
RGB Matrix HAT for Raspberry Pi

Release 1.0

SunFounder

Jul 04, 2022

CONTENTS

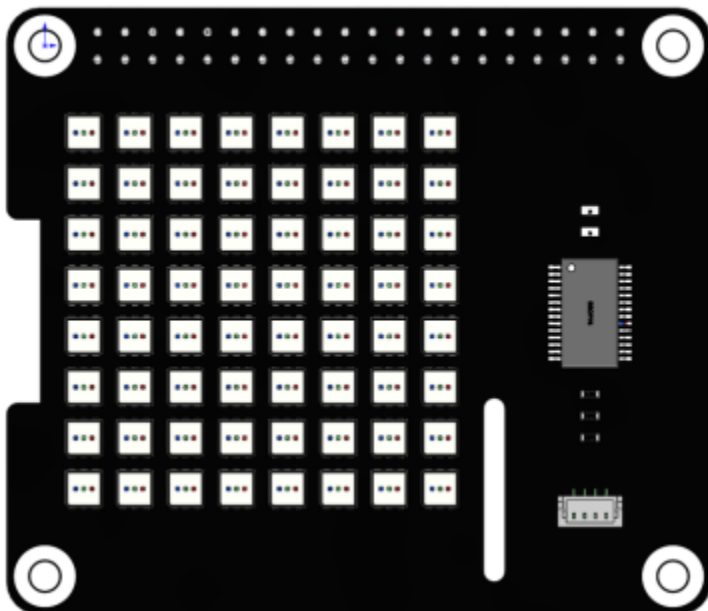
1	Features	3
2	Assemble RGB Matrix HAT	5
3	Preparation	7
3.1	What Do We Need?	7
3.1.1	Required Components	7
3.1.2	Optional Components	8
3.2	Installing the OS	9
3.3	Set up Your Raspberry Pi	15
3.3.1	If You Have a Screen	15
3.3.2	If You Have No Screen	16
4	Projects	23
4.1	Hello Matrix	23
4.2	Dazzling Light	27
4.3	Moving Eye	31
4.4	Christmas Tree	35
4.5	Greedy Snake	38
4.6	Camera Recognition	41
4.7	Custom Shape	43
4.8	Custom Dynamic Shape	46
5	Appendix	51
5.1	I2C Configuration	51
5.2	Remote Desktop	53
5.2.1	VNC	53
5.2.2	XRDP	60
6	Copyright Notice	65

Welcome to use SunFounder RGB Matrix module. You can find the information you need for use here.

This is a module with 8×8 RGB LEDs on board. It also has a SH1106 I2C control interface, which is convenient to connect to other I2C devices or other single-chip microcomputers.

Here is the Email: cs@sunfounder.com.

FEATURES

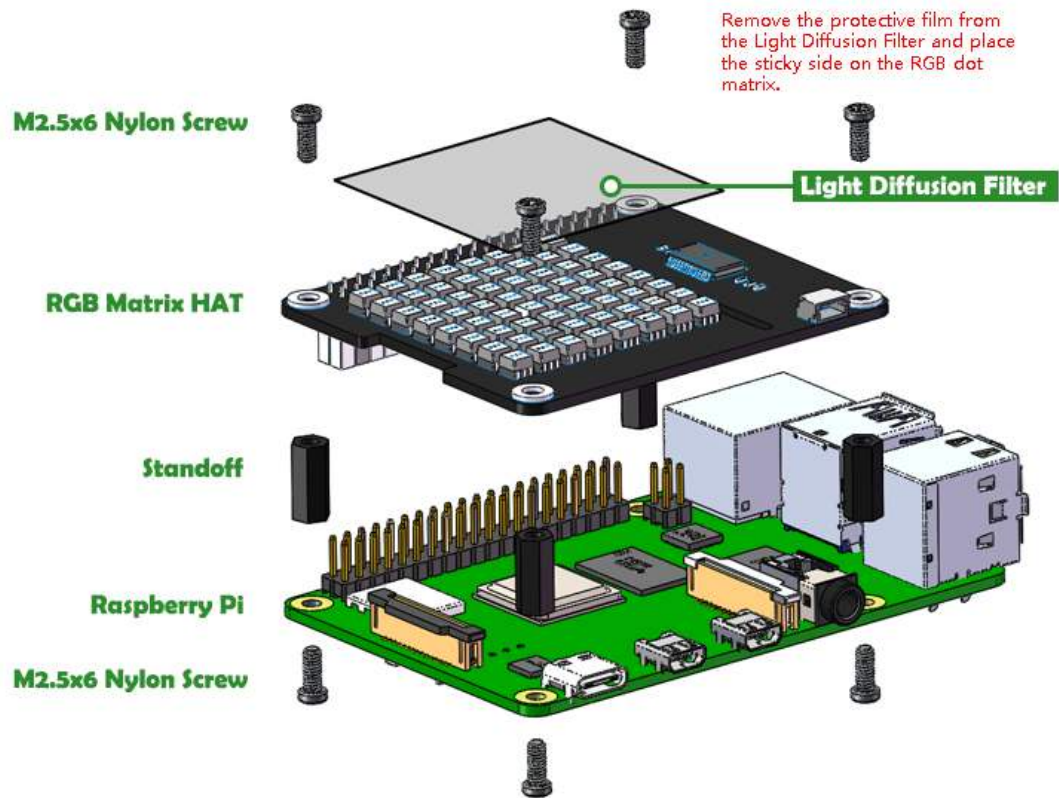


- Working voltage: DC 3.3V
- Lamp bead: FM-N3535RGBW-SH
- Driver: SLED 1734X LED driver
- Communication method: I2C
- Color depth: 24 bit (R/G/B each 8 bit color, $256 \times 256 \times 256 = 16777216$ colors can be combined)
- Resolution: $8 \times 8 = 64$ DOTS
- Pixel pitch: 4.7mm
- matrix size: 36.5mm*36.5mm

Documentation

- PCB
- Schematic
- Datasheet

ASSEMBLE RGB MATRIX HAT



PREPARATION

In this chapter, we firstly learn to start up Raspberry Pi. The content includes installing the OS, Raspberry Pi network and how to open terminal.

Note: You can check the complete tutorial on the official website of the Raspberry Pi: [raspberrypi-setting-up](https://www.raspberrypi.org/documentation/hardware/raspberrypi/raspbian/setting-up.md).
If your Raspberry Pi is set up, you can skip the part and go into the next chapter.

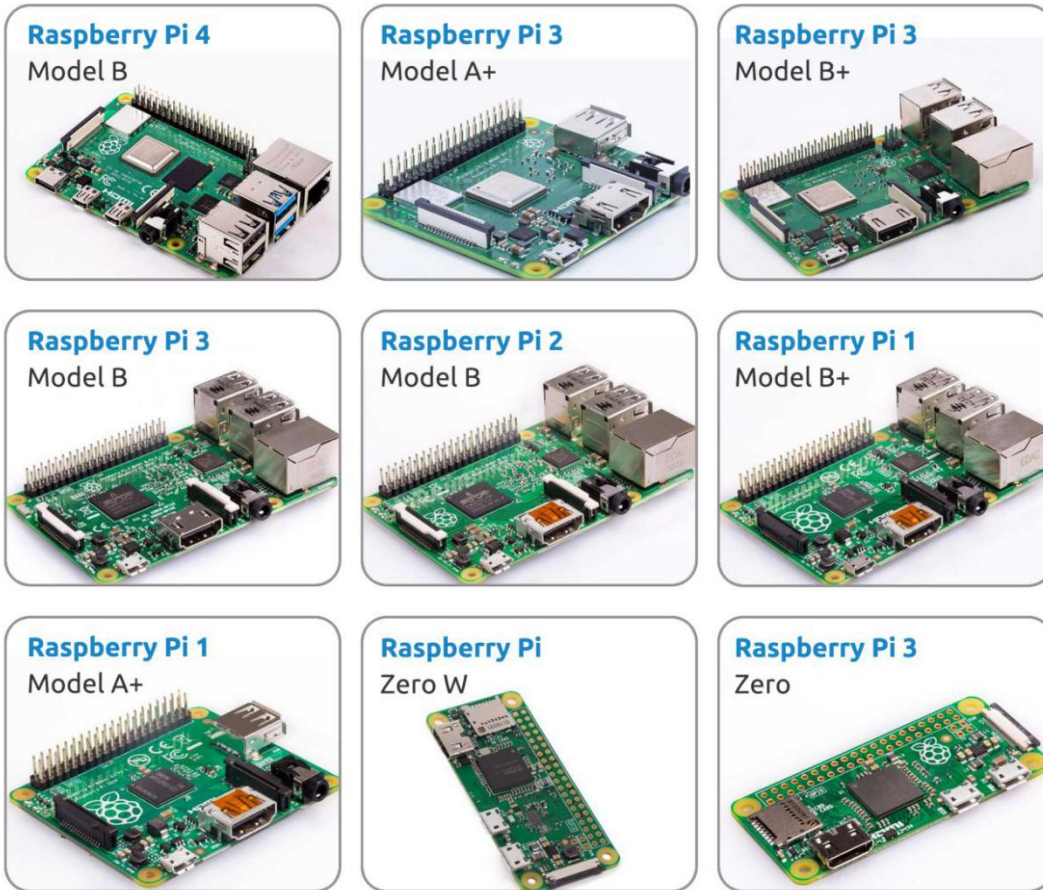
3.1 What Do We Need?

3.1.1 Required Components

Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

Our kit applies to the following versions of the product of Raspberry Pi



Power Adapter

To connect to a power socket, the Raspberry Pi has a micro USB port (the same found on many mobile phones). You will need a power supply which provides at least 2.5 amps.

Micro SD Card

Your Raspberry Pi needs an Micro SD card to store all its files and the Raspberry Pi OS. You will need a micro SD card with a capacity of at least 8 GB

3.1.2 Optional Components

Screen

To view the desktop environment of Raspberry Pi, you need to use the screen that can be a TV screen or a computer monitor. If the screen has built-in speakers, the Pi plays sounds via them.

Mouse & Keyboard

When you use a screen , a USB keyboard and a USB mouse are also needed.

HDMI

The Raspberry Pi has a HDMI output port that is compatible with the HDMI ports of most modern TV and computer monitors. If your screen has only DVI or VGA ports, you will need to use the appropriate conversion line.

Case

You can put the Raspberry Pi in a case; by this means, you can protect your device.

Sound or Earphone

The Raspberry Pi is equipped with an audio port about 3.5 mm that can be used when your screen has no built-in speakers or when there is no screen operation.

3.2 Installing the OS

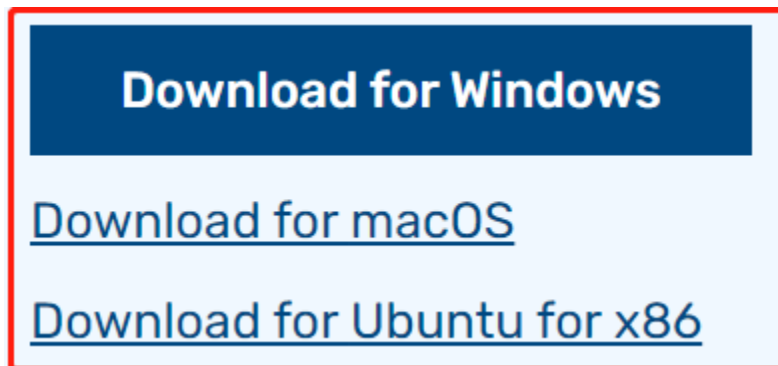
Required Components

Any Raspberry Pi	1 * Personal Computer
1 * Micro SD card	

Step 1

Raspberry Pi have developed a graphical SD card writing tool that works on Mac OS, Ubuntu 18.04 and Windows, and is the easiest option for most users as it will download the image and install it automatically to the SD card.

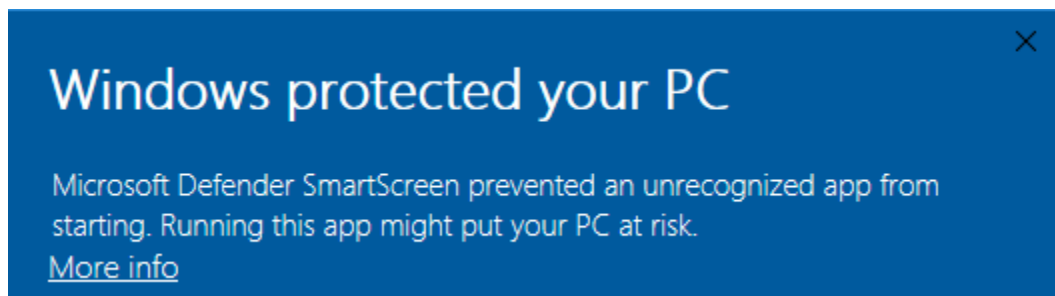
Visit the download page: <https://www.raspberrypi.org/software/>. Click on the link for the Raspberry Pi Imager that matches your operating system, when the download finishes, click it to launch the installer.



Step 2

When you launch the installer, your operating system may try to block you from running it. For example, on Windows I receive the following message:

If this pops up, click on **More info** and then **Run anyway**, then follow the instructions to install the Raspberry Pi Imager.

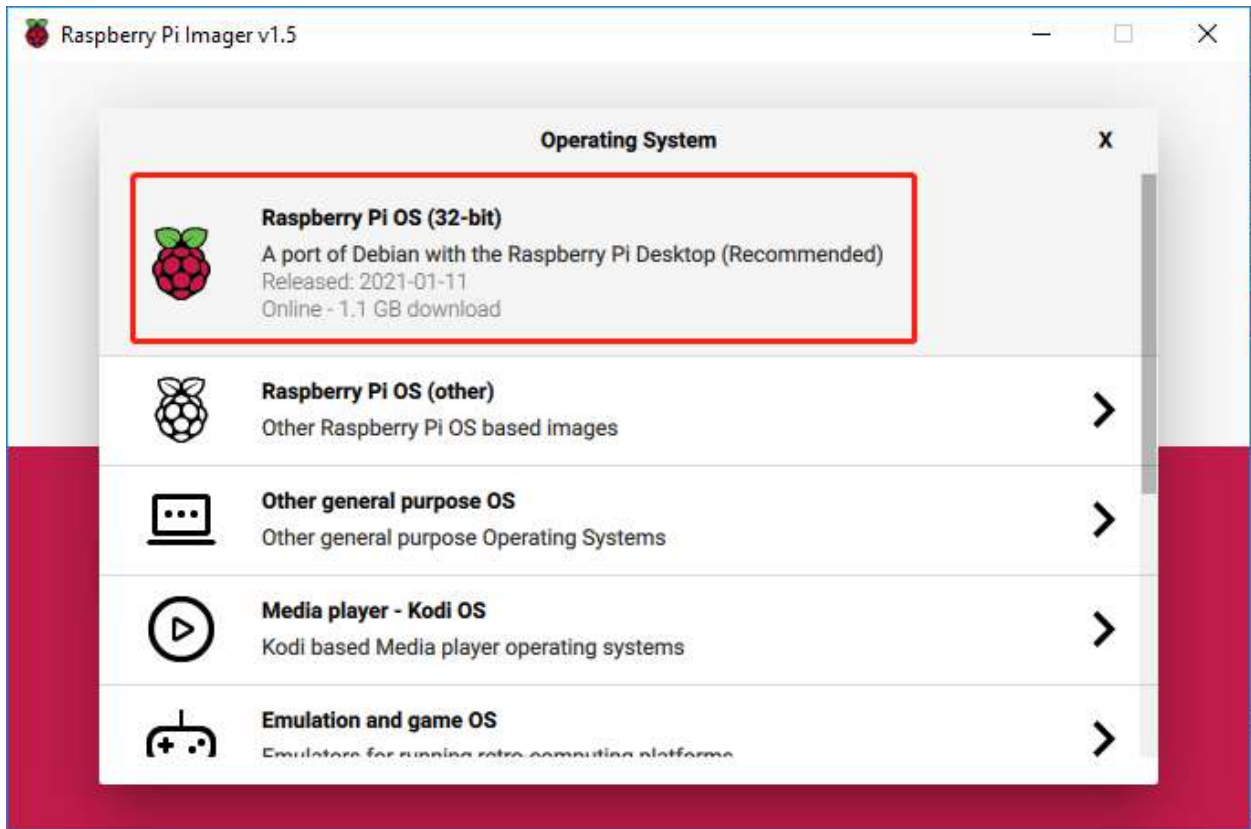


Step 3

Insert your SD card into the computer or laptop SD card slot.

Step 4

In the Raspberry Pi Imager, select the OS that you want to install and the SD card you would like to install it on.

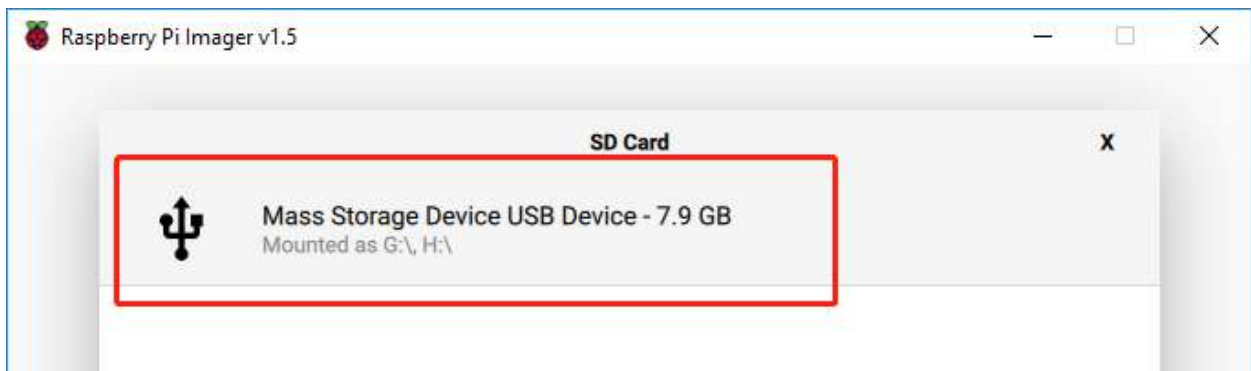


Note:

- 1) You will need to be connected to the internet the first time.
- 2) That OS will then be stored for future offline use (lastdownload.cache, C:/Users/yourname/AppData/Local/Raspberry Pi/Imager/cache). So the next time you open the software, it will have the display "Released: date, cached on your computer".

