MIT AI2 204

IoT with MIT App Inventor

Fundamental

Xincheng Tang

Python Basics

- Comments, literal constants, numbers, quotes.
- Operators and Expressions
- Control Flows
- Functions
- Modules

https://automatetheboringstuff.com/2e/chapter1/

https://automatetheboringstuff.com/2e/chapter2/

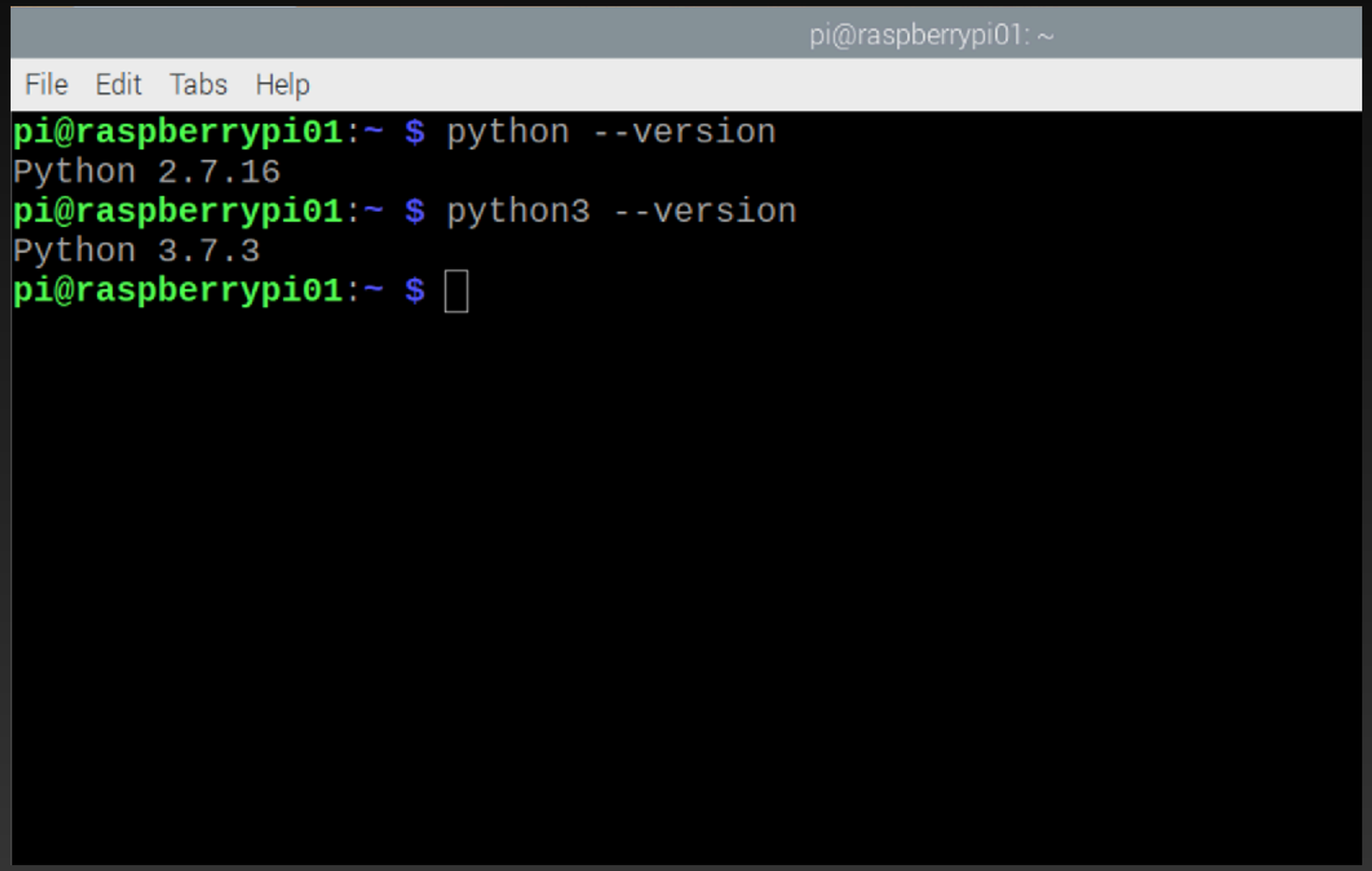
Python Programming Language

Check Python version

Two versions of Python are distributed with Raspberry Pi. Version 2 and 3. To display them, use command in terminal:

```
python --version
```

```
python3 --version
```



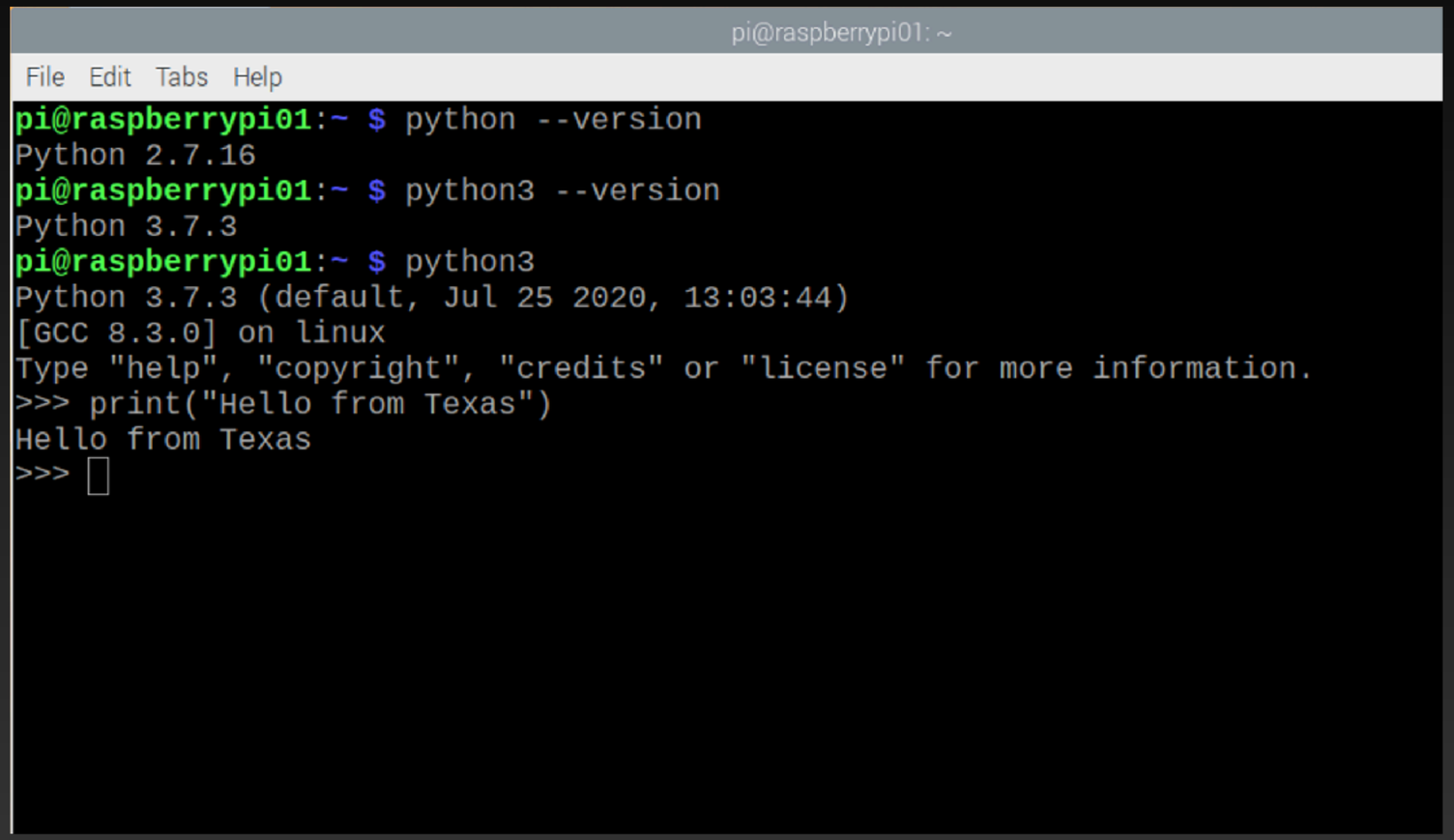
Python Programming Language

Run python program interactively.

In the terminal, type in command: `python3`

Type the program:

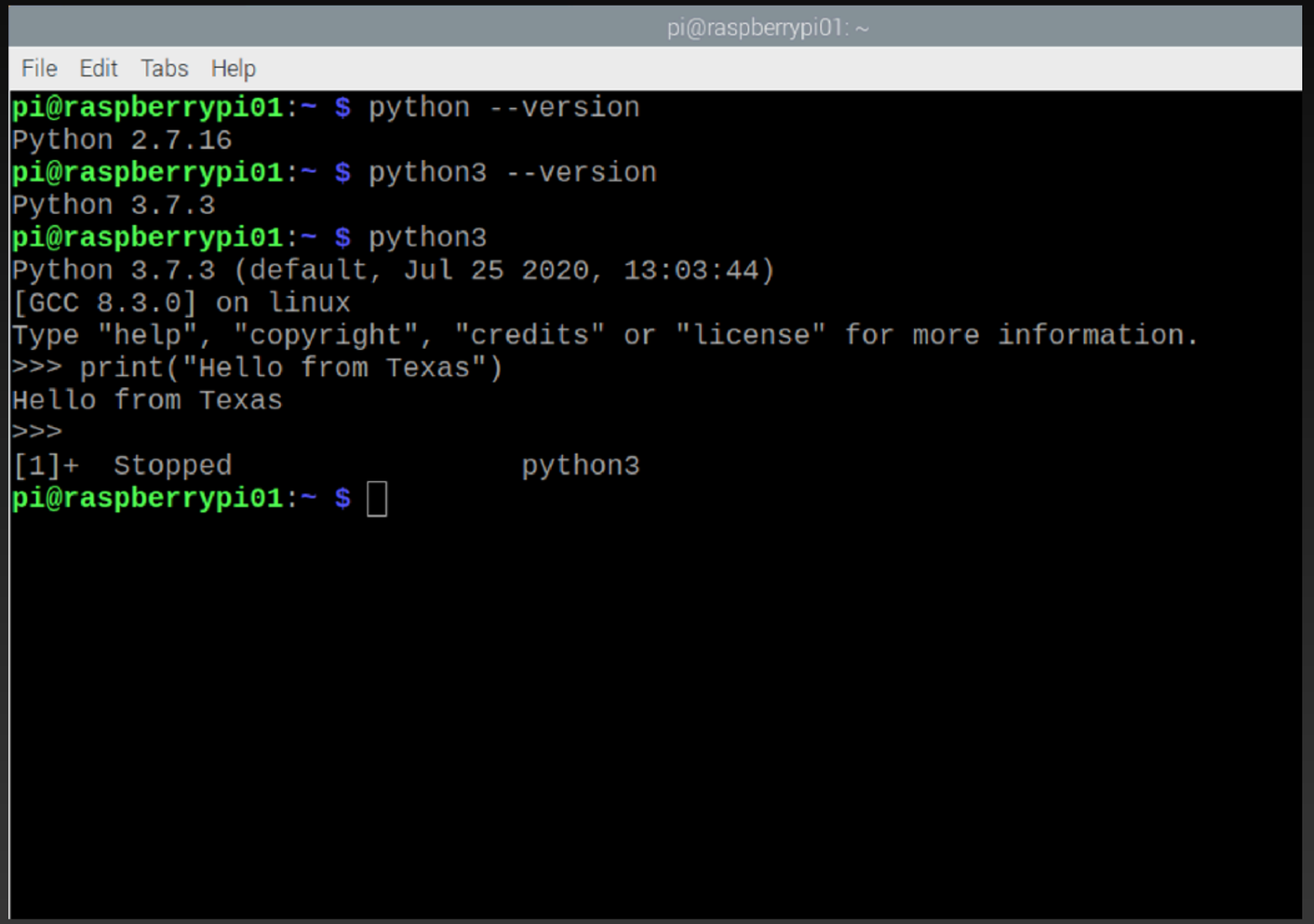
```
print("Hello from Texas”)
```



Python Programming Language

Enter control + Z to exit python

Notice that by default, entering command python invokes version 2.7. If you wish to use version 3.7 then you should enter command python3

