1. Blinking_LED

Introduction

In this lesson, we will learn how to make a blinking LED by programming. Through your settings, your LED can produce a series of interesting phenomena. Now, go for it.

Hardware Required

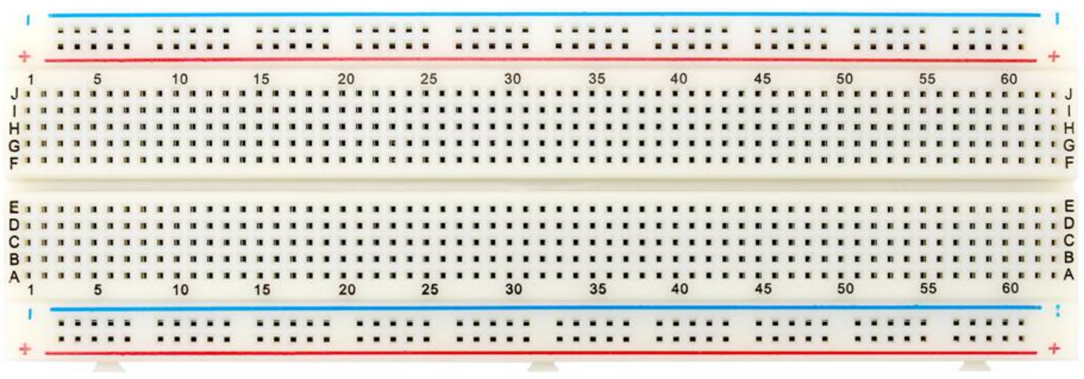
- ✓ 1 * Raspberry Pi
- ✓ 1 * T-Extension Board
- ✓ 1 * LED
- ✓ 1 * 40-pin Cable
- ✓ Several Jumper Wires
- ✓ 1 * Breadboard
- ✓ 1 * Resistor(220Ω)

Note: In order to proceed smoothly, you need to bring your own Raspberry Pi, TF card and Raspberry Pi power.

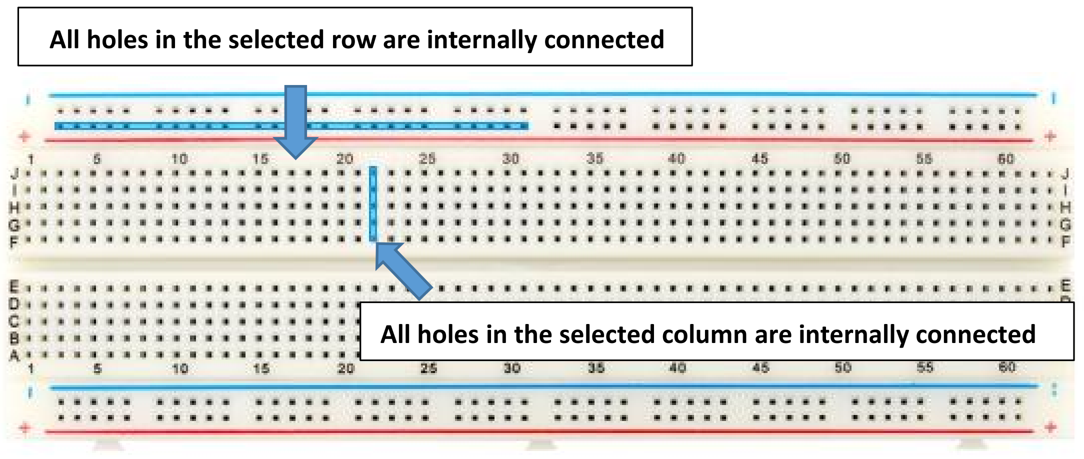
Principle

Breadboard

A breadboard is a solderless device for temporary prototype with electronics and test circuit designs. Most electronic components in electronic circuits can be interconnected by inserting their leads or terminals into the holes and then making connections through wires where appropriate. The breadboard has strips of metal underneath the board and connect the holes on the top of the board. Note that the top and bottom rows of holes are connected horizontally and split in the middle while the remaining holes are connected vertically.