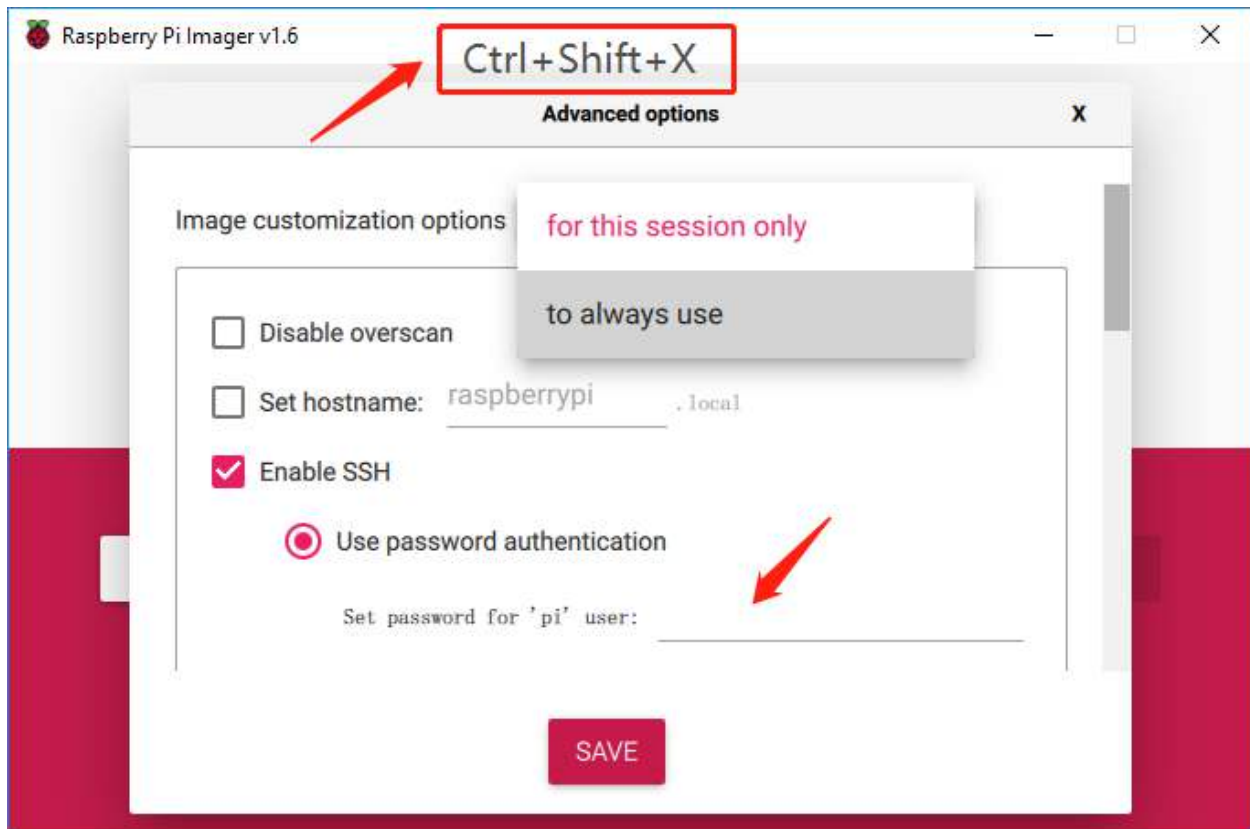
Step 5

Select the SD card you are using.



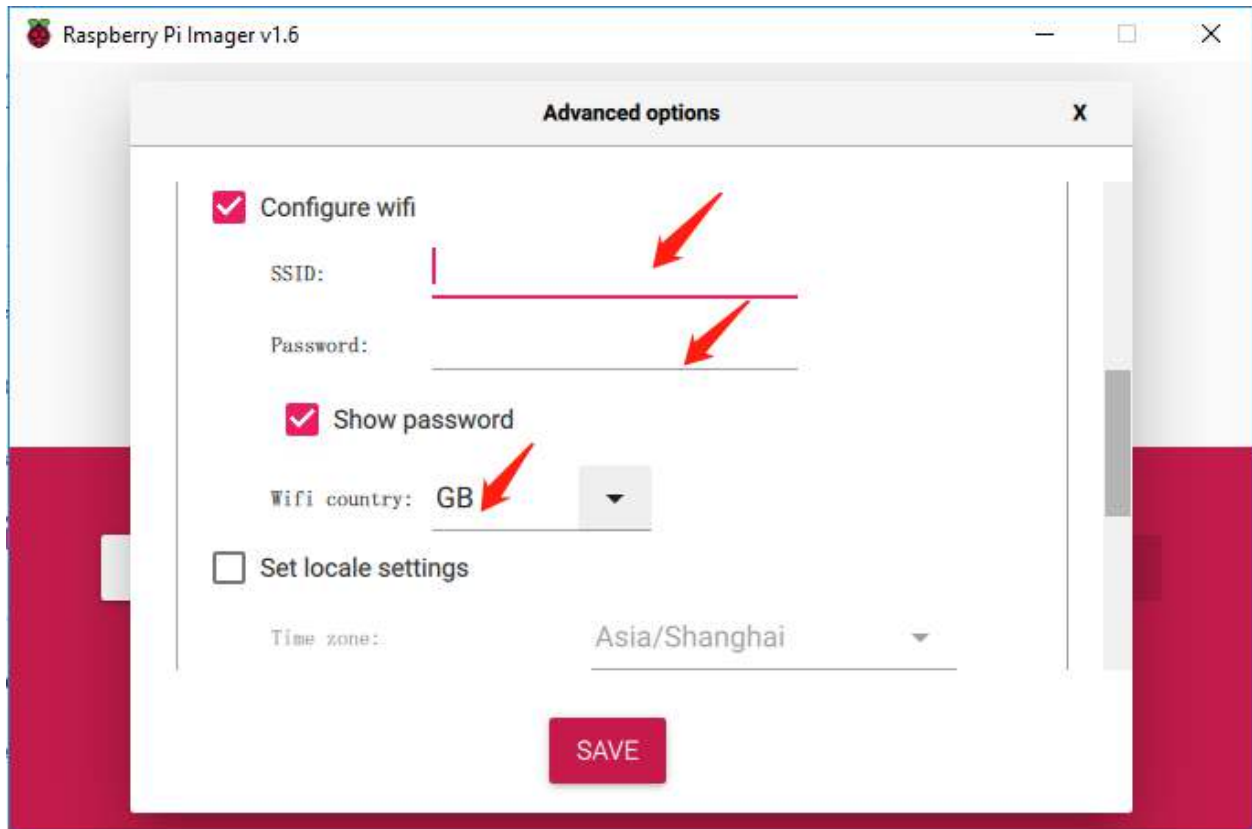
Step 6

Press **Ctrl+Shift+X** to open the **Advanced options** page to enable SSH and configure wifi, these 2 items must be set, the others depend on your choice . You can choose to always use this image customization options.



Then scroll down to complete the wifi configuration and click **SAVE**.

Note: **wifi country** should be set the two-letter [ISO/IEC alpha2 code](https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements) for the country in which you are using your Raspberry Pi, please refer to the following link: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements



Step 7

Click the **WRITE** button.



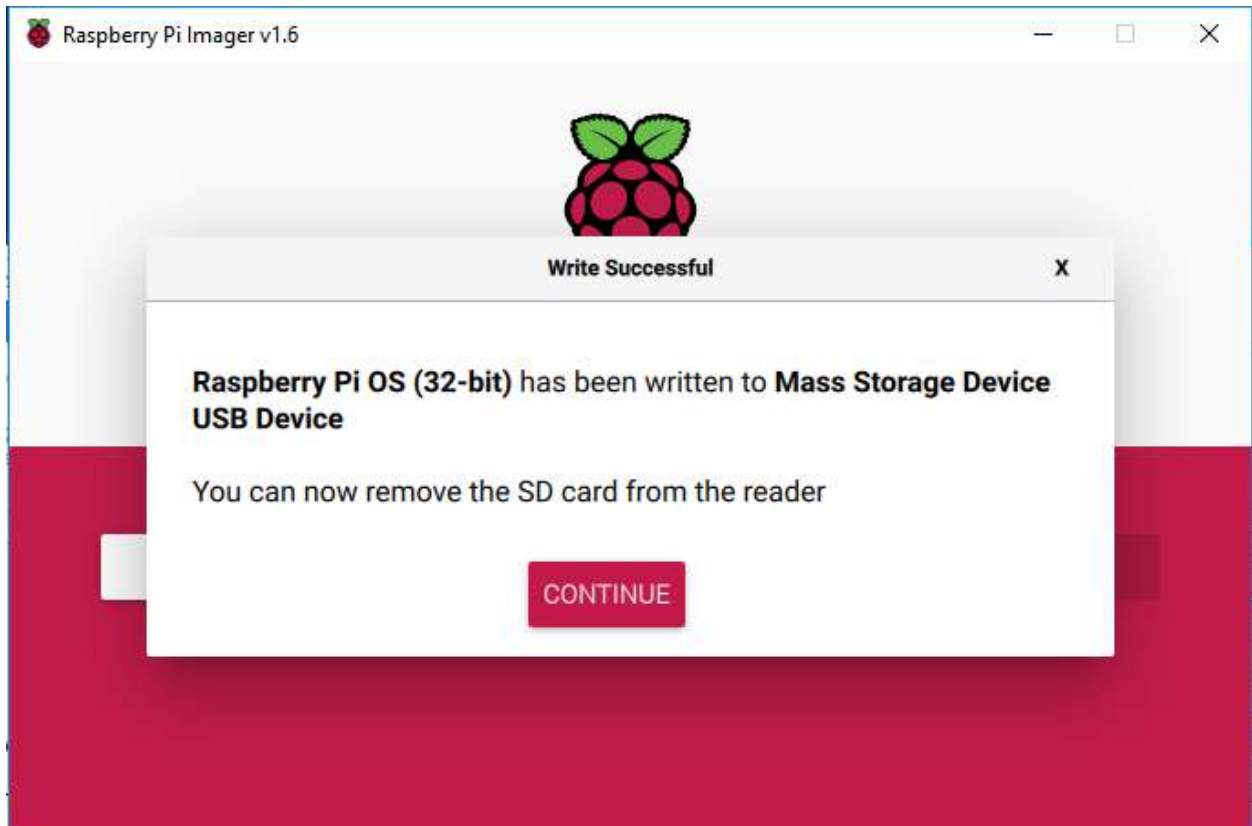
Step 8

If your SD card currently has any files on it, you may wish to back up these files first to prevent you from permanently losing them. If there is no file to be backed up, click **Yes**.



Step 9

After waiting for a period of time, the following window will appear to represent the completion of writing.



3.3 Set up Your Raspberry Pi

3.3.1 If You Have a Screen

If you have a screen, it will be easy for you to operate on the Raspberry Pi.

Required Components

Any Raspberry Pi	1 * Power Adapter
1 * Micro SD card	1 * Screen Power Adapter
1 * HDMI cable	1 * Screen
1 * Mouse	1 * Keyboard

1. Insert the SD card you've set up with Raspberry Pi OS into the micro SD card slot on the underside of your Raspberry Pi.
2. Plug in the Mouse and Keyboard.
3. Connect the screen to Raspberry Pi's HDMI port and make sure your screen is plugged into a wall socket and switched on.

Note: If you use a Raspberry Pi 4, you need to connect the screen to the HDMI0 (nearest the power in port).

4. Use the power adapter to power the Raspberry Pi. After a few seconds, the Raspberry Pi OS desktop will be displayed.



3.3.2 If You Have No Screen

If you don't have a display, you can log in to the Raspberry Pi remotely, but before that, you need to get the IP of the Raspberry Pi.

Get the IP Address

After the Raspberry Pi is connected to WIFI, we need to get the IP address of it. There are many ways to know the IP address, and two of them are listed as follows.

1. Checking via the router

If you have permission to log in the router(such as a home network), you can check the addresses assigned to Raspberry Pi on the admin interface of router.

The default hostname of the Raspberry Pi OS is **raspberrypi**, and you need to find it. (If you are using ArchLinuxARM system, please find alarmpi.)

2. Network Segment Scanning

You can also use network scanning to look up the IP address of Raspberry Pi. You can apply the software, **Advanced IP scanner** and so on.

Scan the IP range set, and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspberry Pi OS is **raspberrypi**, if you haven't modified it.

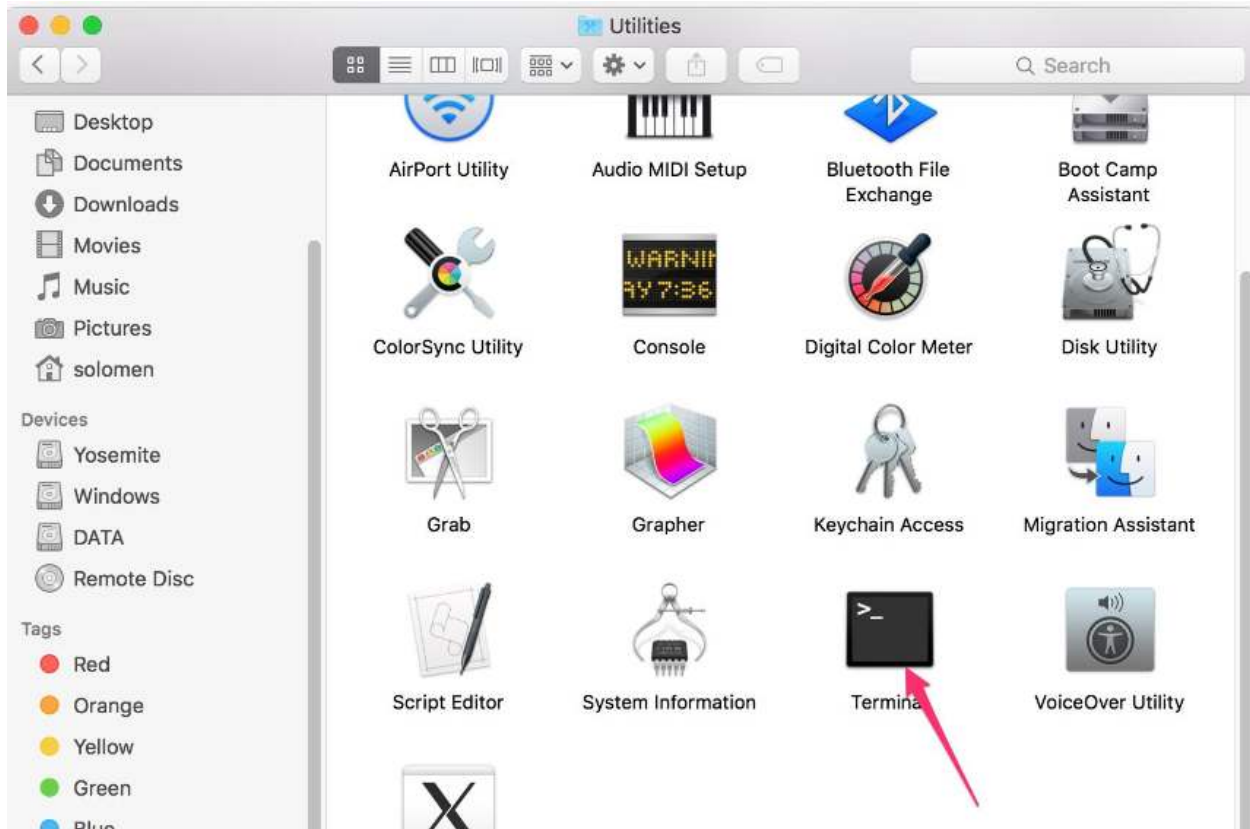
Use the SSH Remote Control

We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The Shell itself is a program written in C that is the bridge linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

For Linux or/Mac OS X Users

Step 1

Go to **Applications->Utilities**, find the **Terminal**, and open it.