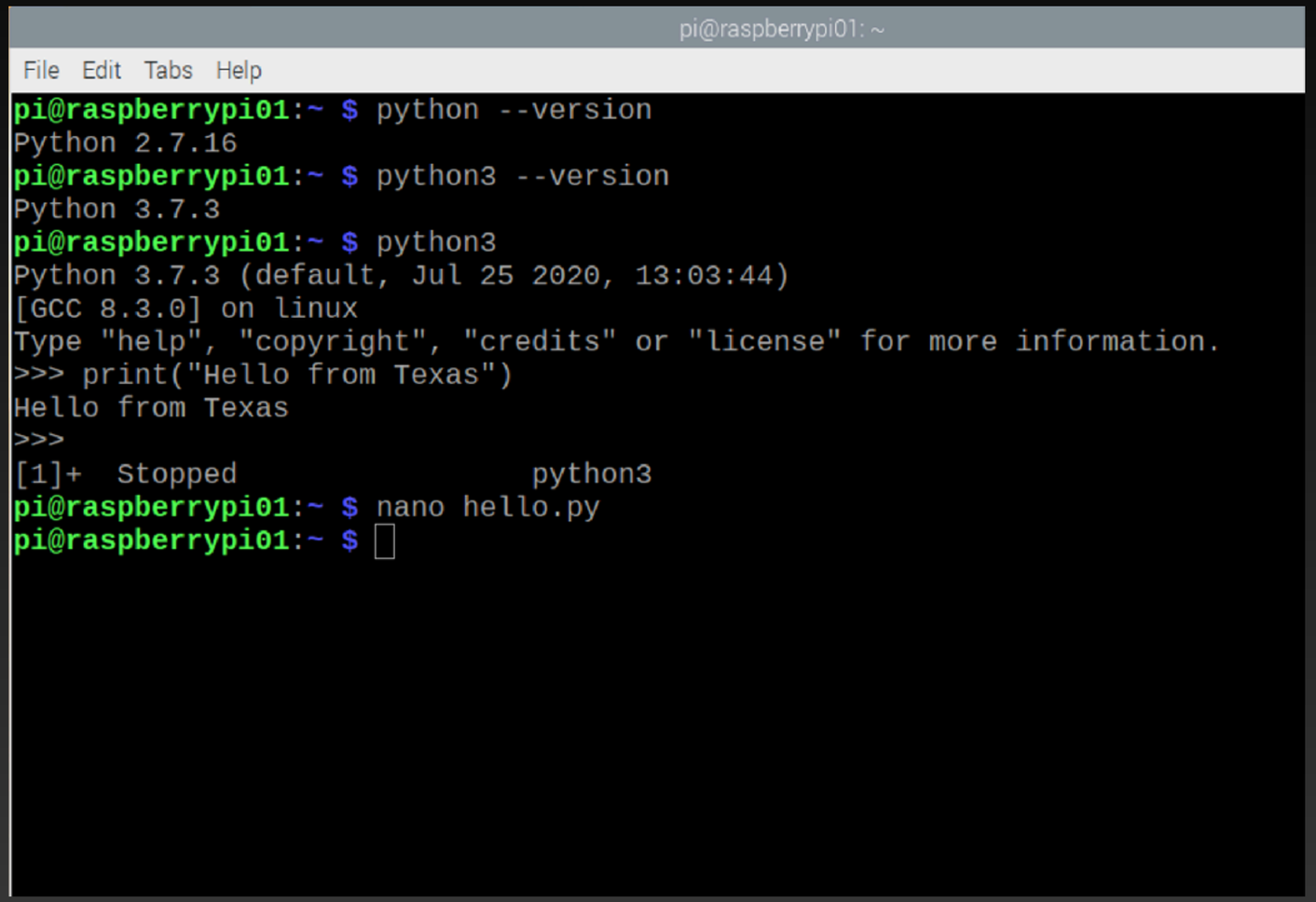


Creating Python File in Command Mode

A python file is simply a text file with the extension .py

We can use a text editor, e.g. the nano text editor to create our file.

Now we are going to create a file called hello.py using nano text editor.

