

---

## Content

<b>Project 0 Get Known to Arduino &amp; Mind+</b> .....	<b>1</b>
What is Arduino?.....	1
History of Arduino.....	1
Arduino UNO.....	2
Explore the Secret of Mind+.....	2
Get Started with Mind+.....	3
Download a “Blinking” Program.....	6
<b>Project 1 LED Blinking</b> .....	<b>10</b>
Let's get started!.....	10
Components.....	10
Graphical Programming.....	12
Command Learning.....	13
Code Learning.....	14
Add-activities.....	16
Hardware Review.....	16
<b>Project 2 S.O.S Distress Signal</b> .....	<b>22</b>
Graphical Programming.....	23
Command Learning.....	25
Code Learning.....	25
Add-activities.....	27
<b>Project 3 Interactive Traffic Lights</b> .....	<b>29</b>
Components.....	29
Hardware Connection.....	29
Graphical Programming.....	30
Code Learning.....	37
Hardware Review.....	43
Add-activities.....	45
<b>Project 4 Breathing LEDs</b> .....	<b>47</b>
Components.....	47
Hardware Connection.....	47
Graphical Programming.....	47

Command Learning.....	48
Code Learning.....	49
Add-activities.....	51
<b>Project 5 Colorful RGB LED.....</b>	<b>53</b>
Components.....	53
Hardware Connection.....	53
Graphical Programming.....	53
Command Learning.....	55
Hardware Review.....	57
Add-activities.....	58
<b>Project 6 Alarm Device.....</b>	<b>60</b>
Components.....	60
Hardware Connection.....	60
Graphical Programming.....	61
Code Learning 1.....	61
Programming.....	62
Code Learning 2.....	63
Hardware Review.....	64
Add-activities.....	65
<b>Project 7 Temperature Alarm.....</b>	<b>66</b>
Components.....	66
Hardware Connection.....	66
Programming.....	66
Code Learning.....	69
The communication Partner of Arduino--Serial.....	69
Hardware Review.....	72
Add- activities.....	73
<b>Project 8 Tilt Sensor.....</b>	<b>75</b>
Components.....	75
Hardware Connection.....	75
Programming.....	76
Code Learning.....	76

---

Hardware Review.....	78
<b>Project 9 Light Sensitive LED.....</b>	<b>80</b>
Components.....	80
Hardware Connection.....	80
Programming.....	81
Code Learning.....	81
Hardware Review.....	82
<b>Project 10 Drive a Servo.....</b>	<b>83</b>
Components.....	83
Hardware Review.....	83
Programming.....	84
Command Learning.....	86
Code Learning.....	86
<b>Project 11 Controllable Servo.....</b>	<b>88</b>
Components.....	88
Hardware Connection.....	88
Programming.....	89
Code Learning.....	89
Hardware Review.....	90
<b>Project 12 Interactive RGB LED.....</b>	<b>92</b>
Components.....	92
Hardware Connection.....	92
Programming.....	92
<b>Project 13 DIY a Fan.....</b>	<b>94</b>
Components.....	94
Hardware Connection.....	94
Programming.....	95
Code Learning.....	96
Hardware Review.....	96
<b>Project 14 IR Remote Controlled LED.....</b>	<b>98</b>
Components.....	98
Hardware Connection.....	98

---

Programming.....	99
Components.....	100
Hardware Connection.....	101
Programming.....	101
Code Learning.....	102
Add-activities.....	103
<b>Project 15 IR Controlled 8-Segment Display.....</b>	<b>104</b>
Components 1.....	104
Hardware Connection.....	104
Programming 1.....	105
Hardware Review.....	107
Code Learning 1.....	108
Programming 2.....	109
Code Learning 2.....	110
Components 2.....	113
Programming 3.....	113
Code Learning 3.....	116
Add-activities.....	118

# Project 0 Get Known to Arduino & Mind+

## What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It is intended for artists, designers, students, hobbyists, professionals and anyone who has an interest in creating interactive projects.

Arduino can sense the environment by receiving inputs from sensors and affects its surroundings by controlling lights, motors and other actuators. The microcontroller is programmed by the Arduino programming language(based on Wiring) and the Arduino development environment(based on Processing). It can run independently or communicate with other software such as Flash, Processing, MaxMSP and more. Arduino IED is an open source design so everyone can download and share thousands of projects for free!

Here are some Arduino projects to help you better understand what you can do with Arduino :

- Make a sound notification when coffee is done
- Get email alerts on mobile
- Blinking fluffy toys
- Steampunk Professor X Wheelchair with voice recognition and drink serving function
- A *Star War* arm gun
- A pulse monitor to record data when biking
- Robots that can run and draw pictures on the floor

## History of Arduino

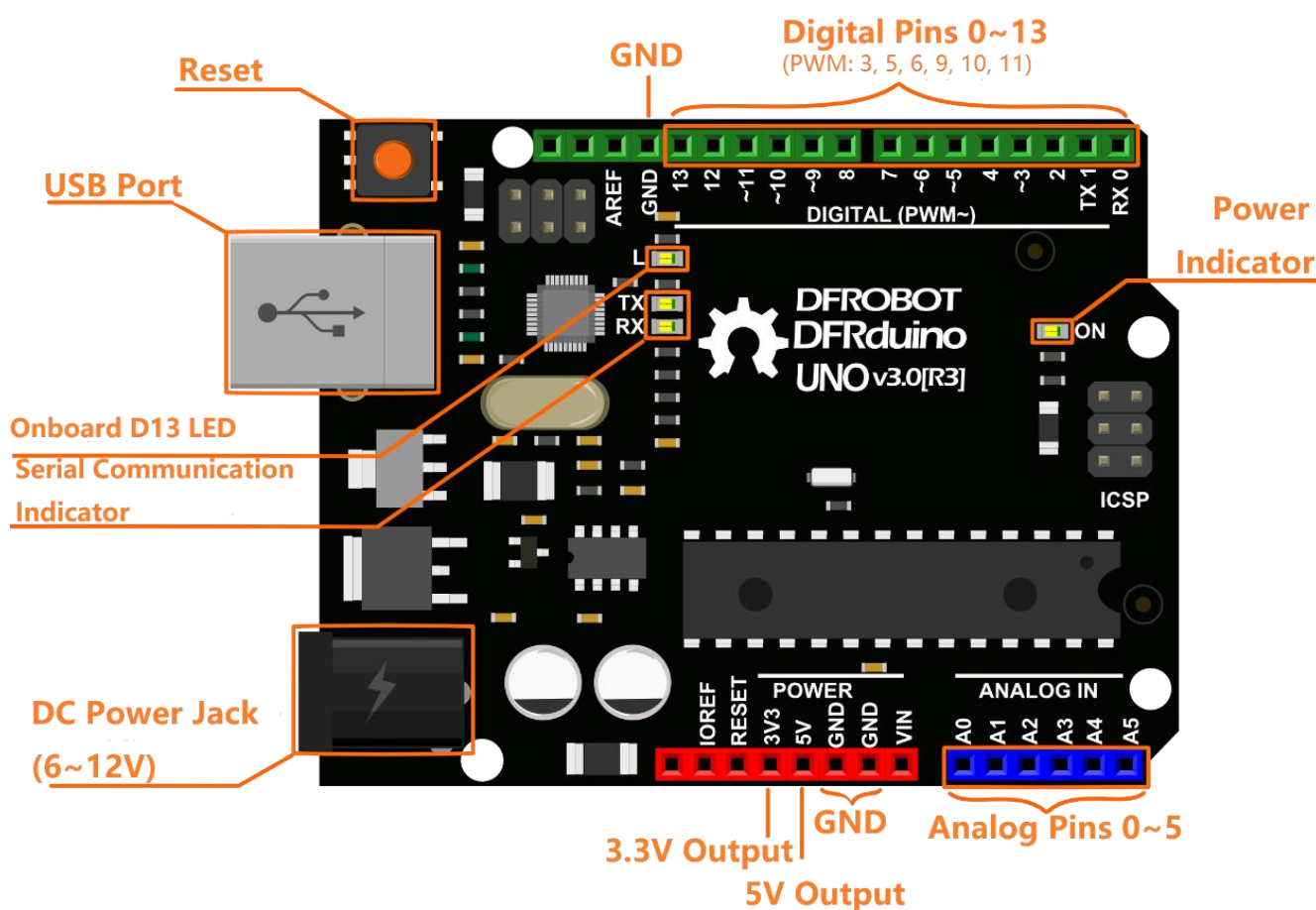
The Arduino project started in 2005 as a program for students at the Interaction Design Institute Ivrea, Italy, aiming to provide a low-cost and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. The name Arduino comes from a bar in Ivrea, Italy, where some of the founders of the project used to meet. The bar was named after Arduin of Ivrea, who was the margrave of the march of Ivrea

and King of Italy from 1002 to 1014.

## Arduino UNO

Now let's take a look at the Arduino UNO board. The commonly-used components have been marked in the figure below. It features:

- ❖ I/O pins, digital 0~13, analog 0~5
- ❖ Supply power from USB connection or DC power jack 6~12V
- ❖ 4 LEDs and 1 reset button. ON->Power indicator; L->Digital P13's LED; TX/RX->Serial Communication indicator, keep flashing when downloading programs.



## Explore the Secret of Mind+

With a USB cable, we can easily connect an Arduino board with a computer physically, however, that's not enough. Just like when we bought a new computer, without any software, we cannot make it start to work right away as soon as we installed all the hardware parts together. How to make the Arduino board communicate with your computer to begin your

journey of creation?

Well, Mind+ can do that. Mind+ likes a bridge that connects microcontroller with PC, and then achieve the functions such as code downloading, serial port connection, and data transmission.

The first few projects in this tutorial give detailed introductions about graphical programming, which helps you understand how to implement your idea in coding. Once getting familiar with the basic commands, you can try designing your unique program and typing it into Mind+ to see the results. After that, we will walk you away from graphical programming and start pure-code learning. Embark on a coding journey and enjoy the charm of programming, we will support you all the way!

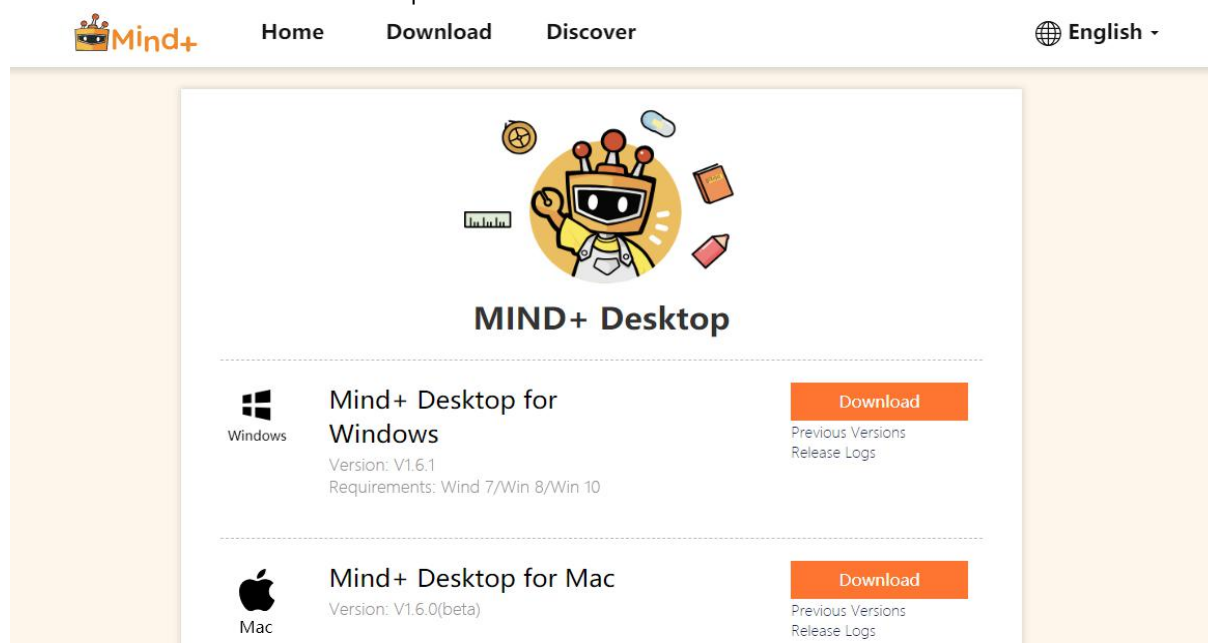
First of all, welcome to the world of Mind+!

## Get Started with Mind+

1. Download Mind+ at <http://mindplus.cc>.

Mind+ is a Scratch3.0-based graphical programming platform that supports all kinds of open source hardware like micro:bit, Arduino, and mPython. Users can drag and snap code blocks to make programs or use advanced language, such as python/c/c++ to code. It could be very easy to find the joy of creating.


- Download Mind+ Desktop



\* If you encounter problems in downloading or using Mind+, please visit our Mind+ official website to find the related solutions. If there is no suitable solution for your problem, please feed us back on our forum at <https://www.dfrobot.com/forum/>, our technique support group will help you as soon as possible.

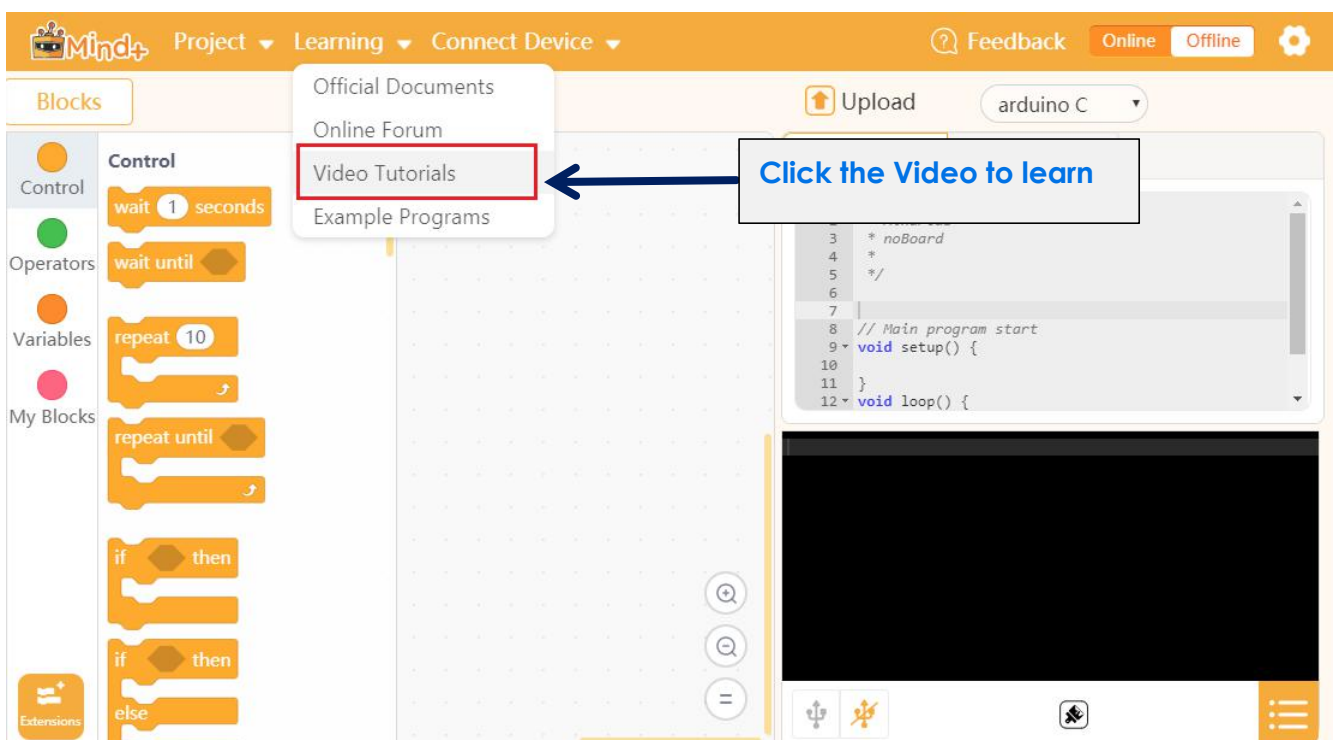
- Install Mind+

Name

 Mind+\_Win\_1.6.1.exe

## 2. Install the driver

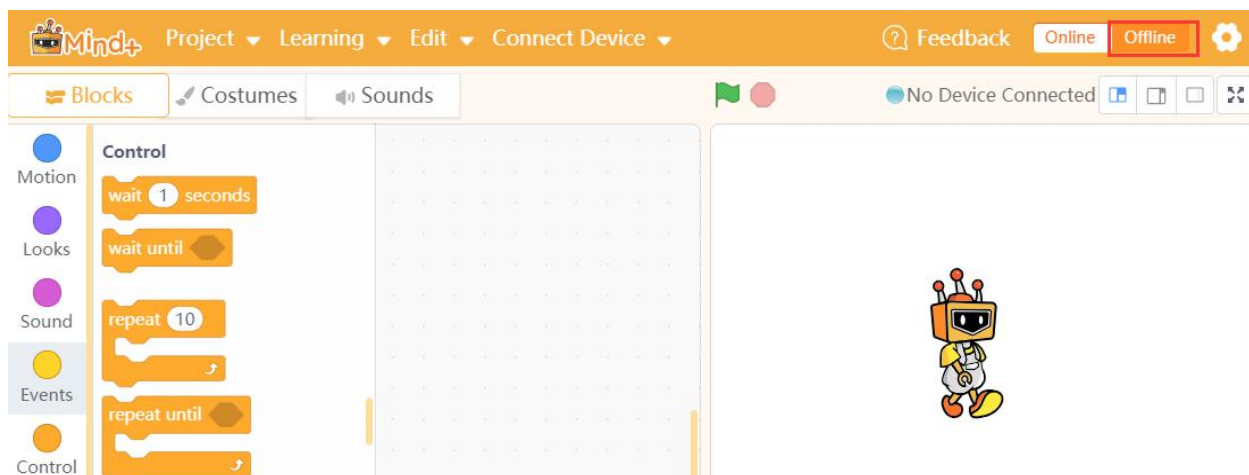
After you downloaded and installed the software, open and click "Learning" to learn how to install the driver.



## 3. Switch to "Offline" mode (Projects in the tutorial are based on Offline mode)

- Click the "Offline" icon to switch mode



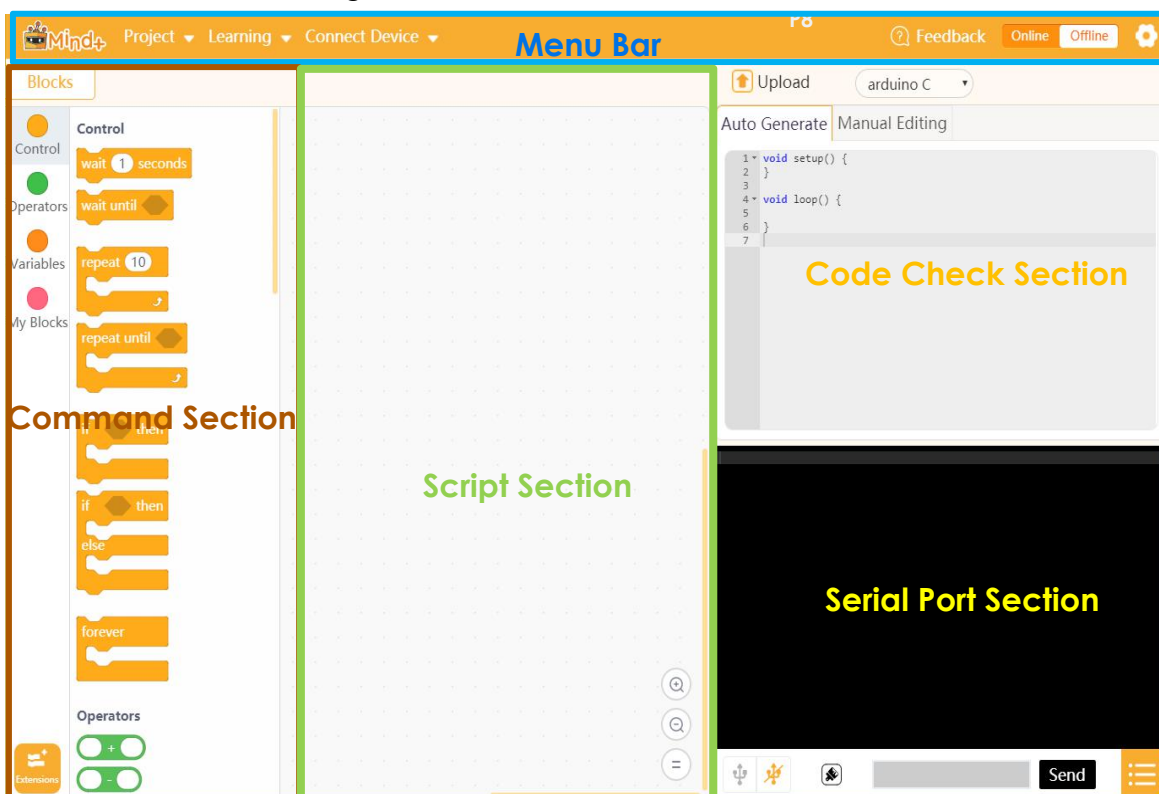


- Switched to Offline mode



#### 4. Introduction to Mind+ Interface

- Mind+ has following several function sections in Offline mode:



**Menu Bar**: the basic software setting functions and all relevant tutorials can be found here.

**“Project”**: create, load, save a project, etc.

**“Learning”**: video tutorials and example programs are here.

**“Connect Device”**: detect the connected device, display serial port, and select to connect or disconnect device.

**“Online/Offline”** Icon: switch mode to run your programs. Under Online mode, you have to upload the programs to your device to run them, while in Offline mode, the scripts can be directly executed when you use the device to interact with Mind+ stage.

**“ (Setting)”** button: set the software theme, language, etc; contact us to ask for help.

**Command Section**: there are many modules and command blocks, drag and snap these blocks logically to realize different functions. More extra functions can be found in “Extension”.

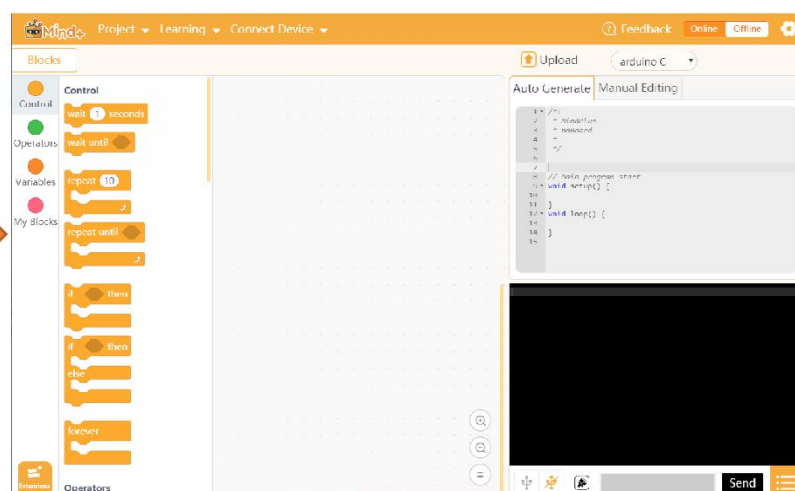
**Script Section**: graphical programming area; drag the blocks you need to here to combine them.

**Code Check Section**: the relevant code of the blocks you drag to the scripts section will be displayed here.

**Serial Port Section**: display the downloading status and serial communication data, check the execution of your programs. Besides, it also includes open/close serialport or scroll display, clear output, band rate setting, serialport input and output format control.

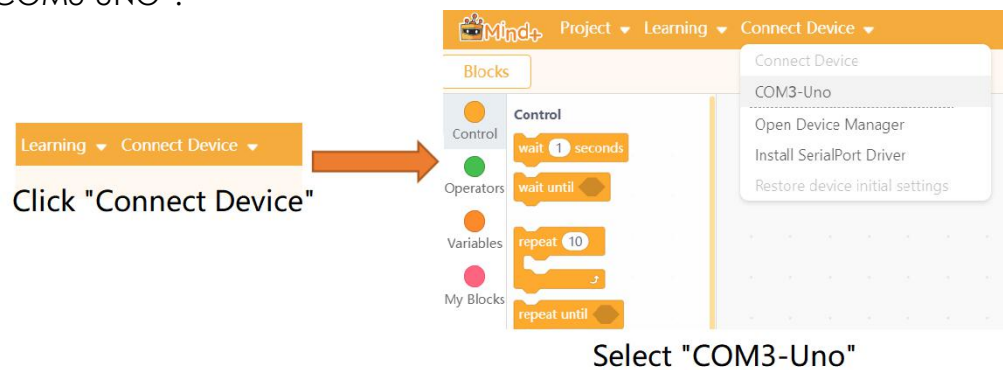
## Download a “Blinking” Program

**Step1**. Click the Mind+ icon to open the software, and switch to “Offline” mode.



Switch to "Offline" mode

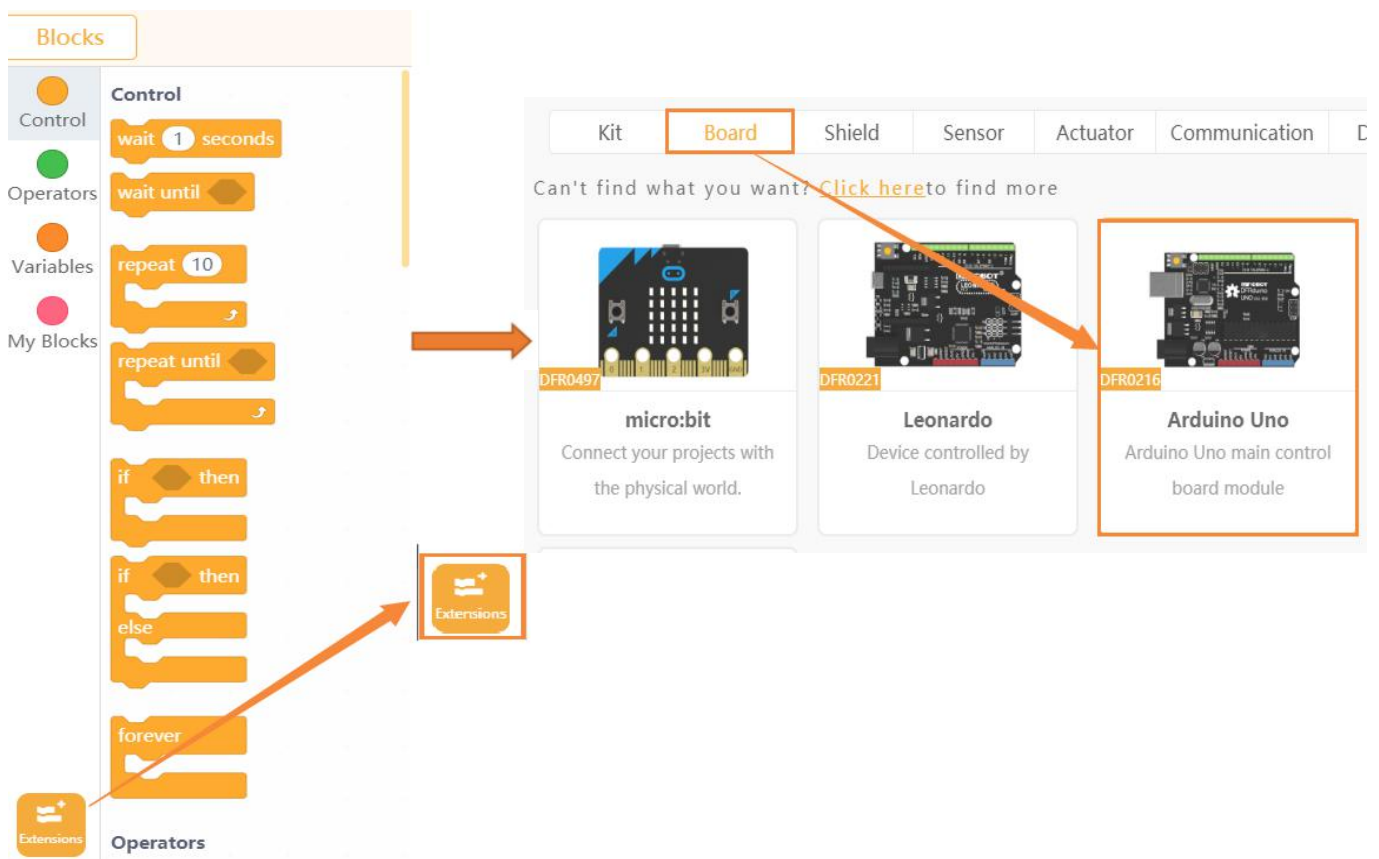
**Step 2.** Connect the Arduino board and your computer via a USB cable, then click "Connect Device" -> "COM3-UNO".



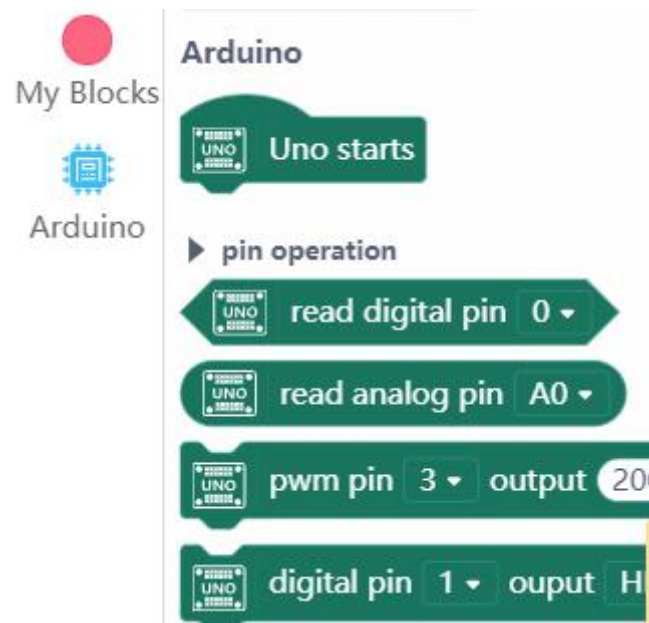
\* The number "3" in "COM3" may change according to different devices, and it has no real impact on your usage process.

\* If there is no COM port appearing on Mind+, please confirm the USB connection, check whether the power indicator on Arduino board is on, or the driver is installed. Feel free to contact us if you still can't fix the problem after trying all of this.