Step 2

Type in `ssh pi@ip_address` . “pi”is your username and “ip_address” is your IP address. For example:

```
ssh pi@192.168.18.197
```

Step 3

Input “yes”.

```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000
# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

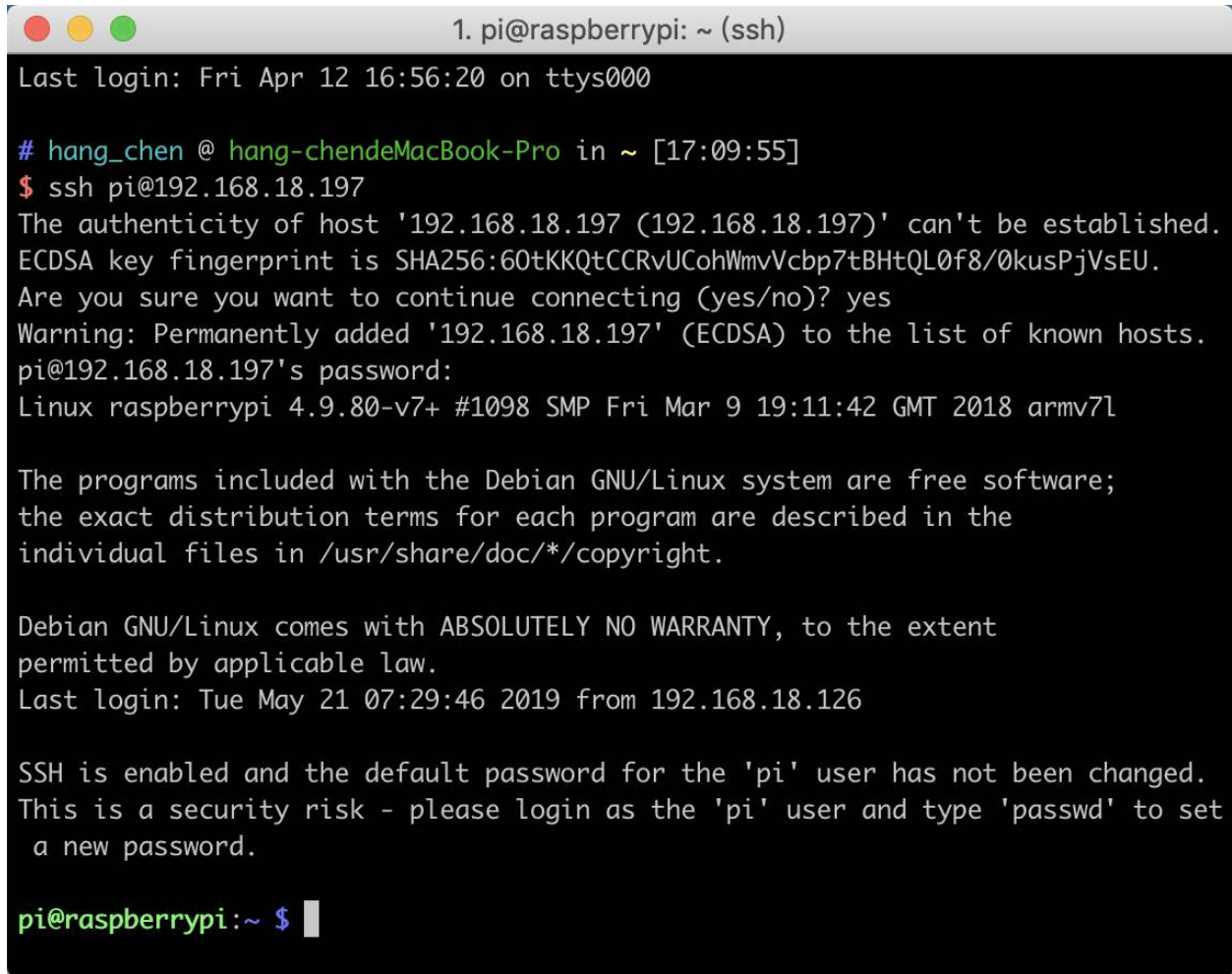
Step 4

Input the passcode and the default password is **raspberry**.

```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000
# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password: ?
```

Step 5

We now get the Raspberry Pi connected and are ready to go to the next step.

A terminal window titled '1. pi@raspberrypi: ~ (ssh)'. The window shows the output of an SSH connection. It starts with 'Last login: Fri Apr 12 16:56:20 on ttys000'. A user named 'hang_chen' connects from a 'hang-chendeMacBook-Pro'. The terminal shows the command '\$ ssh pi@192.168.18.197' and the resulting SSH handshake messages, including the host's ECDSA key fingerprint and a warning that the host has been added to the known hosts list. The terminal then shows the Raspberry Pi's login banner, which includes the system version 'Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l' and a warning about the default password. The prompt 'pi@raspberrypi:~ \$' is visible at the bottom.

```
1. pi@raspberrypi: ~ (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000
# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 21 07:29:46 2019 from 192.168.18.126

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $
```

Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

For Windows Users

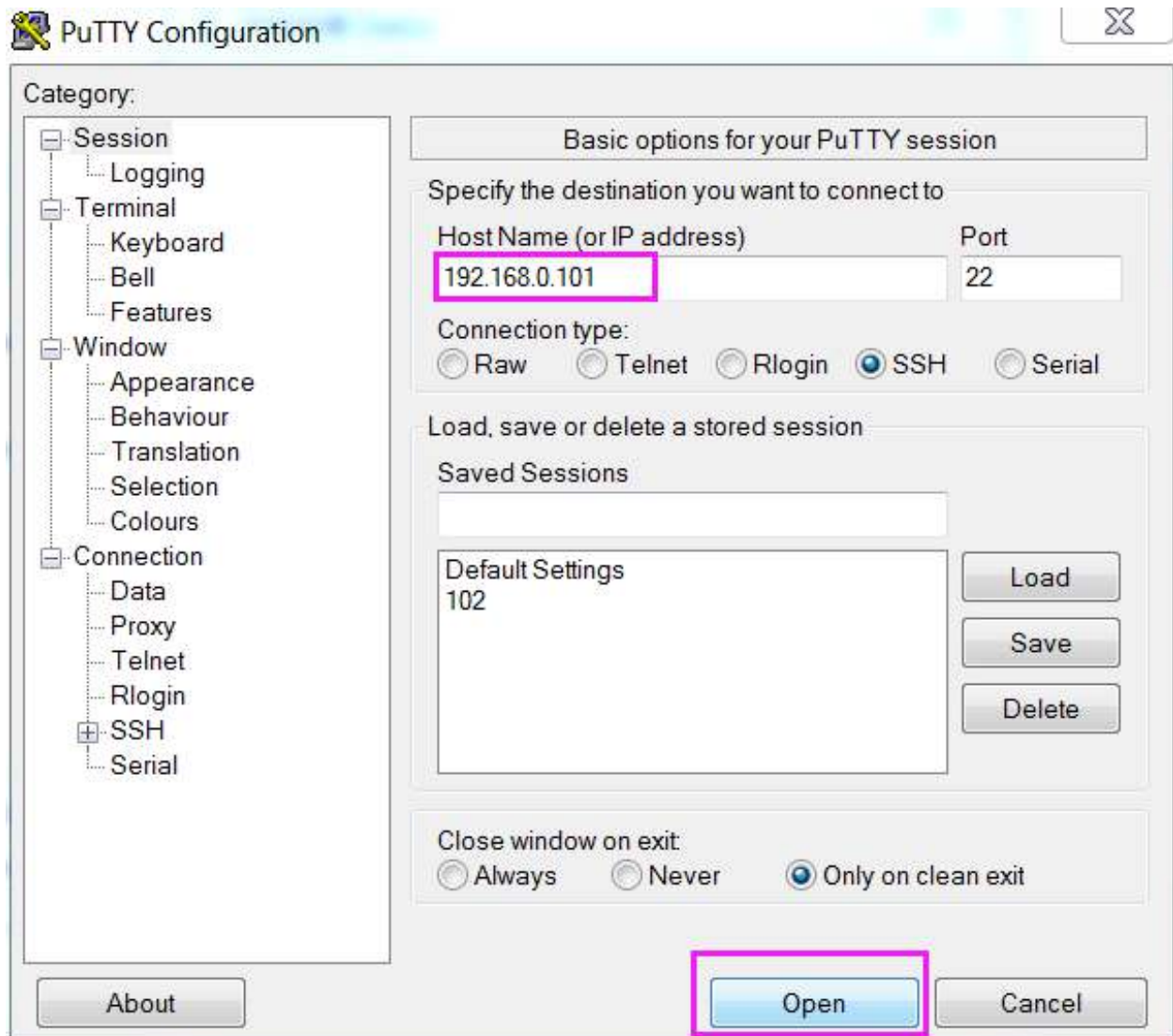
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

Step 1

Download PuTTY.

Step 2

Open PuTTY and click **Session** on the left tree-alike structure. Enter the IP address of the RPi in the text box under **Host Name (or IP address)** and **22** under **Port** (by default it is 22).



Step 3

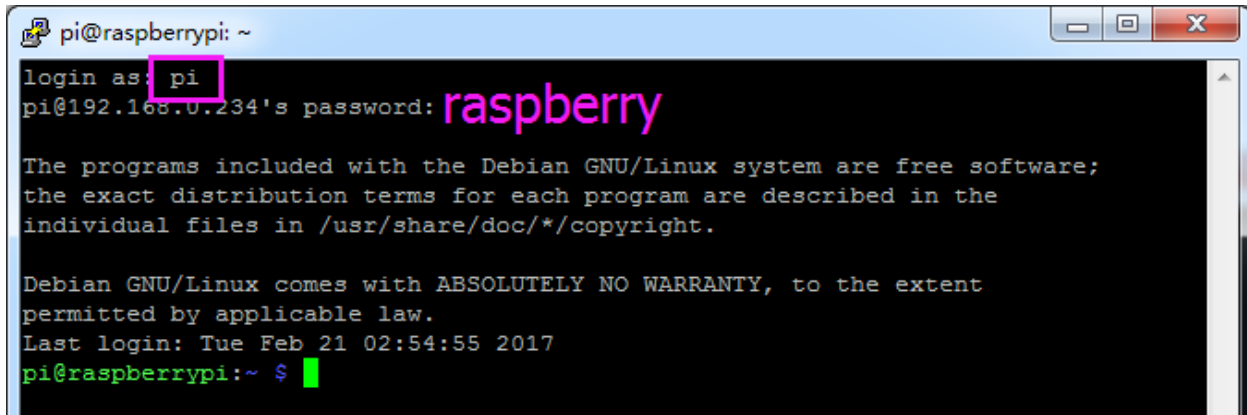
Click **Open**. Note that when you first log in to the Raspberry Pi with the IP address, there prompts a security reminder. Just click **Yes**.

Step 4

When the PuTTY window prompts “**login as:**”, type in “**pi**”(the user name of the RPi), and **password:** “**raspberry**” (the default one, if you haven’t changed it).

Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

If inactive appears next to PuTTY, it means that the connection has been broken and needs to be reconnected.

A terminal window titled 'pi@raspberrypi: ~' with standard window controls. The text inside shows a login sequence: 'login as: pi' (where 'pi' is highlighted with a pink box), followed by 'pi@192.168.0.234's password: raspberry' (where 'raspberry' is in pink). Below this is a copyright notice for Debian GNU/Linux, a warranty disclaimer, and the last login time: 'Last login: Tue Feb 21 02:54:55 2017'. The prompt 'pi@raspberrypi:~ \$' is shown at the bottom with a green cursor.

```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.234's password: raspberry

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 21 02:54:55 2017
pi@raspberrypi:~ $
```

Step 5

Here, we get the Raspberry Pi connected and it is time to conduct the next steps.

Note: If you are not satisfied with using the command window to control the Raspberry Pi, you can also use the remote desktop function, which can help us manage the files in the Raspberry Pi easily.

For details on how to do this, please refer to *Remote Desktop*.

PROJECTS

In this chapter, you will learn how to use RGB Matrix HAT and do some interesting projects.

Download the Code

Use the following command to download the code from the github repository.

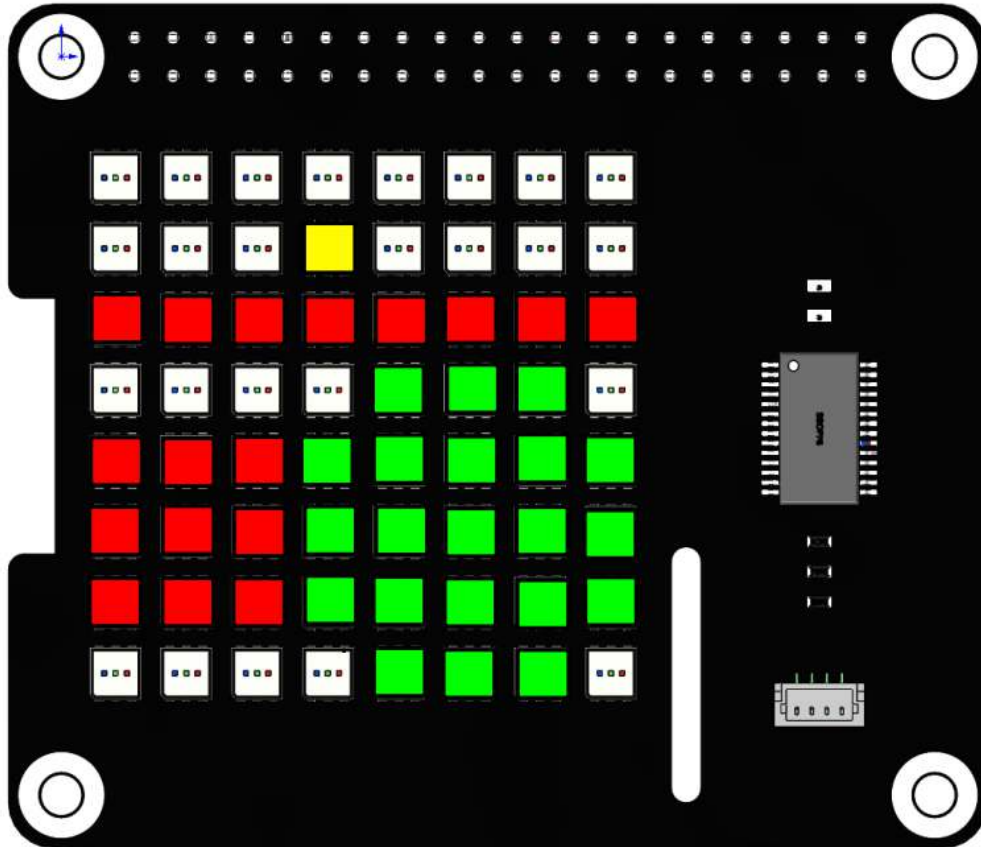
```
git clone https://github.com/sunfounder/rgb_matrix
```

Note: Since RGB Matrix HAT uses I2C for communication, you need to do *I2C Configuration* before running the following projects.

Projects

4.1 Hello Matrix

In this project, you will learn how to make RGB Matix HAT display different patterns and characters in different colors.



Run the code

When the program runs, you will see a point, a line, a rectangle, an ellipse, and the text 'Hi, SunFounder' appears on the RGB Matrix HAT in turn.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 hello_matrix.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
from rgb_matrix import RGB_Matrix
import time

rr = RGB_Matrix(0x74) # create an RGB_Matrix object

point_coor = [3,1]
rr.draw_point(point_coor, fill=(255,255,0)) #draw a point
rr.display()
time.sleep(3)

line_coor = [0,2,7,2]
rr.draw_line(line_coor, fill=(255,0,0)) # draw a line
rr.display()
```

(continues on next page)

(continued from previous page)

```
time.sleep(3)

rectangle_coor = [0,4,2,6]
rr.draw_rectangle(rectangle_coor, fill=(255,0,0))    #draw a rectangle
rr.display()
time.sleep(3)

ellipse_coor = [5,5]
radius = 2
rr.draw_ellipse(ellipse_coor, radius, fill=(0,255,0))    #draw a ellipse
rr.display()    #display the picture which you draw
time.sleep(3)

text = 'Hi, SunFounder'
rr.show_text(text, delay=200, color=(0,0,255))    # show text
rr.display()
time.sleep(4)
```

How it works?

```
from rgb_matrix import RGB_Matrix

rr = RGB_Matrix(0x74)
```

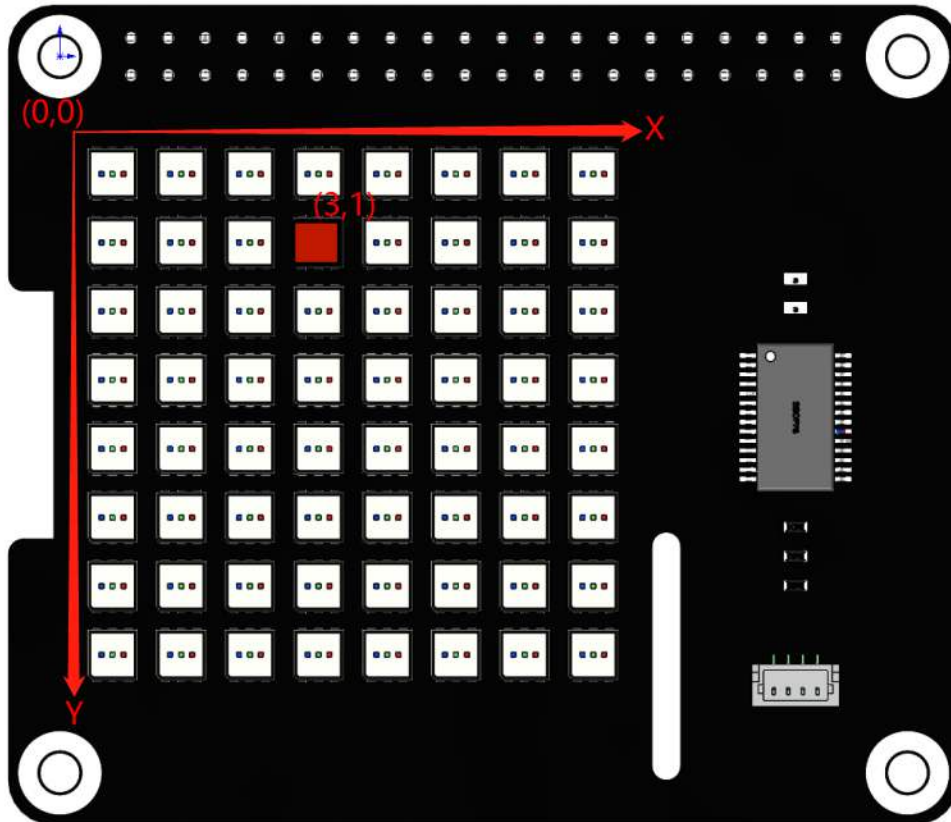
Import the `RGB_Matrix` class, and then create its object `rr` for us to call its class member functions.

```
point_coor = [3,1]
rr.draw_point(point_coor, fill=(255,255,0))    #draw a point
rr.display()
time.sleep(3)
```

The above code is to display a yellow dot on the (3,1) coordinate of the RGB dot matrix.

`draw_point()` is a function that draws a point with 2 parameters: the first parameter is the coordinate on the RGB Matrix HAT, and the second parameter sets the color for the point.

The x,y coordinate directions of the dot matrix are as follows, with the first RGB LED in the upper left corner as the coordinate origin.



The fill tuple contains three elements R, G and B (red, green and blue) in the range 0-255. For example, when `fill=(255,0,0)`, red is displayed. Refer to: https://www.rapidtables.com/web/color/RGB_Color.html for more color value combinations.

Once the coordinate and color are determined, the `display()` function is called to implement on the RGB dot matrix HAT.

```
line_coor = [0,2,7,2]
rr.draw_line(line_coor,fill=(255,0,0)) # draw a line
rr.display()
time.sleep(3)
```

The above code draws a red line starting at coordinate (0,2) and ending at (7,2).