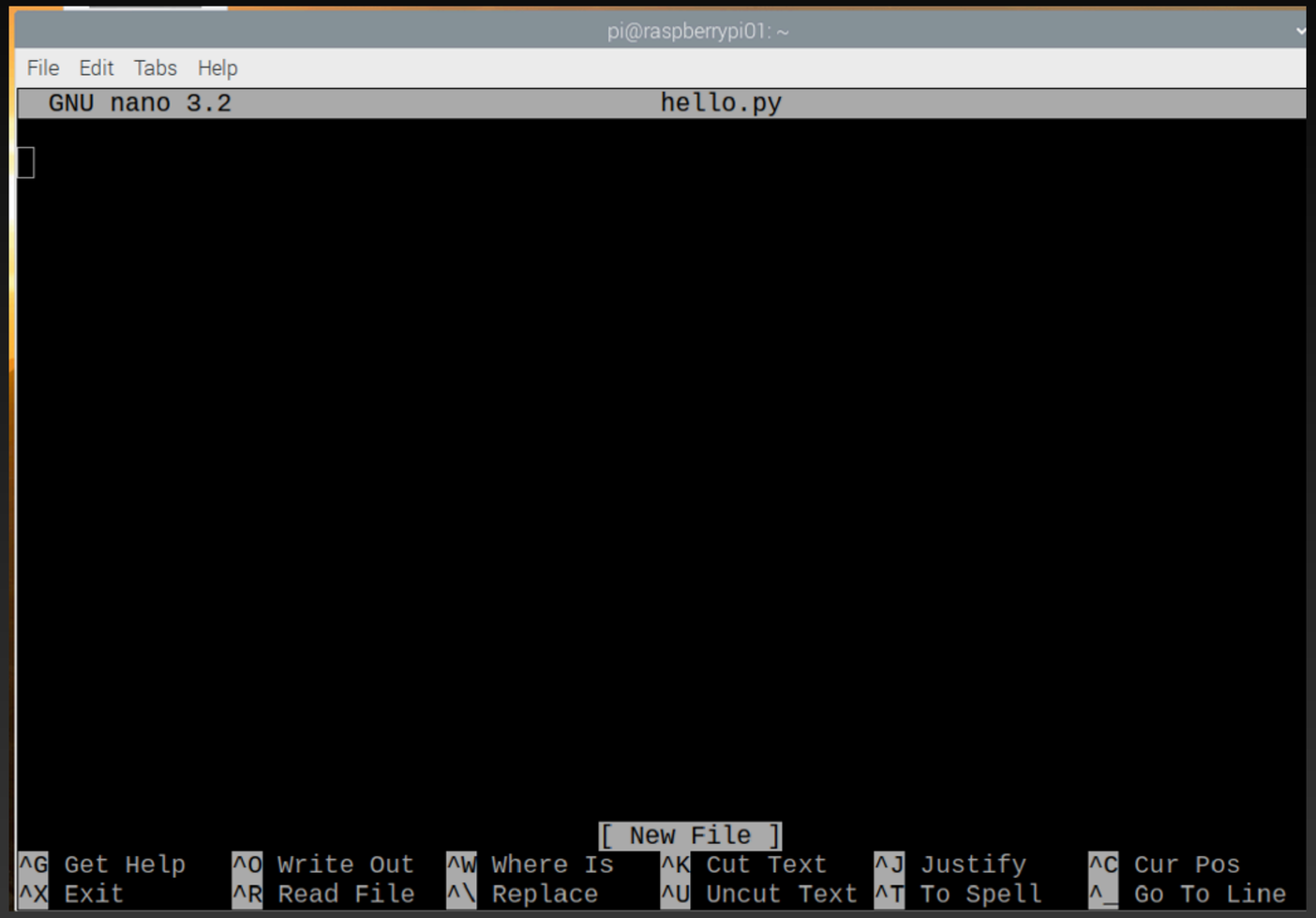


Creating Python File in Command Mode

Type command in terminal:

```
nano hello.py
```

Press enter, you will see the nano text editor screen



Creating Python File in Command Mode

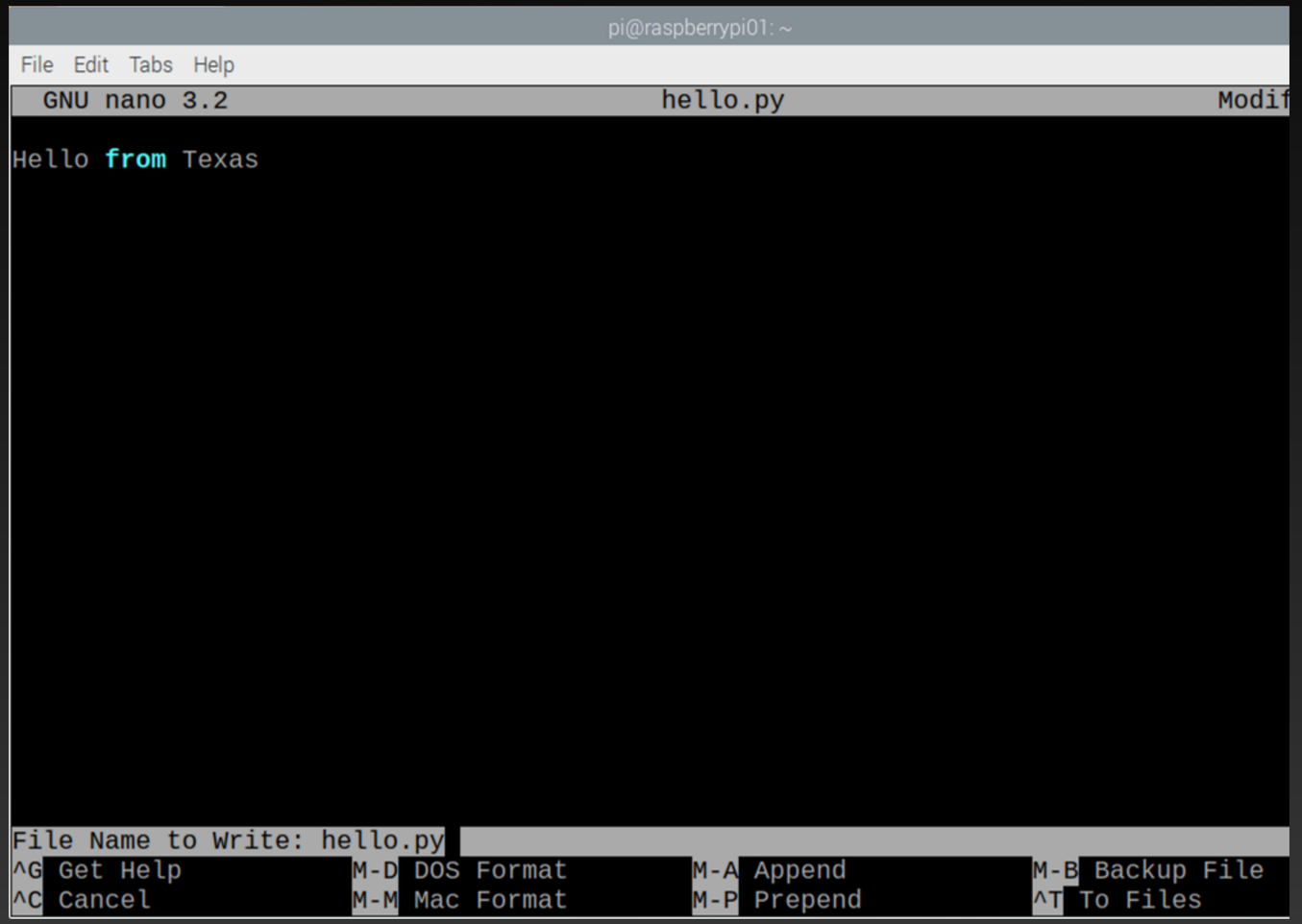
Add some words

Control + O is to save file

Control + X is to exit file

Type Y to accept the saving.

Then press ENTER

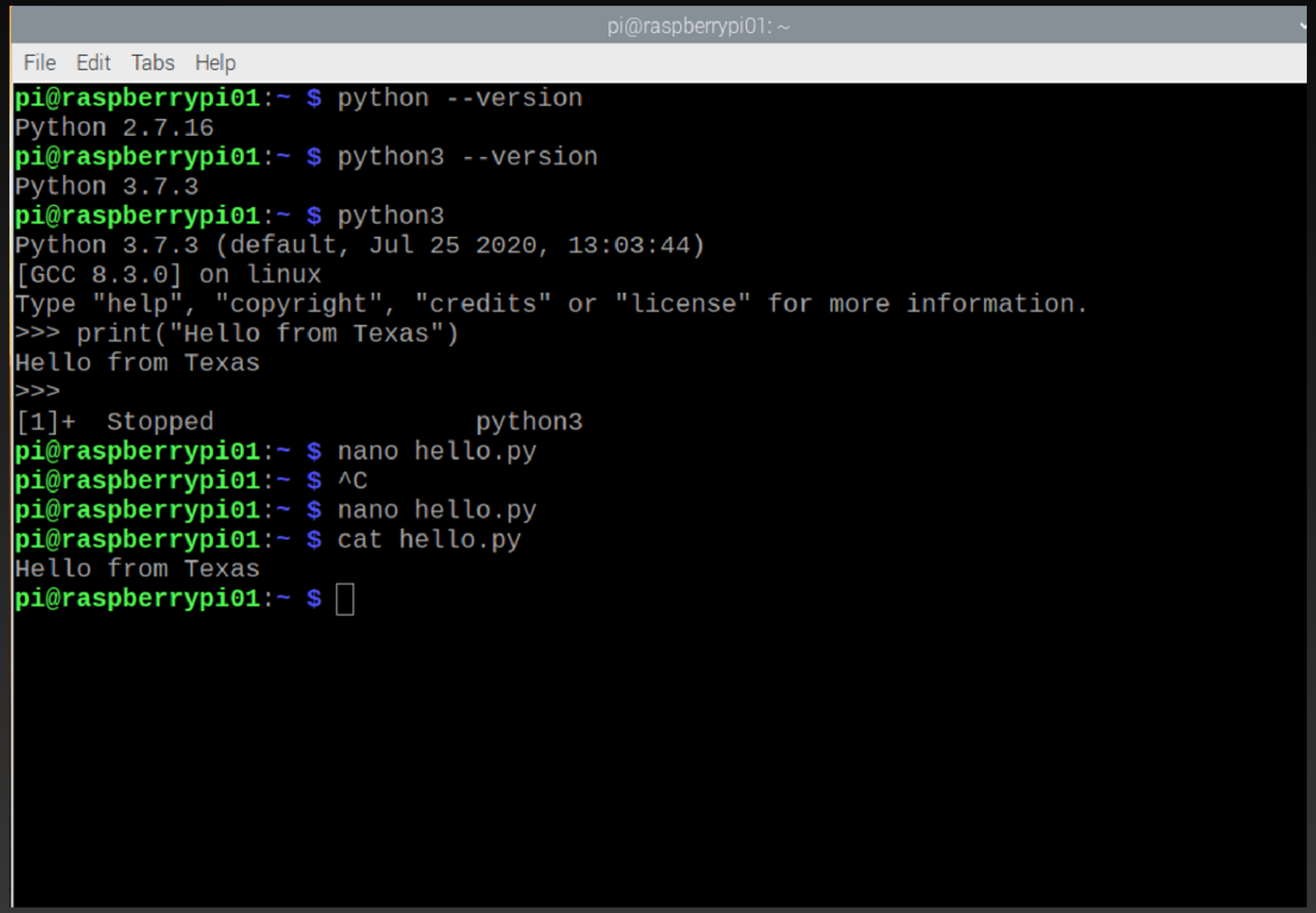


Creating Python File in Command Mode

When you finish previous step, you will back to the terminal screen.

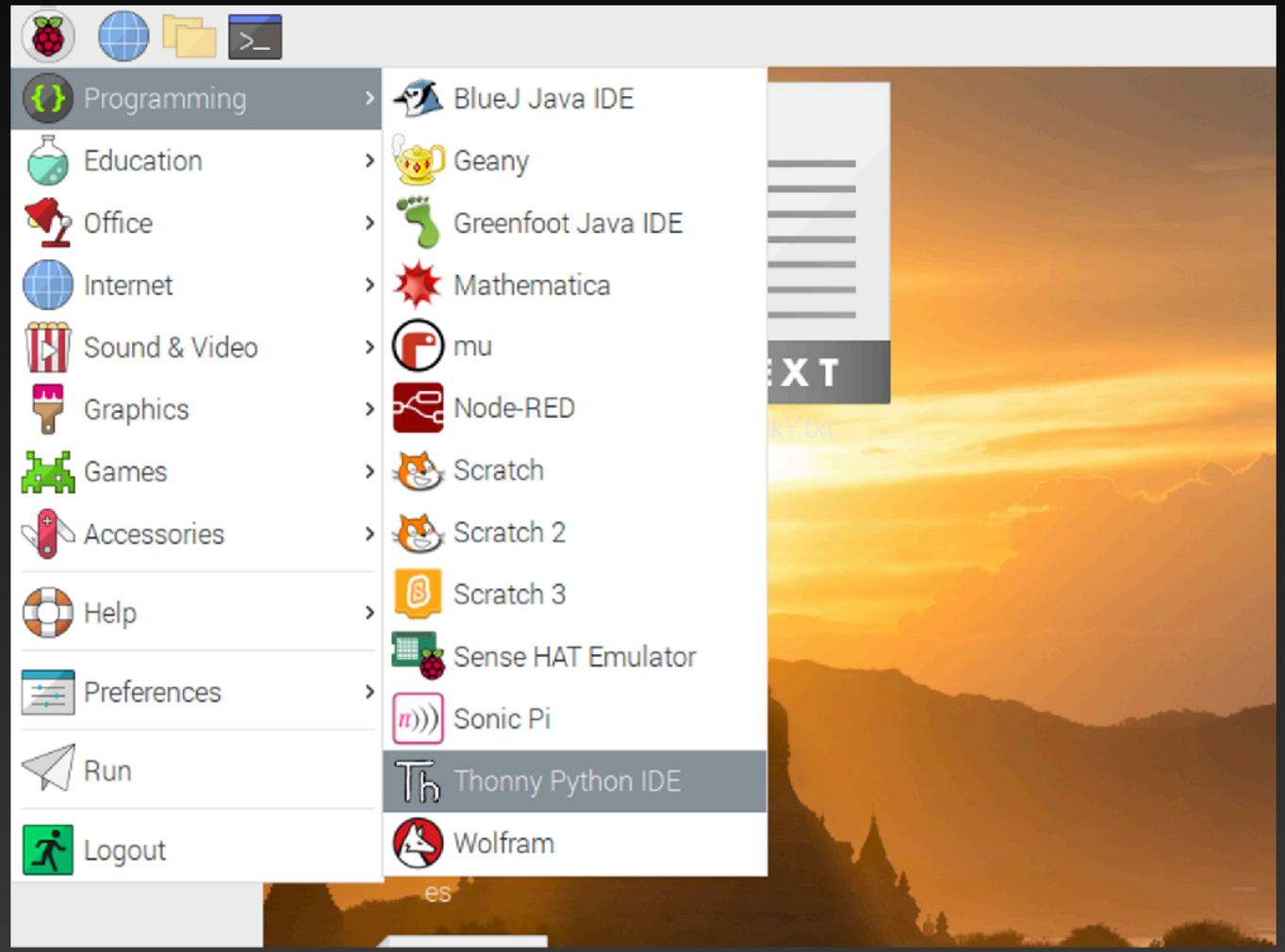
You can display the contents in the file by using command:

```
cat hello.py
```