**Step 3.** Click the "Extension" at the lower-left corner, select "Board" -> "Arduino UNO"

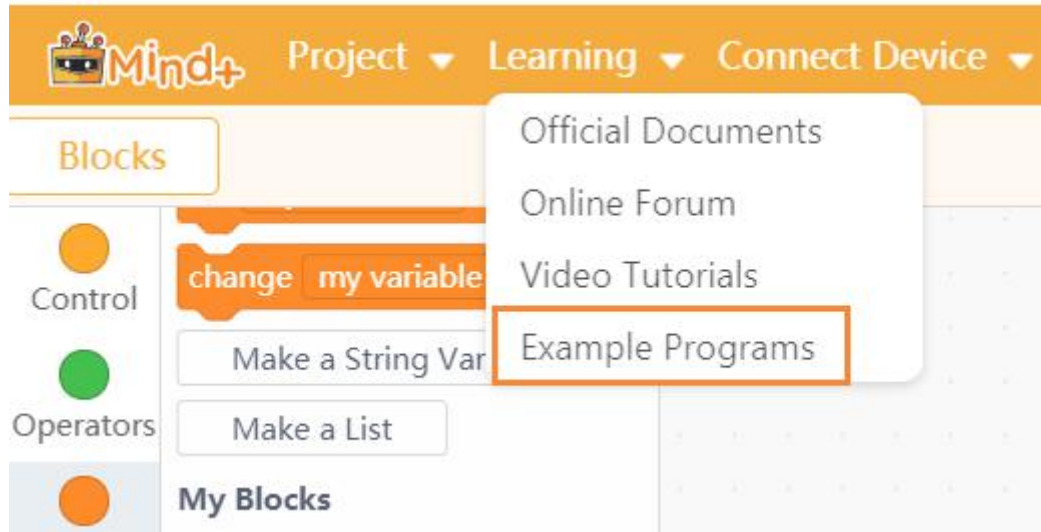


Click "Back", then you can see the Arduino Uno Blocks.

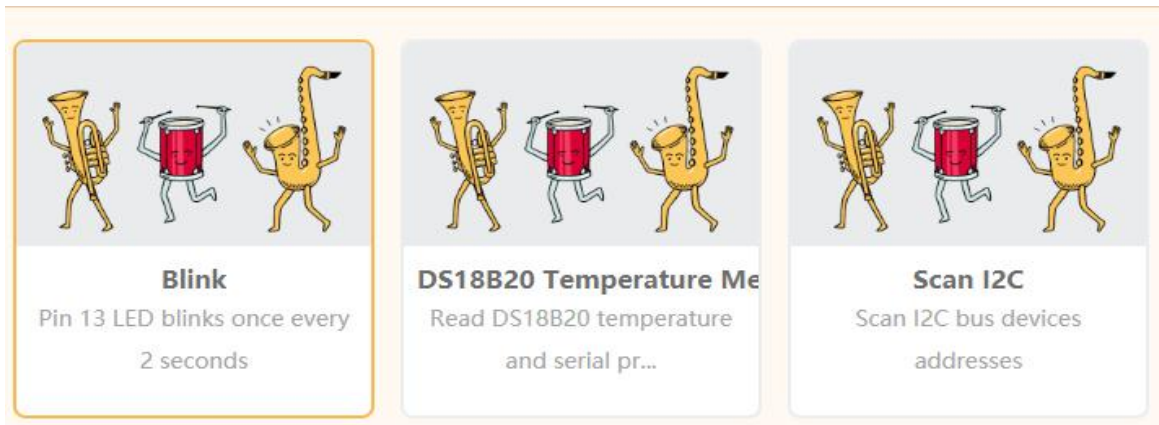


\* Don't forget to load Arduino Uno board in Extension every time you open this software. Otherwise, you can't find the related Arduino blocks.

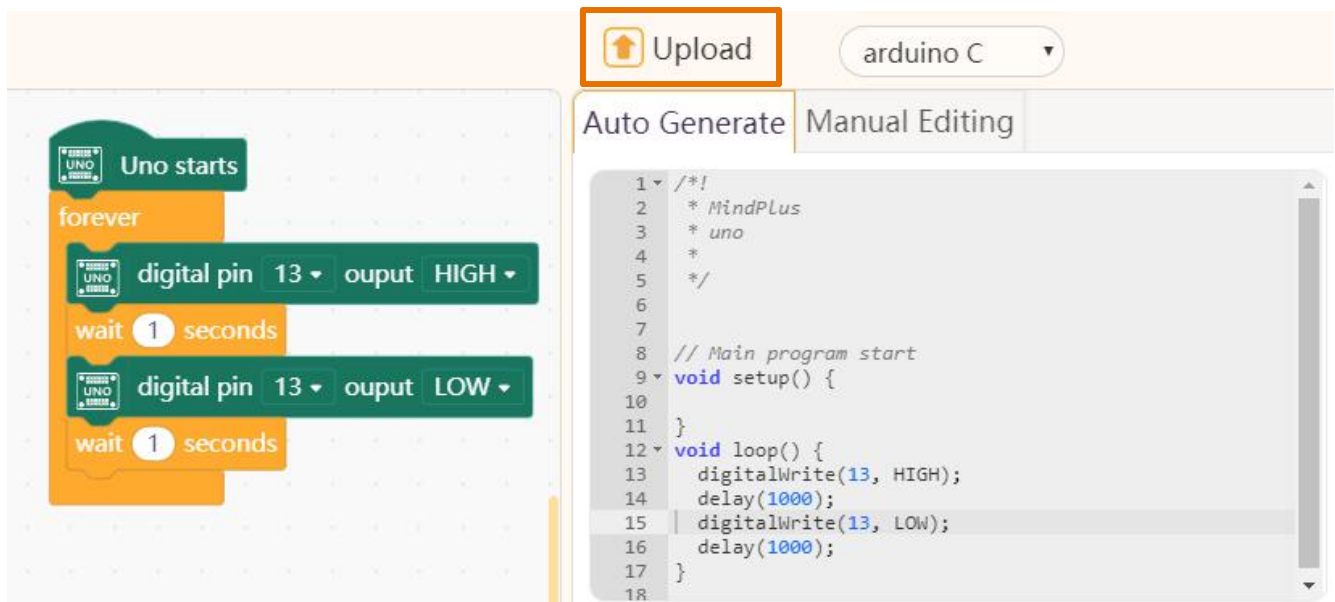
Step 4. Load programs into Mind+. Click "Learning"-> "Example Programs".



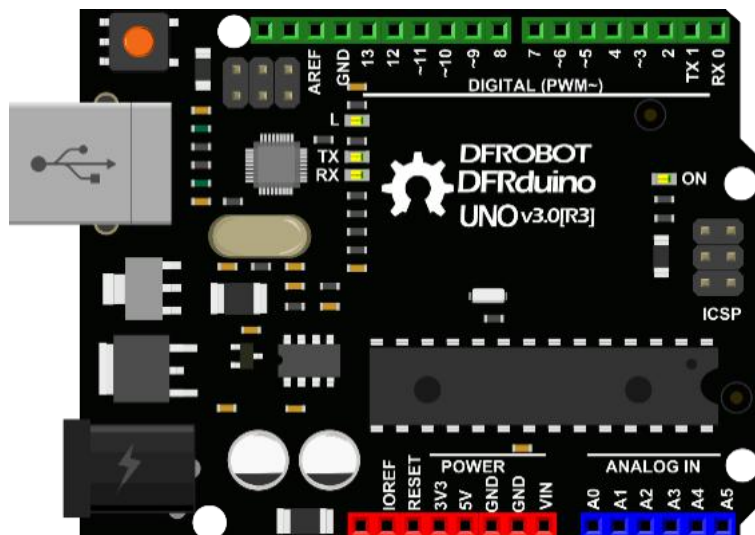
Click to select the first program "Blink".



The program has been preset in Mind+, now click “Upload “ to burn the codes into Arduino board.



Then, we will find that the LED marked with “L” on the Arduino board begins blinking.





# Project 1 LED Blinking

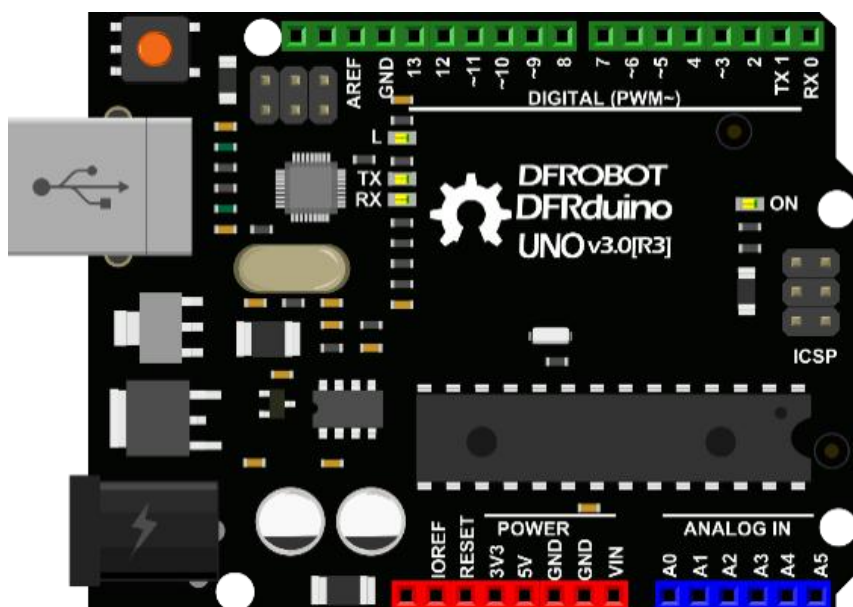
## Let's get started!

Step on the adventure of Arduino from an LED! In the first project, you will learn how to control the output of Arduino, and the basics of components such as LED, button, resistor, and pull-up/down resistor. At the same time, you will first come into contact with graphical programming. By comparing the code blocks and auto-generated programs, you will start to learn something about coding. Now, let's get started: use an Arduino board to control an external LED!

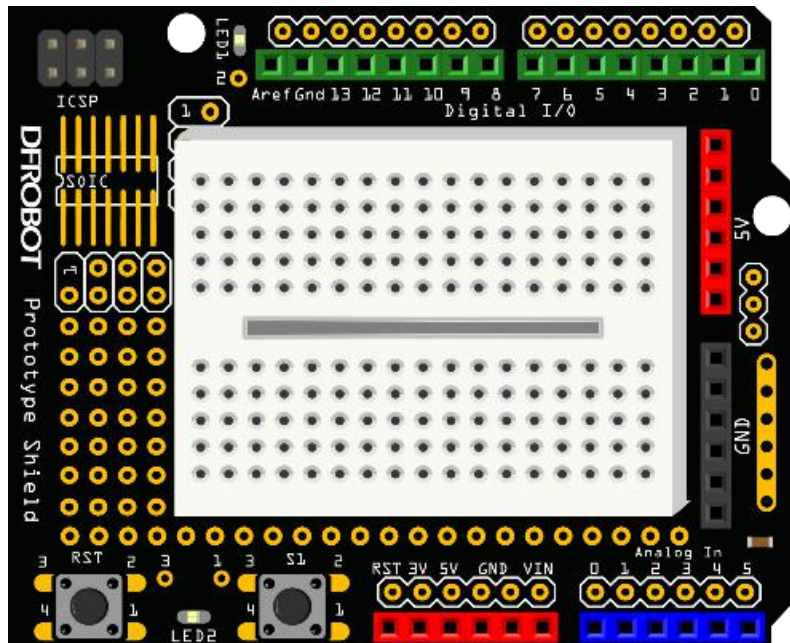
The test code "Blink" in the previous chapter will be used here. However, we are going to connect an external LED to a digital Pin instead of using the onboard Pin 13 LED, so we can learn the principle of LED flashing and circuit building.

## Components

- DFduino UNO R3 and USB cable × 1



- Prototype Shield + Breadboard × 1



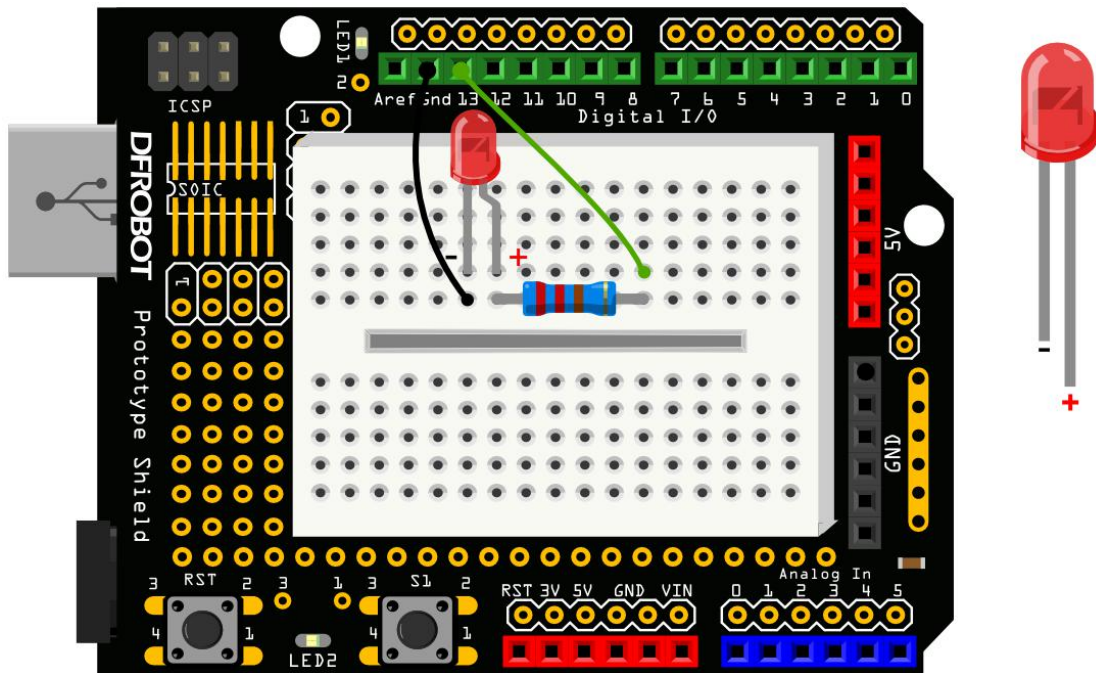
- Several Jumper Wires
- 5mm LED × 1
- 220Ω Resistor × 1

\* The resistor here is for reference only. The resistance of the resistor should be selected according to the LED solely used. We will discuss the details later.

### Hardware Connection

Peel the adhesive tape off the back of the breadboard and stick it on Prototype shield, and then plug the shield onto the UNO board. Connect all parts as the diagram shown below.

\* The UNO board you received may not be the same as the one shown below since it has multiple versions, so please confirm the silkscreen label of each port before connecting.



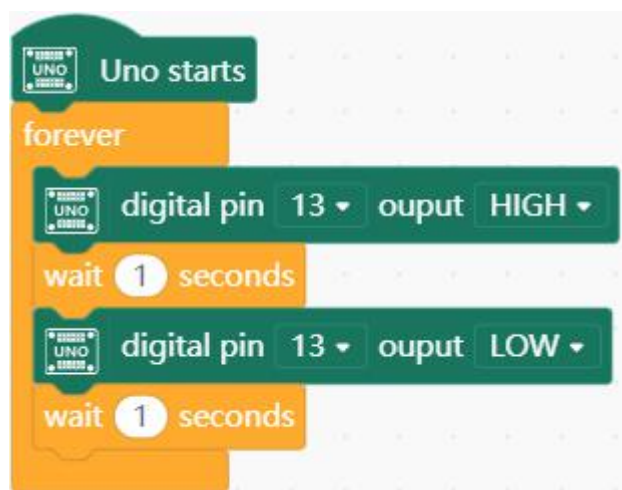
Use green and black jumper wires to connect the components. (Normally, for the DFRobot products, green is for digital port, blue for analog, red for VCC, black for GND.) You can use other holes in the breadboard, but please keep the connection order be the same as the picture above.

The long leg of LED is +, VCC; short leg is -, GND. Please connect the LED correctly. Power the Arduino via USB, and be ready to download codes.

## Graphical Programming

Open Mind+, load the Arduino UNO blocks. Connect your Arduino board to a computer, and then drag the blocks to the script section to complete a program as the example shows.

Example Program 1-1:













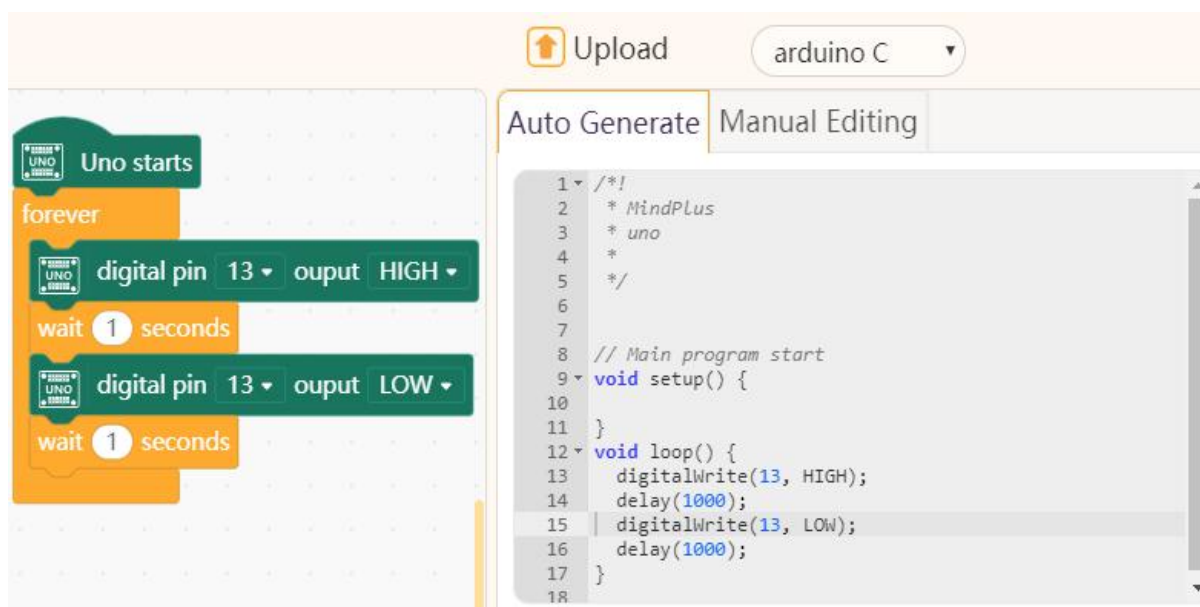
Click “  Upload ” to download the codes into Arduino.

Result: the red LED on the breadboard flashes once every 2 seconds.

Well, now let's learn the commands we used in the program.

## Command Learning

Module	Command	Description
 Arduino		Scripts placed underneath this block will activate after uploading to your device
 Control		Blocks held inside this block will be in a loop and never end
 Arduino		Set the relevant value to the voltage of the pin: high, 5V(or 3.3V for 3.3board); low, 0V
 Control		Delay and wait. Continue to execute the last command




We have realized the LED blinking via graphical programming. Now let's learn the auto-generated codes of this program.

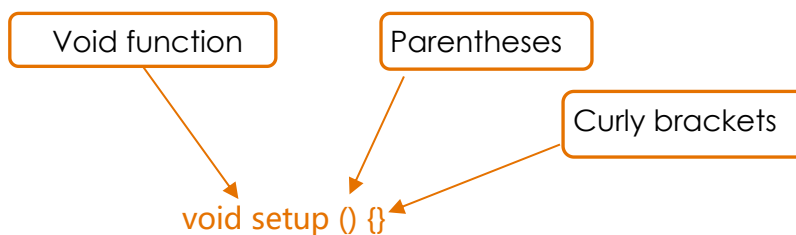
## Code Learning

Here are the C codes of the project, we will introduce the meaning of codes line by line.

```
1. void setup () {  
2. }  
3.  
4. void loop() {  
5.   digitalWrite (13, HIGH);  
6.   delay (1000);  
7.   digitalWrite (13, LOW);  
8.   delay (1000);  
9. }
```

### ▪ void setup() {}

This is a setup() function that corresponds with the block . The function format:



Any code that lives inside setup()'s curly brackets from "{" to "}" runs once at the very beginning of your program and then never runs again--at least not until you reset the Arduino, or upload new code. It is useful for initializing variables, pin modes, initialize libraries, etc.

## Function

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program. **The setup() and loop() cannot be called repeatedly**, and in Arduino sketch, other functions must be created outside the brackets of those two functions.

In addition, we have to comprehend the concept of the return of a function. A return is a value that a function returns to the calling script or function when it completes its task. It can be regarded as feedback. How do we know if the function will return a value? Well, the

function declaration can give us the answer. For instance, "void" is the signal that there is no return in this function. We will use it frequently later. How about function with a return? If you are interested, google it!

- `void loop() {}`--->



\*The functions `setup()` and `loop()` must be included in Arduino program, otherwise, it cannot work properly.

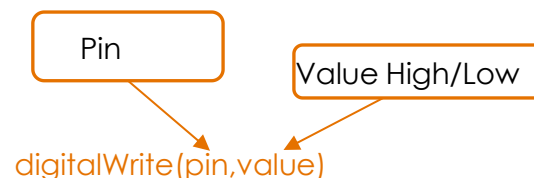
As the name implies, the `loop()` function...loops! The program starts directly after the opening curly bracket "{", runs until it sees the closing curly bracket "}", and jumps back up to the first statement in the `loop()` and starts all over. The function will run over-and-over until the Arduino is reset.

- `digitalWrite(13, HIGH)`

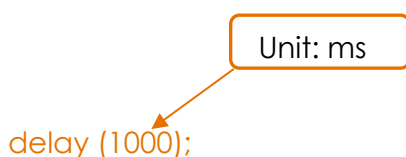
The function `digitalWrite(13, HIGH)` corresponds with the block



It can set the pin 13 to HIGH so as to light up P13 LED. The function format:



- `delay(1000)`



The block  is for the function `delay()`. The unit of the function is millisecond.

For example, if we want to delay 0.5s, the relevant code should be: `delay(500)`.

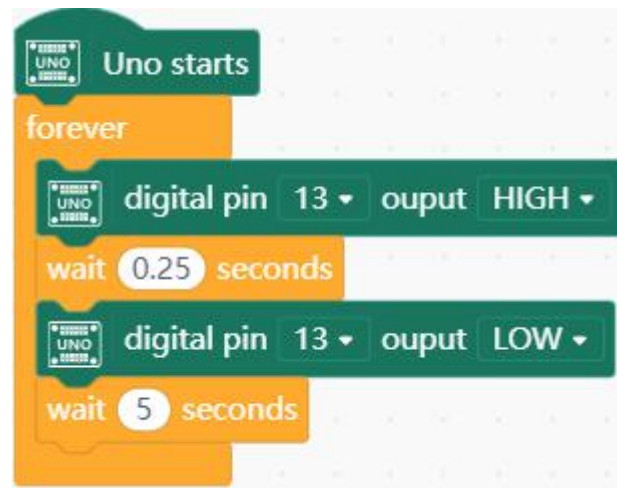
- `digitalWrite(13, LOW);`

With the guide above, it would be quite easy to understand this function, right! Apparently, it means set P13 to Low(0V) to make its LED go off.

## Add-activities

Since we have known how code works, let's change the project a little bit. Keep the LED off for 5s and then make it flash quickly(once every 0.25s), just like the light on the car alarm. Give it a try!

Answer:



Changing the time the LED keeps on or off is able to realize various lighting effects. Come and explore!

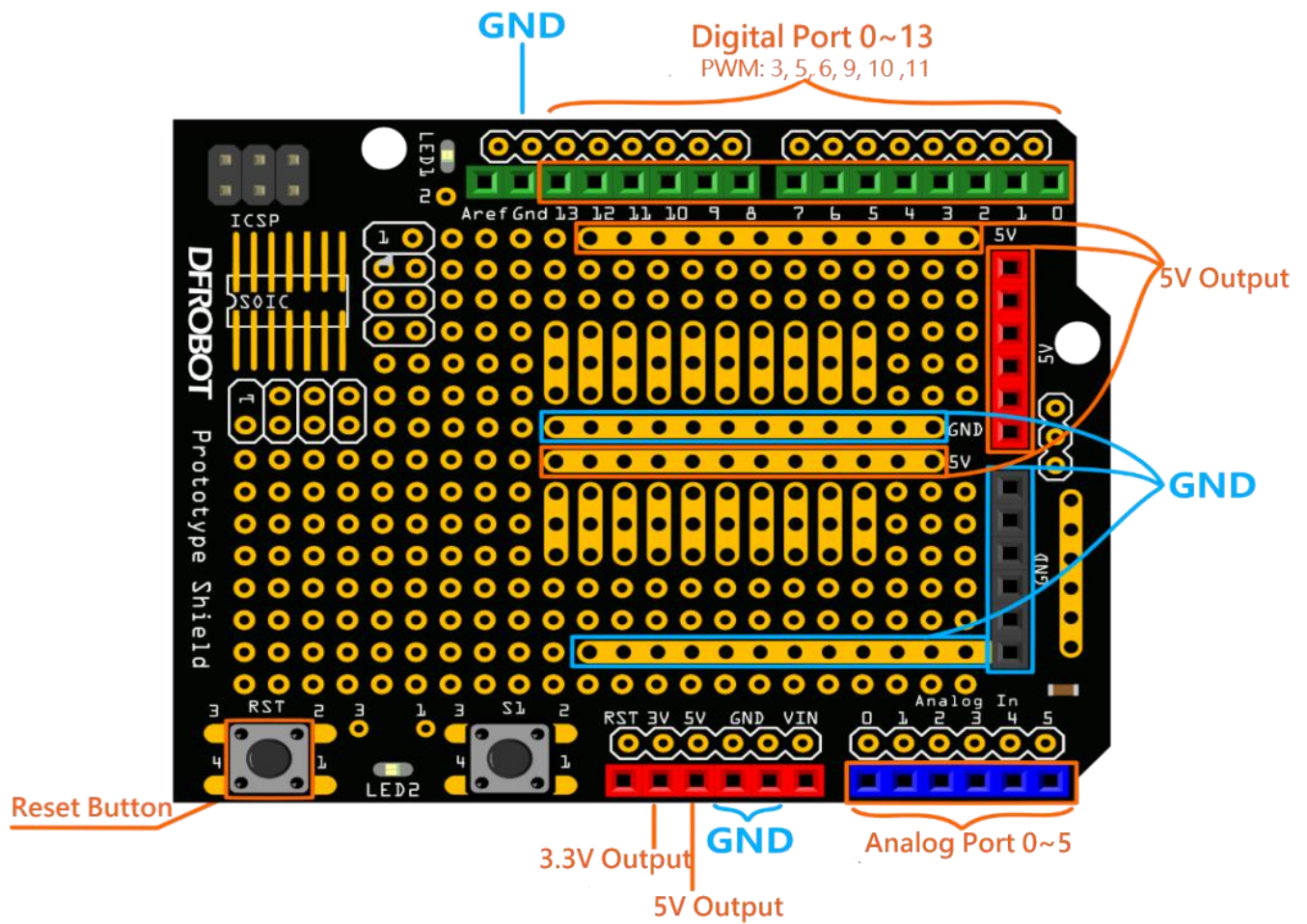
## Hardware Review

### Prototyping Shield

There are limited ports on an Arduino UNO board, especially the 5V and GND ports. Our project development may often be bogged down by the pin resource constraints. So most of the time we need a shield to expand the ports on Arduino board.

This prototyping shield is compatible with Arduino Uno, on which you can build circuits, solder components and so on. There is a breadboard on the shield for you to connect. Its digital and analog ports conform with UNO board. Besides, the 5V ports and GND ports marked below

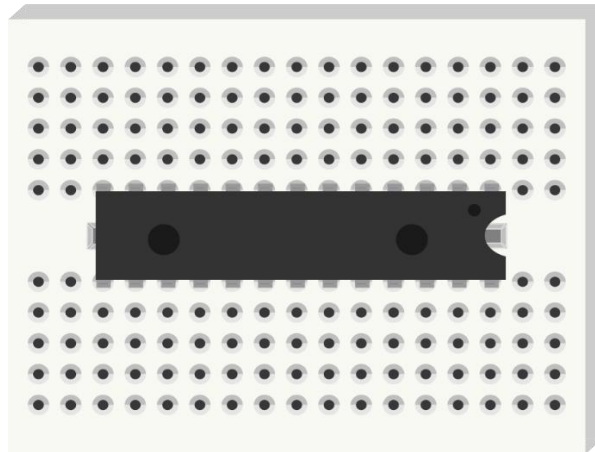
are all the same. (The connection diagrams in later projects are based on the old version of the shield.)







reserved for Narrow DIP IC chip. The figure shown below is a breadboard with a DIP Chip inserted.



## Resistor

A resistor is an electrical component that lowers the electric current. The resistor's ability to reduce the current is called resistance, measured in units of ohms(symbol:  $\Omega$ ). If we make an analogy to water flow through pipes, the resistor is a thin pipe that reduces the water flow.

Resistors can be divided into various types according to the different jobs they do in electronic circuits: pull-up resistor, pull-down resistor, current limiting resistor and so on. In this project, the digital pin 10 outputs 5V, and the input current is 40mA(DC). Typically, an LED requires the power of 2V and 35mA to light up. So here we need a resistor to reduce the voltage from 5V to 2V, the current from 40mA to 35mA. Please be careful, over-current would burn the LED.

## Read Resistor Color

The resistor value will be marked on the outer package of your resistors, but what should we do when the label is missing and there is no measuring tool at hand? The answer is to read the resistor value. This is a group of colored rings around the resistor. Details are available on Google for those interested in trying.



COLOR	FIRST AND SECOND BANDS SIGNIFICANT DIGITS	THIRD BAND MULTIPLIER	FOURTH BAND TOLERANCE
BLACK	0	$10^0$	
BROWN	1	$10^1$	1%
RED	2	$10^2$	2%
ORANGE	3	$10^3$	3%
YELLOW	4	$10^4$	4%
GREEN	5	$10^5$	
BLUE	6	$10^6$	
VIOLET	7	$10^7$	
GRAY	8	$10^8$	
WHITE	9	$10^9$	
GOLD	-	$10^{-1}$	5%
SILVER	-	$10^{-2}$	10%
NONE	-		20%

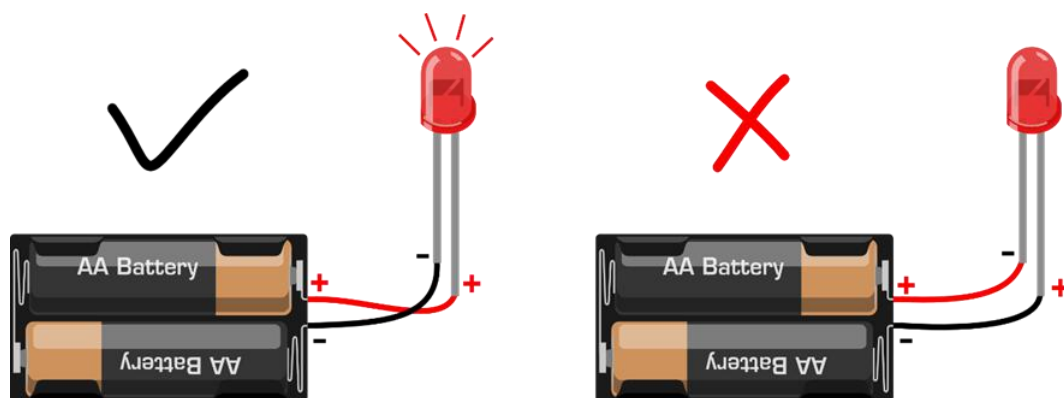
## LED

A light-emitting diode (LED) is a semiconductor device that emits light when an electric current is passed through it. Light is produced when the particles that carry the current (known as electrons and holes) combine together within the semiconductor material.

