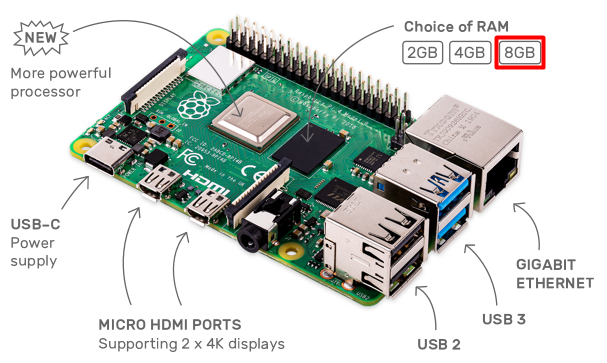
# Objective

Write Python code to control Raspi and light up LED. Furtherly, using PWM to adjust the brightness of LED

1. Learn to control Raspi IO output port；
2. Learn the basic Python programing skill；
3. Use PWM to control LED.

# Components

1. Raspi Board and Power；



1. A 16GB SD Card and a SD Card Reader；

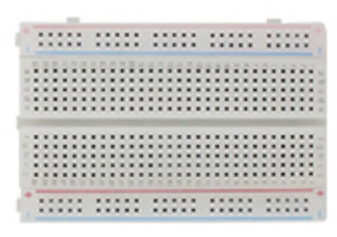


1. HDMI Screen, USB keyboard, USB mouse and HDMI Cable

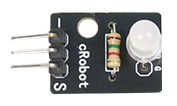




1. Basic Hardware (Adapter board, Wires, Bread Board)



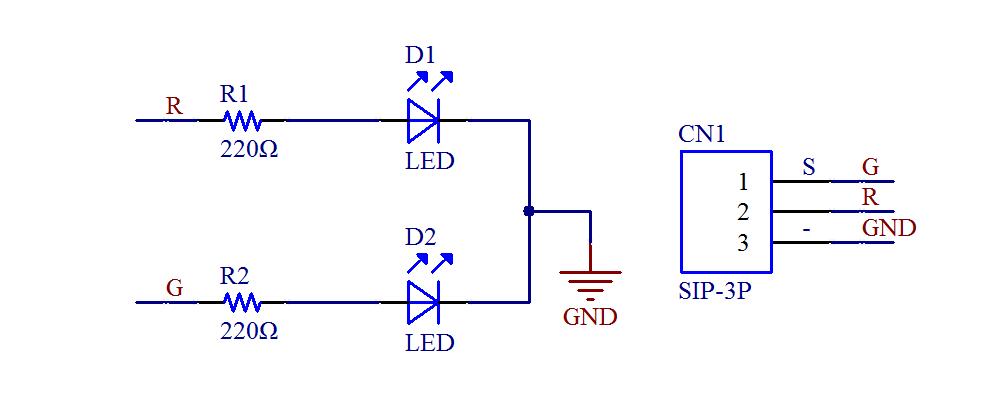
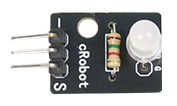
1. Double color LED



# Principle of The Experiment

## Double Color LED

The hardware device and the corresponding schematic of double color LED is shown as below:

