Getting Started

It is recommended to first read **Tutorial.pdf** in the unzipped folder you created.

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be **used only when there is adult supervision present** as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. **Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.**
- When the product is turned ON, activated or tested, some parts will move or rotate. **To avoid injuries to hands and fingers keep them away from any moving parts!**
- It is possible that an improperly connected or shorted circuit may cause overheating. **Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down!** When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro:bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

Copyright

Contents

Chapter 0 Processing

Processing is a software used to write programs that can run on computers. Processing software is free and open source running on the Mac, Windows, and GNU/Linux platforms, which is the same as Arduino software. In fact, the development of Arduino software is based on Processing software, and they still have similar interface.

Programs written with Processing are also called sketches, and Java is the default language. Java language and C++ language have many similarities, so readers who have learned our basic tutorial are able to understand and write simple Processing sketches quickly.
This tutorial will introduce how to install and use processing software on Raspberry Pi through some electronic circuit projects. Chapters and sequence in this tutorial are basically the same as those in the C and python language tutorial. Our elaborate electronic circuits and interactive project with Processing are attached at the end, including virtual instruments, games (2D and 3D versions), etc.

Installing Processing Software

Installing the installation package for Processing Software.

Make sure your RPi always has internet access during the download process. Please download the corresponding Processing software installation package according to your Raspberry Pi system bitness.
You can check the the current system bitness with the following command:

```
getconf LONG_BIT
```

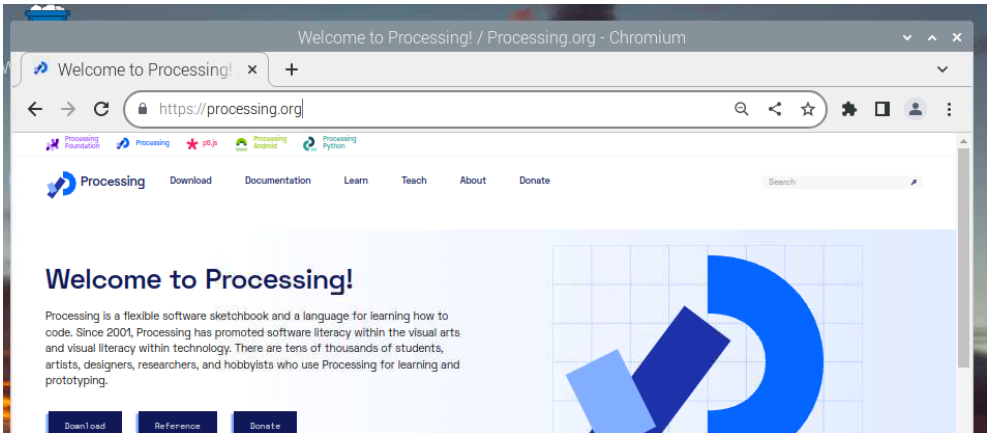
Method 1:
If you have a 32-bit system, enter the following command to install the package.

```
wget https://github.com/processing/processing4/releases/download/processing-1292-4.2/processing-4.2-linux-arm32.tgz
```

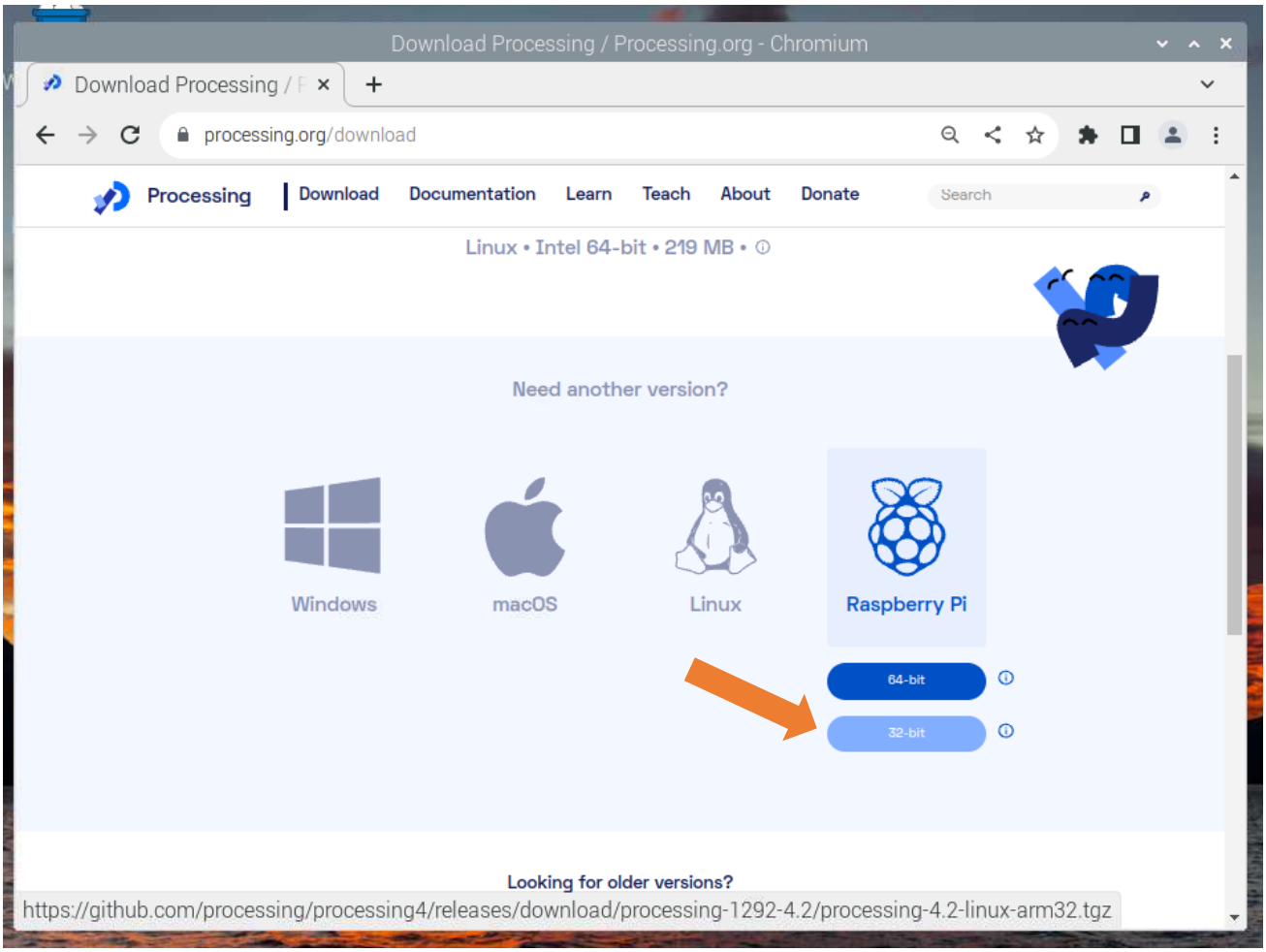
If you have a 64-bit one, enter the following command to install the package.

```
wget https://github.com/processing/processing4/releases/download/processing-1292-4.2/processing-4.2-linux-arm64.tgz
```

Method 2:
You can also download the installation package directly form the Processing official website:
https://processing.org/



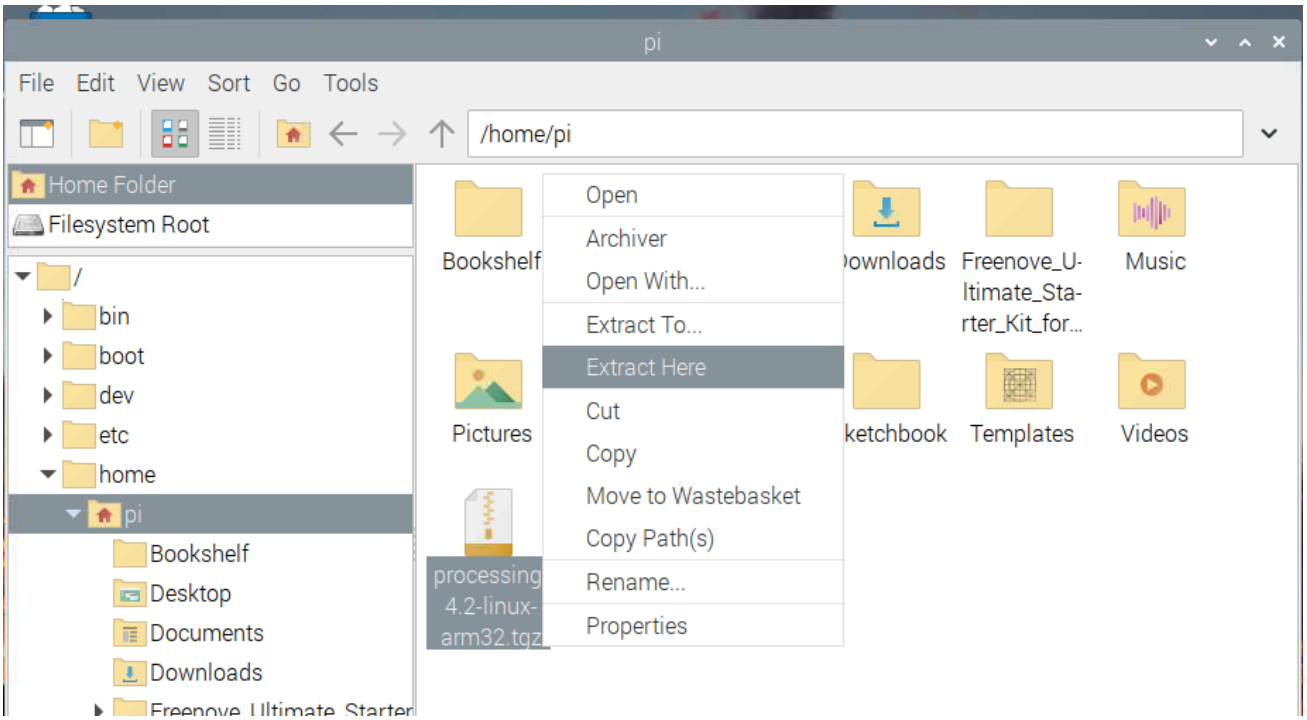
Click“Download”. Choose to download the software installation package corresponding to the current Raspberry Pi system bitness.

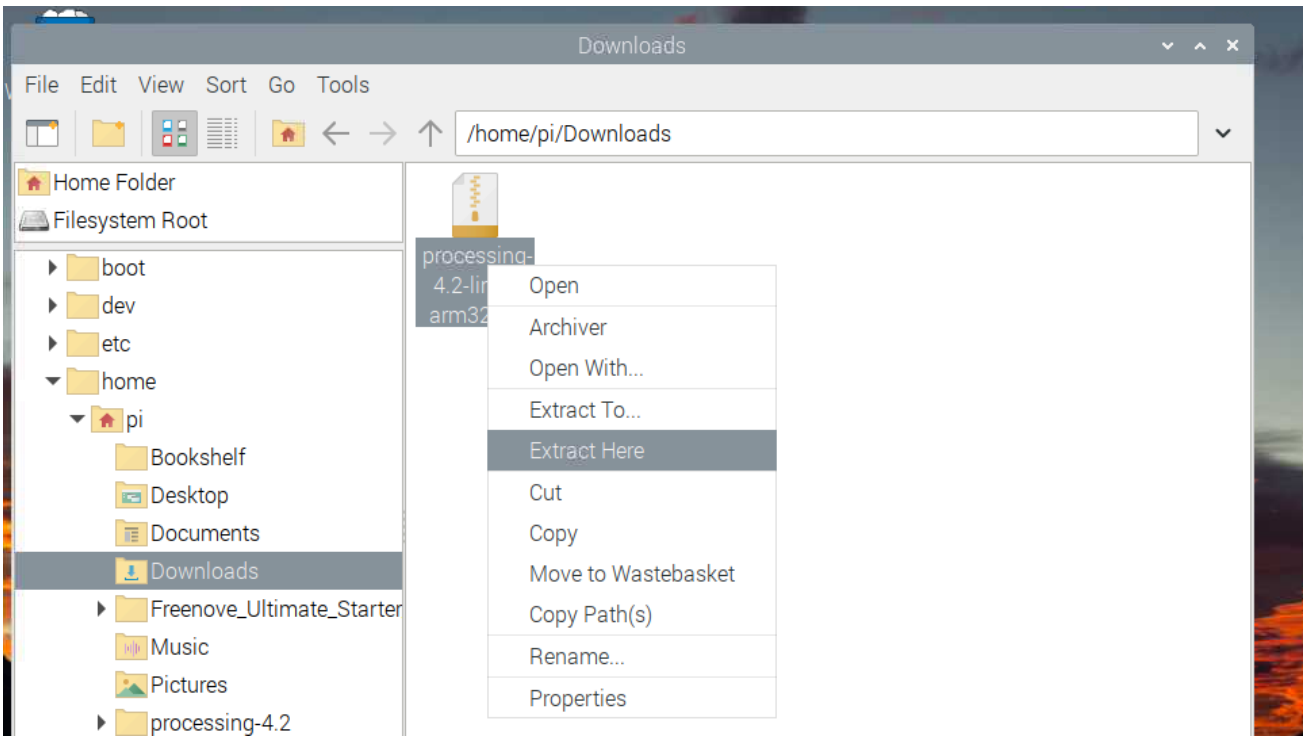


It is recommended to use the first method to download the software package.

Find the directory where the installation package is located and extract it to the current directory.

The default directory of the installation package using the first method is: /home/pi, and with the the second method, it is: /home/pi/Downloads





Take the first method as an example: enter the following command to install processing

1. Run the command to enter the folder.

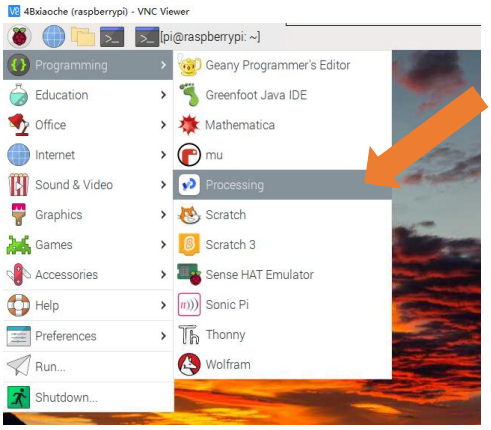
```
cd ~/processing-4.2
```

2. Run the command to install software.

```
sh ./install.sh
```

```
pi@raspberrypi:~ $ cd ~/processing-4.2
pi@raspberrypi:~/processing-4.2 $ sh ./install.sh
Adding desktop shortcut, menu item and file associations for Processing... done!
pi@raspberrypi:~/processing-4.2 $
```

After finishing installation, there will be shortcut in Menu and desktop.



It is worth noting that the Raspberry Pi 4 series is used in this tutorial, which makes the running of Processing smoother. When using other models, there may be a phenomenon of freezing. When the freezing occurs, you cannot complete the experiment. At this time, try to lower the version of Processing, such as the specific version of processing 3.5.3, you can visit the following link:

https://github.com/processing/processing/releases

The installation command for Processing 3.5.3 is as below:

```
wget https://github.com/processing/processing/releases/download/processing-0269-3.5.3/processing-3.5.3-linux-armv6hf.tgz
```

Before installing the old version of Processing, you should uninstall Processing 4.2.

The uninstallation steps are as follows:

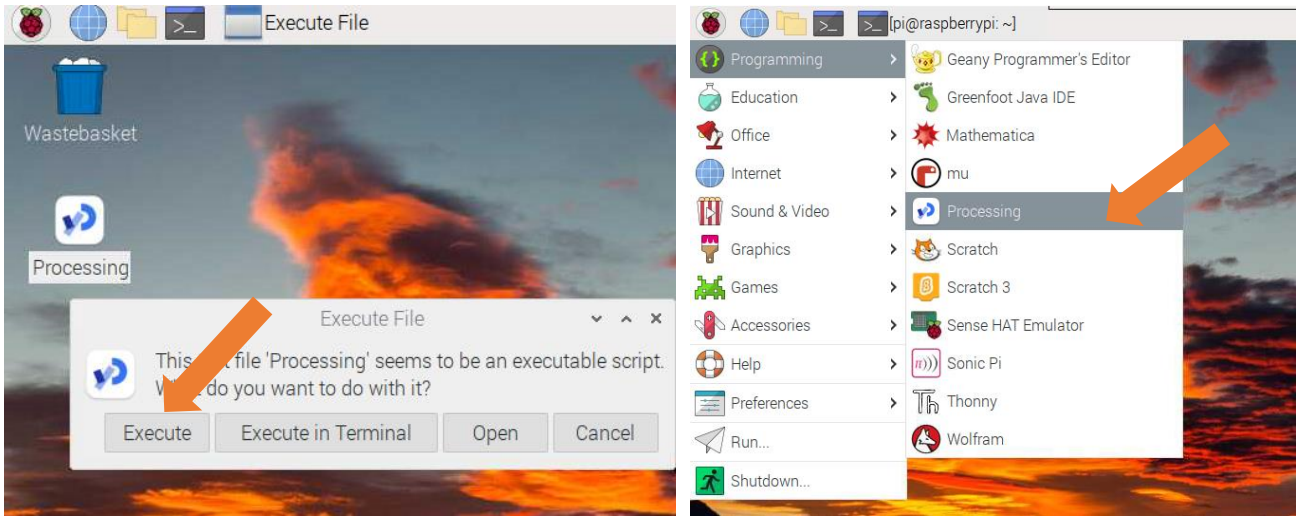
1. Run the command to enter the folder.

```
cd ~/processing-4.2
```

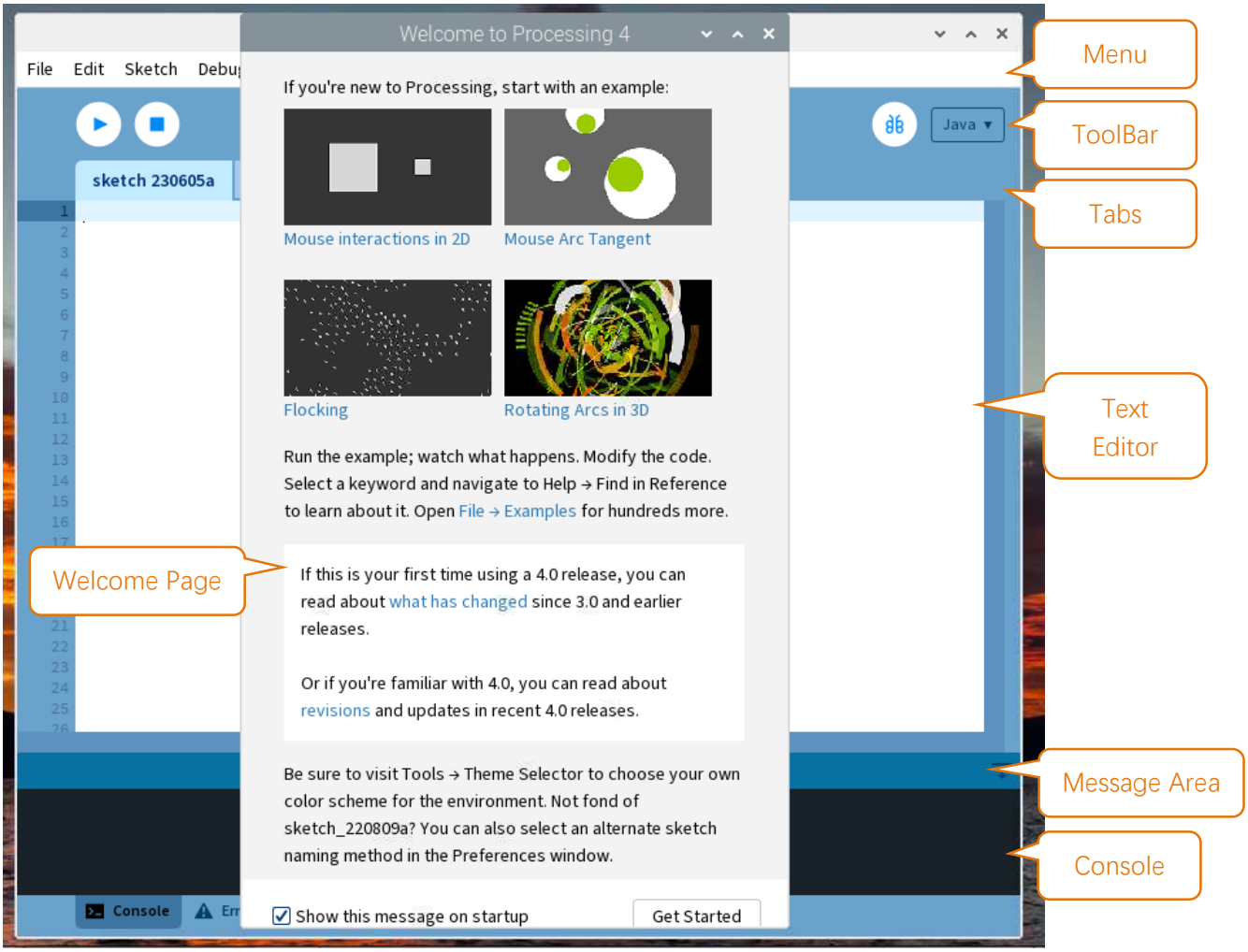
2. Run the command to uninstall software.

```
sh ./uninstall.sh
```

