

Returns void

Name

**mouseX**

Examples

```
void draw() {  
    background(204);  
    line(mouseX, 20, mouseX, 80);  
}
```

Description The system variable `mouseX` always contains the current horizontal coordinate of the mouse.

Note that Processing can only track the mouse position when the pointer is over the current window. The default value of `mouseX` is 0, so 0 will be returned until the mouse moves in front of the sketch window. (This typically happens when a sketch is first run.) Once the mouse moves away from the window, `mouseX` will continue to report its most recent position.

Name

**mouseY**

Examples

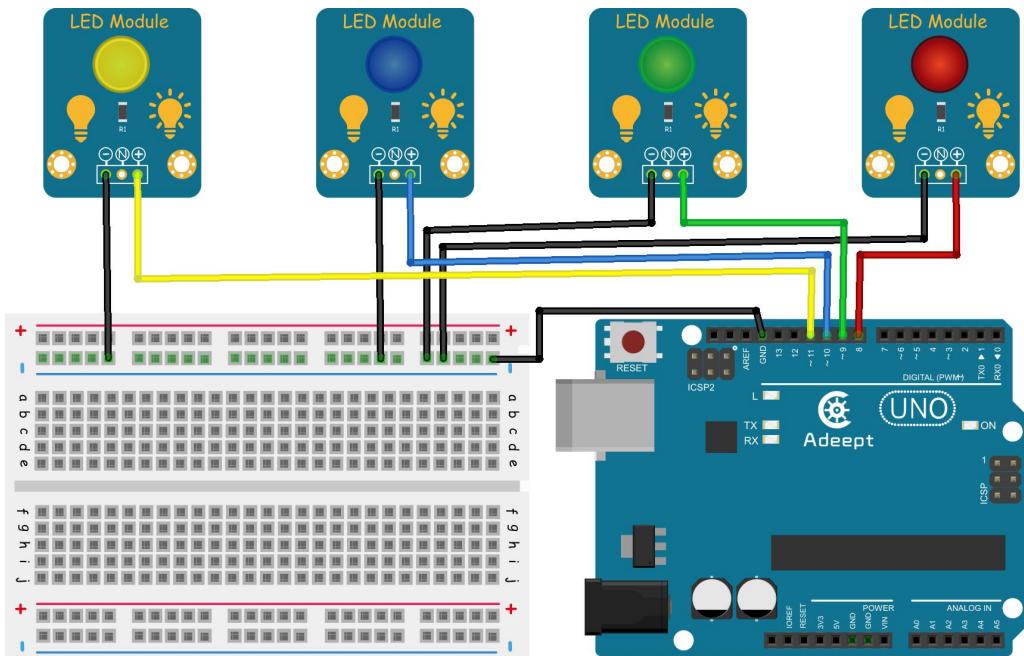
```
void draw() {  
    background(204);  
    line(20, mouseY, 80, mouseY);  
}
```

Description The system variable `mouseY` always contains the current vertical coordinate of the mouse.

Note that Processing can only track the mouse position when the pointer is over the current window. The default value of `mouseY` is 0, so 0 will be returned until the mouse moves in front of the sketch window. (This typically happens when a sketch is first run.) Once the mouse moves away from the window, `mouseY` will continue to report its most recent position.

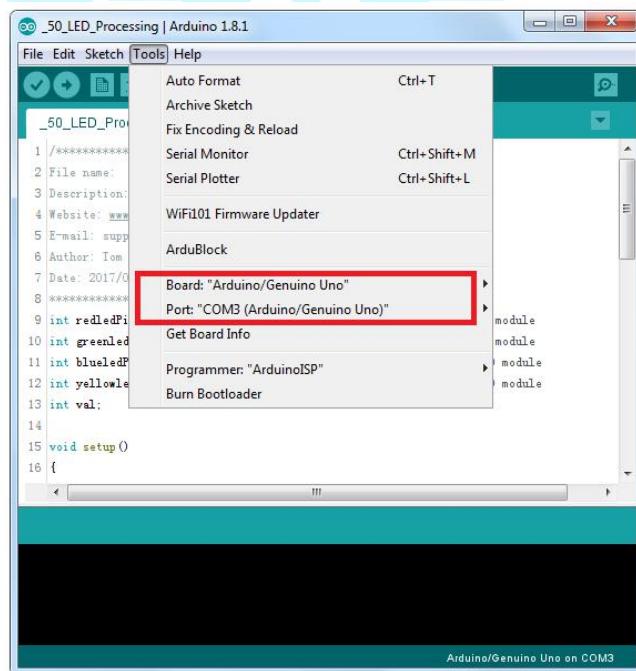
## Experimental Procedures

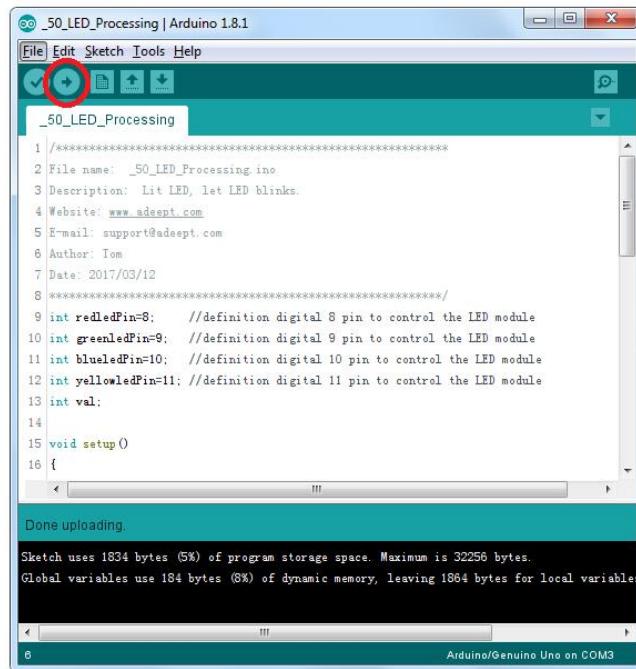
**Step 1:** Build the circuit



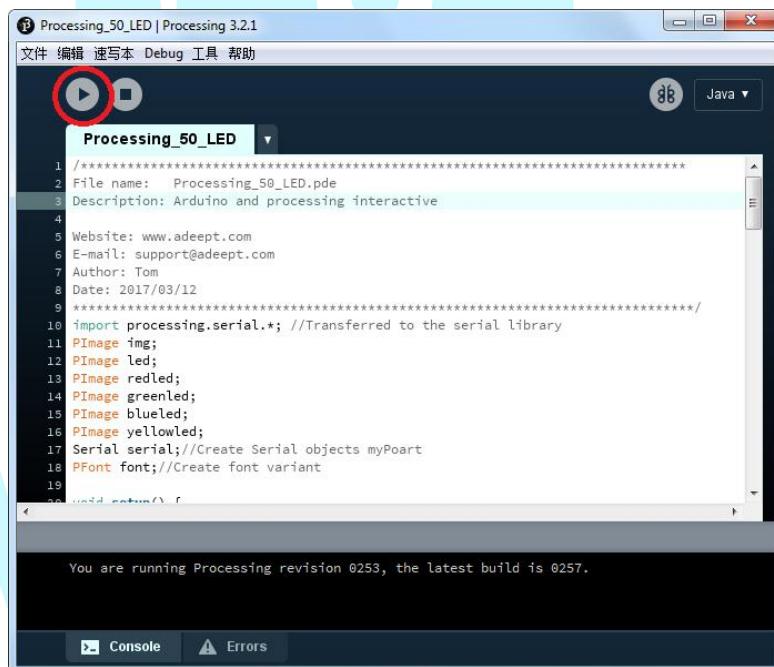
**Step 2:** Program `_50_LED_Processing.ino`

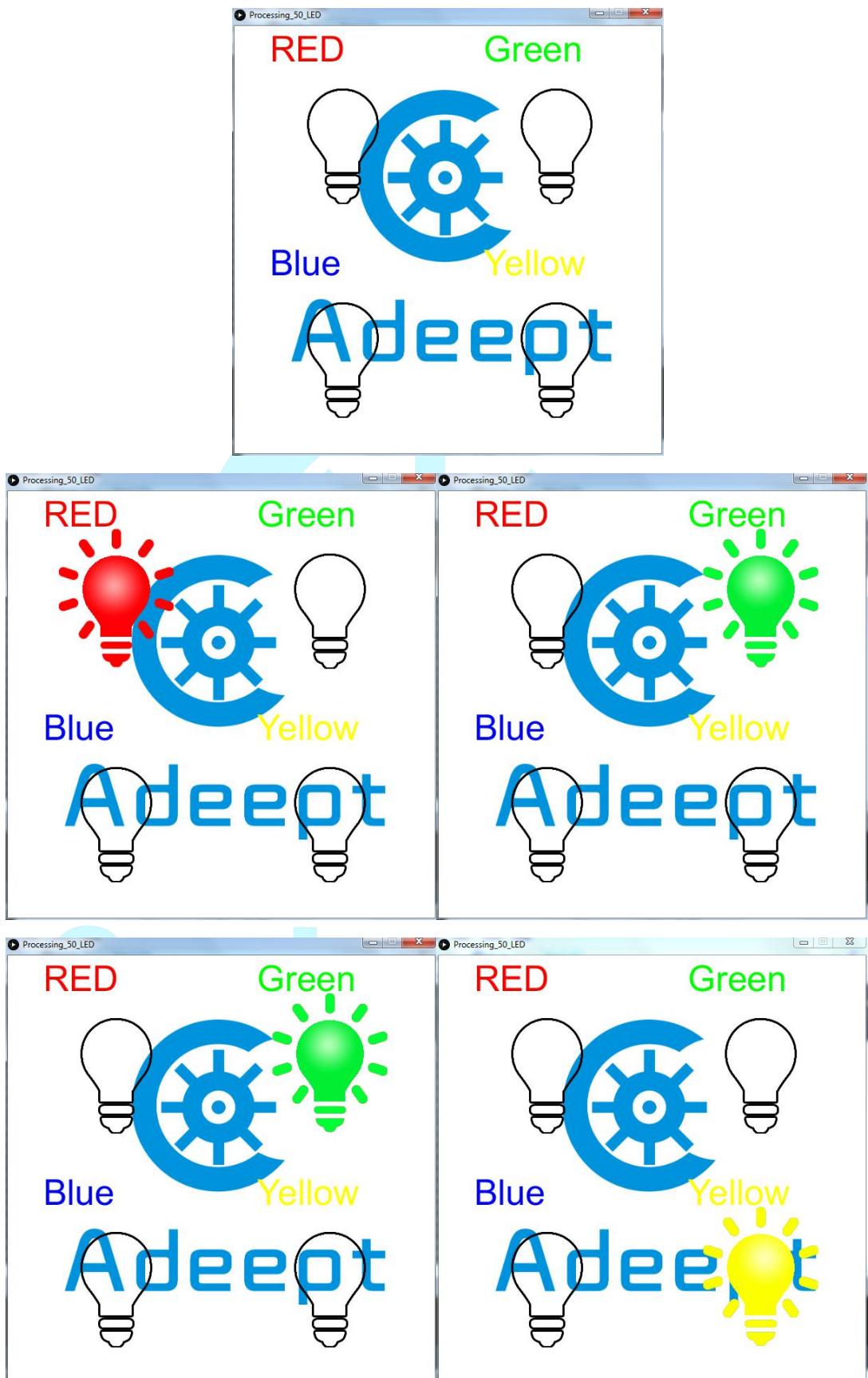
**Step 3:** Compile and download the sketch to the UNO R3 board.

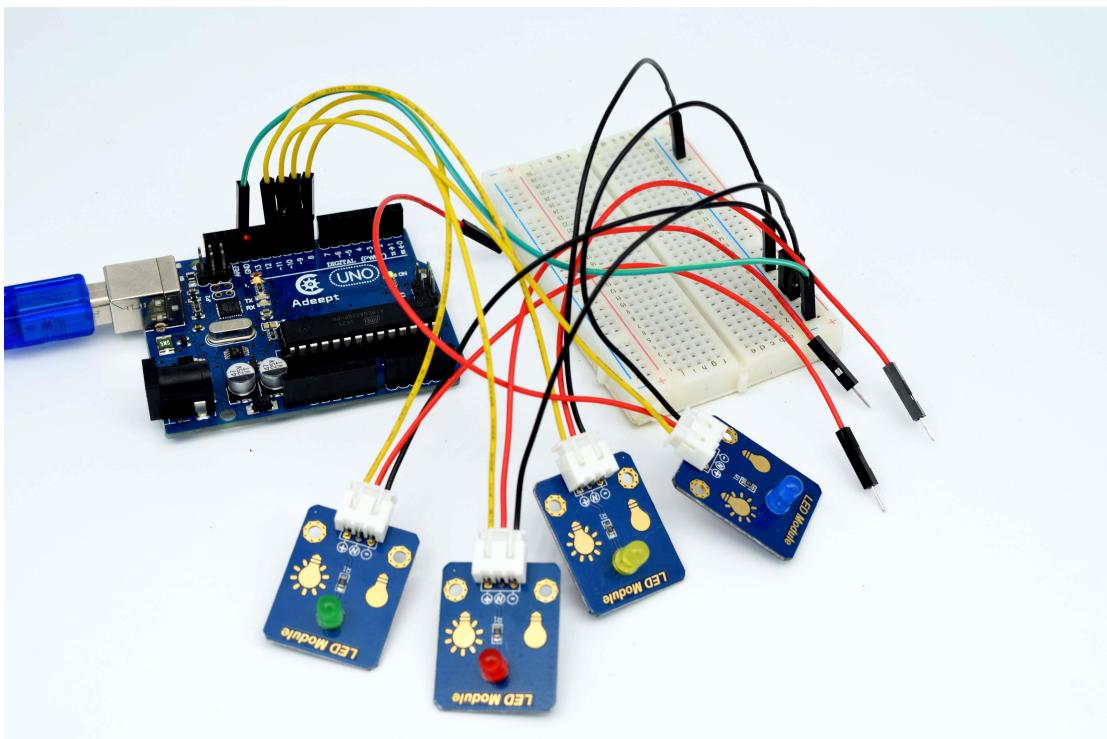




#### Step 4: Run the Processing software (Processing\_50\_LED.pde)







# Adeept

# Lesson 51 Arduino Interacts with Processing(Button Module)

## Introduction

In this lesson, we will create a interface based on Processing, which contains four different colors of virtual buttons. When you press the solid button connected to the Arduino, the corresponding button in the Processing interface will be pressed.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 4 \* Button Module
- 1 \* USB Cable
- 4 \* 3-Pin Wires
- 1 \* Breadboard
- 2 \* Hookup Wire Set

## Principle

Processing key function:

Name

**println()**

Examples

```
String s = "The size is ";
int w = 1920;
int h = 1080;
println(s);
println(w, "x", h);

// This program writes to the console:
// The size is
// 1920 x 1080
print("begin- ");
float f = 0.3;
int i = 1024;
print("f is " + f + " and i is " + 1024);
String s = "-end";
println(s);

// This program writes to the console:
// "begin- f is 0.3 and i is 1024 -end"
```



Adeept

Description The `println()` function writes to the console area, the black rectangle at the bottom of the Processing environment. This function is often helpful for looking at the data a program is producing. Each call to this function creates a new line of output. More than one parameter can be passed into the function by separating them with commas. Alternatively, individual elements can be separated with quotes ("") and joined with the addition operator (+). Before Processing 2.1, `println()` was used to write array data to the console. Now, use `printArray()` to write array data to the console.

Note that the console is relatively slow. It works well for occasional messages, but does not support high-speed, real-time output (such as at 60 frames per second). It should also be noted, that a `println()` within a `for` loop can sometimes lock up the program, and cause the sketch to freeze.

Syntax

**`println()`**

**`println(what)`**

**`println(variables)`**

Parameters

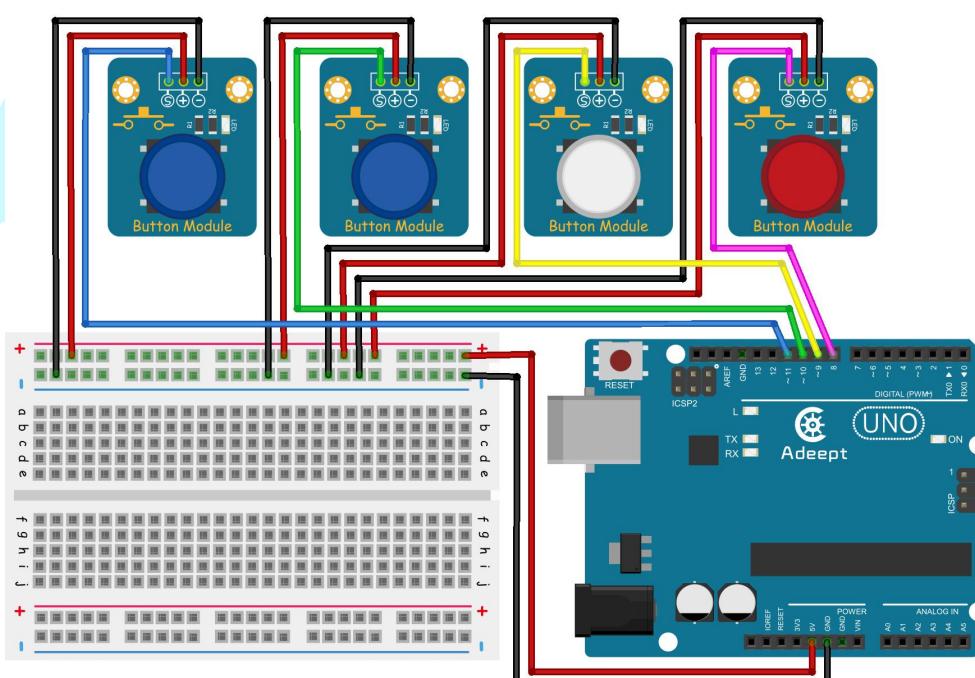
`what` Object, String, float, char, boolean, or byte: data to print to console