Note how all holes in the selected row are connected together, so the holes in the selected column. The set of connected holes can be called a node:


All holes in the selected row are internally connected
All holes in the selected column are internally connected

LED

Light Emitting Diodes (LEDs) are the most widely used semiconductor diodes among all the different types of semiconductor diodes available today. Light emitting diodes emit either visible light or invisible infrared light when forward biased. The LEDs which emit invisible infrared light are used for remote controls.

A light Emitting Diode (LED) is an optical semiconductor device that emits light when voltage is applied. In other words, LED is an optical semiconductor device that converts electrical energy into light energy.

The safe forward voltage ratings of most LEDs is from 1V to 3 V and forward current ratings is from 20 mA to 30 mA.

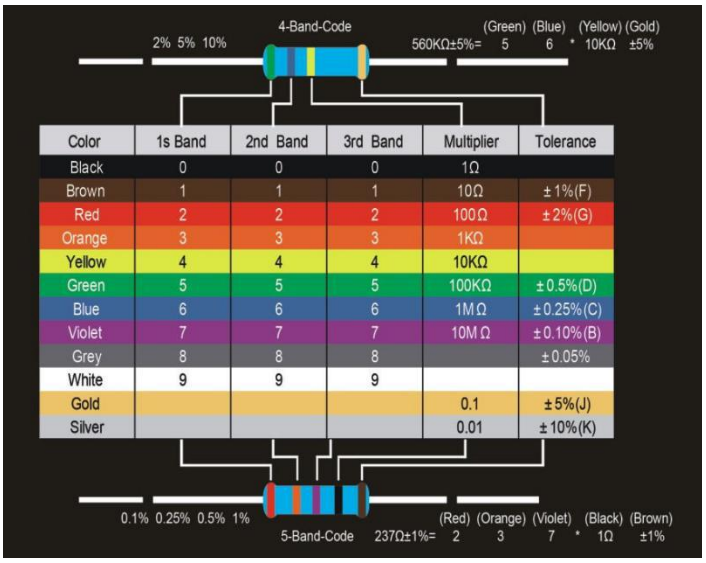
If the voltage applied to LED is in between 1V to 3V, LED works perfectly because the current flow for the applied voltage is in the operating range. However, if the voltage applied to LED is increased to a value greater than 3 volts. The depletion region in the LED breaks down and the electric current suddenly rises. This sudden rise in current may destroy the device.

To avoid this we need to place a resistor (Rs) in series with the LED. The resistor (Rs) must be placed in between voltage source (Vs) and LED.

Resistors

As the name suggests, resistors resist the flow of electricity. The higher the value of the resistor, the more it resists and the less electrical current will flow through it.

The resistor color code has three colored stripes and then a gold stripe at one end.


4-Band-Code
2% 5% 10%
(Green) (Blue) (Yellow) (Gold)
560KΩ±5%= 5 6 * 10KΩ ±5%
0.1% 0.25% 0.5% 1%
5-Band-Code
(Red) (Orange) (Violet) (Black) (Brown)
237Ω±1%= 2 3 7 * 1Ω ±1%

Color	1s Band	2nd Band	3rd Band	Multiplier	Tolerance
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	±1%(F)
Red	2	2	2	100Ω	±2%(G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	±0.5%(D)
Blue	6	6	6	1MΩ	±0.25%(C)
Violet	7	7	7	10MΩ	±0.10%(B)
Grey	8	8	8		±0.05%
White	9	9	9		
Gold				0.1	±5%(J)
Silver				0.01	±10%(K)

Unlike LEDs, resistors do not have a positive and negative lead. They can be onnected either way around.

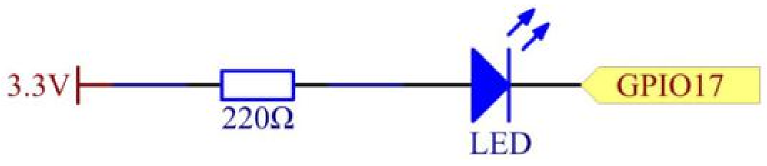
Schematic Diagram

In this experiment, connect a 220 Ω resistor to the anode (the long pin of the LED),then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to GPIO17 of Raspberry Pi. Therefore, to turn on an LED, we need to make GPIO17 low (0V) level. We can get this phenomenon by programming.