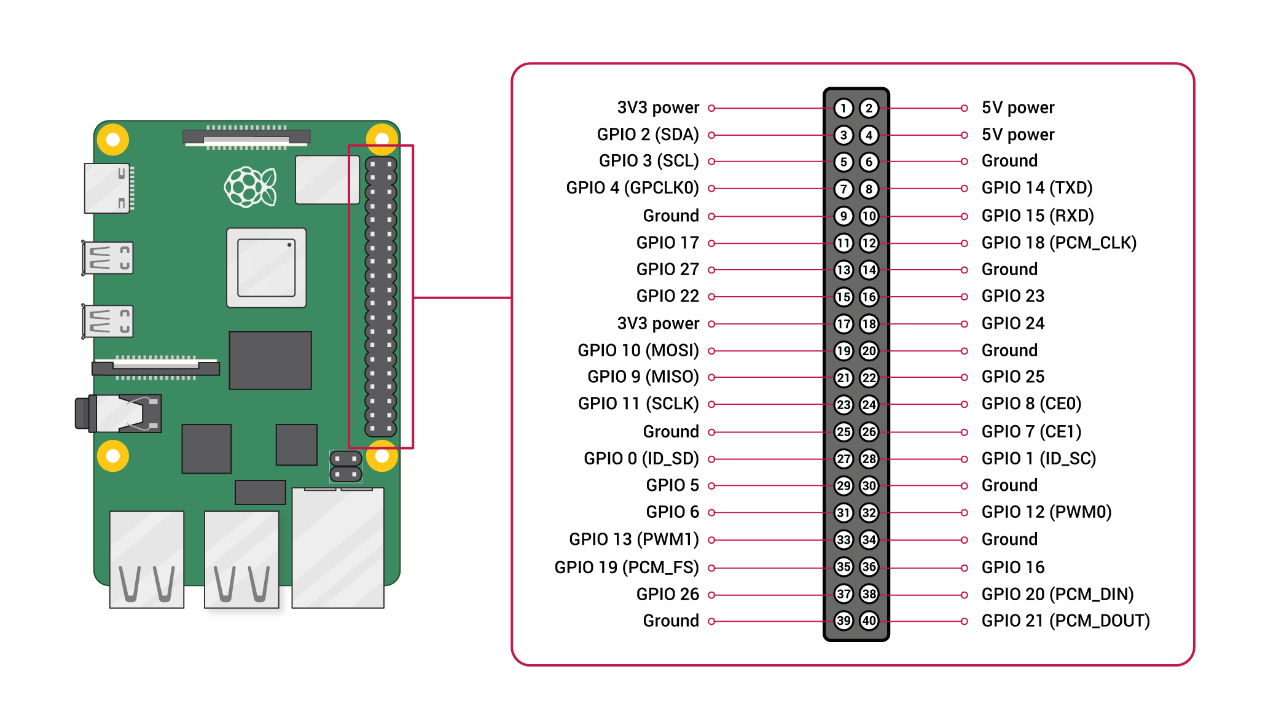


There are two LED luminous source, which emit red and green light, respectively. If Raspi outputs a high level to ‘R’ pin, the red LED on; If Raspi outputs a low level to ‘R’ pin, while a high level to ‘G’ pin, the green LED on. If both ‘R’ and ‘G’ pins are high level, only the red LED on.

## Pin guide

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. The GPIO is used to control LED and adjust the lightness of LED in this experiment.

There are three kinds of pins that are power (VCC), ground (GND) and GPIO. The function of each pin is shown as below:



Two 5V pins and two 3V3 (3.3V) pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The remaining pins are all GPIO pins, which can be designated (in software) as an input or output pin and used for a wide range of purposes.

The numbering of the GPIO pins is not in numerical order.

1. Outputs Pin

A GPIO pin designated as an output pin can be set to high (3V3) or low (0V).

1. Inputs Pin

A GPIO pin designated as an input pin can be read as high (3V3) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins. For example:

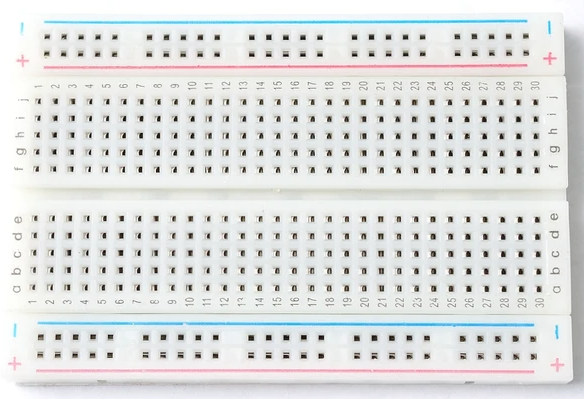
* PWM (pulse-width modulation)

Software PWM available on all pins (For this experiment, we will use software PWM)