`variables` Object[]: list of data, separated by commas

Returns void

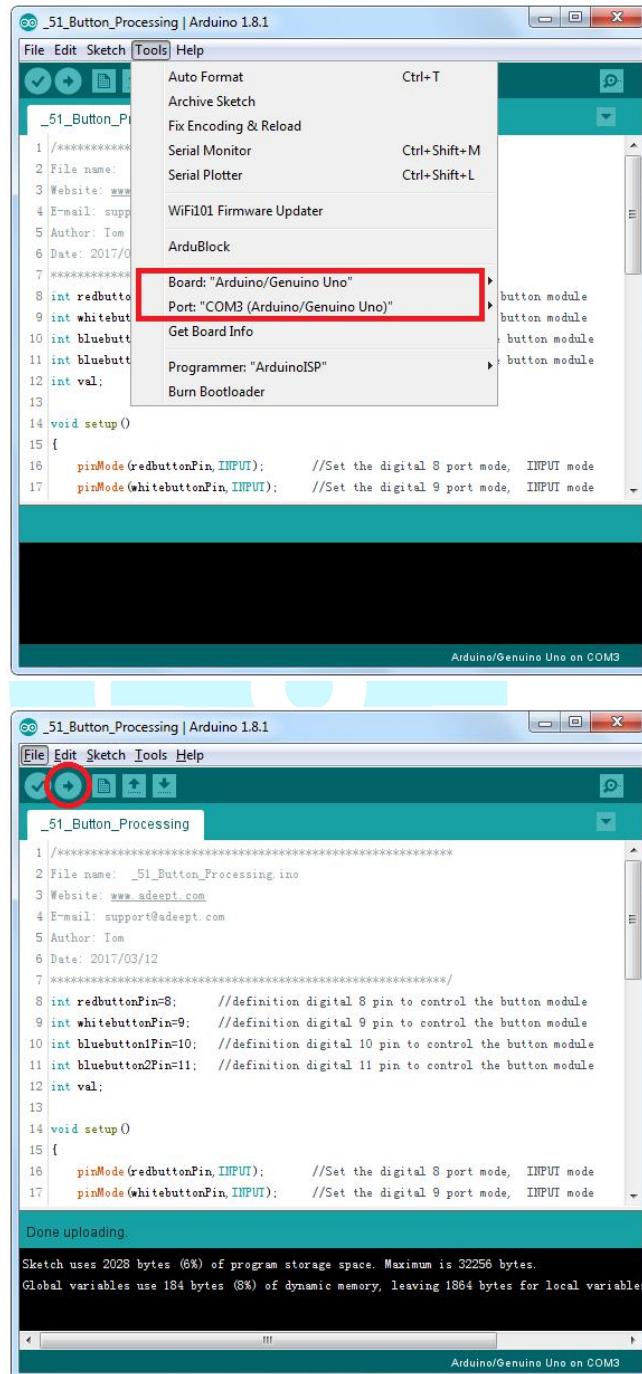
## Experimental Procedures

### Step 1: Build the circuit

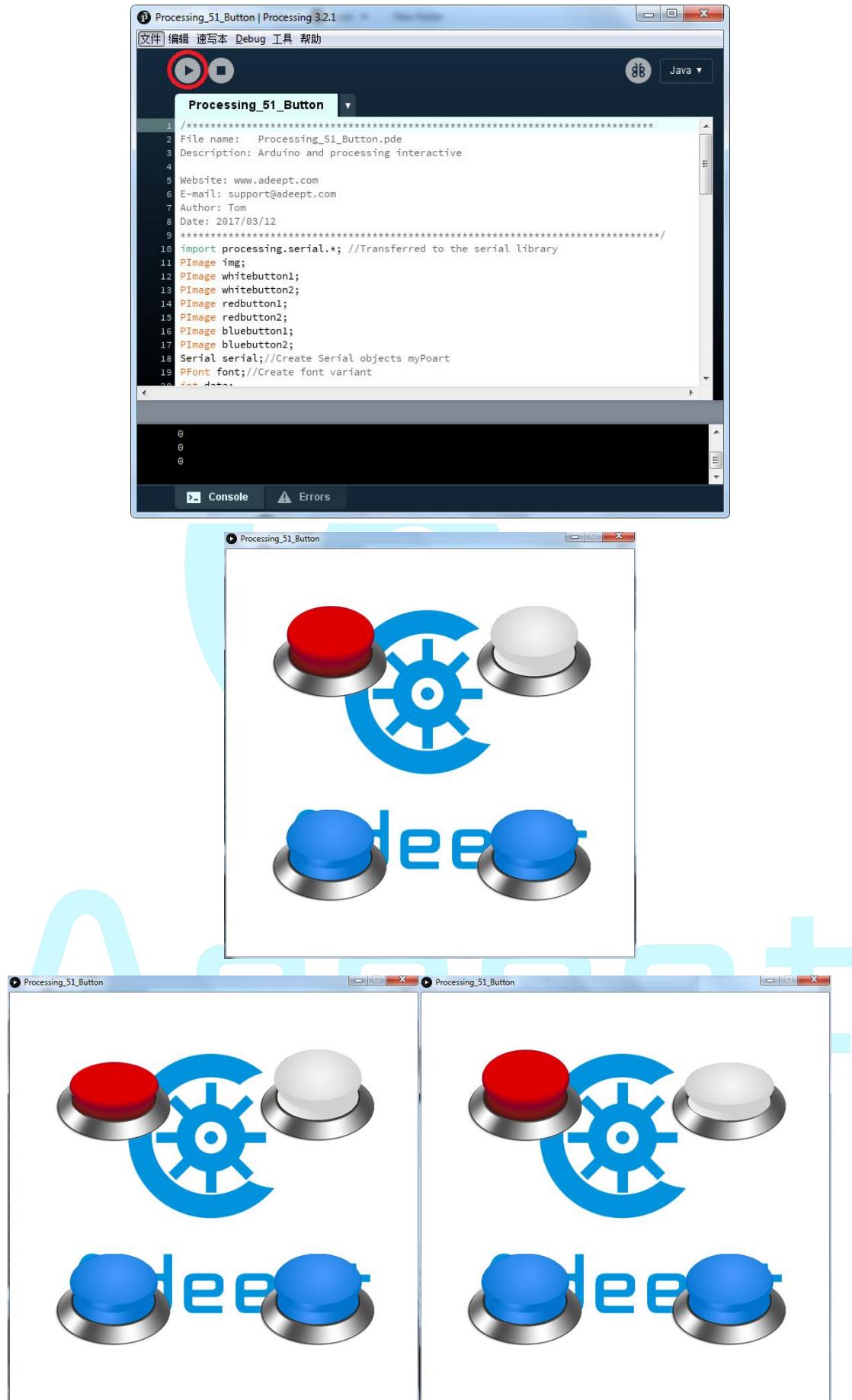


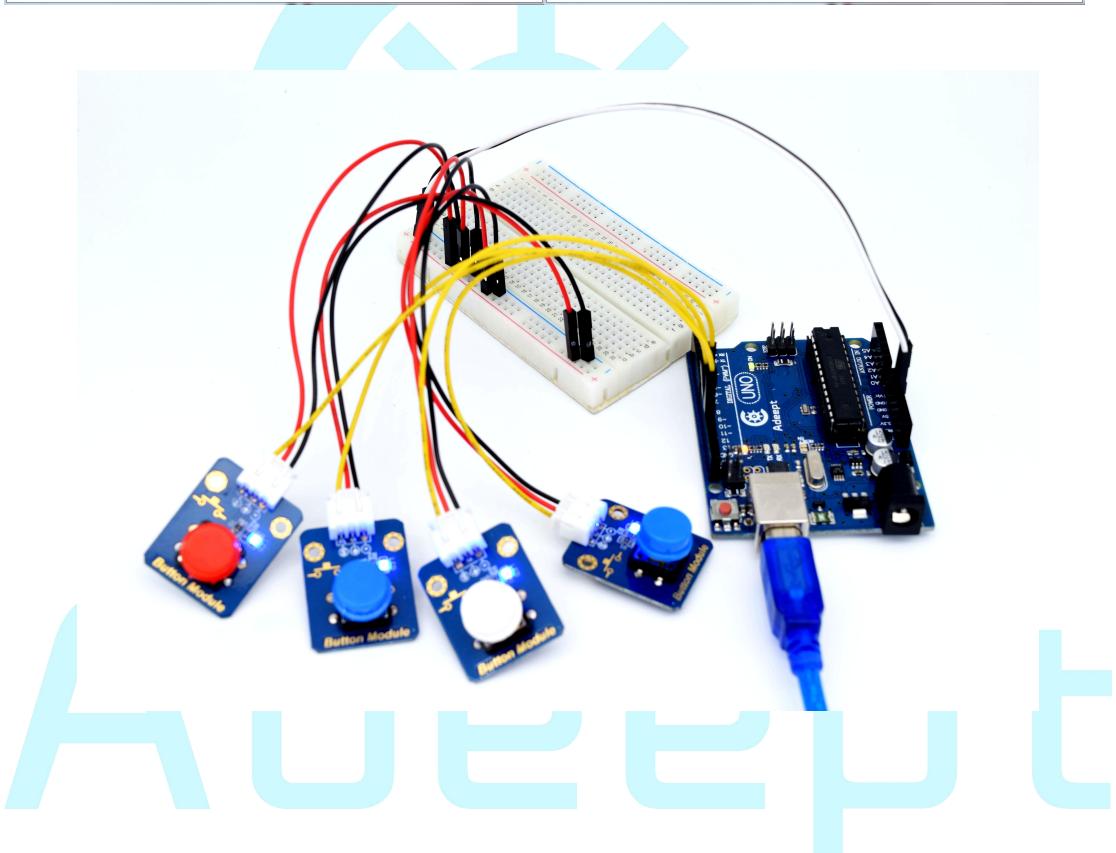
### Step 2: Program \_51\_Button\_Processing.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



**Step 4:** Run the Processing software (Processing\_51\_Button.pde)





# Lesson 52 Arduino Interacts with Processing(RGB Module)

## Introduction

In this lesson, we will create a simple interface based on Processing, which contains three different colors of the circle, when the mouse pointer points to a certain color circle, the RGB LED connected to the Arduino will emit the corresponding color of light, when the mouse pointer to the white background, the RGB LED will emit white light.

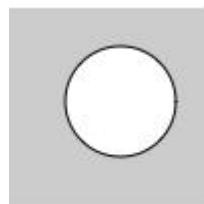
## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* RGB Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

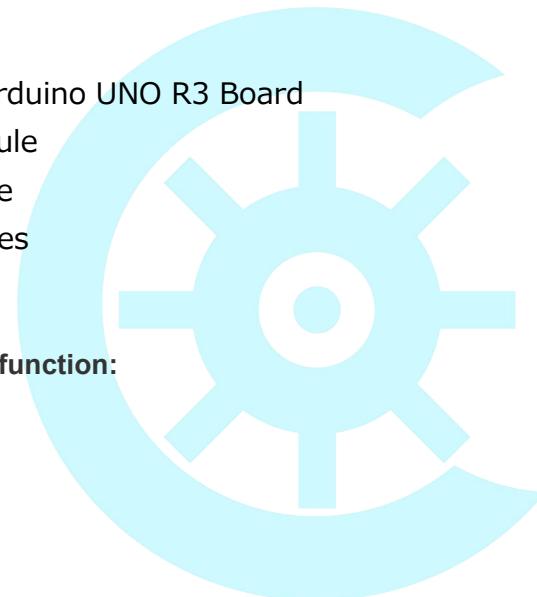
## Principle

Processing key function:

Name  
**ellipse()**



`ellipse(56, 46, 55, 55);`



Draws an ellipse (oval) to the screen. An ellipse with equal width and height is a circle. By default, the first two parameters set the location, and the third and fourth parameters set the shape's width and height. The origin may be changed with the `ellipseMode()` function.

Syntax

`ellipse(a, b, c, d)`

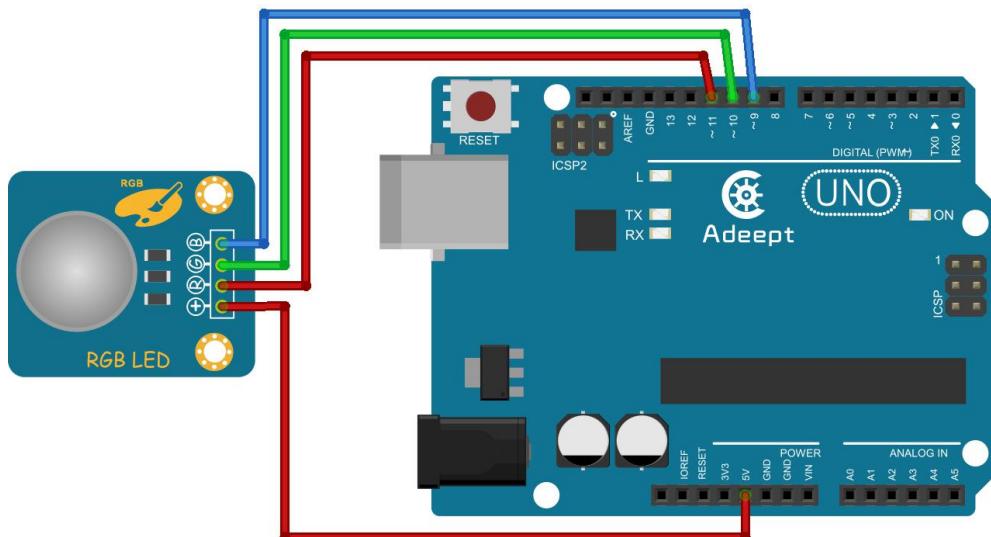
Parameters

- a float: x-coordinate of the ellipse
- b float: y-coordinate of the ellipse
- c float: width of the ellipse by default
- d float: height of the ellipse by default

Returns void

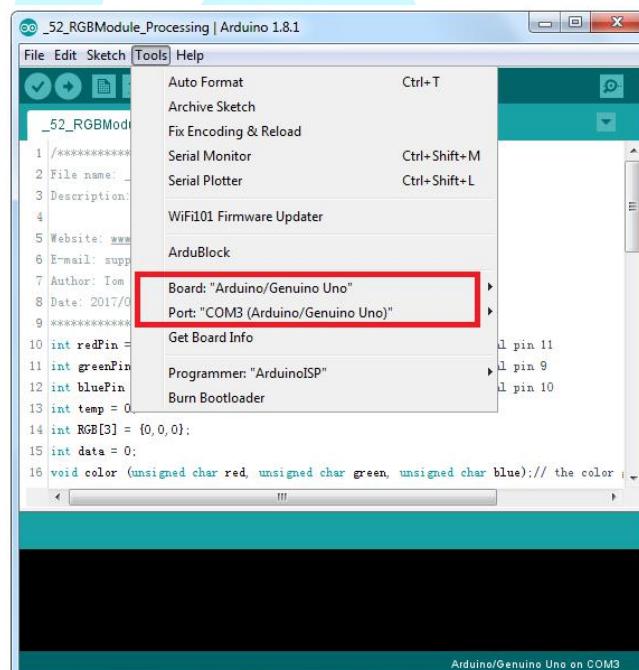
## Experimental Procedures

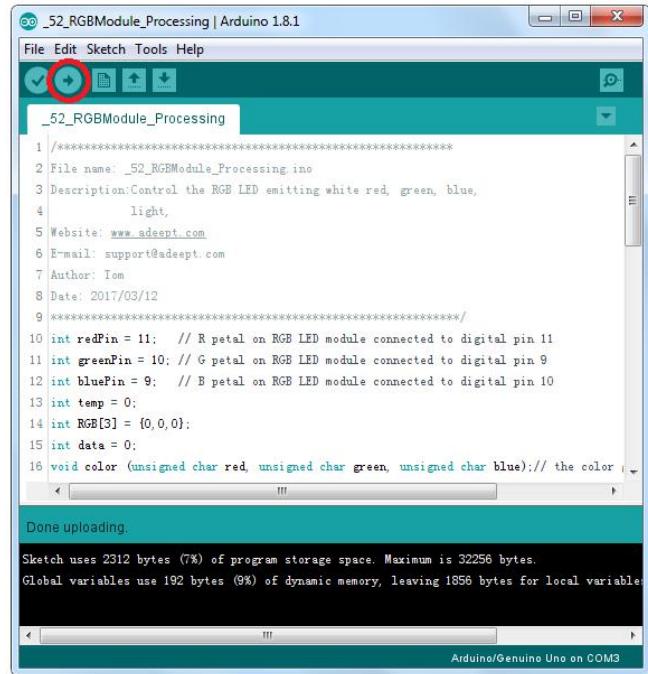
**Step 1:** Build the circuit



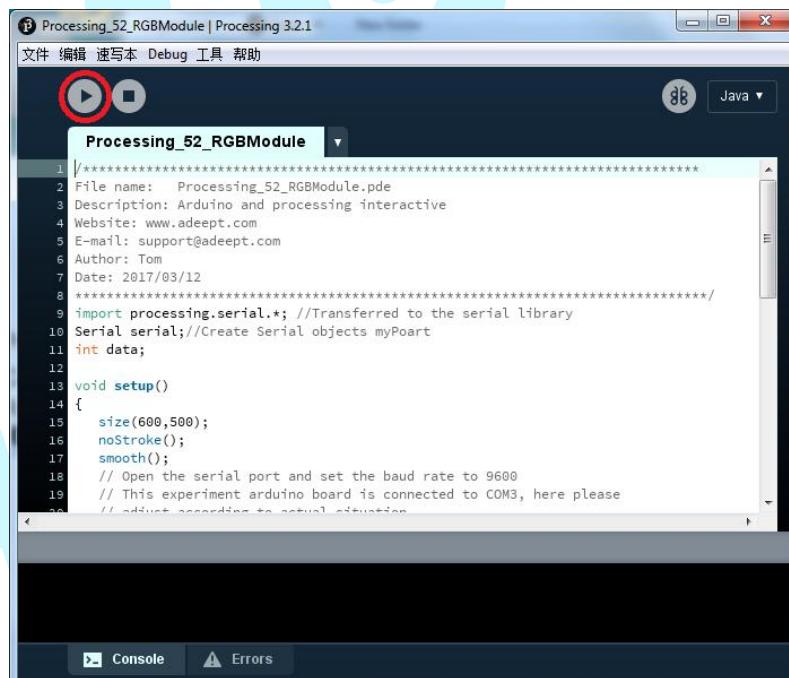
**Step 2:** Program \_52\_RGBModule\_Processing.ino

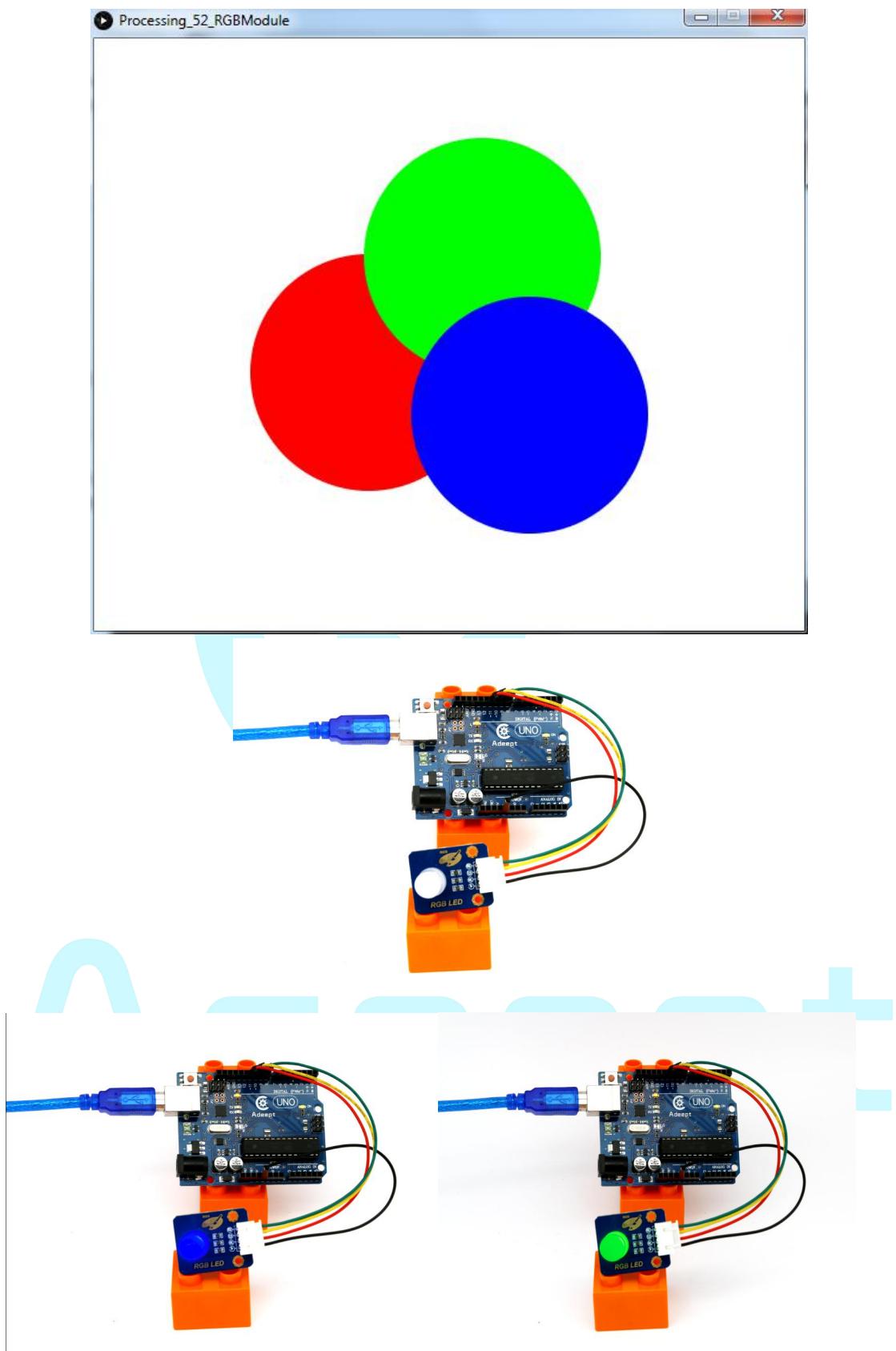
**Step 3:** Compile and download the sketch to the UNO R3 board.

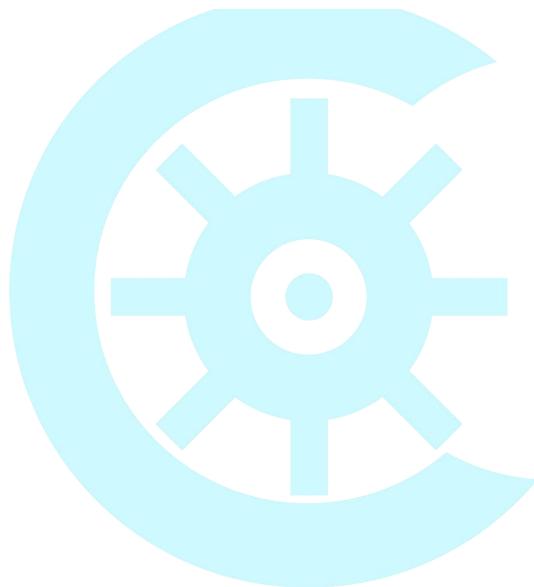
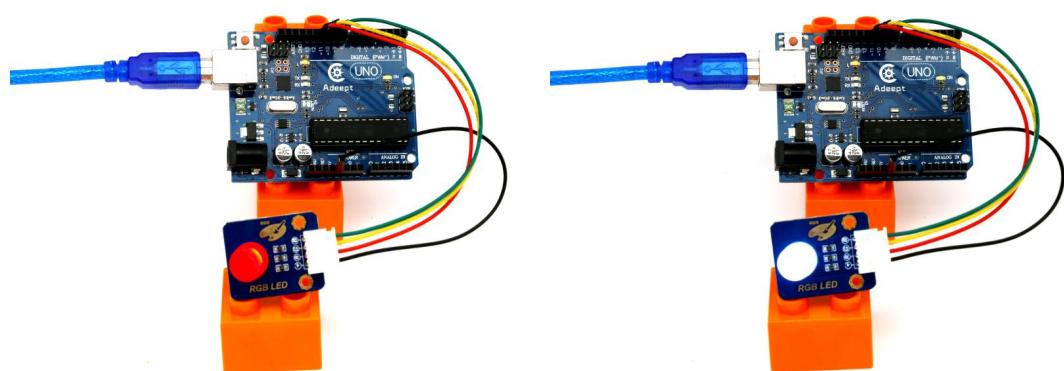




#### Step 4: Run the Processing software (Processing\_52\_RGBModule.pde)







# Adeept

# Lesson 53 Arduino Interacts with Processing(Potentiometer)

## Introduction

In this lesson, we will create a simple interface based on Processing, which contains a spiral circle, when rotate the potentiometer knob of the potentiometer module connected to the UNO, the diameter of the circle will be changed.