After the installation is complete, you can double-click the software icon on the desktop to enter the "Processing" software, or you can open the software processing in the system's start menu, as shown in the following figure:



Interface of processing software is shown below:



You're now running the Processing Development Environment (or PDE). There's not much to it; the large area is the Text Editor, and there's a row of buttons across the top; this is the toolbar. Below the editor is the

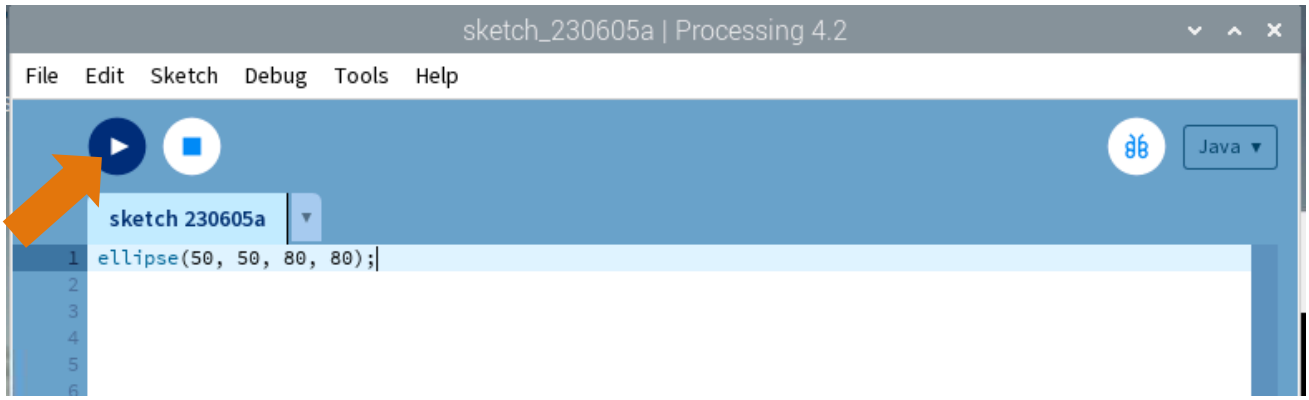
Message Area, and below that is the Console. The Message Area is used for one line messages, and the Console is used for more technical details.

First Use

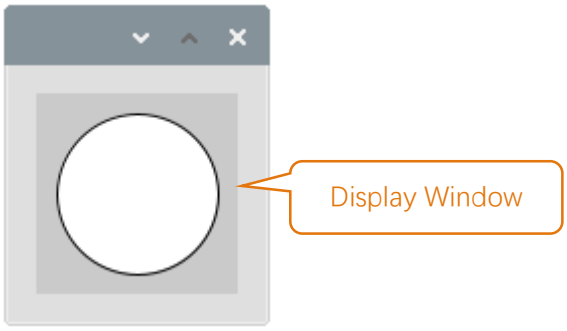
In the editor, type the following:

```
1  ellipse(50, 50, 80, 80);
```

This line of code means "draw an ellipse, with the center 50 pixels over from the left and 50 pixels down from the top, with a width and height of 80 pixels." Click the Run button (the triangle button in the Toolbar).



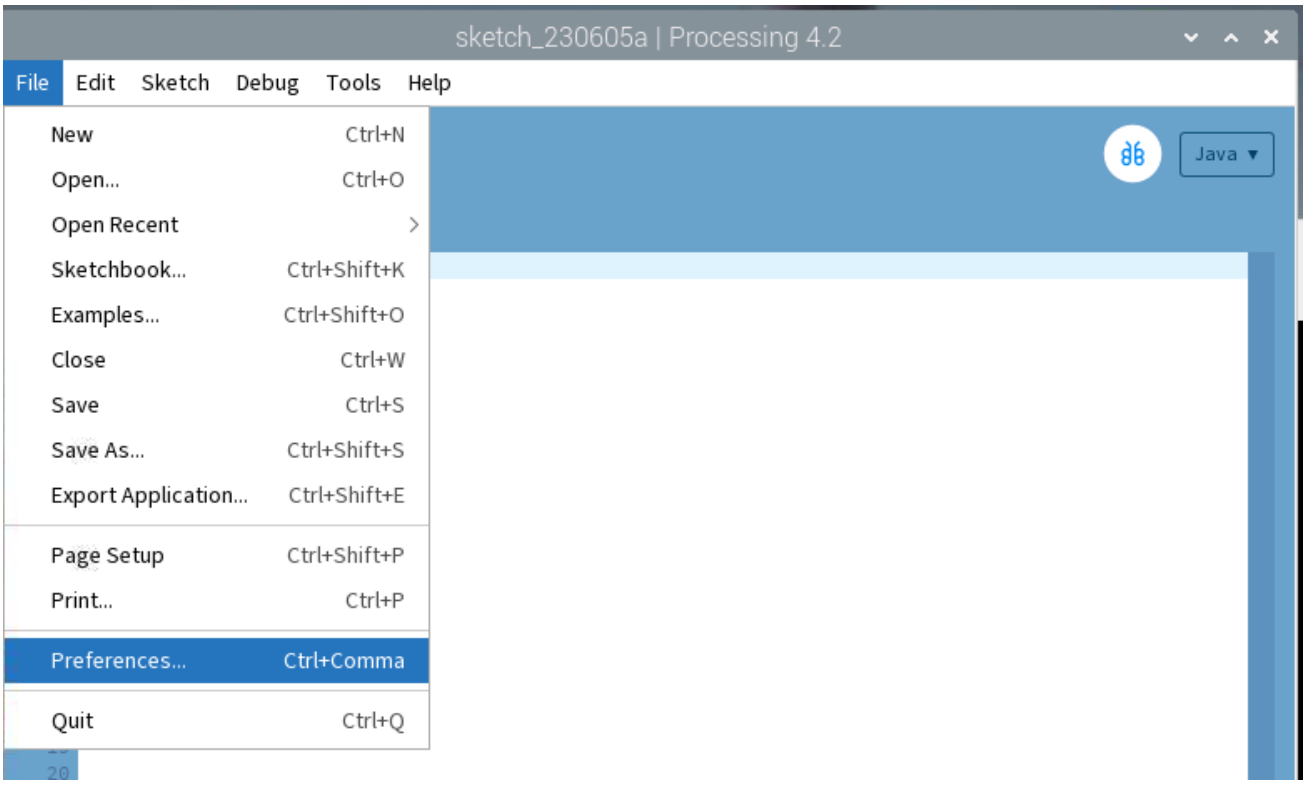
If you've typed everything correctly, you'll see a circle on your screen.



Click on "Stop" (the rectangle button in the Toolbar) or "Close" on Display Window to stop running the program.

If you didn't type it correctly, the Message Area will turn red and report an error. If this happens, make sure that you've copied the example code exactly: the numbers should be contained within parentheses and have commas between each of them, and each line should end with a semicolon.

You can export this sketch to an application to run it directly without opening the Processing.
To export the sketch to the application, you must first save it.



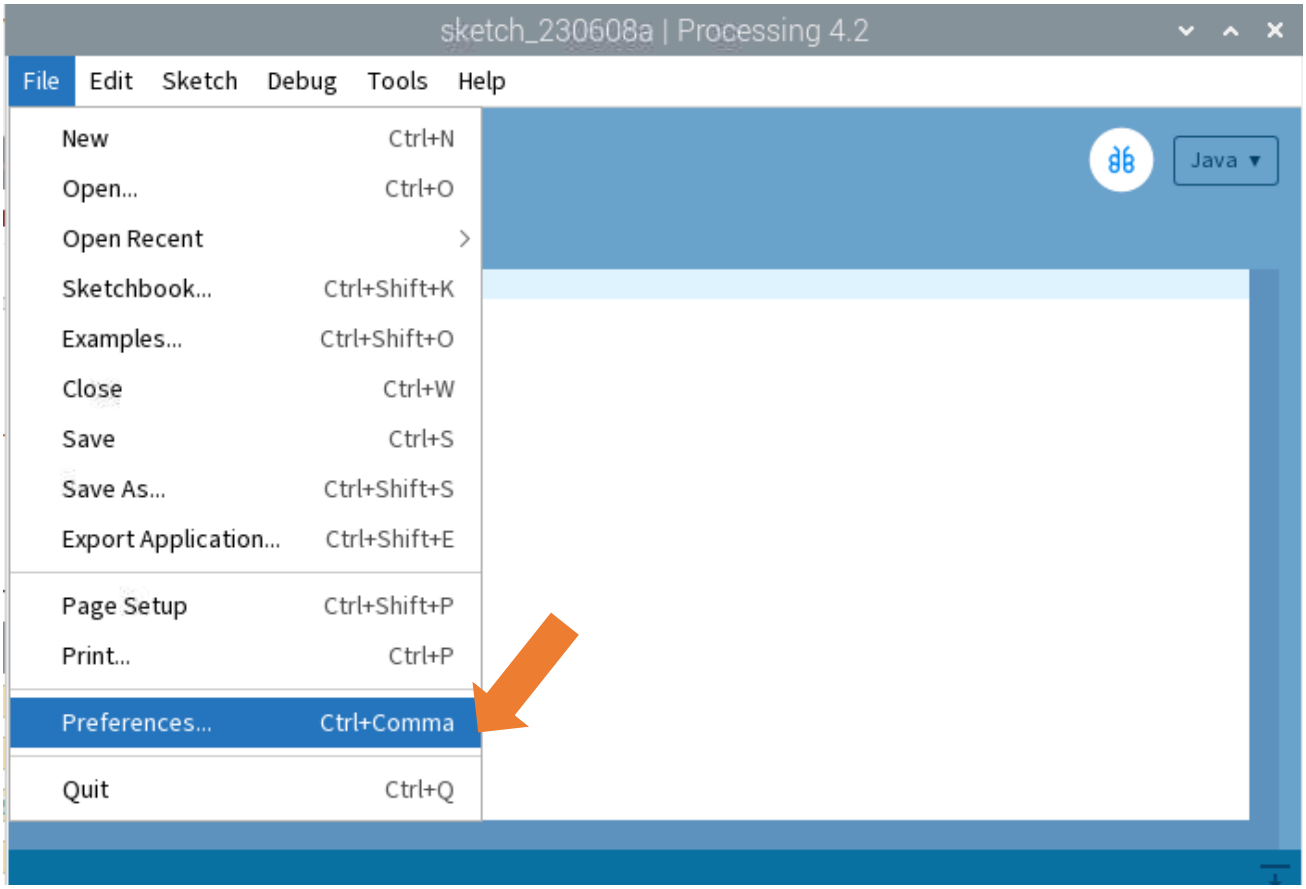
So far, we have completed the first use. I believe you have felt the joy of it.

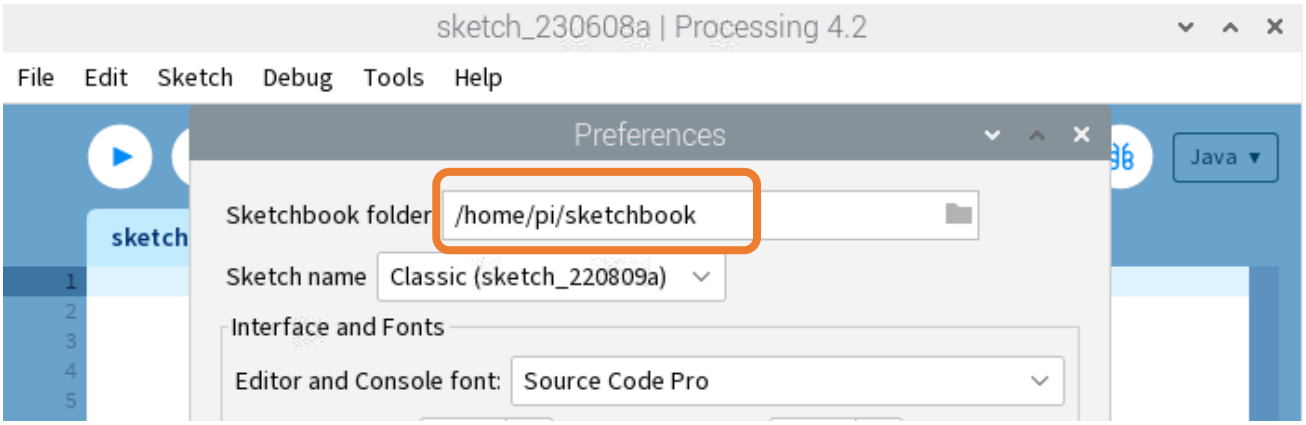
Installing Hardware I/O Library

In this tutorial, the Hardware I/O library needs to be installed in order to perform corresponding experiments. The Hardware I/O library allows access to the Raspberry Pi's hardware peripherals, such as digital inputs and outputs, serial buses, etc., in a manner similar to the Arduino platform. In Processing 4.0 and above, manual installation is required. In previous versions, the Hardware I/O library has been pre-installed.
Therefore, you can also try to use the old version of Processing. For example, processing 3.5.3. You can refer to the linbelow for versions released:
https://github.com/processing/processing/releases

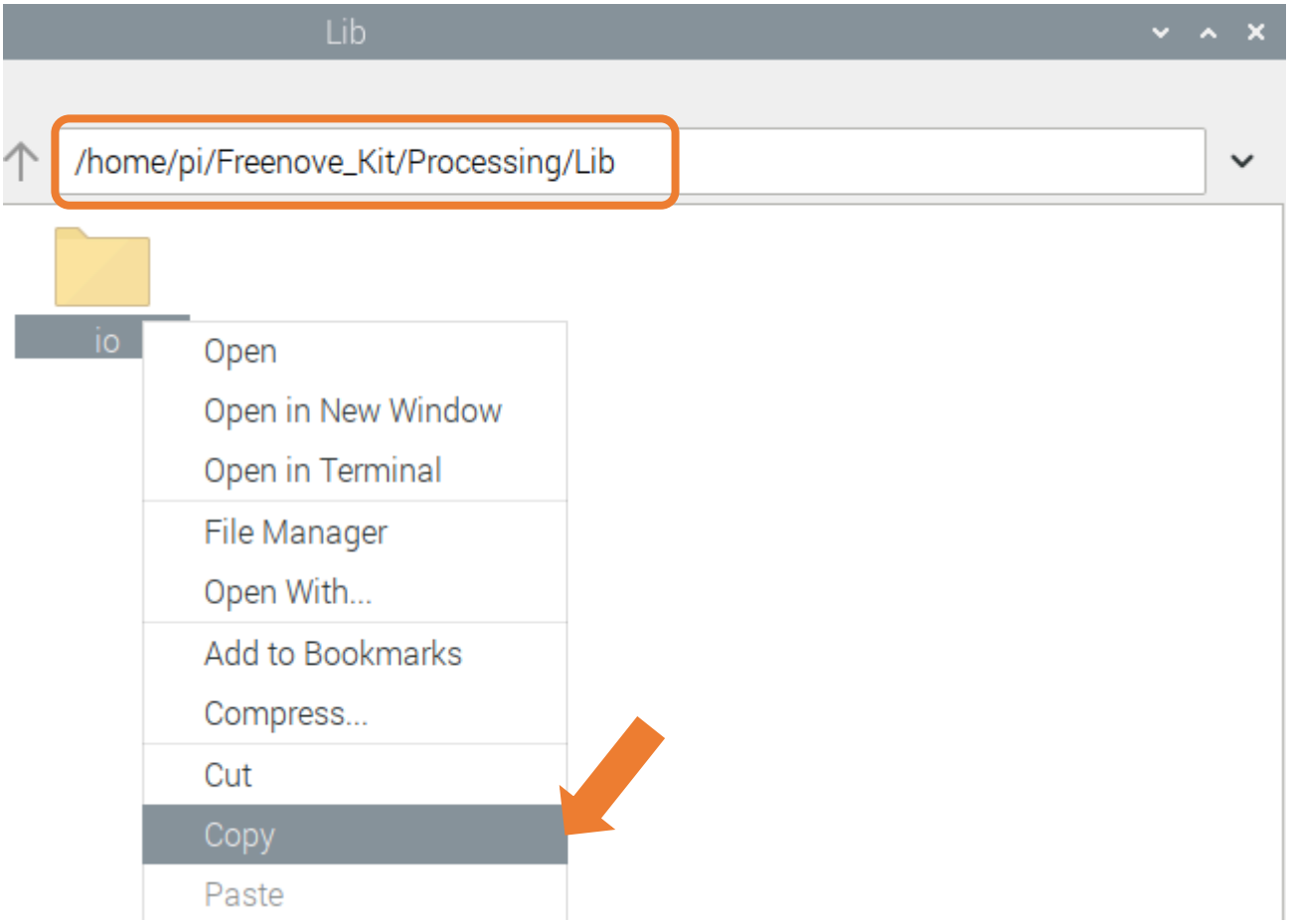
If you are using an old version Processing, you can skip the following steps:
Steps to install Hardware I/O library:

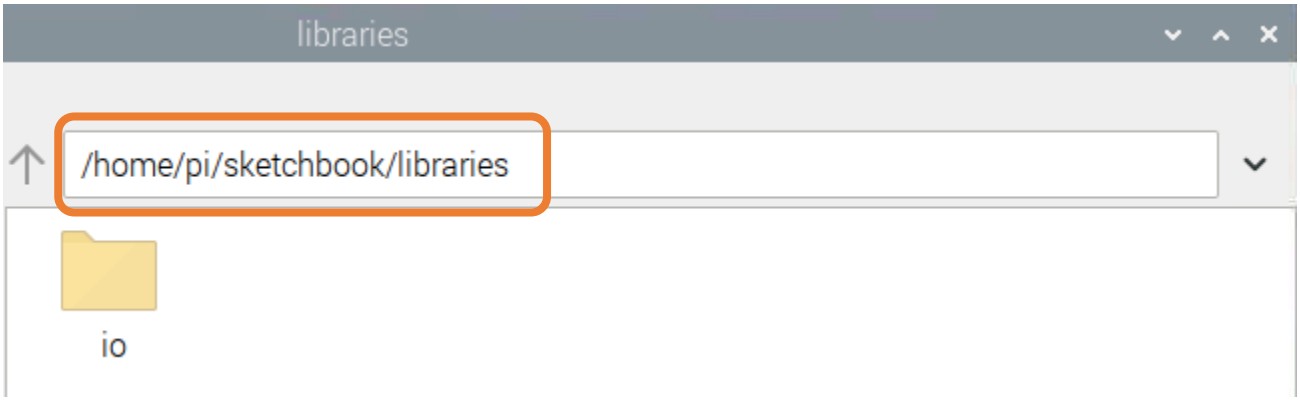
Open Proessing, click File > Preferencesto check the library installation path, which, by default, is /home/pi/sketchbook