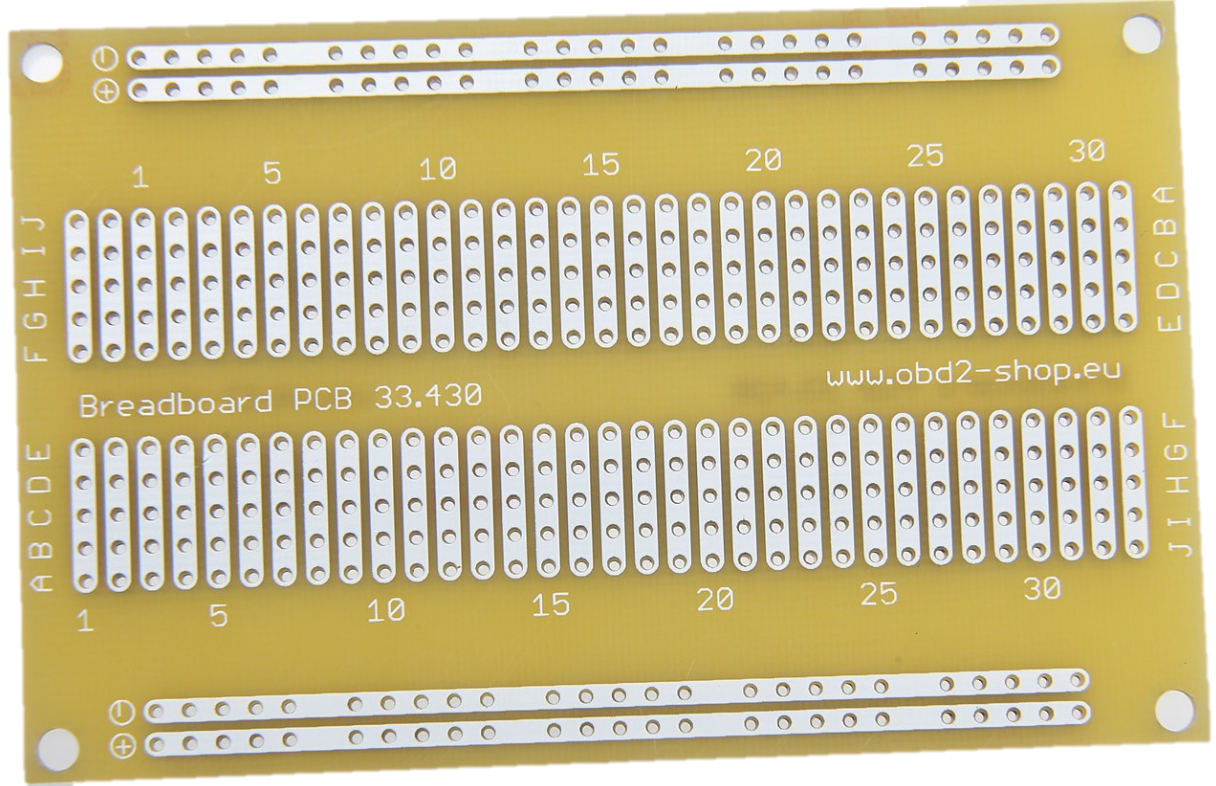
Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19

## Bread Board

Here is the bread board, which is a construction base for prototyping of electronics.

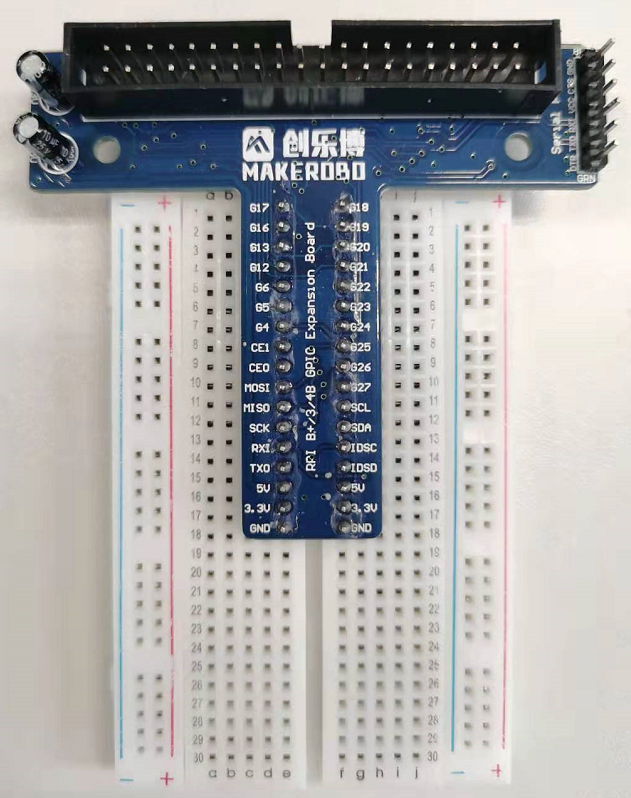


The internal connection is shown as below, which can be used to fan-out the 40 Pins of Raspi:



## Adapter Board

The adapter board enable us connect raspi IO pins with bread board easily. Besides, the adapter board shows the name of each raspi IO pin, which is very convenient for programing.



**The IO name of Raspi, ‘G’ means ‘GPIO’**

## PWM

The pulse-width modulation (PWM) is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.

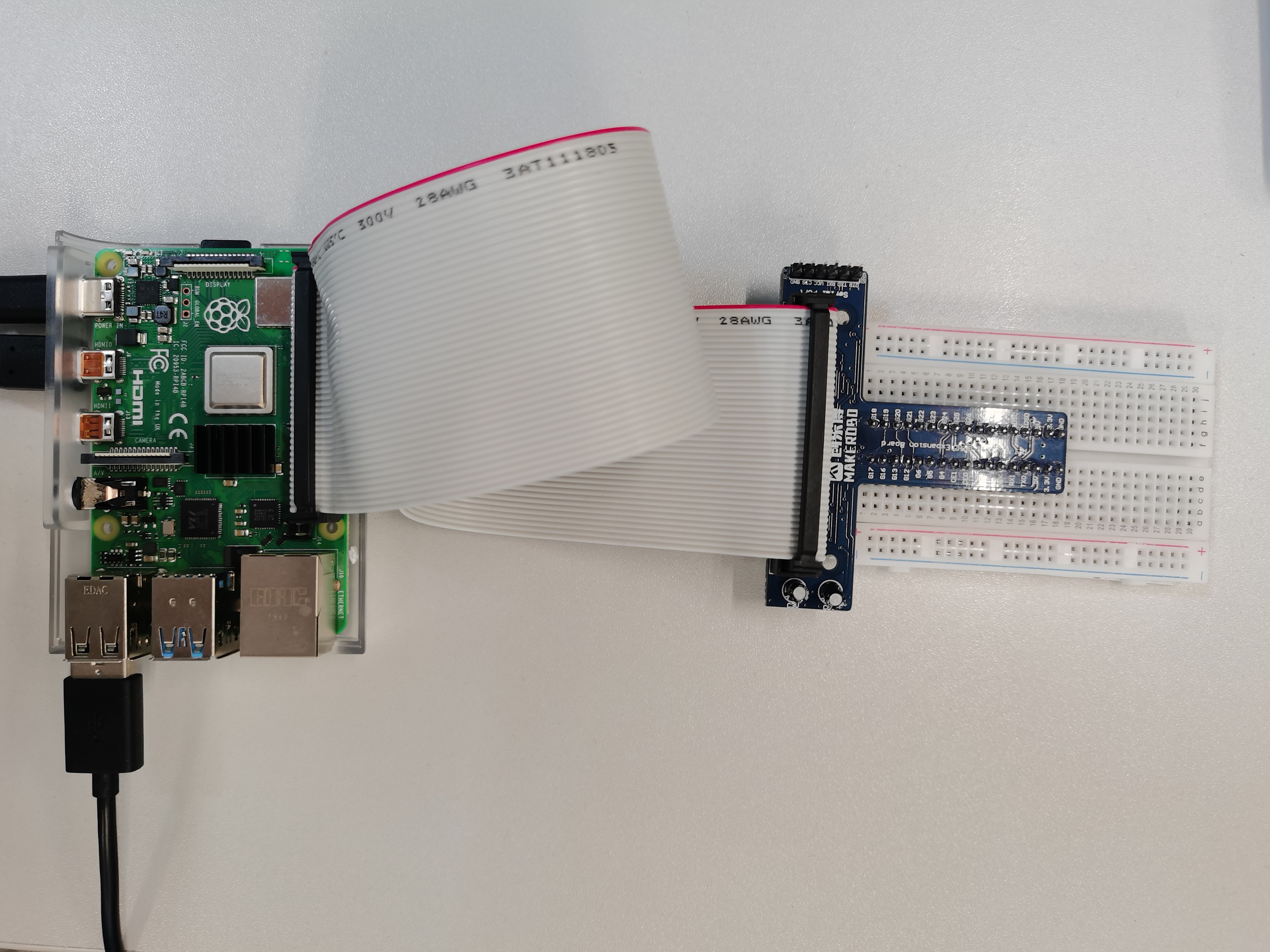
The term duty cycle of PWM describes the proportion of ‘on’ time to the regular interval or ‘period’ of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on. When a digital signal is on half of the time and off the other half of the time, the digital signal has a duty cycle of 50% and resembles a ‘square’ wave. When a digital signal spends more time in the on state than the off state, it has a duty cycle of >50%. When a digital signal spends more time in the off state than the on state, it has a duty cycle of <50%. Here is a pictorial that illustrates these three scenarios:



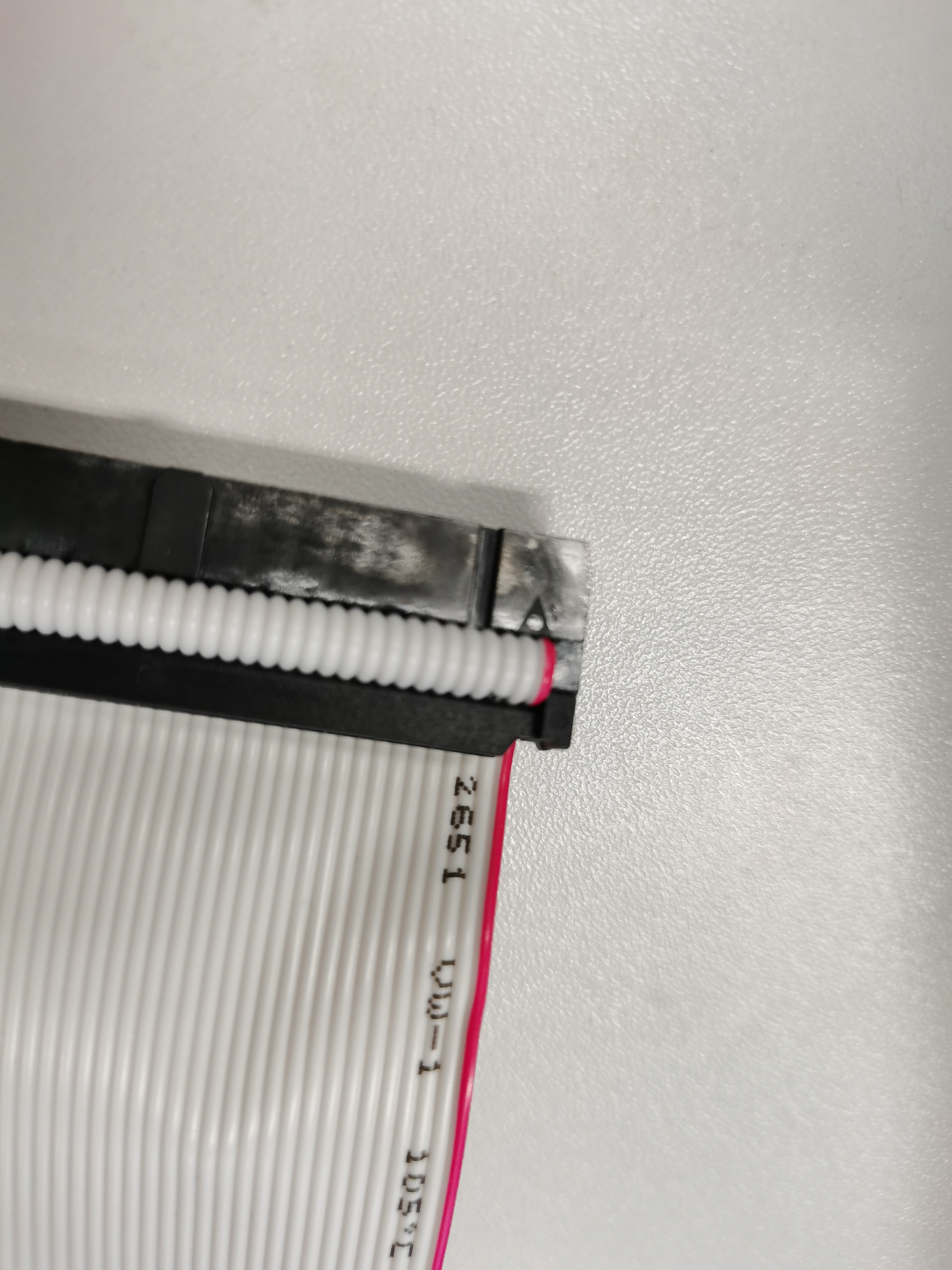
In this experiment, the PWM technology will be used to adjust the brightness of LED.