Typically, LEDs have two leads, one longer than the other, the longer lead is the positive lead (also known as the anode). If the LED's two leads are equal in length, you can look at the metal plate inside the LED. The smaller plate indicates the positive (anode) lead; the larger plate belongs to the negative (cathode) lead. If the LED has a flat area (on the plastic housing), the lead adjacent to the flat area is the negative (cathode) lead.

LEDs are polarized and must therefore be connected in the correct manner. If connected reversely, the component won't work, seen as the diagram below:






In the package, you may find LEDs with 4 leads. Don't get misunderstood, it is just an RGB LED with 3 primary color LEDs embedded into it. This will be explained in detail later.

Since you have known how these hardware works, let's start making something fun!

## Project 2 S.O.S Distress Signal

This project is based on the circuit we built in project 1, and we are gonna change the codes to turn the LED lighting into the S.O.S distress signal. S.O.S is an international call for help. In fact, the signal isn't even really supposed to be three individual letters. It is just a continuous Morse code string of three dots, three dashes, and three dots all run together with no space(...---...).

In the Morse code alphabet, the letter "S" is represented by three dots, and "O" is three dashes. So here we can directly use the blink of an LED to imitate the dot and dash: slow blink for dot, quick blink for dash.

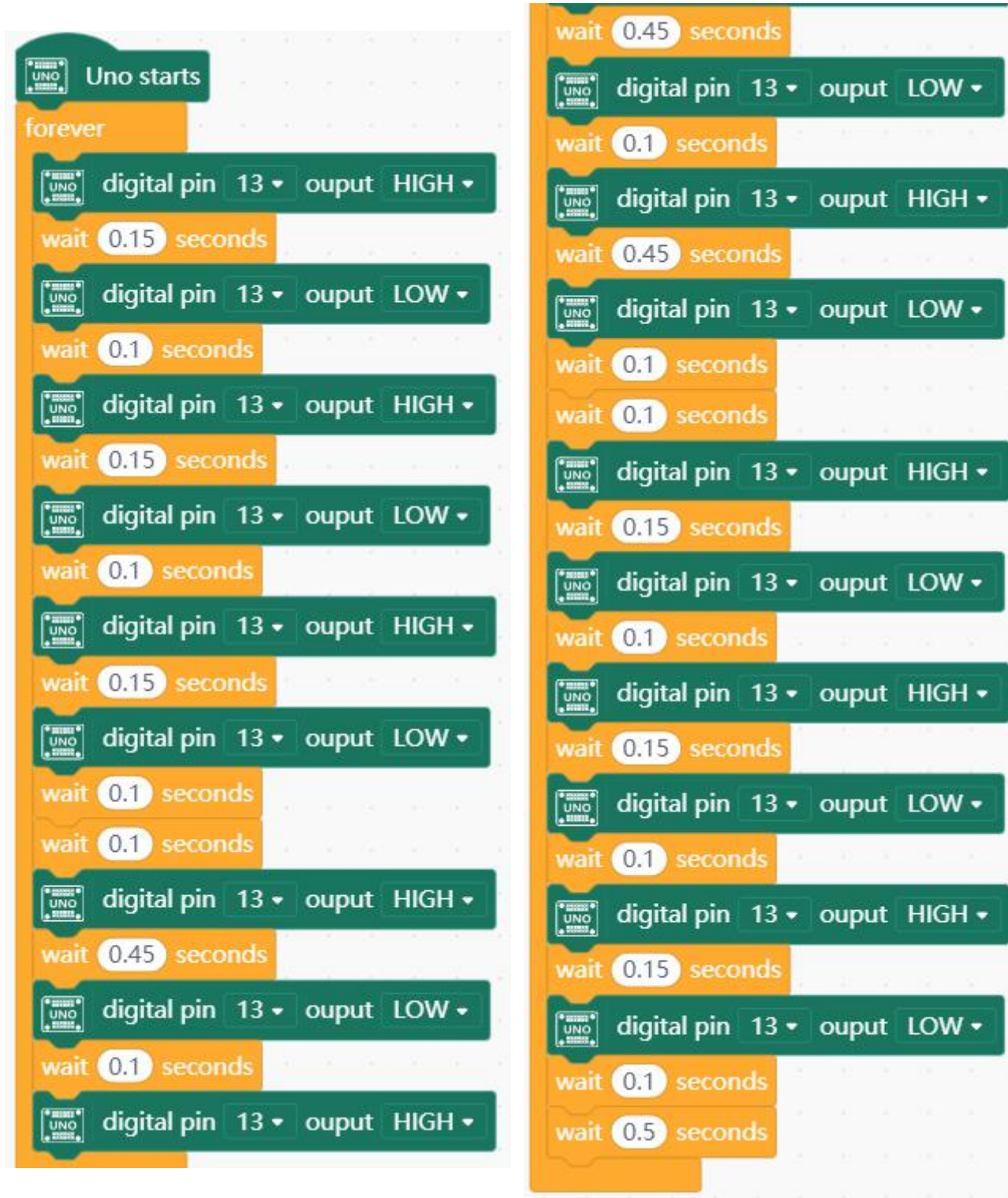


A • -	J • - - -	S • • •
B - • • •	K - • -	T -
C - • - •	L • - • •	U • • -
D - • •	M - -	V • • • -
E •	N - •	W • - -
F • • - •	O - - -	X - • • -
G - - •	P • - - •	Y - • - -
H • • • •	Q - - • -	Z - - • •
I • •	R • - •	

## Graphical Programming

Let's take a look at this example program 2-1 before starting the project.

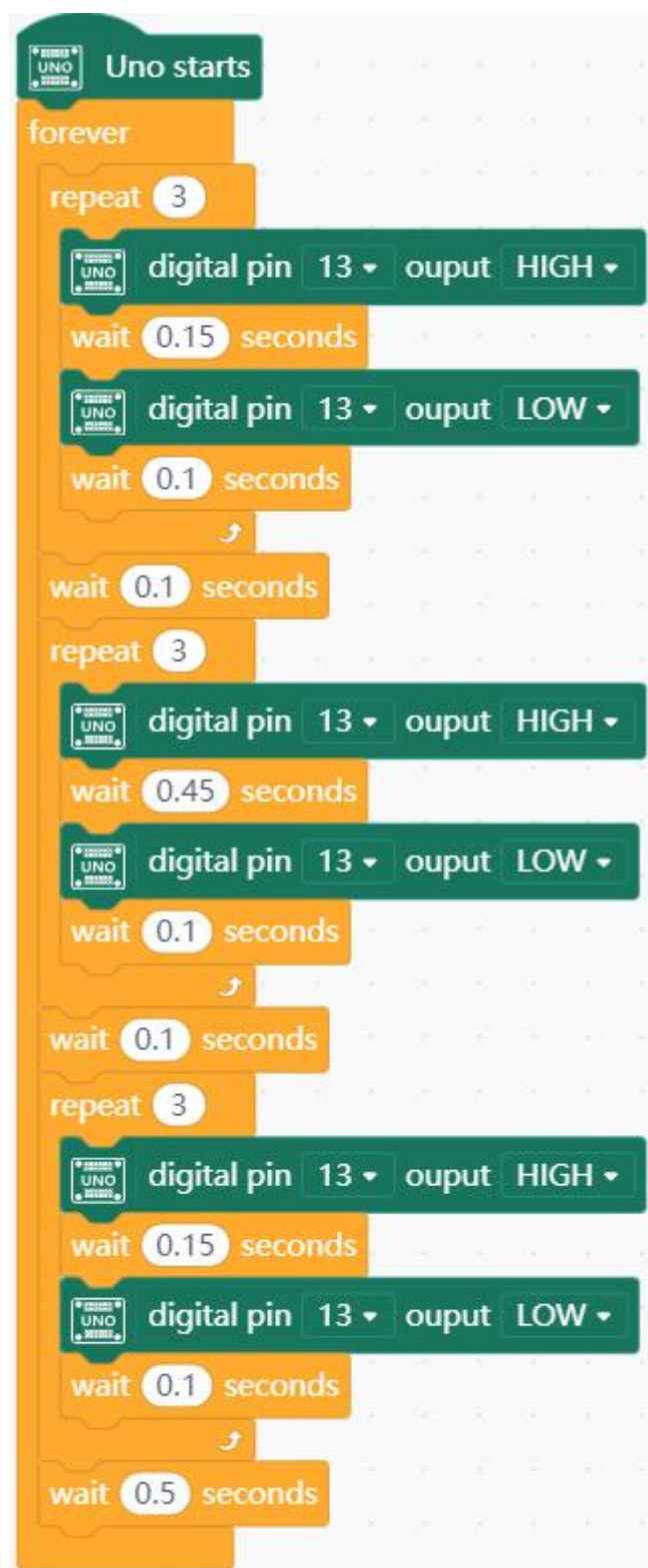
Example Program 2-1:



Although there is nothing wrong with the program, don't you think it is a little bit cumbersome. If there are 100 same actions, do we have to repeat 100 times? Of course not. Actually, the programming inventors have taken this into account and provided us with a solution.


Connect your Arduino board to a computer, open Mind+ and load the Arduino UNO blocks. Input the example code shown below:

Example Program 2-2:



Click "Upload" to download the codes into Arduino Board. If everything is going well, the LED will repeatedly blink according to the Morse code equivalent of the letters in SOS. Connect your board with external power supply, and place it in a waterproof case, then you can use it as a S.O.S signal generator.

## Command Learning

Module	Block	Description
 Control		Blocks held inside this block will loop a given amount of times.

## Code Learning

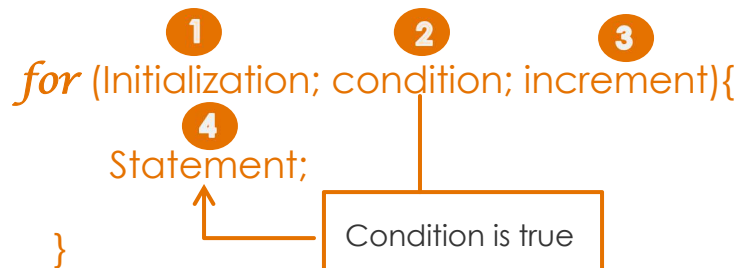
Here are the C codes of the project.

```
1. // Main program start
2. void setup() {
3.
4. }
5. void loop() {
6.   for (int index = 0; index < 3; index++) {
7.     digitalWrite(13, HIGH);
8.     delay(150);
9.     digitalWrite(13, LOW);
10.    delay(100);
11.  }
12.  delay(100);
13.  for (int index = 0; index < 3; index++) {
14.    digitalWrite(13, HIGH);
15.    delay(450);
16.    digitalWrite(13, LOW);
17.    delay(100);
18.  }
19.  delay(100);
20.  for (int index = 0; index < 3; index++) {
21.    digitalWrite(13, HIGH);
22.    delay(150);
23.    digitalWrite(13, LOW);
24.    delay(100);
25.  }
26.  delay(500);
27. }
```

There are three independent code segments that begin with “for” in the loop main function. That's the key to solving the repetition problem we met before.

## for loop

The format of for loop statement:



The sequence of “for loop” is as following.

Round 1: 1 → 2 → 3 → 4

Round 2: 2 → 3 → 4

...

End when “2” is not true.

Now let's analyse the for loop in the program:

```
1. for(int index=0; index<3; index++){  
2.     .....  
3. }
```

Step1: initialize the variable index=0

Step 2: judge if the index is less than 3

Step 3: if the condition in step is true, execute the following statement

Step 4: change the index by 1

(index++ means to increase the index value by 1, that is to say, index=index+1. )

Step 5: go back to step 2, determine if the index is less than 3

Step 6: repeat step 3

...



Once `index=3`, the condition "`index<3`" is false, the program goes out of the "for loop" and executes the next statement.

Here we need it to repeat 3 times, so we set `index<3`. If we need 100 times repetition, then it should be: `for(int index=0;index<100;index++){}`

Please note the curly braces must be in matching pairs, otherwise, there will be errors appearing when compiling code. Pay attention to the details when coding!

Here are some commonly-used operators we may use in programming:

**Comparison Operators** are often used to compare two values, including:

- `==` (equal to)
- `!=` (not equal to)
- `<` (less than)
- `>` (greater than)
- `<=` (less than or equal to)
- `>=` (greater than or equal to)

**Note:** when coding manually, there must be two "`==`" to represent "equal to".

Besides that, **arithmetic operators** are also used frequently such as, `+`, `-`, `*`, `/`.

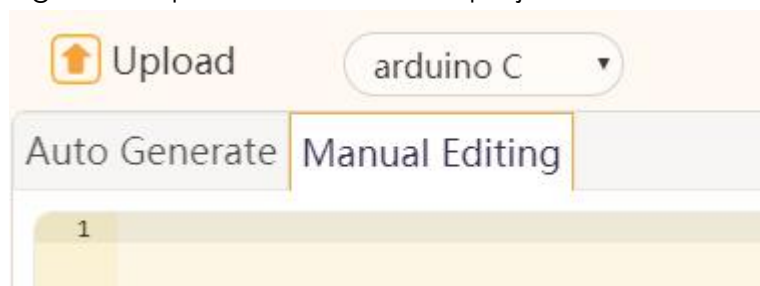
Now we have learned the usage of "for loop". Let's analyze the "for" statement in this program. There are three for loops: the first one has 3 repetitions, which represent 3 quick blinks (the letter "S"); the 3 repetitions of the next for loop are for three slow blinks (letter "O"); the last one indicates the letter "S".

There is a 0.1s pause between each "for loop" to distinguish the three letters, and we set a 0.5s delay when re-executing the main function "loop".

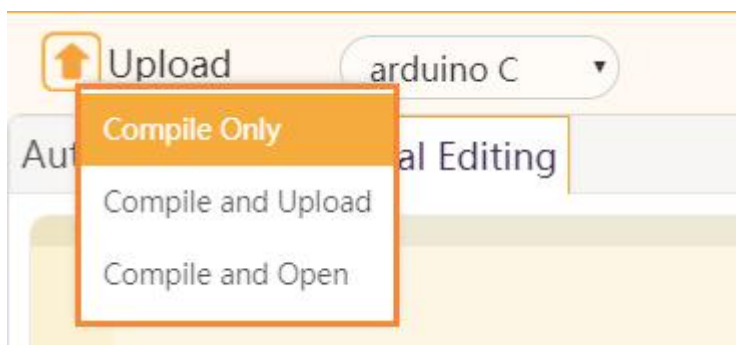
OK, that's all for the SOS distress signal project. What are the takeaway points in this part ? It's quite easy to summarize, have a try!

## Add-activities

Click "Manual Editing", and input the codes of this project into Mind+ manually.



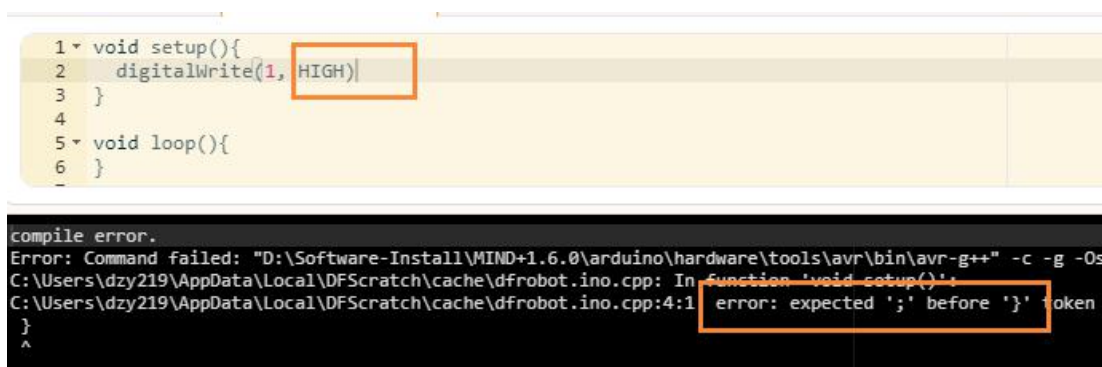
Right-click the up-arrow next to "Upload", select "Compile Only" to check the codes.



\* When we change the program by dragging blocks, the codes in the "Auto Generate" interface will be altered accordingly. But if you are in the "Manual Editing" interface currently and you click "Upload", the revised codes can't be downloaded to your board. You have to switch to "Auto Generate" to update codes to the board.

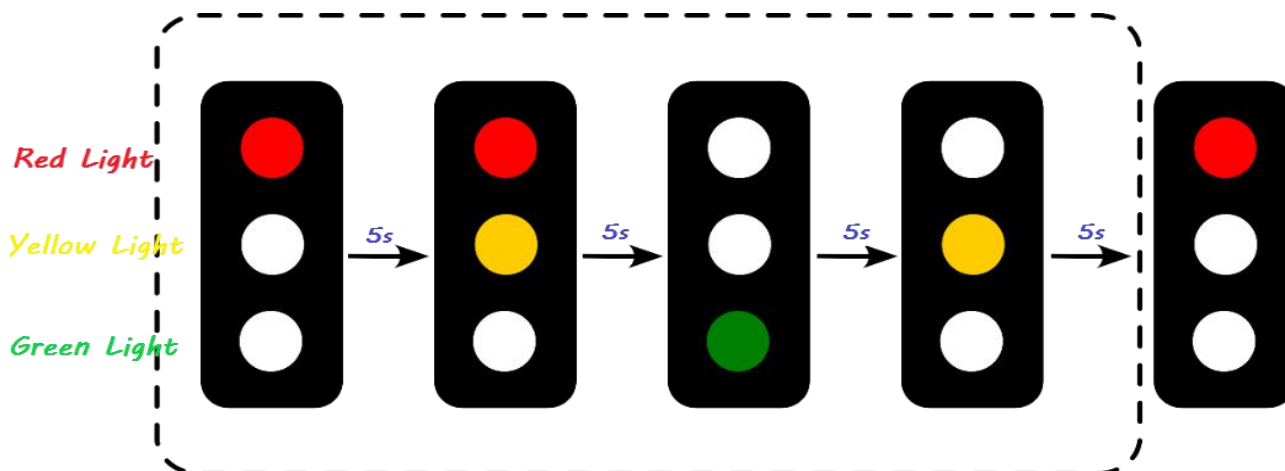
As with writing, formats are critical for C code. Beautiful and compact codes can improve our efficiency a lot. Practice makes perfect!

Look at the figure below and try figuring out what caused error.



Answer: there is no ";" at the end of the statement or it is not typed under English Input Method.

How about using Mind+ to make a traffic light. Tips: use 3 digital pins to control 3 LEDs.





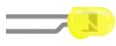




## Project 3 Interactive Traffic Lights

Did you successfully finish the add-activity above? If not, it doesn't matter. After completing this chapter, you will find this is just a piece of cake. The project here is to make a button-controlled traffic light. The Arduino will execute when the button is pressed, and by changing the state of light, we can make the cars stop and allow the pedestrian to cross safely.

We are going to learn how to interact with Arduino and define a block in Mind+ here, so the number of codes in this project is relatively large. But be patient, you will benefit a lot from this section.

### Components

■ 5mm LED	× 2	
■ 5mm LED	× 2	
■ 5mm LED	× 1	
■ 220Ω Resistor	× 6	
■ Button	× 1	

\* The UNO board, prototyping shield+breadboard, and jumper wires are not listed here and won't be listed later, but they are necessary for every project.

Here list 6 resistors, 5 of them are for the 5 LEDs to reduce current, how about the last one? Actually, it is for the button, we call it **pull-down resistor**.

### Hardware Connection

Connect all parts together as the figure 3-1 shows. It's a little bit complicated, please check carefully before powering up the module. The light-green lines marked on the breadboard represent socket connection and are not real wires.

**Note:** always turn off the power before you connect.

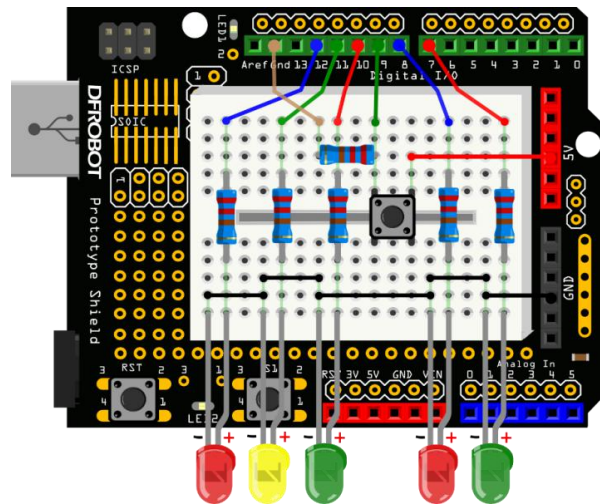


Figure 3-1 Interactive Traffic Lights Hardware Connection

## Graphical Programming

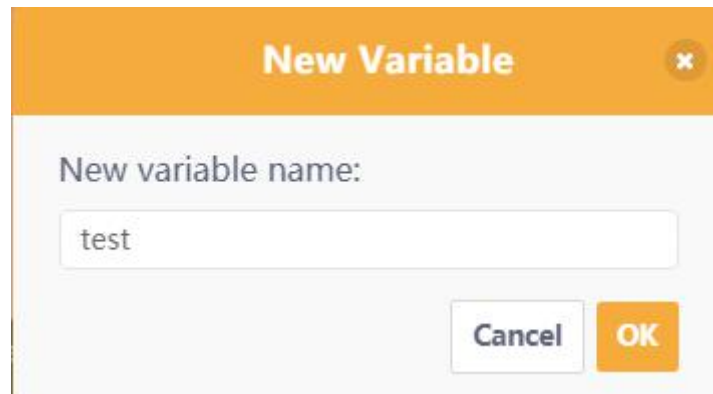
Variable and Block will be used in this project. Let's learn how to make a variable and define a block in Mind+.

### Make a Variable

Step1: click "Make a Numeric Variable"



**Step 2:** input a name for your variable. It can begin with letter, underscore, digits, or Chinese characters(Chinese will be converted into Pinyin in the auto-generated codes).



**Step 3:** the new variable **test** has been created.



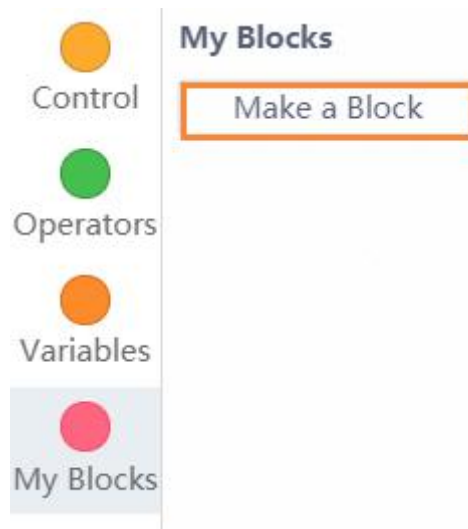
\* You may find that the variable name in auto-generated code is not the same as the block, that's because, to avoid the conflict between the variable name and other codes, we always add a prefix "mind\_n\_" to the variable you named.



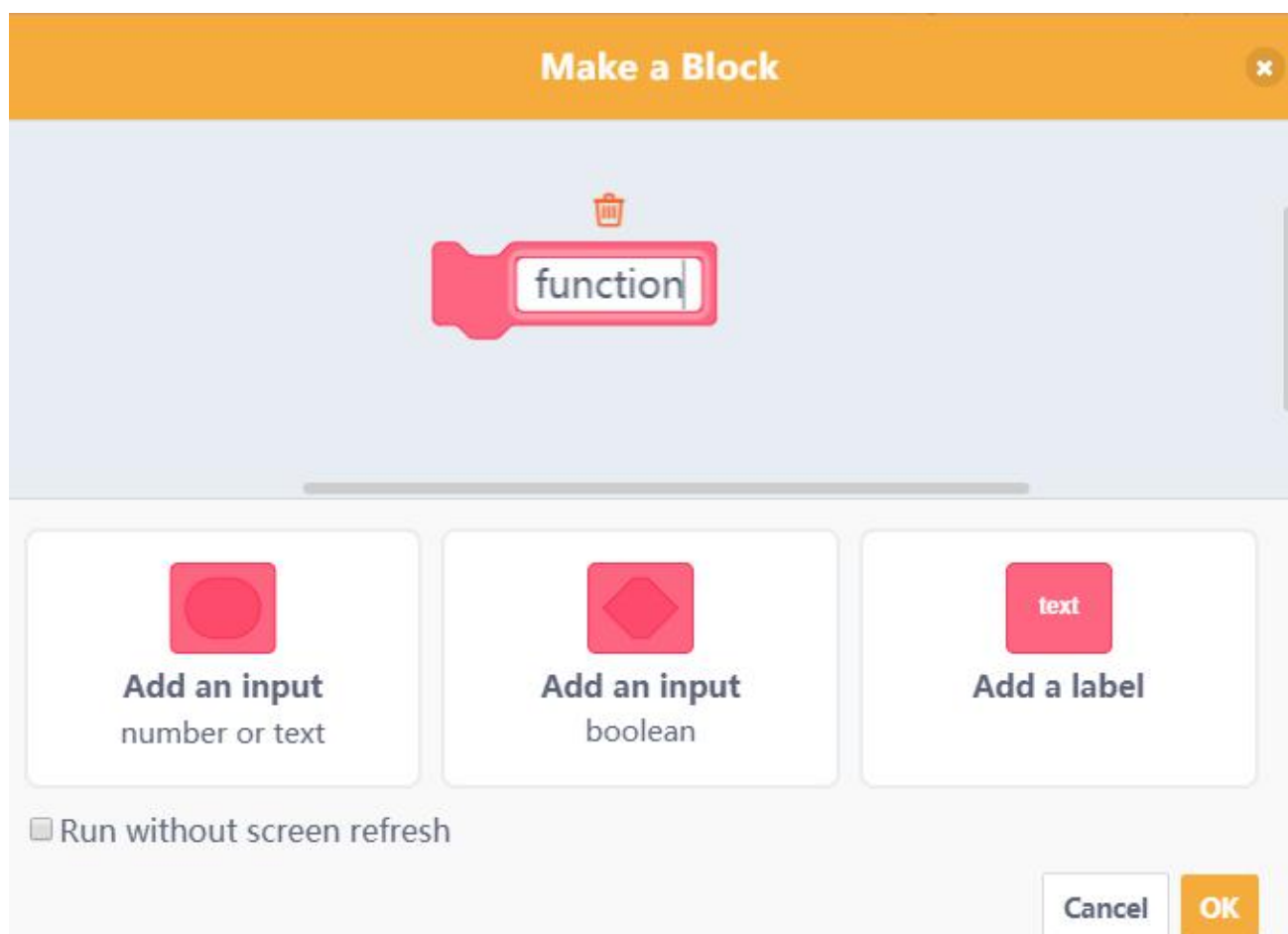
```
7 // Dynamic variables
8 volatile float mind_n_test;
9
```


**Define a block**

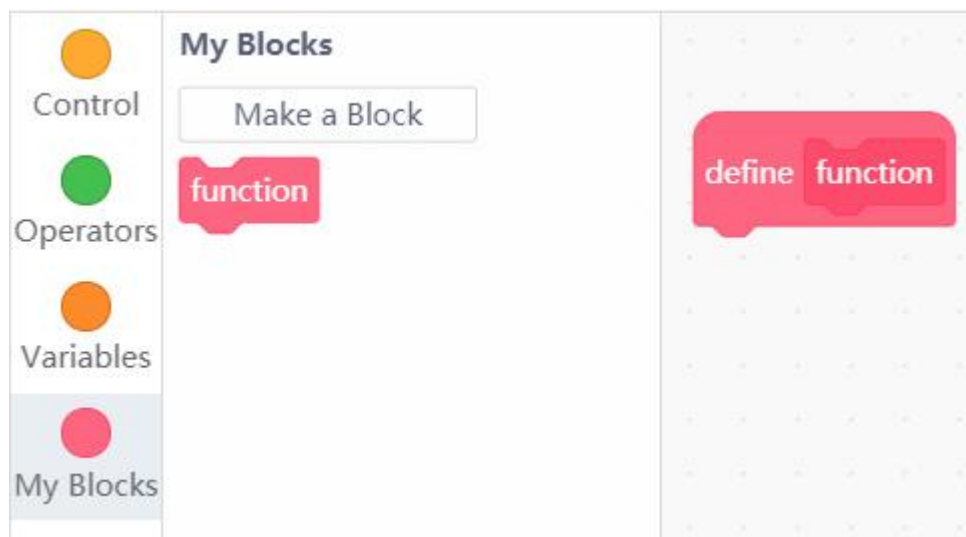
**Step1:** click "Make a Block"



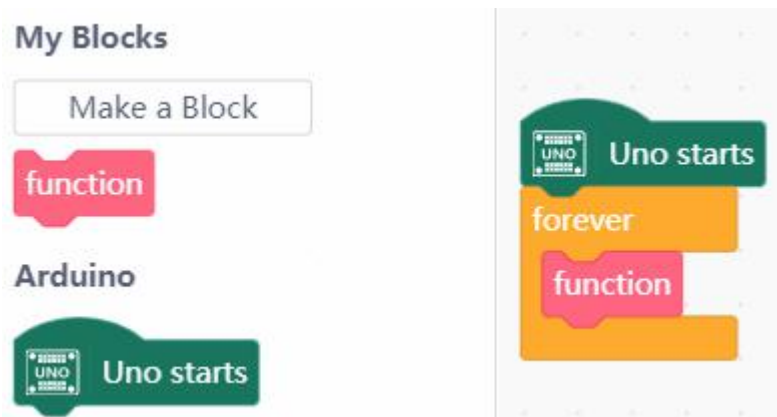
Step 2: input a name for the function.