## Components

- 1 \* Adept Arduino UNO R3 Board
- 1 \* potentiometer Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires
- 1 \* Breadboard
- 2 \* Hookup Wire Set

## Principle

**Processing key function:**

Name: `map()`

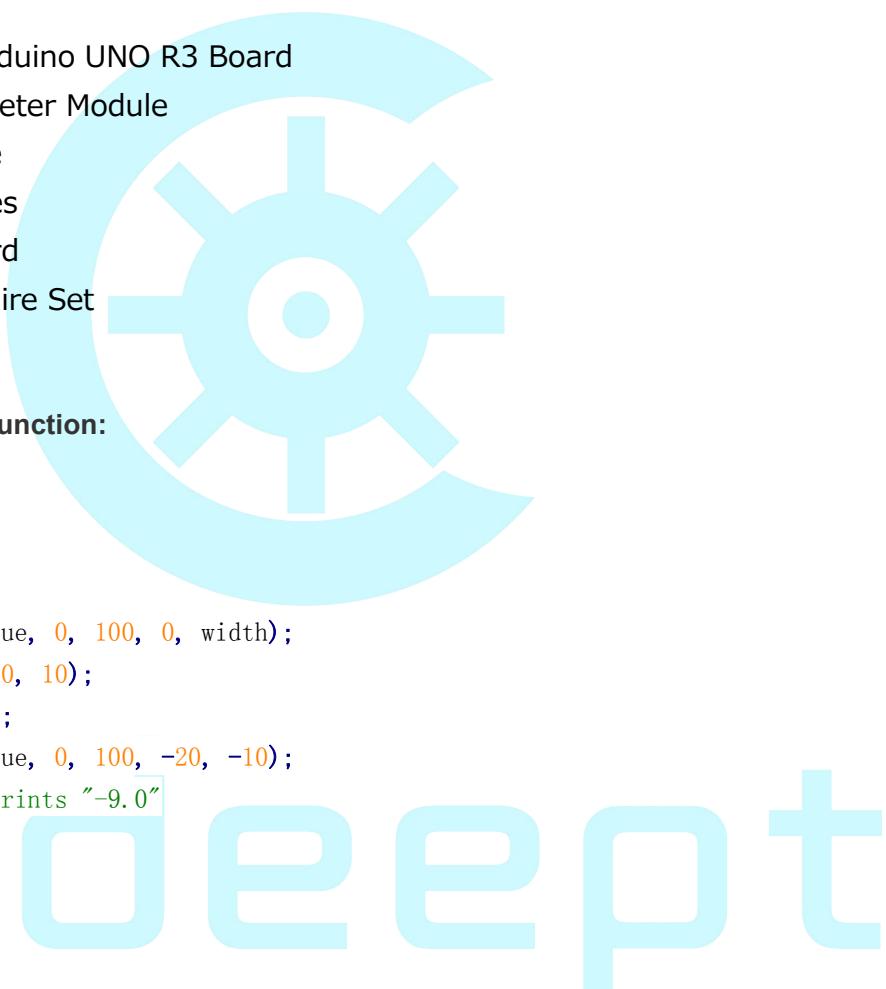
Examples

```
size(200, 200);
float value = 25;
float m = map(value, 0, 100, 0, width);
ellipse(m, 200, 10, 10);
float value = 110;
float m = map(value, 0, 100, -20, -10);
println(m); // Prints "-9.0"
void setup() {
    size(200, 200);
    noStroke();
}
```

```
void draw() {
    background(204);
    float x1 = map(mouseX, 0, width, 50, 150);
    ellipse(x1, 75, 50, 50);
    float x2 = map(mouseX, 0, width, 0, 200);
    ellipse(x2, 125, 50, 50);
}
```

Description:

Re-maps a number from one range to another.



In the first example above, the number 25 is converted from a value in the range of 0 to 100 into a value that ranges from the left edge of the window (0) to the right edge (width). As shown in the second example, numbers outside of the range are not clamped to the minimum and maximum parameters values, because out-of-range values are often intentional and useful.

Syntax: **map(value, start1, stop1, start2, stop2)**

Parameters:

value float: the incoming value to be converted  
start1 float: lower bound of the value's current range  
stop1 float: upper bound of the value's current range  
start2 float: lower bound of the value's target range  
stop2 float: upper bound of the value's target range

Returns: float

Name: **radians()**

Examples

```
float deg = 45.0;  
float rad = radians(deg);  
println(deg + " degrees is " + rad + " radians");
```

Description: Converts a degree measurement to its corresponding value in radians. Radians and degrees are two ways of measuring the same thing. There are 360 degrees in a circle and  $2\pi$  radians in a circle. For example,  $90^\circ = \pi/2 = 1.5707964$ . All trigonometric functions in Processing require their parameters to be specified in radians.

Syntax: **radians(degrees)**

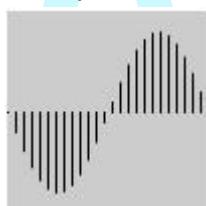
Parameters:

degrees float: degree value to convert to radians

Returns: float

Name: **sin()**

Examples



```
float a = 0.0;  
float inc = TWO_PI/25.0;  
for (int i = 0; i < 100; i+=4) {  
    line(i, 50, i, 50+sin(a)*40.0);  
    a = a + inc;  
}
```

Description: Calculates the sine of an angle. This function expects the values of the angle parameter to be provided in radians (values from 0 to  $6.28$ ). Values are returned in the range -1 to 1.

Syntax: **sin(angle)**

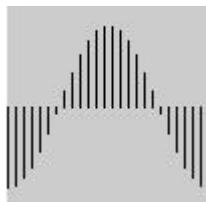
Parameters:

angle float: an angle in radians

Returns: float

Name: **cos()**

Examples



```
float a = 0.0;  
float inc = TWO_PI/25.0;  
for (int i = 0; i < 25; i++) {  
    line(i*4, 50, i*4, 50+cos(a)*40.0);  
    a = a + inc;  
}
```

Description: Calculates the cosine of an angle. This function expects the values of the angle parameter to be provided in radians (values from 0 to PI\*2). Values are returned in the range -1 to 1.

Syntax: **cos(angle)**

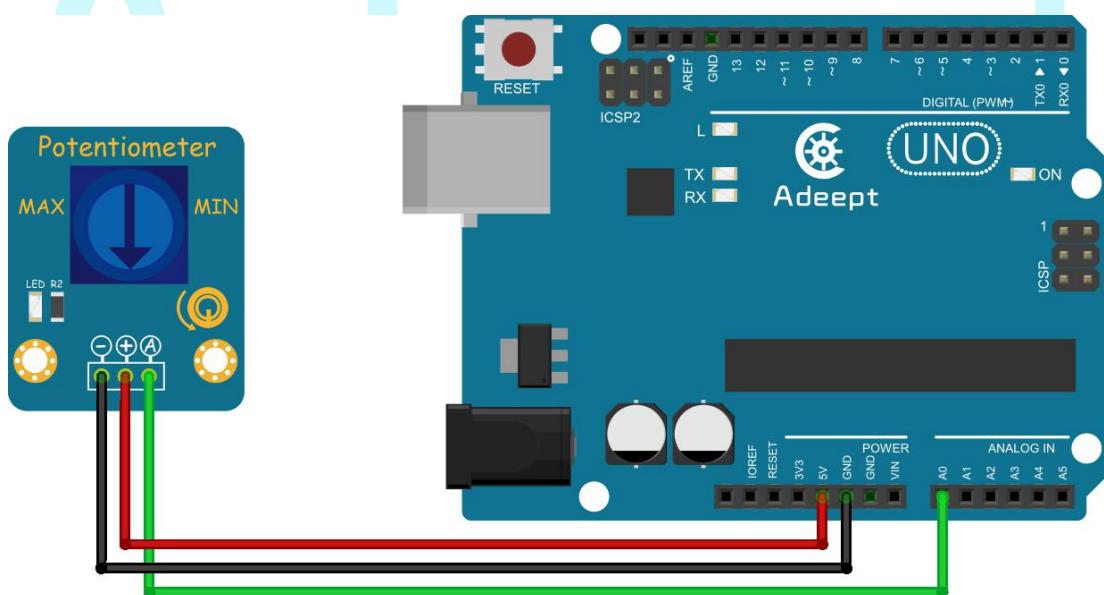
Parameters:

angle float: an angle in radians

Returns: float

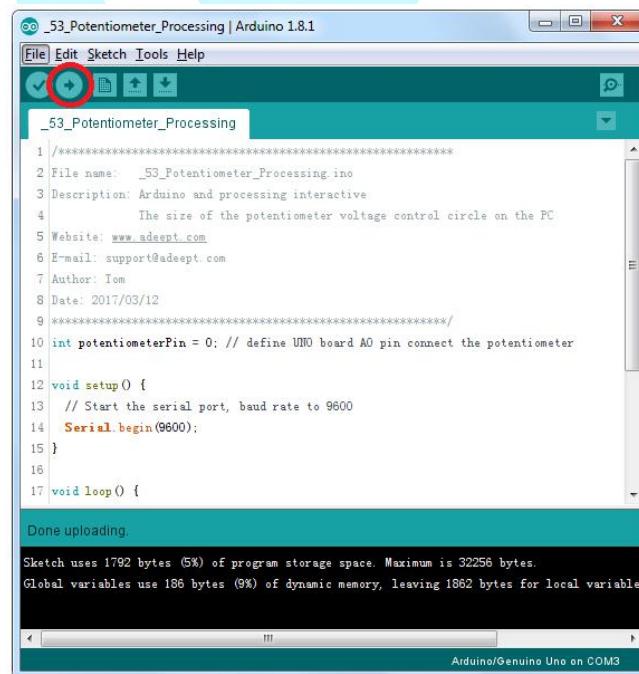
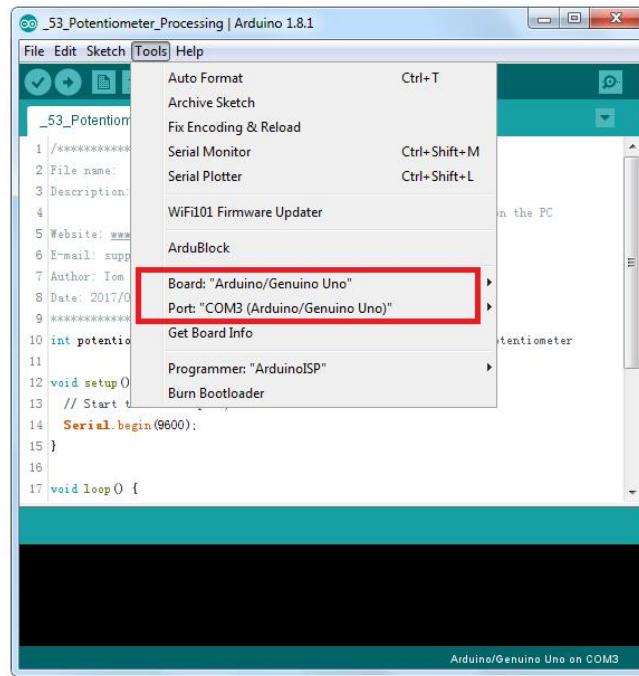
## Experimental Procedures

### Step 1: Build the circuit

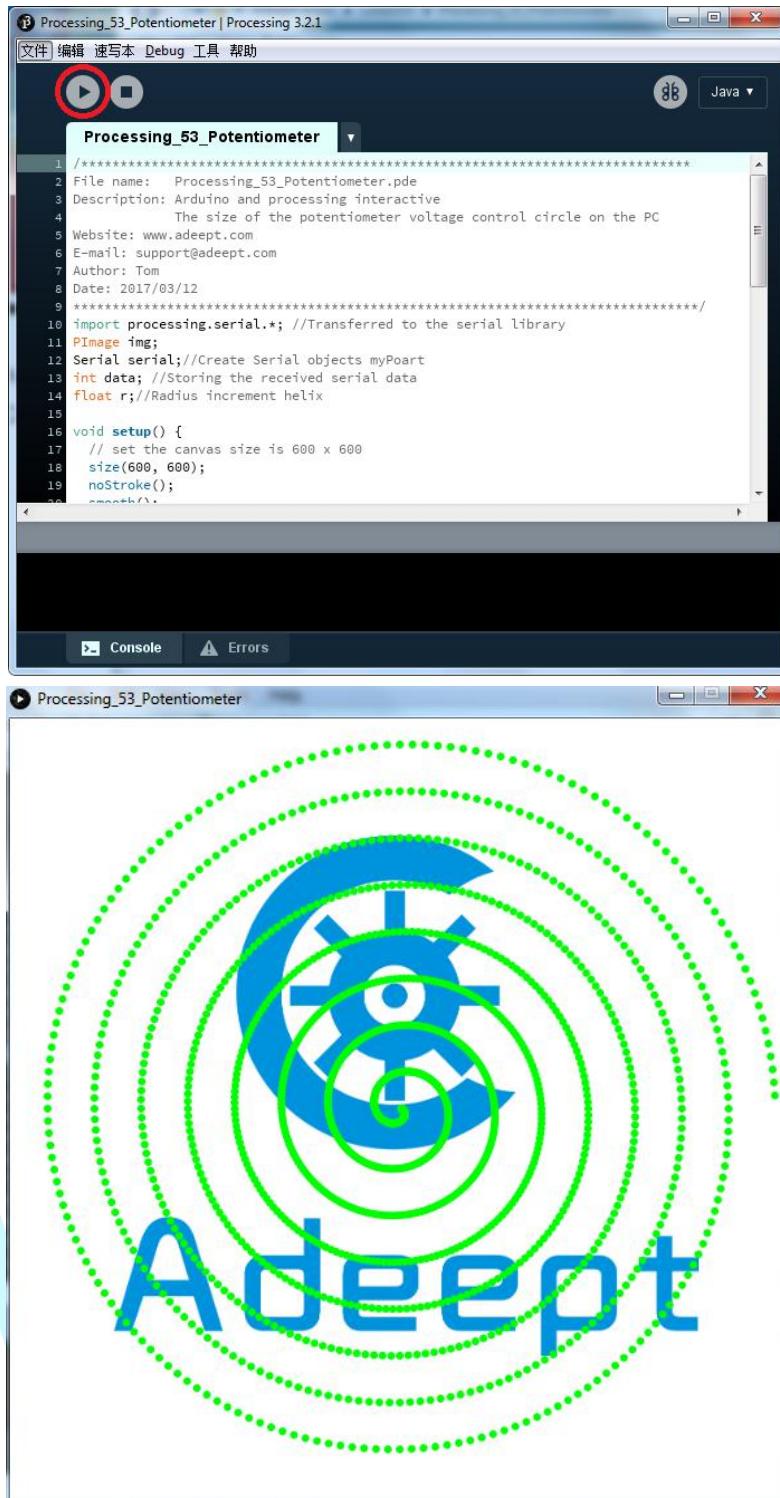


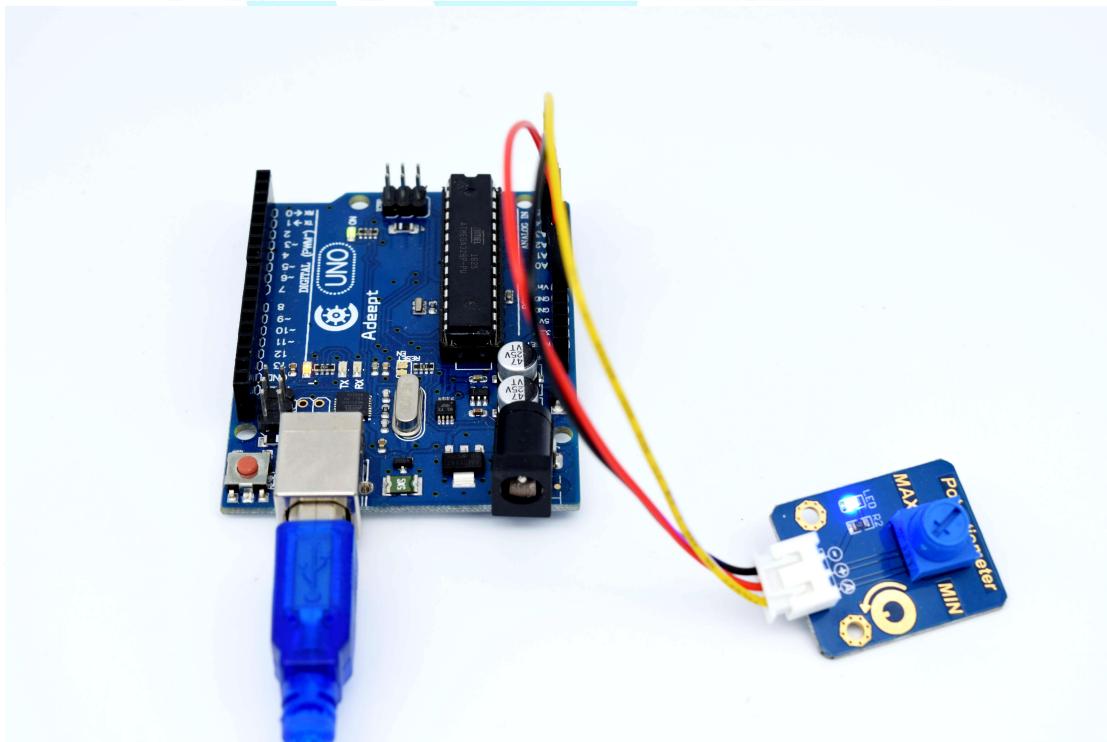
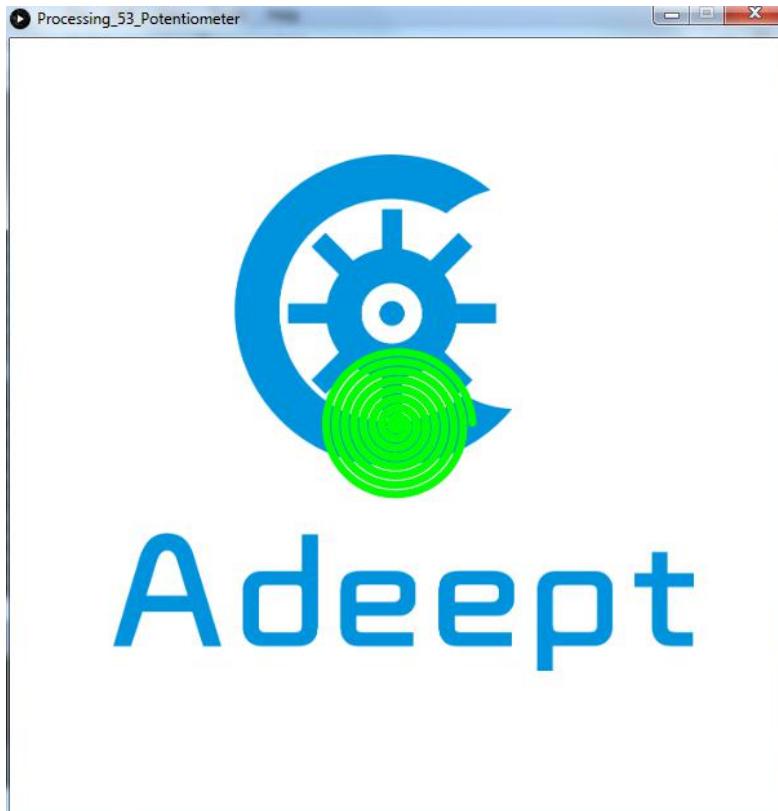
**Step 2:** Program \_53\_Potentiometer\_Processing.ino

**Step 3:** Compile and download the sketch to the UNO R3 board.



**Step 4:** Run the Processing software (Processing\_53\_Potentiometer.pde)





# Lesson 54 Arduino Interacts with Processing(Vibration Module)

## Introduction

When you tap the vibration sensor, you will see that the ball in the Processing interface will randomly hit.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Vibration Sensor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Principle

**Processing key function:**

Name: `random()`

Examples

```
for (int i = 0; i < 100; i++) {  
    float r = random(50);  
    stroke(r*5);  
    line(50, i, 50+r, i);  
}  
  
for (int i = 0; i < 100; i++) {  
    float r = random(-50, 50);  
    println(r);  
}  
  
// Get a random element from an array  
String[] words = {"apple", "bear", "cat", "dog"};  
int index = int(random(words.length)); // Same as int(random(4))  
println(words[index]); // Prints one of the four words
```

Description: Generates random numbers. Each time the `random()` function is called, it returns an unexpected value within the specified range. If only one parameter is passed to the function, it will return a float between zero and the value of the high parameter. For example, `random(5)` returns values between 0 and 5 (starting at zero, and up to, but not including, 5).

If two parameters are specified, the function will return a float with a value between the two values. For example, `random(-5, 10.2)` returns values starting at -5 and up to (but not including) 10.2. To convert a floating-point random number to an integer, use the `int()` function.

Syntax:

`random(high)`

**random(low, high)**

Parameters:

low float: lower limit

high float: upper limit

Returns: float

**Name: sqrt()**

Examples



```
noStroke();
float a = sqrt(6561); // Sets 'a' to 81
float b = sqrt(625); // Sets 'b' to 25
float c = sqrt(1); // Sets 'c' to 1
rect(0, 25, a, 10);
rect(0, 45, b, 10);
rect(0, 65, c, 10);
```

Description: Calculates the square root of a number. The square root of a number is always positive, even though there may be a valid negative root. The square root s of number a is such that s\*s = a. It is the opposite of squaring.