Note: Pin11 refers to the 11th pin of the Raspberry Pi from left to right, and its corresponding wiringPi and BCM pin numbers are shown in the following table.

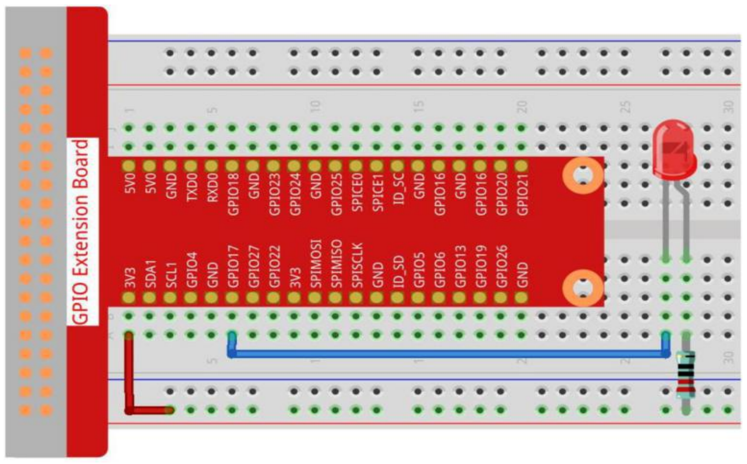
In the C language related content, we make GPIO0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi, Pin 11.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



Experimental Procedures

Step 1: Build the circuit.