`draw_line()` is a line drawing function, `line_coor=[0,2,7,2]` stores the coordinates of the start and end of the line (2 points determine a line). `fill=(255,0,0)` represents the line color is red.

```
rectangle_coor = [0,4,2,6]
rr.draw_rectangle(rectangle_coor,fill=(255,0,0)) #draw a rectangle
rr.display()
time.sleep(3)
```

The above code draws a red rectangle with coordinates (0,4) and coordinates (2,6) as diagonal coordinates.

`draw_rectangle()` is a function that draws a rectangle. The list `rectangle_coor = [0,4,2,6]` represents the two diagonal coordinates of the rectangle (0,4) and (2,6). The `fill=(255,0,0)` indicates that the rectangle color is red.

```

ellipse_coor = [5,5]
radius = 2
rr.draw_ellipse(ellipse_coor,radius,fill=(0,255,0)) #draw a ellipse
rr.display() #display the picture which you draw
time.sleep(3)

```

The above code draws a green circle with the coordinates (5,5) as the center and a radius of 2.

`draw_ellipse()` is a function that draws a circle with three arguments that determine the center, radius and color of the circle.

```

text = 'Hi, SunFounder'
rr.show_text(text, delay=200,color=(0,0,255)) # show text
rr.display()
time.sleep(4)

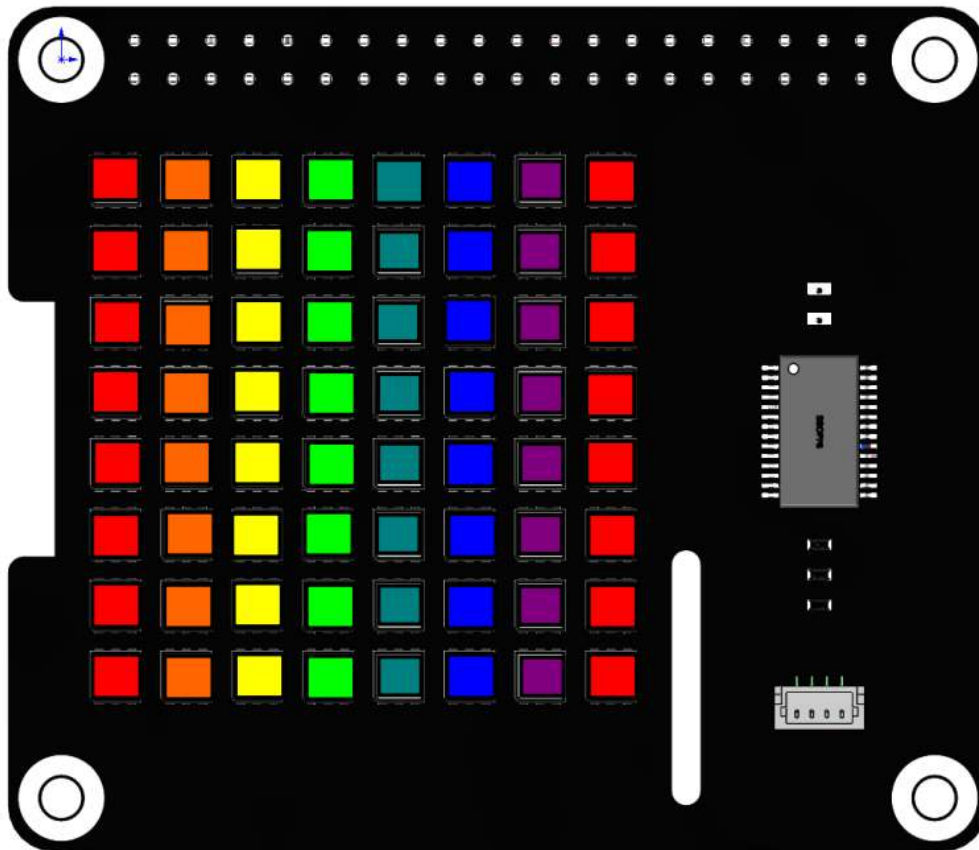
```

The above code is to move and display 'Hi, SunFounder' on the RGB Matrix HAT.

`show_text()` is used to display text information, `text` represents the string to be displayed, `delay` represents the moving time, the larger the value, the slower the text moving speed.

4.2 Dazzling Light

In the previous project, we learned to use some simple functions to make RGB Matrix HAT work. So here, we will use the `draw_line()` function with different colors to make RGB Matrix HAT make more cool effects.



Run the code

As the program runs, you will see the colors on the RGB Matrix HAT changing from right to left.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 dazzling_lights.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
from rgb_matrix import RGB_Matrix
import time

def ColorHSV(hue):

    if hue < 510: # Red to Green-1
        b = 0
        if hue < 255: # Red to Yellow-1
            r = 255
            g = hue # g = 0 to 254
        else: # Yellow to Green-1
            r = 510 - hue # r = 255 to 1
            g = 255

    elif hue < 1020: # Green to Blue-1
        r = 0
        if hue < 765: # Green to Cyan-1
            g = 255
            b = hue - 510 # b = 0 to 254
        else: # Cyan to Blue-1
            g = 1020 - hue # g = 255 to 1
            b = 255

    elif hue < 1530: # Blue to Red-1
        g = 0
        if hue < 1275: # Blue to Magenta-1
            r = hue - 1020 # r = 0 to 254
            b = 255
        else: # Magenta to Red-1
            r = 255
            b = 1530 - hue # b = 255 to 1

    else: # Last 0.5 Red (quicker than % operator)
        r = 255
        g = b = 0

    list = [r, g, b]
    return list

def flash():
    list = [[0, 0, 0, 7],
            [1, 0, 1, 7],
```

(continues on next page)

(continued from previous page)

```

        [2, 0, 2, 7],
        [3, 0, 3, 7],
        [4, 0, 4, 7],
        [5, 0, 5, 7],
        [6, 0, 6, 7],
        [7, 0, 7, 7]]

firsthue = 0
hue = 0
while firsthue < 1530:
    j = 0
    for i in list:
        hue = firsthue + j * 95
        j = j + 1
        if hue > 1530:
            hue = hue - 1530
        temp = ColorHSV(hue)
        #print(temp[0],temp[1],temp[2])
        #time.sleep(2)
        rr.draw_line(i, (temp[0], temp[1], temp[2]))
    rr.display()
    firsthue = firsthue + 11

if __name__ == "__main__":
    rr = RGB_Matrix(0X74)

    while True:
        flash()

```

How it works?

In reality, there are three primary colors of red, yellow, and blue, and there are generally three primary colors of red, green, and blue in the display screen, that is, RGB. Their values are generally used FF0000,00FF00,0000FF means, converted to decimal is (255,0,0),(0,255,0),(0,0,255). This [website](#) can help us better understand the three primary colors.

```

def ColorHSV(hue):

    if hue < 510: # Red to Green-1
        b = 0
        if hue < 255: # Red to Yellow-1
            r = 255
            g = hue # g = 0 to 254
        else: # Yellow to Green-1
            r = 510 - hue # r = 255 to 1
            g = 255

    elif hue < 1020: # Green to Blue-1
        r = 0
        if hue < 765: # Green to Cyan-1
            g = 255
            b = hue - 510 # b = 0 to 254
        else: # Cyan to Blue-1
            g = 1020 - hue # g = 255 to 1
            b = 255

```

(continues on next page)

(continued from previous page)

```

elif hue < 1530: # Blue to Red-1
    g = 0
    if hue < 1275: # Blue to Magenta-1
        r = hue - 1020 # r = 0 to 254
        b = 255
    else: # Magenta to Red-1
        r = 255
        b = 1530 - hue # b = 255 to 1

else: # Last 0.5 Red (quicker than % operator)
    r = 255
    g = b = 0

list = [r, g, b]
return list

```

Reference from [Adafruit_NeoPixel](#).

Because red is centered on the rollover point (the +32768 above, essentially a fixed-point +0.5), the above actually yields 0 to 1530, where 0 and 1530 would yield the same thing. Rather than apply a costly modulo operator, 1530 is handled as a special case below.

So you'd think that the color "hexcone" (the thing that ramps from pure red, to pure yellow, to pure green and so forth back to red, yielding six slices), and with each color component having 256 possible values (0-255), might have 1536 possible items (6*256), but in reality there's 1530. This is because the last element in each 256-element slice is equal to the first element of the next slice, and keeping those in there this would create small discontinuities in the color wheel. So the last element of each slice is dropped... we regard only elements 0-254, with item 255 being picked up as element 0 of the next slice. Like this:

- Red to not-quite-pure-yellow is: 255, 0, 0 to 255, 254, 0
- Pure yellow to not-quite-pure-green is: 255, 255, 0 to 1, 255, 0
- Pure green to not-quite-pure-cyan is: 0, 255, 0 to 0, 255, 254
- and so forth.

Hence, 1530 distinct hues (0 to 1529), and hence why the constants below are not the multiples of 256 you might expect.

```

def flash():
    list = [[0,0,0,7],
            [1,0,1,7],
            [2,0,2,7],
            [3,0,3,7],
            [4,0,4,7],
            [5,0,5,7],
            [6,0,6,7],
            [7,0,7,7]]

```

The list `list` stores the starting and ending coordinates of the 8 vertical lines (from left to right), so that each line can be given a different color in the code later to achieve the colorful effect.

```

firsthue = 0
hue = 0
while firsthue < 1530:
    j = 0
    for i in list:

```

(continues on next page)

(continued from previous page)

```

hue = firsthue + j*95
j = j + 1
if hue > 1530:
    hue = hue-1530
temp = ColorHSV(hue)
rr.draw_line(i, (temp[0],temp[1],temp[2]))
rr.display()
firsthue = firsthue + 11

```

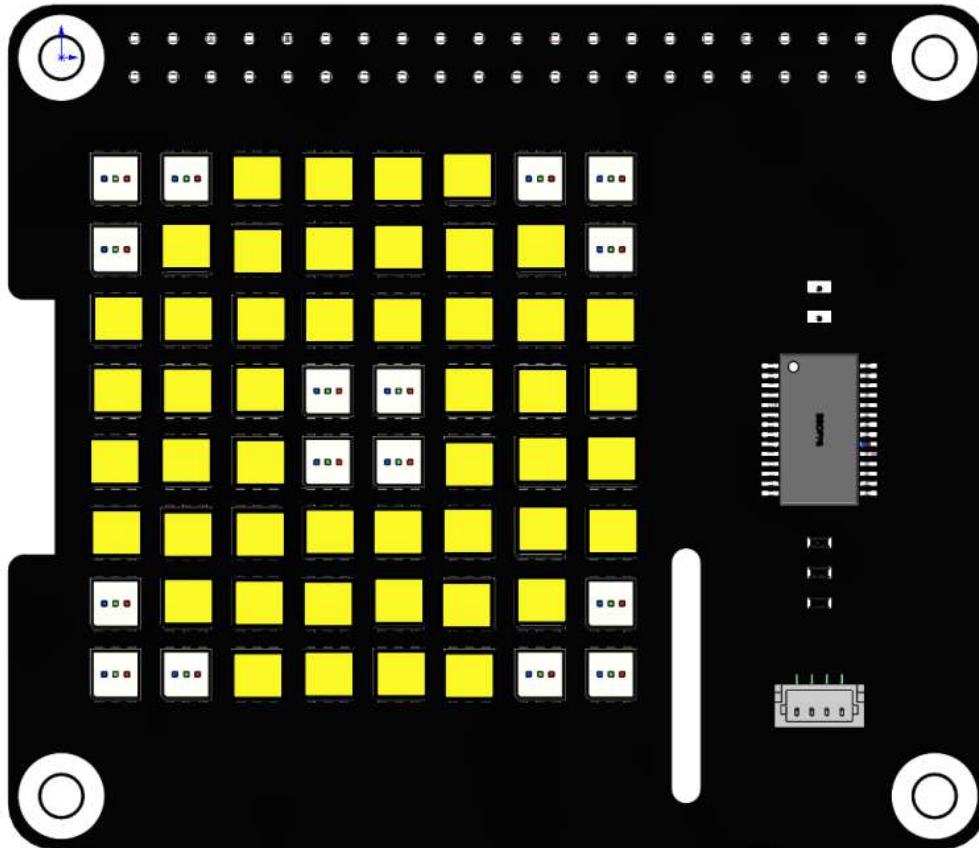
firsthue and hue are passed to `ColorHSV()` as parameters.

Define a two-layer loop, the inner for loop is to draw eight lines in eight different colors, The outer while loop is to add 11 to the hue values of the eight colors to achieve the effect of color flow.

For example, in the first for loop, 0, 95, 190, 285, 380, 475, 570, 665 are used as the hue value of the initial color of the 8 lines, and then enter the outer loop to increase the hue value of each line color by 11 to become 11, 106, 201, 296, 391, 486, 581, 676 to achieve the effect of line color sliding.

4.3 Moving Eye

In this project, we will use the `draw_rectangle()` and `draw_point()` functions to draw an eye pattern and achieve the effect of moving the eye around.



Run the code

When the program is running, you will see an eye moving around on the RGB matrix HAT.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 moving_eyes.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
from rgb_matrix import RGB_Matrix
import time

def up(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[1] -= 1
        list[3] -= 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

def down(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[1] += 1
        list[3] += 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

def left(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[0] -= 1
        list[2] -= 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

def right(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[0] += 1
        list[2] += 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

def left_down(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[0] -= 1
        list[2] -= 1
        list[1] += 1
        list[3] += 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

def left_up(list, step=1):
    for i in range(0, step):
```

(continues on next page)

(continued from previous page)

```

rr.draw_rectangle(list, fill=(251, 248, 40))
list[0] -= 1
list[2] -= 1
list[1] -= 1
list[3] -= 1
rr.draw_rectangle(list, fill=(0, 0, 0))
rr.display()

def right_up(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[0] += 1
        list[2] += 1
        list[1] -= 1
        list[3] -= 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

def right_down(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251, 248, 40))
        list[0] += 1
        list[2] += 1
        list[1] += 1
        list[3] += 1
        rr.draw_rectangle(list, fill=(0, 0, 0))
        rr.display()

if __name__ == "__main__":
    rr = RGB_Matrix(0X74)

    rectangle_coor = [0, 0, 7, 7]
    rr.draw_rectangle(rectangle_coor, fill=(251, 248, 40))

    point_array = [[0, 0], [1, 0], [0, 1], [6, 0], [7, 0], [7, 1], [0, 6], [0, 7], [1, 7], [7, 6], [7, 7],
↪ [6, 7]]
    for i in range(len(point_array)):
        rr.draw_point(point_array[i], fill=(0, 0, 0))

    list = [3, 3, 4, 4]
    rr.draw_rectangle(list, fill=(0, 0, 0), outline=None, width=0)

    rr.display()
    while True:
        up(list, 3)
        down(list, 6)
        up(list, 6)
        down(list, 6)
        up(list, 3)
        time.sleep(1)
        right_down(list, 2)
        up(list, 4)
        left(list, 4)
        down(list, 4)
        right(list, 4)
        left_up(list, 2)
        time.sleep(1)

```

How it works?

```
rectangle_coor = [0,0,7,7]
rr.draw_rectangle(rectangle_coor, fill=(251,248,40))

point_array = [[0,0],[1,0],[0,1],[6,0],[7,0],[7,1],[0,6],[0,7],[1,7],[7,6],[7,7],[6,7]]
for i in range(len(point_array)):
    rr.draw_point(point_array[i], fill=(0,0,0))

list = [3,3,4,4]
rr.draw_rectangle(list, fill=(0,0,0), outline=None, width=0)

rr.display()
```

- The list `rectangle_coor` represents a rectangle (the whole RGB dot matrix) from coordinates (0, 0) to (7, 7), and then use the `draw_rectangle()` function to fill this rectangle with yellow.
- The list `point_array` represents the 12 points in the four corners, then use the `draw_point()` function to set the color of each point to (0, 0, 0), i.e., extinguish these points. This depicts the outline of an eye.
- The `list` represents a small rectangle from (3, 3) to (4, 4), and then use the `draw_rectangle()` function to set the color of this rectangle to (0, 0, 0) to make the rectangle go out. This will describe the outline of the eyeball.
- Finally, the eye pattern is displayed on the RGB Matrix HAT using the `display()` function.

```
while True:
    up(list, 3)
    down(list, 6)
    up(list, 6)
    down(list, 6)
    up(list, 3)
    time.sleep(1)
    right_down(list, 2)
    up(list, 4)
    left(list, 4)
    down(list, 4)
    right(list, 4)
    left_up(list, 2)
    time.sleep(1)
```

The main loop is to make the eyeball keep moving up and down, then turn one cycle, and finally return to the original position.

We call some functions to move the eyeball, for example `up(list, 3)` is to move the eyeball up three squares, now look at how this function is implemented.

```
def up(list, step=1):
    for i in range(0, step):
        rr.draw_rectangle(list, fill=(251,248,40))
        list[1] -= 1
        list[3] -= 1
        rr.draw_rectangle(list, fill=(0,0,0))
        rr.display()
```

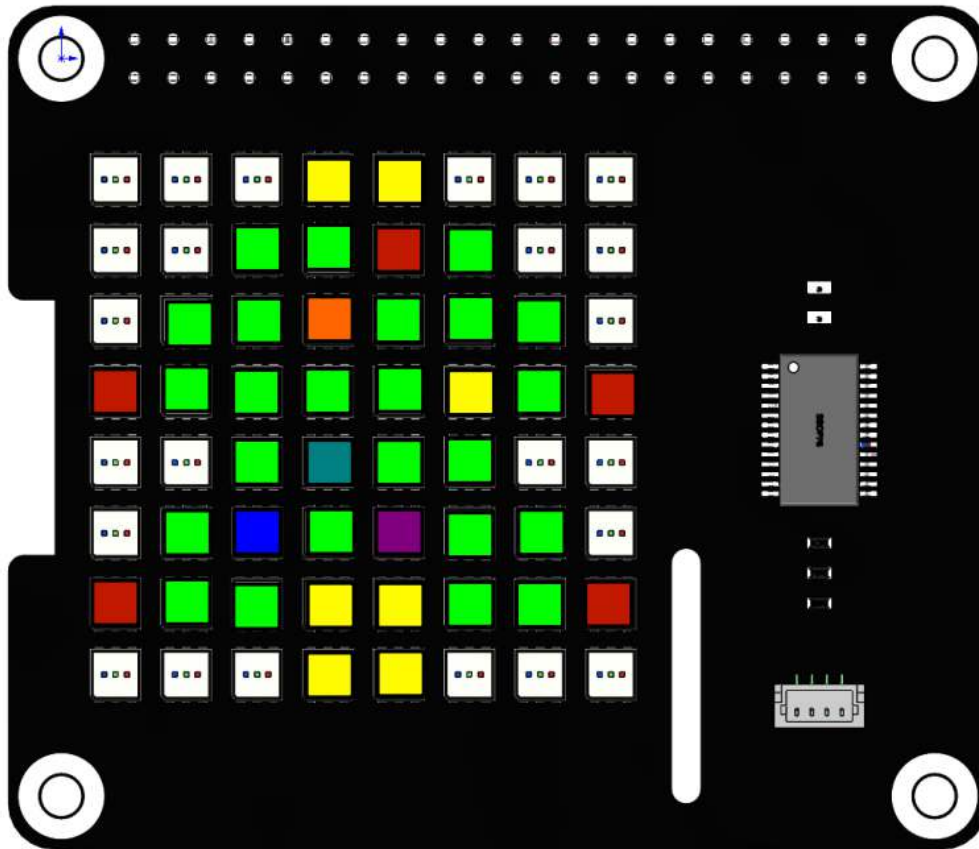
The `up()` function has 2 parameters `list` and `step`, the internal logic is to move the rectangle `list` up `step` squares (default is 1).