# Experiment Steps

1. Connect Raspi and Breadboard using the 40-wires cable as below.



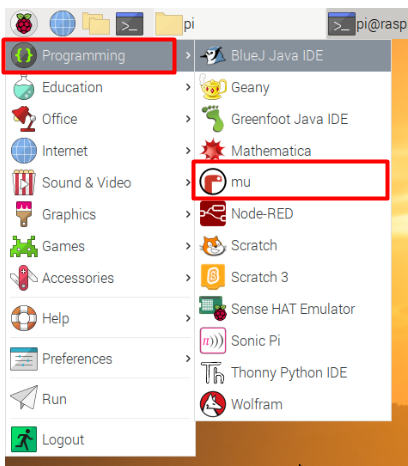
Connect with the first physical Pin of Raspi



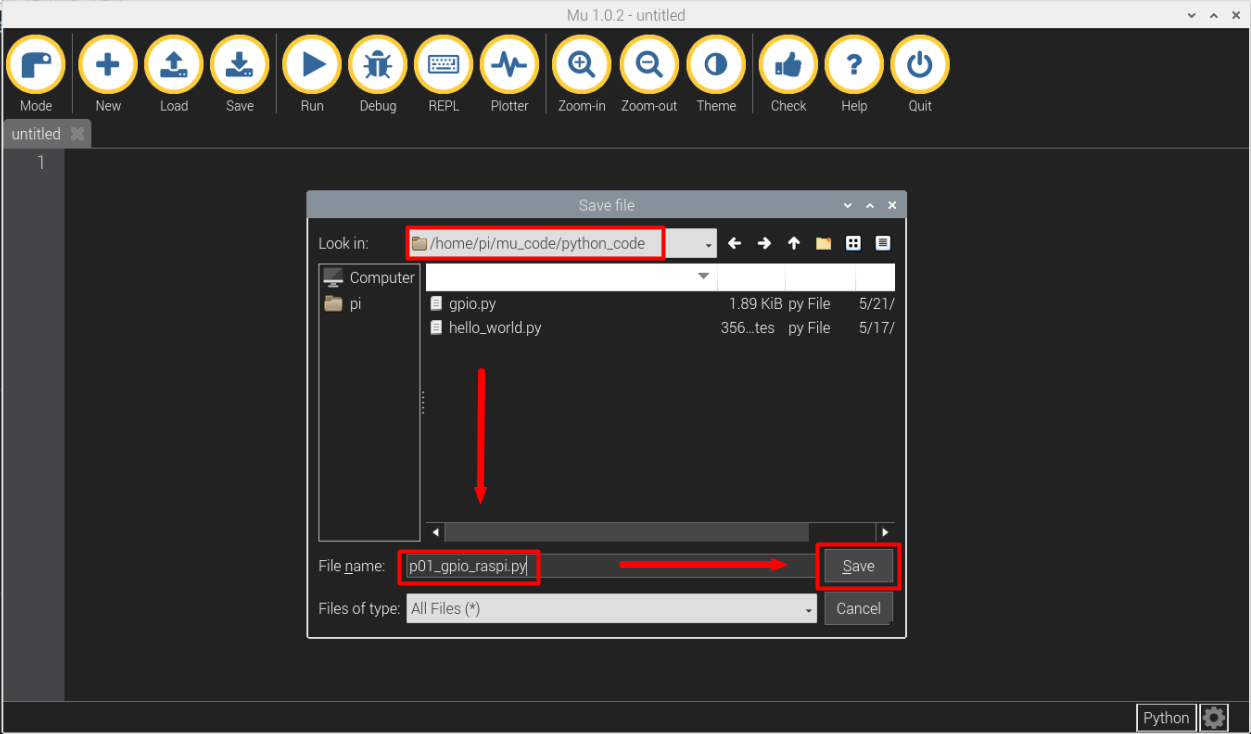
1. As shown below, connect the middle ‘R’ pin of the Double Color LED with GPIO 17 (G17), the ‘G’ pin with GPIO 18 (G18), and the left pin with GND.