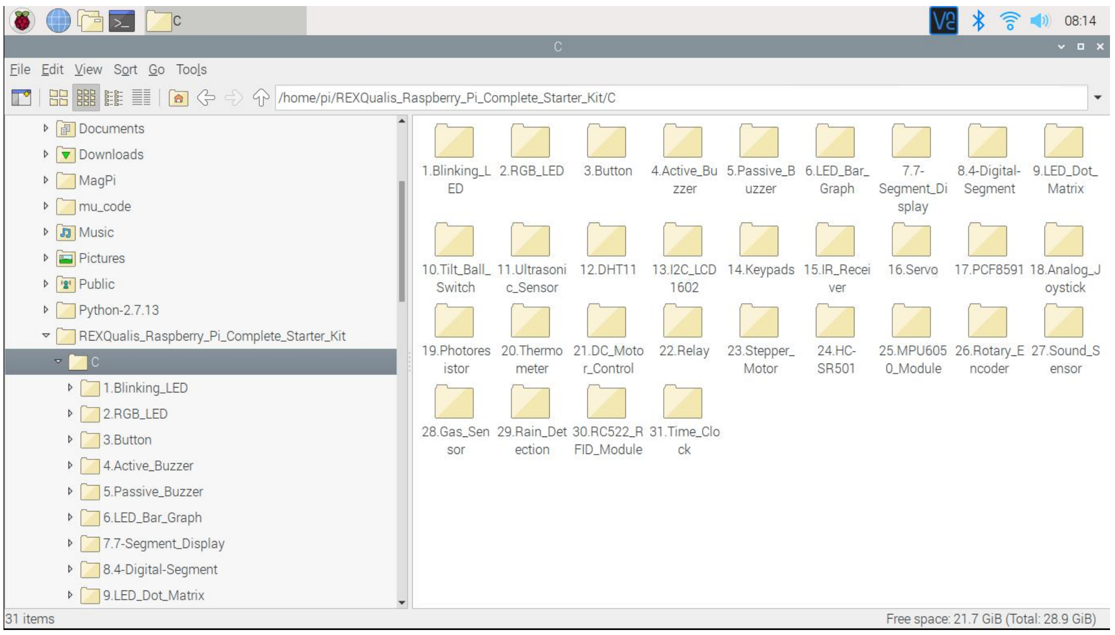


For C Language Users

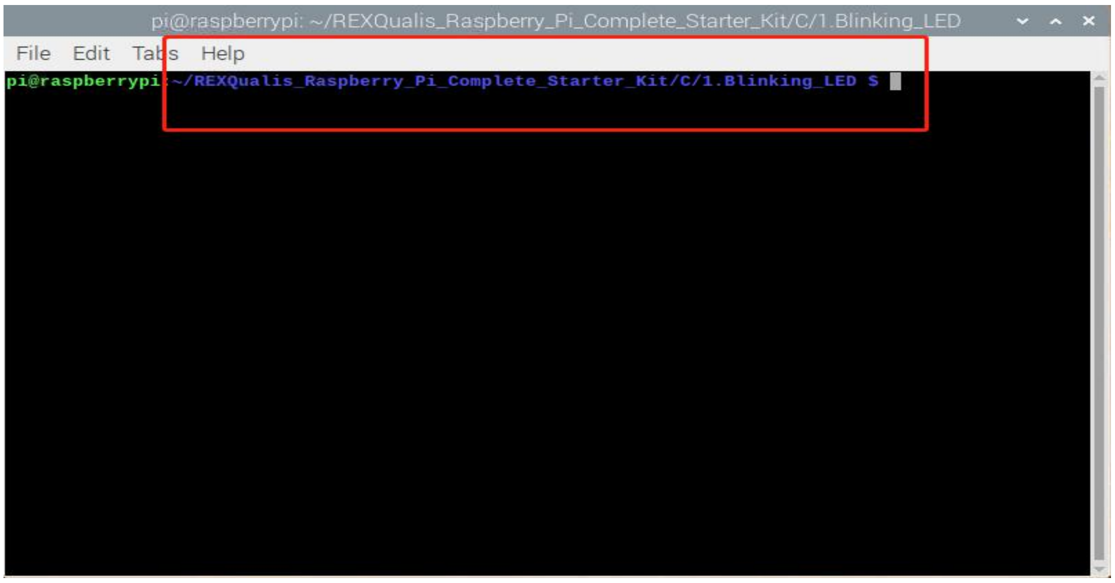
Step 2: Go to the folder of the code.

If you use a monitor, you're recommended to take the following steps.

Go to /home/pi/ and find the folder REXQualis RaspberryPi Complete Starter Kit Find C in the folder, right-click on it and select Open in Terminal.



Then a window will pop up as shown below. So now you've entered the path of the code 1.Blinking_LED.c .


pi@raspberrypi: ~/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED
File Edit Tabs Help

```
pi@raspberrypi:~/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED $
```

In the following lessons, we will use command to enter the code file instead of right-clicking. But you can choose the method you prefer.

If you log into the Raspberry Pi remotely, use “cd” to change directory:

```
cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED
```

Note: Change directory to the path of the code in this experiment via cd.

In either way, now you now are in the folder C. The subsequent procedures based on these two methods are the same. Let's move on.

Step 3: Compile the code

```
gcc 1.Blinking_LED.c -o Blinking_LED.out -lwiringPi
```