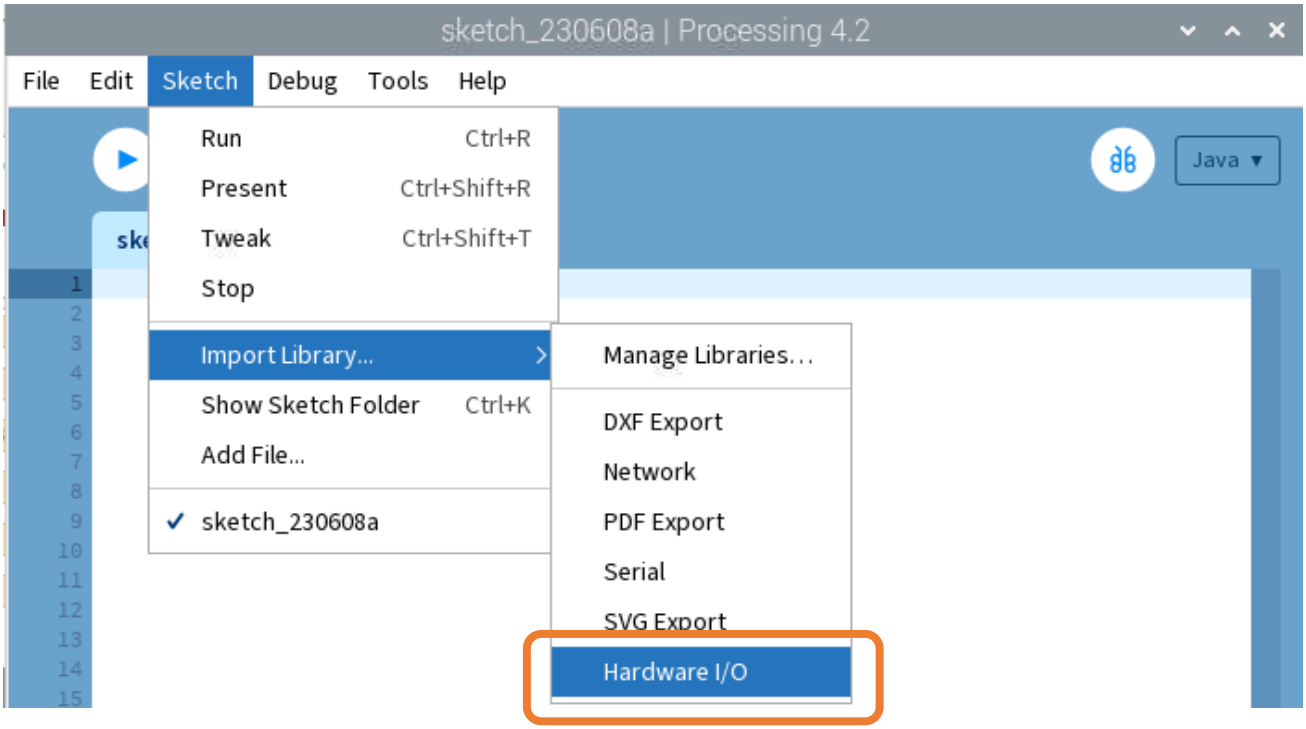


Copy the io folder under /home/pi/Freenove_Kit/Processing/Lib to the Processing library loading directory:
It is worth noting that when opening the file path /home/pi/sketchbook,if there is no folder "libraries", create a folder and name it "libraries".



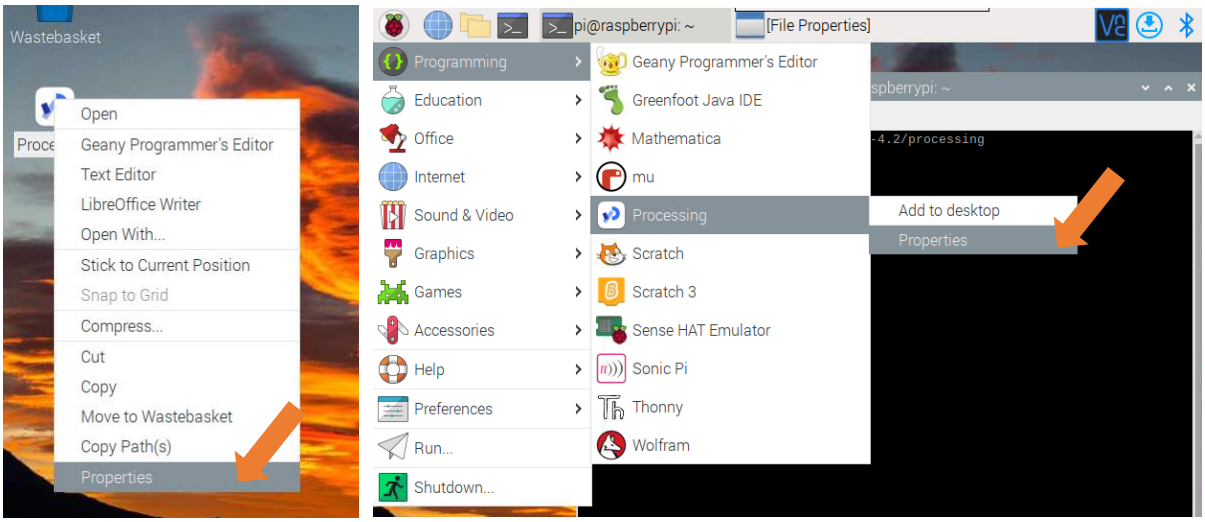


Re-open Processing, clickt Sketch> Import Library, and you can see that the Hardware I/O library has been successfully installed.



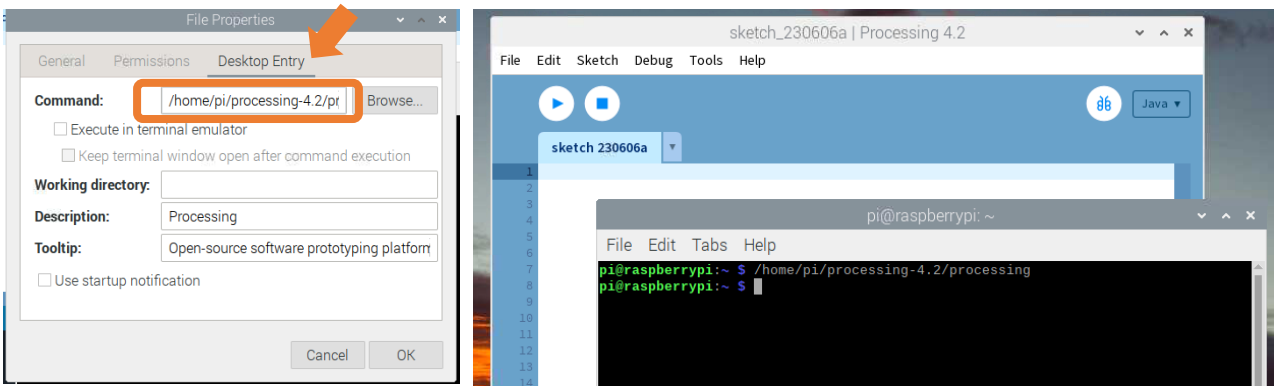
Set Commands to run on the Terminal

Check the current Processing startup command. Find the Processing execution file on the desktop, right-click and select Properties. Or open the software Processing Properties option in the system's start menu, as shown in the figure below:



Select Desktop Entry, the content in Command is the current Processing terminal startup command, enter the following content in the terminal to open Processing. The command is different according to the installation path.

```
/home/pi/processing-4.2/processing
```



Define an alias for the command

For the convenience of use, we set an alias for the Processing terminal startup command.

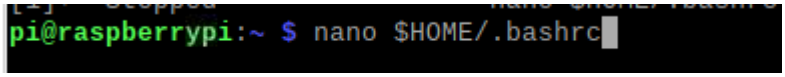
The specific steps are as follows:

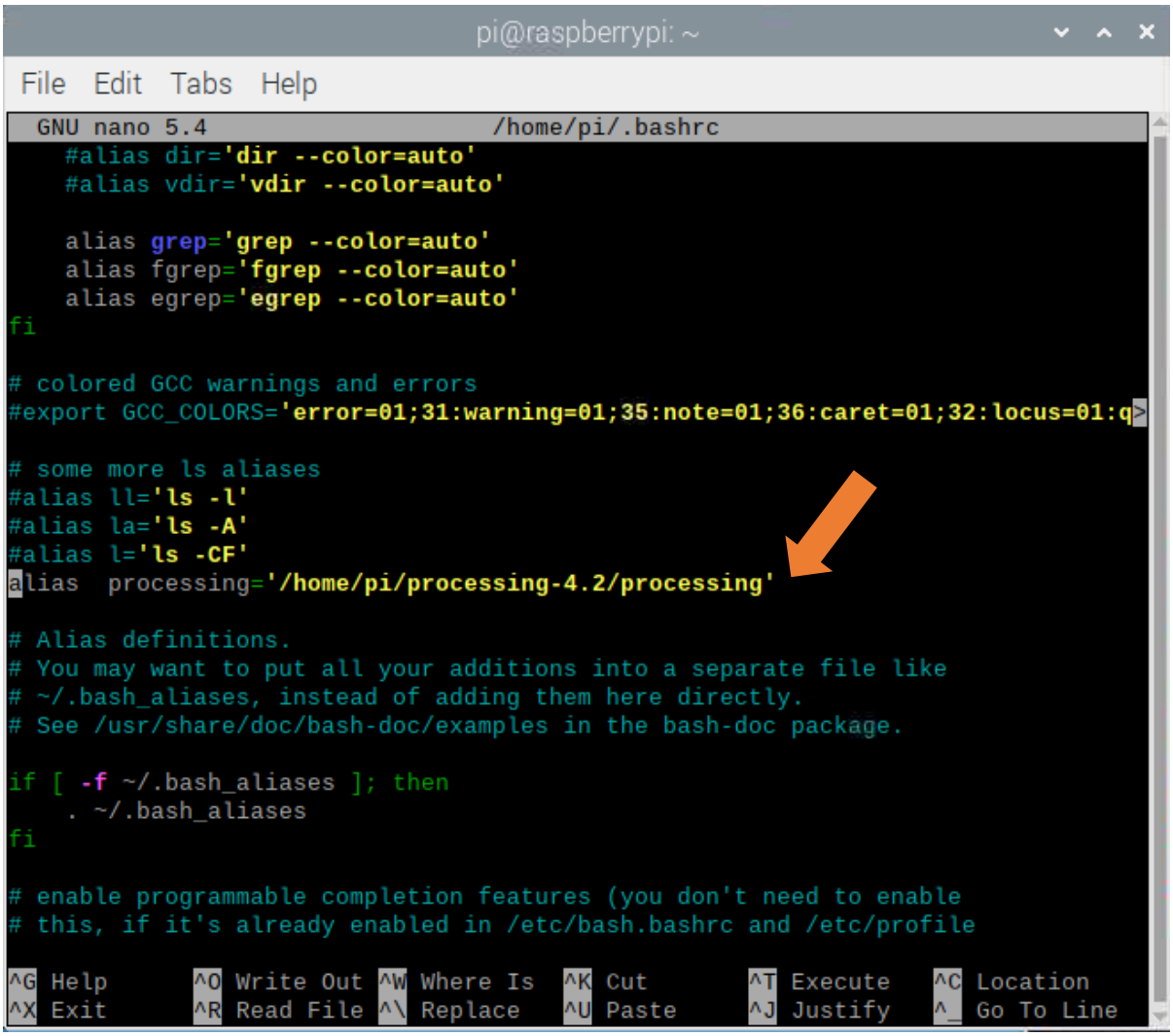
1. Enter the following command to edit the $HOME/.bashrc file.

```
nano $HOME/.bashrc
```

2. Add processing command alias.

```
alias  processing='/home/pi/processing-4.2/processing'
```

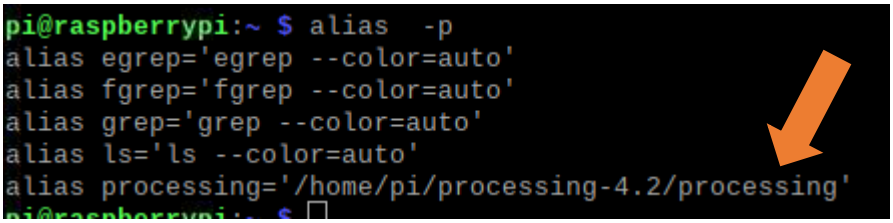




Press "CTRL"+"O" and then "Enter" to save the modified content. Then press "CTRL"+"X" to exit editing.

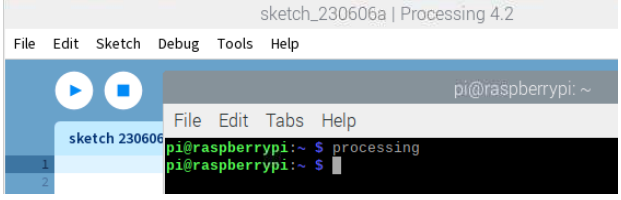
Close all current terminal pages, open a new terminal page again, enter the following command, open the command list of defined alias to check whether the addition is successful:

```
alias -p
```



Open the terminal and enter the following to test the terminal command

```
processing
```



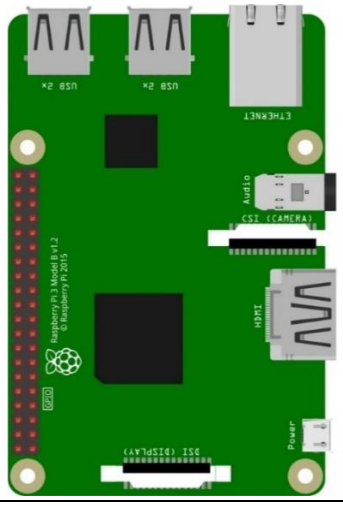
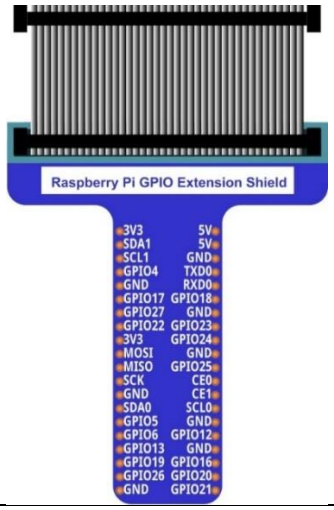
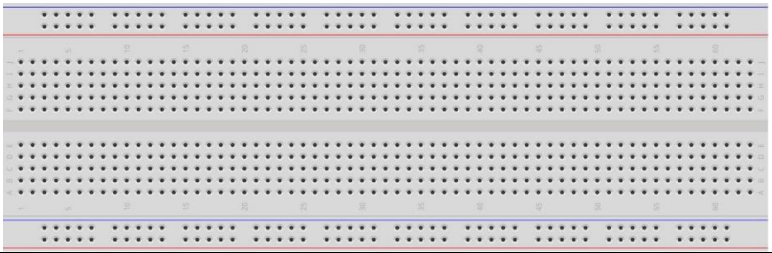
Chapter 1 LED

We will still start from Blink LED in this chapter, and also learn the usage of some commonly used functions of Processing Software.

Project 1.1 Blink

In this project, we will make a Blink LED and let Display window of Processing Blink at the same time.

Component List

<table>
<tr><td colspan="2">Raspberry Pi x1
  </td><td colspan="2">GPIO Extension Board & Wire x1
 
</td></tr>
<tr><td colspan="4">Breadboard x1
  </td></tr>
<tr><td>LED x1
  </td><td>Resistor 220Ω x1
  </td><td colspan="2">Jumper Wire M/M x2
  </td></tr>
</table>

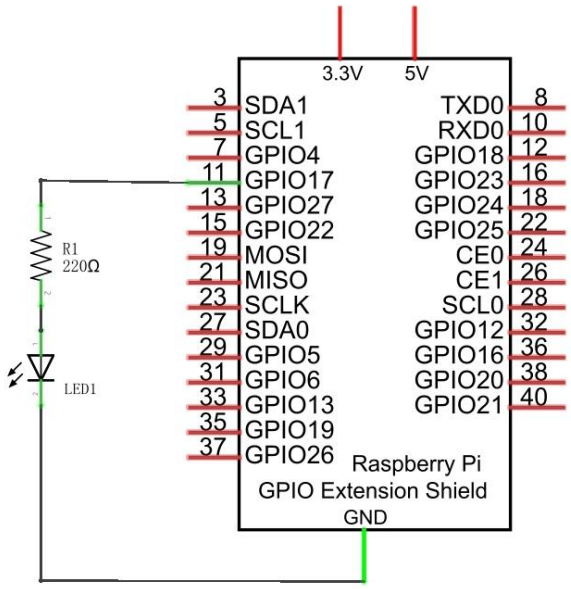
In the components list, Raspberry Pi, GPIO Extension Shield and Breadboard are necessary for each experiment. They will be listed only in text form.

Circuit

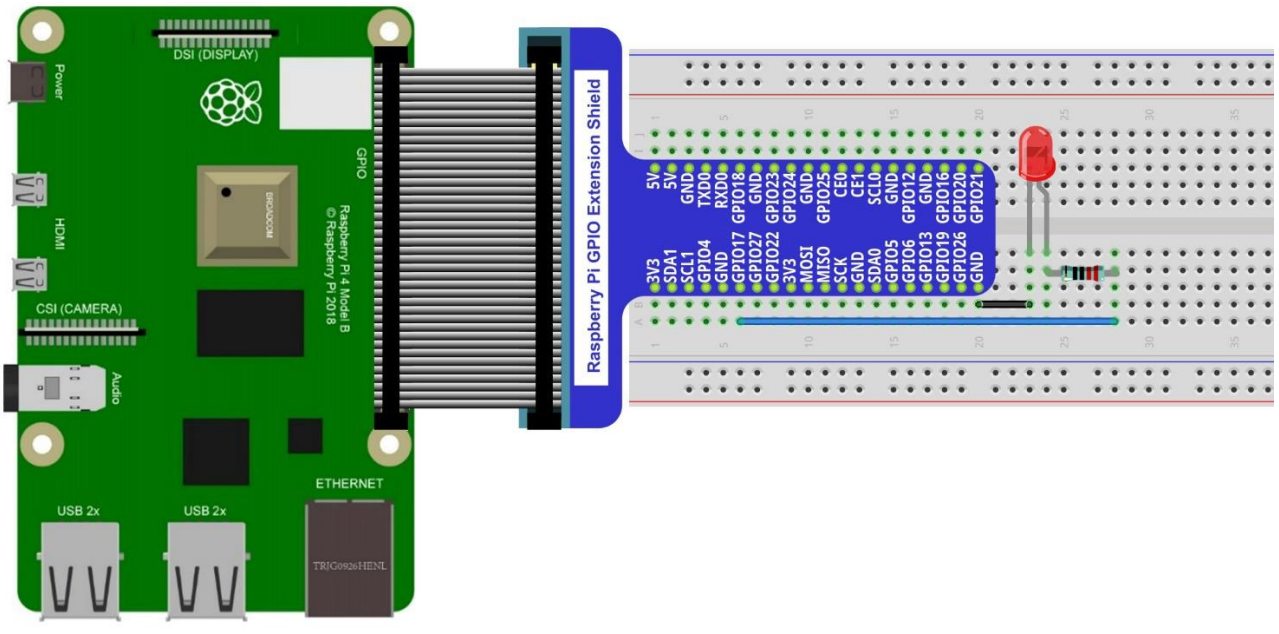
Build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the RPi to GPIO Extension Shield. CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!

WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your RPi!

Schematic diagram



Hardware connection



Because the numbering of the GPIO Extension Shield is the same as that of the RPi GPIO, future hardware connection diagrams will only show that part of breadboard and GPIO Extension Shield.

Sketch

Sketch 1.1.1 Blink

Because the resource folder name is too long, for convenience, the folder will be named as "Freenove_Kit". If you have already renamed it, skip this command. Assume the absolute path is "/ home / pi" or "~ /", execute the following command in the user directory.

```
mv Freenove_Basic_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

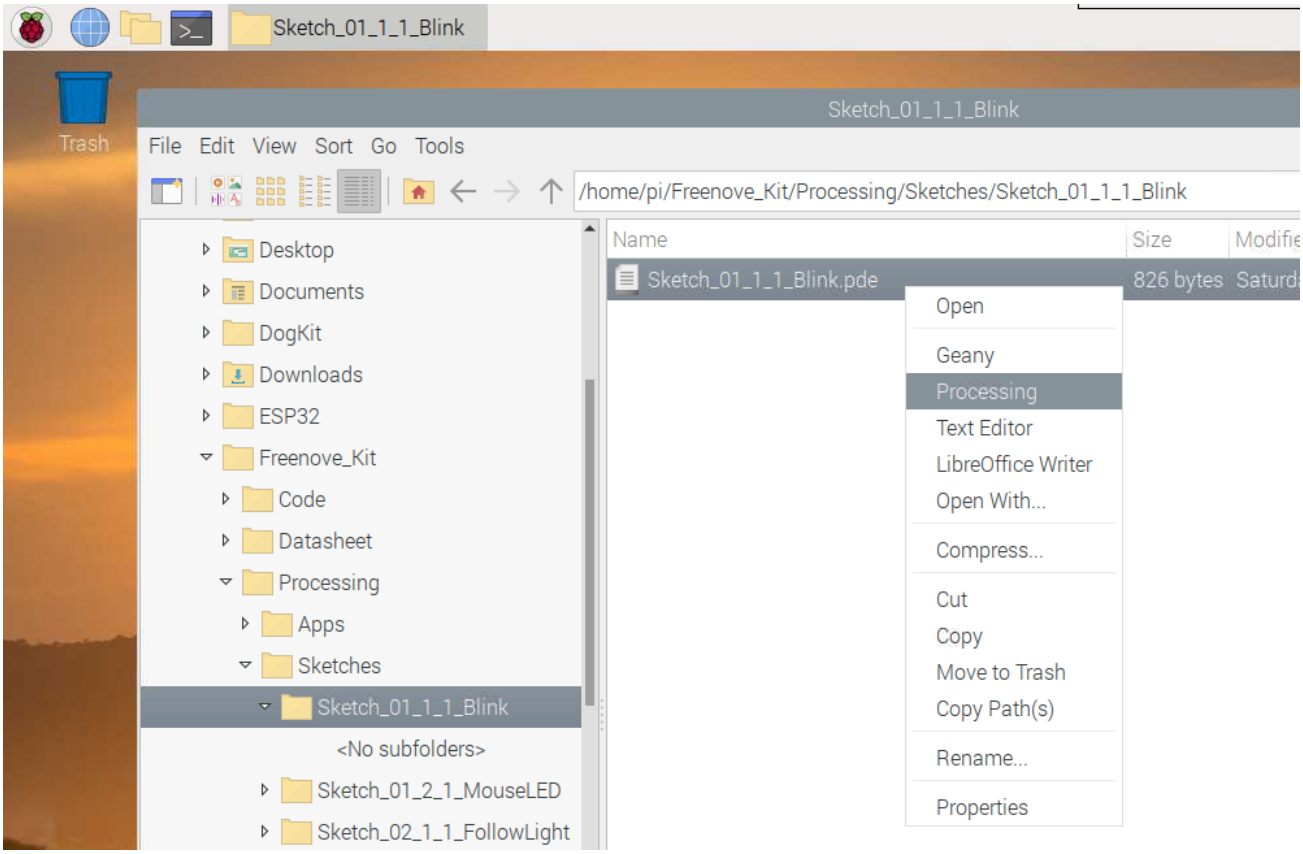
First, observe the result after running the sketch, and then learn about the code in detail.

1. Use Processing to open the file Sketch_01_1_1_Blink. (The following is only one line of command. There is a Space after Processing.)

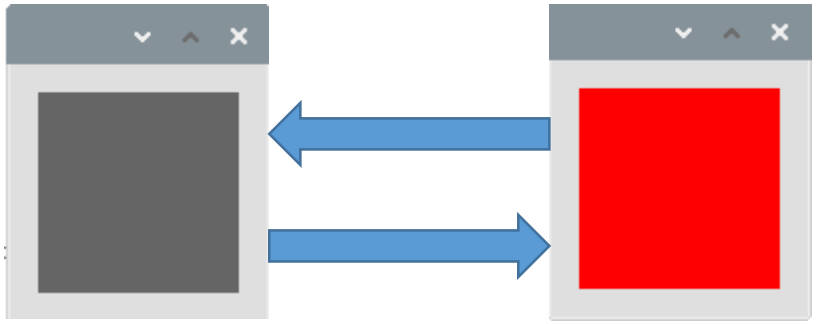
```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_01_1_1_Blink/Sketch_01_1_1_Blink.pde
```

Before using this command, please set the command, otherwise Processing cannot be opened.
Click on "RUN" to run the code.

You can also open it as follows.
Click Raspberry Pi file manager. Find the file under path:
/home/pi/Freenove_Kit/Processing/Sketches/Sketch_01_1_1_Blink
And then right-click it and select Processing.



After the program is executed, LED will start Blinking and the background of Display window will change with the change of LED state.



The following is program code:

```
import processing.io.*;

int ledPin = 17;    //define ledPin
boolean ledState = false;    //define ledState

void setup() {
  size(100, 100);
  frameRate(1);        //set frame rate
  GPIO.pinMode(ledPin, GPIO.OUTPUT);    //set the ledPin to output mode
}

void draw() {
  ledState = !ledState;
  if (ledState) {
    GPIO.digitalWrite(ledPin, GPIO.HIGH);    //led on
    background(255, 0, 0); //set the fill color of led on
  } else {
    GPIO.digitalWrite(ledPin, GPIO.LOW);    //led off
    background(102); //set the fill color of led off
  }
}
```

Processing code usually have two functions: setup() and draw(), where the function setup() is only executed once while the function draw() will be executed repeatedly. In the function setup(), size(100, 100) specifies the size of the Display Window to 100x100pixl. FrameRate(1) specifies the refresh rate of Display Window to once per second, which means the draw() function will be executed once per second. GPIO.pinMode (ledPin, GPIO.OUTPUT) is used to set ledPin to output mode.

```
void setup() {
  size(100, 100);
  frameRate(1);        //set frame rate
  GPIO.pinMode(ledPin, GPIO.OUTPUT);    //set the ledPin to output mode
}
```

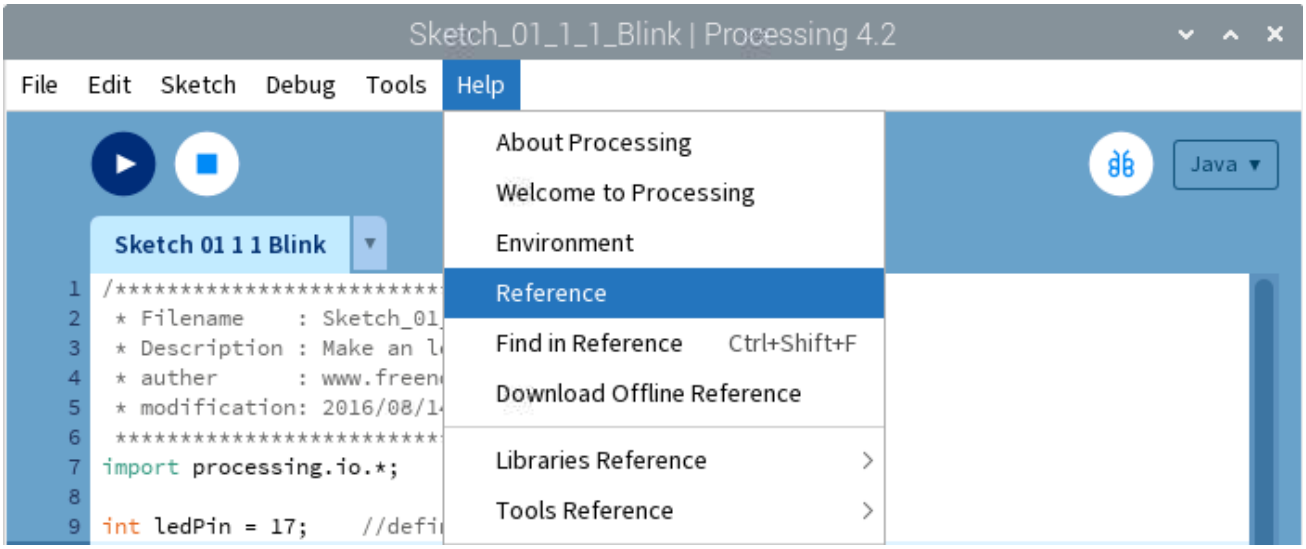
In draw() function, each execution will invert the variable "ledState". When “ledState” is true, LED is turned ON, and the background color of display window is set to red. And when the “ledState” is false, the LED is turned OFF and the background color of display window is set to gray. Since the function draw() is executed once per second, the background color of Display Window and the state of LED will also change once per second. This process will repeat in an endless loop to achieve the effect of blinking.

```
void draw() {
  ledState = !ledState;
  if (ledState) {
    GPIO.digitalWrite(ledPin, GPIO.HIGH);    //led on
    background(255, 0, 0); //set the fill color of led on
  } else {
    GPIO.digitalWrite(ledPin, GPIO.LOW);    //led off
    background(102); //set the fill color of led off
  }
}
```

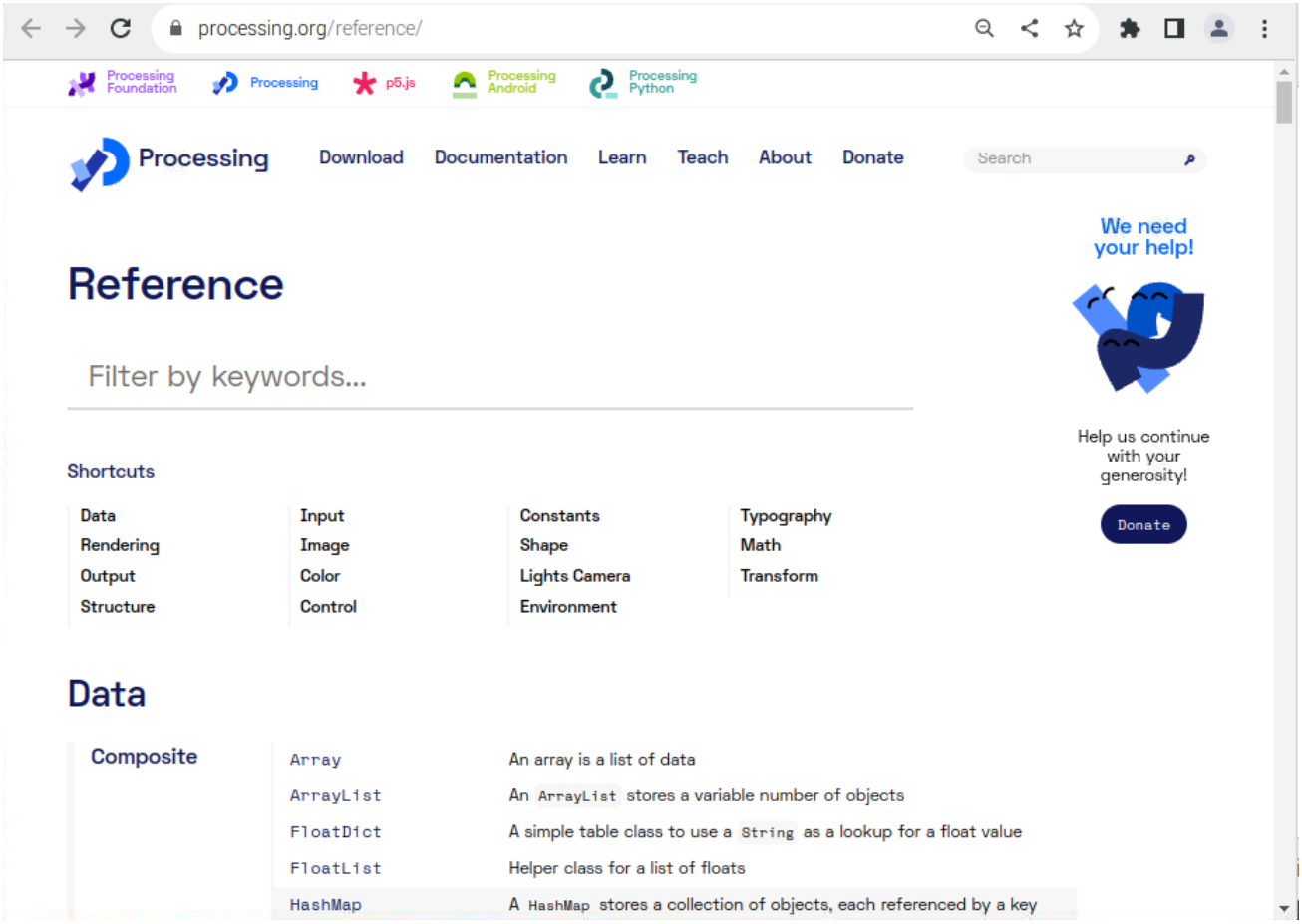
The following is brief descriptions of some functions:

setup()
The setup() function is run once when the program starts.
draw()
It is called directly after the setup() function. The draw() function continuously executes the lines of code within its block until the program stops or noLoop() is called. draw() is called automatically and should never be called explicitly.
size()
Defines width and height of the display window in pixels.
framerate()
Specifies the number of frames to be displayed every second.
background()
Set the color of the background of the display window.
GPIO.pinMode()
Configures a pin to act either as input or output.
GPIO.digitalWrite()
Sets an output pin to be either high or low.

All functions used in this code can be found in the Reference of Processing Software, in which built-in functions are described in details, and there are some sample programs. It is recommended that beginners learn more about usage and function of those functions. The localization of Reference can be opened with the following steps: click the menu bar "Help"→"Reference".



Then the following page will be displayed in the web browser:



Or you can directly access to the official website for reference:http://processing.org/reference/

Project 1.2 MouseLED

In this project, we will use the mouse to control the state of LED.
The components and circuits of this project are the same as the previous section.

Sketch

Sketch 1.2.1 MouseLED

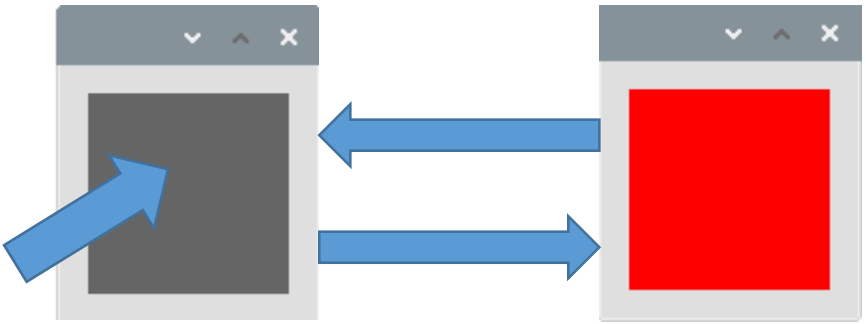
First, observe the result after running the sketch, and then learn the code in detail.

1. Use Processing to open the file Sketch_01_2_1_MouseLED.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_01_2_1_MouseLED/Sketch_01_2_1_MouseLED.pde
```

2. Click on "RUN" to run the code.

After the program is executed, the LED is in OFF-state, and background color of Display window is gray. Click the grey area of the Display Window with the mouse, LED is turned ON and Display window background color becomes red. Click on the Display Window again, the LED is turned OFF and the background color becomes gray, as shown below.



The following is program code:

```
1  import processing.io.*;
2
3  int ledPin = 17;
4  boolean ledState = false;
5  void setup() {
6    size(100, 100);
7    GPIO.pinMode(ledPin, GPIO.OUTPUT);
8    background(102);
9  }
10
11 void draw() {
12   if (ledState) {
13     GPIO.digitalWrite(ledPin, GPIO.HIGH);
14     background(255,0,0);
15   } else {
```

```
16      GPIO.digitalWrite(ledPin, GPIO.LOW);
17      background(102);
18    }
19  }
20
21  void mouseClicked() { //if the mouse Clicked
22    ledState = !ledState;  //Change the led State
23  }
```

The function mouseClicked() in this code is used to capture the mouse click events. Once the mouse is clicked, the function will be executed. We can change the state of the variable "ledState" in this function to realize controlling LED by clicking on the mouse.

```
void mouseClicked() { //if the mouse Clicked
  ledState = !ledState;  //Change the led State
}
```

Chapter 2 LED Bar Graph

We have learned how to control an LED to blink. Next we will learn how to control a number of LEDs.

Project 2.1 FollowLight

In this project, we will use the mouse to control the LED Bar Graph

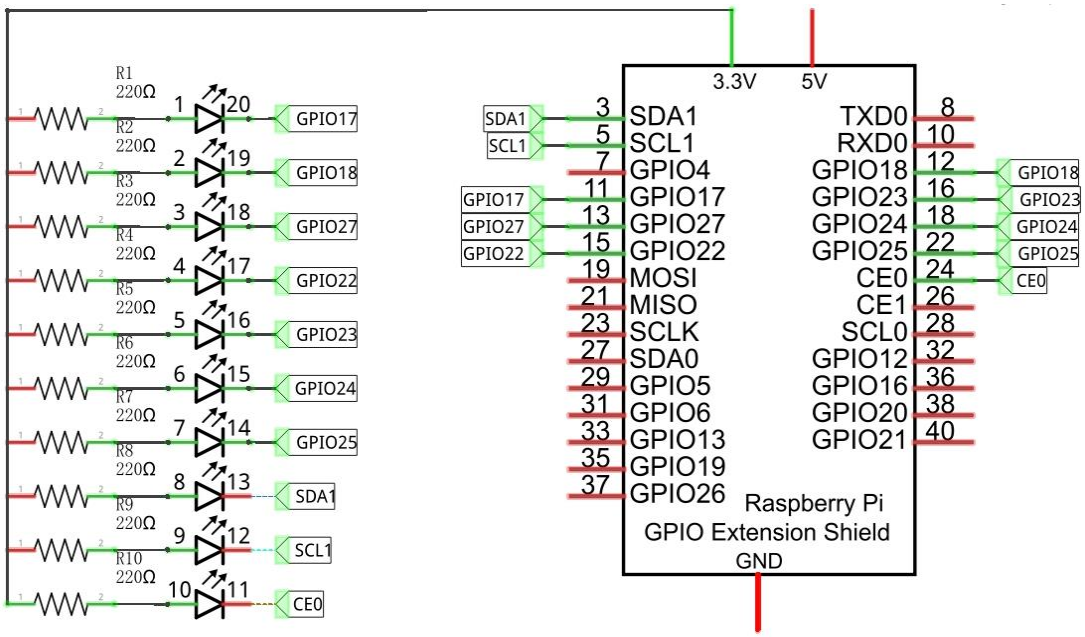
Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 Breadboard x1	LED bar graph x1 	Resistor 220Ω x10 
Jumper M/M x11 		

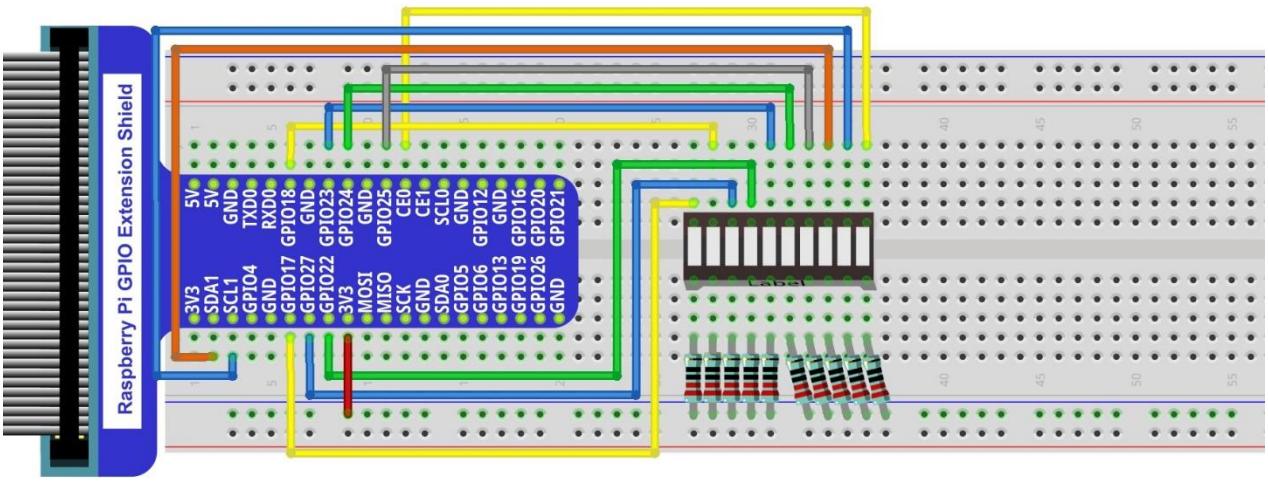
Circuit

A reference system of labels is used in the circuit diagram below, and the pins with the same network label are connected together.