Syntax:

**sqrt(n)**

Parameters:

n float: non-negative number

Returns: float

**Name: atan2()**

Examples

```
void draw() {
    background(204);
    translate(width/2, height/2);
    float a = atan2(mouseY-height/2, mouseX-width/2);
    rotate(a);
    rect(-30, -5, 60, 10);
}
```

Description: Calculates the angle (in radians) from a specified point to the coordinate origin as measured from the positive x-axis. Values are returned as a float in the range from PI to -PI. The atan2() function is most often used for orienting geometry to the position of the cursor. Note: The y-coordinate of the point is the first parameter, and the x-coordinate is the second parameter, due to the structure of calculating the tangent.

Syntax:

### **atan2(y, x)**

Parameters:

y float: y-coordinate of the point  
x float: x-coordinate of the point

Returns: float

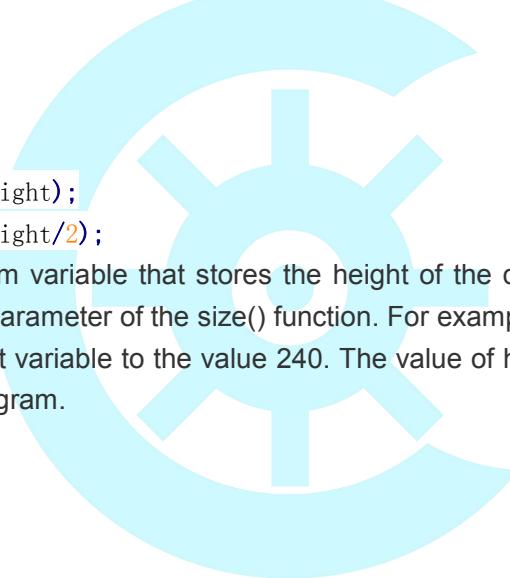
### Name: **height**

Examples



```
noStroke(0);  
background(0);  
rect(40, 0, 20, height);  
rect(60, 0, 20, height/2);
```

Description: System variable that stores the height of the display window. This value is set by the second parameter of the size() function. For example, the function call size(320, 240) sets the height variable to the value 240. The value of height defaults to 100 if size() is not used in a program.



### Name: **width**

Examples



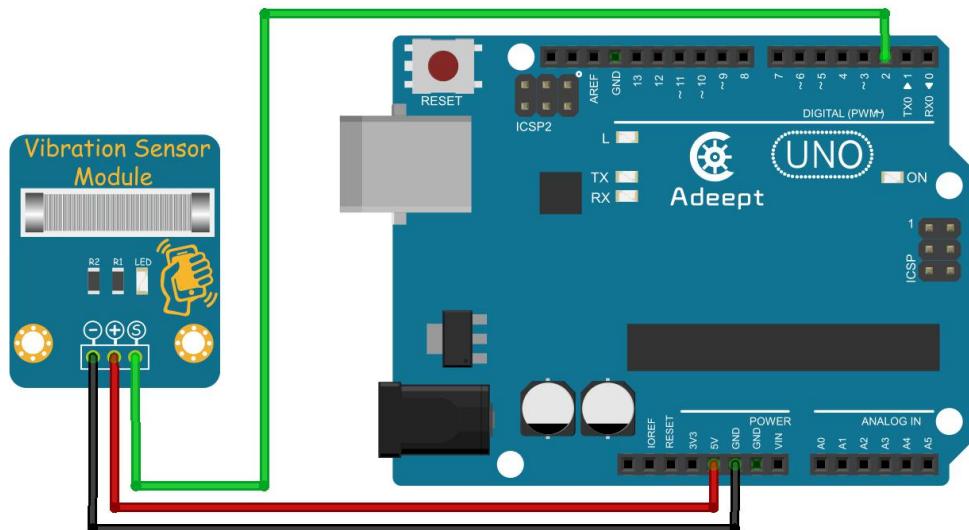
```
noStroke(0);  
background(0);  
rect(0, 40, width, 20);  
rect(0, 60, width/2, 20);
```



Description: System variable that stores the width of the display window. This value is set by the first parameter of the size() function. For example, the function call size(320, 240) sets the width variable to the value 320. The value of width defaults to 100 if size() is not used in a program.

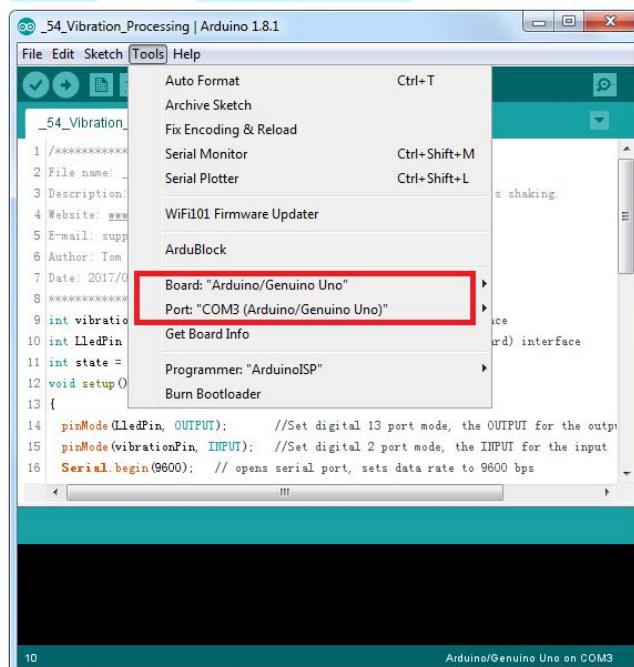
## **Experimental Procedures**

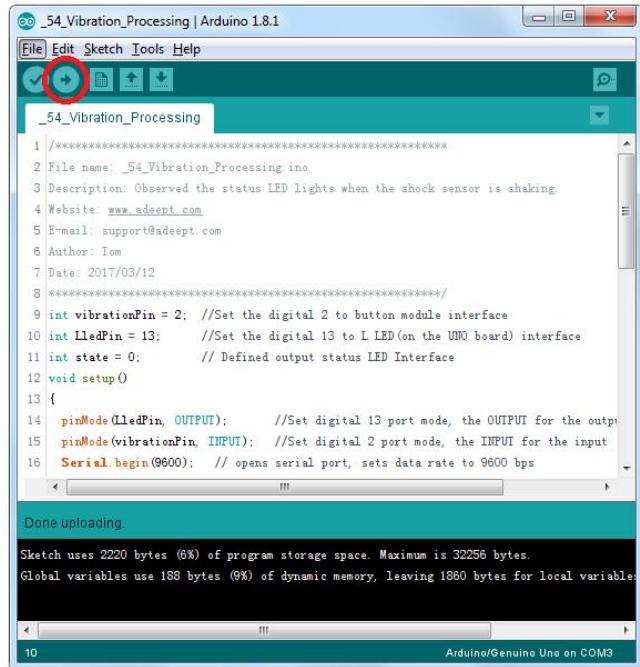
### **Step 1: Build the circuit**



**Step 2:** Program `_54_Vibration_Processing.ino`

**Step 3:** Compile and download the sketch to the UNO R3 board.





```

_54_Vibration_Processing | Arduino 1.8.1
File Edit Sketch Tools Help
_54_Vibration_Processing
1 //*****
2 File name: _54_Vibration_Processing.ino
3 Description: Observed the status LED lights when the shock sensor is shaking.
4 Website: www.adeept.com
5 E-mail: support@adeept.com
6 Author: Tom
7 Date: 2017/03/12
8 ****
9 int vibrationPin = 2; //Set the digital 2 to button module interface
10 int lLedPin = 13; //Set the digital 13 to L LED(on the UNO board) interface
11 int state = 0; // Defined output status LED Interface
12 void setup()
13 {
14   pinMode(lLedPin, OUTPUT); //Set digital 13 port mode, the OUTPUT for the output
15   pinMode(vibrationPin, INPUT); //Set digital 2 port mode, the INPUT for the input
16   Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

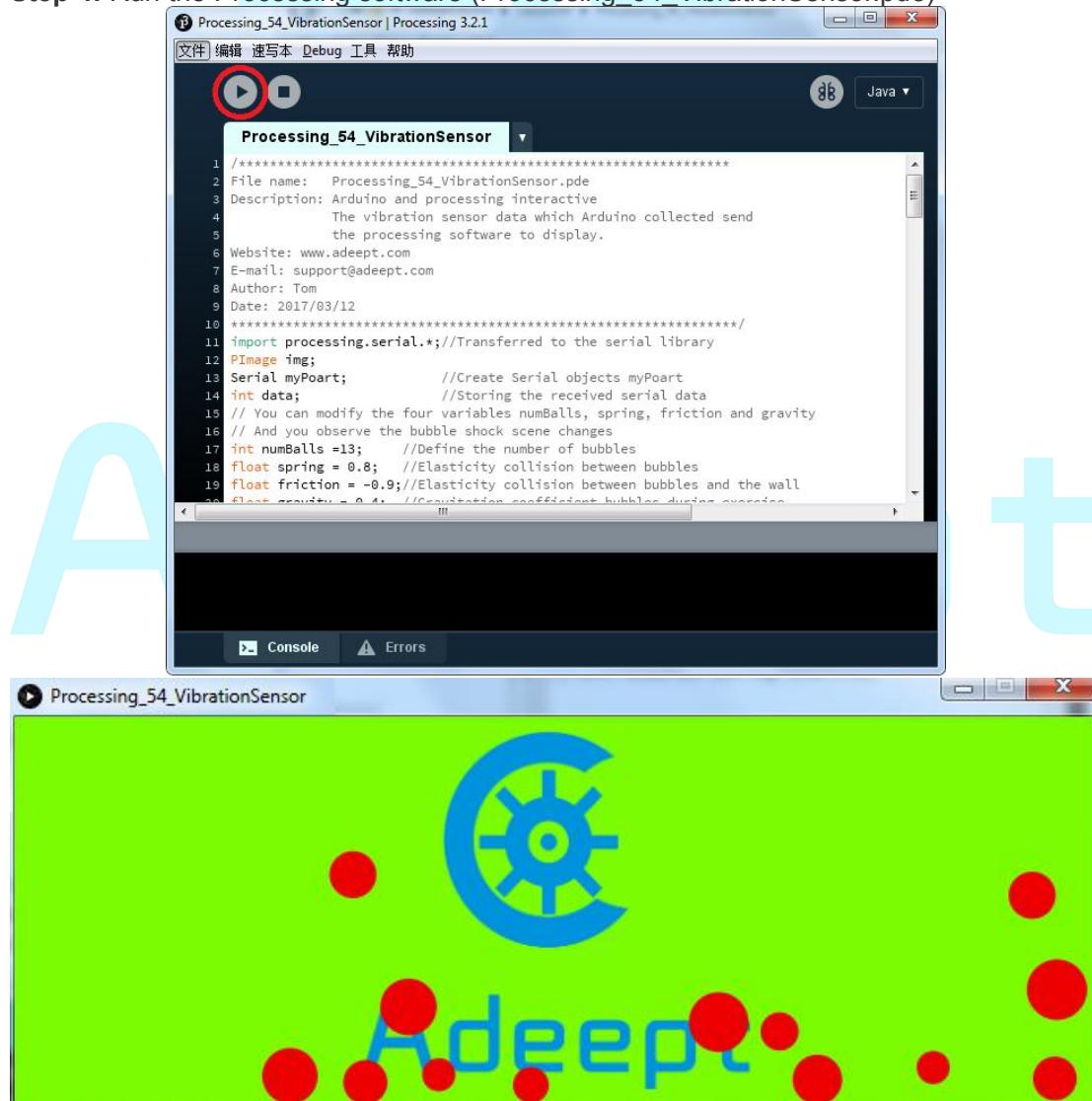
```

Done uploading.

Sketch uses 2220 bytes (6%) of program storage space. Maximum is 32256 bytes.  
Global variables use 188 bytes (9%) of dynamic memory, leaving 1860 bytes for local variables.

10 Arduino/Genuino Uno on COM3

#### Step 4: Run the Processing software (Processing\_54\_VibrationSensor.pde)



Processing\_54\_VibrationSensor | Processing 3.2.1

文件 编辑 速写本 Debug 工具 帮助

Java

Processing\_54\_VibrationSensor

```

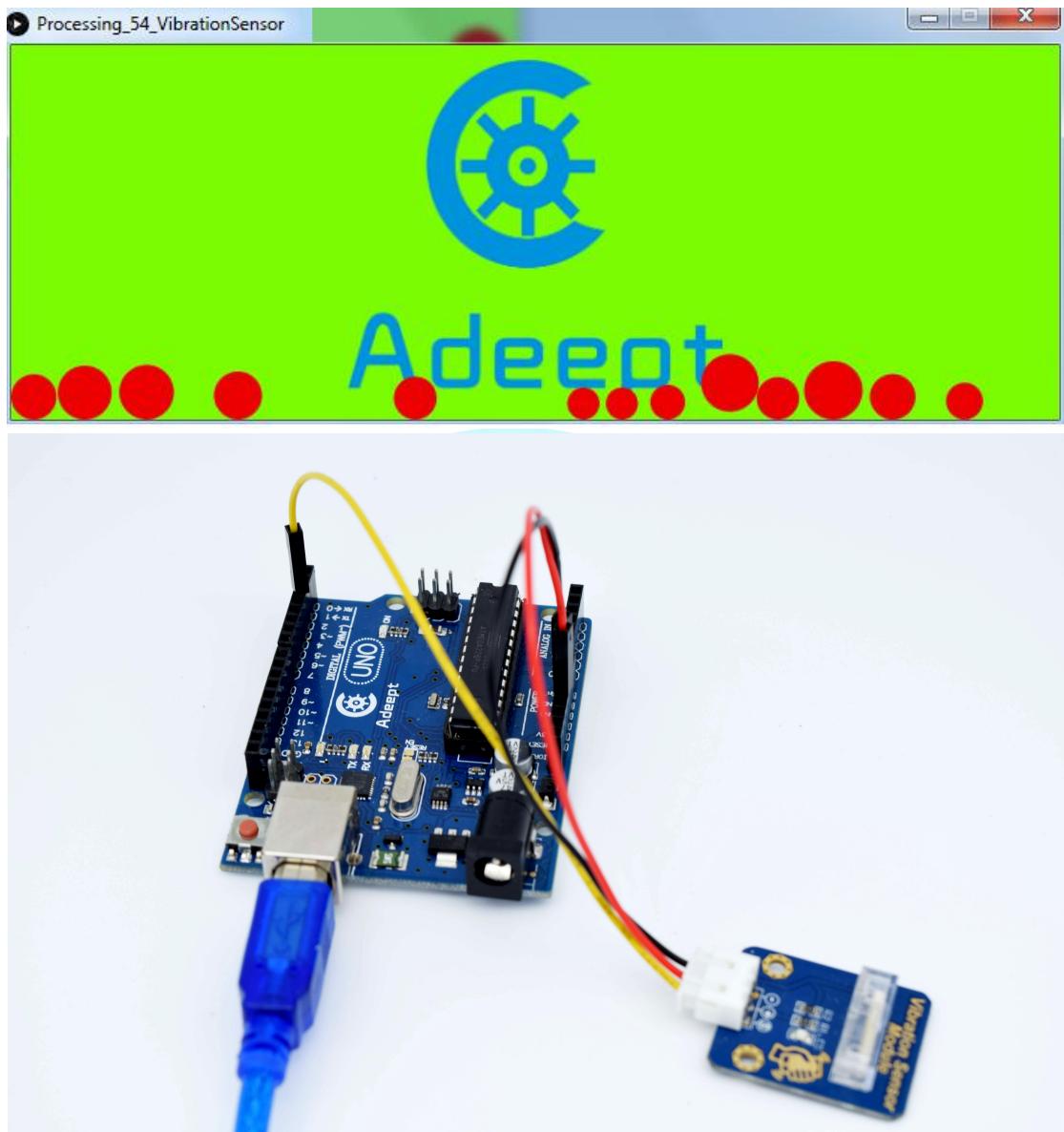
1 //*****
2 File name: Processing_54_VibrationSensor.pde
3 Description: Arduino and processing interactive
4   The vibration sensor data which Arduino collected send
5   the processing software to display.
6 Website: www.adeept.com
7 E-mail: support@adeept.com
8 Author: Tom
9 Date: 2017/03/12
10 ****
11 import processing.serial.*; //Transferred to the serial library
12 PImage img;
13 Serial myPoart; //Create Serial objects myPoart
14 int data; //Storing the received serial data
15 // You can modify the four variables numBalls, spring, friction and gravity
16 // And you observe the bubble shock scene changes
17 int numBalls = 13; //Define the number of bubbles
18 float spring = 0.8; //Elasticity collision between bubbles
19 float friction = -0.9; //Elasticity collision between bubbles and the wall
20 float gravity = 0.4; //Friction coefficient bubbles during collision

```

Console Errors

Processing\_54\_VibrationSensor





# Adeept

# Lesson 55 Arduino Interacts with Processing(Photoresistor)

## Introduction

In this lesson, the brightness of the picture in the Processing interface changes with ambient light intensity.

## Components

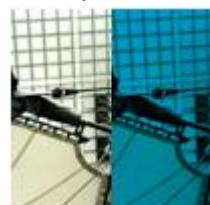
- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Photoresistor Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

## Principle

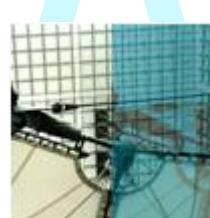
Processing key function:

Name: tint()

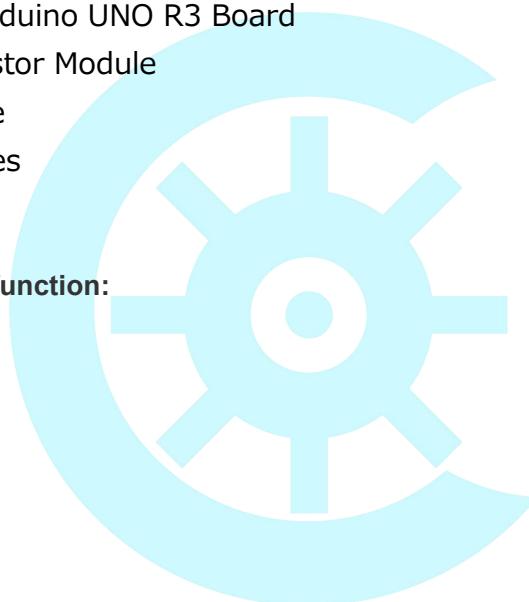
Examples

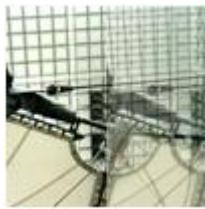


```
PIImage img;  
img = loadImage("laDefense.jpg");  
image(img, 0, 0);  
tint(0, 153, 204); // Tint blue  
image(img, 50, 0);
```



```
PIImage img;  
img = loadImage("laDefense.jpg");  
image(img, 0, 0);  
tint(0, 153, 204, 126); // Tint blue and set transparency  
image(img, 50, 0);
```





```
PImage img;  
img = loadImage("laDefense.jpg");  
image(img, 0, 0);  
tint(255, 126); // Apply transparency without changing color  
image(img, 50, 0);
```

**Description** Sets the fill value for displaying images. Images can be tinted to specified colors or made transparent by including an alpha value.

To apply transparency to an image without affecting its color, use white as the tint color and specify an alpha value. For instance, tint(255, 128) will make an image 50% transparent (assuming the default alpha range of 0-255, which can be changed with colorMode()).

When using hexadecimal notation to specify a color, use "#" or "0x" before the values (e.g., #CCFFAA or 0xFFCCFFAA). The # syntax uses six digits to specify a color (just as colors are typically specified in HTML and CSS). When using the hexadecimal notation starting with "0x", the hexadecimal value must be specified with eight characters; the first two characters define the alpha component, and the remainder define the red, green, and blue components.

The value for the gray parameter must be less than or equal to the current maximum value as specified by colorMode(). The default maximum value is 255.

The tint() function is also used to control the coloring of textures in 3D.