Creating Python File in GUI - Thonny

Access Thonny from programming dropdown

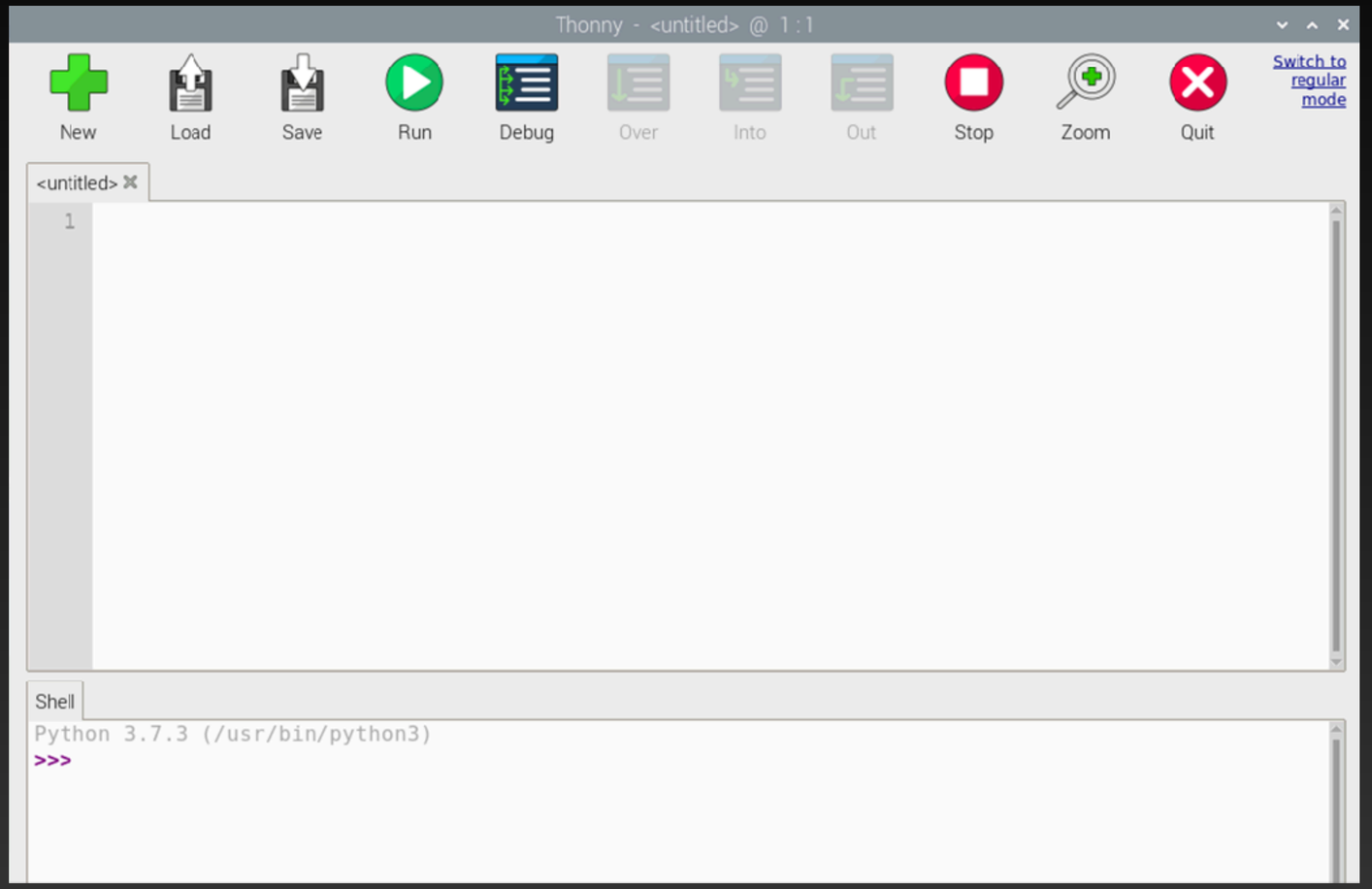


Creating Python File in GUI - Thonny

We can create a file and save it, test it out!

Which method to choose?

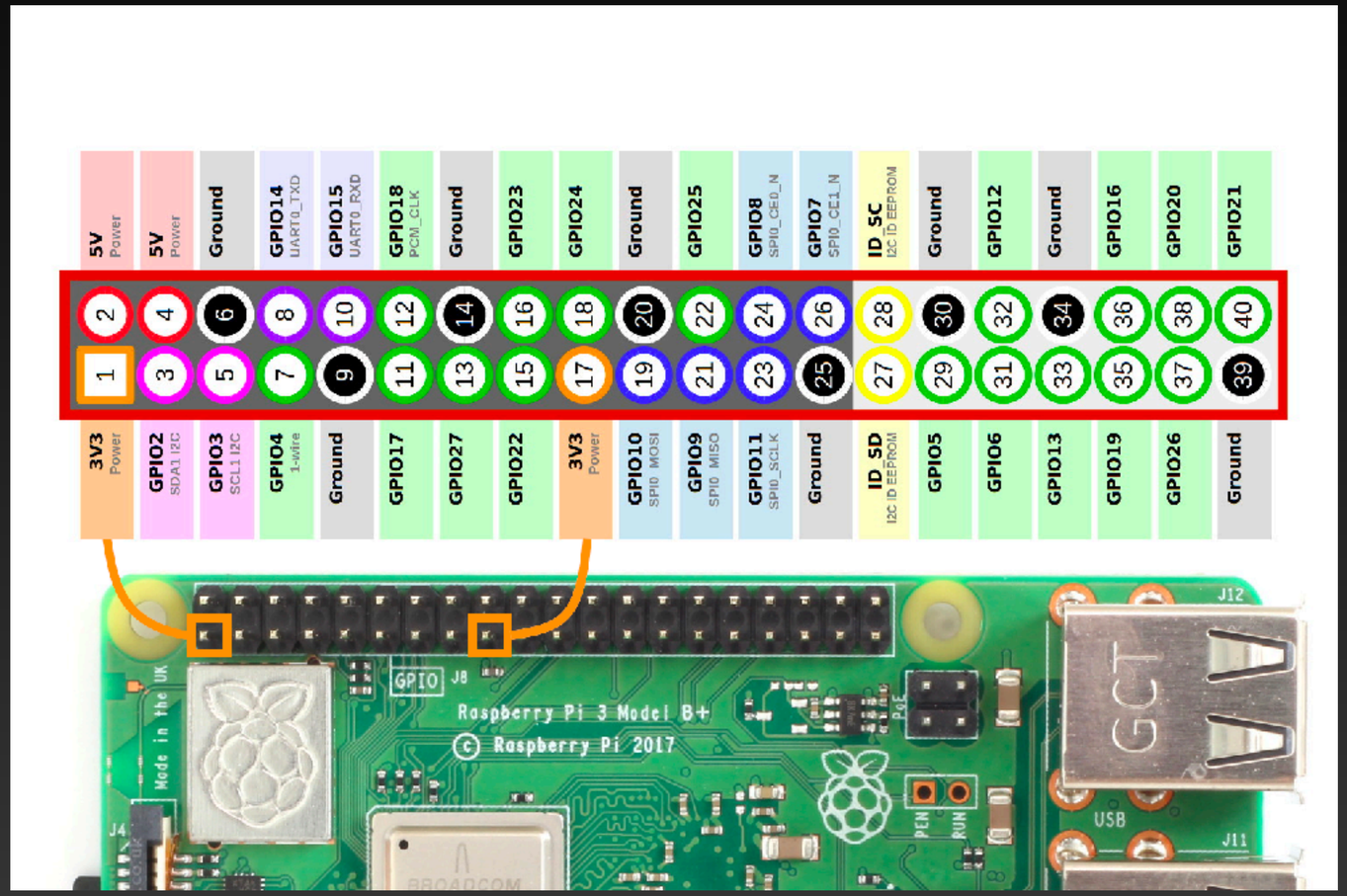
Interactively or program file?



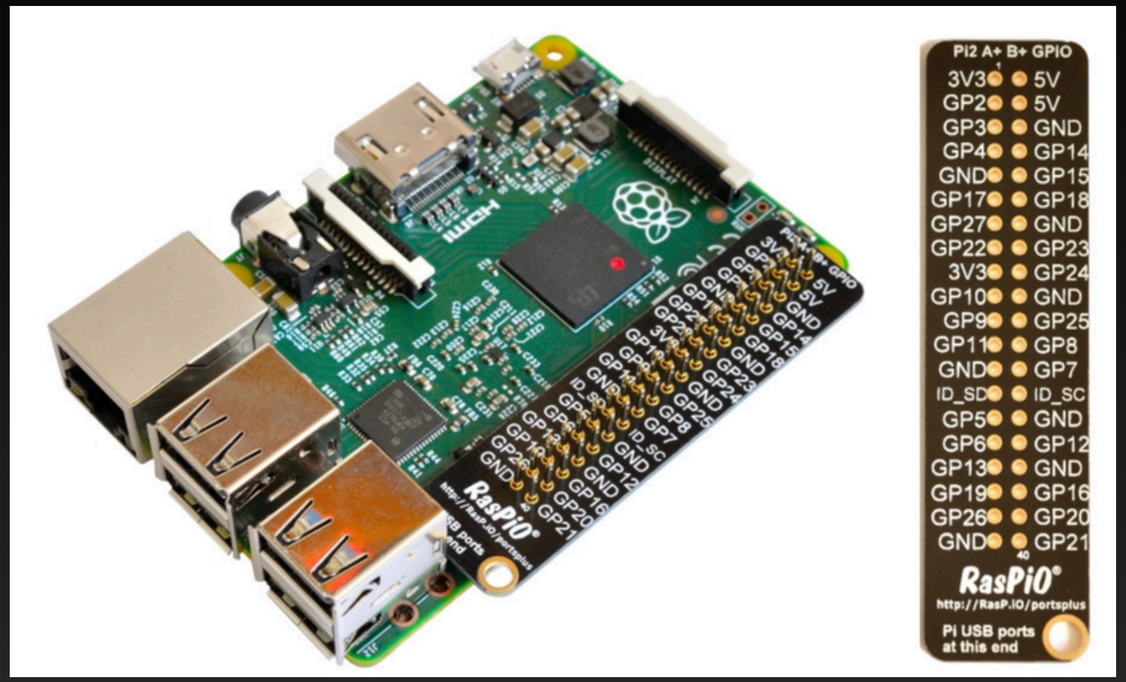
Physical Computing from Python - GPIO

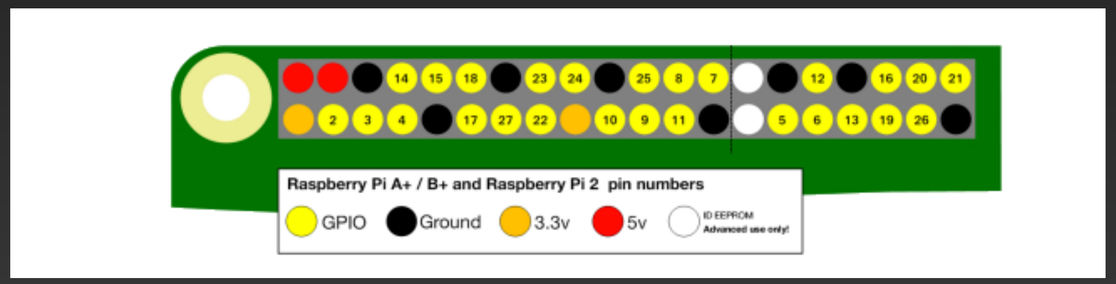
Before going into the details of the hardware interface, we will take a look at the Raspberry Pi 4 GPIO connector.

The Raspberry header is the key to its ability to interface with the real world. The Pi either uses a 40-pin or 26-pin depending on the model and it is important to understand how those pins are arranged and labelled.