Schematic diagram



Hardware connection



In this circuit, the cathodes of LEDs are connected to the GPIO, which is different from the previous circuit. Therefore, the LEDs turn ON when the GPIO outputs low level in the program.

Sketch

Sketch 2.1.1 FollowLight

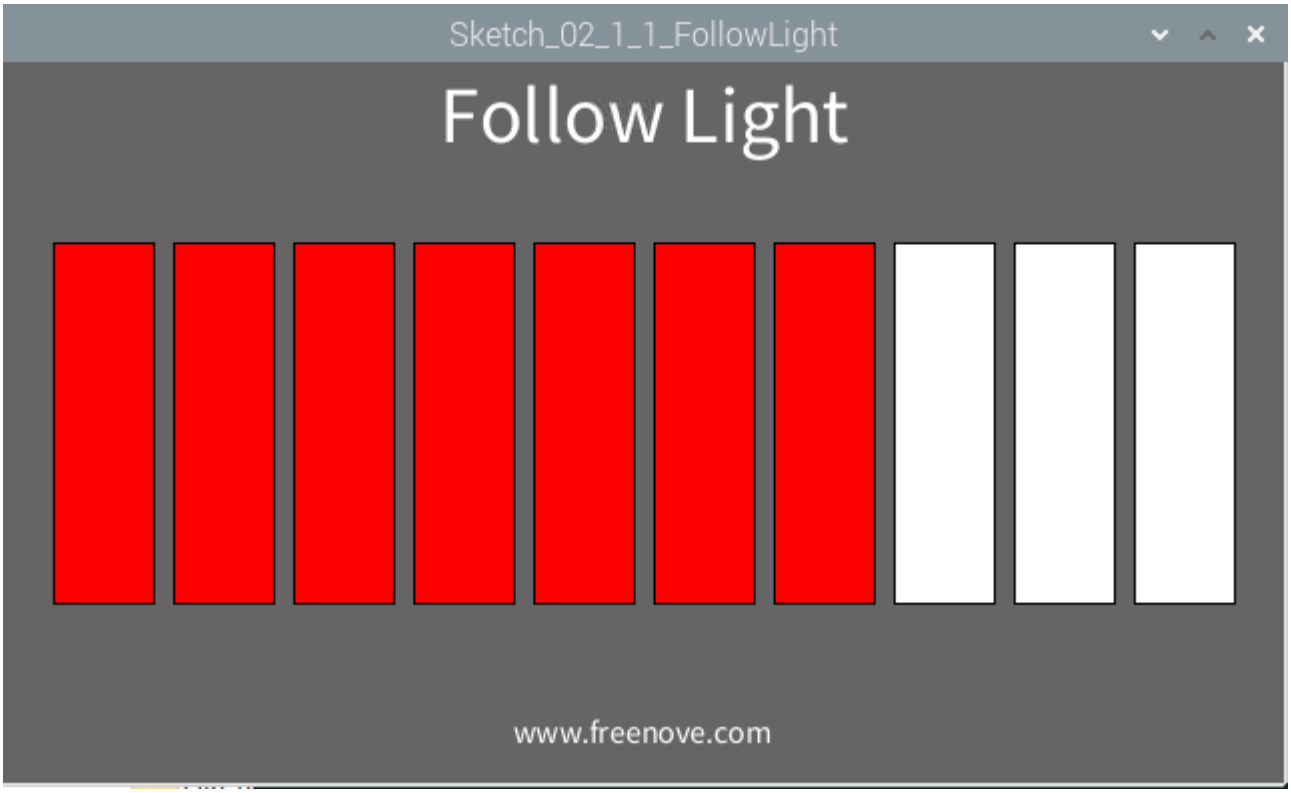
First, observe the result after running the sketch, and then learn about the code in detail.

1. Use Processing to open the file Sketch_02_1_1_FollowLight.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_02_1_1_FollowLight/Sketch_02_1_1_FollowLight.pde
```

2. Click on "RUN" to run the code.

After the program is executed, slide the mouse in the Display Window, then the state of LED Bar Graph will be changed, as shown below.



The following is program code:

```
1  import processing.io.*;
2
3  int leds[]={17, 18, 27, 22, 23, 24, 25, 2, 3, 8}; //define ledPins
4
5  void setup() {
6    size(640, 360);  //display window size
7    for (int i=0; i<10; i++) {  //set led Pins to output mode
8      GPIO.pinMode(leds[i], GPIO.OUTPUT);
9    }
10   background(102);
```

```
11    textAlign(CENTER);    //set the text centered
12    textSize(40);         //set text size
13    text("Follow Light", width / 2, 40);    //title
14    textSize(16);
15    text("www.freenove.com", width / 2, height - 20);    //site
16  }
17
18  void draw() {
19    for (int i=0; i<10; i++) {    //draw 10 rectangular box
20      if (mouseX>(25+60*i)) {    //if the mouse cursor on the right of rectangular box
21        fill(255, 0, 0);         //fill the rectangular box in red color
22        GPIO.digitalWrite(leds[i], GPIO.LOW);  //turn on the corresponding led
23      } else {
24        fill(255, 255, 255);  //else fill the rectangular box in white color
25        GPIO.digitalWrite(leds[i], GPIO.HIGH);  //and turn off the led
26      }
27      rect(25+60*i, 90, 50, 180);    //draw a rectangular box
28    }
29  }
```

In the function draw(), we draw 10 rectangles to represent 10 LEDs of LED Bar Graph. We make rectangles on the left of mouse filled with red, corresponding LEDs turned ON. And make We make rectangles on the right of mouse filled with red, corresponding LEDs turned OFF. In this way, when slide the mouse to right, the more LEDs on the left of mouse will be turned ON. When to the left, the reverse is the case.

```
void draw() {
  for (int i=0; i<10; i++) {    //draw 10 rectangular box
    if (mouseX>(25+60*i)) {    //if the mouse cursor on the right of rectangular box
      fill (255, 0, 0);         //fill the rectangular box in red color
      GPIO.digitalWrite(leds[i], GPIO.LOW);  //turn on the corresponding led
    } else {
      fill(255, 255, 255);  //else fill the rectangular box in white color
      GPIO.digitalWrite(leds[i], GPIO.HIGH);  //and turn off the led
    }
    rect(25+60*i, 90, 50, 180);    //draw a rectangular box
  }
}
```

Chapter 3 PWM

In this chapter, we will learn how to use PWM.

Project 3.1 BreathingLED

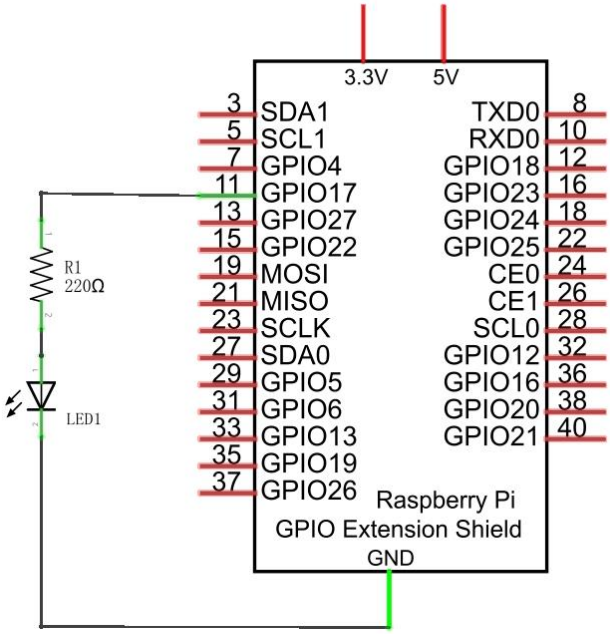
In this project, we will make a breathing LED, which means that an LED that is OFF will then turn ON gradually and then gradually turn OFF like "breathing". and the Display Window will show a breathing LED pattern and a progress bar at the same time.

Component List

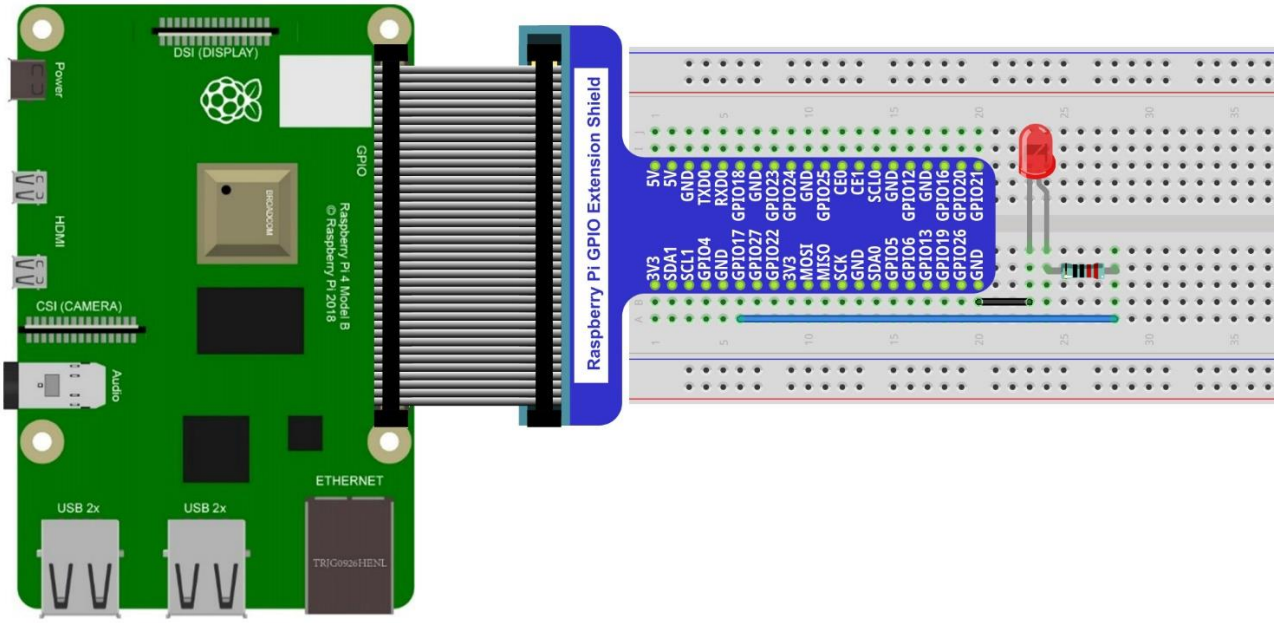
Raspberry Pi x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1 	Resistor 220Ω x1 
Jumper M/M x2 		

Circuit

Schematic diagram



Hardware connection



Sketch

Sketch 3.1.1 BreathingLED

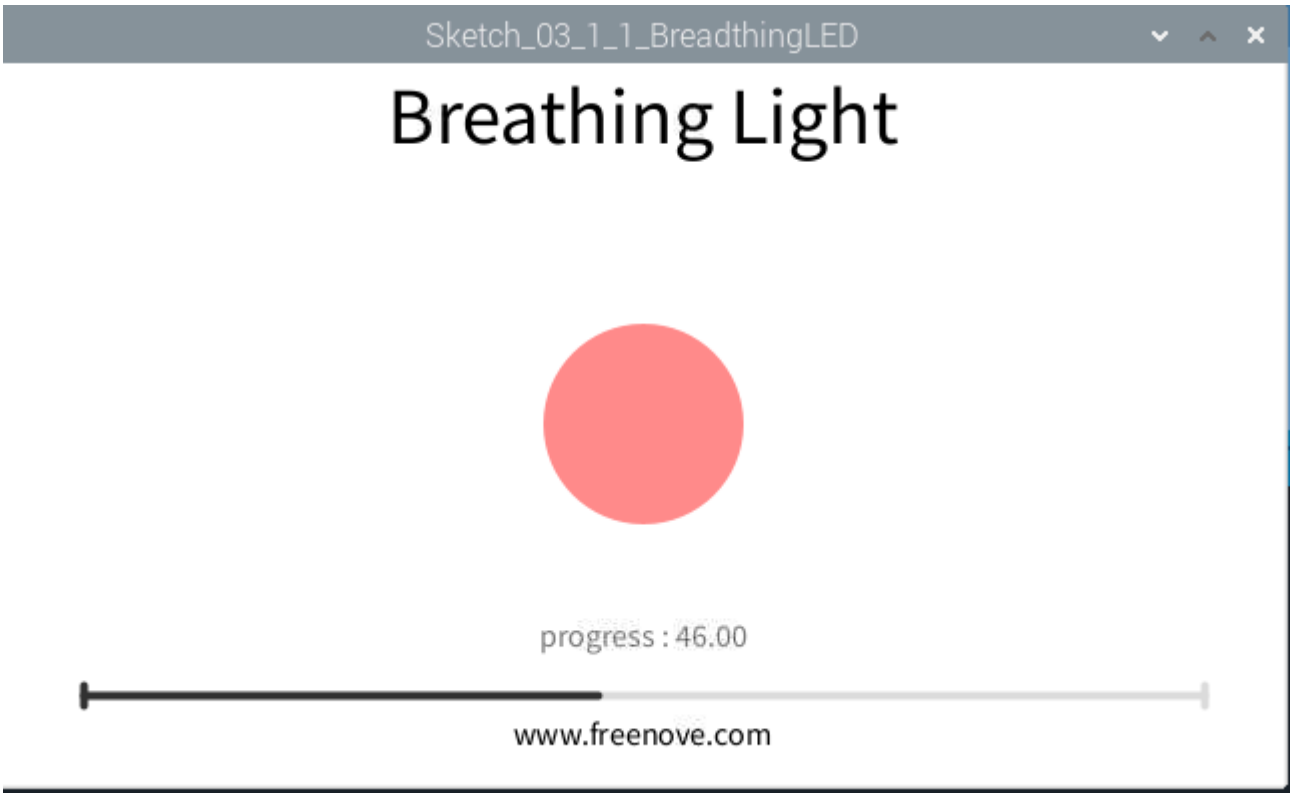
First, observe the result after running the sketch, and then learn about the code in detail.

1. Use Processing to open the file Sketch_03_1_1_BreathingLED.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_03_1_1_BreadthingLED/Sketch_03_1_1_BreadthingLED.pde
```

2. Click on "RUN" to run the code.

After the program is executed, the LED in the circuit will be brightened gradually, and the color of LED pattern in Display Window will deepen gradually at the same time. The progress bar under the paten shows the percentage of completion, and clicking on the inside of window with the mouse can change the progress.



The following is program code:

```
1  import processing.io.*;
2
3  int ledPin = 17;    //led Pin
4  int borderSize = 40;  //
5  float t = 0.0;      //progress percent
6  float tStep = 0.004;    // speed
7  SOFTPWM p = new SOFTPWM(ledPin, 10, 100);    //Create a PWM pin,initialize the duty cycle
8  and period
9  void setup() {
```

```
10    size(640, 360);  //display window size
11    strokeWeight(4);  //stroke Weight
12  }
13
14  void draw() {
15    // Show static value when mouse is pressed, animate otherwise
16    if (mousePressed) {
17      int a = constrain(mouseX, borderSize, width - borderSize);
18      t = map(a, borderSize, width - borderSize, 0.0, 1.0);
19    } else {
20      t += tStep;
21      if (t > 1.0) t = 0.0;
22    }
23    p.softPwmWrite((int)(t*100)); //write the duty cycle according to t
24    background(255);  //A white background
25    titleAndSiteInfo();  //title and Site information
26
27    fill(255, 255-t*255, 255-t*255);  //cycle
28    ellipse(width/2, height/2, 100, 100);
29
30    pushMatrix();
31    translate(borderSize, height - 45);
32    int barLength = width - 2*borderSize;
33
34    barBgStyle();  //progressbar bg
35    line(0, 0, barLength, 0);
36    line(barLength, -5, barLength, 5);
37
38    barStyle();  //progressbar
39    line(0, -5, 0, 5);
40    line(0, 0, t*barLength, 0);
41
42    barLabelStyle();    //progressbar label
43    text("progress : "+nf(t*100,2,2),barLength/2,-25);
44    popMatrix();
45  }
46
47  void titleAndSiteInfo() {
48    fill(0);
49    textAlign(CENTER);    //set the text centered
50    textSize(40);         //set text size
51    text("Breathing Light", width / 2, 40);    //title
52    textSize(16);
53    text("www.freenove.com", width / 2, height - 20);    //site
```

```
54  }
55  void barBgStyle() {
56    stroke(220);
57    noFill();
58  }
59
60  void barStyle() {
61    stroke(50);
62    noFill();
63  }
64
65  void barLabelStyle() {
66    noStroke();
67    fill(120);
68  }
```

First, use SOFTPWM class to create a PWM pin, which is used to control the brightness of LED. Then define a variable "t" and a variable "tStep" to control the PWM duty cycle and the rate at which "t" increases.

```
float t = 0.0;       //progress percent
float tStep = 0.004;    // speed
SOFTPWM p = new SOFTPWM(ledPin, 10, 100);
```

In the function draw, if there is a click detected, the coordinate in X direction of the mouse will be mapped into the duty cycle "t"; Otherwise, duty cycle "t" will be increased gradually and PWM with the duty cycle will be output.

```
  if (mousePressed) {
    int a = constrain(mouseX, borderSize, width - borderSize);
    t = map(a, borderSize, width - borderSize, 0.0, 1.0);
  } else {
    t += tStep;
    if (t > 1.0) t = 0.0;
  }
  p.softPwmWrite((int)(t*100)); //write the duty cycle according to t
```

The next code is designed to draw a circle filled with colors in different depth according to the "t" value, which is used to simulate LEDs with different brightness.

```
  fill(255, 255-t*255, 255-t*255);  //cycle
  ellipse(width/2, height/2, 100, 100);
```

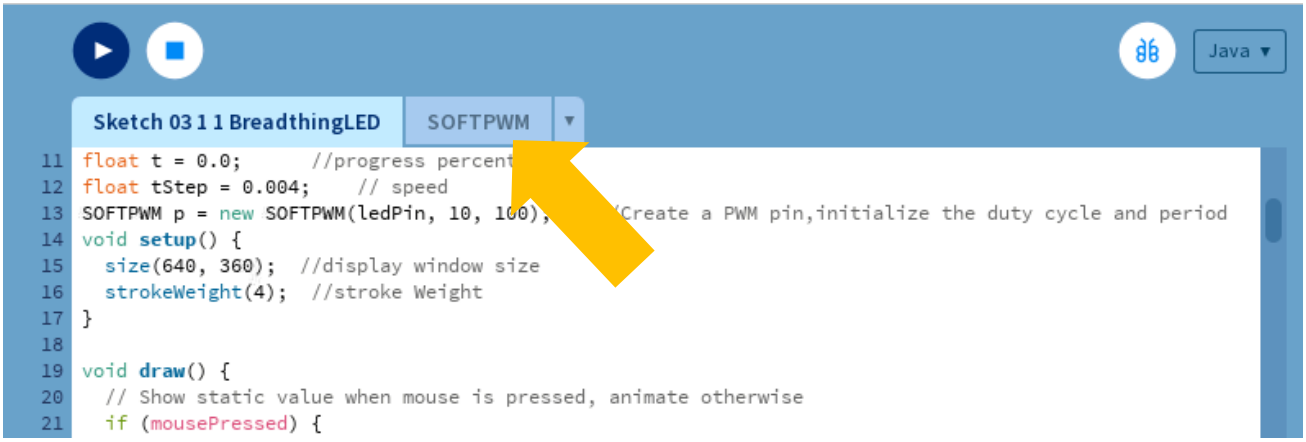
The last code is designed to draw the progress bar and the percentage of the progress.

```
  barBgStyle();  //progressbar bg
  line(0, 0, barLength, 0);
  line(barLength, -5, barLength, 5);
```

```
    barStyle();  //progressbar
    line(0, -5, 0, 5);
    line(0, 0, t*barLength, 0);

    barLabelStyle();     //progressbar label
    text("progress : "+nf(t*100,2,2),barLength/2,-25);
```

In processing software, you will see a tag page "SOFTPWM" in addition to the above code.