**Syntax**

```
tint(rgb)  
tint(rgb, alpha)  
tint(gray)  
tint(gray, alpha)  
tint(v1, v2, v3)  
tint(v1, v2, v3, alpha)
```

**Parameters**

rgb int: color value in hexadecimal notation

alpha float: opacity of the image

grayfloat: specifies a value between white and black

v1 float: red or hue value (depending on current color mode)

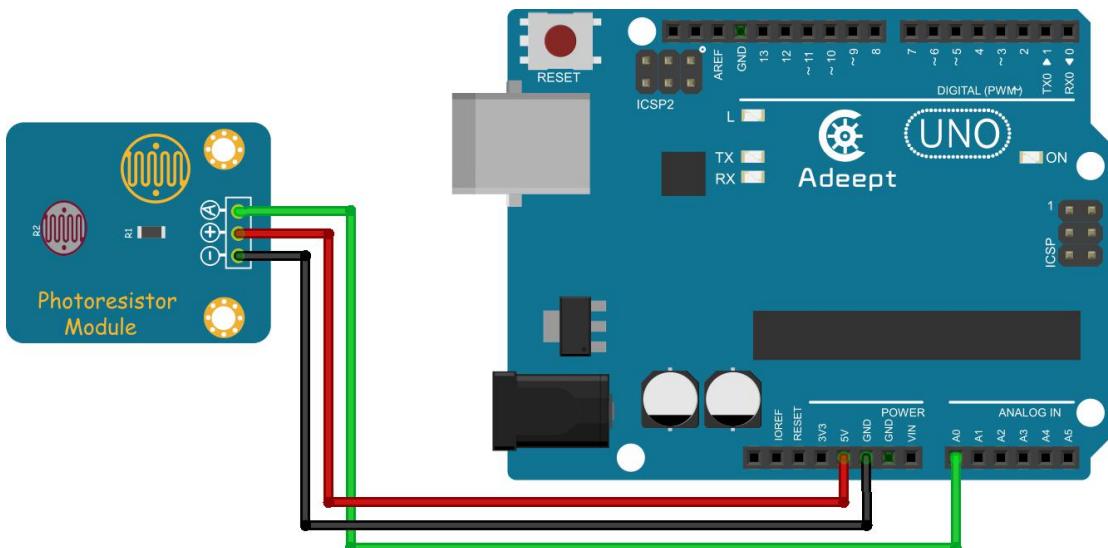
v2 float: green or saturation value (depending on current color mode)

v3 float: blue or brightness value (depending on current color mode)

**Returns** void

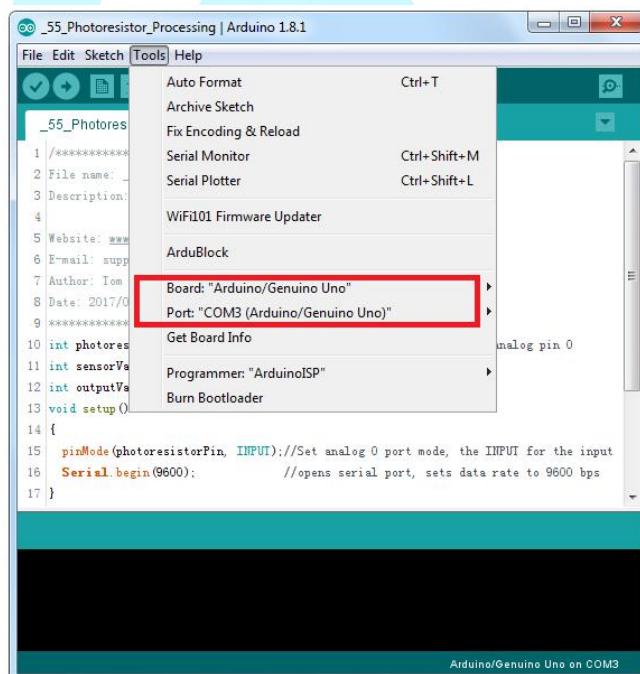
## Experimental Procedures

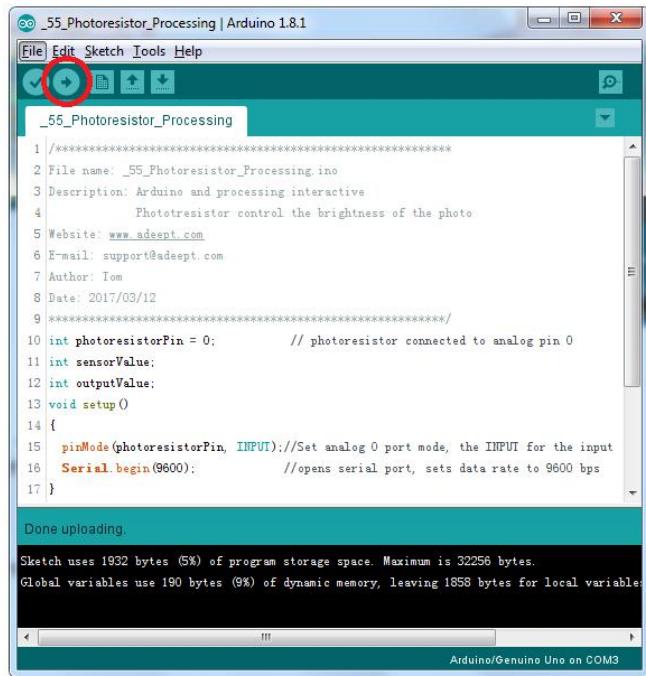
### Step 1: Build the circuit



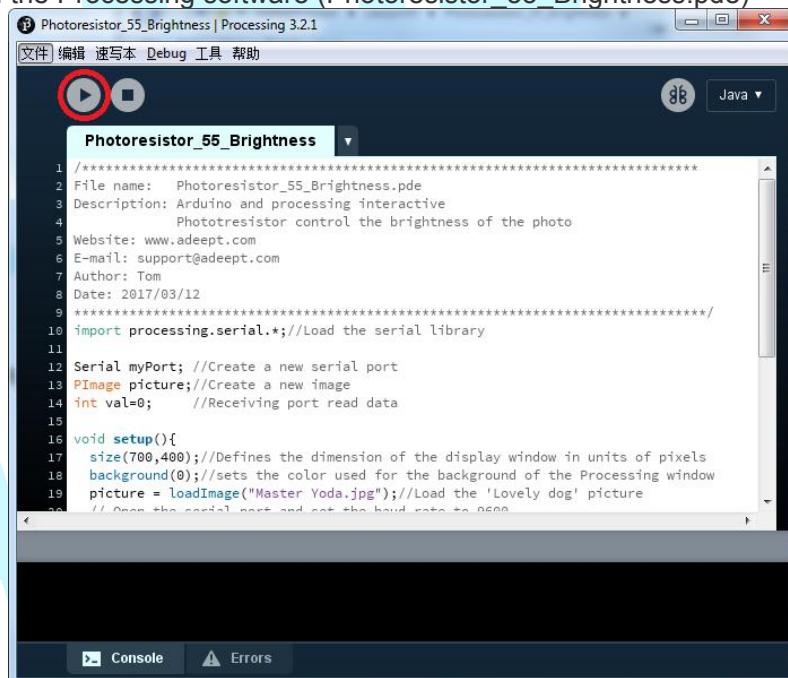
**Step 2:** Program `_55_Photoresistor_Processing.ino`

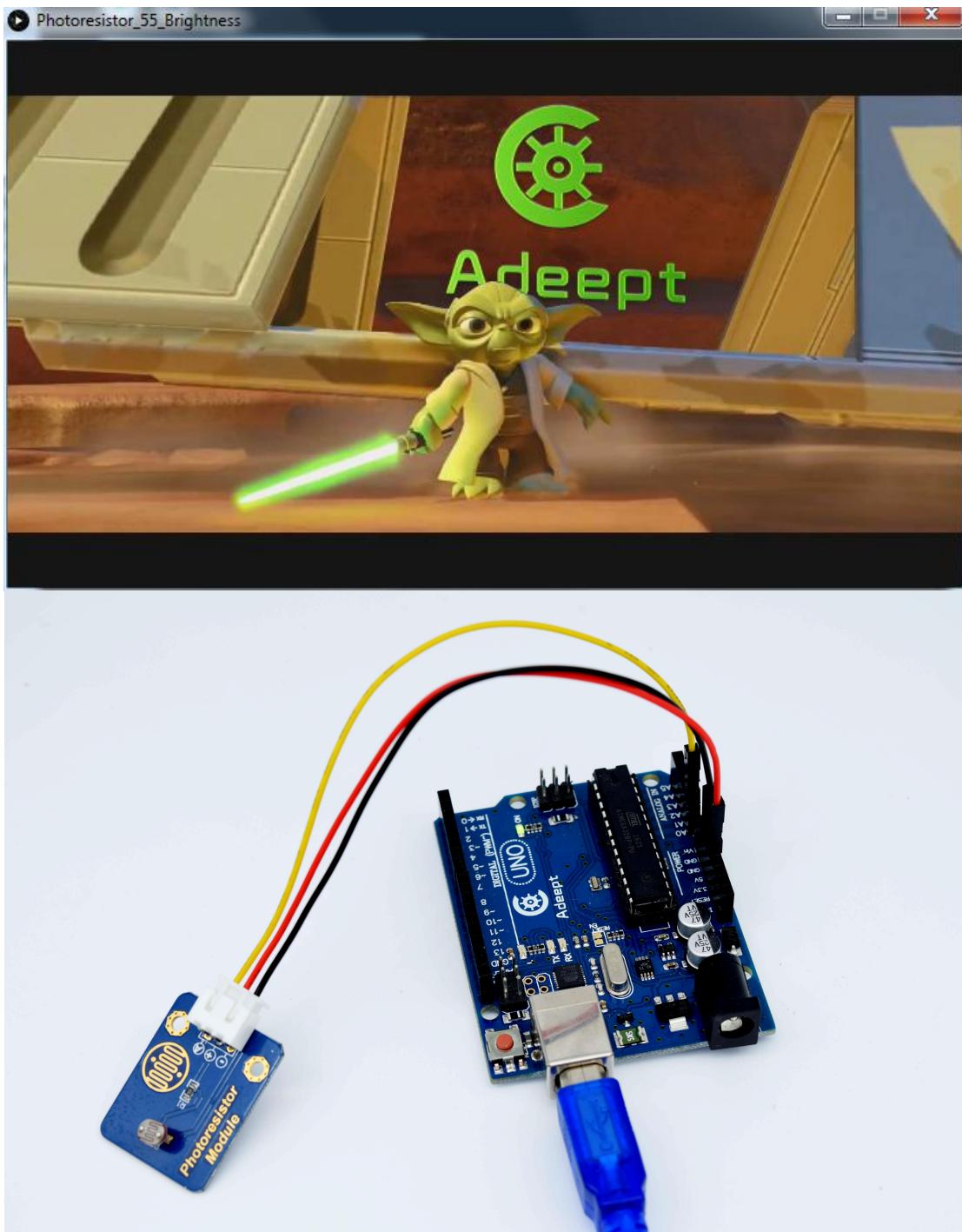
**Step 3:** Compile and download the sketch to the UNO R3 board.





#### Step 4: Run the Processing software (Photoresistor\_55\_Brightness.pde)





# Lesson 56 Arduino Interacts with Processing(DS18B20 Module)

## Introduction

In this lesson, the Arduino detects the ambient temperature through the DS18B20 and then displays the temperature data as a curve in the Processing interface.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* DS18B20 Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

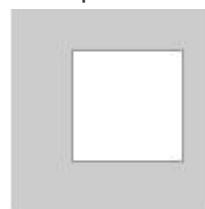
## Principle

Processing key function:

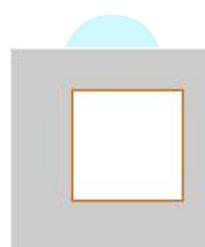
Name

**stroke()**

Examples



```
stroke(153);  
rect(30, 20, 55, 55);
```



```
stroke(204, 102, 0);  
rect(30, 20, 55, 55);
```

Description Sets the color used to draw lines and borders around shapes. This color is either specified in terms of the RGB or HSB color depending on the current `colorMode()`. The default color space is RGB, with each value in the range from 0 to 255.

When using hexadecimal notation to specify a color, use "#" or "0x" before the values (e.g., #CCFFAA or 0xFFCCFFAA). The # syntax uses six digits to specify a color (just as colors are typically specified in HTML and CSS). When using the hexadecimal notation starting with "0x", the hexadecimal value must be specified with eight characters; the first two characters define the alpha component, and the remainder define the red, green, and blue

components.

The value for the gray parameter must be less than or equal to the current maximum value as specified by colorMode(). The default maximum value is 255.

When drawing in 2D with the default renderer, you may need hint(ENABLE\_STROKE\_PURE) to improve drawing quality (at the expense of performance). See the hint() documentation for more details.

Syntax

**stroke(rgb)**  
**stroke(rgb, alpha)**  
**stroke(gray)**  
**stroke(gray, alpha)**  
**stroke(v1, v2, v3)**  
**stroke(v1, v2, v3, alpha)**

Parameters

rgb int: color value in hexadecimal notation

alpha float: opacity of the stroke

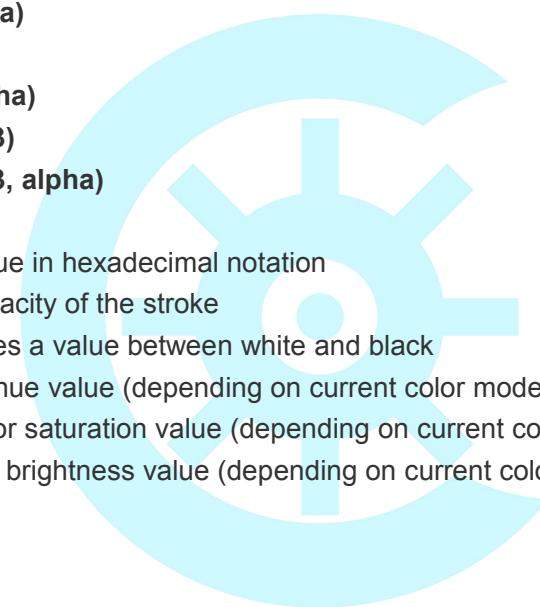
grayfloat: specifies a value between white and black

v1 float: red or hue value (depending on current color mode)

v2 float: green or saturation value (depending on current color mode)

v3 float: blue or brightness value (depending on current color mode)

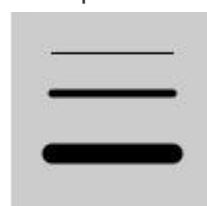
Returns void



Name

**strokeWeight()**

Examples



```
strokeWeight(1); // Default  
line(20, 20, 80, 20);  
strokeWeight(4); // Thicker  
line(20, 40, 80, 40);  
strokeWeight(10); // Beastly  
line(20, 70, 80, 70);
```

Description Sets the width of the stroke used for lines, points, and the border around shapes. All widths are set in units of pixels.

Syntax

**strokeWeight(weight)**

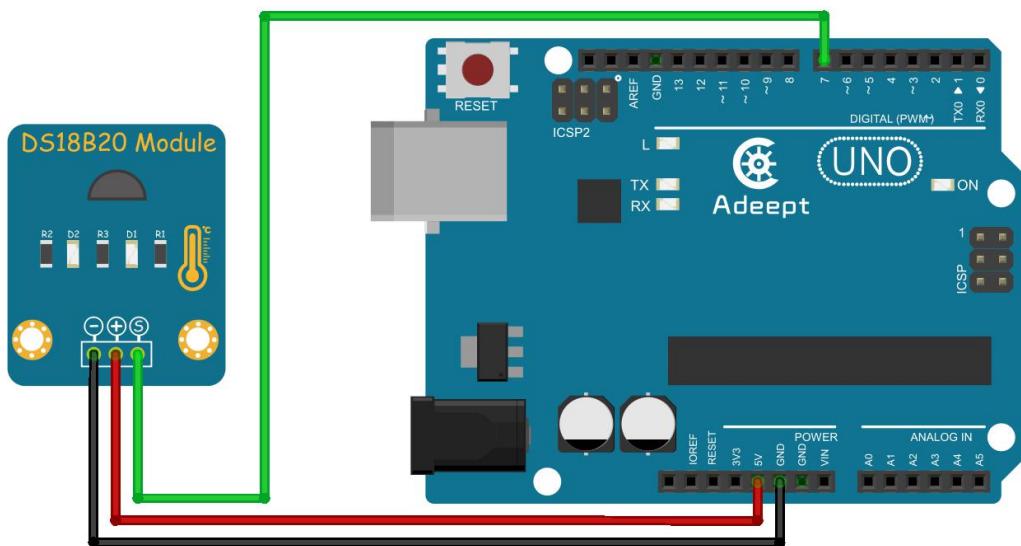
Parameters

weight float: the weight (in pixels) of the stroke

Returns void

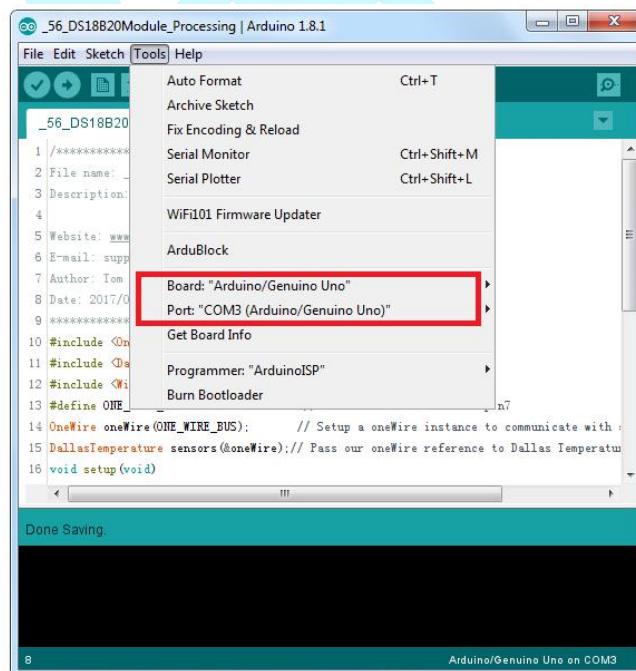
## Experimental Procedures

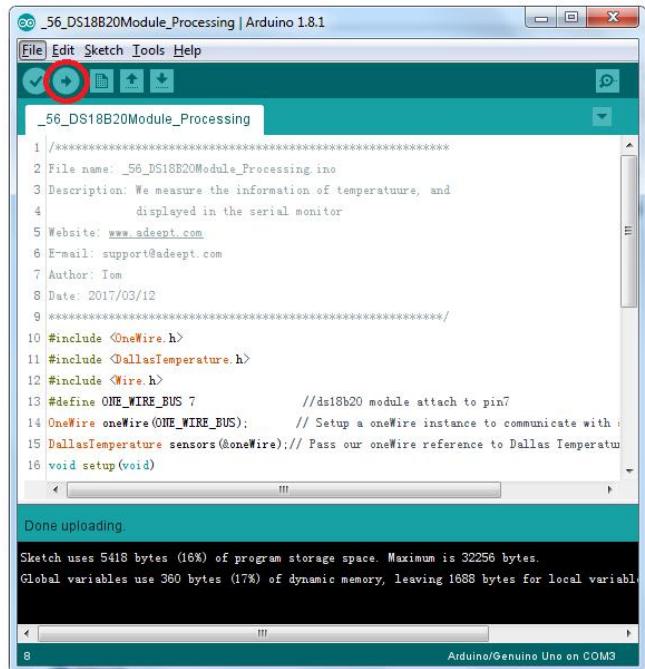
### Step 1: Build the circuit



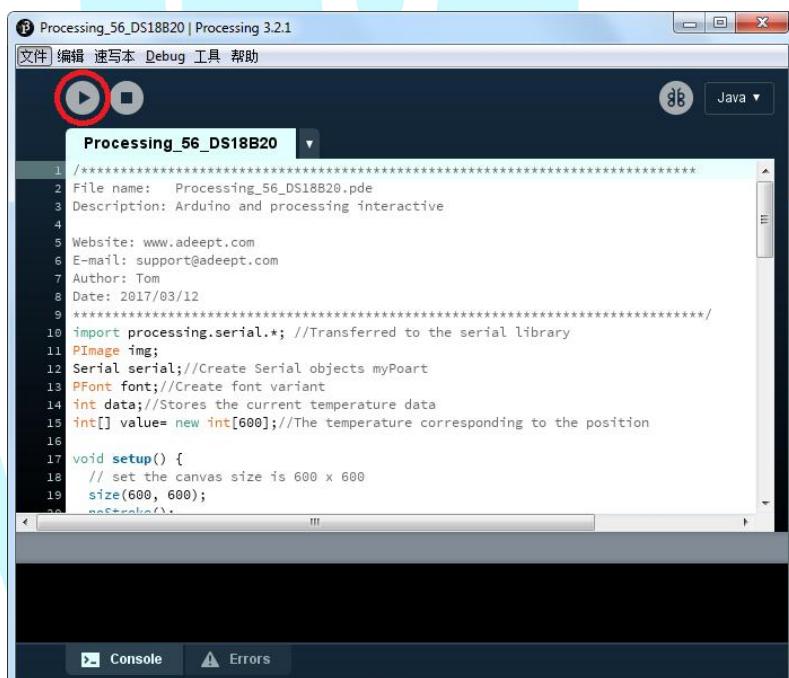
**Step 2:** Program `_56_DS18B20Module_Processing.ino`

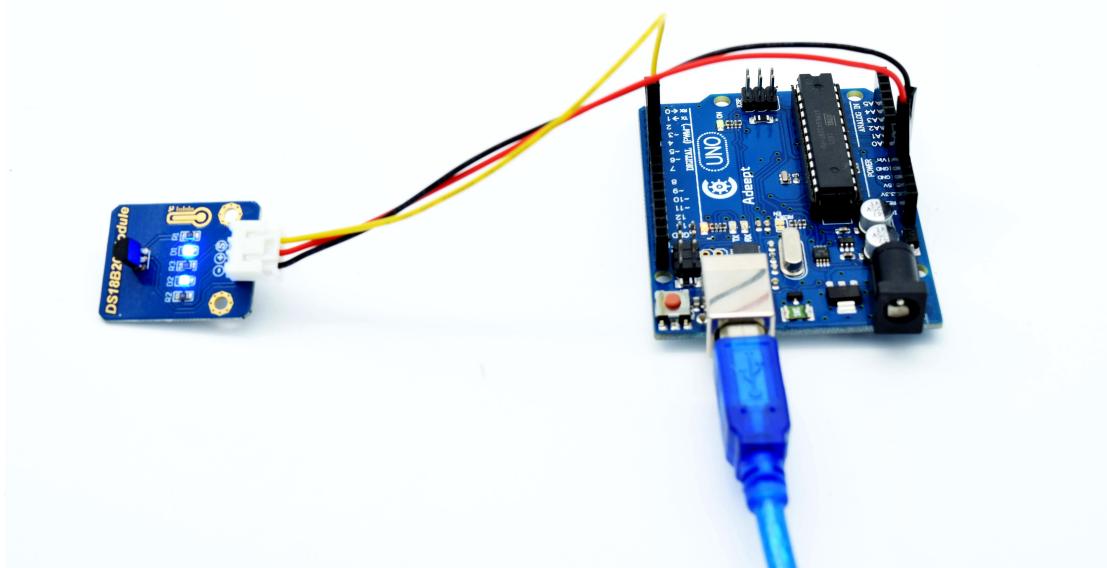
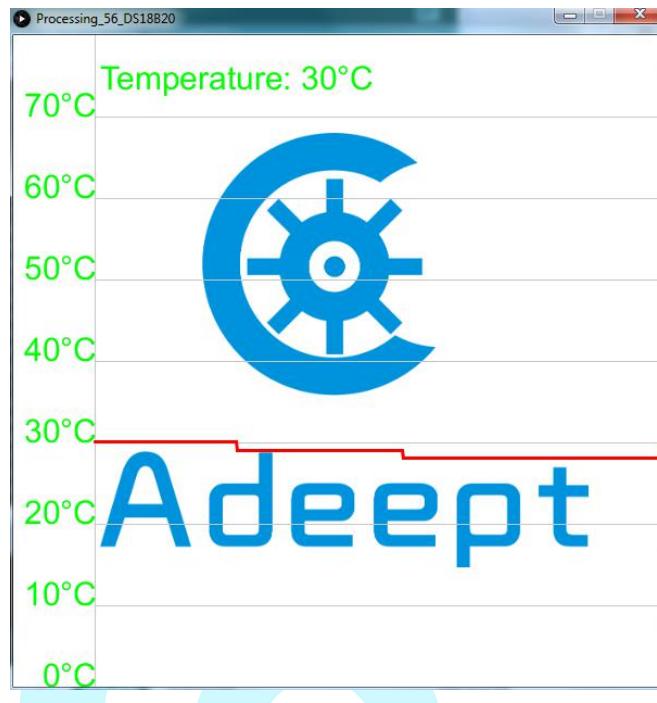
**Step 3:** Compile and download the sketch to the UNO R3 board.