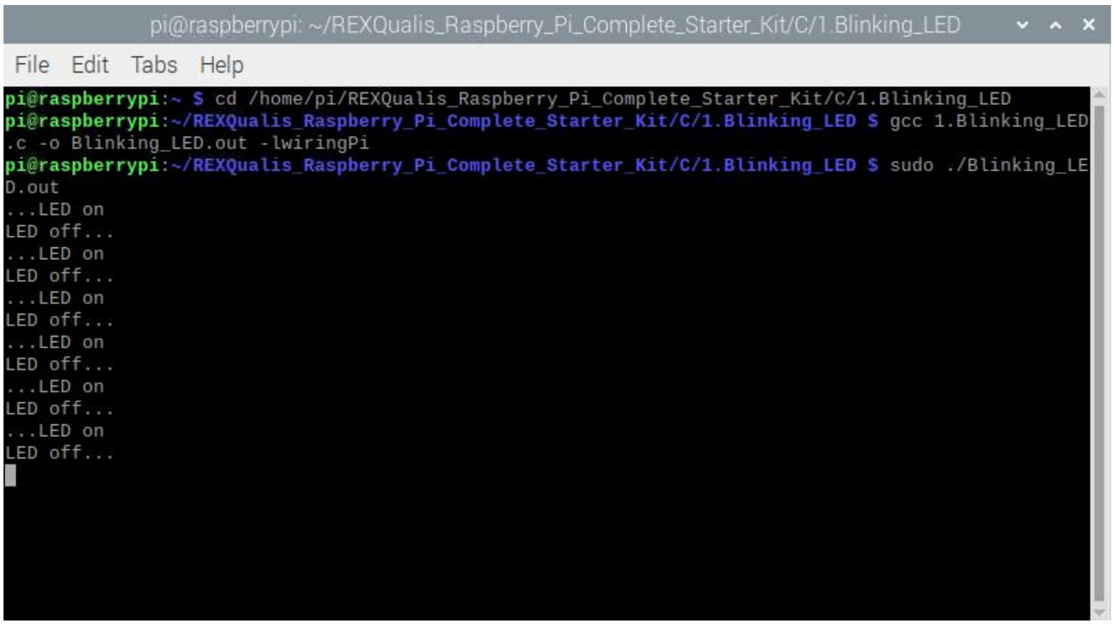
Note: gcc is GNU Compiler Collection. Here, it functions like compiling the C language file 1.Blinking_LED.c and outputting an executable file.

In the command, -o means outputting (the character immediately following -o is the filename output after compilation, and an executable named BlinkingLed will generate here) and -lwiringPi is to load the library wiringPi (l is the abbreviation of library).

Step 4: Run the executable file output in the previous step.

```
sudo ./Blinking_LED.out
```

Note: To control the GPIO, you need to run the program, by the command, sudo(superuser do). The command "./" indicates the current directory. The whole command is to run the BlinkingLed in the current directory.


pi@raspberrypi: ~/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED
File Edit Tabs Help

```
pi@raspberrypi:~ $ cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED
pi@raspberrypi:~/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED $ gcc 1.Blinking_LED
.c -o Blinking_LED.out -lwiringPi
pi@raspberrypi:~/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED $ sudo ./Blinking_LE
D.out
...LED on
LED off...
...LED on
LED off...
...LED on
LED off...
...LED on
LED off...
...LED on
LED off...
...LED on
LED off...
```

After the code runs, you will see the LED flashing.

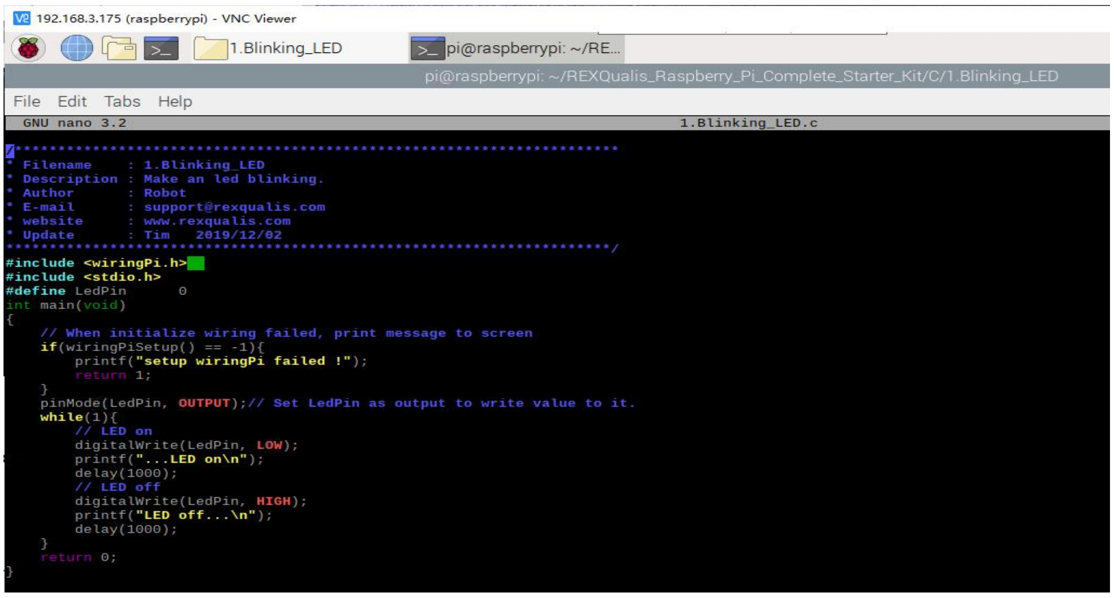
If you want to edit the code file 1.Blinking_LED.c , press Ctrl + C to stop running the code. Then type the following command to open it:

```
nano 1.Blinking_LED.c
```