Reference

class SOFTPWM

`public SOFTPWM(int iPin, int dc, int pwmRange);`

Constructor, used to create a PWM pin, set the pwmRange and initial duty cycle. The minimum of pwmRange is 0.1ms. So pwmRange=100 means that the PWM duty cycle is 0.1ms*100=10ms.

`public void softPwmWrite(int value)`

Set PMW duty cycle.

`public void softPwmStop()`

Stop outputting PWM.

Chapter 4 RGBLED

In this chapter, we will learn how to use RGBLED.

Project 4.1 Multicolored LED

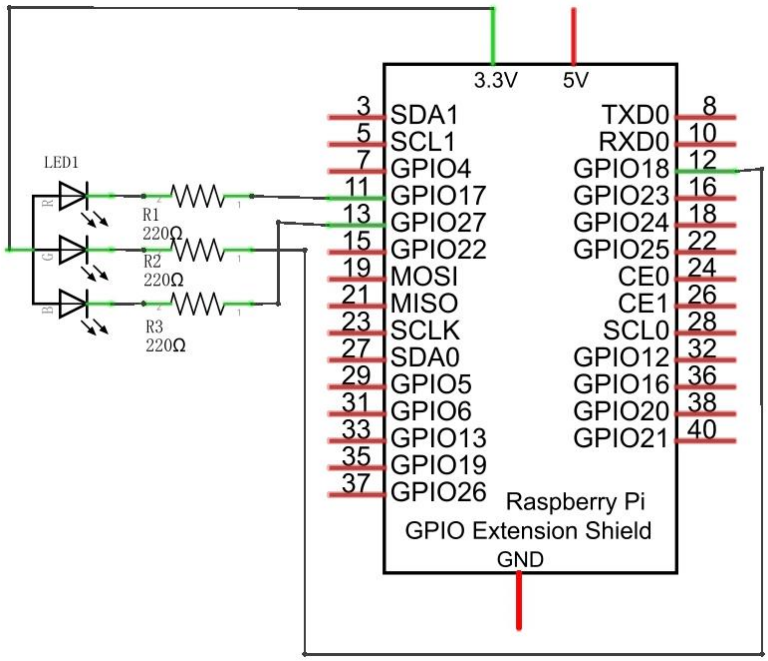
This project will make a Multicolored LED, namely, use Processing to control the color of RGBLED.

Component List

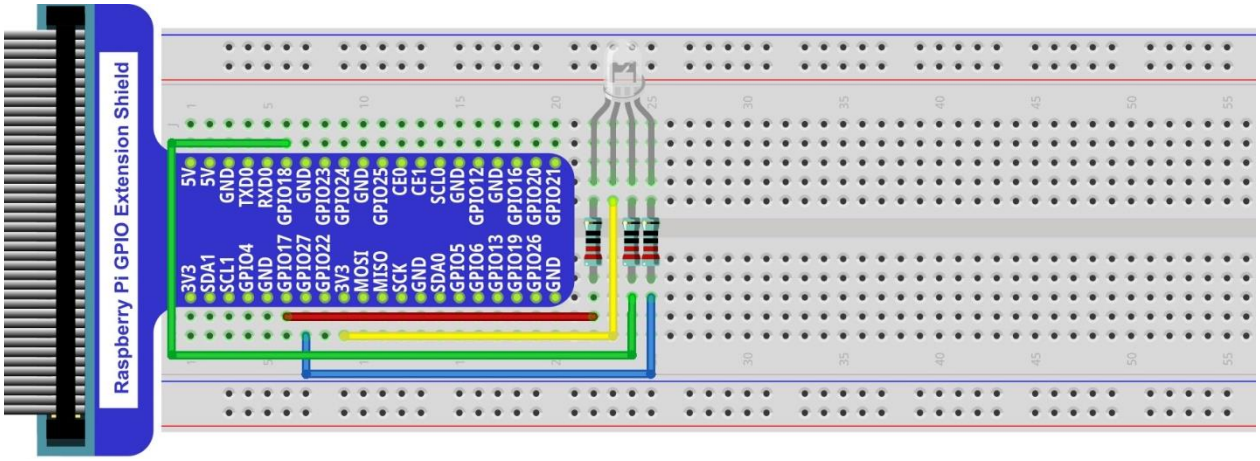
Raspberry Pi x1 GPIO Extension Board & Wire x1 Breadboard x1	RGBLED x1 	Resistor 220Ω x3 
Jumper M/M x4 		

Circuit

Schematic diagram



Hardware connection



Sketch

Sketch 1.1.1 ColorfulLED

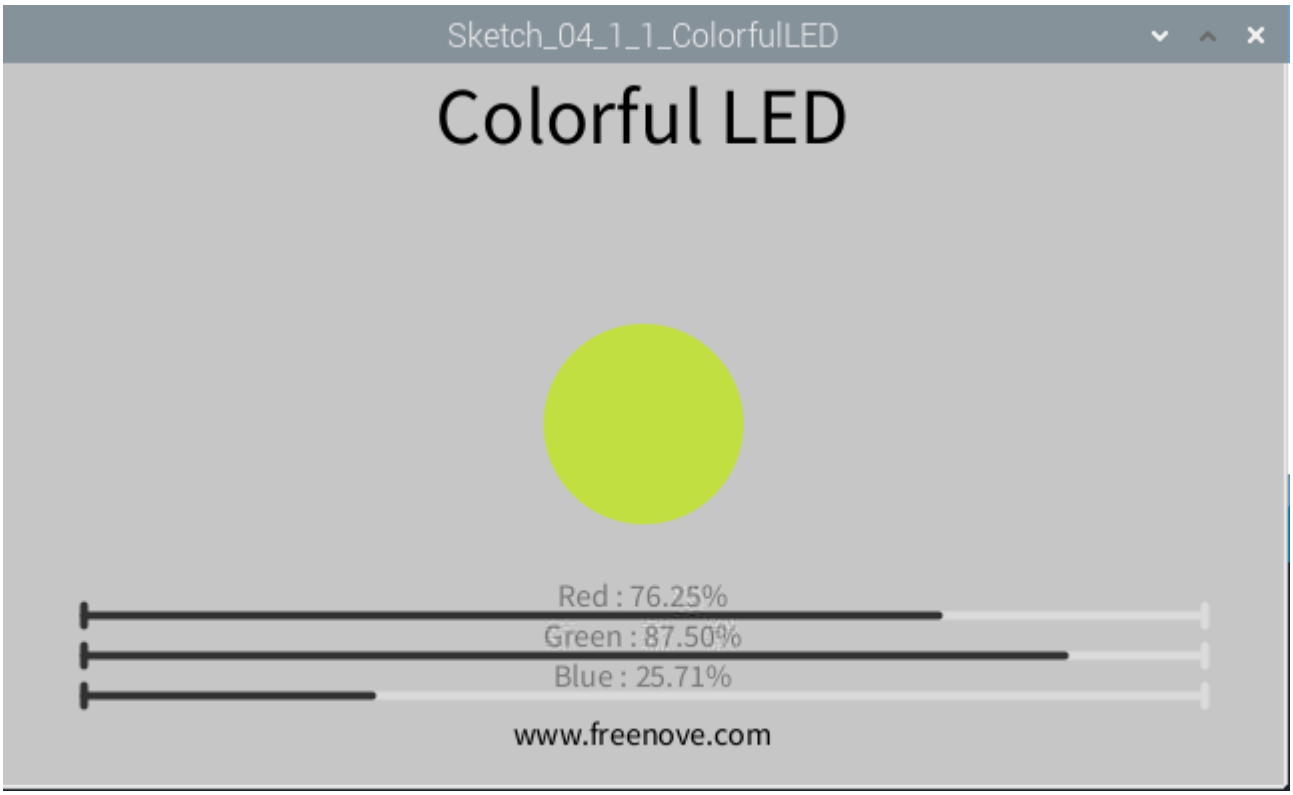
First, observe the result after running the sketch, and then learn about the code in detail.

1. Use Processing to open the file Sketch_01_1_1_ColorfulLED.

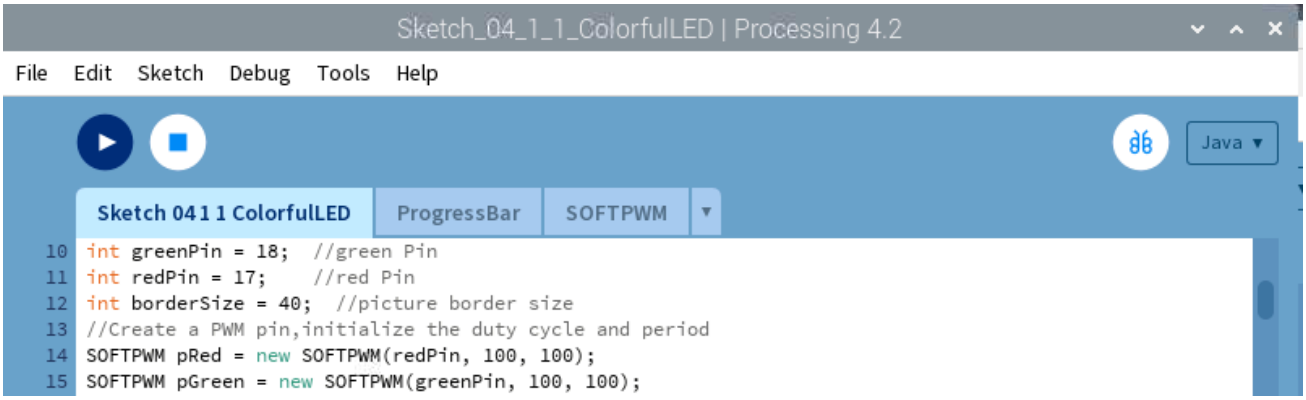
```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_01_1_1_ColorfulLED/Sketch_01_1_1_ColorfulLED.pde
```

2. Click on "RUN" to run the code.

After the program is executed, RGBLED is in OFF-state. And in Display Window, the pattern used to simulate LED is black. Red, Green and Blue progress bars are at 0%. By using mouse to click on and drag any progress bar, you can set the PWM duty cycle of color channels, and then RGBLED in the circuit will show corresponding colors. At the same time, the pattern in Display Window will show the same color.



This project contains a lot of code files, and the core code is contained in the file Sketch_01_1_1_ColorfulLED. The other files only contain some custom classes.



The following is program code:

```
import processing.io.*;

int bluePin = 27;    //blue Pin
int greenPin = 18;  //green Pin
int redPin = 17;    //red Pin
int borderSize = 40;  //picture border size
//Create a PWM pin, initialize the duty cycle and period
SOFTPWM pRed = new SOFTPWM(redPin, 100, 100);
SOFTPWM pGreen = new SOFTPWM(greenPin, 100, 100);
SOFTPWM pBlue = new SOFTPWM(bluePin, 100, 100);
//instantiate three ProgressBar Object
ProgressBar rBar, gBar, bBar;
boolean rMouse = false, gMouse = false, bMouse = false;
void setup() {
  size(640, 360);  //display window size
  strokeWeight(4);  //stroke Weight
  //define the ProgressBar length
  int barLength = width - 2*borderSize;
  //Create ProgressBar Object
  rBar = new ProgressBar(borderSize, height - 85, barLength);
  gBar = new ProgressBar(borderSize, height - 65, barLength);
  bBar = new ProgressBar(borderSize, height - 45, barLength);
  //Set ProgressBar's title
  rBar.setTitle("Red");gBar.setTitle("Green");bBar.setTitle("Blue");
}

void draw() {
  background(200);  //A white background
  titleAndSiteInfo();  //title and Site information

  fill(rBar.progress*255, gBar.progress*255, bBar.progress*255);  //cycle color
  ellipse(width/2, height/2, 100, 100);  //show cycle

  rBar.create();  //Show progressBar
  gBar.create();
  bBar.create();
}

void mousePressed() {
  if ( (mouseY< rBar.y+5) && (mouseY>rBar.y-5) ) {
    rMouse = true;
  } else if ( (mouseY< gBar.y+5) && (mouseY>gBar.y-5) ) {
    gMouse = true;
```

```
44    } else if ( (mouseY< bBar.y+5) && (mouseY>bBar.y-5) ) {
45      bMouse = true;
46    }
47  }
48  void mouseReleased() {
49    rMouse = false;
50    bMouse = false;
51    gMouse = false;
52  }
53  void mouseDragged() {
54    int a = constrain(mouseX, borderSize, width - borderSize);
55    float t = map(a, borderSize, width - borderSize, 0.0, 1.0);
56    if (rMouse) {
57      pRed.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
58      rBar.setProgress(t);
59    } else if (gMouse) {
60      pGreen.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
61      gBar.setProgress(t);
62    } else if (bMouse) {
63      pBlue.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
64      bBar.setProgress(t);
65    }
66  }
67
68  void titleAndSiteInfo() {
69    fill(0);
70    textAlign(CENTER);    //set the text centered
71    textSize(40);         //set text size
72    text("Colorful LED", width / 2, 40);    //title
73    textSize(16);
74    text("www.freenove.com", width / 2, height - 20);    //site
75  }
```

In the code, first create three PWM pins and three progress bars to control RGBLED.

```
SOFTPWM pRed = new SOFTPWM(redPin, 100, 100);
SOFTPWM pGreen = new SOFTPWM(greenPin, 100, 100);
SOFTPWM pBlue = new SOFTPWM(bluePin, 100, 100);
//instantiate three ProgressBar Object
ProgressBar rBar, gBar, bBar;
```

And then in function setup(), define position and length of progress bar according to the size of Display Window, and set the name of each progress bar.

```
void setup() {
  size(640, 360);  //display window size
```

```
  strokeWeight(4);  //stroke Weight
  //define the ProgressBar length
  int barLength = width - 2*borderSize;
  //Create ProgressBar Object
  rBar = new ProgressBar(borderSize, height - 85, barLength);
  gBar = new ProgressBar(borderSize, height - 65, barLength);
  bBar = new ProgressBar(borderSize, height - 45, barLength);
  //Set ProgressBar's title
  rBar.setTitle("Red");gBar.setTitle("Green");bBar.setTitle("Blue");
}
```

In function draw(), first set background, header and other basic information. Then draw a circle and set its color according to the duty cycle of three channels of RGB. Finally draw three progress bars.

```
void draw() {
  background(200);  //A white background
  titleAndSiteInfo();  //title and Site information

  fill(rBar.progress*255, gBar.progress*255, bBar.progress*255);  //cycle color
  ellipse(width/2, height/2, 100, 100);  //show cycle

  rBar.create();  //Show progressBar
  gBar.create();
  bBar.create();
}
```

System functions mousePressed(), mouseReleased() and mouseDragged() are used to determine whether the mouse drags the progress bar and set the schedule. If the mouse button is pressed in a progress bar, then the mousePressed () sets the progress flag rgbMouse to true, mouseDragged (mouseX) maps progress value to set corresponding PWM. When the mouse is released, mouseReleased() sets the progress flag rgbMouse to false..

```
void mousePressed() {
  if ( (mouseY< rBar.y+5) && (mouseY>rBar.y-5) ) {
    rMouse = true;
  } else if ( (mouseY< gBar.y+5) && (mouseY>gBar.y-5) ) {
    gMouse = true;
  } else if ( (mouseY< bBar.y+5) && (mouseY>bBar.y-5) ) {
    bMouse = true;
  }
}
void mouseReleased() {
  rMouse = false;
  bMouse = false;
  gMouse = false;
}
```

```
void mouseDragged() {
  int a = constrain(mouseX, borderSize, width - borderSize);
  float t = map(a, borderSize, width - borderSize, 0.0, 1.0);
  if (rMouse) {
    pRed.softPwmWrite((int)(100-t*100)); //write the duty cycle according to t
    rBar.setProgress(t);
  } else if (gMouse) {
    pGreen.softPwmWrite((int)(100-t*100)); //write the duty cycle according to t
    gBar.setProgress(t);
  } else if (bMouse) {
    pBlue.softPwmWrite((int)(100-t*100)); //write the duty cycle according to t
    bBar.setProgress(t);
  }
}
```

Reference

class ProgressBar

This is a custom class that is used to create a progress bar.

`public ProgressBar(int ix, int iy, int barlen)`

Constructor, used to create ProgressBar, the parameters for coordinates X, Y and length of ProgressBar.

`public void setTitle(String str)`

Used to set the name of progress bar, which will be displayed in the middle of the progress bar.

`public void setProgress(float pgress)`

Used to set the progress of progress bar. The parameter: 0<pgress<1.0.

`public void create() & public void create(float pgress)`

Used to draw progress bar.

Chapter 5 Buzzer

In this chapter we will learn how to use a buzzer.

Project 5.1 ActiveBuzzer

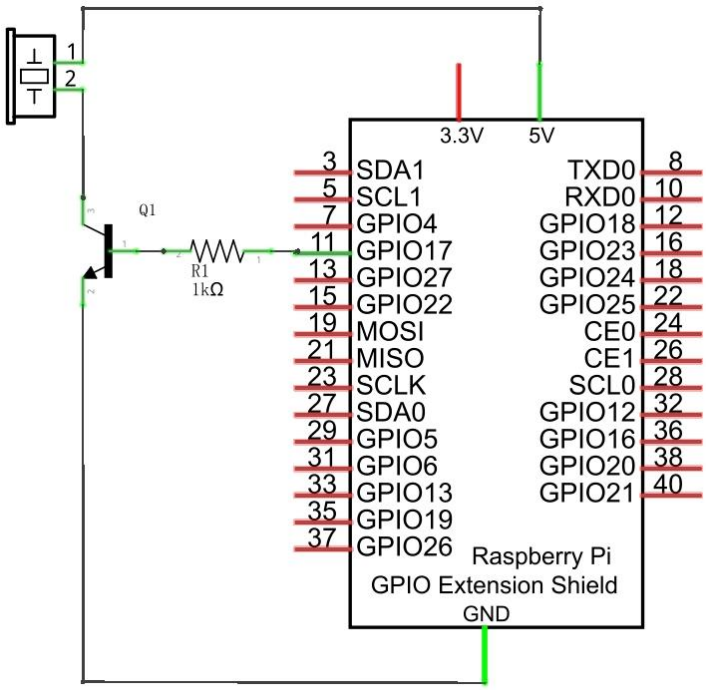
In this project, we will use the mouse to control an active buzzer.