#### Step 4: Run the Processing software (Processing\_56\_DS18B20.pde)





# Lesson 57 Arduino Interacts with Processing(Buzzer Module)

## Introduction

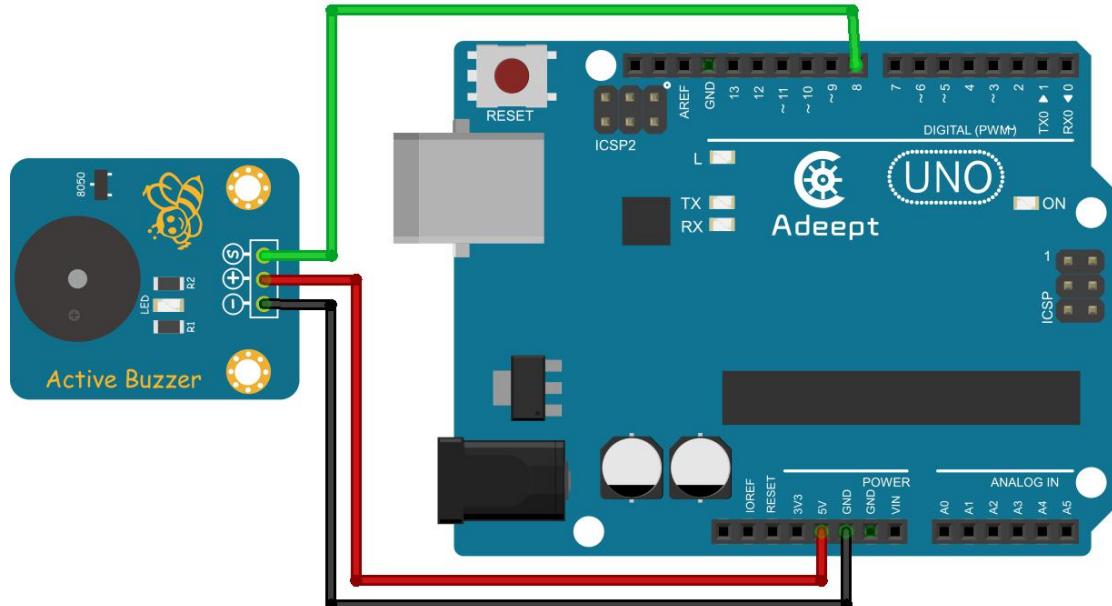
In this lesson, when the mouse pointer points to the buzzer in the Processing interface, the buzzer connected to the Arduino will sound.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Active Buzzer Module
- 1 \* USB Cable
- 1 \* 3-Pin Wires

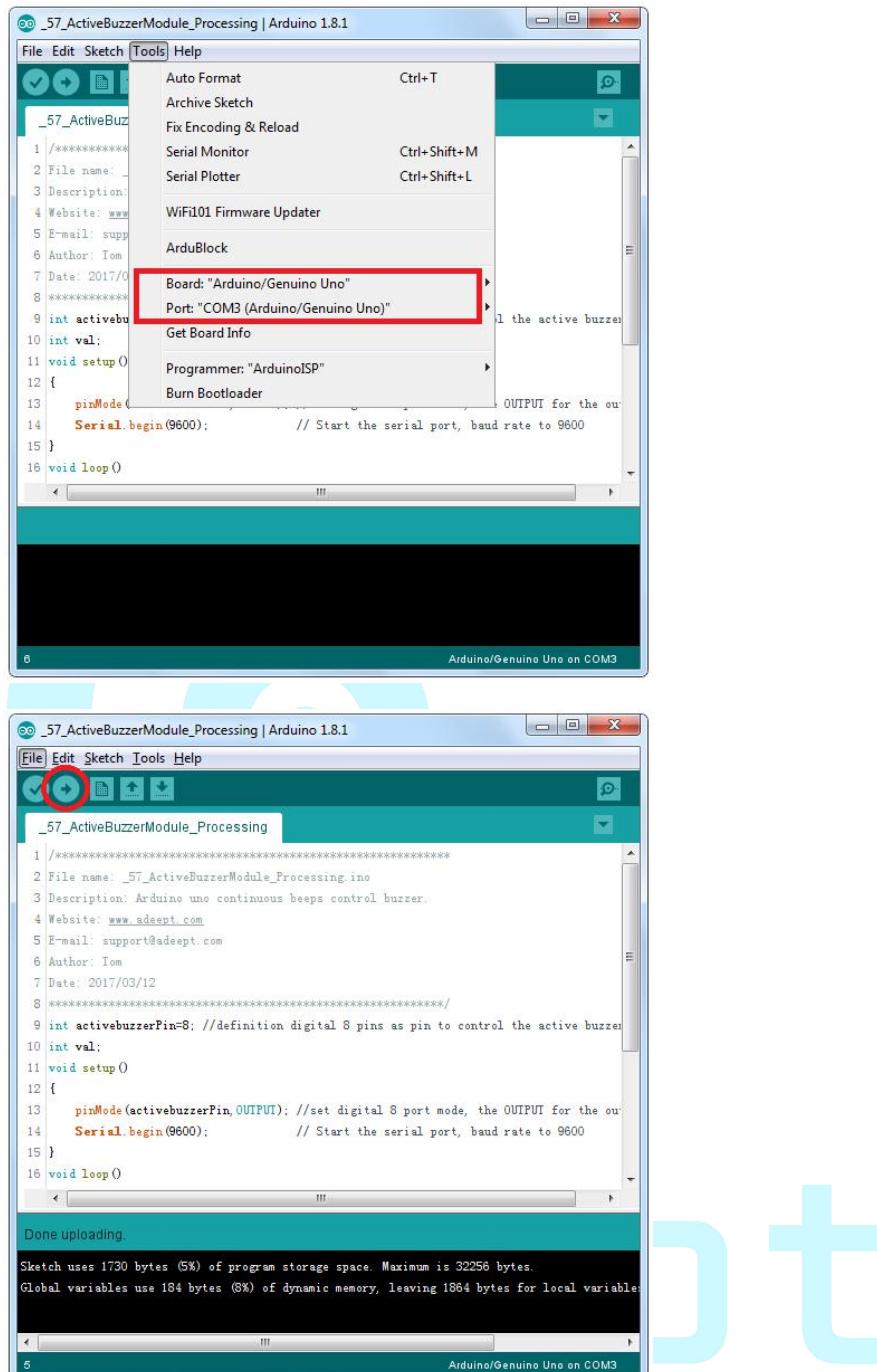
## Experimental Procedures

### Step 1: Build the circuit



**Step 2:** Program `_57_ActiveBuzzerModule_Processing.ino`

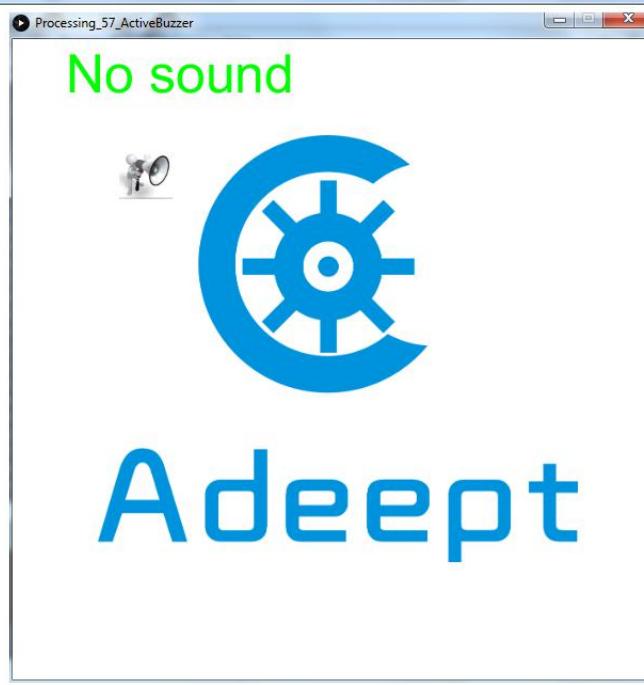
**Step 3:** Compile and download the sketch to the UNO R3 board.

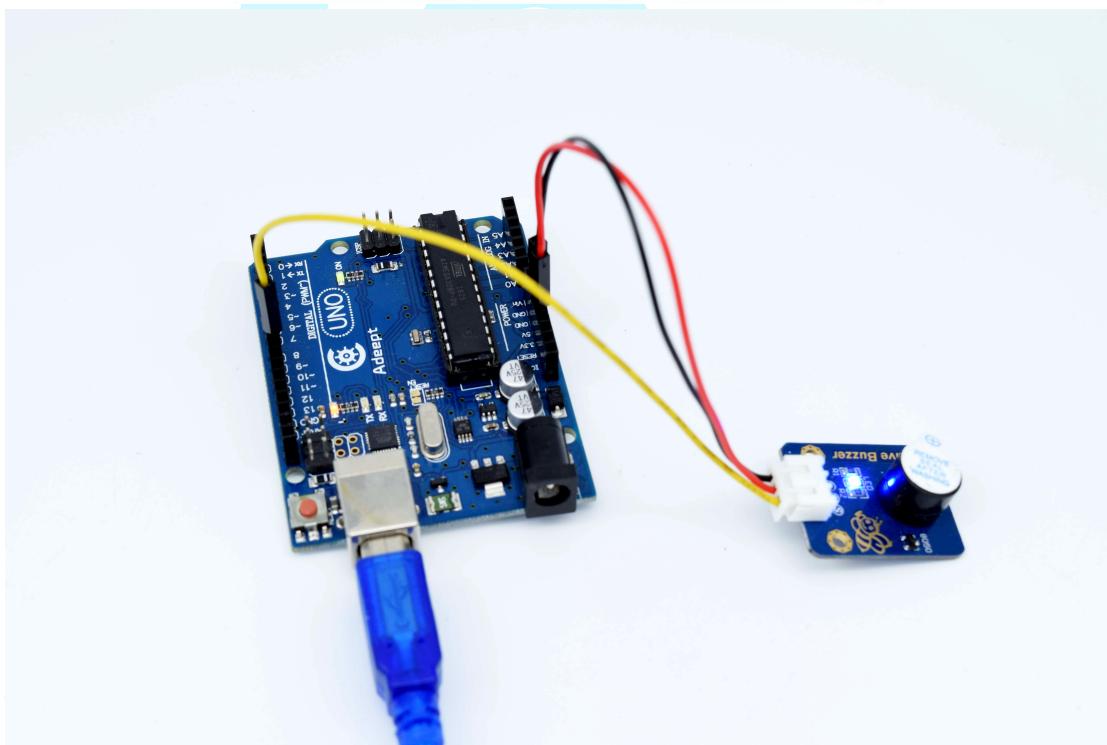
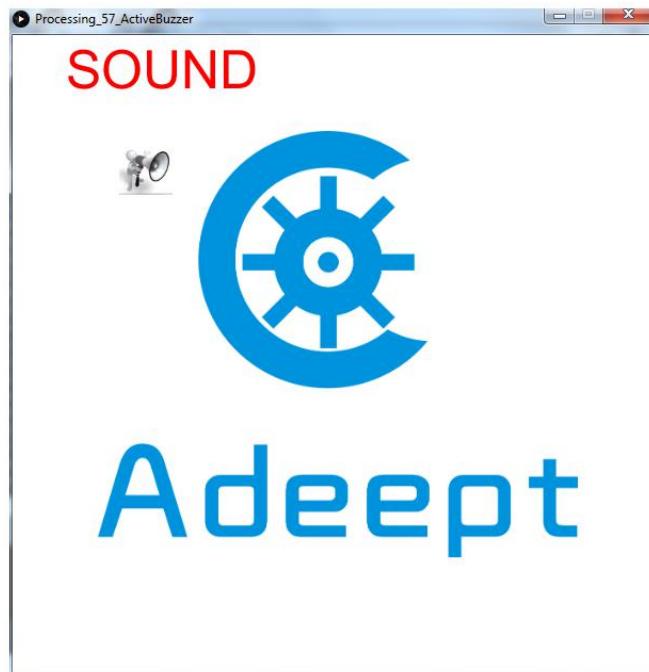


**Step 4:** Run the Processing software (Processing\_57\_ActiveBuzzer.pde)

The screenshot shows the Processing IDE interface. The title bar reads "Processing\_57\_ActiveBuzzer | Processing 3.2.1". The menu bar includes "文件" (File), "编辑" (Edit), "速写本" (Sketchbook), "Debug", "工具" (Tools), and "帮助" (Help). A toolbar with icons for play, stop, and other functions is visible. The code editor contains the following PDE code:

```
1 //*****  
2 File name: Processing_57_ActiveBuzzer.pde  
3 Description: Arduino and processing interactive  
4  
5 Website: www.adeept.com  
6 E-mail: support@adeept.com  
7 Author: Tom  
8 Date: 2017/03/12  
9 *****  
10 import processing.serial.*; //Transferred to the serial library  
11 PImage img;  
12 PImage sound;  
13 Serial serial; //Create Serial objects myPort  
14 PFont font; //Create font variant  
15  
16 void setup() {  
17     // set the canvas size is 600 x 600  
18     size(600, 600);  
19     noStroke();  
20     smooth();  
21 }
```





# Lesson 58 Arduino Interacts with Processing(Rotary Encoder)

## Introduction

When you rotate the knob of the rotary encoder, the stripes in the color bar will be increased or decreased.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Rotary Encoder Module
- 1 \* USB Cable
- 1 \* 5-Pin Wires

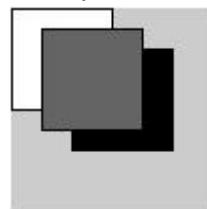
## Principle

Processing key function:

Name

**pushMatrix()**

Examples



```
fill(255);  
rect(0, 0, 50, 50); // White rectangle
```

```
pushMatrix();  
translate(30, 20);  
fill(0);  
rect(0, 0, 50, 50); // Black rectangle  
popMatrix();
```

```
fill(100);  
rect(15, 10, 50, 50); // Gray rectangle
```

### Description

Pushes the current transformation matrix onto the matrix stack. Understanding `pushMatrix()` and `popMatrix()` requires understanding the concept of a matrix stack. The `pushMatrix()` function saves the current coordinate system to the stack and `popMatrix()` restores the prior coordinate system. `pushMatrix()` and `popMatrix()` are used in conjunction with the other transformation functions and may be embedded to control the scope of the transformations.

### Syntax

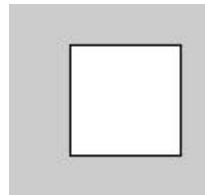
**pushMatrix()**

Returns void

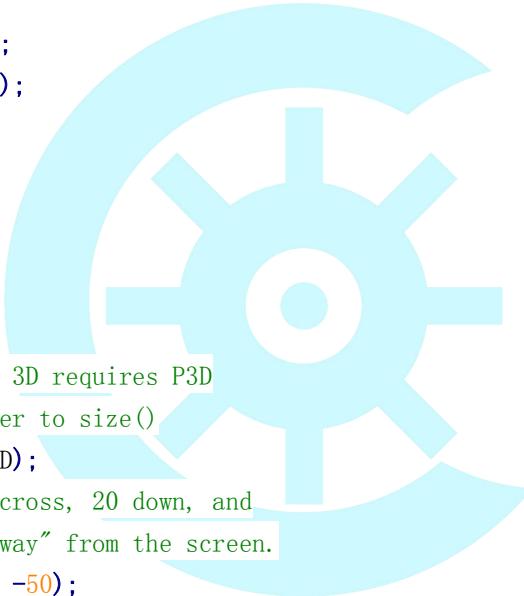
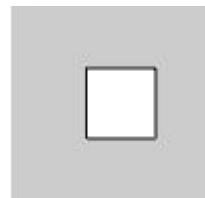
Name

**translate()**

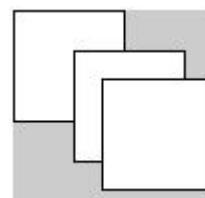
Examples



```
translate(30, 20);
rect(0, 0, 55, 55);
```



```
// Translating in 3D requires P3D
// as the parameter to size()
size(100, 100, P3D);
// Translate 30 across, 20 down, and
// 50 back, or "away" from the screen.
translate(30, 20, -50);
rect(0, 0, 55, 55);
```



```
rect(0, 0, 55, 55); // Draw rect at original 0,0
translate(30, 20);
rect(0, 0, 55, 55); // Draw rect at new 0,0
translate(14, 14);
rect(0, 0, 55, 55); // Draw rect at new 0,0
```

Description Specifies an amount to displace objects within the display window. The x parameter specifies left/right translation, the y parameter specifies up/down translation, and the z parameter specifies translations toward/away from the screen. Using this function with the z parameter requires using P3D as a parameter in combination with size as shown in the above example.

Transformations are cumulative and apply to everything that happens after and subsequent calls to the function accumulates the effect. For example, calling translate(50,

0) and then translate(20, 0) is the same as translate(70, 0). If translate() is called within draw(), the transformation is reset when the loop begins again. This function can be further controlled by using pushMatrix() and popMatrix().

Syntax

**translate(x, y)**

**translate(x, y, z)**

Parameters

x float: left/right translation

y float: up/down translation

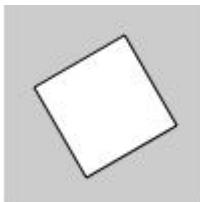
z float: forward/backward translation

Returns void

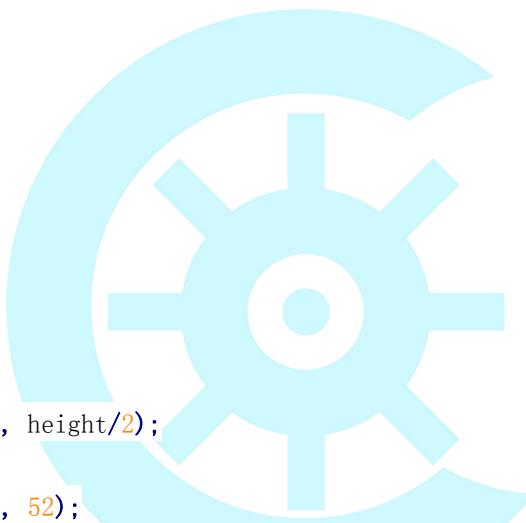
Name

**rotate()**

Examples



```
translate(width/2, height/2);  
rotate(PI/3.0);  
rect(-26, -26, 52, 52);
```



Description Rotates the amount specified by the angle parameter. Angles must be specified in radians (values from 0 to TWO\_PI), or they can be converted from degrees to radians with the radians() function.

The coordinates are always rotated around their relative position to the origin. Positive numbers rotate objects in a clockwise direction and negative numbers rotate in the counterclockwise direction. Transformations apply to everything that happens afterward, and subsequent calls to the function compound the effect. For example, calling rotate(PI/2.0) once and then calling rotate(PI/2.0) a second time is the same as a single rotate(PI). All transformations are reset when draw() begins again.

Technically, rotate() multiplies the current transformation matrix by a rotation matrix. This function can be further controlled by pushMatrix() and popMatrix().

Syntax

**rotate(angle)**

Parameters

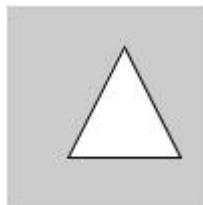
angle float: angle of rotation specified in radians

Returns void

Name

**triangle()**

Examples



**triangle(30, 75, 58, 20, 86, 75);**

Description A triangle is a plane created by connecting three points. The first two arguments specify the first point, the middle two arguments specify the second point, and the last two arguments specify the third point.