- Define a `for()` loop with the number of loops determined by `step`. In the `for()` loop, set the color of the rectangle `list` to yellow.

- `list = [3, 3, 4, 4]` are the 2 diagonal coordinates (3,3) and (4,4), `list[1]` and `list[3]` are subtracted by one, meaning that the y-values of the 2 diagonal coordinates are subtracted by one.
- Then the modified `list = [3, 2, 4, 3]` color is set to (0,0,0) by the function `draw_rectangle()` and displayed on the dot matrix by the function `display()`.
- After one for loop in this way, the pupil is moved up one square.

4.4 Christmas Tree

In this project, we will use the `draw_point()` function to make a colorful Christmas tree.



Run the code

When the program runs, you will see a shiny Christmas tree appear on the RGB Matrix HAT.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 christmas_tree.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
from rgb_matrix import RGB_Matrix
import time
from color import Color

def tree():
    for i in green_coor:
        rr.draw_point(i, (0,255,0))

    for i in yellow_coor:
        rr.draw_point(i, (255,255,0))

    for i in red_coor:
        rr.draw_point(i, (255,0,0))

    rr.display()

def dot():
    col = Color()

    for i in flash_coor:
        rr.draw_point(i,col.random())
    rr.display()

if __name__ == "__main__":
    rr = RGB_Matrix(0X74)

    rectangle_coor = [0,0,7,7]

    green_coor = [[3,0],[4,0],
                 [2,1],[3,1],[5,1],
                 [1,2],[2,2],[4,2],[5,2],[6,2],
                 [1,3],[2,3],[3,3],[4,3],[6,3],
                 [2,4],[4,4],[5,4],
                 [1,5],[3,5],[5,5],[6,5],
                 [1,6],[2,6],[3,6],[4,6],[5,6],[6,6]
                 ]

    flash_coor = [[4,1],[3,2],[5,3],[3,4],[2,5],[4,5]]
    red_coor = [[0,3],[7,3],[0,6],[7,6]]
    yellow_coor = [[3,0],[4,0],[3,6],[4,6],[3,7],[4,7]]

    tree()
    while True:
        dot()
```

How it works?

```
from color import Color
```

Import the color class `Color`, which is a class that we encapsulate to manipulate RGB Matrix HAT colors. In this project, we will use a class function `random()` to display random colors.

```
green_coor = [[3,0],[4,0],
              [2,1],[3,1],[5,1],
```

(continues on next page)

(continued from previous page)

```

    [1,2],[2,2],[4,2],[5,2],[6,2],
    [1,3],[2,3],[3,3],[4,3],[6,3],
    [2,4],[4,4],[5,4],
    [1,5],[3,5],[5,5],[6,5],
    [1,6],[2,6],[3,6],[4,6],[5,6],[6,6]]

flash_coor = [[4,1],[3,2],[5,3],[3,4],[2,5],[4,5]]
red_coor = [[0,3],[7,3],[0,6],[7,6]]
yellow_coor = [[3,0],[4,0],[3,6],[4,6],[3,7],[4,7]]

```

Divide the Christmas tree into four parts, the red part, the yellow part, the green part, and the blinking part, so we need four lists to store these coordinates.

```

def tree():

    for i in green_coor:
        rr.draw_point(i, (0,255,0))

    for i in yellow_coor:
        rr.draw_point(i, (255,255,0))

    for i in red_coor:
        rr.draw_point(i, (255,0,0))

    rr.display()

```

Define a `tree()` function to draw the green(0, 255, 0), yellow(255, 255, 0) and red(255, 0, 0) parts of the Christmas tree.

```

def dot():
    col = Color()

    for i in coor:
        rr.draw_point(i,col.random())
    rr.display()

```

For the blinking points in the Christmas tree, we can use the `random()` function in the `Color` class to achieve. The function of `random()` is to return a random RGB value, that is, to display random colors in a loop to achieve a blinking effect.

```

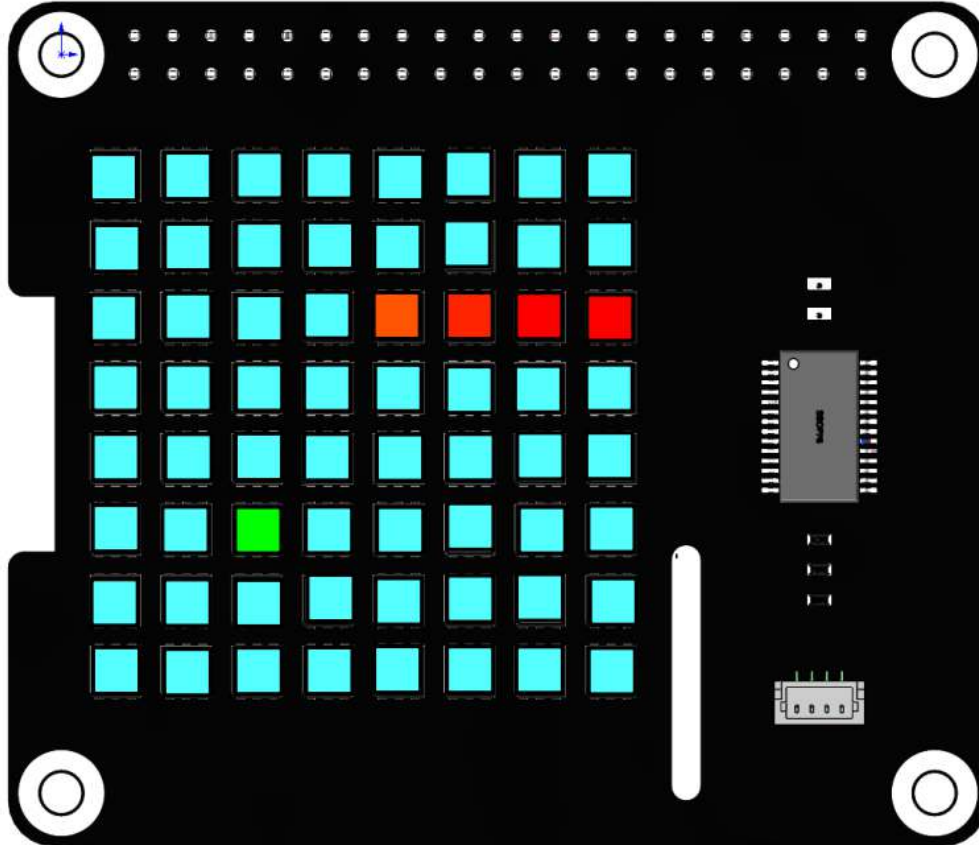
tree()
while True:
    dot()

```

Finally, two functions are called to draw the Christmas tree. The blinking is continuous, so `dot()` should be called in the loop.

4.5 Greedy Snake

In this project, we use the RGB matrix HAT as the display screen to create a snake eating game by reading the values of the keyboard keys to change the display effect.



Run the code

When the program runs, the snake game starts, the keyboard `a` and `d` controls the snake to turn left and right. Each time the snake eats a bean, score plus one. When it encounters itself, the score returns to zero. After a period of time, the screen will turn blue, and the score will be displayed after a while. Press `q` to end the game, and `Ctrl+C` to exit the program.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 snake_game.py
```

Code

You can view the code by typing the command `nano snake_game.py` in Terminal or by clicking on [github-snake_game.py](#).

How it works?

```
def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
```

(continues on next page)

(continued from previous page)

```

finally:
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
return ch

```

This function is used to read keyboard input and return the entered characters.

```

def Keyboard_control():
    while True:

        global power_val, key
        # key = 'n'
        key=readkey()
        time.sleep(0.22)
        if key=='q':
            print ("quit")
            break

```

Keyboard_control() is used for keyboard control. Call readkey() in an infinite loop to receive the characters input by the keyboard. In addition, the logic of snake-eating is also an infinite loop, then multithreading may be needed when there are multiple infinite loops in a program.

```

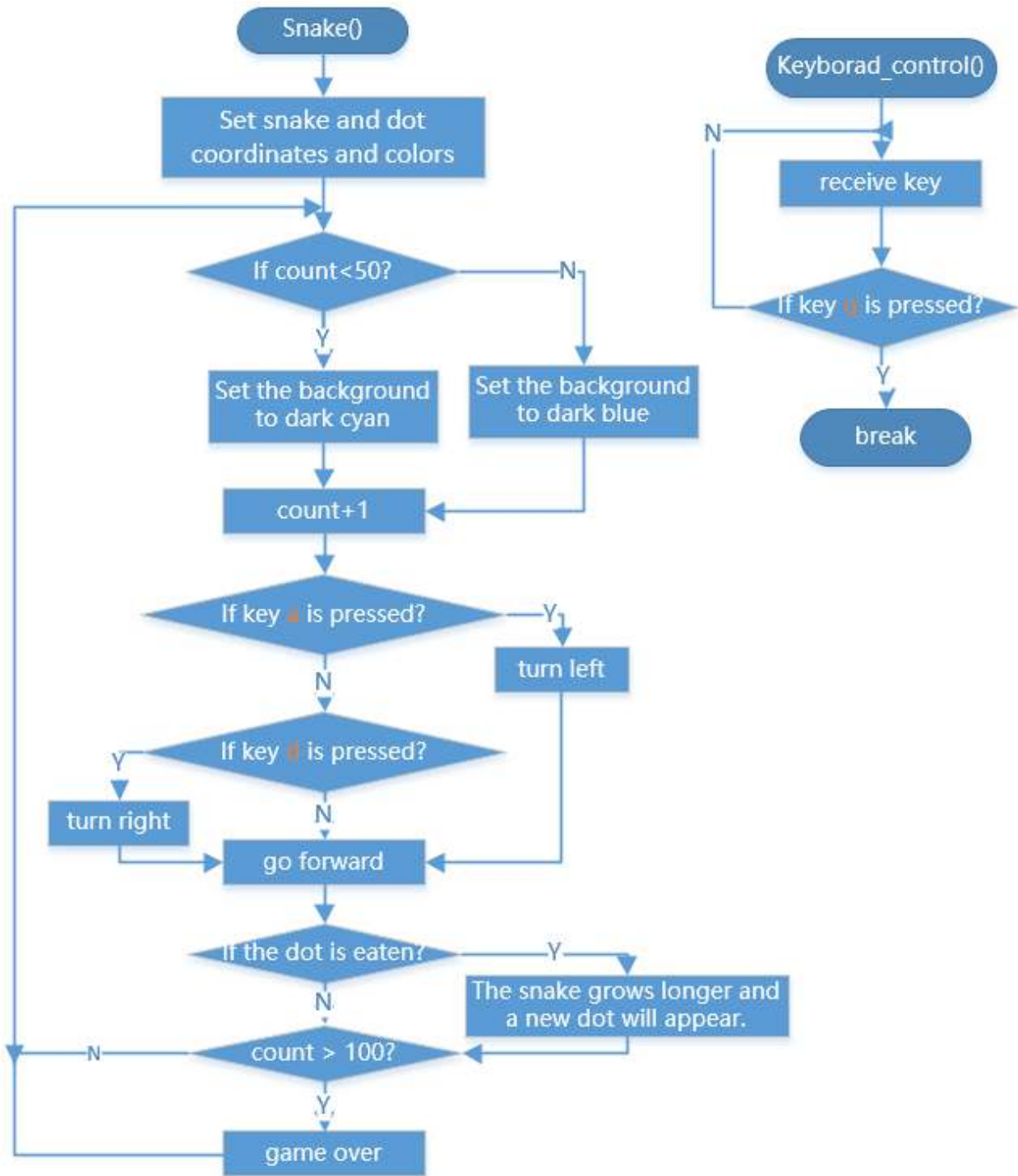
def snake_game():
    global key
    rr = RGB_Matrix(0X74)
    rectangle_coor = [0,0,7,7]
    #rr.draw_rectangle(rectangle_coor, fill=(51,51,0)) #draw a rectangle
    coor_1 = np.asarray([0,2])
    coor_2 = np.asarray([1,2])
    coor_3 = np.asarray([2,2])
    coor_4 = np.asarray([3,2])
    coor_list = [coor_1, coor_2, coor_3, coor_4]
    ...

```

snake_game() is used to represent snake-eating logic. The received key value is a character entered by the keyboard, which needs to be declared as a global variable with global.

There are three main parts of snake logic:

- In the first part, when the a or d key is not pressed, first judge the horizontal and vertical, and then judge the forward direction, and then add 1 or subtract 1 to the horizontal or vertical coordinates of each point to achieve the effect of moving up, down, left, and right.
- The second part is to judge whether the a or d key is pressed. If yes, then judge the horizontal and vertical direction and then determine the forward direction, and then add one or subtract one to the coordinates of each point of the snake head, and the snake head coordinates are additionally processed to achieve the effect of turning the head.
- The third part is to determine whether the snake head is in contact with dot. If yes, set eat_flag to False and add an element to the list of snakes to achieve the effect of growing snakes.



```

if __name__ == "__main__":
    t1 = threading.Thread(target=Keyboard_control)
    t2 = threading.Thread(target=snake_game)
    t1.setDaemon(True)
    t2.setDaemon(True)
    t1.start()
    t2.start()
    while True:

```

(continues on next page)

(continued from previous page)

`pass`

The `Thread` method in the `threading` ``class can help us create a thread, and the parameter is ``target=function name.

- `SetDaemon(True)` sets the thread as a daemon thread. It is generally used in an unimportant thread with an infinite loop.
- `Start()` starts the thread.

4.6 Camera Recognition

In this project, we will make a color recognizer, connect a camera module to the Raspberry Pi, use `PiCamera` and `OpenCV` to process the objects captured by the camera, and express its colors with RGB Matrix HAT.

Note: This example needs to enable the Raspberry Pi Camera function.

Then install third-party dependencies

```
sudo pip3 install opencv-contrib-python
sudo apt-get install -y python3-h5py libatlas-base-dev
sudo pip3 install -U numpy
```

Run the code

When the program is running, hold the camera module and aim at some brightly colored objects, you will find that the RGB Matrix HAT also shows similar colors.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 camera.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
# To run this example, you need to turn on Camera on raspi-config => Interfacing =>
↳ Camera
# Then run the following command to install dependencies
# sudo pip3 install opencv-contrib-python
# sudo apt-get install -y python3-h5py libatlas-base-dev
# sudo pip3 install -U numpy

from picamera.array import PiRGBArray # Generates a 3D RGB array
from picamera import PiCamera
import time
import cv2

from PIL import Image
from rgb_matrix import RGB_Matrix

rr = RGB_Matrix(0X74)
```

(continues on next page)

(continued from previous page)

```
camera = PiCamera()
camera.resolution = (1280, 720)
raw_capture = PiRGBArray(camera, size=(1280, 720))
# Allow the camera to warmup
time.sleep(0.1)
# Grab an image from the camera
for frame in camera.capture_continuous(raw_capture, format="rgb", use_video_
↳port=True):

    image = frame.array

    # Convert image to 8x8 for RGB matrix
    img = cv2.resize(image, (8, 8), interpolation = cv2.INTER_AREA)
    im_pil = Image.fromarray(img)

    # Render
    rr.image(list(im_pil.getdata()))

    raw_capture.truncate(0)
```

How it works?

```
from picamera.array import PiRGBArray # Generates a 3D RGB array
from picamera import PiCamera
import time
import cv2

from PIL import Image
from rgb_matrix import RGB_Matrix
```

- Import PiCamera to support the use of the camera.
- Import PiRGBArray to help the Raspberry Pi output the captured images in the form of an array.
- Import OpenCV vision library where cv2` is the name of the C++ namespace of Opencv.
- Import the image processing library PIL of the python platform.

```
camera = PiCamera()
camera.resolution = (1280, 720)
raw_capture = PiRGBArray(camera, size=(1280, 720))
```

Create a PiCamera object and call PiRGBArray() to generate an RGB three-dimensional array with a resolution of (1280, 720) and pass it to raw_capture.

```
for frame in camera.capture_continuous(raw_capture, format="rgb", use_video_
↳port=True):

    image = frame.array
```

Traverse the images captured by the camera and pass them to the image in the form of an RGB three-dimensional array.

```
img = cv2.resize(image, (8, 8), interpolation = cv2.INTER_AREA)
im_pil = Image.fromarray(img)
```

Convert the picture into an 8x8 RGB Matrix HAT and pass it to im_pil in the form of an array.