Component List

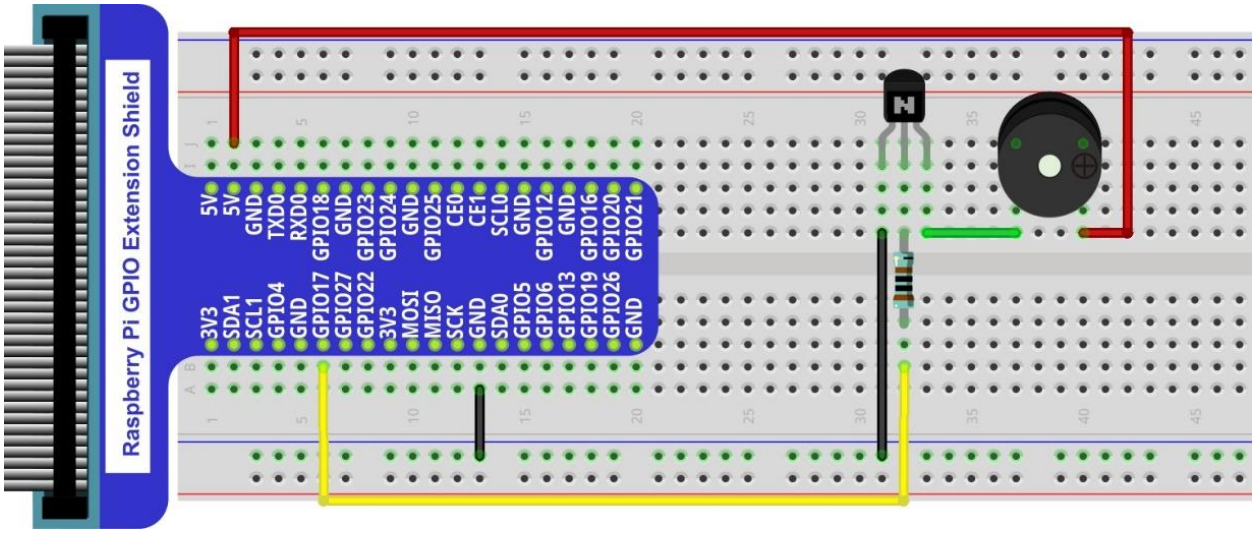
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Wire x1
Breadboard x1</td><td>Jumper M/M x7
  </td></tr>
<tr><td>NPN transistorx1
  </td><td>Active buzzer x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection



Note: in this circuit, the power supply for the buzzer is 5V, and pull-up resistor of the push button switch is connected to the 3.3V power feed. Actually, the buzzer can work when connected to the 3.3V power feed but this will produce a weak sound from the buzzer (not very loud).

Sketch

Sketch 2.1.1 ActiveBuzzer

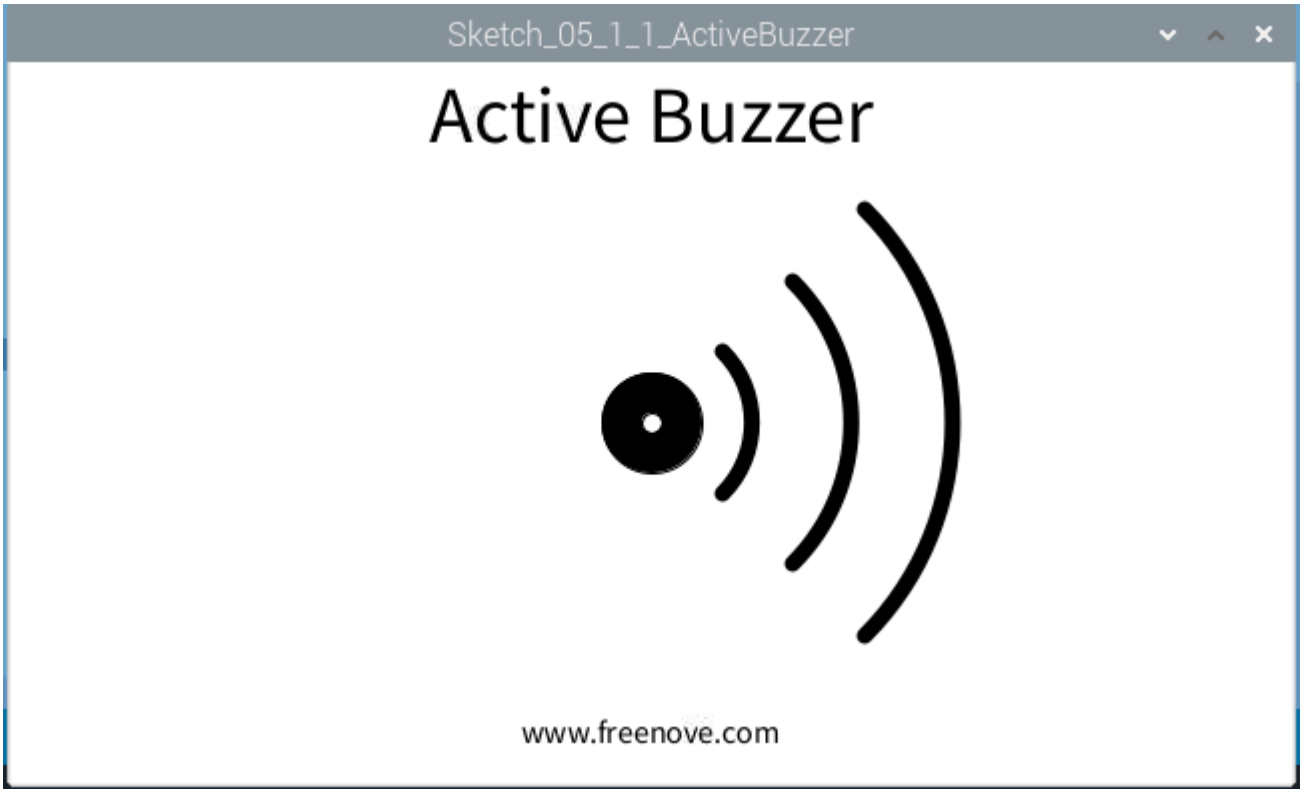
First, observe the result after running the sketch, and then learn about the code in detail.

1. Use Processing to open the file Sketch_02_1_1_ActiveBuzzer.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_02_1_1_ActiveBuzzer/Sketch_02_1_1_ActiveBuzzer.pde
```

2. Click on "RUN" to run the code.

After the program is executed, use the mouse to click on any position of the Display Window, then Active Buzzer begins to sound and arc graphics (Schematic of sounding) will appear next to the buzzer pattern on Display Window. Click the mouse again, then Active Buzzer stops sounding and arc graphics disappear.



The following is program code:

```
import processing.io.*;

int buzzerPin = 17;
boolean buzzerState = false;
void setup() {
  size(640, 360);
  GPIO.pinMode(buzzerPin, GPIO.OUTPUT);
}

void draw() {
```

```
  background(255);
  titleAndSiteInfo();    //title and site information
  drawBuzzer();       //buzzer img
  if (buzzerState) {
    GPIO.digitalWrite(buzzerPin, GPIO.HIGH);
    drawArc();      //Sounds waves img
  } else {
    GPIO.digitalWrite(buzzerPin, GPIO.LOW);
  }
}

void mouseClicked() { //if the mouse Clicked
  buzzerState = !buzzerState;  //Change the buzzer State
}
void drawBuzzer() {
  strokeWeight(1);
  fill(0);
  ellipse(width/2, height/2, 50, 50);
  fill(255);
  ellipse(width/2, height/2, 10, 10);
}
void drawArc() {
  noFill();
  strokeWeight(8);
  for (int i=0; i<3; i++) {
    arc(width/2, height/2, 100*(1+i), 100*(1+i), -PI/4, PI/4, OPEN);
  }
}
void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(40);         //set text size
  text("Active Buzzer", width / 2, 40);    //title
  textSize(16);
  text("www.freenove.com", width / 2, height - 20);    //site
}
```

Code in this project is logically the same as previous "MouseLED" project. And the difference is that this project needs to draw the buzzer pattern and arc graphics after the buzzer sounding.

App 1 Snake Game

In this chapter, we will play a classic game, snake.

App 1.1 Snake Game

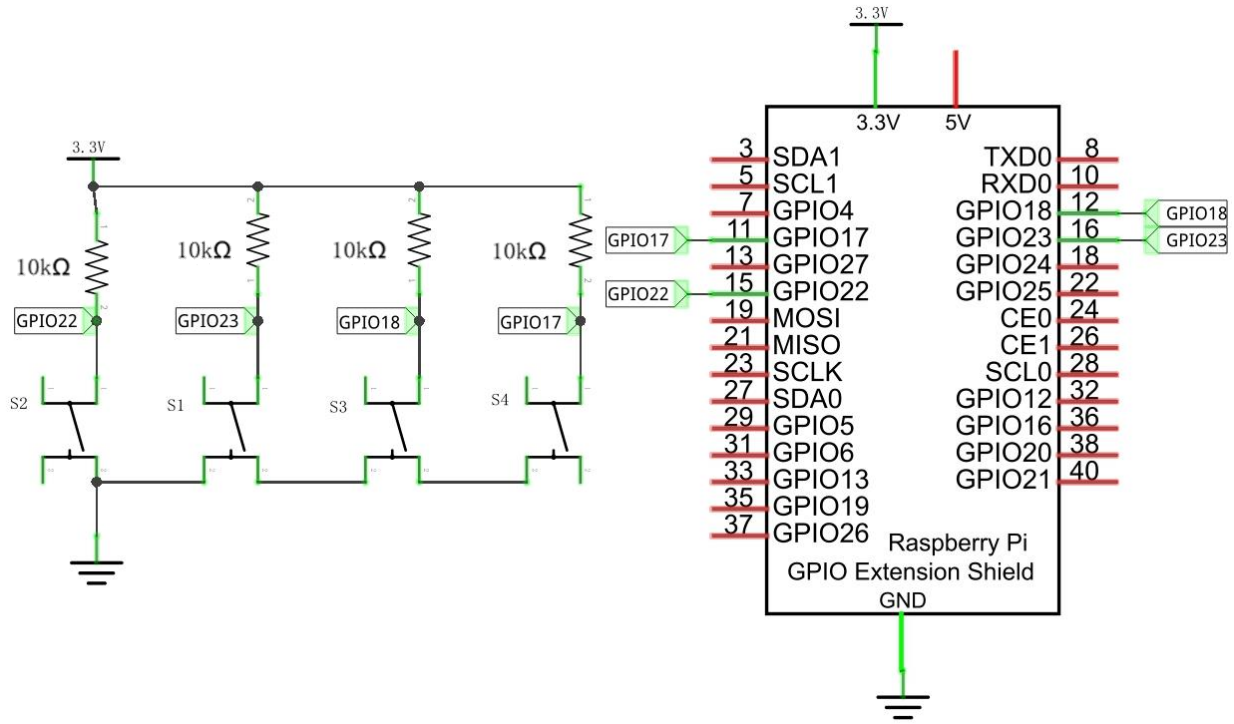
Now, let's create and experience our own game.

Component List

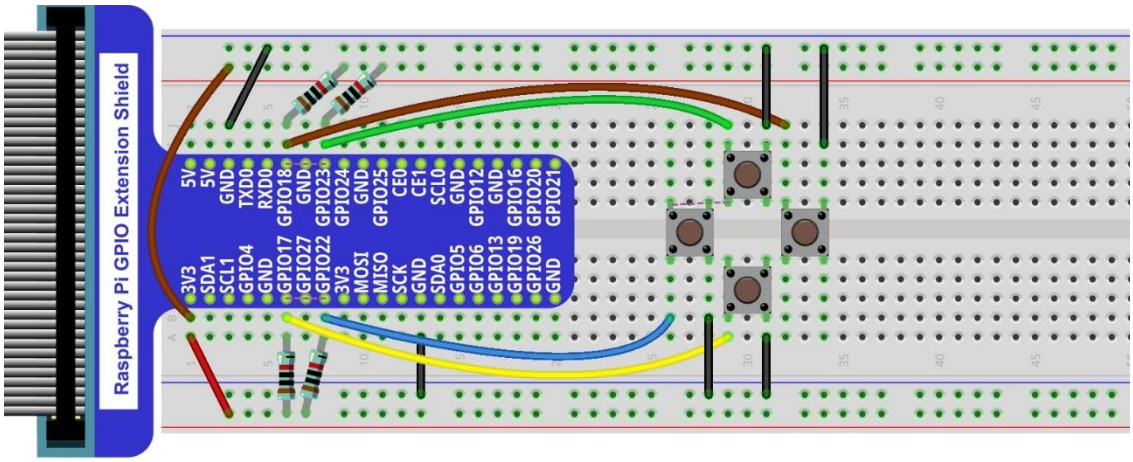
Raspberry Pi x1 GPIO Extension Board & Wire x1 Breadboard x1	Resistor 10KΩ x4 	Push button x4 
Jumper M/M x12 		

Circuit

Schematic diagram



Hardware connection



Sketch

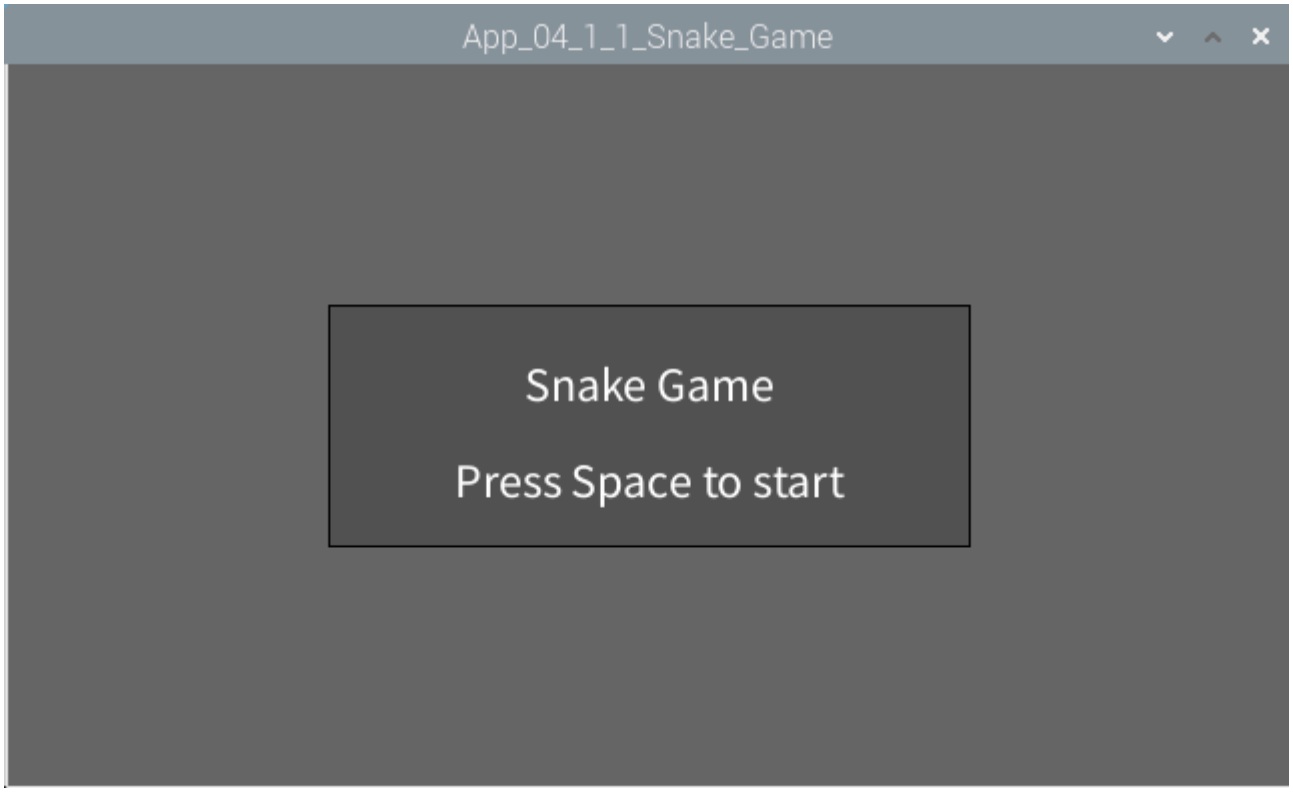
Sketch 1.1.1 SnakeGame

1. Use Processing to open the file Sketch_01_1_1_SnakeGame.

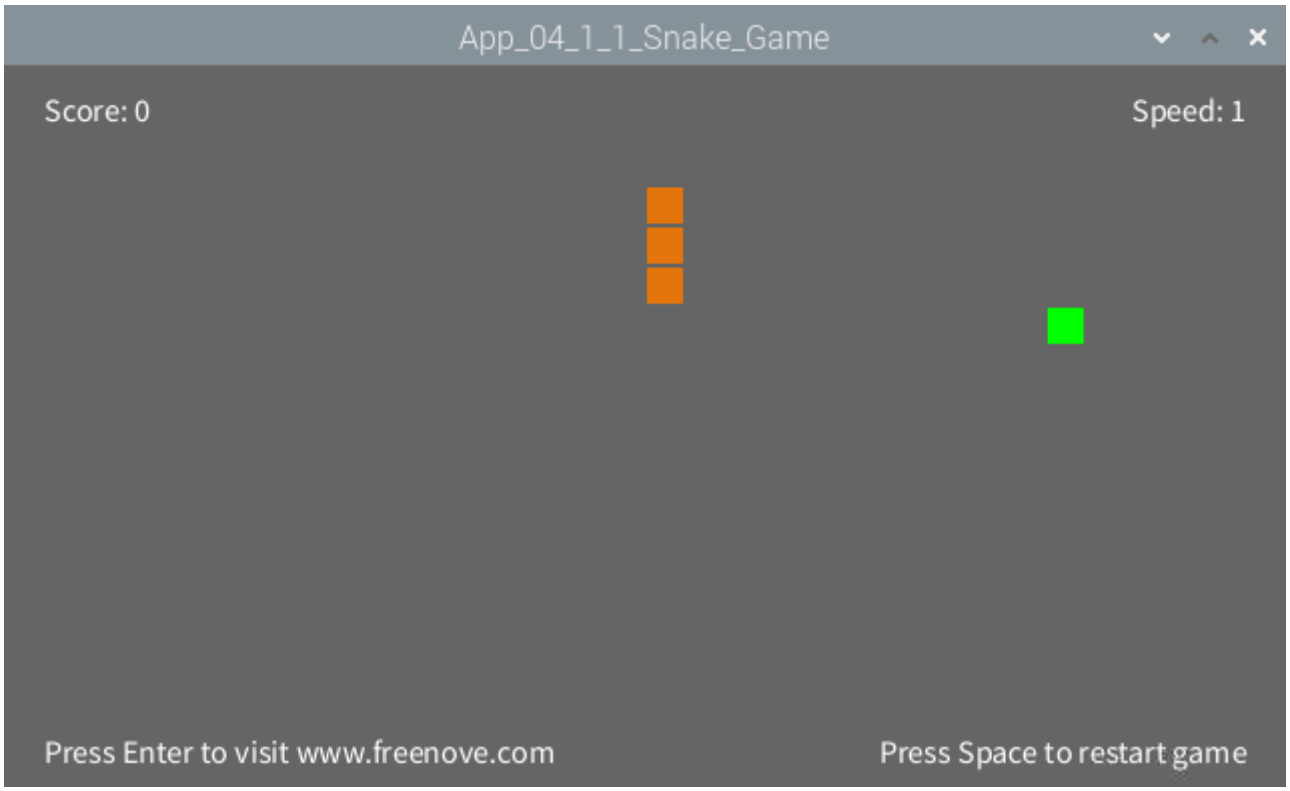
```
processing ~/Freenove_Kit/Processing/Apps/App_01_1_1_Snake_Game/App_01_1_1_Snake_Game.pde
```

2. Click on "RUN" to run the code.

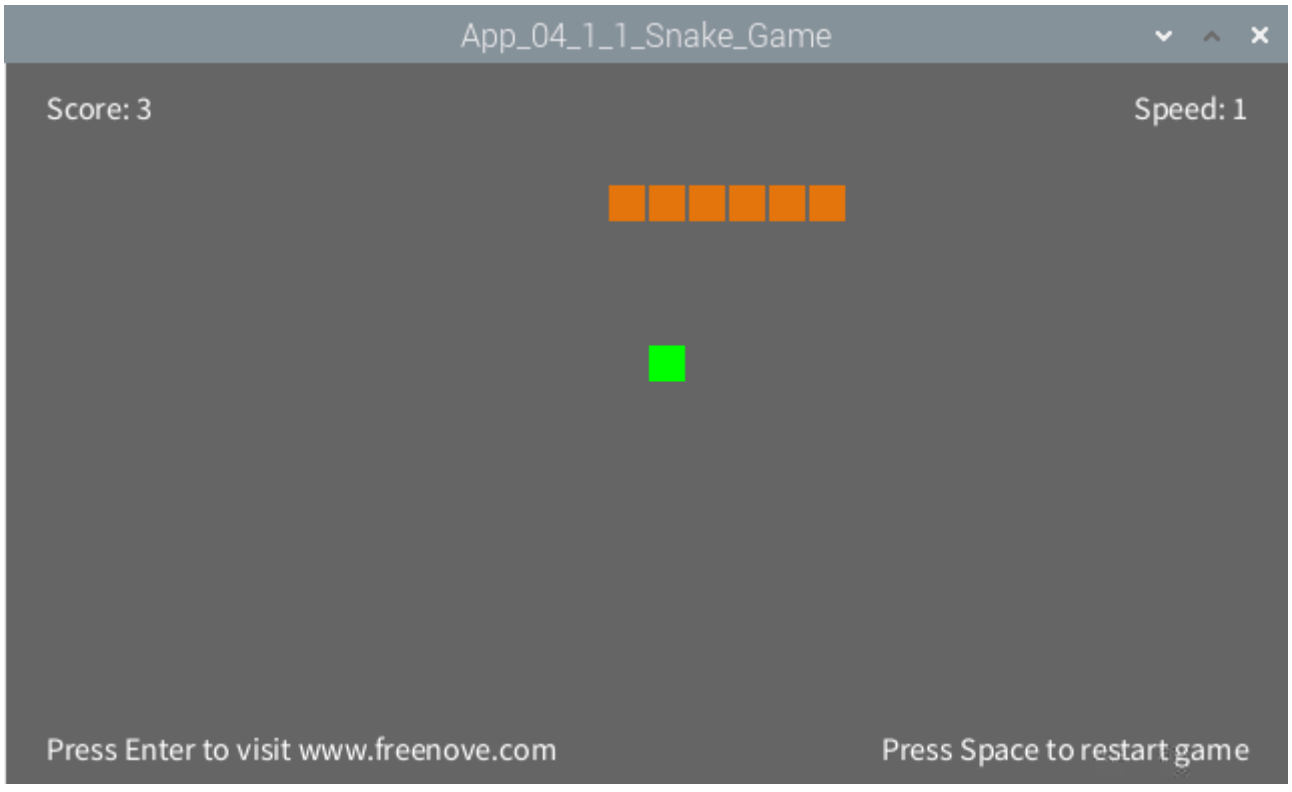
After the program is executed, Display Window displays as below.



Pressing the space can start the game:



You can control the movement direction of the snake through the four buttons in circuit or four arrow keys on the keyboard. The rules are the same as the classic Snake game:



When game is over, pressing the space can restart the game:

You can restart the game by pressing the space bar at any time during the game.

App 2 Tetris Game

In this chapter, we will play a game, Tetris game.

App 2.1 Tetris Game

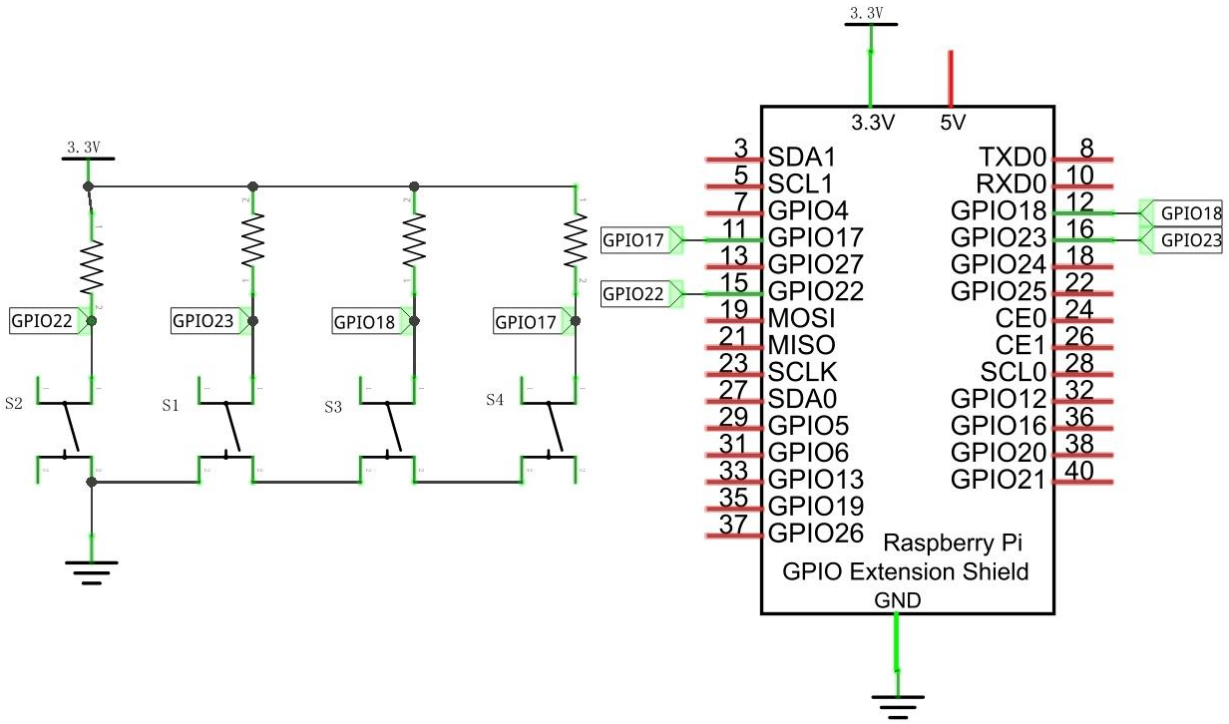
Now, let's create and experience our own game.

Component List

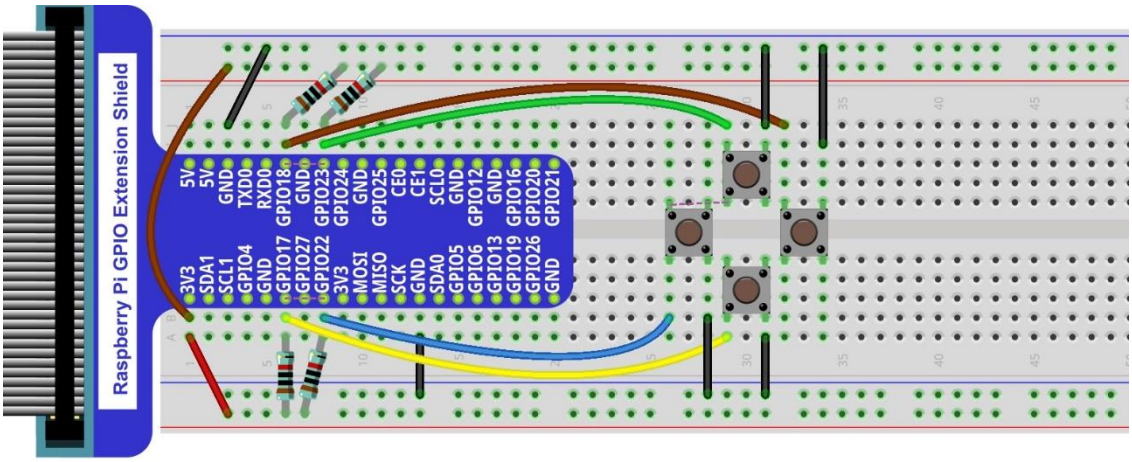
Raspberry Pi x1 GPIO Extension Board & Wire x1 Breadboard x1	Resistor 10KΩ x4 	Push button x4 
Jumper M/M x12 		

Circuit

Schematic diagram



Hardware connection



Sketch

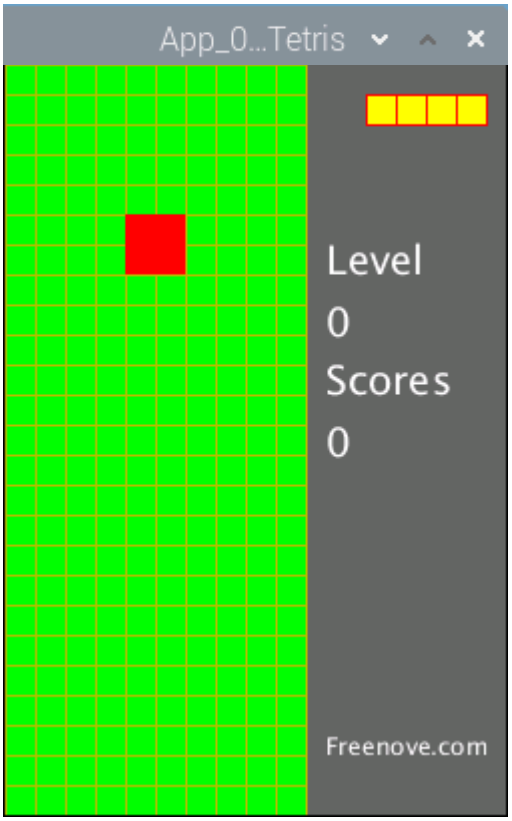
Sketch 2.1.1 TetrisGame

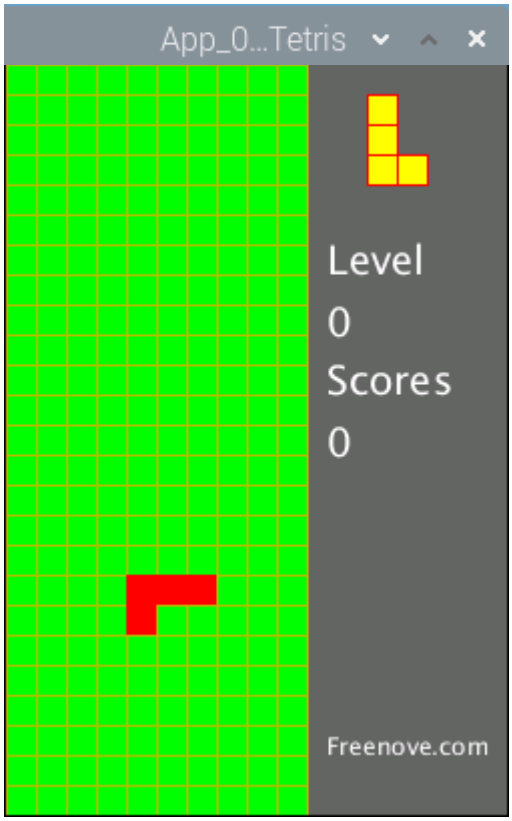
1. Use Processing to open the file Sketch_02_1_1_TetrisGame.