Syntax

**triangle(x1, y1, x2, y2, x3, y3)**

Parameters

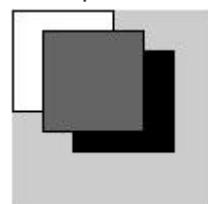
x1 float: x-coordinate of the first point  
y1 float: y-coordinate of the first point  
x2 float: x-coordinate of the second point  
y2 float: y-coordinate of the second point  
x3 float: x-coordinate of the third point  
y3 float: y-coordinate of the third point

Returns void

Name

**popMatrix()**

Examples



```
fill(255);
rect(0, 0, 50, 50); // White rectangle

pushMatrix();
translate(30, 20);
fill(0);
rect(0, 0, 50, 50); // Black rectangle
popMatrix();

fill(100);
rect(15, 10, 50, 50); // Gray rectangle
```

Description Pops the current transformation matrix off the matrix stack. Understanding pushing and popping requires understanding the concept of a matrix stack. The **pushMatrix()** function saves the current coordinate system to the stack and **popMatrix()** restores the prior coordinate system. **pushMatrix()** and **popMatrix()** are used in conjunction with the other transformation functions and may be embedded to control the scope of the

transformations.

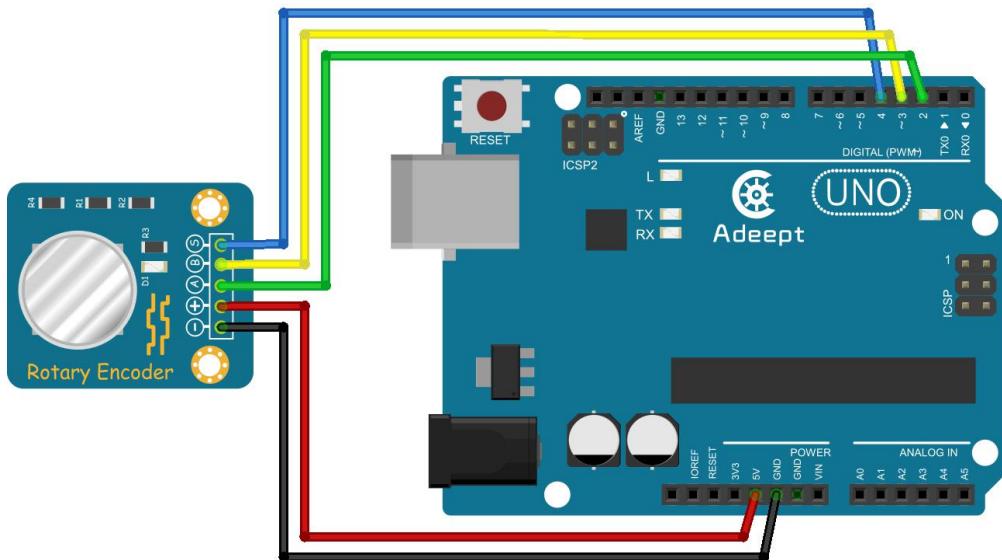
Syntax

**popMatrix()**

Returns void

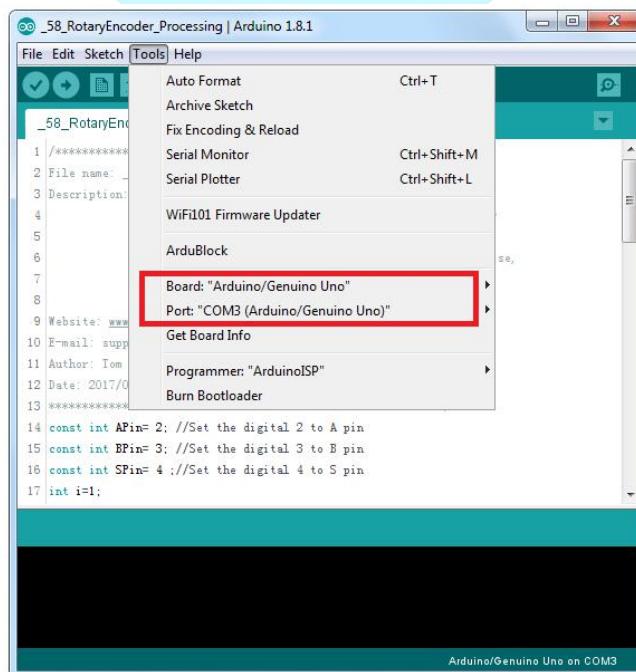
## Experimental Procedures

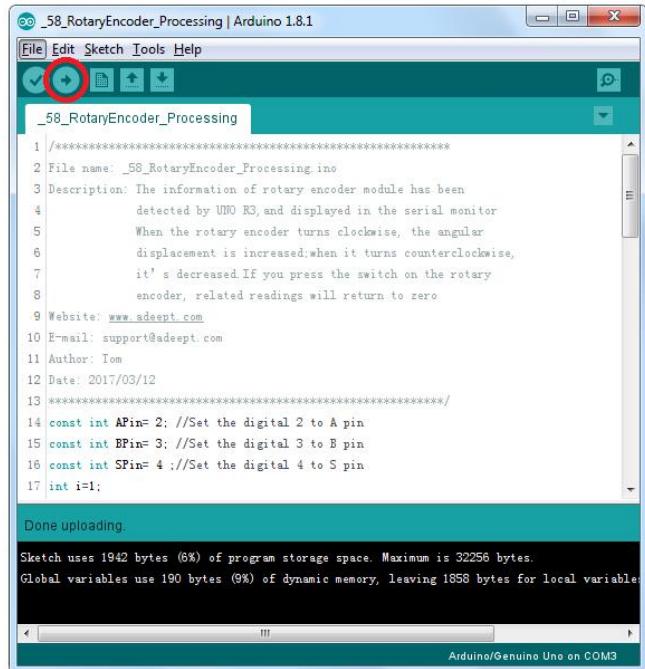
**Step 1:** Build the circuit



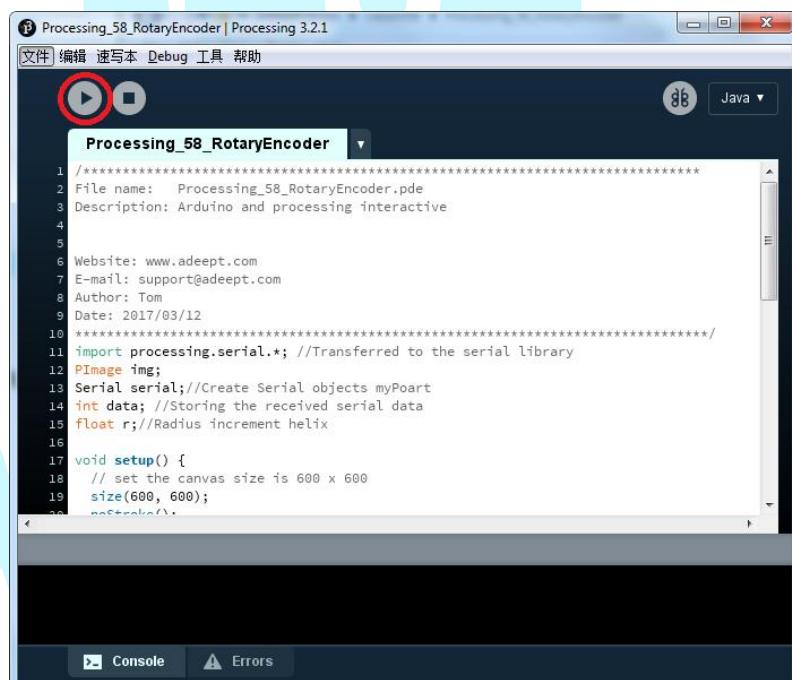
**Step 2:** Program \_58\_RotaryEncoder\_Processing.ino

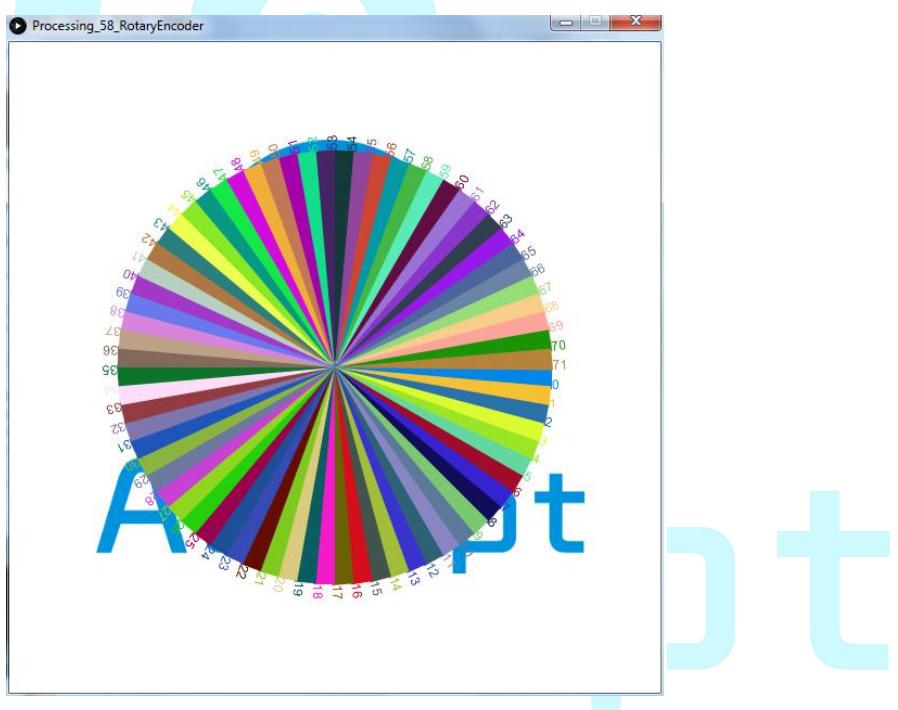
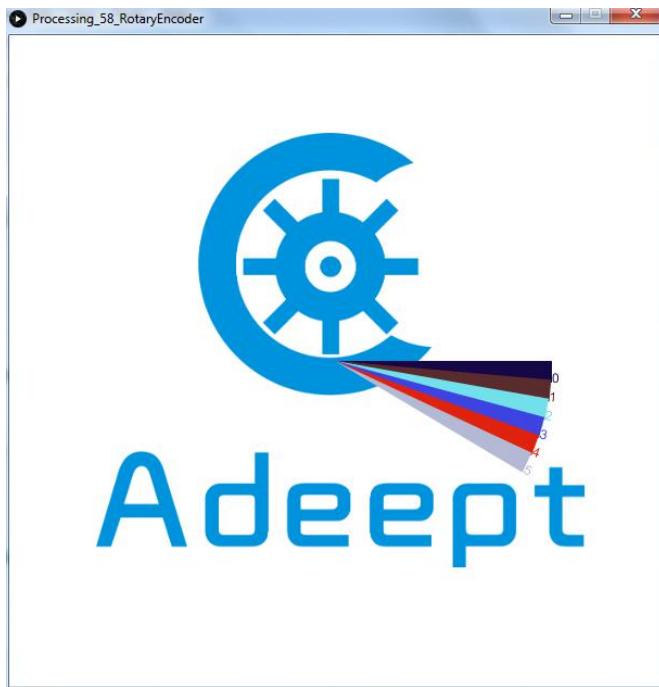
**Step 3:** Compile and download the sketch to the UNO R3 board.

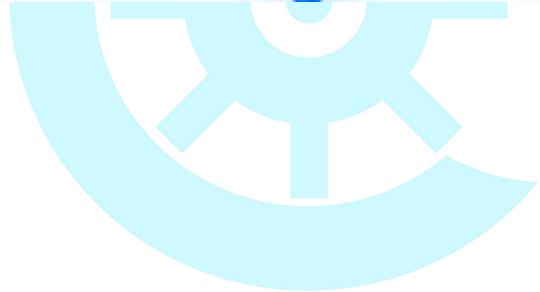
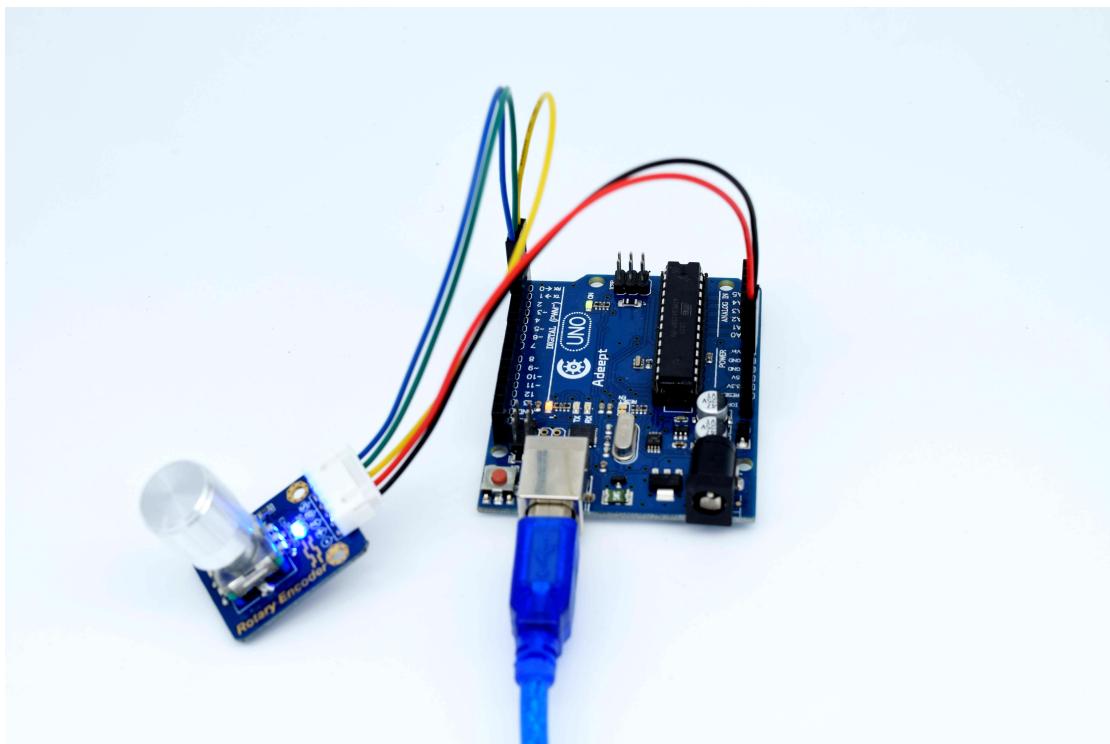




#### Step 4: Run the Processing software (Processing\_58\_RotaryEncoder.pde)







# Adeept

# Lesson 59 Arduino Interacts with Processing(Joystick Module)

## Introduction

In this lesson, we will collect the state of a joystick by programming the Arduino UNO Board, and then send the data to the Processing through the serial communication.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Joystick Module
- 1 \* USB Cable
- 1 \* 5-Pin Wires

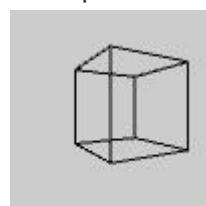
## Principle

Processing key function:

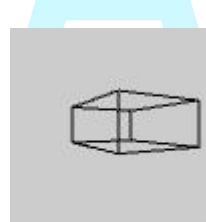
Name

**box()**

Examples



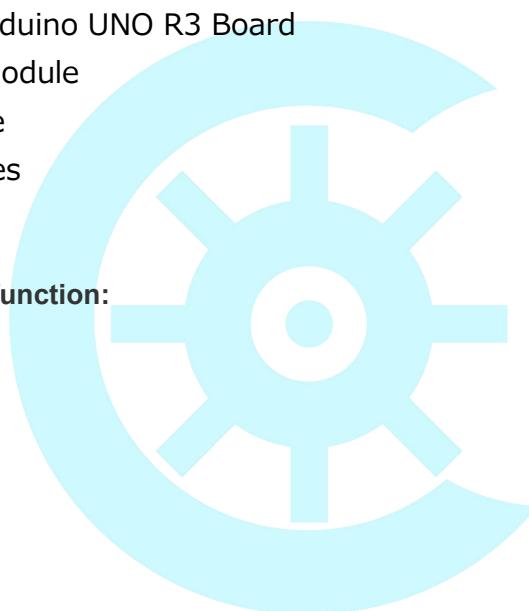
```
size(100, 100, P3D);
translate(58, 48, 0);
rotateY(0.5);
noFill();
box(40);
```



```
size(100, 100, P3D);
translate(58, 48, 0);
rotateY(0.5);
noFill();
box(40, 20, 50);
```

Description A box is an extruded rectangle. A box with equal dimensions on all sides is a cube.

Syntax



Adeept

**box(size)**

**box(w, h, d)**

Parameters

size float: dimension of the box in all dimensions (creates a cube)

w float: dimension of the box in the x-dimension

h float: dimension of the box in the y-dimension

d float: dimension of the box in the z-dimension

Returns void

Name

**split()**

Examples

```
String men = "Chernenko, Andropov, Brezhnev";
String[] list = split(men, ',');
// list[0] is now "Chernenko", list[1] is "Andropov"...
String numbers = "8 67 5 309";
int[] nums = int(split(numbers, ' '));
// nums[0] is now 8, nums[1] is now 67...
String men = "Chernenko ] Andropov ] Brezhnev";
String[] list = split(men, " ] ");
// list[0] is now "Chernenko", list[1] is "Andropov"...
```

Description The `split()` function breaks a `String` into pieces using a character or string as the delimiter. The `delim` parameter specifies the character or characters that mark the boundaries between each piece. A `String[]` array is returned that contains each of the pieces.

If the result is a set of numbers, you can convert the `String[]` array to a `float[]` or `int[]` array using the datatype conversion functions `int()` and `float()`. (See the second example above.)

The `splitTokens()` function works in a similar fashion, except that it splits using a range of characters instead of a specific character or sequence.

Syntax

**split(value, delim)**

Parameters

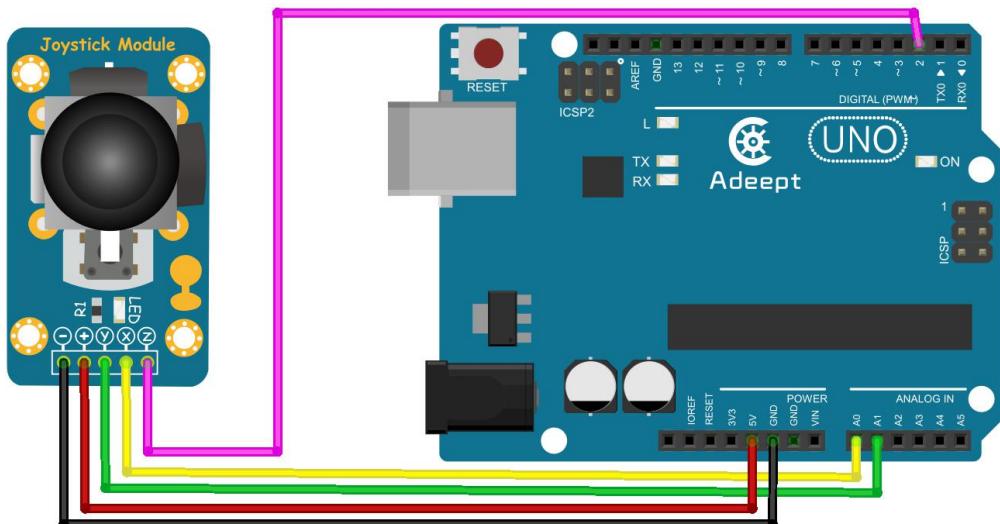
value String: the `String` to be split

delim char: the character or `String` used to separate the data

Returns `String[]`

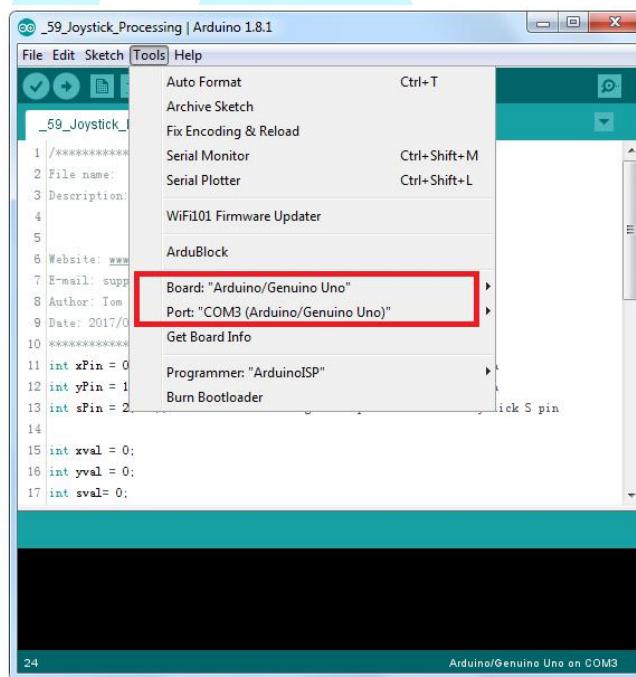
## Experimental Procedures

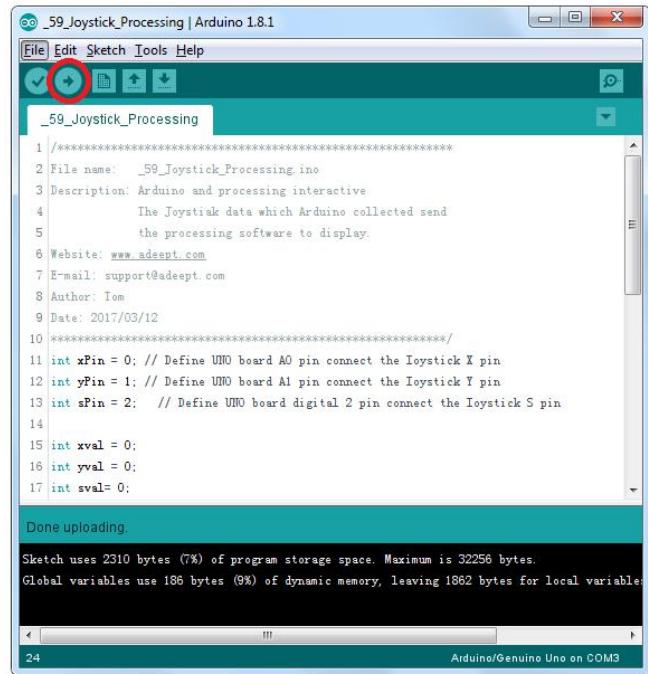
### Step 1: Build the circuit



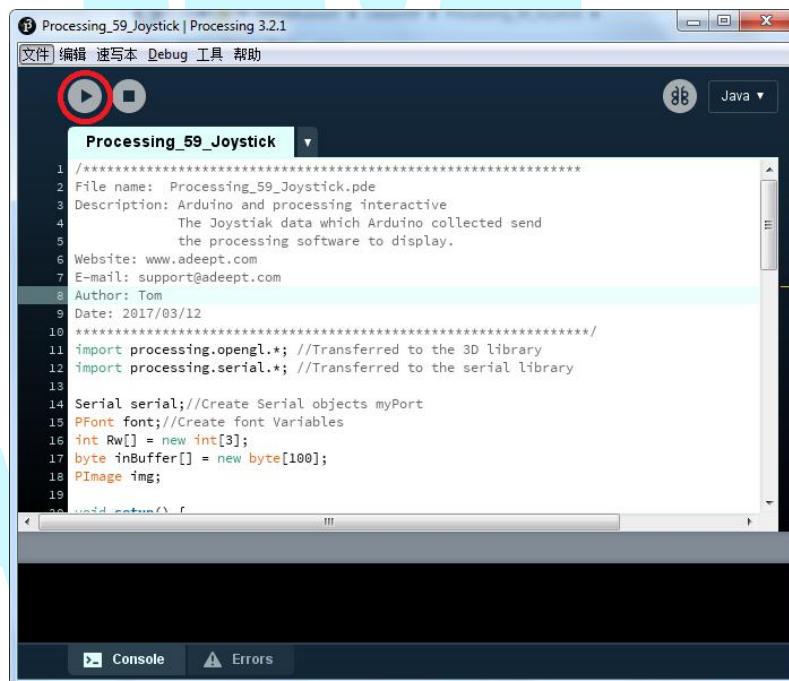
**Step 2:** Program `_59_Joystick_Processing.ino`