1. Power on Raspi and start-up Mu from the Raspi menu: **Menu 🡪 Programming 🡪 mu**, as below



1. Click the New button, then before typing anything, click the Save button and enter the “python\_code” folder, and give your file the name, “p01\_gpio\_raspi.py”, then click Save button.



1. Type the following Python code into the text area to import the useful python libraries:

from gpiozero import \*

from time import sleep

1. Define two variables to correlate GPIO 17 and GPIO 18.

gpio\_red = 17 # define the gpio number controlling red LED

gpio\_gre = 18 # define the gpio number controlling green LED

1. Use class LED() to initialize the red LED and green LED.

led\_red = LED(gpio\_red, active\_high=True, initial\_value=False)

led\_gre = LED(gpio\_gre, active\_high=True, initial\_value=False)

From the schematic of double colour LED shown in 3.2. , we can get that LED is turned on when GPIO output high voltage, so we set active\_high=True. We set initial\_value=False, as we hope the initial state of LED is off. For more information about LED() class, please visit [*this site*](https://gpiozero.readthedocs.io/en/stable/api_output.html#led).

1. Type the following code to turn on and turn off LEDs.

led\_red.on()

sleep(0.5) # delay 0.5 second

led\_red.off()

sleep(0.5)

led\_gre.on()

sleep(0.5)

led\_gre.off()

sleep(0.5)

led\_red.close()

led\_gre.close()

1. Enter the Debug mode by clicking the Debug button to test our code. And click Step Over button and see the changes of the “debug inspector” area on the right of the window. When led\_red.on() and led\_red.off() are executed, we can observe if the red LED can be turned on and off, respectively. When debug is finished, don’t forget to exist debug mode by clicking Stop button.

If LED can’t be turned on, we should check if the hardware connection is well.

1. As LED is derived by GPIO, so we can use class DigitalOutputDevice() to realize the same function. We can type the following code and enter Debug mode to verify it.

# using gpio class

led\_red = DigitalOutputDevice(gpio\_red, active\_high=True, initial\_value=False)

led\_gre = DigitalOutputDevice(gpio\_gre, active\_high=True, initial\_value=False)

led\_red.on()

sleep(0.5)

led\_red.off()

sleep(0.5)

led\_gre.on()

sleep(0.5)

led\_gre.off()

sleep(0.5)

led\_red.close()

led\_gre.close()

1. Use PWM class, PWMOutputDevice(), to adjust the light of LED.

# using PWM adjust the light

PWM\_led\_red = PWMOutputDevice(gpio\_red, initial\_value=0, frequency = 100)

PWM\_led\_gre = PWMOutputDevice(gpio\_gre, initial\_value=0, frequency = 10)

This two columns are used to initialize GPIO 17 and 18 into PWM mode, and the initial duty cycle is 0%. The initial PWM frequencies are 100Hz and 10Hz, respectively. For more information about PWMOutputDevice() class, please visit [this site](https://gpiozero.readthedocs.io/en/stable/api_output.html#pwmoutputdevice).

1. Type the following code to adjust the lightness of LED.

PWM\_led\_red.value = 1.0

sleep(1)

PWM\_led\_red.value = 0.2

sleep(1)

PWM\_led\_red.off()

PWM\_led\_gre.value = 1.0

sleep(1)

PWM\_led\_gre.value = 0.2

sleep(1)

PWM\_led\_gre.off()

The value variable in PWM class defines the duty cycle of the PWM device. And 0.0 is off (0%), 1.0 is fully on (100%).