Physical Computing from Python - GPIO



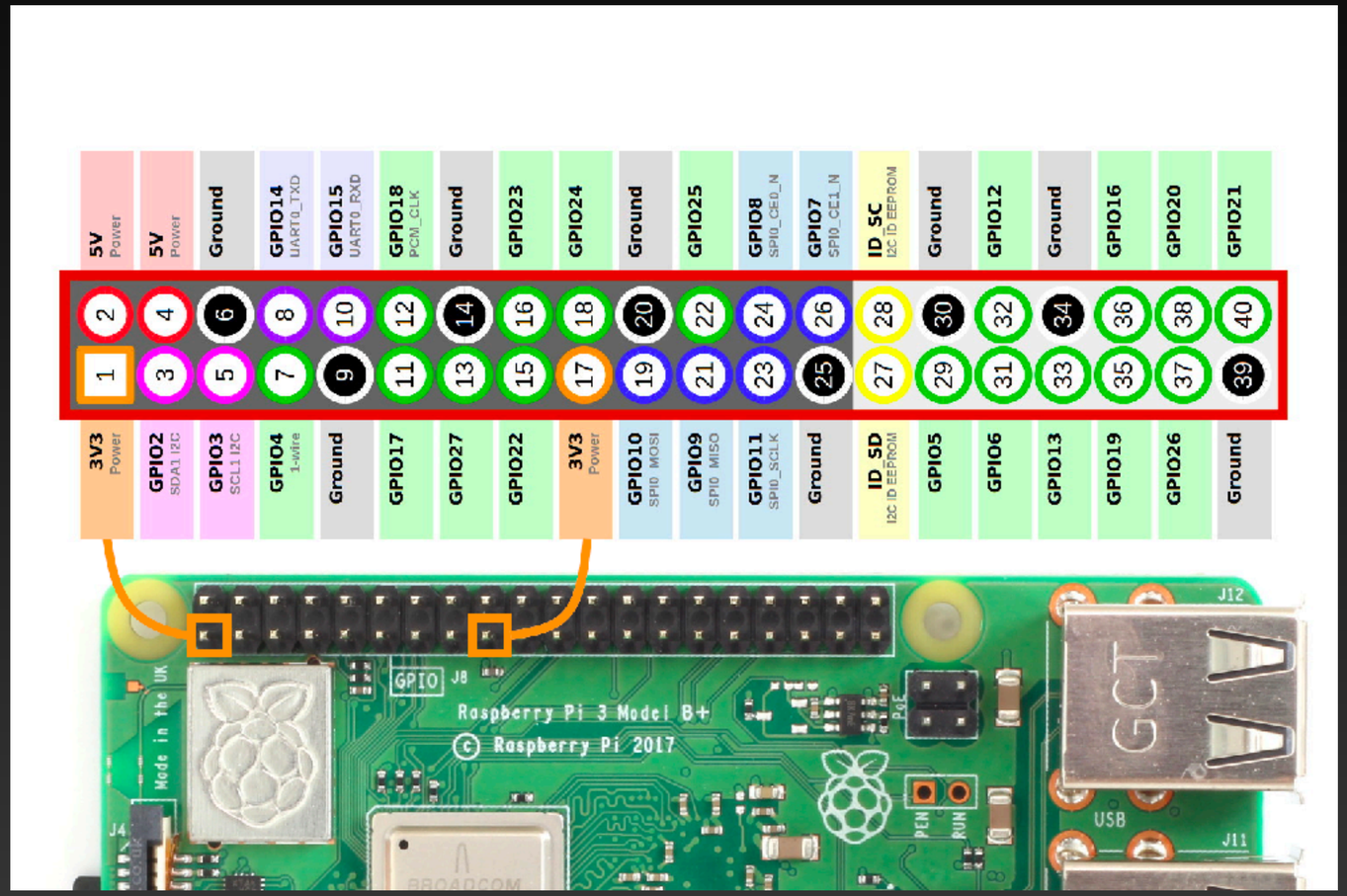


3V3	3.3 volts	Anything connected to these pins will always get 3.3V of power
5V	5 volts	Anything connected to these pins will always get 5V of power
GND	ground	Zero volts, used to complete a circuit
GP2	GPIO pin 2	These pins are for general-purpose use and can be configured as input or output pins
ID_SC/ ID_SD/ DNC	Special purpose pins	

Physical Computing from Python - GPIO

Pin Labels – Clear as Mud

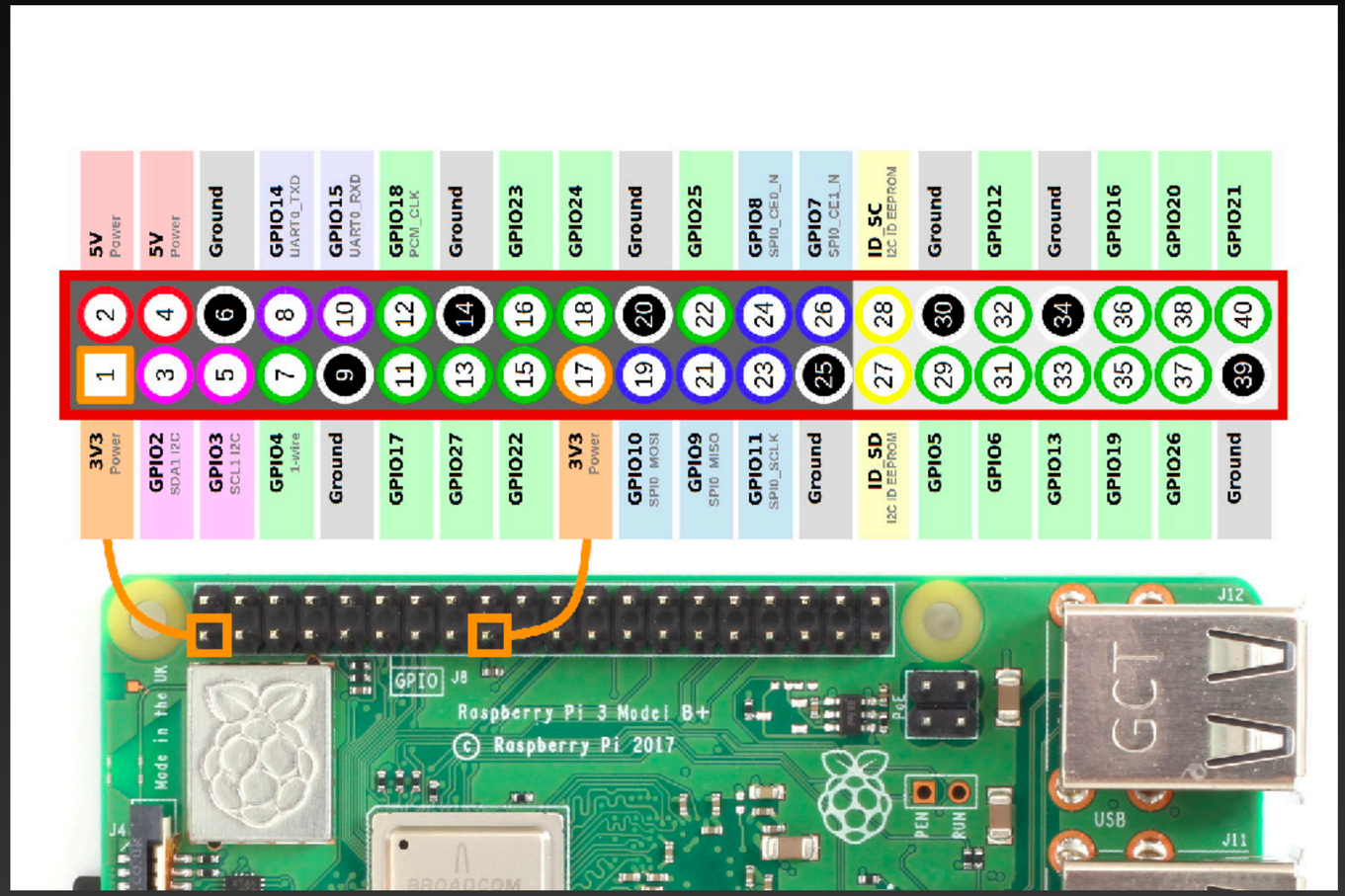
The labels above are the names of the pins on the Broadcom system chip to which the pin is physically connected. Much of the confusion around the GPIO is due to these labels, their relationship to the Broadcom labels and how they are referred to in your programs. To confuse things even more the GPIO pins are sometimes renamed with another set of numbers. In order to avoid damaging your Pi you need to be sure what pins you are connecting to other hardware and that your program is referring to the correct pins.



Physical Computing from Python - GPIO

GPIO Header Power Pins

The header provides 5V on Pin 2 and 3.3V on Pin 1. The 3.3V supply is limited to 50mA. The 5V supply draws current directly from your microUSB supply so can use whatever is left over after the board has taken its share. A 1A power supply could supply up to 300mA once the board has drawn 700mA. Power management has been improved with each iteration of hardware.

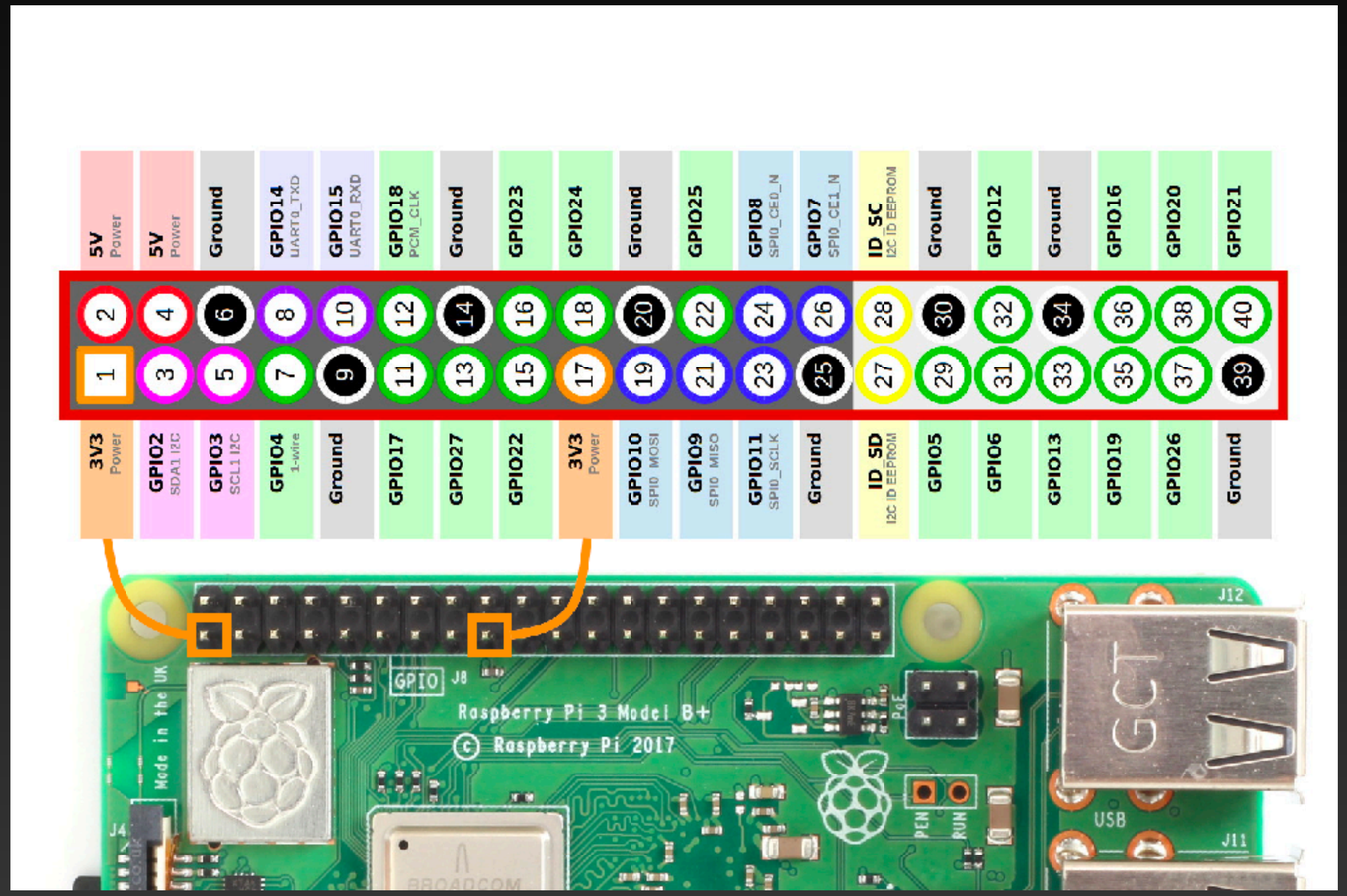


Physical Computing from Python - GPIO

Basic GPIO

The header provides 17 Pins that can be configured as inputs and outputs. By default they are all configured as inputs except GPIO 14 & 15.