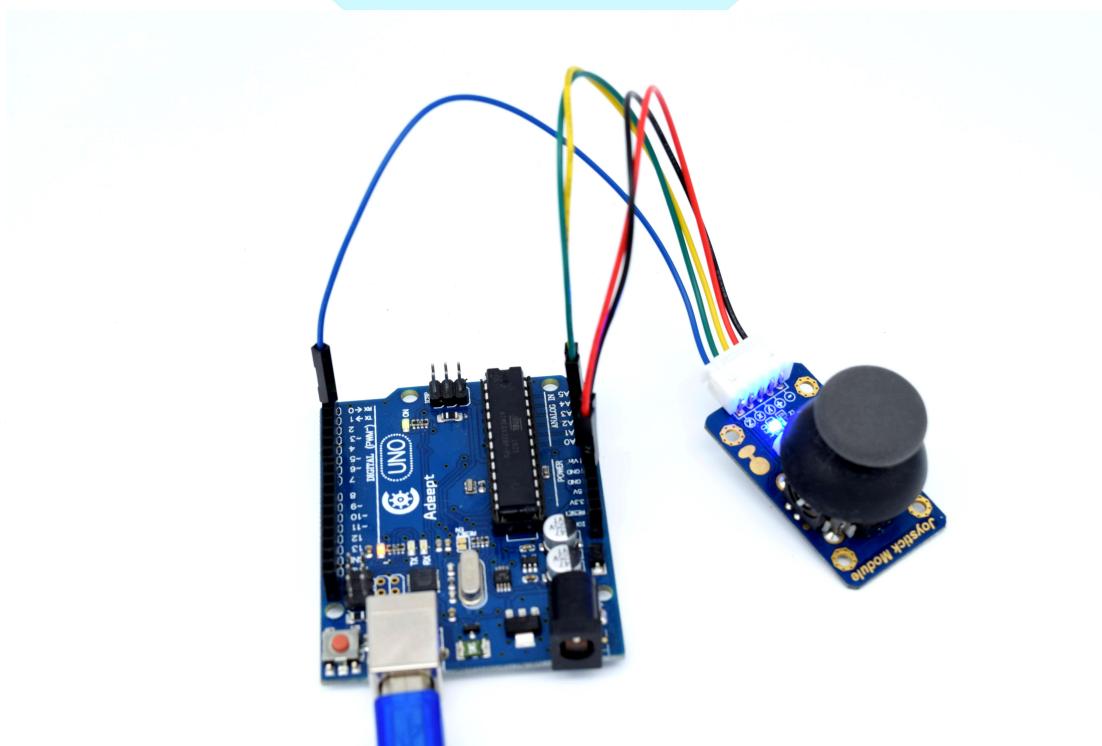
**Step 3:** Compile and download the sketch to the UNO R3 board.





#### Step 4: Run the Processing software (Processing\_59\_Joystick.pde)





# Lesson 60 Arduino Interacts with Processing(Ultrasonic Distance Module)

## Introduction

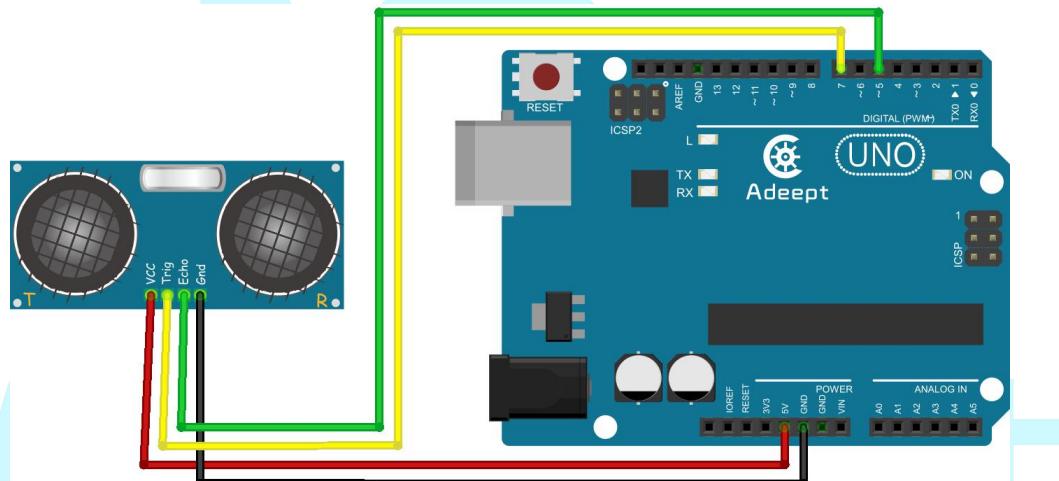
In this lesson, Arduino detects the distance between the ultrasonic module and the front obstacle through the ultrasonic module, the smaller the distance, the greater the radius of the circle in the Processing interface.

## Components

- 1 \* Adeept Arduino UNO R3 Board
- 1 \* Ultrasonic Distance Module
- 1 \* USB Cable
- 1 \* 4-Pin Wires

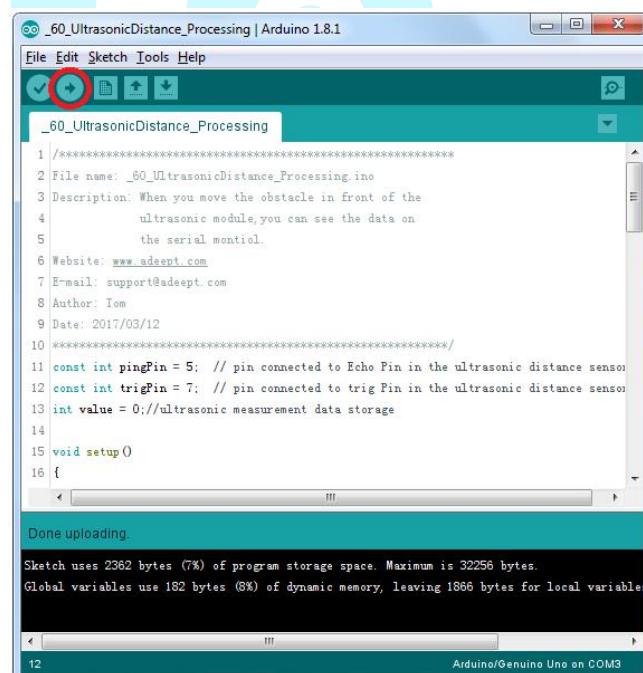
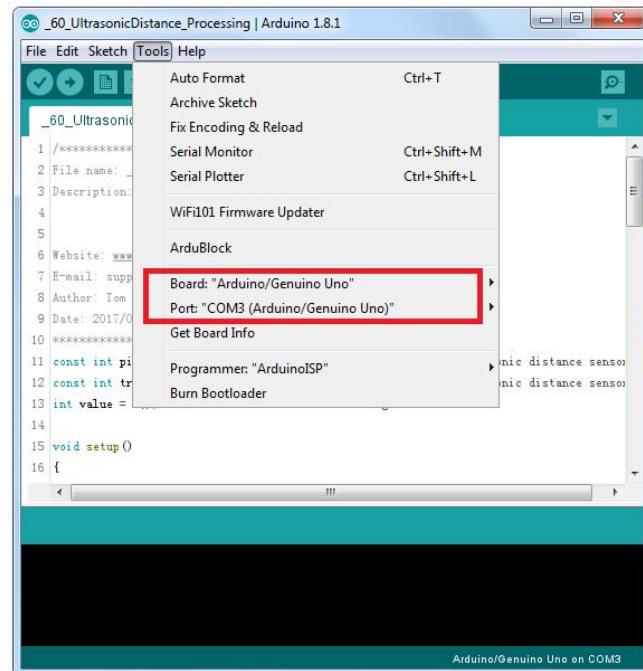
## Experimental Procedures

### Step 1: Build the circuit

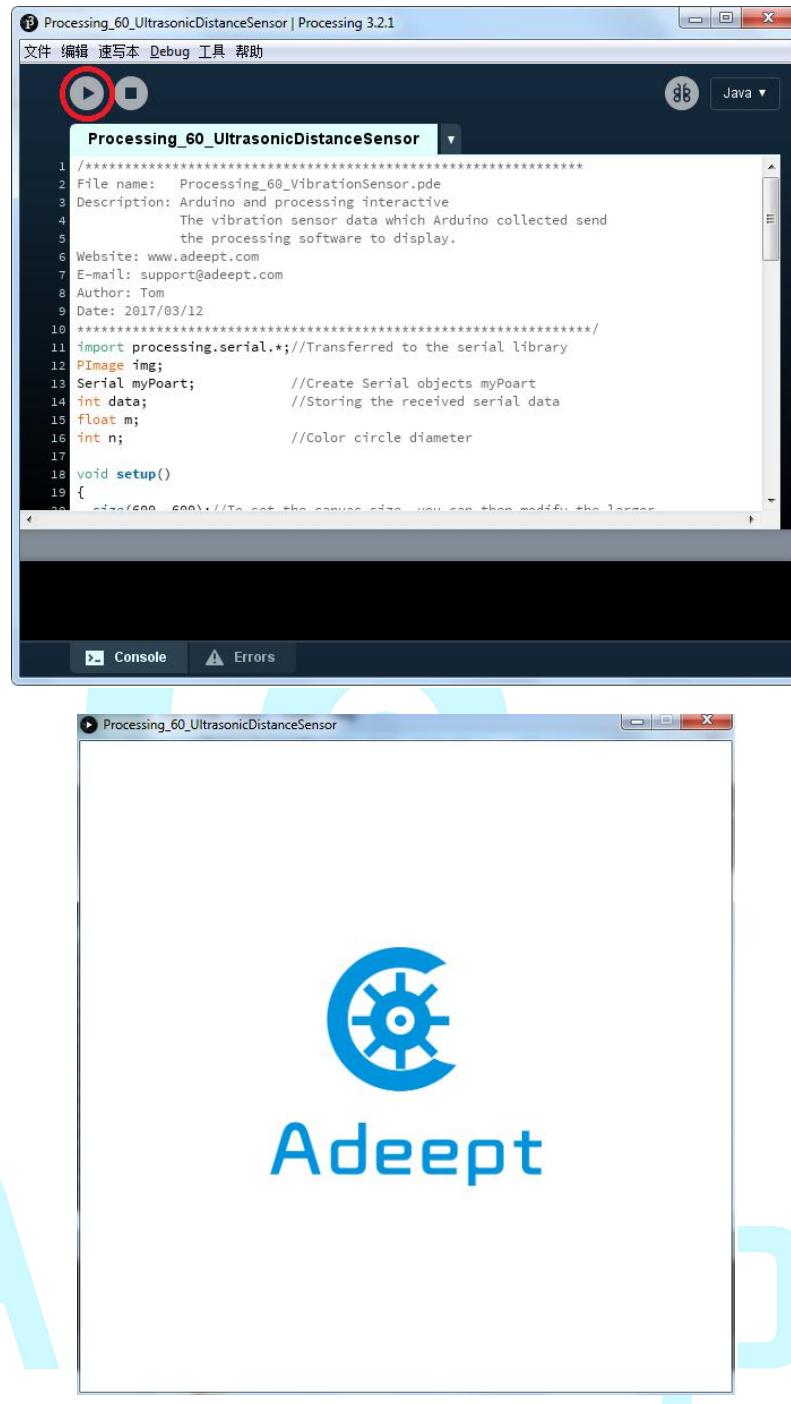


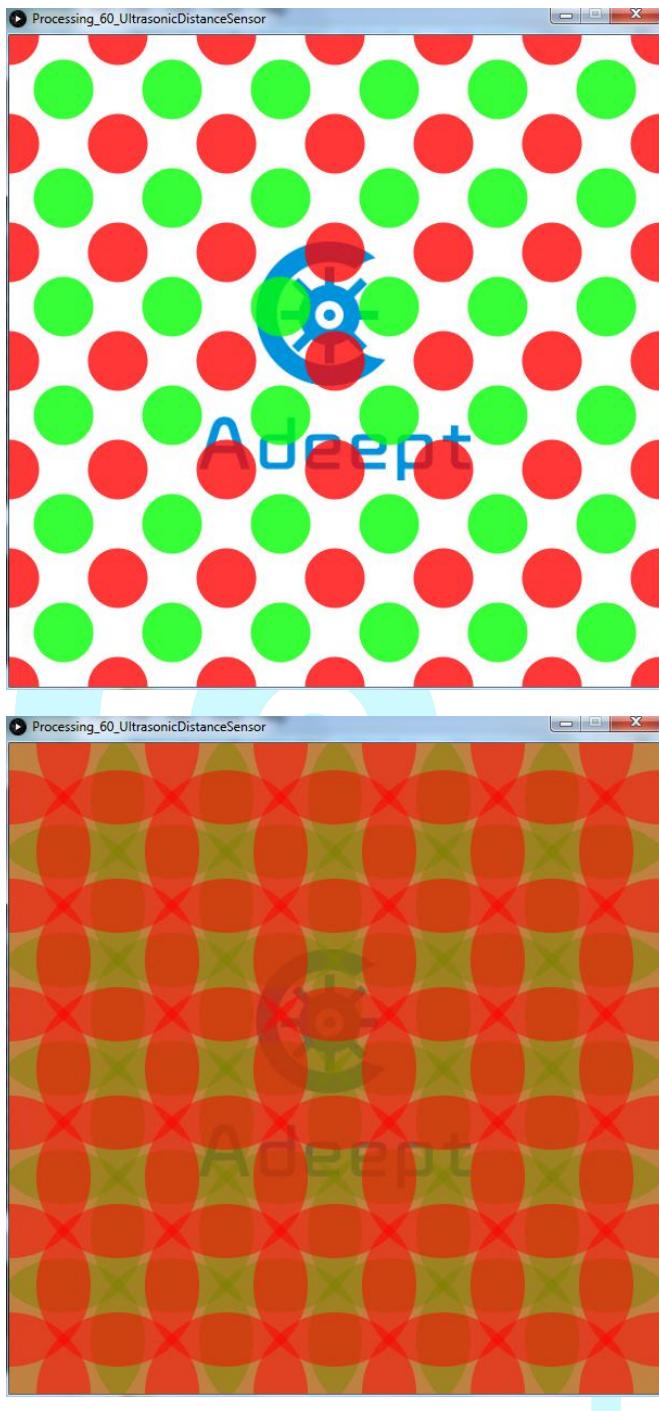
**Step 2:** Program `_60_UltrasonicDistance_Processing.ino`

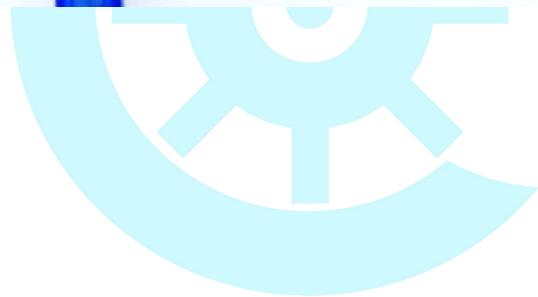
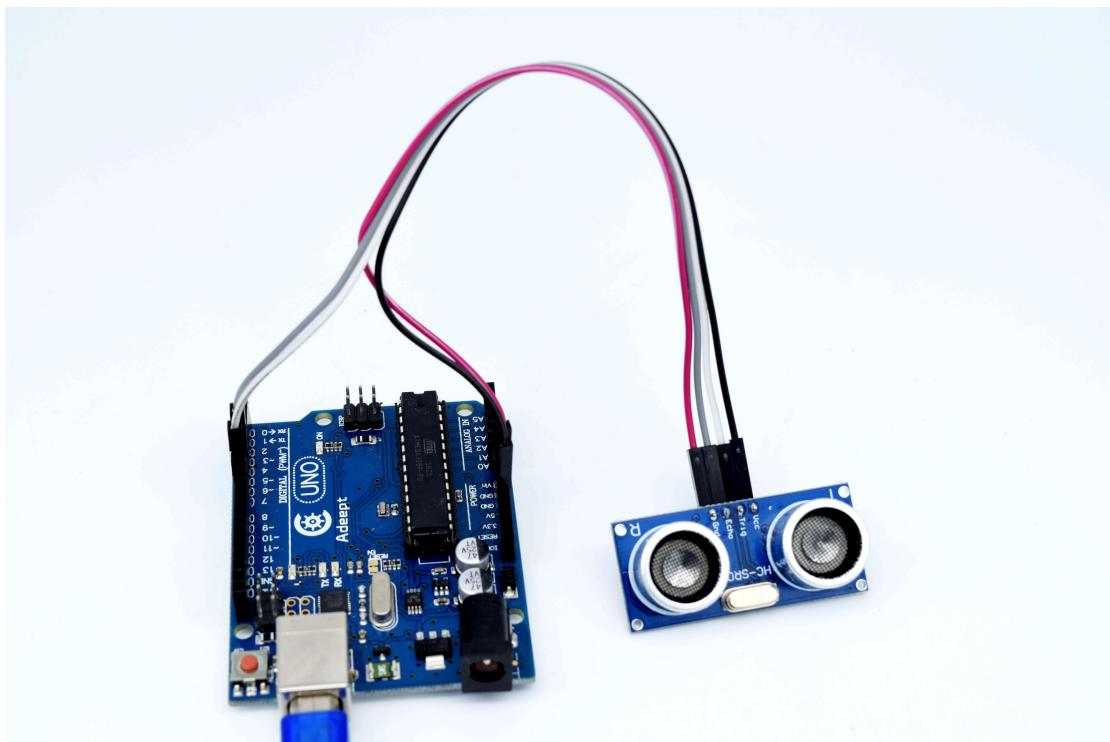
**Step 3:** Compile and download the sketch to the UNO R3 board.



**Step 4:** Run the Processing software (Processing\_60\_UltrasonicDistanceSensor.pde)







# Adeept

## Afterword

Thanks for purchasing our product and reading the manual! If you spot any errors or have any ideas or questions for the product and this guide, welcome to contact us! We will correct them if any as quickly as possible.

After completing all projects in the guide, you should have some knowledge of the book and Arduino, thus you can try to change the car into other projects by adding more Adeept modules or changing the code for extended functions.

For more information about Arduino, Raspberry Pi, smart car robot, or robotics, etc., please follow our website [www.adeept.com](http://www.adeept.com). We will introduce more cost-effective, innovative and intriguing products!

Thanks again for choose Adeept product!



# Adeept



Adeept

E-mail: support@adeept.com  
website: www.adeept.com