```
rr.image(list(im_pil.getdata()))
```

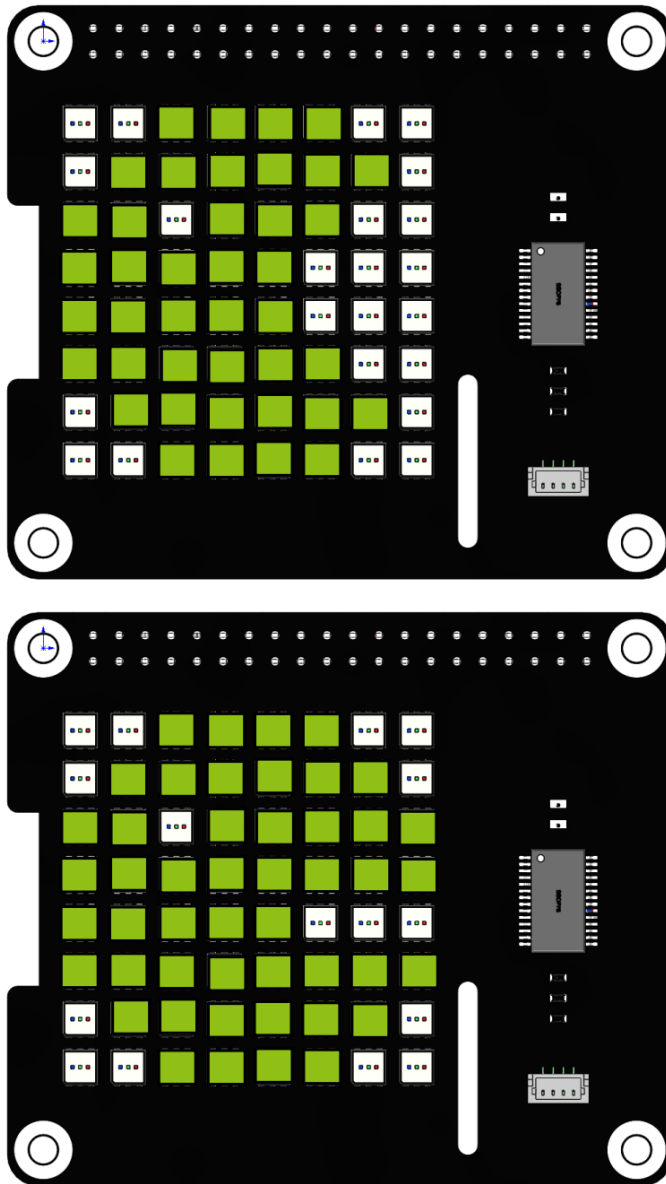
Convert `im_pil` into a list form to be used as a parameter of `rr.image` to light up the RGB Matrix HAT.

```
raw_capture.truncate(0)
```

Clear `raw_capture` in this loop.

4.7 Custom Shape

In the previous project, we made a Christmas tree with point coordinates. In this project, we used straight lines to piece together a pattern of Pac-Man.



Run the code

When the program runs, you will see a Pac-Man appearing on the RGB Matrix HAT, and its mouth is continuously opening and closing.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 custom_shape.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
from rgb_matrix import RGB_Matrix
import time
import random

def pacman():

    rectangle_coor = [0,0,7,7]

    list = [[2,0,5,0],
            [1,1,6,1],
            [0,2,1,2],
            [3,2,5,2],
            [0,3,4,3],
            [0,4,4,4],
            [0,5,5,5],
            [1,6,6,6],
            [2,7,5,7]]

    fill = (144,192,22)
    for i in list:
        rr.draw_line(i,fill)

    rr.display()
    time.sleep(1)

    rr.draw_rectangle(rectangle_coor,fill=(0,0,0))

def pacman2():

    rectangle_coor = [0,0,7,7]

    list = [[2,0,5,0],
            [1,1,6,1],
            [0,2,1,2],
            [3,2,7,2],
            [0,3,7,3],
            [0,4,3,4],
            [0,5,7,5],
            [1,6,6,6],
            [2,7,5,7]]

    fill = (144,192,22)
    for i in list:
        rr.draw_line(i,fill)
```

(continues on next page)

(continued from previous page)

```

rr.display()
time.sleep(1)

rr.draw_rectangle(rectangle_coor, fill=(0,0,0))

if __name__ == "__main__":
    rr = RGB_Matrix(0X74)

    rectangle_coor = [0,0,7,7]

    while True:
        pacman()
        time.sleep(0.5)
        pacman2()

```

How it works?

```

def pacman():

    rectangle_coor = [0,0,7,7]

    list = [[2,0,5,0],
            [1,1,6,1],
            [0,2,1,2],
            [3,2,5,2],
            [0,3,4,3],
            [0,4,4,4],
            [0,5,5,5],
            [1,6,6,6],
            [2,7,5,7]]

```

```

def pacman2():

    rectangle_coor = [0,0,7,7]

    list = [[2,0,5,0],
            [1,1,6,1],
            [0,2,1,2],
            [3,2,7,2],
            [0,3,7,3],
            [0,4,3,4],
            [0,5,7,5],
            [1,6,6,6],
            [2,7,5,7]]

```

Define two functions `pacman()` and `pacman2()` to represent the two states of Pac-Man. These two states are composed of many lines, and two lists are defined to store the starting and ending coordinates of these lines respectively. `rectangle_coor` represents the entire RGB matrix HAT, which can be used to clear the screen.

```

fill = (144,192,22)
for i in list:
    rr.draw_line(i, fill)

rr.display()
time.sleep(1)

```

(continues on next page)

(continued from previous page)

```
rr.draw_rectangle(rectangle_coor, fill=(0,0,0))
```

The above code exists in both functions `pacman()` and `pacman2()` and is used to display the 2 states of Pac-Man in yellow in the RGB Matrix HAT and then clear the screen.

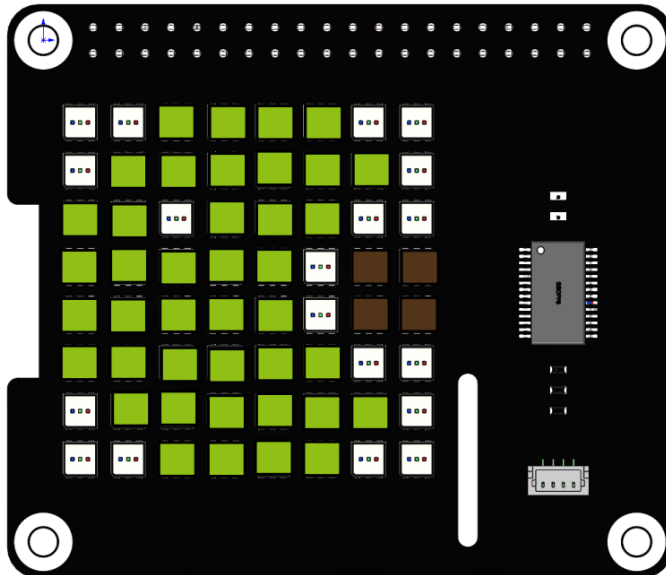
```
while True:
    pacman()
    time.sleep(0.5)
    pacman2()
```

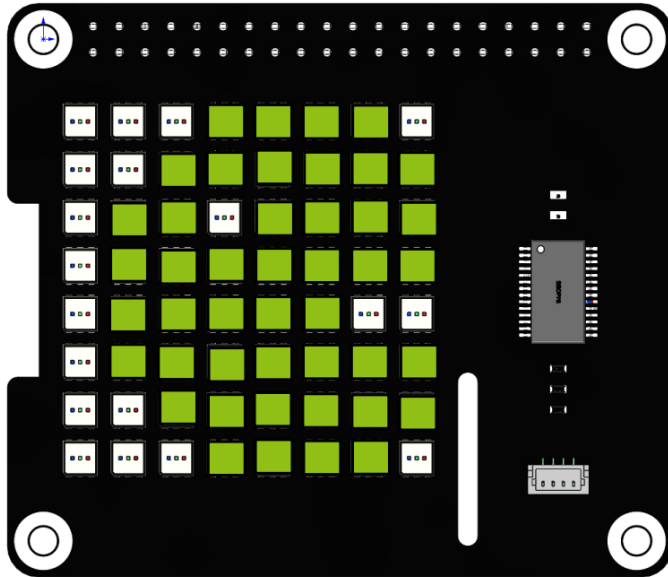
Call `pacman()` and `pacman2()` cyclically to increase the dynamic effect of Pac-Man.

You can also imagine other more interesting patterns, this [website](#) may be able to get some references.

4.8 Custom Dynamic Shape

Here, based on the previous project, a continuous animation of Pac-Man eating dots will be created.





Run the code

When the program is running, you will see Pac-Man on the RGB matrix cap move from left to right and leave after eating the rightmost dot.

```
cd /home/pi/rgb_matrix/raspberrypi
sudo python3 custom_dynamic_shape.py
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `rgb_matrix/raspberrypi`. After modifying the code, you can run it directly to see the effect.

```
from rgb_matrix import RGB_Matrix
import time
import random

def pacman(a, k):

    list2 = [[a-4,0,a-1,0],
             [a-5,1,a,1],
             [a-6,2,a-5,2],
             [a-3,2,a-1,2],
             [a-6,3,a-2,3],
             [a-6,4,a-2,4],
             [a-6,5,a-1,5],
             [a-5,6,a,6],
             [a-4,7,a-1,7]]

    fill = (144,192,22)
    for i in range(0,k+1):
        for j in list2:
            rr.draw_line(j,fill)

    rr.display()
```

(continues on next page)

```
    for j in list2:
        rr.draw_line(j, fill=(0,0,0))

    for i in range(0,9):
        list2[i][0] += 1
        list2[i][2] += 1

    time.sleep(0.1)

def pacman2():

    rr.draw_rectangle(rectangle_coor, fill=(0,0,0))

    list = [[2,0,5,0],
            [1,1,6,1],
            [0,2,1,2],
            [3,2,7,2],
            [0,3,7,3],
            [0,4,3,4],
            [0,5,7,5],
            [1,6,6,6],
            [2,7,5,7]]

    fill = (144,192,22)
    for i in list:
        rr.draw_line(i, fill)

    rr.display()
    time.sleep(0.1)

    rr.draw_rectangle(rectangle_coor, fill=(0,0,0))

def pac():

    coor = [6,3,7,4]
    rr.draw_rectangle(coor, fill=(82,52,25))

if __name__ == "__main__":
    rr = RGB_Matrix(0X74)

    rectangle_coor = [0,0,7,7]

    while True:
        pac()
        pacman(0,6)
        pacman2()
        pacman(6,7)
```

How it works?

```
while True:
    pac()
    pacman(0,6)
    pacman2()
    pacman(6,7)
```

We can disassemble Pac-Man into three actions, `pac()` represents the position of the dot.

- `pacman(0, 6)` means that Pac-Man moves from the far left to the side of the dot.
- `pacman2()` mouth closed to indicate the action of eating.
- `pacman(6, 7)` indicates to continue to leave after eating.

```
def pacman(a, k):
    list2 = [[a-4, 0, a-1, 0],
             [a-5, 1, a, 1],
             [a-6, 2, a-5, 2],
             [a-3, 2, a-1, 2],
             [a-6, 3, a-2, 3],
             [a-6, 4, a-2, 4],
             [a-6, 5, a-1, 5],
             [a-5, 6, a, 6],
             [a-4, 7, a-1, 7]]

    fill = (144, 192, 22)
    for i in range(0, k+1):
        for j in list2:
            rr.draw_line(j, fill)

        rr.display()

        for j in list2:
            rr.draw_line(j, fill=(0, 0, 0))

        for i in range(0, 9):
            list2[i][0] += 1
            list2[i][2] += 1

    time.sleep(0.1)
```

The `pacman()` function is used to make Pac-man move from the left to the right in an open-mouthed state until it disappears. It has two parameters `a` and `k`, `a` represents the starting position of Pac-man and `k` represents the number of squares moved to the right.

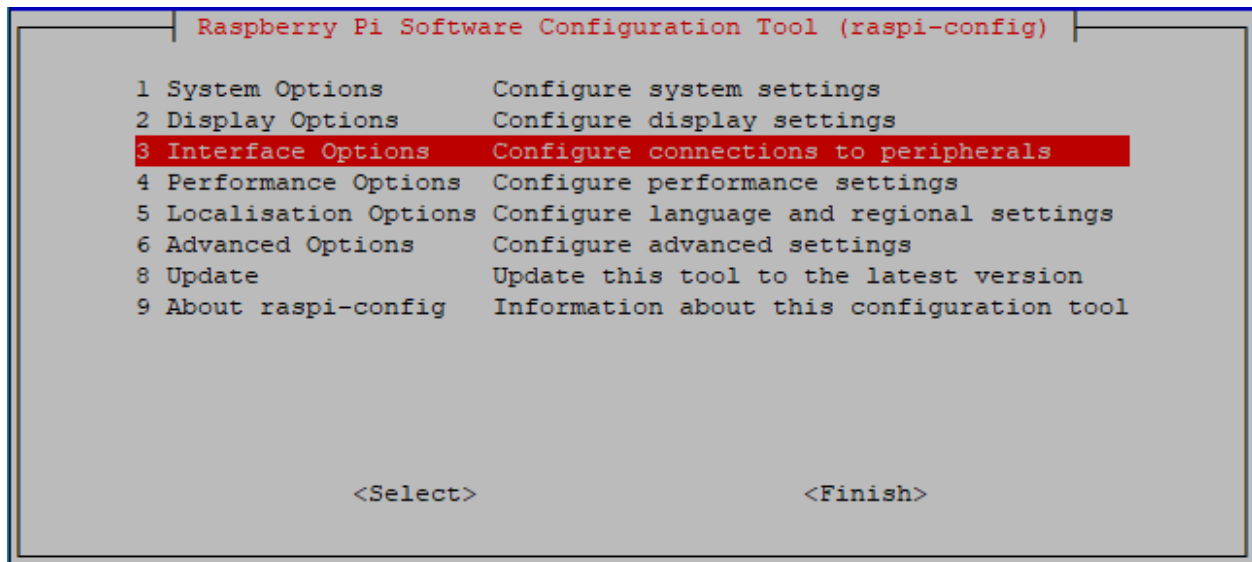
- `list2` stores the coordinates of Pac-man's open-mouth state, drawn as lines, with the x-coordinate of each line determined by `a`.
- Define a two-level for loop. The inner loop does three things: draws Pac-Man, moves each line in `list2` one square to the right, and removes the movement.
- The outer layer repeats the loop `k` times, which means Pac-Man moves `k` squares to the right.

5.1 I2C Configuration

Step 1: Enable the I2C port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

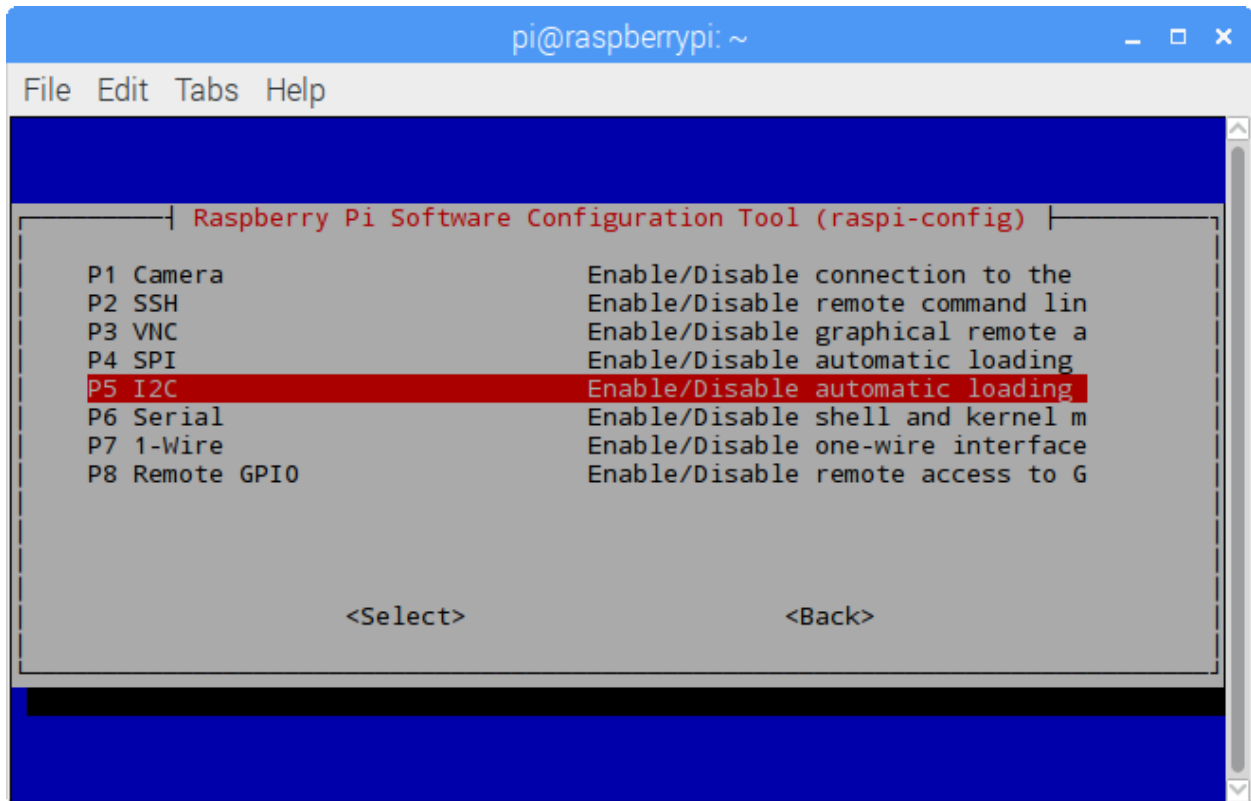
3 Interfacing options



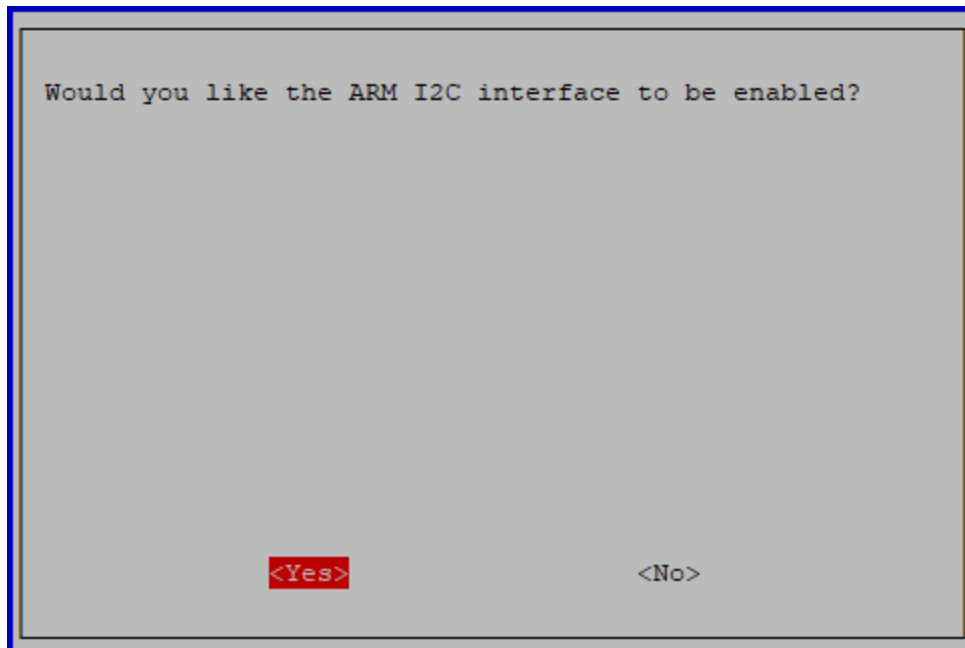
```
Raspberry Pi Software Configuration Tool (raspi-config)
1 System Options          Configure system settings
2 Display Options        Configure display settings
3 Interface Options      Configure connections to peripherals
4 Performance Options    Configure performance settings
5 Localisation Options   Configure language and regional settings
6 Advanced Options       Configure advanced settings
8 Update                 Update this tool to the latest version
9 About raspi-config     Information about this configuration tool

<Select>                <Finish>
```

P5 I2C



<Yes>, then <Ok> -> <Finish>



Step 2: Check whether the i2c modules are loaded and active.