Step 3: the block we created  will appear in the command section, we can place other blocks under this function to define it:



Step 4: drag this block into program to call it.



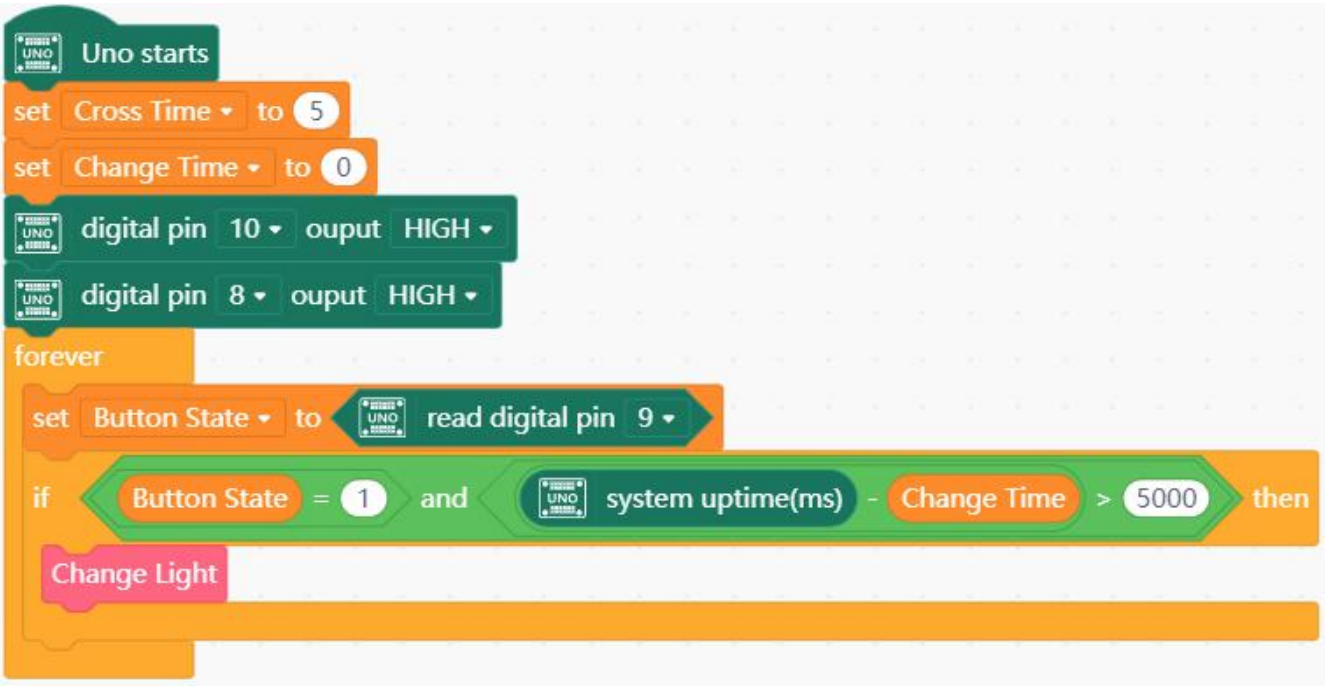
### Input Example Program

Connect the Arduino board and your computer, open Mind+ and load the Arduino library. Complete the program shown in picture 3-1:





Three variables are required here. You can jump to the command learning part if encountering problems in using new blocks.








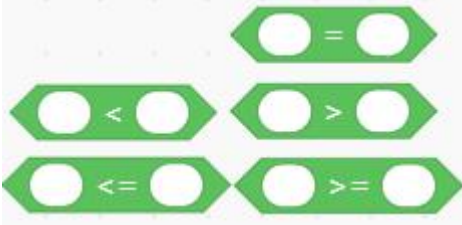



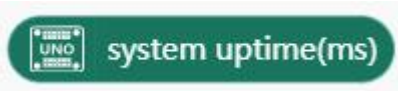

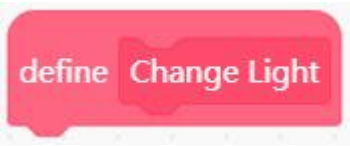








Click "Upload" to download the program into Arduino. The left three LEDs are for cars and the rest two are for pedestrians. Firstly, the green traffic light and the red pedestrian light turn on to allow cars to pass. Once you pressed the button, the pedestrian light changes from red to green, and the traffic light changes from green to yellow and red, then pedestrians can go. There is a Cross Time for people to cross the road. When it comes to an end, the green pedestrians flash quickly to notify people. After that, all lights back to the initial state.

Module	Block	Description
 Variables		To store changeable data. Right-click it to find more options (in script section).
 Variables	<div>Set the value of variable (positive/negative number, or decimal)</div> <div></div> <div>Select, name, and delete variable<div><div>✓ my variable</div><div>Rename numeric variable</div><div>Delete the numeric variable [my variable]</div></div></div>	Assign a value to your variable. Click the variable name to select, rename and delete variable in the drop-down menu.

 Arduino		Read the value a pin received, 0 or 1.
 Control		If the condition is true, the blocks held inside it will run, and then the script involved will continue. If the condition is false, the code inside the block will be ignored and the script will move on.
 Operators		Arithmetic operators: +, -, *, / Fill the blank with number, or other blocks such as variable.
 Operators		Comparison operators: >, <, =, >=, <=.
 Operators		Logic Operators: and, or, not.
 Arduino		Similar to Variable. It returns the number of milliseconds the current sketch has been running since the last reset.
 My Blocks		Click <b>Make a Block</b> to create a block, and then you can build your program under this block.
 My Blocks		Call the function directly after it has been defined.

It seems to be very complex, but don't worry, we will lead you step by step to learn all commands used in the codes. You will comprehend the entire program soon.



## Code Learning

```
1. // Dynamic variable
2. volatile float mind_n_Cross_Time, mind_n_Change_Time, mind_n_Button_State;
3. // Function declaration
4. void DF_Change_Light();
5.
6.
7. // Main program starts
8. void setup() {
9.     mind_n_Cross_Time = 5;
10.     mind_n_Change_Time = 0;
11.     digitalWrite(10, HIGH);
12.     digitalWrite(8, HIGH);
13. }
14. void loop() {
15.     mind_n_Button_State = digitalRead(9);
16.     if (((mind_n_Button_State==1) && ((millis()-mind_n_Change_Time)>5000))) {
17.         DF_Change_Light();
18.     }
19. }
20. // Define function
21. void DF_Change_Light() {
22.     digitalWrite(10, LOW);
23.     digitalWrite(11, HIGH);
24.     delay(2000);
25.     digitalWrite(11, LOW);
26.     digitalWrite(12, HIGH);
27.     delay(1000);
28.     digitalWrite(8, LOW);
29.     digitalWrite(7, HIGH);
30.     delay(mind_n_Cross_Time * 1000);
31.     for (int index = 0; index < 10; index++) {
32.         digitalWrite(7, HIGH);
33.         delay(250);
34.         digitalWrite(7, LOW);
35.         delay(250);
36.     }
37.     digitalWrite(8, HIGH);
38.     delay(500);
39.     digitalWrite(12, LOW);
```

```
40.    digitalWrite(11, HIGH);
41.    delay(1000);
42.    digitalWrite(11, LOW);
43.    digitalWrite(10, HIGH);
44.    mind_n_Change_Time = millis();
45. }
```

Let's begin with the first line.

```
1.  // Dynamic variable
```

This is the text description of the code, we call it comment.

## Comment

In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters. The syntax of comments varies from programming languages.

C++ has block comments delimited by `/*` and `*/` that can span multiple lines and line comments delimited by `//`.

For example:

```
/* Texts between these two operators are comments.
   And they should be ignored by compilers
   Comments are shown in grey */
```

Then let's move to the third and fourth lines.

```
3.  // Function declaration
4.  void DF_Change_Light();
```

## Function Declaration

In graphical programming, before calling a function, we have to create and name it first, similarly, in coding, a function must be declared first when defining or calling it. Most of the functions used in this tutorial do not come with input and output. The format is shown below:

```
void FunctionName ();
```

The format of function with input and output will be given later.

The second line of code is used to declare a variable.

```
2. volatile float mind_n_Cross_Time, mind_n_Change_Time, mind_n_Button_State;
```

Every time we use a new variable, we have to declare it first. Here are the blocks they correspond with:



## What is a Variable?

We can compare a variable as a container or box where we can store things. The thing we put into the box must be smaller than the box itself, otherwise, it may overflow. Same as that, the data stored in variable should be within a certain range. Sometimes the value of variable could be changed in the process of programming running, and we will analyze it deeply later. We have learned how to make and name a variable in Mind+'s graphical programming part, however, there are a lot of rules for naming a variable when coding manually. For instance, in C:

- Variable name must begin with letter or underscore.
- Variables are case sensitive.
- They can be constructed with digits, letters.
- No special symbols are allowed other than underscore.
- Some specified words like, main, if, or while are unacceptable.

## Can the box capacity of a variable be infinitely big?

Do variables have different sizes? The answer is Yes. Seen as the chart 3-1.

Date Type	RAM	Range
boolean	1 byte	0 ~ 1 (True or False)
char	1 byte	-128 ~ 127
unsigned char	1 byte	0~255
int	2 byte	-32768 ~ 32768
unsigned int	2 byte	0 ~ 65535

long	4 byte	-2147483648 ~ 2147483647
unsigned long	4 byte	0 ~ 4294967295
float	4 byte	-3.4028235E38 ~ 3.4028235E38
double	4 byte	-3.4028235E38 ~ 3.4028235E38

There are various types of variables corresponding with different data types. Int and long are for integers, char for characters, float and double for variables with decimal point.

Theoretically, the size of the variable box can be all the same, however, here is the thing, the storage space for a micro-controller is limited. There are only 32K flash memory for Arduino Uno main chip (Atmega328) so if we could save some storage space, why not do it?

In Mind+'s graphical programming, the numerical variable is set to be the type of float with volatile specifier, which is compatible with decimal, integers or variables in interrupt. When coding manually, the type and specifier of variables can be selected freely.

### Global variable and local variable

Here a new knowledge point needs to be introduced: **global variable and local variable**.

**Global variables**, the variable declared in the second line, **are declared outside any function**, and they can be accessed on any function in the program.

```
2. volatile float mind_n_Cross_Time, mind_n_Change_Time, mind_n_Button_State;
```

The index in the 32<sup>nd</sup> line is a local variable.

```
32. for (int index = 0; index < 10; index++) { }
```

Local variables **are declared inside a function, and can be used only inside that function**. It is possible to have local variables with the same name with different functions. In project 2, every for loop has a variable index, but they do not conflict with each other since every index can be used only in its own loop.

The index used in parenthesis is a local variable.

```
for (Initialization; condition; increment){
    Statement;
}
```

Can be used in loop statement.

We will find a new function in line 15: digitalRead()

```
15.    mind_n_Button_State = digitalRead(9);
```

### ▪ digitalRead()

The format is shown below:

Pin number  
digitalRead(pin)

The related block in Mind+:



This function is used to read the state of the digital pin, High or Low(Hight is 1, low is 0). A parameter pin has to be transferred to the function. In the project, we have to read the button signal, and the button is connected pin 9.

The read button signal will be passed to the variable mind\_n\_Button\_State. When the variable is equal to 1(High), the button is pressed, if it is 0(Low), button unpressed.

We can directly check the value of Button State to determine if the button is pressed.

```
15.    if (((mind_n_Button_State==1) && ((millis()-mind_n_Change_Time)>5000)))  
16.    {  
17.        Change_Light();  
18.    }
```

Here comes a new statement---if.

### ▪ if Statement

An if statement is a programming conditional statement that, if proved true, performs a function or displays information, if not, exit if statement.

The basic format of if statement is:

```
if (expression){  
    Statement;  
}
```

The expression refers to the condition, if it is evaluated to be true, the statement block will get executed, or it will get skipped.

In line 15, the first condition is Button State is HIGH. When the button is pressed, the Button State will be HIGH. The second condition is the value `millis()` minus Chang Time is over 5000. The two conditions are linked by "&&", a logic operator that returns True if both operands are True and returns False otherwise.

There is a new function in the if statement above.

#### ▪ `millis()`

It returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow(go back to zero), after approximately 50 days. Here we use it to calculate if there is a break more than 5s when the button is pressed once again. If there is not, the codes will be skipped to avoid errors that may be caused by pressing the button accidentally.

### Logical Operators

The common operators:

- &&-----AND (Returns true if both operands are true and false otherwise)
- | |-----OR (Any of its arguments are true, it returns true, otherwise, false)
- !-----NOT (Returns the inverse value)

<i>NOT</i>		<i>AND</i>			<i>OR</i>		
<i>Input</i>	<i>Return</i>	<i>Input 1</i>	<i>Input 2</i>	<i>Return</i>	<i>Input 1</i>	<i>Input 2</i>	<i>Return</i>
0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1
		1	0	0	1	0	1
		1	1	1	1	1	1

The 1 and 0 in the truth table can represent High/Low, or True/False. For example,  $(10 > 8) \&\& (9 < 5)$ , what the value it will return?

Apparently,  $10 > 8$  is true so the return is 1.  $9 < 5$  is false, return 0. The whole statement can be simplified as "1 && 0", from the table above, we can easily know that the return of this statement is 0.

There is one function inside the if statement.

- `DF_Change_Light();`

This is an example of function call. It is declared outside of the `loop()`. We can directly write the name of the function to call it. This function is void and has no return. When being called, the function will be executed and then back to the main function after completing. Please note that the parenthesis behind the function cannot be omitted and the function name should be the same as the one you declared before.

## Hardware Review

### Button Switch

The button we used here has 4 pins. Once it is pressed, the left and right sides will be connected while the top and bottom are always connected.

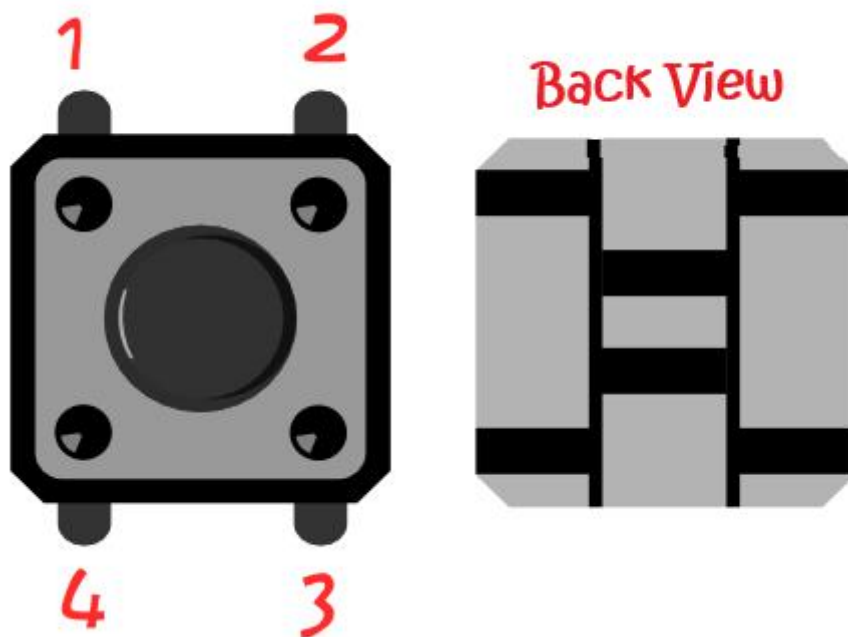


Figure 3-2 The Structure of Button



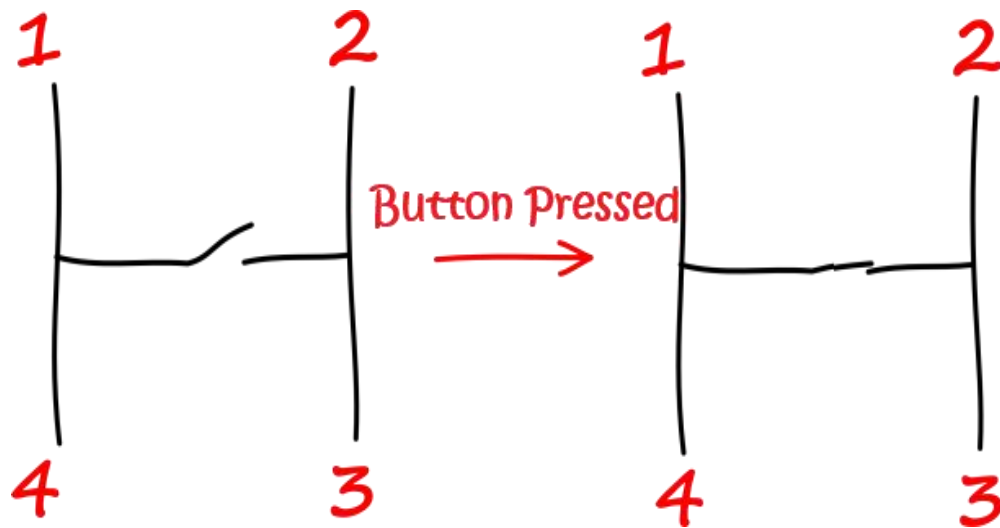


Figure 3-3 The Working Principle of Button

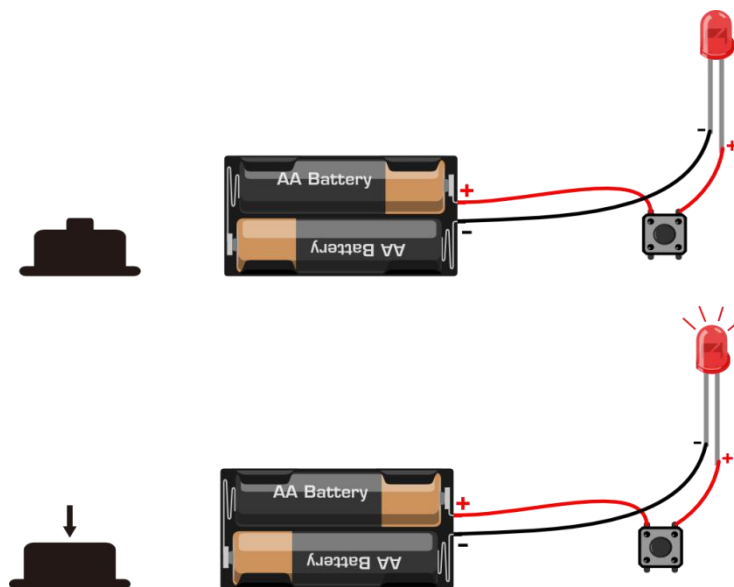


Figure 3-4

A push-button can be used to control the electricity flowing on the circuit. In the project, when the button is pressed, D Pin 9 will be connected to 5V, and it will be read as High. Otherwise, it is in Low state(GND).

### What is a Pull-down Resistor?

Pull-down resistors are used in electronic logic circuits to ensure that inputs to the Arduino settle at expected logic levels if external devices are disconnected or are at high-impedance. As the name suggests, a pull-down resistor weakly “pull” the voltage of the wire it is connected to towards its voltage source level when the other components on the line are inactive.

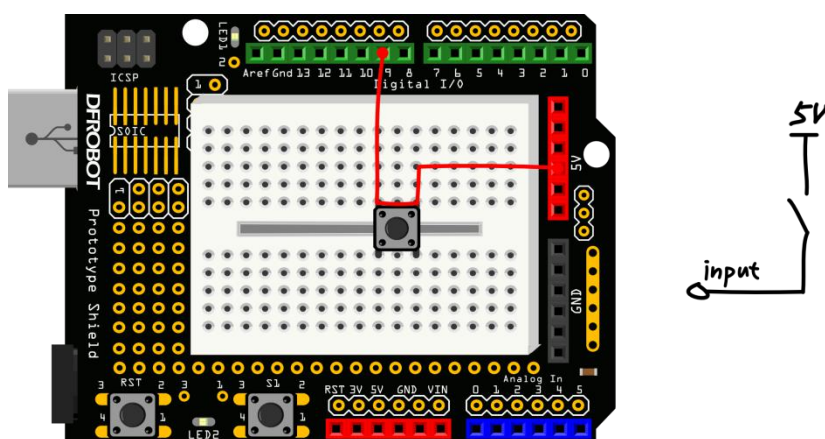


Figure 3-5 Without Pull-down Resistor

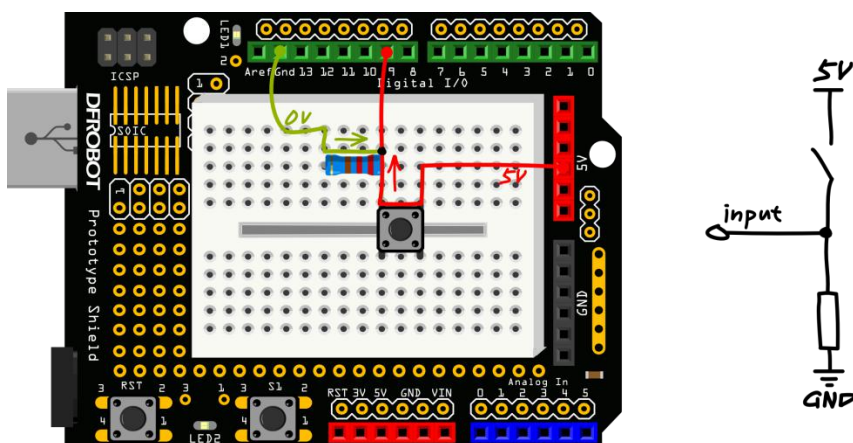


Figure 3-6 With Pull-down Resistor

When the switch on the line is open, it has high impedance and acts like it is disconnected. Since the other components act as though they are disconnected, the circuit acts as though it is disconnected, and the pull-down resistor brings the wire up to the low logic level. When another component on the line goes active, it will override the low logic level set by the pull-down resistor. The pull-down resistor assures that the wire is at a defined low logic level even if no active devices are connected to it.

## Add-activities

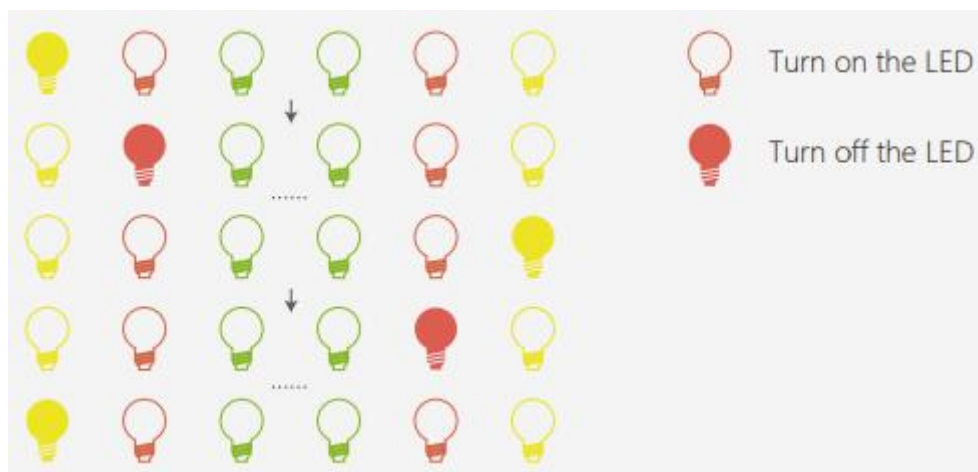
1. Input codes into Mind+, never miss any chance to practice! But be careful, don't lose any necessary elements otherwise it will fail to compile, just like the picture shown below:

```
1 void setup(){
2   digitalWrite(1, HIGH);
3   |
4   |
5 void loop(){
6 }
```

compile error.  
Error: Command failed: "D:\Software-Install\MIND+1.6.0\arduino\hardware\tools\avr\bin\avr-g++" -c -g -Os -C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp: In function 'void setup()':  
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp:6:12: error: a function-definition is not allowed here before 'void loop()' declaration  
void loop(){  
^  
C:\Users\dzy219\AppData\Local\DFScratch\cache\dfrobot.ino.cpp:7:1: error: expected '}' at end of input  
}  
^

A curly brace"}" is missed

2. Use 6 LEDs to make a lighting effect of water flowing.



3. Light up the LEDs from the middle towards both sides.



4. Light up from left to right in 1, 2, 3....



## Project 4 Breathing LEDs

Till now, we have learned how to control an LED connected onto Arduino by programming. In fact, we can do way too much with Arduino. In this project we are going to control the brightness of LEDs on Arduino. There are 6 digital pins marked with "~" on the UNO board: 3, 5, 6, and 9. This mark indicates that these pins can be used to generate PWM. When the circuit slowly fades an LED on and off, it forms an effect of breathing, so we call it breathing LEDs.

### Components

5mm LED × 1

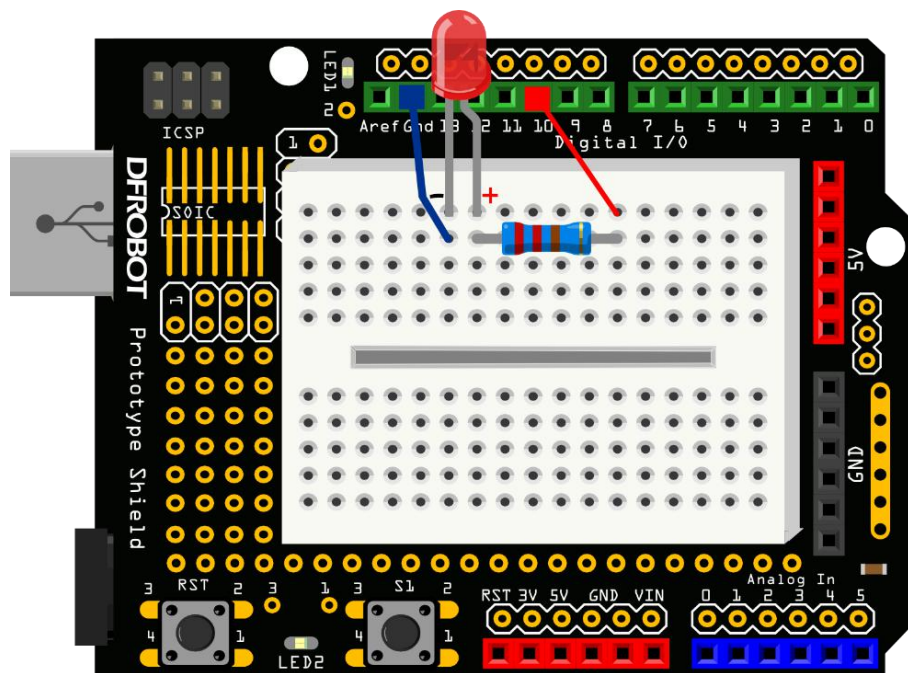


220Ω Resistor × 1



### Hardware Connection

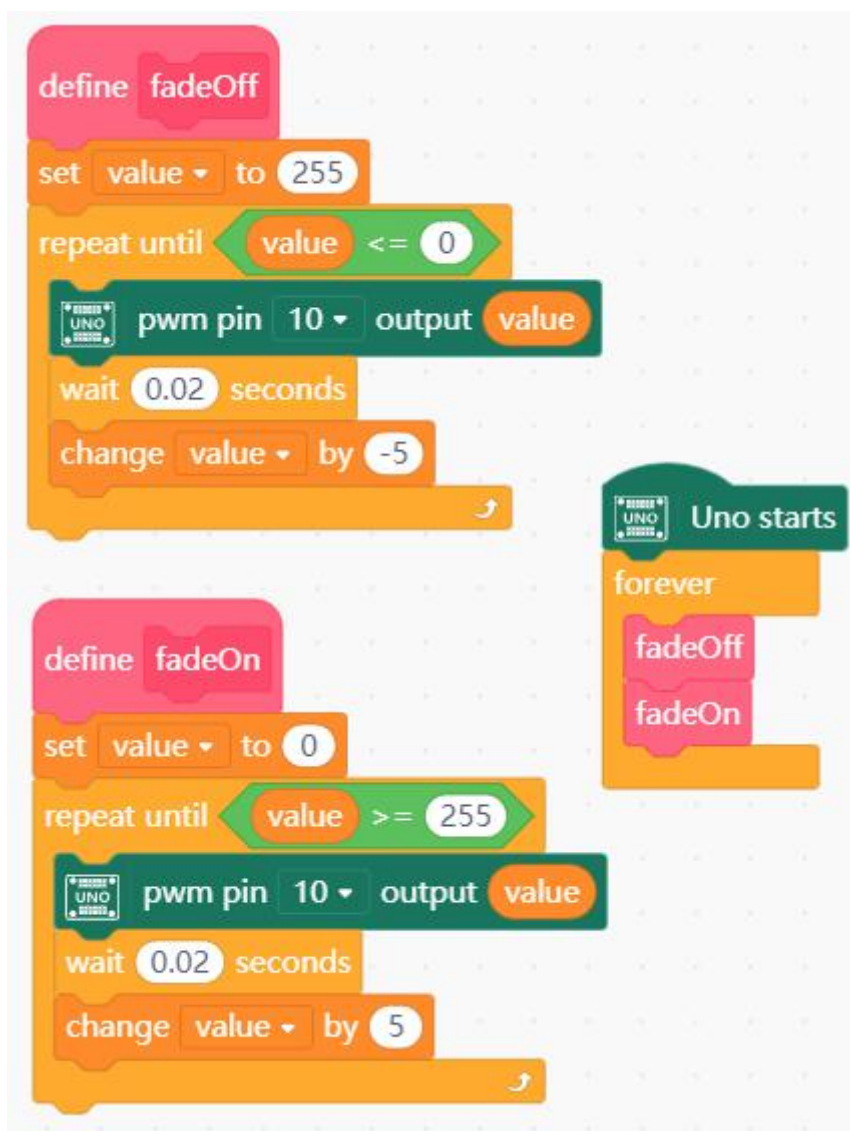
The connection is the same as the one of project 2.



### Graphical Programming



Connect the board with your computer, open Mind+ and load the Arduino library. Input the example program below:

Example Program 4-1:



Load the codes to your Arduino board then you will see the LED gradually brightening and dimming.

## Command Learning

Module	Block	Description
 Variables		Blocks held inside this block will loop until the specified statement is true, in which case the code beneath the block (if any) will execute.

		PWM output a value; control brightness via PWM signal.
---	---	--

## Code Learning

```

1. volatile float mind_n_value;
2. // function declaration
3. void DF_fadeOff();
4. void DF_fadeOn();
5.
6.
7. // Main program starts
8. void setup() {
9.
10. }
11. void loop() {
12.     DF_fadeOff();
13.     DF_fadeOn();
14. }
15.
16.
17. // define function
18. void DF_fadeOff() {
19.     mind_n_value = 255;
20.     while (!(mind_n_value<=0)) {
21.         analogWrite(10, mind_n_value);
22.         delay(20);
23.         mind_n_value -= 5;
24.     }
25. }
26. void DF_fadeOn() {
27.     mind_n_value = 0;
28.     while (!(mind_n_value>=255)) {
29.         analogWrite(10, mind_n_value);
30.         delay(20);
31.         mind_n_value += 5;
32.     }
33. }

```

We have been familiar with most codes such as, variable declaration, pin setup, and function call.