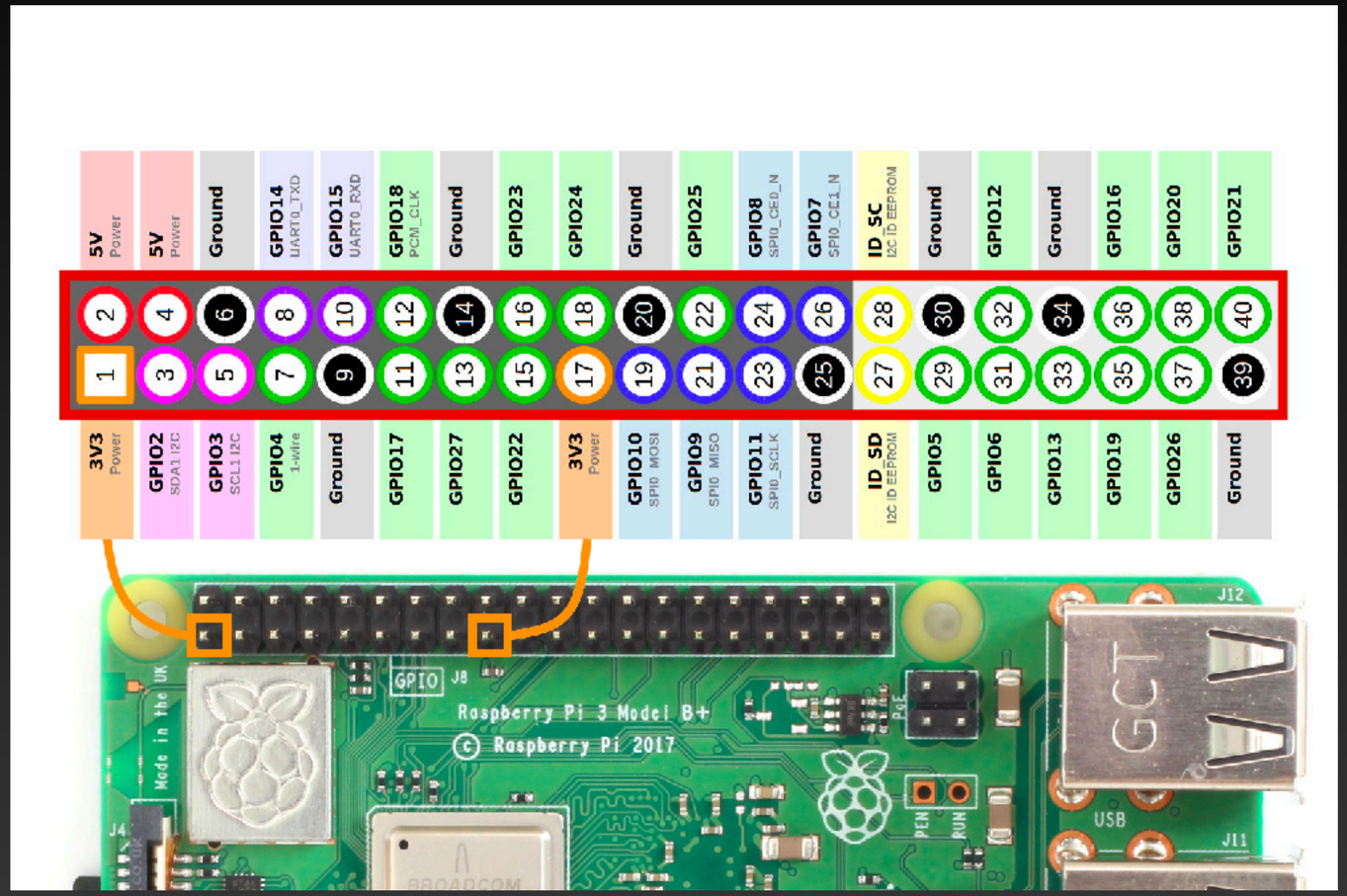
In order to use these pins you must tell the system whether they are inputs or outputs. This can be achieved a number of ways and it depends on how you intend to control them.



Physical Computing from Python - GPIO

Pin Protection

Most of the pins in the header go directly to the Broadcom chip. It is important to carefully design the components you attach to them as there is a risk you will permanently damage your Pi. Short circuits and wiring mistakes could also ruin your day so double check everything. A multimeter is probably going to help a lot here as you can double check wiring before you connect to the Pi.



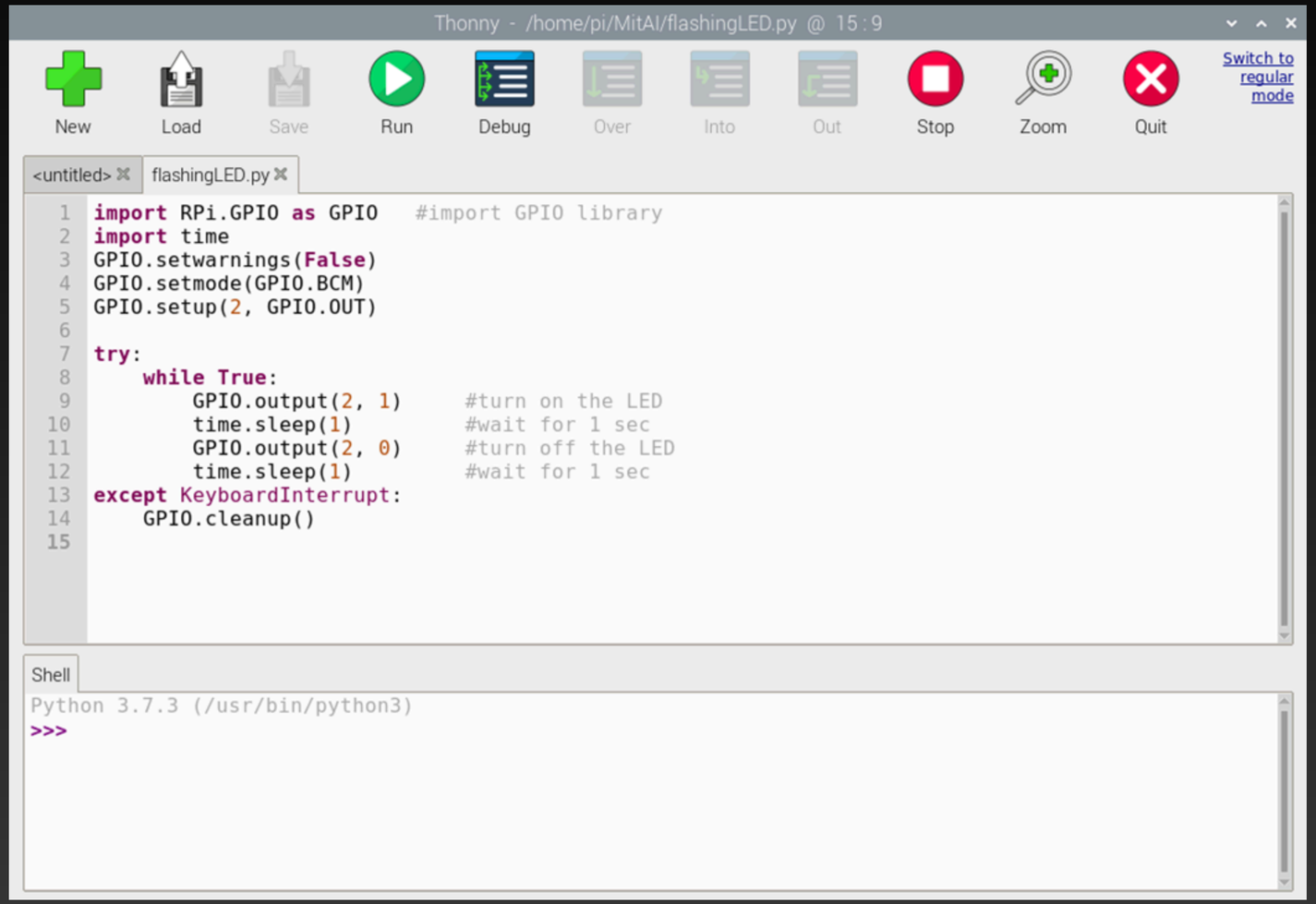
Physical Computing from Python - GPIO Library

The GPIO Library

The GPIO library is called Rip.GPIO and it should already be installed on your raspberry Pi 4.

When you programming in python, this library must be included in your code. The statement is:

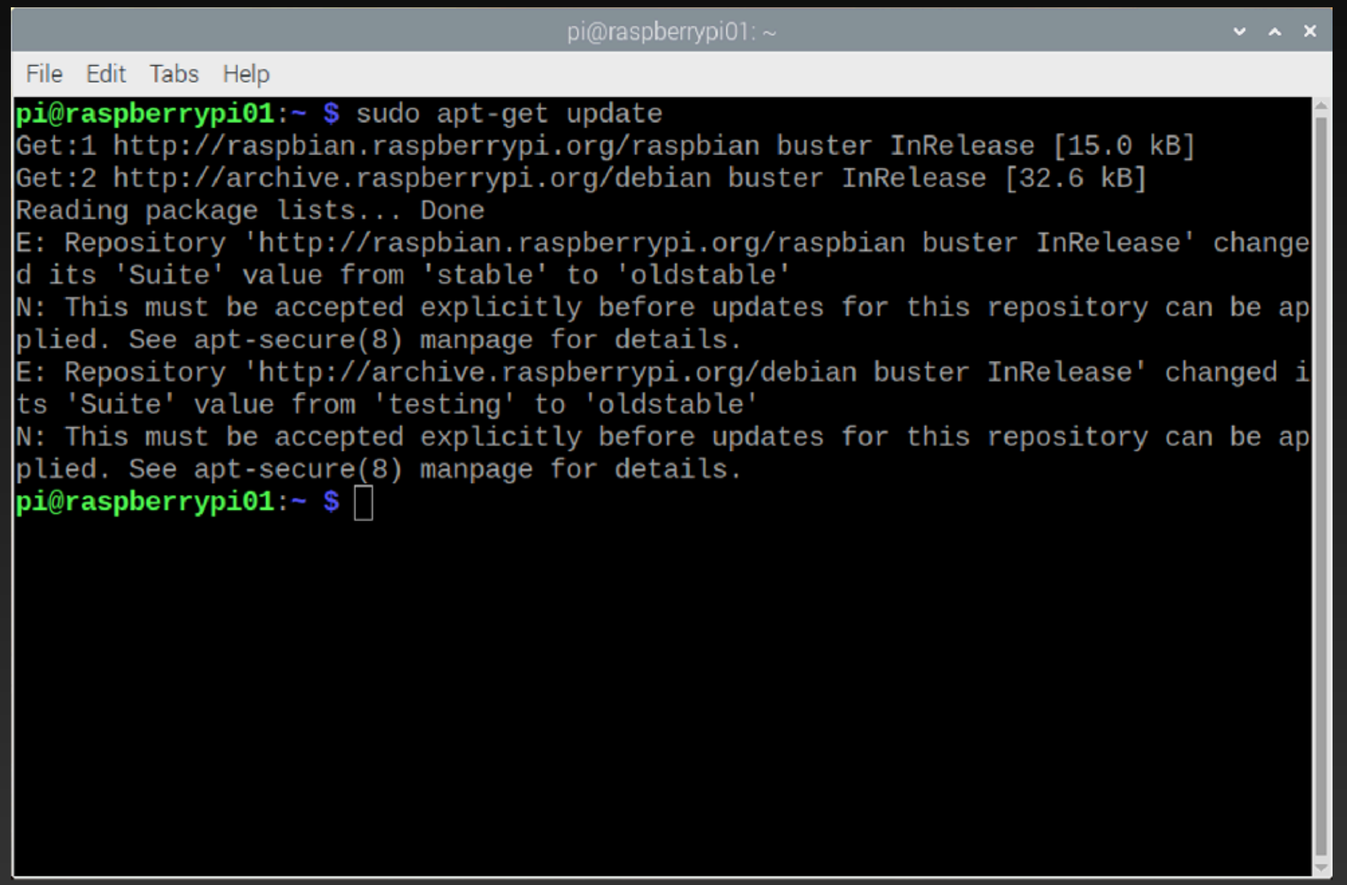
```
import RPi.GPIO as GPIO
```



Physical Computing from Python - GPIO Library

Install GPIO Library

```
sudo apt-get update
```

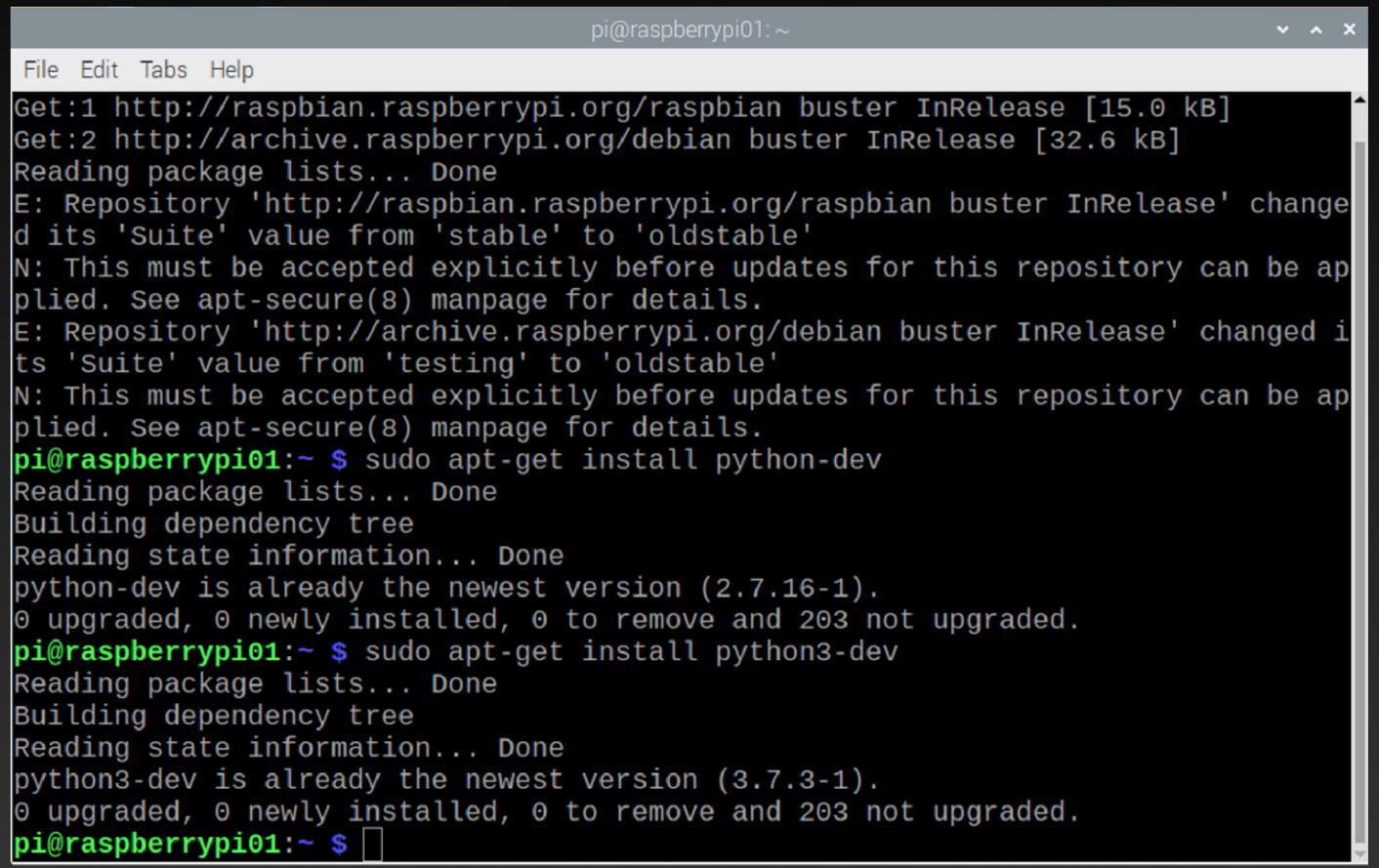


Physical Computing from Python - GPIO Library

Install GPIO Library

```
sudo apt-get install
python-dev
```

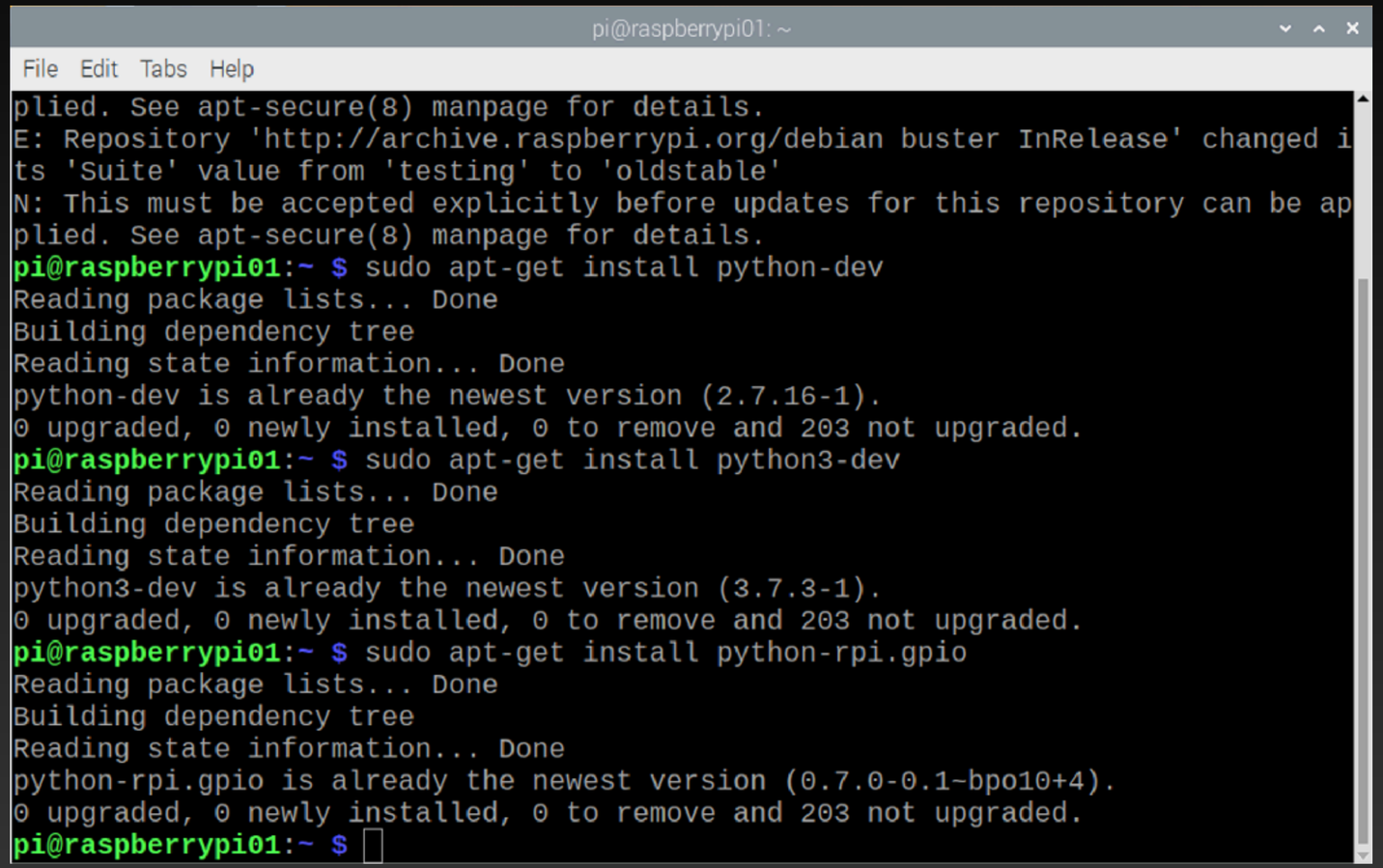
```
sudo apt-get install
python3-dev
```