Note: nano is a text editor tool. The command is used to open the code file 1.Blinking_LED.c by this tool.

Press Ctrl+X to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in Y (save) or N (don ’ t save). Then press Enter to exit.

Repeat Step 3 and Step 4 to see the effect after modifying.


192.168.3.175 (raspberrypi) - VNC Viewer
1.Blinking_LED
pi@raspberrypi: ~/RE...
pi@raspberrypi: ~/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/1.Blinking_LED
File
Edit
Tabs
Help

```
 GNU nano 3.2                                                           1.Blinking_LED.c

/**********************************************************************
* Filename    : 1.Blinking_LED
* Description : Make an led blinking.
* Author      : Robot
* E-mail      : support@rexqualis.com
* website     : www.rexqualis.com
* Update      : Tim   2019/12/02
**********************************************************************/
#include <wiringPi.h>
#include <stdio.h>
#define LedPin      0
int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(LedPin, OUTPUT);// Set LedPin as output to write value to it.
    while(1){
        // LED on
        digitalWrite(LedPin, LOW);
        printf("...LED on\n");
        delay(1000);
        // LED off
        digitalWrite(LedPin, HIGH);
        printf("LED off...\n");
        delay(1000);
    }
    return 0;
}
```

Code

The program code is shown as follows:

```
#include <wiringPi.h>

#include <stdio.h>

#define LedPin 0

int main(void)

{

    // When initialize wiring failed, print message to screen

    if(wiringPiSetup() == -1){

        printf("setup wiringPi failed !");

        return 1;

    }

    pinMode(LedPin, OUTPUT);// Set LedPin as output to write value to it.

    while(1){

        // LED on

        digitalWrite(LedPin, LOW);

        printf("...LED on\n");

        delay(1000);
        // LED off
        digitalWrite(LedPin, HIGH);
        printf("LED off...\n");
        delay(1000);
    }
    return 0;
}
```

Code Explanation

```
#include <wiringPi.h>
```

The hardware drive library is designed for the C language of Raspberry Pi. Adding this library is conducive to the initialization of hardware, and the output of I/O ports,

```
#include <stdio.h>
```

PWM, etc.

Standard I/O library. The pintf function used for printing the data displayed on the screen is realized by this library. There are many other performance functions for you to explore.

```
#define LedPin      0
```

Pin GPIO17 of the T_Extension Board is corresponding to the GPIO0 in wiringPi. Assign GPIO0 to LedPin, LedPin represents GPIO0 in the code later.

```
 if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
```

This initialises wiringPi and assumes that the calling program is going to be using the wiringPi pin numbering scheme.