There are two functions `fadeOff` and `fadeOn` in main function, and they are basically all the same except the parameters.

```
19. void fadeOff() {
20.     mind_n_value = 255;
21.     while (!(mind_n_value < 0)) {
22.         analogWrite(10, mind_n_value);
23.         delay(20);
24.         mind_n_value -= 5;
25.     }
26. }
```

There is a new statement in `fadeOff` function: `while`. The while loop is used to repeat a section of code an unknown number of times until a specific condition is met.

While ( ) { }

The format of the function:

```
while (condition){
    statements;
}
```

The related block in Mind+:

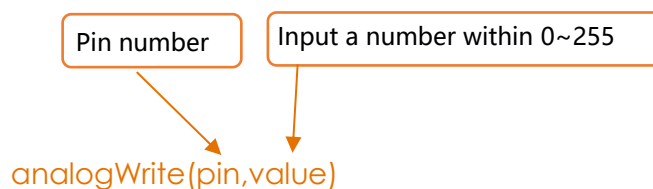


The while statement includes another new function:

- `analogWrite( 10 ,mind_n_value );`

How to send analog values to a digital pin? This function is the answer. The premise to use the function is to get a digital pin with PWM function. The 6 pins (3, 5, 6, 9, 10, 11) can be used to output PWM signal.

The format of the function:



Block in Mind+:



The “`analogWrite()`” function can assign the PWM pin an analog value between 0~255.

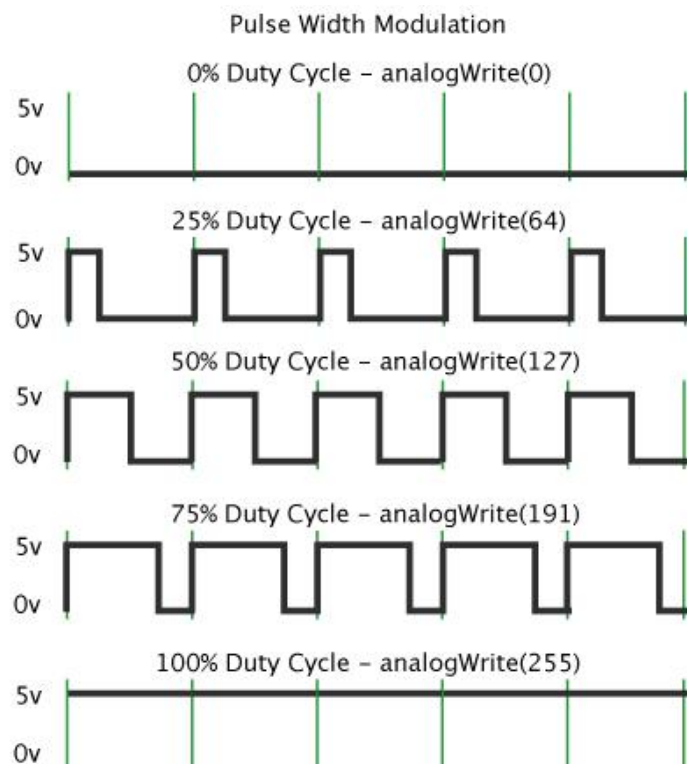
## PWM

PWM is a technique for getting analog results with digital means. Digital control is used to



create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5V) and off (0V) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of “on time” is called the pulse width.

We can learn it well through the graphical below.



In the graphical above, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to `analogWrite()` is on a scale of 0-255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.

PWM can be used to adjust the brightness of LED or the speed of motor, for instance, controlling an Arduino-based robot car.

## Add-activities

1. Type the codes of this chapter into Mind+, compile and debug.

\* If encountering any strange compiling errors, please feed us back on our forum.

2. Use LEDs to create an effect of flickering flame by controlling its brightness via PWM.

Main components:

Red LED × 1

Yellow LED × 2

220Ω Resistor × 1

The function `random()` can be very useful in this project.

A Scratch 'pick random' block with the range set from 1 to 10.

It can

generate a random within a designated range.

Note: set a brightness value for the LED first, and then generate a random around it. For example, `random(120)+135`. The related block should be:

A Scratch 'pick random' block with the range set from 0 to 120, followed by an addition of 135.

The brightness of the LED will be changed constantly around 135.

[Click the link below to find more references about various commands.](#)

<https://www.arduino.cc/reference/en/>

3. Try a more challenging project: control the LED with 2 buttons, one for brightening it, another for dimming it.

## Project 5 Colorful RGB LED

That's all for single color LED. Now let's try this colorful RGB LED. It combines red, green and blue three colors to produce over 16 million hues of light. In this project, we are gonna make an RGB LED generate different colors randomly.

### Components

- 5mm RGB LED × 1



- 220Ω Resistor × 3



### Hardware Connection

Please figure out whether your RGB LED is common cathode or common anode before connecting. If you don't know how to determine, jump to hardware review part to learn. Pay attention to the pin order.

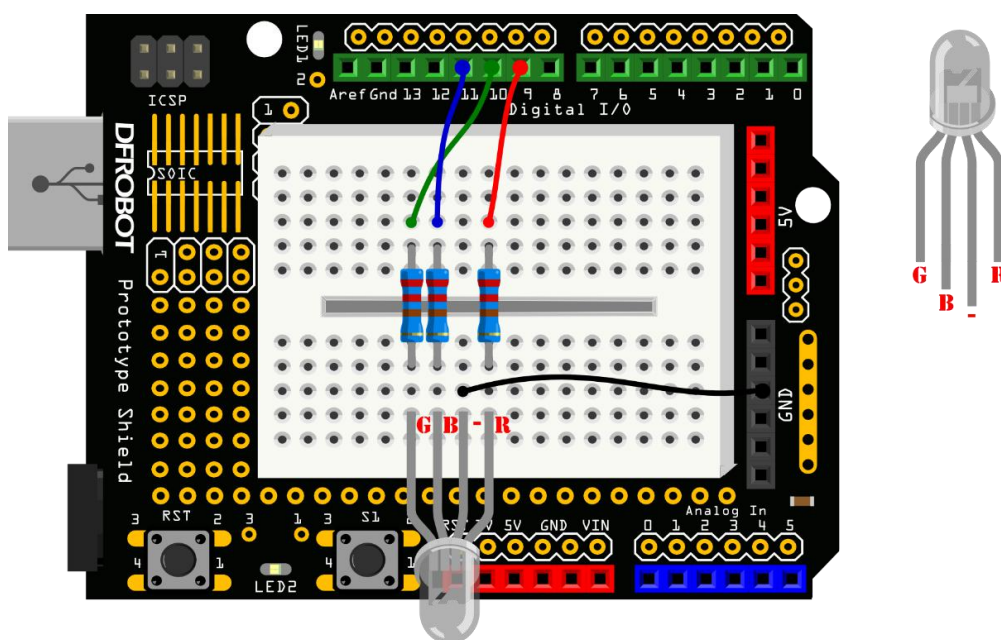
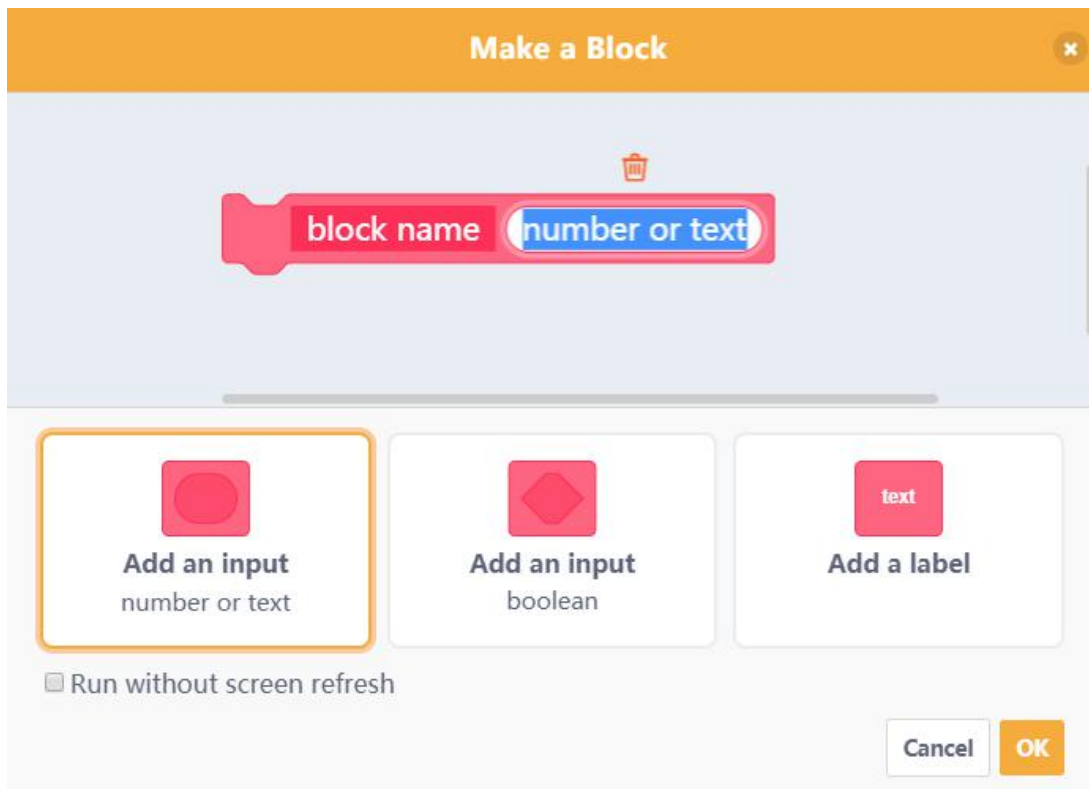


Figure 5-1 Colorful RGB LED Connection

### Graphical Programming

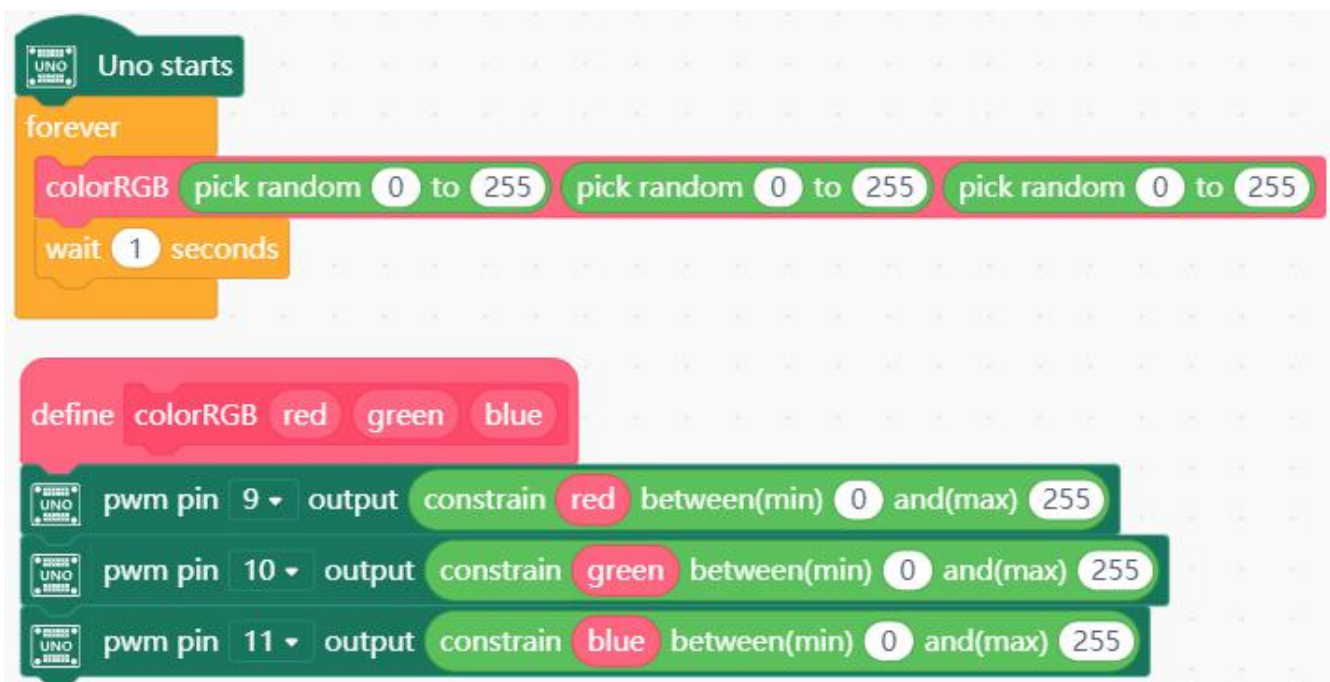
Input the example code 5-1. We need to create a new block. Click "Add an Input" to name the function.



Drag out the “red” to use it as a variable. It is a local variable and can be called only in this function.







Example Program 5-1:



Upload the codes into Arduino Board, then we can see that the LED changes constantly in random color.

## Command Learning

Module	Block	Description
 Operators		Pick a random in a designated range.
 Operators		Constrain a value or variable in a certain range.

## Code Learning

```

1. // function declaration
2. void colorRGB(float red, float green, float blue);
3.
4.
5. // main program starts
6. void setup() {
7.   dfrobotRandomSeed();
8. }
9. void loop() {
10.   colorRGB((random(0, 255+1)), (random(0, 255+1)), (random(0, 255+1)));
11.   delay(1000);
12. }
13.
14.
15. // define function
16. void colorRGB(float red, float green, float blue) {
17.   analogWrite(9, (constrain(red, 0, 255)));
18.   analogWrite(10, (constrain(green, 0, 255)));
19.   analogWrite(11, (constrain(blue, 0, 255)));
20. }

```

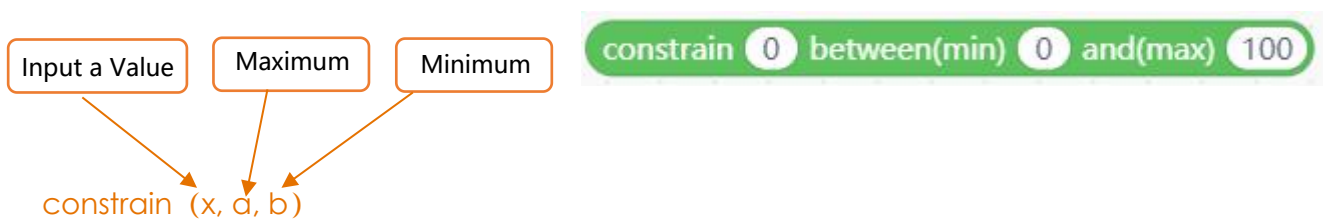
The RGB LED is a combination of 3 LEDs in just one package, so we need 3 PWM pins to control the RGB LED.

The most important part, the main function, has a function `colorRGB()` with three parameters to be input. They are used to write value into red, green and blue. When we want to use a color in the program, we can assign a value to them directly.

There are two new functions `constrain()` and `random()`. Try looking them up with the websites we mentioned in the last chapter.

#### ■ `constrain (x, a, b)`

Block in Mind+:



There are three parameters in the `constrain()`: `x`, `a` and `b`. `x` is the number to constrain; `a` is the lower end of the range; `b` is the upper end of the range.

#### Returns

`x`: if `x` is between `a` and `b`.

`a`: if `x` is less than `a`.

`b`: if `x` is greater than `b`.

In the program, the red, green and blue are constrain within 0~255 (the range of PWM).

#### ■ `dfrobotRandomSeed();`

The function `dfrobotRandomSeed()` is used to set random seed, a random number generator. Read the uncertain noise in air via the analog pin (A6, A7), use it as random seed and then we can get an ideal random number.

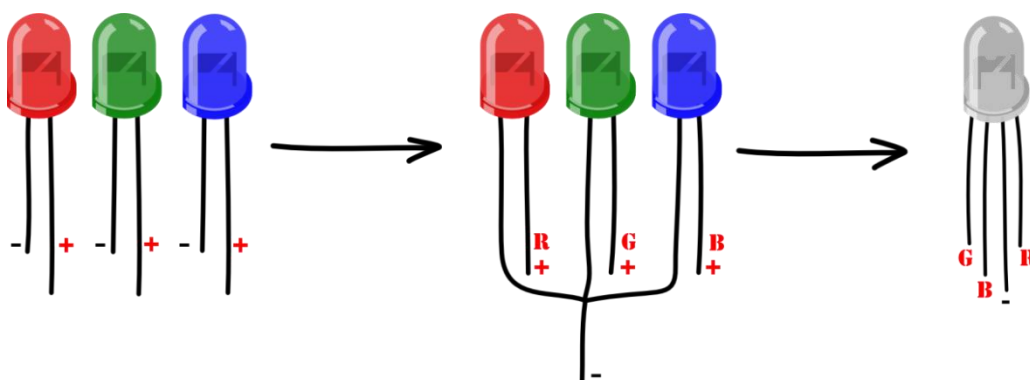
#### ■ `random (min,max)`

`random()` is used to generate a random, `min` is the lower bound of the random value, and `max` is the upper bound. To ensure the number "255" can be picked in Mind+, the maximum 255 in the auto-generated code is written as "255+1".

## Hardware Review

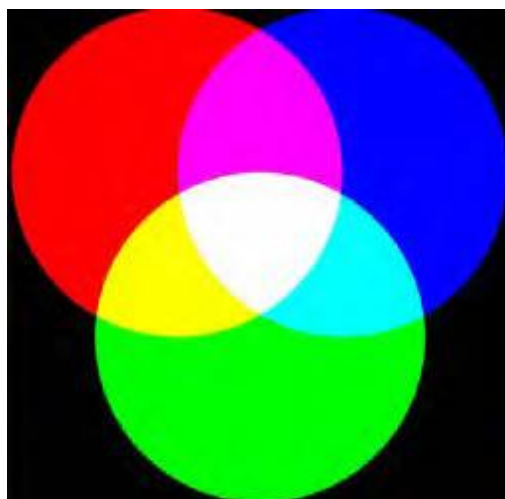
### RGB LED

RGB LEDs have four pins—one for each LED and another for the common anode or cathode. The RGB LED we use here is common cathode. In a common cathode RGB LED, all three LEDs share a negative connection(cathode), while in a common anode RGB LED, the three LEDs share a positive connection(anode). The figure below illustrates how three LEDs change into an RGB LED.



How to use an RGB LED? How to create different colors?

An RGB LED is actually three LEDs, red, green and blue inside one package. By configuring the intensity of each LED via the PWM port on Arduino(`analogWrite()`), you can produce any colors you want.



Red	Green	Blue	Color
255	0	0	red
0	255	0	green



0	0	255	blue
255	255	0	yellow
0	255	255	aquamarine
255	0	255	purplish
255	255	255	white

Chart 5-1 Colors generated from combined PWM values of 3 LEDs

We can configure  $256 \times 256 \times 256$  (1677216) kinds of colors by assigning different PWM to these three LEDs.

### Distinguish between RGB LED common anode and common cathode

What's the difference between common anode and common cathode? From the figure below, we could see that they have similar appearance.

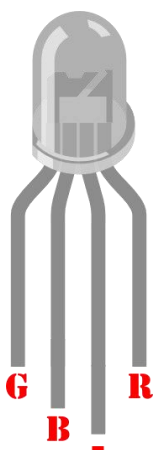


Figure 5-3 Common cathode RGB LED

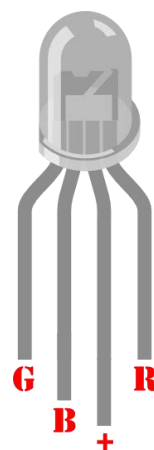


Figure 5-4 Common anode RGB LED

The best way to distinguish them is using a multimeter. Put your multimeter in continuity mode.

- If the LED lights up with the red multimeter tip on the longest lead and the black on one of the other leads--you have a common anode RGB LED.
- If the LED lights up with the black multimeter tip on the longest lead and the red on one of the other leads--you have a common cathode RGB LED.

### Add-activities

1. Input the codes of this project into Mind+ Manually, compile and debug.

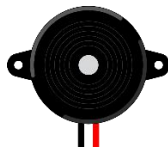
2. Based on the colorful RGB LED, change the codes to make a Rainbow RGB light. Use the function `colorRGB()` and adjust the value of red, blue and green in 0-255 to create the color you want.
3. Combine the breathing LED and Rainbow light together to display a smooth rainbow lighting effect.

## Project 6 Alarm Device

In this chapter we are gonna try a new electronic component: buzzer, a audio signaling device. We will use it to make an alarm device. The buzzer generates sounds at different frequencies when driven by sine wave signals. Combine a LED with the same sine wave, then we can make an alarm device.

### Components

Buzzer x 1



### Hardware Connection

Connect the positive of the buzzer to digital pin 8, negative to GND.

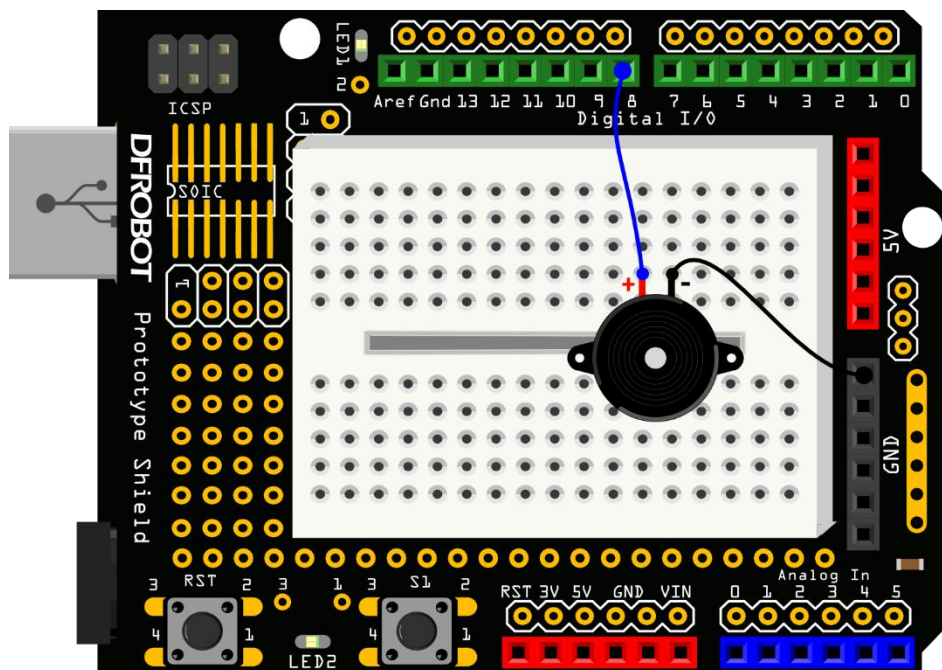
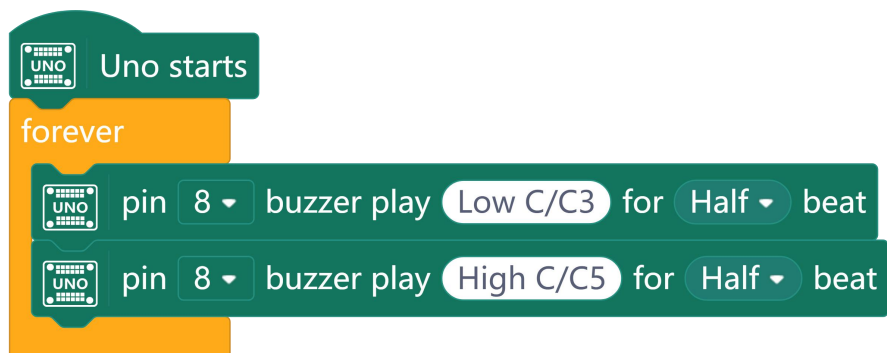




Figure 6-1 Connection

## Graphical Programming

Example Program 6-1:



Download the programs into Mind+, then you will hear that the buzzer buzzes a low pitch and a high pitch alternatively, just like a car alarm.

Module	Block	
		Control the pitch and beat the buzzer output.

## Code Learning 1

```

1.#include <DFRobot_Libraries.h>
2.// Create an object
3.DFRobot_Tone DFTone;
4.
5.// Main program starts
6.void setup() {
7.
8.}
9.void loop() {
10.    DFTone.play(8, 131, 250);
11.    DFTone.play(8, 523, 250);
12.}

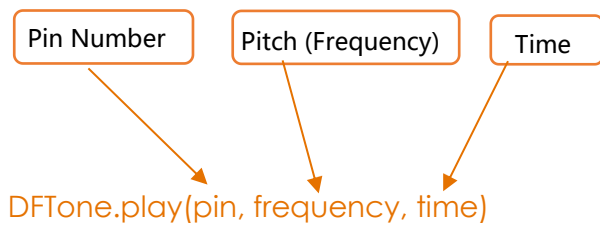
```

There is only one new function in the loop: DFTone.Play().

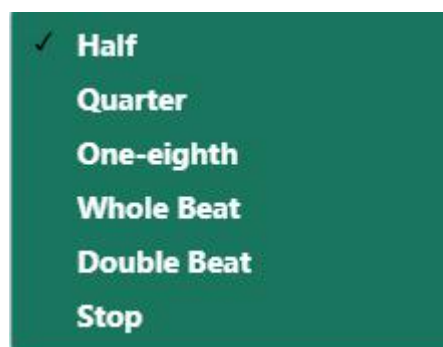
Block in Mind+:



The format of the function:



If you want to use buzzer to make more various sounds, then graphical programming Mind+ may cannot meet your requirements since there are limited options of beat in the block(1=1 second).



Although the graphical blocks in Mind+ are easy to use, the flexibility of the program is greatly reduced to a certain extent. So now, let's get rid of the graphical programming and complete the following project only by coding.

## Programming

Input the example program 6-2 into Mind+:

```
//Project 6 Alarm Device
float sinVal;
int toneVal;
#include <DFRobot_Libraries.h>
// Create an object
DFRobot_Tone DFTone;

void setup(){
  pinMode(8, OUTPUT);
}

void loop(){
  for(int x=0; x<180; x++){
    //Convert the degree of sin function into radian
```

```

        sinVal = (sin(x*(3.1415/180)));
        //Use sin function to generate the sound frequency
        toneVal = 2000+(int(sinVal*1000));
        //A signal for Pin 8 buzzer to output
        DFTone.play(8, toneVal,2);
    }
}

```

Once you have upload the code, the buzzer will start to emit sounds with pitches from low to high, and then high to low. Unlike the graphical block, coding makes the output result more flexible.

## Code Learning 2

First of all, define two variables:

```

float sinVal;
int toneVal;

```

As we mentioned in project 3, the "float" is a data type that allows a variable to store decimal values. The sinVal float variable will hold the sine value that will cause the tone to rise and fall. The toneVal variable will be used to take the value in sinVal and convert it to a frequency we require.

```

#include <DFRobot_Libraries.h>
// Create an object
DFRobot_Tone DFTone;

```

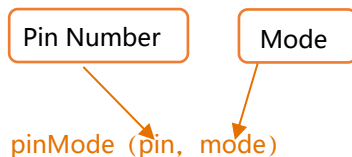
We will go into details about library and object in Chapter 10. It will be much easier to understand after having been used several times.

```

void setup(){
    pinMode(8, OUTPUT);
}

```

There is function `pinMode()` in the setup function, and its format as follows:



The `pinMode` function is used to configure a specific pin to behave either as an input or an output. We may not find this function in Mind+ since it has been encapsulated in other commands such as `digitalWrite(pin)`.

We use the function `sin()` to calculate the sine value of an angle. Unit: rad. To prevent the function outputting negative number, the range of the for loop is set to 0~179, which is 0~180° .

```
for(int x=0; x<180; x++){
```

Since the `sin()` function adopts the unit rad, we have to convert degree to rad:

```
sinVal = (sin(x*(3.1415/180)));
```

Then, convert the value into sound frequency the buzzer requires:

```
toneVal = 2000+(int(sinVal*1000));
```

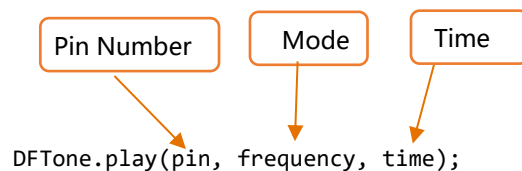
There is a new point to learn---convert float into integer.

The `sinVal` is a float variable that includes decimal value, but we want the frequency to be an integer. So the following statement can help us to change the decimal to integer.

```
int(sinVal*1000)
```

Multiply the `sinVal` by 1000 and plus 2000, then assign this value to `toneVal`. The output of the `tone()` will be taken as the sound frequency of the buzzer.

```
DFTone.play(8, toneVal,2);
```



## Hardware Review

### Buzzer

A buzzer is an electronic component that generates sound. There are generally two types: piezo buzzer and magnetic buzzer.

#### The difference of piezo buzzer and magnetic buzzer

A piezo buzzer generates sound because of the piezoelectric effect from the piezoelectric ceramic which drives the metallic diaphragm to vibrate. Normally, we have to provide voltage over 9V to have enough SPL. On the other hand, a magnetic buzzer can be driven to generate 85 dB by only 1.5V, but the consumption of the current will be much higher than the Piezo one.



According to the difference of control method, we can divide buzzer into active type and passive type.

### The difference between active buzzer and passive buzzer

The active buzzer has an internal oscillating source, and the buzzer will sound as soon as it is energized. While the passive buzzer does not have internal oscillating source, and it has to be driven with square wave and different frequency needed. It is like an electromagnetic changing input signal produces the sound, rather than producing a tone automatically.

In terms of appearance, the active buzzer has a long lead(positive) and a short lead(negative), and the passive buzzer only has two same leads.

In the kit package, we select the magnetic passive buzzer that is suitable for beginners. If you are interested in active buzzer, get one and try it!

Buzzers are widely used in various applications, for instance, infrared sensors and ultrasonic sensors for monitoring object approaching and altering people, gas sensors for gas leaking alarm, or using them as musical instruments to play notes. So amazing, right!

## Add-activities



1. Input the codes of this project into Mind+ manually.
2. Combine with an LED to make an alarm device. Tip: use the sin function to make the sound and lighting keep the same frequency.
3. Use the button we introduced in chapter 3 to make a doorbell. When the button is pressed, the buzzer makes a sound.