```
processing ~/Freenove_Kit/Processing/Apps/App_02_1_1_Tetris/App_02_1_1_Tetris.pde
```

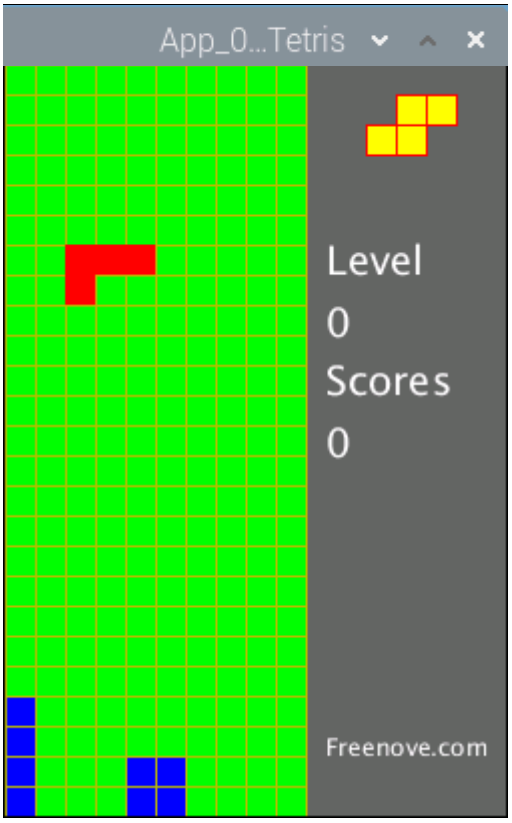
2. Click on "RUN" to run the code.

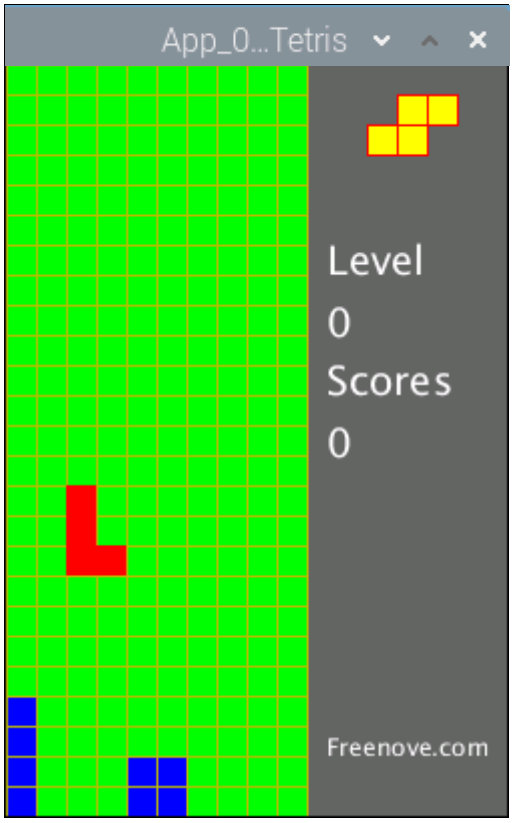
After the program is executed, Display Window displays as below.



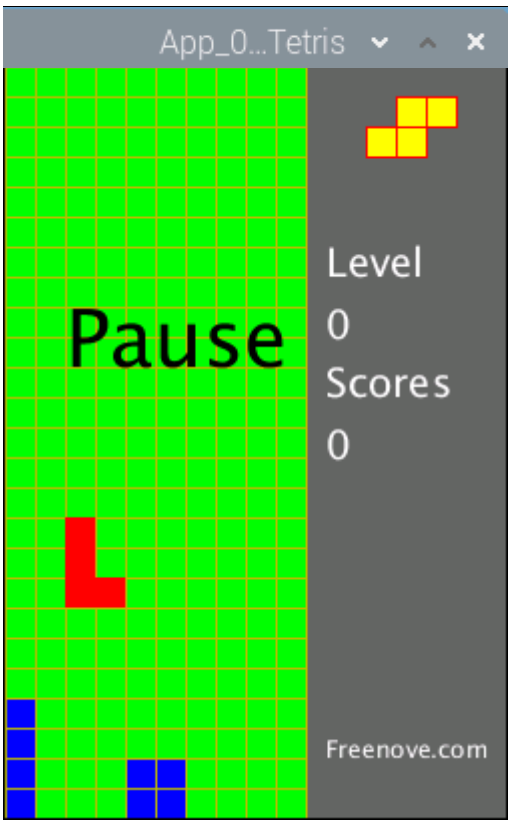


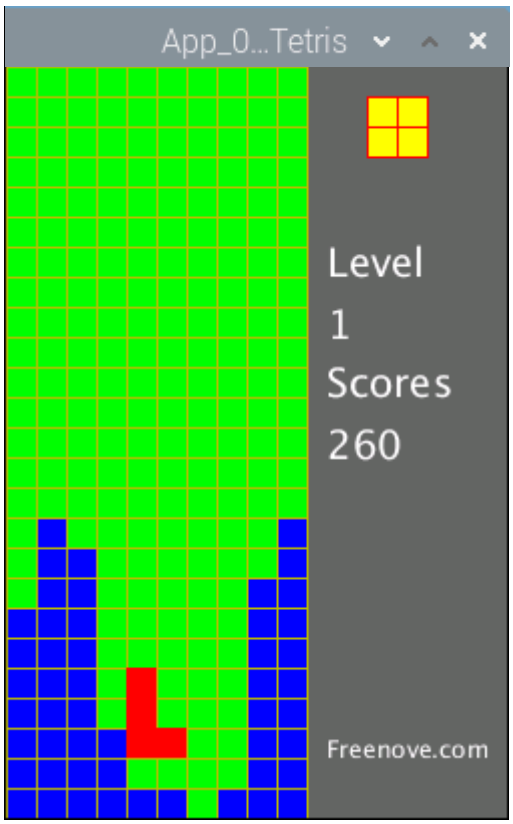
The left and right button in the circuit can control the movement of the falling block to left or right. And the button below can accelerate falling of the block. The button above is used for rotating of the block. Four direction keys on keyboard can also be used to play the game.



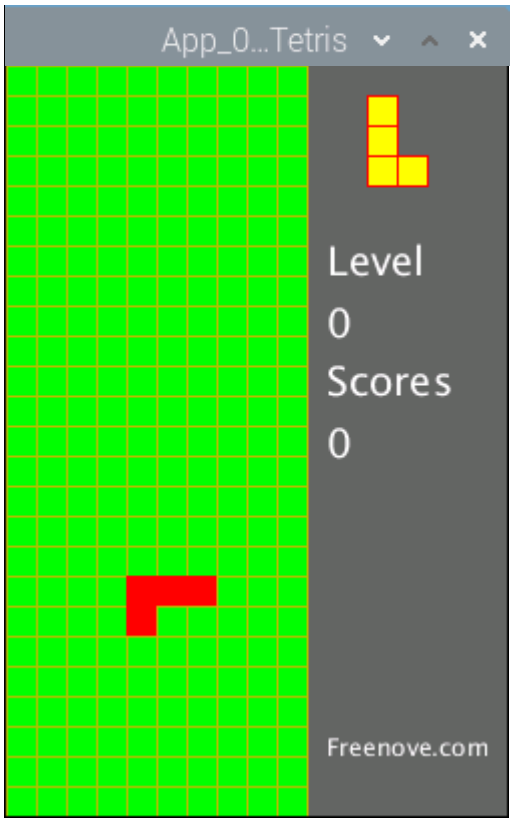


In the process of game, pressing the space bar on the keyboard can pause the game. The right side of the Display Window shows the upcoming block, the current game speed and the current score. The more lines you eliminate once, the higher the scores you will get. If you eliminate one line once, you will get 10 points. If you eliminate 4 lines once, you will get 70 points.





When the blocks are beyond the screen, the game is over. After the game is over, press the space bar to start a new game.



What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
http://www.freenove.com/

Thank you again for choosing Freenove products.