```
lsmod | grep i2c
```

Then the following codes will appear (the number may be different).


```
i2c_dev          6276    0
i2c_bcm2708     4121    0
```

Step 3: Install i2c-tools.

```
sudo apt-get install i2c-tools
```

Step 4: Check the address of the I2C device.

```
i2cdetect -y 1      # For Raspberry Pi 2 and higher version
```

```
i2cdetect -y 0      # For Raspberry Pi 1
```

```
pi@raspberrypi ~ $ i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  48  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

If there is an I2C device connected, the address of the device will be displayed.

5.2 Remote Desktop

There are two ways to control the desktop of the Raspberry Pi remotely:

VNC and XRDP, you can use any of them.

5.2.1 VNC

You can use the function of remote desktop through VNC.

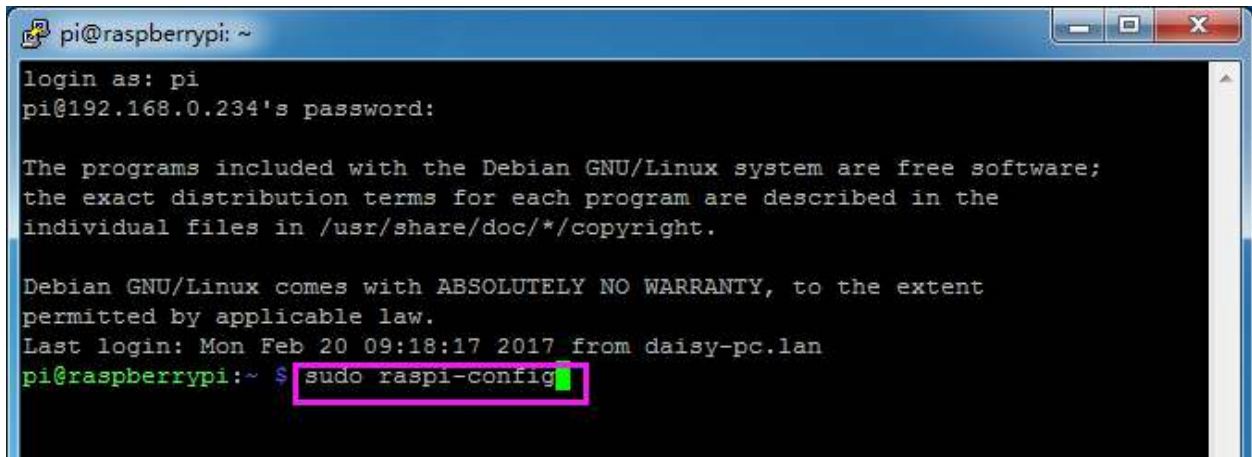
Enable VNC service

The VNC service has been installed in the system. By default, VNC is disabled. You need to enable it in config.

Step 1

Input the following command:

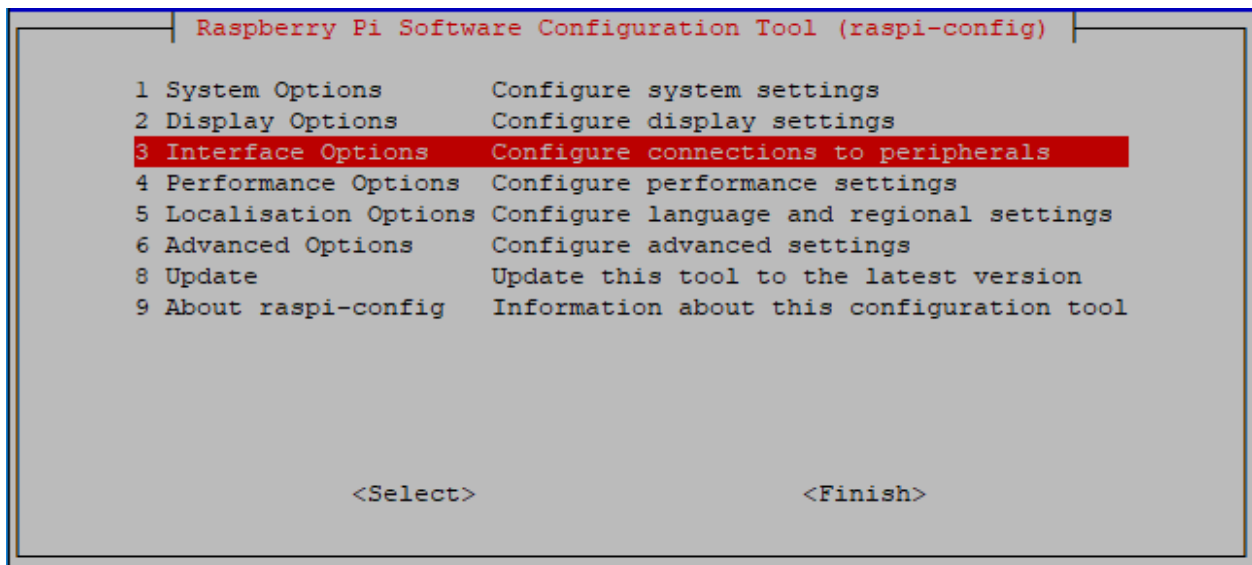
```
sudo raspi-config
```



```
pi@raspberrypi: ~  
login as: pi  
pi@192.168.0.234's password:  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Mon Feb 20 09:18:17 2017 from daisy-pc.lan  
pi@raspberrypi:~ $ sudo raspi-config
```

Step 2

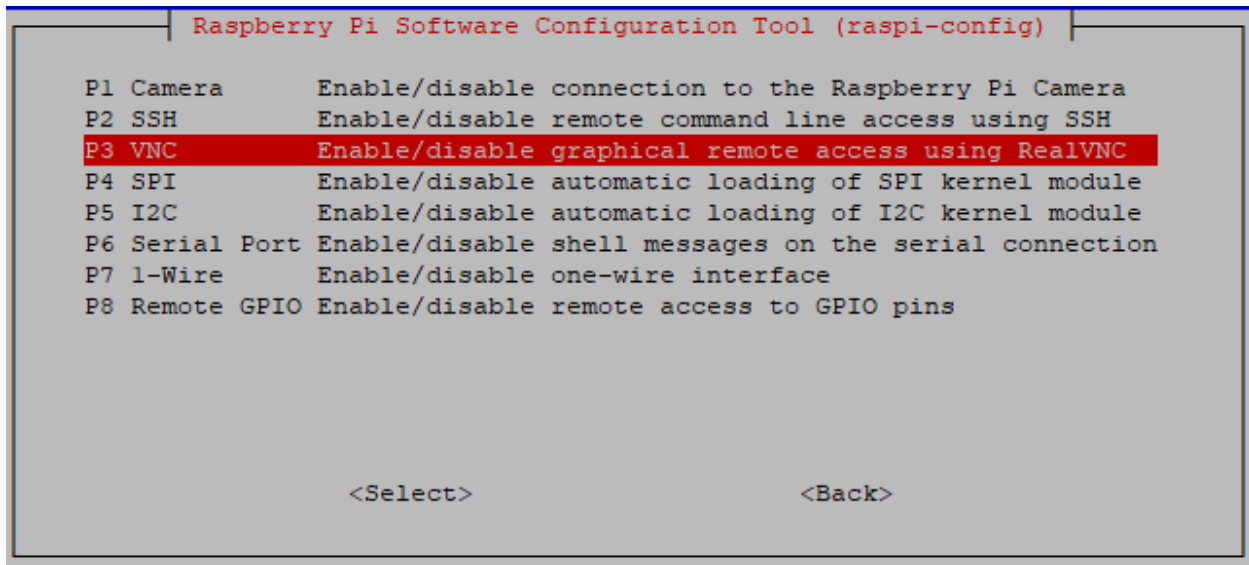
Choose **3 Interfacing Options** by press the down arrow key on your keyboard, then press the **Enter** key.



```
Raspberry Pi Software Configuration Tool (raspi-config)  
  
1 System Options          Configure system settings  
2 Display Options        Configure display settings  
3 Interface Options      Configure connections to peripherals  
4 Performance Options    Configure performance settings  
5 Localisation Options   Configure language and regional settings  
6 Advanced Options       Configure advanced settings  
8 Update                 Update this tool to the latest version  
9 About raspi-config     Information about this configuration tool  
  
<Select>                <Finish>
```

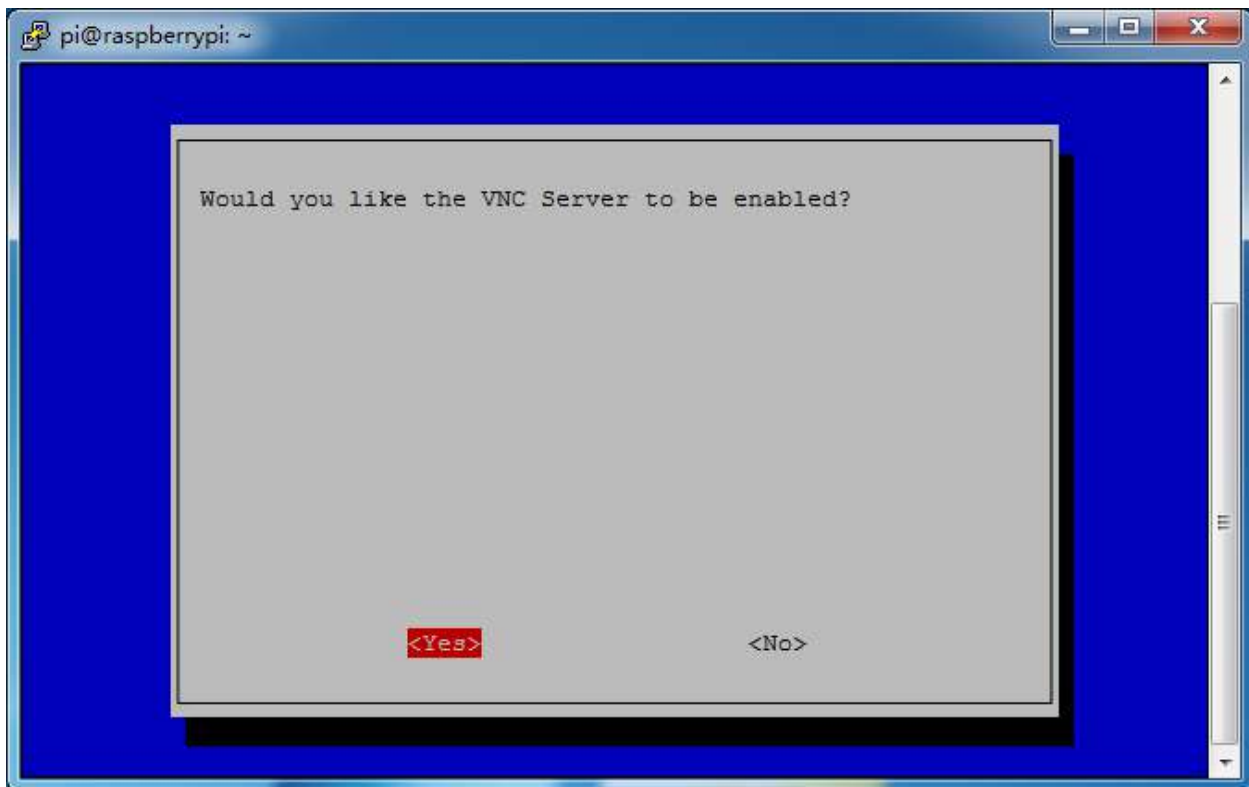
Step 3

P3 VNC



Step 4

Select **Yes** -> **OK** -> **Finish** to exit the configuration.



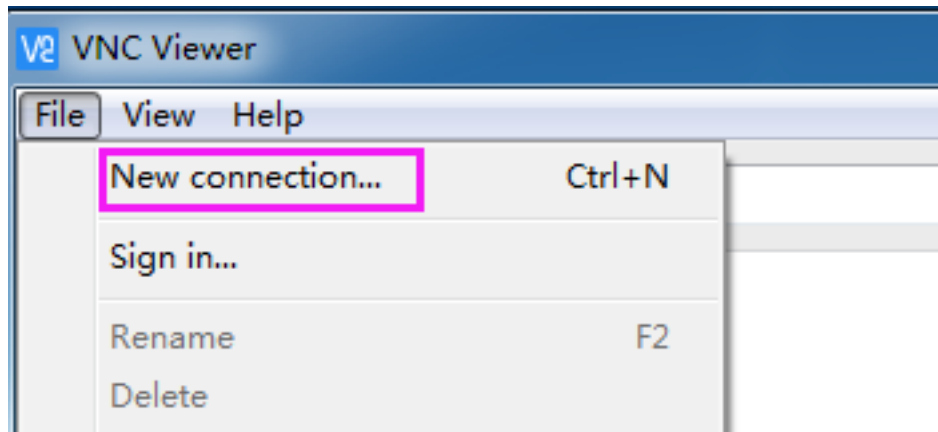
Login to VNC

Step 1

You need to download and install the [VNC Viewer](#) on personal computer. After the installation is done, open it.

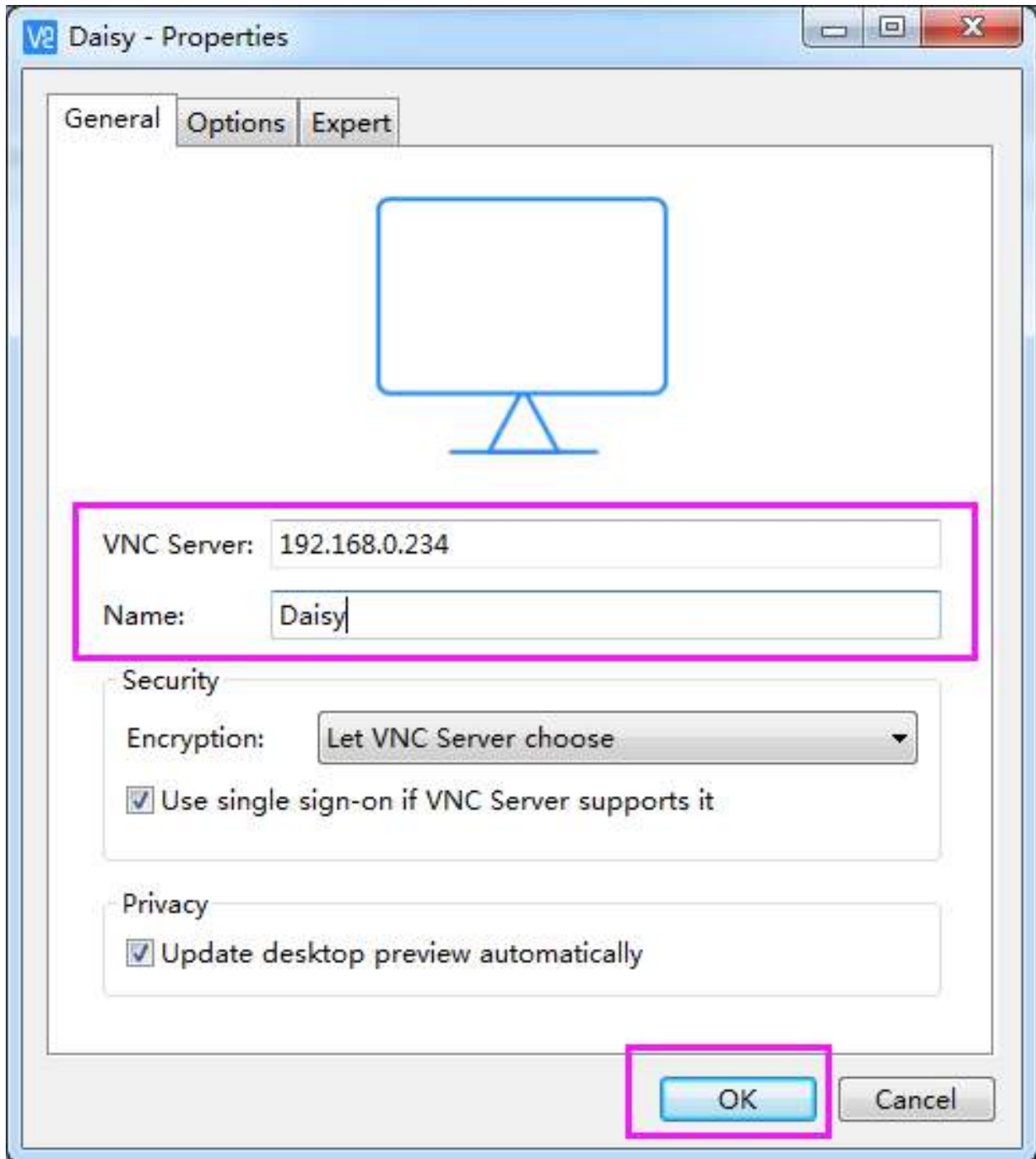
Step 2

Then select “**New connection**”.