## Project 7 Temperature Alarm

Based on the last project, we are going to make a more practical application--temperature alarm. When the temperature arrives at the preset value, the buzzer makes sounds. Besides buzzer, we also need an LM35 temperature sensor here.

This is our first time to meet with sensors, what is a sensor? Literally, a sensor is a device that detects events and changes in its environment, and send the information to other electronics, usually a processor.

### Components

Buzzer    × 1        

LM35 Temperature    × 1

### Hardware Connection

Connect all parts together as below. Let the front side of the sensor (marked with LM35) face you, connect the three pins to 5V, Analog0, GND from left to right orderly.

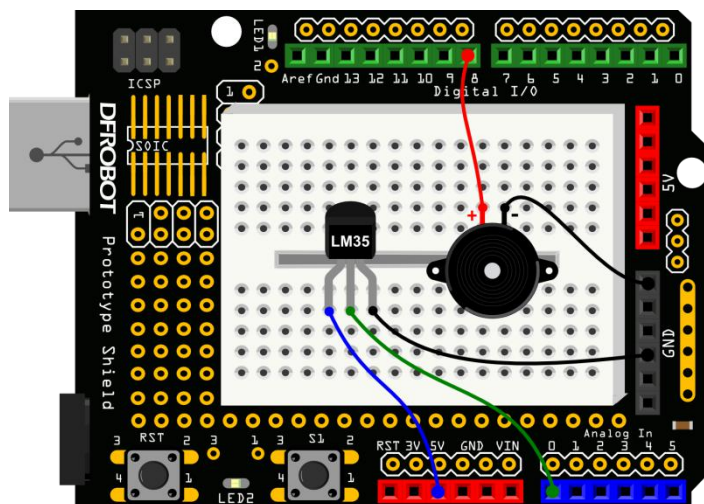
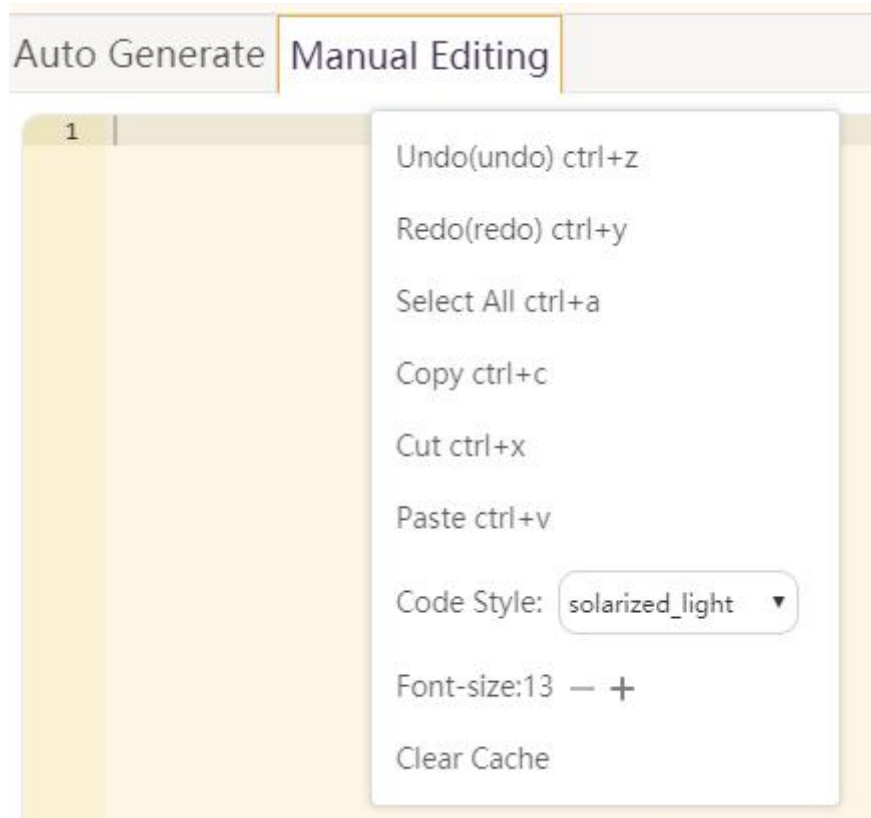


Figure 7-1 Temperature Alarm Connection

### Programming

From this chapter, we will start using coding to complete the project.

In Manual Editing interface, right-click the mouse, then you can select the code style and adjust the font size and more.



Example program 7-1:

```
//Project 7 Temperature Alarm
#include <DFRobot_Libraries.h>
// Create an object
DFRobot_Tone DFTone;

float sinVal;
int toneVal;
unsigned long tepTimer ;

void setup(){
    pinMode(8, OUTPUT);    // Configure the pin of buzzer
    Serial.begin(9600);    //Set band rate to 9600 bps
}

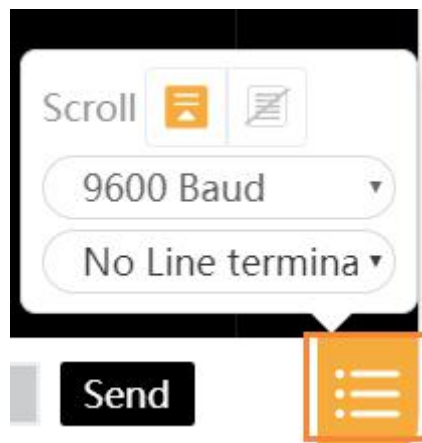
void loop(){
    int val;               //Store the value of LM35
    double data;           //Store the converted value of temperature
    val=analogRead(0);     //Connect LM35 to a analog port, and read value from that port
    data = (double) val * (5/10.24); // Convert the voltage to temperature

    if(data>27){           // If the temperature is higher than 27, the buzzer starts to make sounds
        for(int x=0; x<180; x++){
            //Convert the degree of sin function into rad

```

```
    sinVal = (sin(x*(3.1412/180)));  
    //Use sin function to generate the sound frequency  
    toneVal = 2000+(int(sinVal*1000));  
    //A signal for Pin 8 buzzer to output  
    DFTone.play(8,toneVal,2);  
  }  
} else {          // If the temperature is lower than 27, turn off the buzzer  
  DFTone.play(8,0,2);    //Turn off the buzzer  
}  
  
if(millis() - tepTimer > 500){    // Serial output temperature value every500ms  
  tepTimer = millis();  
  Serial.print("temperature: ");    // Serial output "Temperature"  
  Serial.print(data);              // Serial output temperature value  
  Serial.println("°C");            // Serial output temperature unit  
}  
}
```

After downloading the codes, click the box selected below to set the band rate to 9600.



Click the box below to open the serial port of Mind+.



Then the temperature measured by LM35 will be displayed on the terminal.

```
temperature: 27°C  
temperature: 27°C  
temperature: 27°C  
temperature: 27°C
```

## Code Learning

Most statements in this segment of codes are learned before. Can you understand the codes by yourself?

Firstly, load two libraries and create an object. Both of them will be fully explained in Chapter 10.

```
#include <DFRobot_Libraries.h>
// Create an object
DFRobot_LM35 LM35;
#include <DFRobot_Libraries.h>
// create an object
DFRobot_Tone DFTone;
```

Then, there are three variables followed:

```
float sinVal;
int toneVal;
unsigned long tepTimer ;
```

We have learned the first two variables before, let's just skip to the third one, tepTimer. It is an unsigned long data type to store time and output temperature value to serial port. Since the machine will run for a relatively long time, we choose a long integer. And the time cannot be negative, so we use the unsigned long variable.

We have been quite familiar with the first statement in setup(), set the buzzer to output mode. You may ask why we don't set output mode for LM35. That's because, LM35 is an analog and we don't need to set pin mode for analog value. pinMode is only used for defining digital pins.

Let's take a look at the second statement in setup(): serial.

```
Serial.begin(9600); //Set band rate to 9600 bps
```

## The communication Partner of Arduino---Serial

Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port. On UNO, pins 0(RX) and 1(TX) are used for serial communication. If you use this function, you cannot also use pins 0 and 1 for digital input or output.

On Arduino, the codes downloading are completed by serial. So when downloading, the digital pin0(RX) and 1(TX) will be occupied by USB. Besides, RX and TX are expected to be connected with nothing, otherwise, conflict may occur.

The pin RX and TX of Arduino are often occupied by other modules in using, especially when using with a wireless communication module. Therefore, we suggest you insert components after downloading your codes.

There are abundant functions for serial port, and the following are frequently used:

```
Serial.begin(9600); //Set band rate to 9600 bps
```

The function here is used to initialize the serial monitor to set the data transmission speed and to enable serial data transmission. Normally, the default setting in the serial monitor works for most applications, but for some wireless applications, we may have to set the band rate as the module's requirement.

Inside the loop(), there are two variables declared: val and data. They have been explained in the comments. Different from the former variables, these two are local variables and can only be used in the loop(). Refer to project 2 to review the difference between global variable and local variable.

```
val=analogRead(0); //Connect LM35 to a analog port, and read value from that port
```

Here comes a new function---analogRead(pin).

This function is used to read a value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3V) into integer values between 0 and 1023. Each read value corresponds to a value of voltage.

The readings of temperature are voltage values and will be output in the form of numbers within 0~1023. Every 10mV corresponds to 1 degree for LM35.

```
data = (double) val * (5/10.24); // Convert the voltage to temperature
```

The number read from the sensor will be a voltage value between 0~1023. It has to be divided by 1024 and multiplied by 5 to become a voltage value in 0~5V. Since 1 degree corresponds to 100mV, we have to multiply the converted voltage by 100 to get a temperature value in double datatype, and then assign it to the variable data.

Then the next part is an if statement for temperature judgement. However, unlike the if statement we introduced before, this is an if...else statement and can be used to determine two conditions.

The format of if...else:

```
if (expression) {  
    Statement 1;  
} else {  
    Statement 2;  
}
```

How if...else works?

If the expression is evaluated to true,

- Statements inside the body of if are executed.
- Statements inside the body of else are skipped from execution.

If the expression is evaluated to false,

- Statements inside the body of else are executed
- Statements inside the body of if are skipped from execution.

Well, let's back to our codes, the statements in the if statement are the same as that of project 6, so we will skip them here.

```
if(data>27){  
    for(int x=0; x<180; x++){  
        ...  
    }  
} else {  
    ...  
}
```

Do the if judgment first, if the temperate data is more than 27, the buzzer starts to make sound. Otherwise, execute the statement inside else, turn off the buzzer.

Besides the temperate alarm, we have to display the temperature on the serial constantly. Here we used the millis() function(explained in project 3) again to help to send out date every 500ms.

How to display the received data on serial monitor?

```
Serial.print(data);           // Serial output temperature value  
Serial.println("°C");         // Serial output temperature unit
```

Prints data to the serial as human-readable a ASCII text.



This command can take many forms.

1. Numbers are printed using an ASCII character for each digit.
2. Floats are similarly printed as ASCII digits, defaulting to two decimal places.
3. Bytes are sent as a single character. Characters and strings are sent as. For example:
  - `Serial.print(78)` gives "78"
  - `Serial.print(1.23456)` gives "1.23"
  - `Serial.print('N')` gives "N"
  - `Serial.print("Hello world.")` gives "Hello world."

Besides that, an optional second parameter specifies the base format to use: permitted values are BIN(binary, or base 2), OCT(octal, or base 8), DEC(decimal, or base 10), HEX(hexadecimal, or base 16).

The main difference between `Print()` and `println()` is that `println` method prints the string and moves the cursor to a new line while `print()` won't make the cursor go to a new line.

`Serial.write()` is also an common statement. It is used to write binary data to the serial port.

The code below may get you confused. Is the data here a character string? How does it output a number?

```
| Serial.print(data); // Serial output temperature value
```

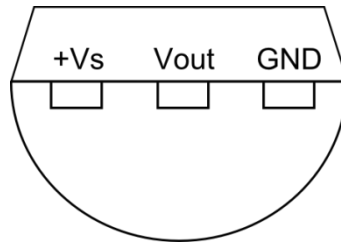
Actually, the data is a variable we declared before and we have assigned a value into it.

## Hardware Review

### LM 35

LM is a common temperature sensor, easy to use. With special calibration, the accuracy of the sensor can reach up to  $\pm 1/4^{\circ}\text{C}$ .

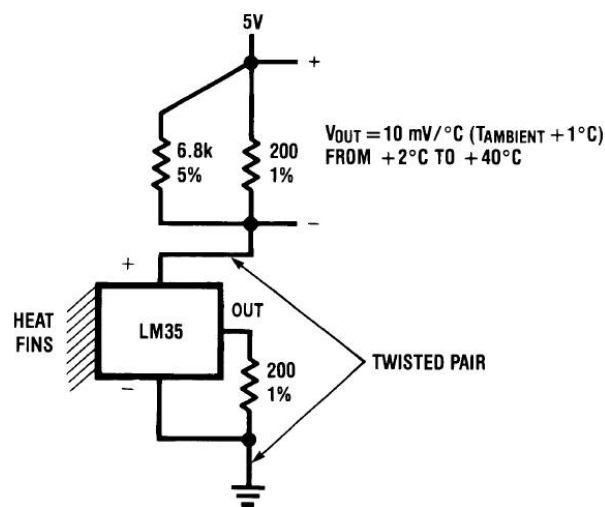
The LM35 temperature sensor has 3 pins:  $V_s$  for power source, GND for -, and  $V_{out}$  for voltage output.



Calculation Formula:

$$V_{out} = 10\text{mV}/^{\circ}\text{C} * T(^{\circ}\text{C}) \text{ (Temperature Range: } +2^{\circ}\text{C} \sim 40^{\circ}\text{C)}$$

This formula comes from the datasheet of LM35. And you can find more details about how to convert the temperature data into voltage.



## Add- activities

1. Input the codes into Mind+, compile and debug.
2. Add an LED for this project. Configure different colors and buzzer sounds for the various temperature range.

For example:

- ❖ When the temperature is lower than 10 or higher than 35, a red LED turns on and the buzzer makes a rapidly-oscillating sound.
- ❖ When the temperature falls between 25 and 35, a yellow LED turns on and buzzer makes a smooth-oscillating sound.
- ❖ When the temperature falls between 10 and 25, a green LED turns on and the buzzer is off.

The temperature alarm can be used in plant cultivation. Think what else you can do with this sensor!

## Project 8 Tilt Sensor

A tilt sensor has a metallic ball that is designed to move the two pins of the instrument from the “on” and “off” position. Tilt sensors allow you to detect orientation or inclination. Here we are gonna use it together with an LED. When vibration is detected, the LED turns on, otherwise, the LED keeps off.

### Components

Tilt sensor SW200D × 1 

5mm LED × 1 

220Ω Resistor × 1 

### Hardware Connection

Like the push button in project 3, we need to add a pull-down resistor for the sensor, and a current-limiting resistor to the LED.

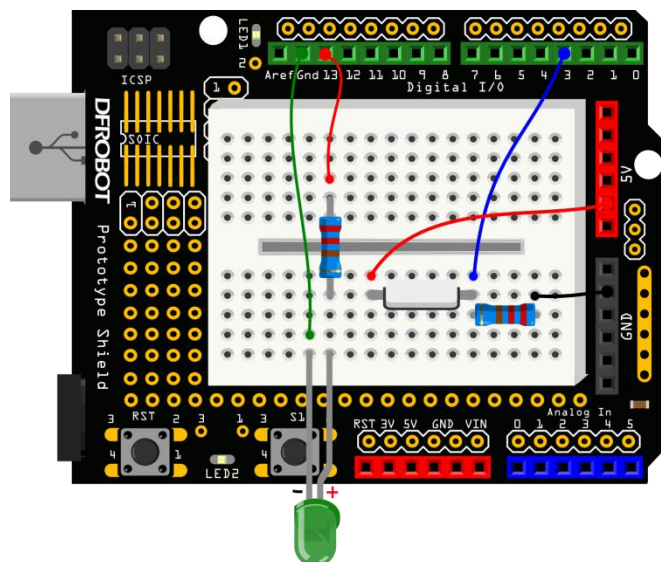


Figure 8-1 Tilt Sensor Connection Diagram

## Programming

Input the example program 8-1:

```
//Project 8 – Tilt Sensor
int SensorLED = 13;           //Define LED to digital pin 13
int SensorINPUT = 3;          //Connect the tilt sensor to interrupt 1, also digital pin 3
volatile unsigned char state = 0;
void blink();                  //Declare function before using

void setup() {
    pinMode(SensorLED, OUTPUT);    //Configure LED as output mode
    pinMode(SensorINPUT, INPUT);   //Configure the tilt sensor as input mode

    //When voltage level change from Low to High, interrupt 1 will be triggered and the blink function will be called
    attachInterrupt(1, blink, RISING);
}

void loop(){
    if(state!=0){                // If state is not 0
        state = 0;               // Assign 0 to the state
        digitalWrite(SensorLED,HIGH); // Turn on the LED
        delay(500);              //Delay 500ms
    }
    else{
        digitalWrite(SensorLED,LOW); //Otherwise, turn off the LED
    }
}

void blink(){                    //Interrupt the function blink()
    state++;                     //Once the interrupt is triggered, the state keeps incrementing.
}
```

Upload the codes to your board, when shaking the board, the LED turns on. Once no shaking is detected, the LED keeps off.

## Code Learning

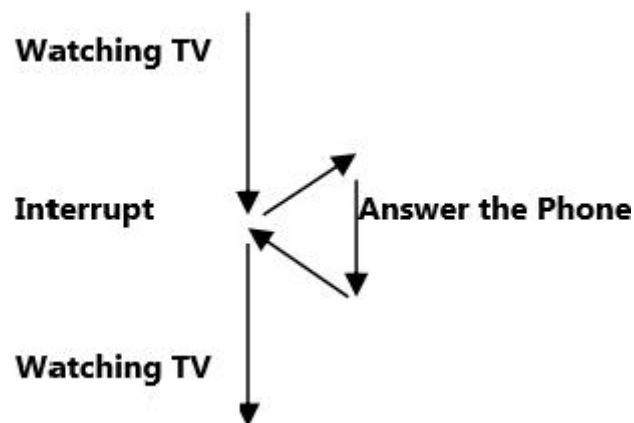
The whole program works as follows: when there is no interruption, the program keeps running and the LED stays off(Low). When there is an interrupt(shake the board and the tilt sensor is activated), the program jumps to the interrupt blink(), at the same time, the state keeps incrementing. When the state is not detected to be 0 by "if" statement, the LED turns on and

assign 0 to the state again for the next interrupt. If there is no interrupt detected, the LED backs to the initial state(off).

Then let's look at this new function: `attachInterrupt()`.

## What is an Interrupt?

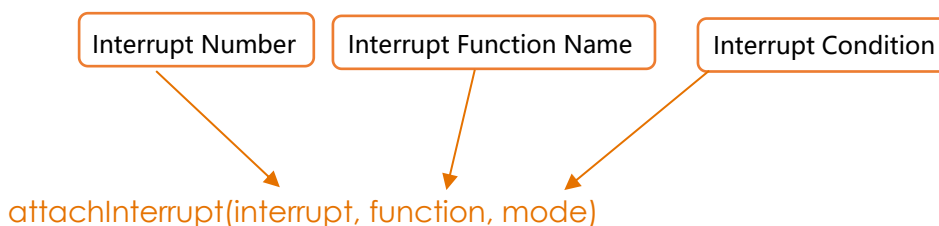
Imagine you are watching TV at home, and the phone rings, this time you have to stop to answer the phone. After that, you can continue to watch TV. In this case, the phone call is an interrupt and the ringing of the phone marks the interrupt.



Now let's back to the `attachInterrupt()`. It sets a function to be called when an external interrupt occurs. On the Arduino Uno, there are two external interrupt pins: INT0(pin 2) and INT1(pin 3). 0 and 1 are the interrupt numbers. The specific pins with interrupts and their mapping to interrupt number vary from board to board.

Check the details on Arduino website at <http://arduino.cc/en/Reference/AttachInterrupt>.

There are three parameters in `attachInterrupt()`:



**interrupt:** the number of the interrupt, 0 or 1. If it is selected to 0, please connect to digital pin 2. For 1, connect to pin 3.

**function:**

- The function is called upon when the interrupt occurs.
- It takes no parameters and returns nothing.
- Inside the attached function, **delay()** won't work and the value returned by **millis()** will not increment.
- Serial data received while in the function may be lost.

**mode:** define when the interrupt should be triggered. Four constants are predefined as valid values:

**LOW** to trigger the interrupt whenever the pin is low.

**CHANGE** to trigger the interrupt whenever the pin changes value.

**RISING** to trigger when the pin goes from low to high.

**FALLING** to trigger when the pin goes from high to low.

That's all about the function `attachInterrupt()`. Now let's back to our codes:

```
attachInterrupt(1, blink, RISING);
```

The first parameter, the interrupt number, is 1. so connect the tilt sensor to digital pin 3; blink is the interrupt function we will call later; RISING to trigger when the pin goes from low to high.

Why we selected "RISING" mode here?

The tilt sensor is like a push button, when it doesn't detect any vibration, the pin 3 is Low. Once the signal is detected, the pin becomes High. The pin 3 goes from low to high in this process, so we chose "RISING".

## Hardware Review

### Tilt Sensor

Tilt sensor, may also be known as tilt switches, rolling ball sensors or vibration sensor, can control the flow of current in a circuit by the rolling of two balls. Seen as below:



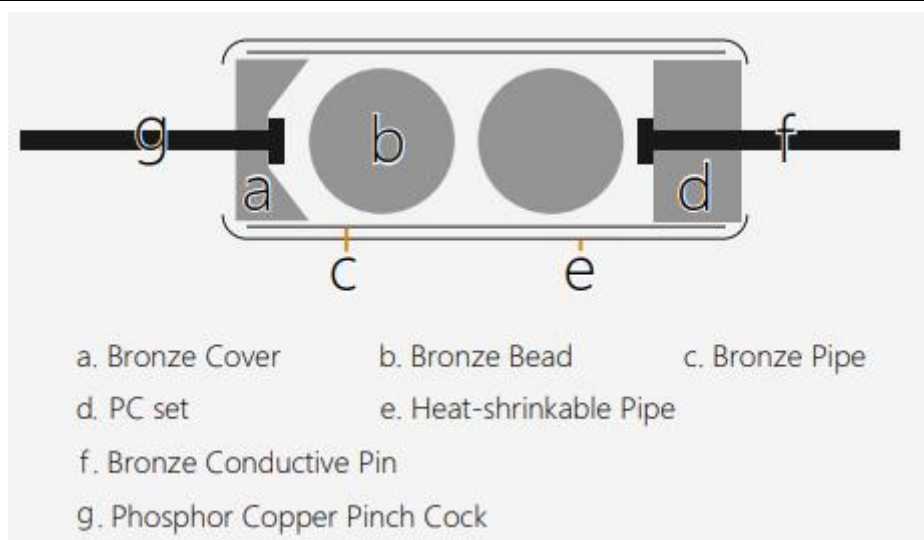


Figure 8-2 Tilt Sensor Diagram


This type of sensor contains a cylinder and two balls inside. Only one end of the cylinder (Golden color) has two conductive poles. When the sensor is oriented so that that end is downwards, the ball rolls onto the poles and a circuit is completed. When tilting the sensor to another end (silver color), the circuit cannot be connected so the LED won't turn on.

## Project 9 Light Sensitive LED

Here we are gonna introduce you to a new component: photodiode(light sensor). It is used to convert the light into current or voltage. As one type of photoresistor, a photodiode usually comprises of special photoconductive materials whose electrical resistance drops dramatically when exposed to light. The stronger light in the surroundings, the lower the resistance of the photodiode.

In this project, we will use the photodiode to make a light-sensitive LED that can automatically adjust itself in accordance with the intensity of light around it. During the night, the LED keeps on, and in the daytime, it turns off.

### Components

Light Sensor    × 1    

5mm LED        × 1    

220 $\Omega$  Resistor × 1    

10K Resistor    × 1    

### Hardware Connection

Like an LED, the photodiode has a long lead(+) and short lead(-), and it should be connected with the 10k resistor.

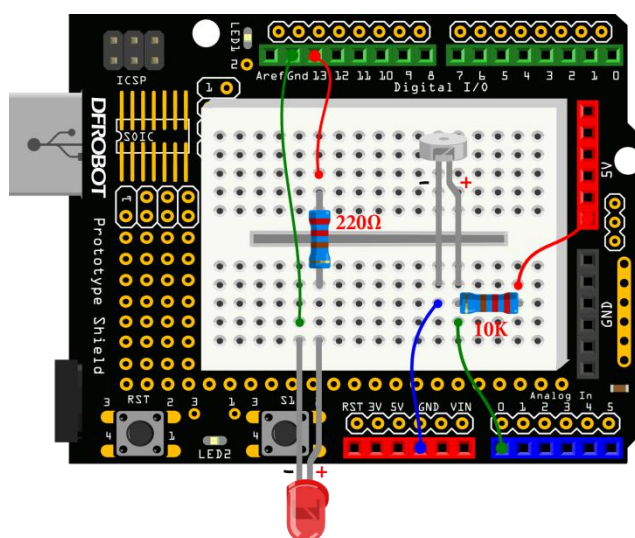


Figure 9-1 Light Sensitive LED

## Programming

Input the example program 9-1:

```
//Project 9 - Light Sensitive LED
int LED = 13;           //Configure LED to digital pin 13
int val = 0;            //Configure the voltage value of photodiode in analog pin0

void setup(){
  pinMode(LED,OUTPUT);  // Set LED as output mode
  Serial.begin(9600);   // Set serial band rate to 9600
}

void loop(){
  val = analogRead(0);  // Read voltage from 0~1023
  Serial.println(val);  // Read the change of the voltage from serial monitor
  if(val<1000){         // Turn off the LED when lower than preset value
    digitalWrite(LED,LOW);
  }else{               // Otherwise, the LED turns on
    digitalWrite(LED,HIGH);
  }
  delay(10);           // Delay 10ms
}
```

After uploading the code, the LED will be on. If you shine the photodiode with strong light, the LED will gradually die out. When the light is removed, it'll light up again.

## Code Learning

The codes of this project must be much easier to comprehend, aren't they? So we are gonna just give a simple explanation here.

Similar to the LM35 temperature sensor in project 7, the output from the photodiode is an analog and can be read via an analog pin. We don't need to set input/output mode for the analog.

Once there is light shining on the photodiode, the read analog value will decrease. We set the upper limit value to 1000 here. You can adjust it according to the actual scene. Put the whole device in a place where the LED will turn off, then open the serial monitor, and check the displayed value. Replace 1000 with the number you get here.

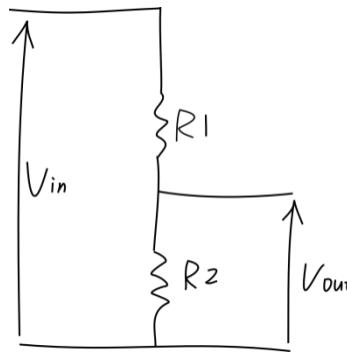
It is a good way to debug the program by analyzing the readings in the serial monitor.

## Hardware Review

### Photodiode

A photodiode is a semiconductor device that converts light into an electrical current. The current is generated when photons are absorbed in the photodiode. The stronger light in the surroundings, the lower the resistance of the photodiode. The voltage decreases as the resistance of two ends reduces. As a result, the readings got from the analog port become smaller(the analog value from 0~1023 corresponds to voltage 0~5V).

Why does the voltage decrease? Actually, it is acting as the “voltage divider rule”. Let's see this typical voltage divider circuit below:



The input voltage  $V_{in}(5V)$  is connected to the two resistors. By measuring the voltage passing through  $R_2$ , we can get the resistance of the photodiode with the following formula:

$$V_{out} = \frac{R_2}{R_1 + R_2} \times V_{in}$$

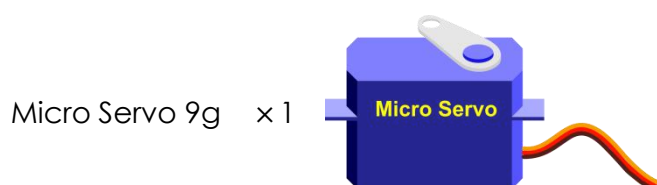
In the project,  $R_1$  is the 10K resistor and  $R_2$  is the photodiode. In a dark place, the resistance of  $R_2$  is relatively large, so the  $V_{out}$  will almost reach 5V. Once light shines on the photodiode, the value of  $R_2$  decreases, so does its voltage. The resistance of  $R_1$  should not be too low, and 1~10K would be the best, otherwise, the voltage dividing ratio will not be obvious.

## Project 10 Drive a Servo

Servo motor is a type of motors whose output shaft can be moved to a specific position by sending it the coded signal. Commonly, most servos can rotate 180 degrees. While some of them can turn 360 degrees. Servos are suitable for applications requiring angle controlling, such as smart robot car and robotic joints.

The servo in the kit is a micro 9g 180° servo. We plan to make it rotate back and forth in 0~180 degrees in this project.

### Components



### Hardware Review

The servo has three leads. The color of the leads varies between servos but the red lead is always 5V and GND will either be black or brown. The other lead is the signal lead and it is usually orange or yellow. The signal lead should be connected to digital pin 9.