This function needs to be called with root privileges. When initialize wiring failed, print message to screen. The function “return” is used to jump out of the current function. Using return in main() function will end the program.

```
    pinMode(LedPin, OUTPUT);
```

Set LedPin as output to write value to it.

```
    digitalWrite(LedPin, LOW);
```

Set GPIO0 as 0V (low level). Since the cathode of LED is connected to GPIO0, thus the LED will light up if GPIO0 is set low. On the contrary, set GPIO0 as high level, digitalWrite (LedPin, HIGH): LED will go out.

```
  printf("...LED off\n");
```

The printf function is a standard library function and its function prototype is in the header file "stdio.h". The general form of the call is: printf(" format control string ", output table columns). The format control string is used to specify the output format, which is divided into format string and non-format string. The format string starts with '%' followed by format characters, such as' %d 'for decimal integer output. Unformatted strings are printed as prototypes. What is used here is a non-format string, followed by "\n" that is a newline character, representing automatic line wrapping after printing a string.

```
  delay(1000);
```

Delay (1000) keeps the current HIGH or LOW state for 1s.

This is a function that suspends the program for a period of time. And the speed of the program is determined by our hardware. Here we turn on or off the LED. If there is no delay function, the program will run the whole program very fast and continuously loop. So we need the delay function to help us write and debug the program.

```
  return 0;
```

Usually, it is placed behind the main function, indicating that the function returns 0 on successful execution.

For Python Language Users

Step 2: Go to the folder of the code and run it.

If you use a monitor, you're recommended to take the following steps.

Find 1.Blinking_LED.py and double click it to open. Now you're in the file.

Click Run ->Run Module in the window and the following contents will appear.

To stop it from running, just click the X button on the top right to close it and then you'll back to the code. If you modify the code, before clicking Run Module (F5) you need to save it first. Then you can see the results.

If you log into the Raspberry Pi remotely, type in the command:

```
cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/Python
```

Note: Change directory to the path of the code in this experiment via cd.

Step 3: Run the code

```
sudo python3 1.Blinking_LED.py
```

Note: Here sudo - superuser do, and python means to run the file by Python.

After the code runs, you will see the LED flashing.

Step 4:

If you want to edit the code file 1.Blinking_LED.py , press Ctrl + C to stop running the code.

Then type the following command to open 1.Blinking_LED.py:

```
nano 1.Blinking_LED.py
```

Note: nano is a text editor tool. The command is used to open the code file 1.Blinking_LED.py by this tool.

Press Ctrl+X to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in Y (save) or N (don’t save).

Then press Enter to exit. Type in nano 1.Blinking_LED.py again to see the effect after the change.

Code

The program code is shown as follows:

```
#!/usr/bin/env python3

import RPi.GPIO as GPIO
import time
LedPin = 17
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    # Define a main function for main process
def main():
        while True:
        print ('...LED ON')
        # Turn on LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(1)
        print ('LED OFF...')
        # Turn off LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(1)
# Define a destroy function for clean up everything after the script finished
def destroy():
    # Turn off LED
    GPIO.output(LedPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()
# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

Code Explanation

```
#!/usr/bin/env python3
```

When the system detects this, it will search the installation path of python in the env setting, then call the corresponding interpreter to complete the operation. It's to prevent the user not installing the python onto the /usr/bin default path.

```
import RPi.GPIO as GPIO
```

In this way, import the RPi.GPIO library, then define a variable, GPIO to replace RPI.GPIO in the following code.

Import time package, for time delay function in the following program.

```
import time
```

LED connects to the GPIO17 of the T-shape extension board, namely, BCM 17.

```
LedPin = 17
```

Set LedPin's mode to output, and initial level to High (3.3v).

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
```

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO: BOARD numbers and BCM numbers. In our lessons, what we use is BCM numbers. You need to set up every channel you are using as an input or an output.

```
            GPIO.output(LedPin, GPIO.LOW)
```

Set GPIO17(BCM17) as 0V (low level). Since the cathode of LED is connected to GPIO17, thus the LED will light up.

```
            time.sleep(1)
```

Delay for 1 second. Here, the statement is similar to delay function in C language, the unit is second.

```
def destroy():
        GPIO.cleanup()
```

Define a destroy function for clean up everything after the script finished.

```
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

This is the general running structure of the code. When the program starts to run, it initializes the pin by running the setup(), and then runs the code in the main() function to set the pin to high and low levels. When 'Ctrl+C' is pressed, the program, destroy() will be executed.

Phenomenon Picture