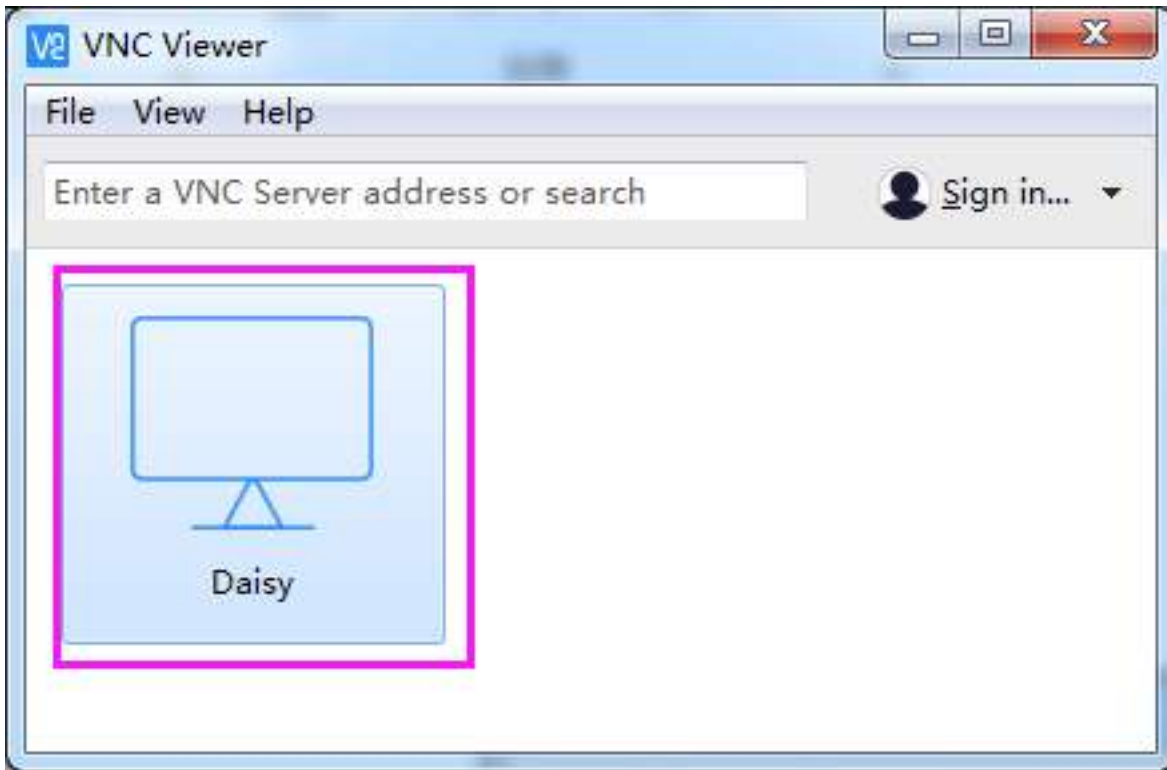
Step 3

Input IP address of Raspberry Pi and any **Name**.



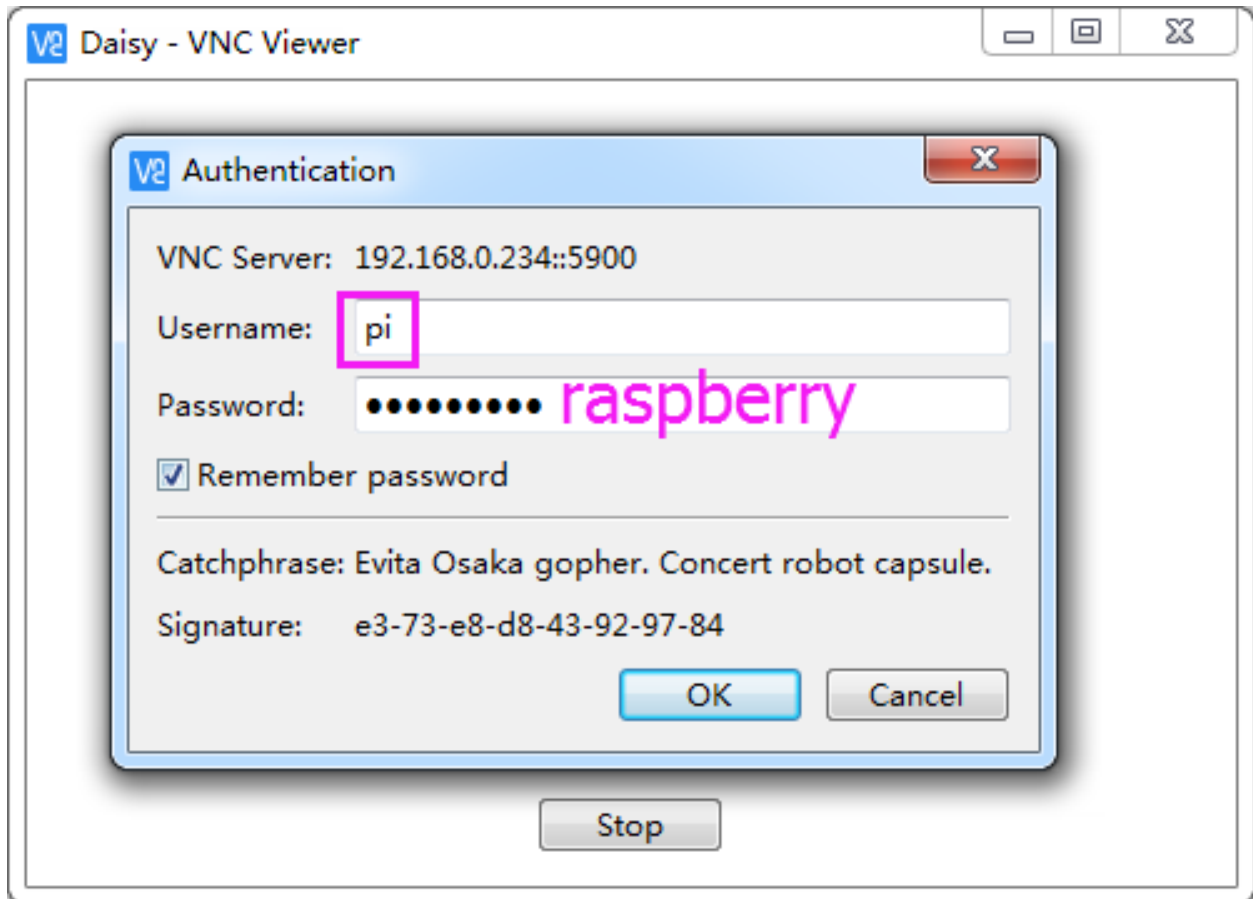
Step 4

Double click the **connection** just created:

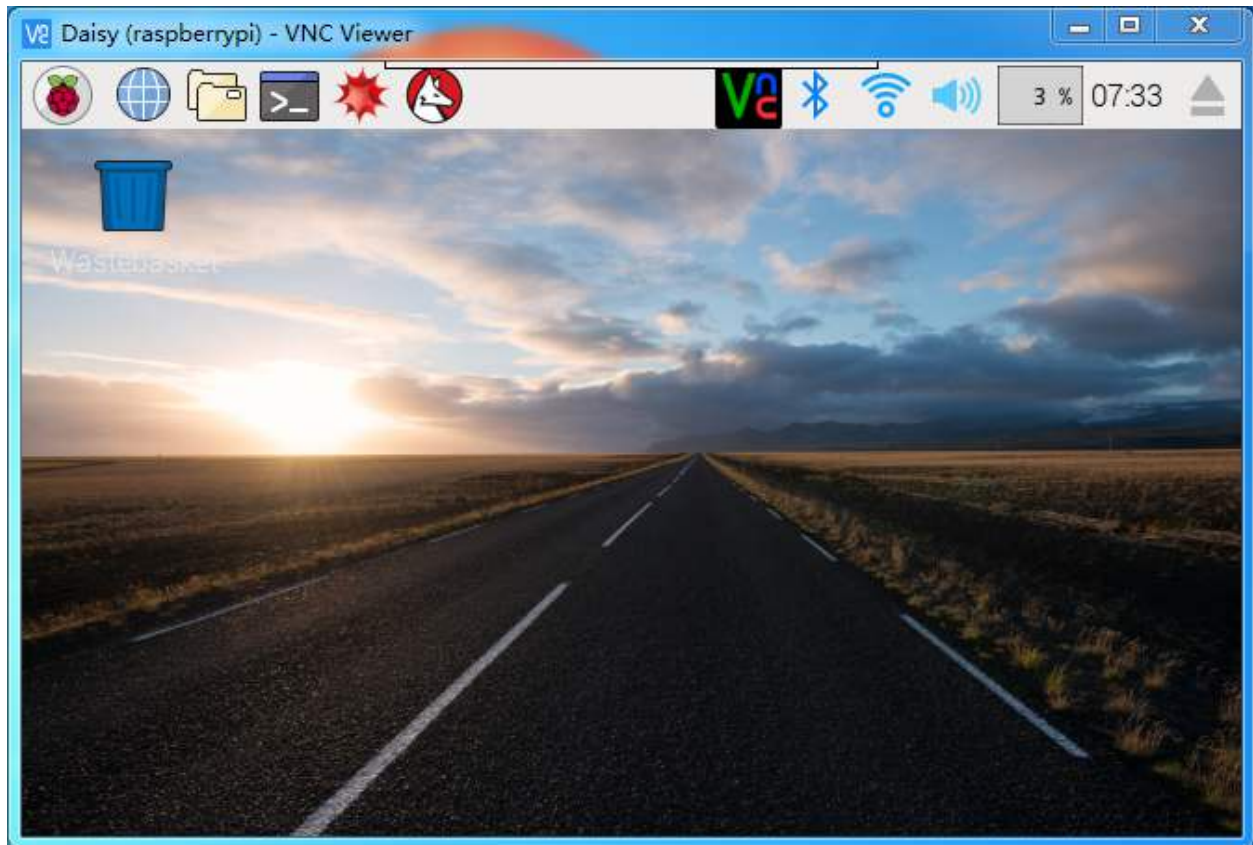


Step 5

Enter Username (**pi**) and Password (**raspberrypi** by default).

**Step 6**

Now you can see the desktop of the Raspberry Pi:



That's the end of the VNC part.

5.2.2 XRDP

Another method of remote desktop is XRDP, it provides a graphical login to remote machines using RDP (Microsoft Remote Desktop Protocol).

Install XRDP

Step 1

Login to Raspberry Pi by using SSH.

Step 2

Input the following instructions to install XRDP.

```
sudo apt-get update
sudo apt-get install xrdp
```

Step 3

Later, the installation starts.

Enter “Y”, press key “Enter” to confirm.


```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo apt-get install xrdp
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
Suggested packages:
  vnc-java mesa-utils x11-xfs-utils
The following NEW packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
  xrdp
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,468 kB of archives.
After this operation, 17.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

Step 4

Finished the installation, you should login to your Raspberry Pi by using Windows remote desktop applications.

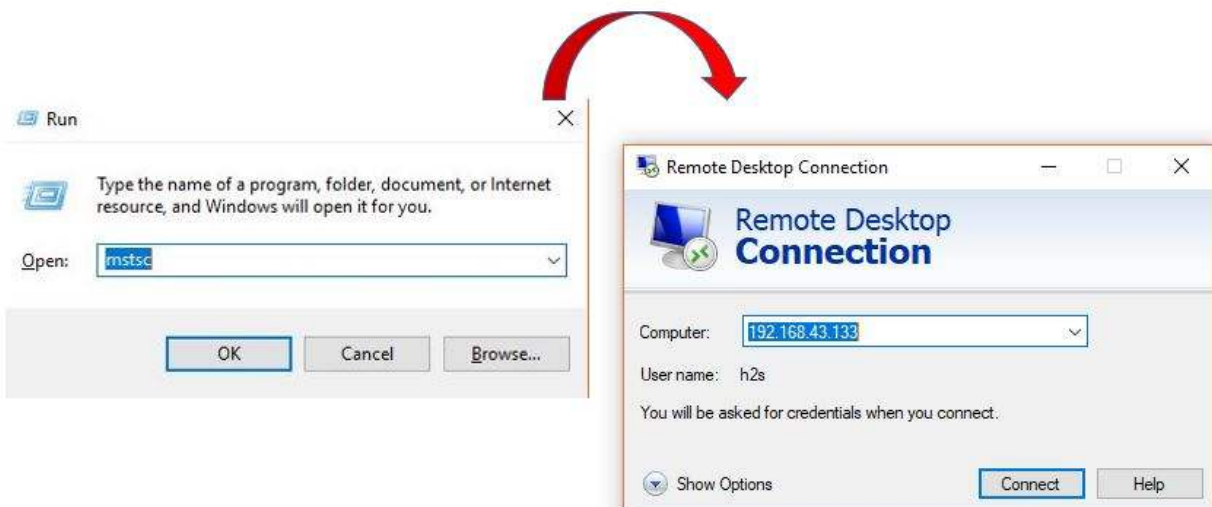
Login to XRDP

Step 1

If you are a Windows user, you can use the Remote Desktop feature that comes with Windows. If you are a Mac user, you can download and use Microsoft Remote Desktop from the APP Store, and there is not much difference between the two. The next example is Windows remote desktop.

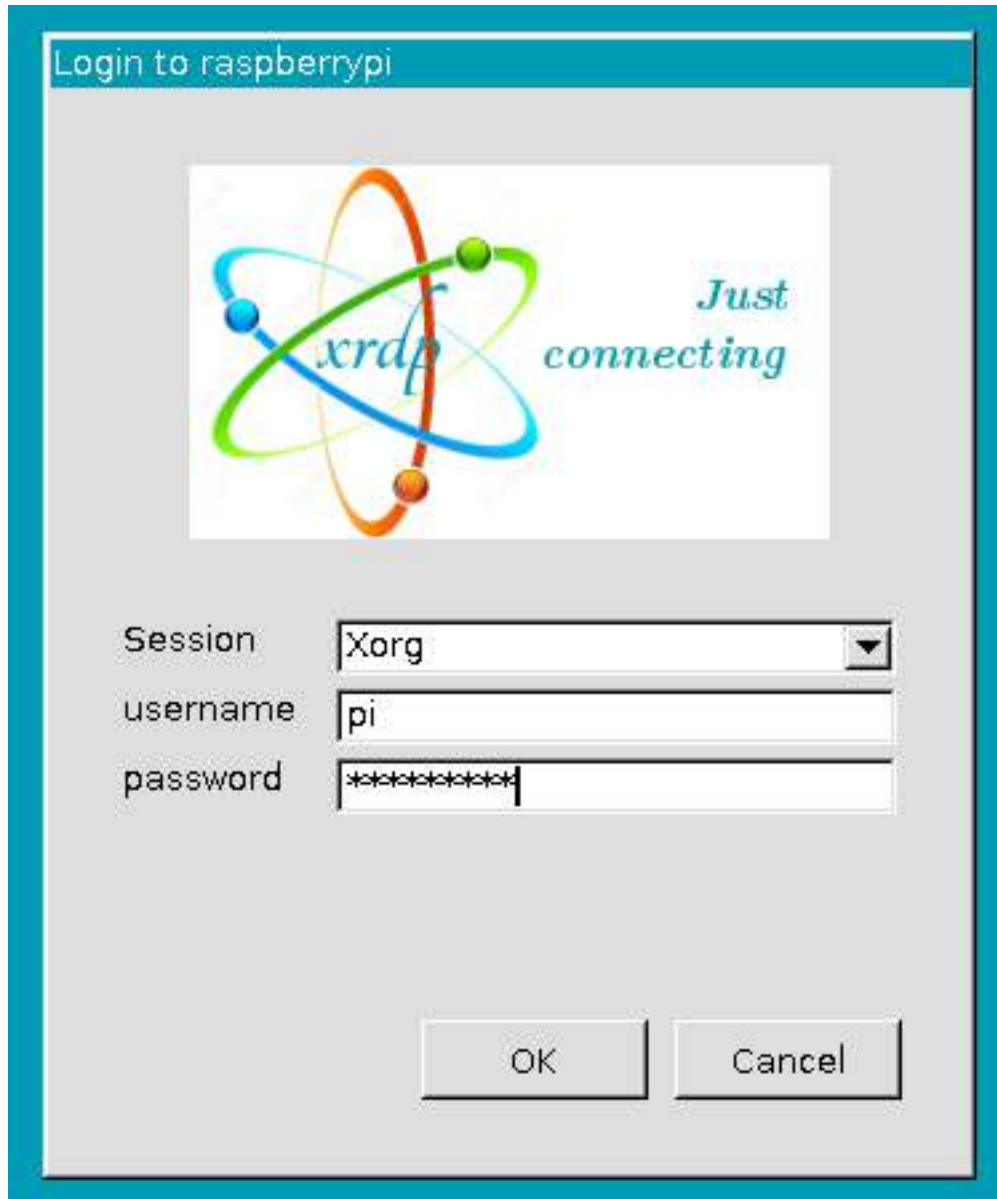
Step 2

Type in “mstsc” in Run (WIN+R) to open the Remote Desktop Connection, and input the IP address of Raspberry Pi, then click on “Connect”.



Step 3

Then the xrdp login page pops out. Please type in your username and password. After that, please click “OK”. At the first time you log in, your username is “pi” and the password is “raspberrypi”.



Step 4

Here, you successfully login to RPi by using the remote desktop.



COPYRIGHT NOTICE

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.