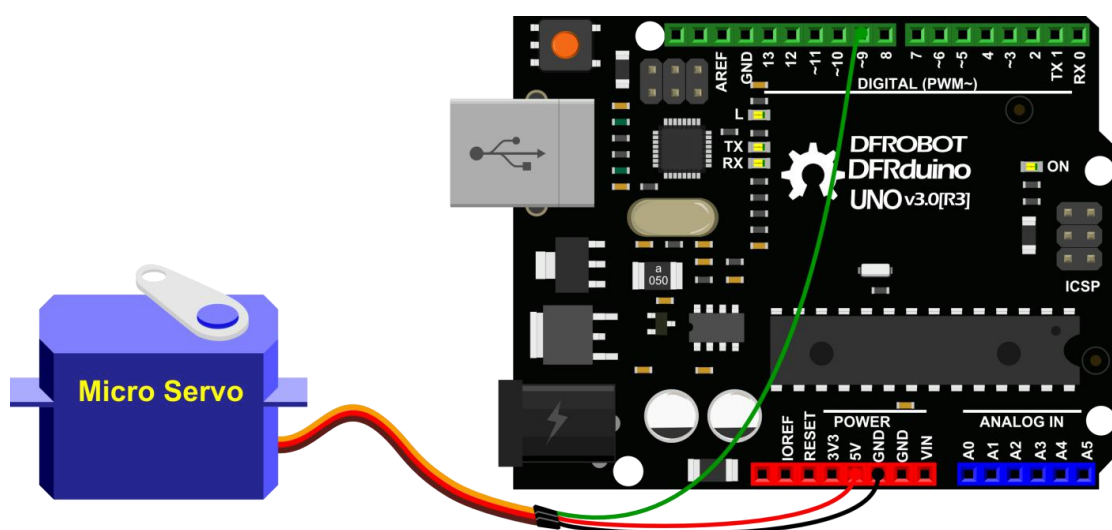


Figure 10-1 Servo Connection

## Programming

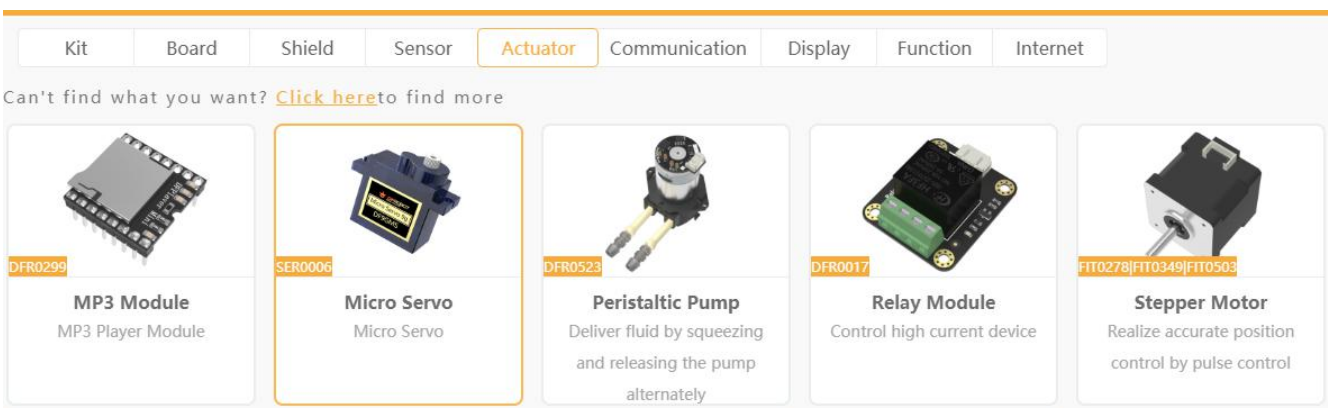
Input the example program 10-1:

```
//Project 10 - Drive a Servo
#include <DFRobot_Servo.h>           // Declare and call the servo.h library
Servo myservo;                       // Create a servo object
int pos = 0;                         // Variable pos to store the position of servo
void setup() {
    myservo.attach(9);               // Attach the servo to digital pin 9
}

void loop() {
    for(pos = 0; pos < 180; pos += 1){ // Servo turns from 0° to 180° in steps of 1°
        myservo.write(pos);           // Write angle into the servo
        delay(15);                    // Waits 15ms for the servo to reach the specified position
    }
    for(pos = 180; pos >= 1; pos -= 1){ // Servo turns from 180° to 0° in steps of 1°
        myservo.write(pos);           // Write angle into the servo
        delay(15);                    // Waits 15ms for the servo to reach the specified position
    }
}
```

The sketch starts from inserting <Servo.h> library. We haven't learned it before, but we can get familiar with the operation and codes of servo in Mind+.

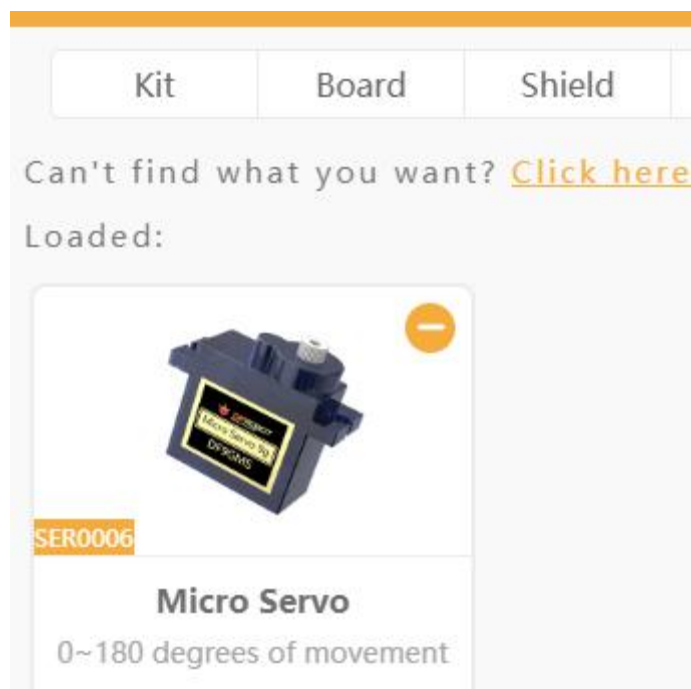
Step 1: click "Extension"->"Actuator"



Step 2: Search "Servo" in the search bar at the upper-right corner.



Step 3: select "0~180° " servo



Step 4: back to the main interface, drag the block of servo to the "Manual Editing" section.





Drag the blocks into the "Manual Editing" section, the related c codes will appear and these codes can be freely copied, but do not forget to revise the parameters.

Download the codes, then the servo will rotate from 0 to 180 degrees, one degree at a time. When it has rotated 180 degrees, it will begin to rotate in the other direction until it returns to the home position.

## Command Learning

Module	Block	Description
 Actuator		Set the rotation angle of servo

## Code Learning

The program begins with the calling of the library <DFRobot\_Servo.h>

```
#include <DFRobot_Servo.h>           // Declare and call the servo.h library
```

This library is already in Mind+1.6.1\Arduino\libraries\DFRobot\_Servo\DFRobot\_Servo.h.

## What is a library?

In programming, a library refers to a collection of files, programs, routines, scripts or functions that can be referenced in the programming code. It is like a big locker with a lot of small boxes for storing things.

## Object

We need to put a label for the big locker, academically, it is called "**object**".

```
Servo myservo;                       // Create a servo object
```

## Call a function in the library

There is a statement in setup():


```
myservo.attach(9);
```

Here we need to call a function in the Servo library. Different the function calling mentioned before, we have to point out the object first, and then write out the function name so that the program can know where to find the function.

The format of function calling in a library:

`object.function ();`

Don't miss the dot "." in the middle. myservo is the object we set before, and the function is:



`attach (pin);`

There is a parameter pin in the function. We can set it to any digital pin except 0 and 1. Here we use pin 9.

In the main program, there are two "for" statements. The first one makes the servo rotate from 0 to 180 degrees in steps of 1 ° while the latter starts from 180 to 0 degrees in steps of 1 degree.

The "for" loop includes a function: myservo.angle(pos). Similarly, point out the name of the library first. The parameter of the function is "angle", unit: degree.

If you want to know more functions in DFRobot\_Servo, go to our website.

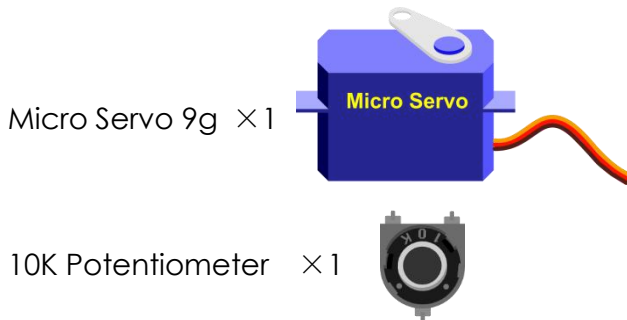
<http://mindplus.cc/en.html>

<https://www.dfrobot.com/>

## Project 11 Controllable Servo

We have learned how to drive a servo in the last chapter, now let's go further than that! Here we are gonna control the rotation of a servo by external signal--a potentiometer. Of course, you can replace the potentiometer with other devices, such as tilt switch, push button or light sensor. So many ways to play, just try it.

### Components



### Hardware Connection

The potentiometer can work as a variable resistor, and it has three pins. Connect the two pins at the same side to 5V and GND, the pin on the other side to analog A0 for signal input.

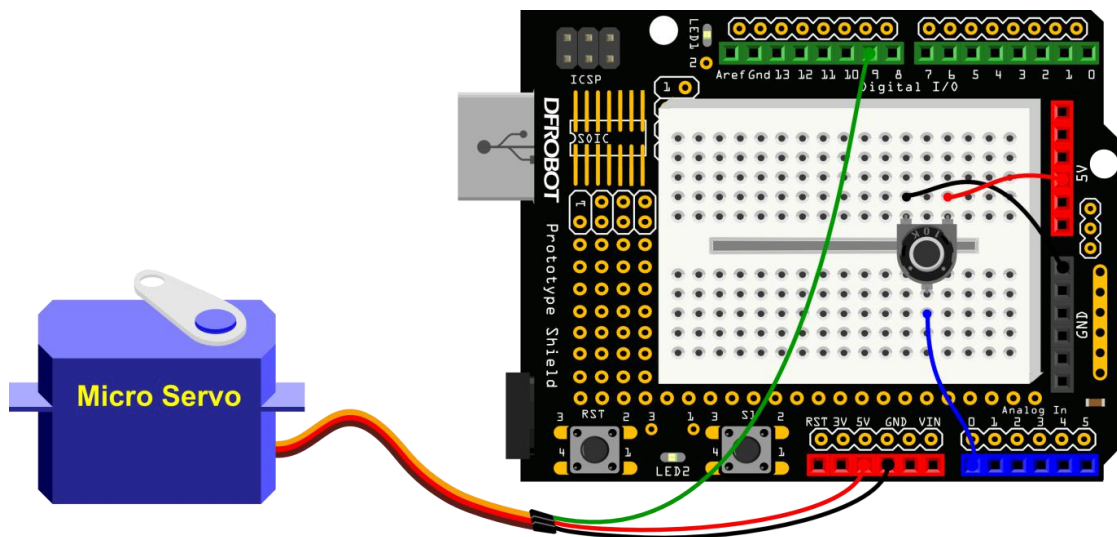


Figure 11-1 Controllable Servo

## Programming

Example Program 11-1:

```
//Project 11 - Controllable Servo
#include <DFRobot_Servo.h>           //Declare and call the library Servo.h
Servo myservo;                      //Create a servo object

int potpin = 0;                     //Connect the potentiometer to analog A0
int val;                             //Variable to store the value read from pin A0

void setup() {
    myservo.attach(9);               //Attach the servo on pin 9 to the servo object
}

void loop() {
    val = analogRead(potpin);        //Read the value of analog A0 (potentiometer) between
    0~1023
    val = map(val, 0, 1023, 0, 180); //Scale the potentiometer value to angle value for servo in
    0 and 180
    myservo.write(val);               //Write angles into the servo
    delay(15);                       //Delay 15ms to turn the servo to the specified position
}
```

Upload the codes to the board, rotate the potentiometer, then the servo will rotate accordingly.

## Code Learning

First, call the library <DFRobot\_Servo.h> and create the related object, and then configure the potentiometer to Analog pin 0 to read its value.

Next, let's move to the map function:

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

The function `map()` will re-map a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

The related block in Mind+ is:

`map` `0` from `[ 0 , 1023 ]` to `[ 0 , 255 ]`

### The parameters in the function map():

value: the number to map

fromLow: the lower bound of the value's current range

fromHigh: the upper bound of the value's current range

toLow: the lower bound of the value's target range

toHigh: the upper bound of the value's target range

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

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

The function also handles negative numbers as well, so that this example is also valid and works well.

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

```
val = map(val, 0, 1023, 0, 180); //Scale the potentiometer value to angle value for servo in  
0 and 180
```

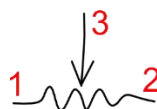
This code aims to map the readings of analog A0 from 0~1023 to 0~180° .

## Hardware Review

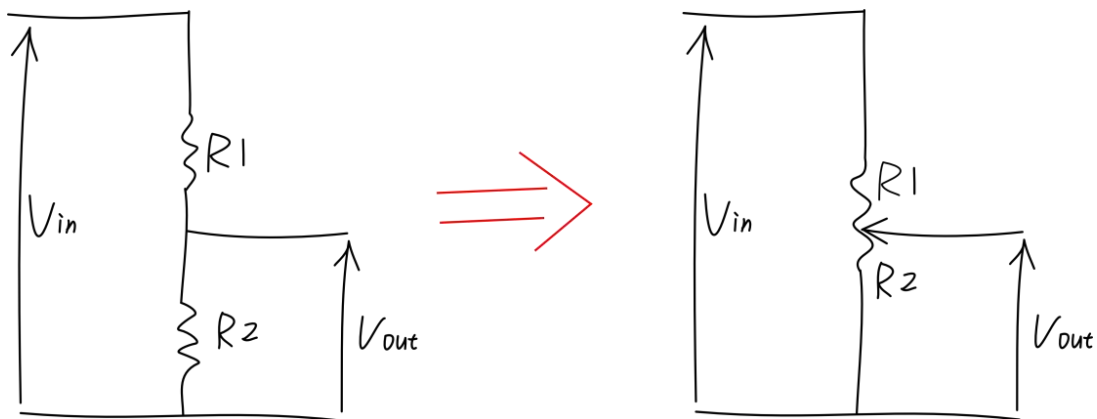
### Potentiometer

A potentiometer is a device that allows you to manually modify its resistance between a minimum (normally 0ohms) and a maximum (5K, 10k, or 20K ohms). We selected a 0~10 kΩ potentiometer in this project. The left and right ends need to connect to power source. The resistance can be changed by the middle pin, and the voltage in the circuit changes with the resistance. We read the voltage via analog pin0 and convert it to the corresponding angle value for the servo.

The potentiometer in the circuit can be marked as the diagram shows below:



A potentiometer works in a similar way of the voltage divider in project 9. Simply put, the potentiometer can be divided into 2 resistors ( $R_1$  and  $R_2$ ) by the shaft. The position of the shaft determines the resistance of  $R_1$  and  $R_2$ . As  $R_1$  and  $R_2$  are changed, the voltage changes. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and we read 0. When the shaft goes all the way in the other direction to the limit, there would be 5 volts going to the pin, and we read 1023. In between, "analogRead()" returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.



## Project 12 Interactive RGB LED

We have learned how to make an RGB LED to light up in various colors, now let's make it interactive: use 3 potentiometers to control the LED so that we can make any colors on our own.

### Components

5mm RGB LED × 1



220Ω Resistor × 1



10K Potentiometer × 3



### Hardware Connection

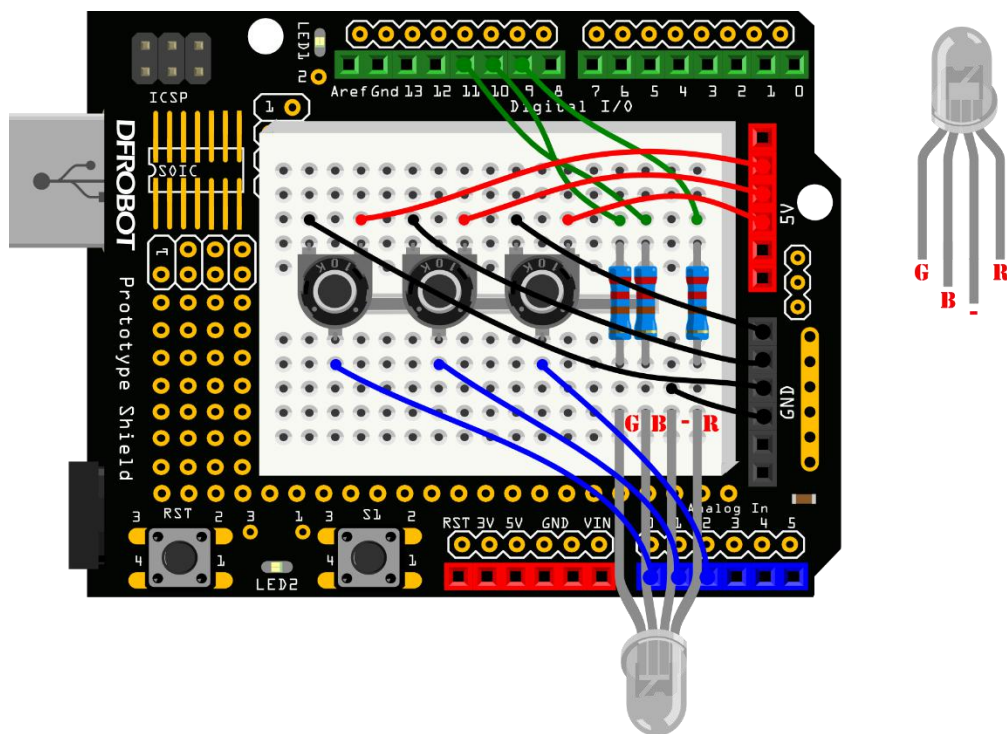


Figure 12-1 Interactive RGB LED Connection

### Programming

Example Program 12-1:



```
//Project 12 -Interactive RGB LED
int redPin = 9;           // R – digital 9
int greenPin = 10;        // G – digital 10
int bluePin = 11;         // B – digital 11
int potRedPin = 0;         // Potentiometer 1 – analog 0
int potGreenPin = 1;       // Potentiometer 2 – analog 1
int potBluePin = 2;        // Potentiometer 3 – analog 2
void colorRGB(int, int, int); // Declare a function
void colorRGB(int red, int green, int blue){ //Display colors
    analogWrite(redPin,constrain(red,0,255));
    analogWrite(greenPin,constrain(green,0,255));
    analogWrite(bluePin,constrain(blue,0,255));
}

void setup(){
    pinMode(redPin,OUTPUT);
    pinMode(greenPin,OUTPUT);
    pinMode(bluePin,OUTPUT);
    Serial.begin(9600); // Initialize the serial port
}

void loop(){
    int potRed = analogRead(potRedPin); // potRed to store value read from analog P0
    int potGreen = analogRead(potGreenPin); // potGreen to store value read from analog P1
    int potBlue = analogRead(potBluePin); // potBlue to store value read from analog P2







    int val1 = map(potRed,0,1023,0,255); //Map the voltage value from 0~1023 to analog value 0~255
    int val2 = map(potGreen,0,1023,0,255);
    int val3 = map(potBlue,0,1023,0,255);
    //Serial output the value of Red, Green, Blue
    Serial.print("Red:");
    Serial.print(val1);
    Serial.print("Green:");
    Serial.print(val2);
    Serial.print("Blue:");
    Serial.println(val3);
    colorRGB(val1,val2,val3); // Configure the analog value for the RGB LED
}
```

Download the codes, rotate the three potentiometers to make the LED emit different light.

## Project 13 DIY a Fan

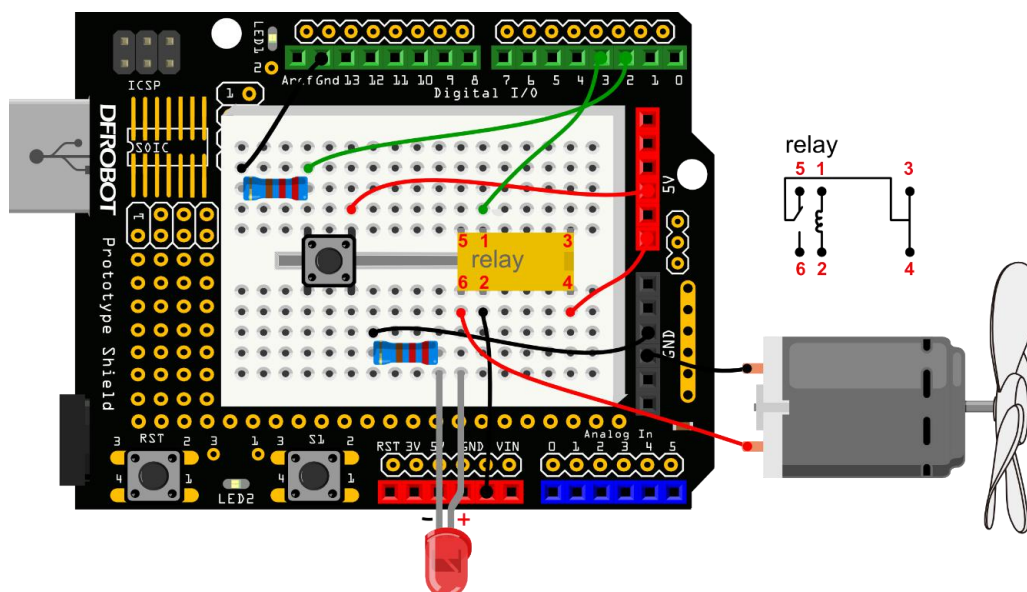
We will DIY a fan in this chapter. There are two new components: relay and DC motor. A relay is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current. Here the relay is used to control the motor.

### Components

5mm LED	x 1	
220 $\Omega$ Resistor	x 2	
Button	x 1	
Relay HRS1H-S -DC5V	x 1	
Micro Motor	x 1	
Fan	x 1	

### Hardware Connection

Connect all parts together as the diagram below: one ends of the button goes to 5V, the other to GND. Add a 220  $\Omega$  resistor for the button to stop the unused inputs from floating about randomly when there is no input. The relay has 6 pins marked with numbers 1 to 6. Pin 1 and 2 are the signal inputs and should be connected to a digital pin and GND of Arduino respectively. The rest are signal outputs and we will only use pin 4 and 6.



## Programming

Example Program 13-1:

```
//Project 13 – DIY a Fan
int buttonPin = 2;           // Configure the button digital pin 2
int relayPin = 3;           // Configure the relay to digital pin3
int relayState = HIGH;      //The initial state of relay is High
int buttonState;            // Record the current button state
int lastButtonState = LOW;  // Record the last button state
long lastDebounceTime = 0;
long debounceDelay = 50;    //Eliminate the debounce

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, relayState);    // Set the initial state of relay
}

void loop() {
  int reading = digitalRead(buttonPin);  //read the value of button and store it in 'reading'

  // Once the change of data is detected, record the current time
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  // Wait for 50ms to evaluate if the last button state is the same as the current one
  // If it is not, change the button state
  // At the same time, if the button state is High(pressed), change the relay state
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

      if (buttonState == HIGH) {
        relayState = !relayState;
      }
    }
  }
  digitalWrite(relayPin, relayState);

  // Change the last button state
  lastButtonState = reading;
}
```

Control the motor and LED via a button.

## Code Learning

We have learned the most statements in this segment of codes so here we will only introduce the new one: debounce.

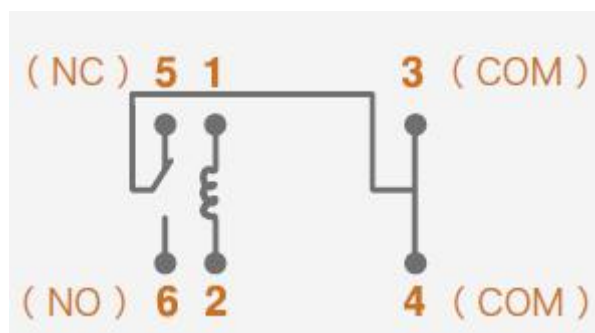
```
if (reading != lastButtonState) {  
    lastDebounceTime = millis();  
}  
if ((millis() - lastDebounceTime) > debounceDelay) {  
    if (reading != buttonState) {  
        ...  
    }  
}
```

Push buttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program. Each time the input pin goes from LOW to HIGH (because of a push-button press), the output pin is toggled from LOW to HIGH or High to LOW. There is a minimum delay between the toggles to debounce the circuit. So we need to debounce an input, which means checking twice in a short period of time to make sure the push button is definitely pressed, and the `millis()` function is used to keep track of the time passed since the button was pressed.

## Hardware Review

### Relay

A relay is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current. Let's take a look at the internal structure of a relay:



This relay has 6 pins. Pin 1 and 2 should be connected to the digital pin and GND to power the relay. There is a coil between the pin 1 and pin 2. When powered up, the current flows through the wire of the coil to generate a magnetic field, then the NO pin and the COM pin will be connected together. Otherwise, the NC pin will be connected with COM pin.

We used pin 4 and 6 to control the motor and LED, you can replace them with pin 3 and 6.

### What's the difference between DC motor, stepper motor and servo?

DC motor is a two-wire continuous rotation motor and the two wires are power and ground. When the supply is applied, a DC motor will start rotating until that power is detached. Reverse the flow of current to change the rotation direction. A motor runs continuously at a high RPM. These revolutions of the motor shaft can not be controlled to a specific angle, but the speed can be adjusted. The speed of a DC motor is pretty fast so we don't recommend using it on a smart robot car directly.

Stepper motor has a gearing set on the DC motor to step down the speed and increase its torque. This makes it more suitable for vehicle applications. Its speed can be controlled by PWM.

A servo is also a motor. It controls the position of the motor by a feedback system, as we saw in the servo projects covered. Servos are practical for using in robotics arms.

## Project 14 IR Remote Controlled LED

In this project, we are gonna learn a new component: IR Receiver. IR receivers are devices designed to send and receive a coded infrared signal from one device to another. They can be easily found in our daily life, such as TV remote controller, audio system, projectors, etc.

We will use an IR receiver to make a remote-controlled LED. Turn on or off the LED via the red power button on the remote controller.

First of all, let's carry out a warm-up experiment to get to know how to use IR receiver and remote controller.

### Components

IR Receiver x 1



Mini IR Remote Controller x 1



### Hardware Connection

The connection is quite easy. Just do it as the diagram shows. The pin Vout should go to the digital pin 3 on Arduino.

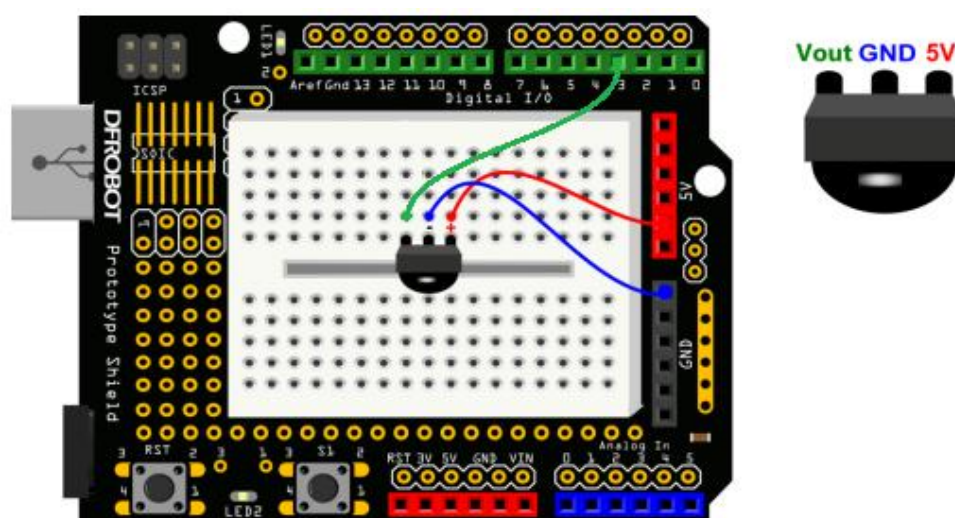


Figure 14-1 IR Receiver Connection

## Programming

Example Program 14-1:

```
#include <DFRobot_IRremote.h>      //Import an IR library
String result;                     // Create a variable
IRremote_Receive remoteReceive_3; // Create an object

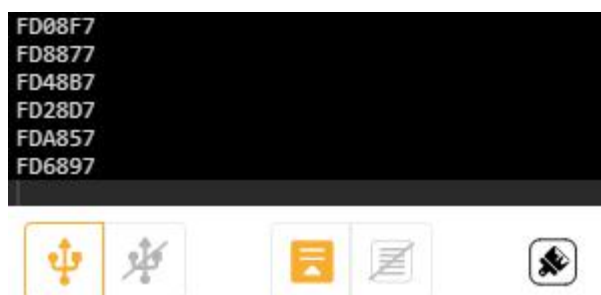
// Main program starts
void setup() {
    Serial.begin(9600);             // Configure serial band rate
    remoteReceive_3.begin(3);       //Enable IR decode
}
void loop() {
    result = (remoteReceive_3.getIrCommand()); //Store the IR readings into the variable
    if ((result!="0")) {                //Determine if the received value is not 0
        Serial.println(result);        //Serial output the effective readings
    }
}
```

Upload the codes to you board, open the serial port and set the band rate to 9600.



\*Drag the left line of the serial monitor to enlarge its window, and then the band rate will be displayed.

When the configuration is finished, press the button on the remote controller towards the infrared receiver. Each time you press a button on the remote control, a unique hexadecimal code is generated and can be seen on the serial monitor. As the following shows, if everything is right, then the received code should be displayed in the form of six digits starting with FD.

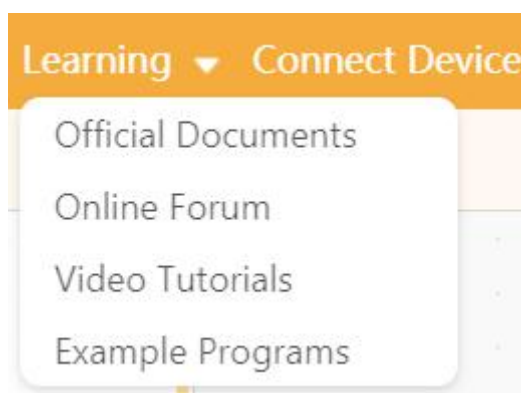




```
203EC837
FDB04F
FD8877
FD807F
FDA05F
FD906F
FDA05F
FCC837
```

If there is interference during signal transmission or the controller didn't aim at the IR receiver, we may get wrong codes like "203EC837" and "FCC837" in the picture above.

\*In Mind+, it is stipulated for a library that the IR receiver can only be connected to pin 2 and 3. If encountering any problems, you can refer to the FAQ in official documents or feed us back on our forum.



In this program, the key point we need to learn is:


```
result = (remoteReceive_3.getIrCommand());    //Store the IR readings into the variable
```

In Mind+, after the command **remoteReceive\_3.getIrCommand()** got readings from the IR receiver, its data will be stored in the register in the form of string and then be removed when it has been used. Therefore, the data cannot be used repeatedly and we have to create a new string variable to store it.

Since the IR decoding is quite complicated, we will not get into the details here. The most difficult part has already been done by the advanced programmers, and they provided us with this DFRobot\_IRremote library. We can directly use this library without completely comprehending its principle.

Now we have finished our warm-up experiment, let's back to the subject.

## Components

5mm LED × 1 



220 $\Omega$  Resistor     $\times 1$     IR Receiver     $\times 1$     Mini Remote Controller     $\times 1$ 

## Hardware Connection

We add an LED and Resistor based on the warm-up experiment: connect the LED to digital pin 10.