1. Click the column number of PWM\_led\_red.value = 1.0 to insert a break print, and a red round will appear on the left of the column as below. Click the Debug button, and the program will run to this column directly after a few seconds. 
2. Click Step Over button and observe the lightness of red LED with 100Hz PWM frequencies and two different duty cycles (100% and 20%).
3. Continue to Step Over the program and observe the lightness of green LED with 10Hz PWM frequencies and two different duty cycles (100% and 20%).

For the red LED, the lightness should be reduced when the duty cycle of PWM reduce to 20% from 100%. While the lightness of green LED would not reduce but flash quickly with 20% duty cycle. Explain why?

1. Stop debug mode, and type following code to set PWM frequency to 100Hz for both red and green LED. And insert a break point at the column PWM\_led\_gre.frequency = 100

PWM\_led\_red.frequency = 100

PWM\_led\_gre.frequency = 100

1. Type following code to gradually increase the duty cycle of PWM. And Debug the program. When the program run to the first break point, we click Continue button and the program will run to the second break point inserted in step 16) directly. Then we click Continue button again and observe LED at the same time.

for i in range(1000): # i start from 0 and increase to 999

PWM\_led\_gre.value = i / 1000

sleep(0.001)

1. Enter the following code, and insert a break point at the column PWM\_led\_gre.off(). Then Debug the program similar with step 17) and observe LED.

If your code is right, the LED should fade in and out just like breathing. Please analyze the program and explain why?

PWM\_led\_gre.off()

for j in range(5):

for i in range(1000):

PWM\_led\_red.value = i / 1000

sleep(0.001)

for i in range(1000):

PWM\_led\_red.value = (999-i) / 1000

sleep(0.001)

for j in range(5):

for i in range(1000):

PWM\_led\_gre.value = i / 1000

sleep(0.0002)

for i in range(1000):

PWM\_led\_gre.value = (999-i) / 1000

sleep(0.0002)

For more knowledge about Pyhton **gpiozero** library, please click [*here*](https://gpiozero.readthedocs.io/en/stable/index.html).

# **appendix**

Here are all the code:

from gpiozero import \*

from time import sleep

gpio\_red = 17 # define the gpio number controlling red LED

gpio\_gre = 18 # define the gpio number controlling green LED

led\_red = LED(gpio\_red, active\_high=True, initial\_value=False)

led\_gre = LED(gpio\_gre, active\_high=True, initial\_value=False)

led\_red.on()

sleep(0.5)

led\_red.off()

sleep(0.5)

led\_gre.on()

sleep(0.5)

led\_gre.off()

sleep(0.5)

led\_red.close()

led\_gre.close()

# using gpio function

led\_red = DigitalOutputDevice(gpio\_red, active\_high=True, initial\_value=False)

led\_gre = DigitalOutputDevice(gpio\_gre, active\_high=True, initial\_value=False)

led\_red.on()

sleep(0.5)

led\_red.off()

sleep(0.5)

led\_gre.on()

sleep(0.5)

led\_gre.off()

sleep(0.5)

led\_red.close()

led\_gre.close()

# using PWM adjust lightness

PWM\_led\_red = PWMOutputDevice(gpio\_red, frequency = 100)

PWM\_led\_gre = PWMOutputDevice(gpio\_gre, frequency = 10)

PWM\_led\_red.value = 1.0

sleep(1)

PWM\_led\_red.value = 0.2

sleep(1)

PWM\_led\_red.off()

PWM\_led\_gre.value = 1.0

sleep(1)

PWM\_led\_gre.value = 0.2

sleep(1)

PWM\_led\_gre.off()

PWM\_led\_red.frequency = 100

PWM\_led\_gre.frequency = 100

for i in range(1000): % i start from 0 and increase to 999

PWM\_led\_red.value = i / 1000

sleep(0.001)

PWM\_led\_red.off()

for j in range(5):

for i in range(1000):

PWM\_led\_red.value = i / 1000

sleep(0.001)

for i in range(1000):

PWM\_led\_red.value = (999-i) / 1000

sleep(0.001)

for j in range(5):

for i in range(1000):

PWM\_led\_gre.value = i / 1000

sleep(0.0002)

for i in range(1000):

PWM\_led\_gre.value = (999-i) / 1000

sleep(0.0002)

**End of Tutorial 2**