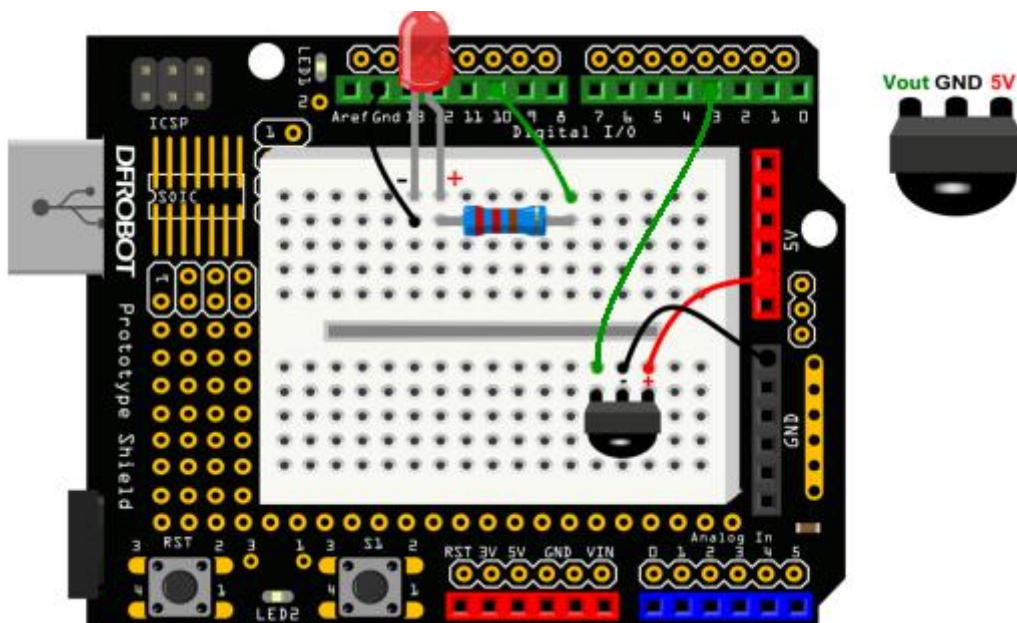


Figure 14-2 IR controlled LED Connection

## Programming

Example Program 14-2

```

#include <DFRobot_IRremote.h>

String result;                                // Create a register variable
IRremote_Receive remoteReceive_3;           // Create an object
int RECV_PIN = 3;
int ledPin = 10;                             // LED – digital 10
boolean ledState = LOW;                      // ledstate for storing the state of the LED

// Main Program Starts
void setup() {

```

```

    Serial.begin(9600);
    remoteReceive_3.begin(RECV_PIN);
    pinMode(ledPin,OUTPUT);           // Configure the LED as "Output"
}
void loop() {
    result = (remoteReceive_3.getIrCommand());
    if ((result!="0")) {
        Serial.println(result);
        //Once the code of power button is received, the LED state changes from HIGH to LOW, or from LOW
        to HIGH.
        if(result == "FD00FF"){
            ledState = !ledState;      //Reverse
            digitalWrite(ledPin,ledState); //Change the state of the LED
        }
    }
}

```

## Code Learning

Define the IR receiver at the beginning of the program.

```

#include <DFRobot_IRremote.h>

String result;           // Define a register variable
IRremote_Receive remoteReceive_3; // Create an object
int RECV_PIN = 3;
int ledPin = 10;         // LED – digital 10
boolean ledState = LOW;  // ledstate for storing the state of the LED

```

We define a variable "ledState" here to store the LED state. Since there are only two states (0 or 1) for the LED, we use the boolean type of variable .

The setup() function includes the configuration of serial port and digital pin, and enabling the IR decoding.

In the loop(), test if the IR code is received, and then store the data in the variable "result".

```

    if ((result!="0")) {

```

Once there is data received, the program will execute these two steps: 1. determine if the IR code of the power button is received.

```

        if(result == "FD00FF"){

```

2. Change the state of the LED

```

            ledState = !ledState;      //Reverse
            digitalWrite(ledPin,ledState); //Change the state of the LED

```

You might not be so familiar with "!". It is a logical NOT. "!ledState" is the opposite state of "ledState" "!" is used in the variable that only holds 2 states, or boolean type of variable.

## Add-activities

1. Add a fan for the project. Use the remote controller to control the LED and the fan.
2. DIY a remote-controlled project, for example, a small figure that can move with servos controlled by infrared signals.

## Project 15 IR Controlled 8-Segment Display

An 8-segment display is a form of electric display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays. They are used in many day to day consumer devices like microwave ovens, washing machines, and air conditioners. Each segment on the display can be controlled individually, just like a regular LED. Now, let's feel its glamour together!

### Components 1

8-Segment Display × 1



220Ω Resistor × 8



### Hardware Connection

As the following picture shows, the 8-segment display has 10 pins. The pins on the top are connected to Arduino's digital pin 2 to 5. The pins on the bottom (including decimal point pin) go to the digital pin 6 to 9, and there are 8 current-limiting resistors for the display.

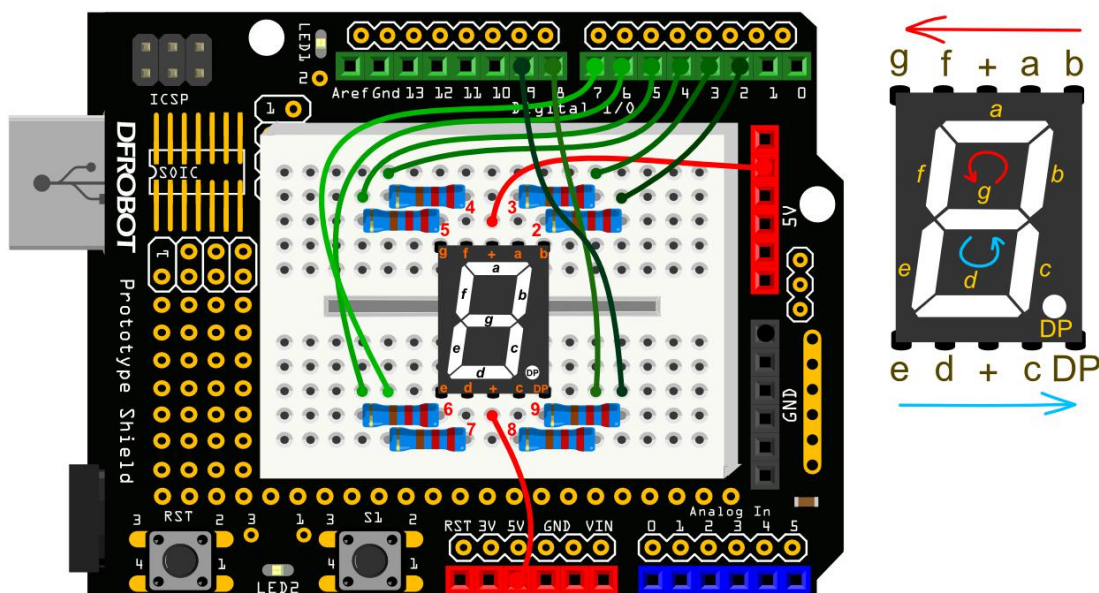


Figure 15-1 8-segment Display Connection

## Programming 1

Example Program 15-1:

```
//Project 15 – IR Controlled 8-segment Display
void setup(){
    for(int pin = 2 ; pin <= 9 ; pin++){          // Define digital pin 2 to 9 as output
        pinMode(pin, OUTPUT);
        digitalWrite(pin, HIGH);
    }
}

void loop() {
    // Display number 0
    int n0[8]={0,0,0,1,0,0,0,1};
    //Display the array of n0[8] in digital pins 2~9
    for(int pin = 2; pin <= 9 ; pin++){
        digitalWrite(pin,n0[pin-2]);
    }
    delay(500);

    // Display number 1
    int n1[8]={0,1,1,1,1,1,0,1};
    // Display the array of n1[8] in digital pins 2~9
    for(int pin = 2; pin <= 9 ; pin++){
        digitalWrite(pin,n1[pin-2]);
    }
    delay(500);

    // Display number 2
    int n2[8]={0,0,1,0,0,0,1,1};
    // Display the array of n2[8] in digital pins 2~9
    for(int pin = 2; pin <= 9 ; pin++){
        digitalWrite(pin,n2[pin-2]);
    }
    delay(500);

    // Display number 3
    int n3[8]={0,0,1,0,1,0,0,1};
    // Display the array of n3[8] in digital pins 2~9
    for(int pin = 2; pin <= 9 ; pin++){
        digitalWrite(pin,n3[pin-2]);
    }
    delay(500);

    // Display number 4
```

```
int n4[8]={0,1,0,0,1,1,0,1};
// Display the array of n4[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n4[pin-2]);
}
delay(500);

// Display number 5
int n5[8]={1,0,0,0,1,0,0,1};
// Display the array of n5[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n5[pin-2]);
}
delay(500);

// Display number 6
int n6[8]={1,0,0,0,0,0,0,1};
// Display the array of n6[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n6[pin-2]);
}
delay(500);

// Display number 7
int n7[8]={0,0,1,1,1,1,0,1};
// Display the array of n7[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n7[pin-2]);
}
delay(500);

// Display number 8
int n8[8]={0,0,0,0,0,0,0,1};
// Display the array of n8[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n8[pin-2]);
}
delay(500);

// Display number 9
int n9[8]={0,0,0,0,1,1,0,1};
// Display the array of n9[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n9[pin-2]);
}
```

```
delay(500);
```

```
}
```

Once the code is uploaded, the number 0 to 9 will be constantly display on the 8-segment display. To understand the code, we have to learn the structure of the 8-segment display first.

## Hardware Review

### 8-segment Display

An 8-segment display is a combination of 8 LEDs(the decimal point-DP-is the 8th), which can be lit in different combinations to display the numbers 0 to 9 and letters A to G. Each LED is used to illuminate one segment of unit and the 8<sup>th</sup> LED is for illuminating the DOT. With 8 digital pins, all the LEDs on the 8-segment display can be completely controlled. For example, if we want to display the number "0", then we just need to glow the LEDs on the pins b, a, f, e, d, c.

The following is the pinout of an 8-segment display. The pins b→a→f→g→e→d→c→DP should be connected to Arduino digital pins 2 to 9.

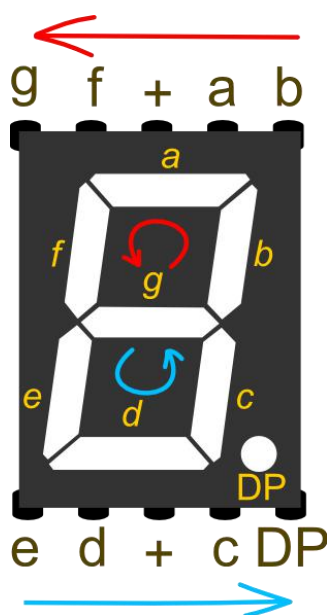


Figure 15-2 Pinout

### What's the difference between common cathode and common anode?

All the anodes or cathodes of an 8-segment display are connected to one point, and it becomes a common anode or common cathode. You need to connect the common anode to the +5V and each individual LED to a resistor. Connect that resistor to an output pin, then

write a LOW to that pin to turn the LED on, High to turn it off. However, With a common cathode you connect it to ground and connect each LED's anode through a resistor to the output pin, then give that pin a HIGH to turn the LED on, Low to turn it off.

We are using a common anode 8-segment display in this project.

## Code Learning 1

As we mentioned in the hardware review part, the 8-segment display needs to be connected to 8 digital pins. So at first, we have to define 8 digital pins as output in coding, and here a for loop can greatly simplify the work. The LED b, a, f, g, e, d, c and DP on the segment display respectively correspond to the Arduino's digital pin 2 to 9.

```
for(int pin = 2 ; pin <= 9 ; pin++){           // Define digital pin 2~9 as output
    pinMode(pin, OUTPUT);
    digitalWrite(pin, HIGH);
}
```

Set the digital pins 2 to 9 as output mode and initialize them as High. Since it is the common anode 8 segment display, the LED will be in off state currently. (You can also configure the pins one by one, but the program would be quite redundant in that way. )

Well, in the main program part, we need to make the display show numbers 0 to 9. Take the displaying number "0" as an example:

## Array

```
int n0[8]={0,0,0,1,0,0,0,1};
```

Here we have to dive into dissecting the concept of array first. An array is a collection of variables that are accessed with an index number. Note that when declaring an array of char type, one more element than your initialization is required to hold the required null character. In the project, we declare an array of int type, name it as n0, and initialize it with 8 values. Arrays are zero-indexed, that is to say, the first element of the array is at index 0, hence, the index number of this array is 0 to 7. The fourth element is at index 3 (n0[3]), and its value is 1. The number 8 inside the square brackets (n0[8]) means that there are 8 elements in the array.

Enter a for loop again when defined the array.

```
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n0[pin-2]);
}
```



In this loop, we will configure the state of pin 2~9 as High(1) or LOW(0) using an array. For instance, when pin=2, the n0[pin -2] will become n0[0] that is the first element in the array, and its value is 0. In this way the pin 2 is set to Low, the LED connected to pin 2(LED b) will light up. When pin =3, the LED turns on, and so forth.

The whole loop process is shown below:

```
pin=2  n0[0] =0  digitalWrite(2,0)  LED b on
pin=3  n0[1] =0  digitalWrite(3,0)  LED a on
pin=4  n0[2] =0  digitalWrite(4,0)  LED f on
pin=5  n0[3] =1  digitalWrite(5,1)  LED g off
pin=6  n0[4] =0  digitalWrite(6,0)  LED e on
pin=7  n0[5] =0  digitalWrite(7,0)  LED d on
pin=8  n0[6] =0  digitalWrite(8,0)  LED c on
pin=9  n0[7] =1  digitalWrite(9,1)  LED DP off
```

That's all for displaying the number "0" on an 8-segment display. Now can you try displaying other numbers?

Once you have totally understood the example program 1, you may be interested in the following example that provides you with a more easier way to realize the same effect.

## Programming 2

### Example Program 15-2

```
//Project 15 – IR-controlled 8-segment Display
int number[10][8] =
{
  {0,0,0,1,0,0,0,1}, //Display 0
  {0,1,1,1,1,1,0,1}, //Display 1
  {0,0,1,0,0,0,1,1}, //Display 2
  {0,0,1,0,1,0,0,1}, //Display 3
  {0,1,0,0,1,1,0,1}, //Display 4
  {1,0,0,0,1,0,0,1}, //Display 5
  {1,0,0,0,0,0,0,1}, //Display 6
  {0,0,1,1,1,1,0,1}, //Display 7
  {0,0,0,0,0,0,0,1}, //Display 8
  {0,0,0,0,1,1,0,1}  //Display 9
};

void numberShow(int i){ //Call the function to display number
```

```
    for(int pin = 2; pin <= 9 ; pin++){
        digitalWrite(pin, number[i][pin - 2]);
    }
}

void setup(){
    for(int pin = 2 ; pin <= 9 ; pin++){        // Set digital pin 2~9 as output mode
        pinMode(pin, OUTPUT);
        digitalWrite(pin, HIGH);
    }
}

void loop() {
    for(int j = 0; j <= 9 ; j++){
        numberShow(j);        //Call the function numberShow(), display number 0~9
        delay(500);
    }
}
```

## Code Learning 2

Compare the two examples, can you find the differences between them? Apparently, the second is more simplified. In example 1, we created 10 one-dimensional arrays while in this one, we use only one two-dimensional array and all things done. Don't make it complicated. The usages of them are pretty much the same.

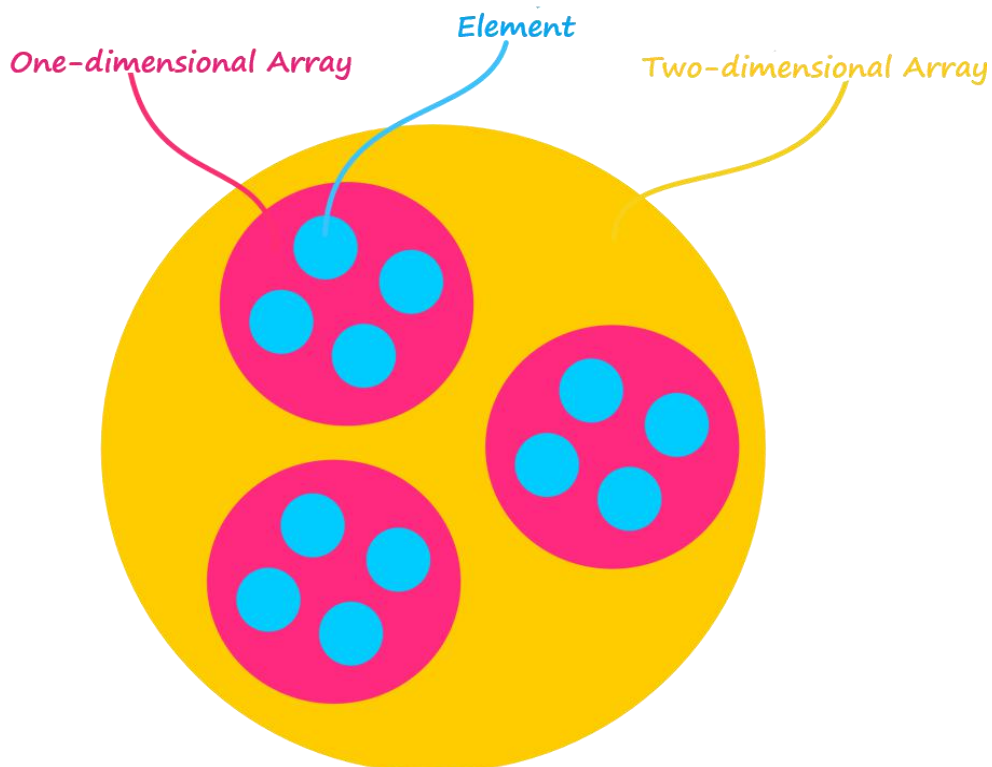
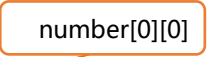


Figure 15-3

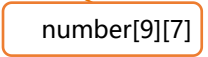
It can be seen from the picture above that, the one-dimensional array is composed of elements and the two-dimensional array is composed of one-dimensional arrays.

In the last sketch, we created 10 arrays and each array includes 8 elements that correspond to the pin state on the 8-segment display from b to DP. Now, we are combining the 10 one-dimensional arrays into a two-dimensional array.



```

int number[10][8] =
{
  {0,0,0,1,0,0,0,1}, //Display 0
  {0,1,1,1,1,1,0,1}, //Display 1
  {0,0,1,0,0,0,1,1}, //Display 2
  {0,0,1,0,1,0,0,1}, //Display 3
  {0,1,0,0,1,1,0,1}, //Display 4
  {1,0,0,0,1,0,0,1}, //Display 5
  {1,0,0,0,0,0,0,1}, //Display 6
  {0,0,1,1,1,1,0,1}, //Display 7
  {0,0,0,0,0,0,0,1}, //Display 8
  {0,0,0,0,1,1,0,1}  //Display 9
};
  
```



This index number of the two-dimensional array starts with 0. Can you find the number[0][0]? It is the first element of the first line, and its value is 0. How about the number[9][7]?

```

void numberShow(int i){          //Call the function to display number
  for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin, number[i][pin - 2]);
  }
}

void loop() {
  for(int j = 0; j <= 9 ; j++){
    numberShow(j);              //Call the function numberShow(), display number 0~9
    delay(500);
  }
}
  
```

In the main function loop(), the for loop will make the variable “j” increment by 1 from 0 to 9, Once “j” is assigned with a value, the “numberShow()” function runs accordingly. As shown below:

To begin with,  $j=0$ , the `numberShow(j)` is `numberShow(0)`, then it skips to the function `numberShow()` above. Now the  $i=0$ . Since the pin's initial value is 2, the `digitalWrite()` should be `digitalWrite(2,number[0][0])`. The value of `number[0][0]` is equal to 0 so the function becomes `digitalWrite(2, 0)`, which means the pin 2 is configured as LOW. As a result, the LED b turns on. After that, the loop goes from  $\text{pin}=3$ ,  $\text{pin}=4$ , ..., to  $\text{pin}=9$ , then the loop finishes and the number "0" will be showed on the 8-segment display.

Let's review how did we manage to display the number "0" in the first sketch.

```
// Display number 0
int n0[8]={0,0,0,1,0,0,0,1};
//Display the array of n0[8] in digital pins 2~9
for(int pin = 2; pin <= 9 ; pin++){
    digitalWrite(pin,n0[pin-2]);
}
```

There is no much difference in the principle: write the pin 2~9 to control the LED b~DP to display any numbers we want.

When the loop of `numberShow(0)` completed, it goes back to the for loop function:

$j=1$    `numberShow(1)`    $i=1$    `number[1][pin-2]`   Display the number 1

$j=2$    `numberShow(2)`    $i=2$    `number[2][pin-2]`   Display the number 2

$j=3$    `numberShow(3)`    $i=3$    `number[3][pin-2]`   Display the number 3

.....

$j=9$    `numberShow(9)`    $i=9$    `number[9][pin-2]`   Display the number 9

That's all for the code analysis. Now, take you time, try to comprehend the difference of these two programs.

Since we have got known to the principle of 8-segment display and IR receiver, now let's combine them together to make an IR-controlled Display. The Arduino controller processes and delivers the signal transferred from the Mini remote controller to the IR receiver to the LED module. Then when we press 0 on the remote controller, a number "0" will be displayed on the 8-segment display. Besides, the functions of + and - are also feasible.



```
//Project 15 - IR-controlled 8-segment Display
```

```
#include <DFRobot_IRremote.h> //Insert IRremote.h library
```

```
int RECV_PIN = 2; //Define the variable RECV_PIN to 2
```

```
IRremote_Receive remoteReceive_2; //Create an object
```

```
String results; //Define the register variable results
```

```
int currentNumber = 0; //The variable is for storing the current number
```

```
String codes[12]= //The array is for holding the IR codes sent by the IR remote controller
```

```
{
  "FD30CF", "FD08F7", // 0,1
  "FD8877", "FD48B7", // 2,3
  "FD28D7", "FDA857", // 4,5
  "FD6897", "FD18E7", // 6,7
  "FD9867", "FD58A7", // 8,9
  "FD20DF", "FD609F", // +, -
};
```

```
int number[10][8] = //The array is for holding the number displayed on the 8-segment Display
```

```
{
  {0,0,0,1,0,0,0,1}, //0
  {0,1,1,1,1,1,0,1}, //1
  {0,0,1,0,0,0,1,1}, //2
  {0,0,1,0,1,0,0,1}, //3
  {0,1,0,0,1,1,0,1}, //4
  {1,0,0,0,1,0,0,1}, //5
  {1,0,0,0,0,0,0,1}, //6
  {0,0,1,1,1,1,0,1}, //7
  {0,0,0,0,0,0,0,1}, //8
  {0,0,0,0,1,1,0,1} //9
};
```

```
void numberShow(int i) { //Display numbers on the 8-segment display
```

```
  for(int pin = 3; pin <= 10 ; pin++){
    digitalWrite(pin, number[i][pin - 3]);
  }
}
```

```
void setup(){
```

```
  Serial.begin(9600); //Set band rate to 9600
```

```
  remoteReceive_2.begin(RECV_PIN); //Enable the IR decoding
```

```
  for(int pin = 3 ; pin <= 10 ; pin++){ //Set the digital pin2~9 to output mode
    pinMode(pin, OUTPUT);
```

```
        digitalWrite(pin, HIGH);
    }
}

void loop() {
    //Determine if the decoded data is received, and store the received data into the variable results
    results = (remoteReceive_2.getIrCommand());
    if (results != "0") {
        for(int i = 0; i <= 11; i++){
            //Determine if the IR code of button 0~9 is received
            if(results == codes[i]&& i <= 9){
                numberShow(i);        //Display the number 0~9 accordingly
                currentNumber = i;    //Assign the current displayed value to variable currentNumber
                Serial.println(i);
                break;
            }

            // Determine if the decreasing IR codes are received, and the current value should not be 0.
            else if(results == codes[10]&& currentNumber != 0){
                currentNumber--;        //Decrease the value of variable currentNumber by 1
                numberShow(currentNumber);    //Show the decreased value on the display
                Serial.println(currentNumber); //Serial print the decreased value
                break;
            }

            //Determine if the increasing IR codes are received, and the current value should not be 0.
            else if(results == codes[11]&& currentNumber != 9){
                currentNumber++;        //Increase the value of variable currentNumber by 1
                numberShow(currentNumber);    //Show the increased value on the display
                Serial.println(currentNumber); //Serial print the increased value
                break;
            }
        }
    }

    Serial.println(results); //Check the IR codes on the serial monitor
}
}
```

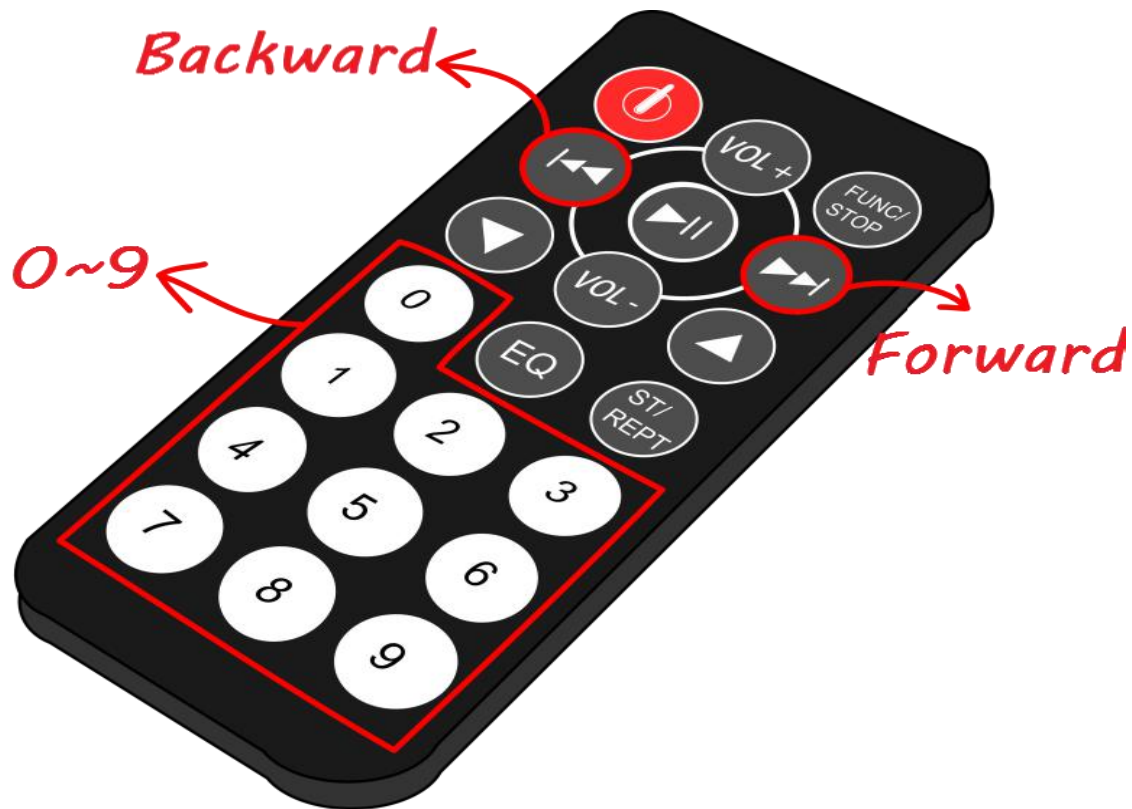


Figure 15-5 IR Remote Controller

Upload the codes to your board, press the button on the remote controller, and see whether the related number is displayed on the 8-segment display.

### Code Learning 3

```
int currentNumber = 0; //The variable is for storing the current number
```

We define a variable of int type to store the current number, and it acts as a reference point for number decreasing or increasing.

An array is applied to store these IR codes. Replace the codes to realize other more functions through the buttons on the remote controller.

```
String codes[12]= //The array is for holding the IR codes sent by the IR remote controller
{
  "FD30CF", "FD08F7", // 0,1
  "FD8877", "FD48B7", // 2,3
  "FD28D7", "FDA857", // 4,5
  "FD6897", "FD18E7", // 6,7
  "FD9867", "FD58A7", // 8,9
}
```



```
"FD20DF", "FD609F",          // +, -
};
```

Then, define a two-dimensional array `number[10][8]`. By calling the elements in the array, we can assign these values to the Arduino digital pins to control the connected LEDs of the 8-segment display. At last, use the `numberShow()` function to make the number showed on the display.

Similarly, the `setup()` function is for configuring band rate and pin mode, and enabling IR decoding, etc.

In the main function `loop()`, determine if the IR code is received first, and then store the received data in the variable `results`.

```
if (results != "0") {
```

Once the data is received, the program needs to do two actions: 1. Find which button is pressed according to the received IR code. 2. Let the 8-segment display do reaction when the pressed button is found. Let's see how to complete these two tasks in coding.

Task 1:

There are three possibilities: 1) The 8-segment display shows number 0 to 9 when the related button is pressed. 2) Every time the backward button pressed, the number displayed currently decreases by 1 until it reaches 0. 3) Every time the forward button pressed, the number displayed currently increases by 1 until it reaches 9.

How to determine the three conditions? The `if` statement totally works here. However, we selected "if...else if" instead of "if...else" here. What's the difference between them? The `if` portion is absolutely mandatory for both, but `else if` should always be followed by a condition expression.

Back to our program, the three condition are shown as follow:

```
if(results == codes[i]&& i <= 9){
```

```
if(results == codes[10]&& currentNumber != 0){
```

```
if(results == codes[11]&& currentNumber != 9){
```

In the first if statement, we have to judge if the received data results.value are from the IR code array codes[0] to codes[9]. If the condition is evaluated to be true, display number 0 to 9.

The second one is for determining whether the backward command is received, also means code[10]=0xFD20DF, at the same time, the currently displayed number should not be 0.

The third if statement is for judging whether the forward command is received, code[11]=0xFDA857, and the currently displayed number should not be 9.

Now, there is only one question left---how to find the element in the array? Actually, we just need to add a for loop before the if statement to make the variable i change within 0 to 11 repeatedly.

After the received IR codes are confirmed, the program starts to execute the second task, so there should be corresponding codes following each "if" statement. The last statement we are gonna learn is **break**.

The break statement is used to terminate the loop. When the two tasks are completed, we use a break statement to jump out of the current loop, by which to ensure the 8-segment display only refreshes once in one for loop, eliminate the mis-displaying caused by interference, and improve the efficiency of code execution.

That's all for the code analysis. Perhaps you cannot understand the whole program at first since they are the most complicated among all projects, but take it easy, true knowledge comes from practice.

## Add-activities

Make a DIY remote controlled-robot based on this project, for instance, a movable toy robot. You could use servos, press buttons, and more other components to control the robot. Give free rein to your imagination, you can achieve a lot with Mind+ and Arduino!