Physical Computing from Python - GPIO Library

Install GPIO Library

```
sudo apt-get install
python-rpi.gpio
```



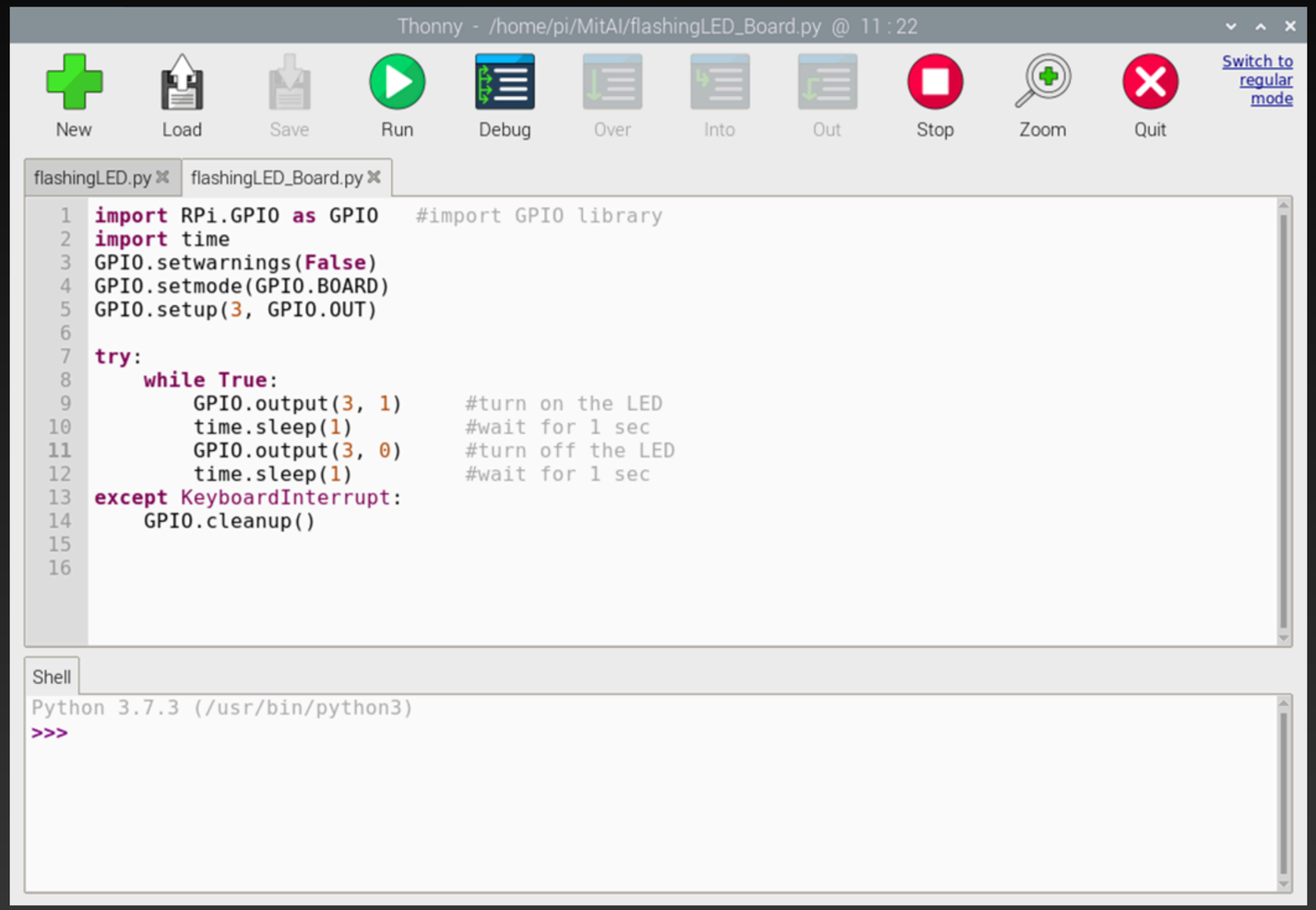
Physical Computing from Python - GPIO Library

Pin Numbering

There are two ways to refer to the GPIO pins.

1. Use the BOARD numbering, where the pin numbers on the GPIO connector of the Raspberry Pi 4 are used.

```
GPIO.setmode(GPIO.BOARD)
```



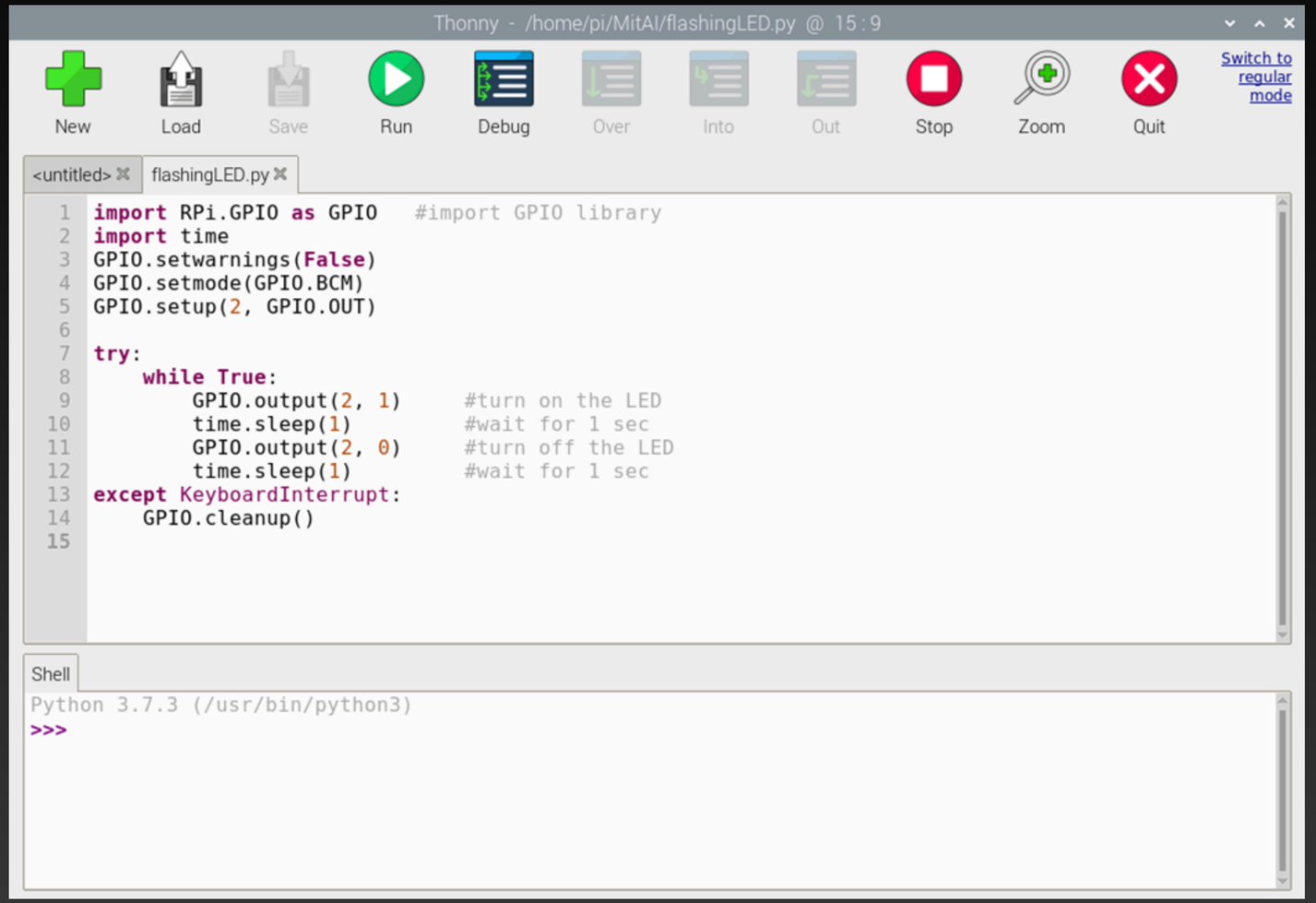
Physical Computing from Python - GPIO Library

Pin Numbering

There are two ways to refer to the GPIO pins.

2. Use the BCM, which is preferred. It uses the channel numbers refers to which pin on the board.

```
GPIO.setmode(GPIO.BCM)
```



Physical Computing from Python - GPIO Library

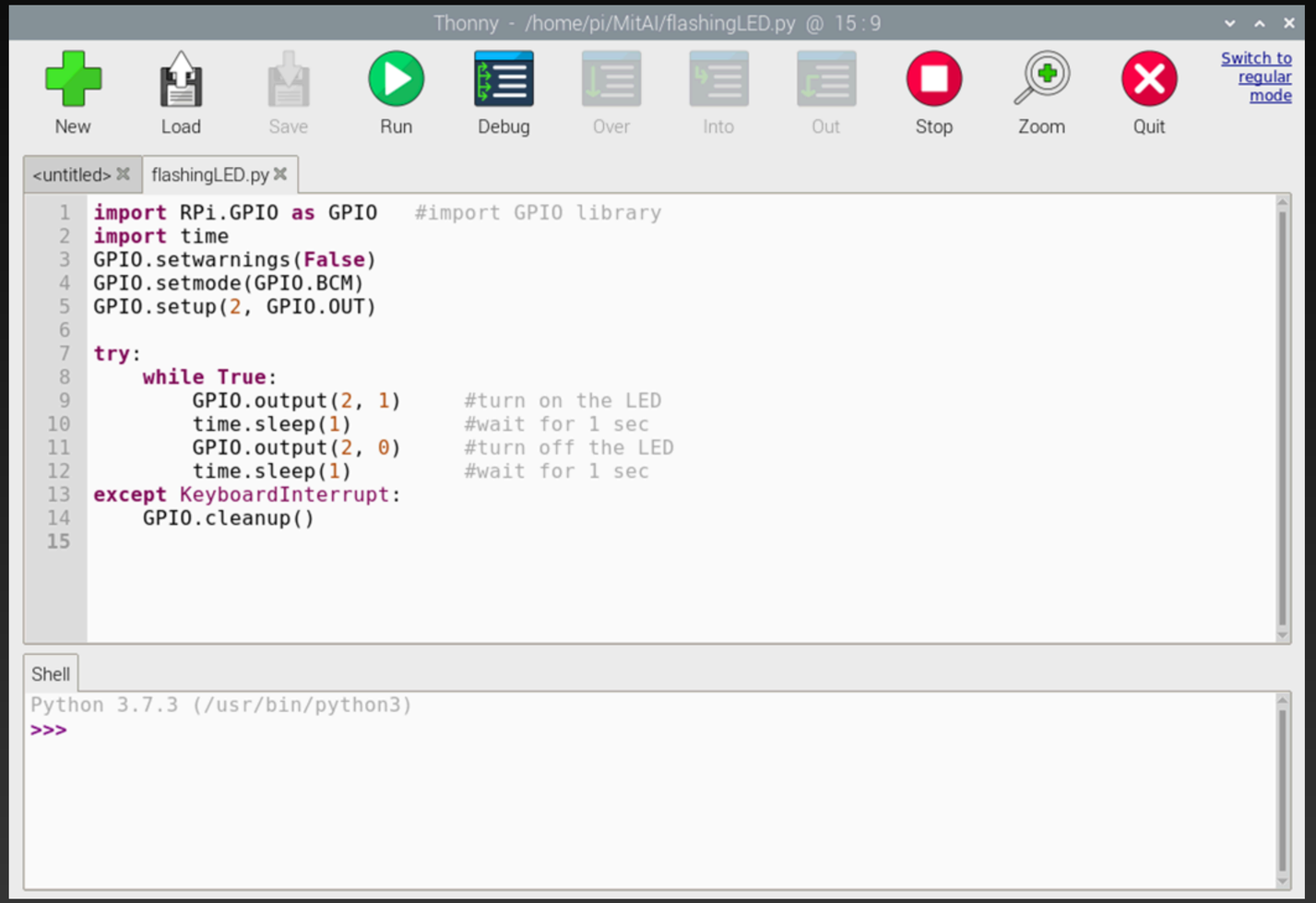
Channel(I/O port pin) Config

Input configuration

You need to configure the channels(or port pins) you are using whether they are input or output channels.

```
GPIO.setup(channel,GPIO.IN)
```

Channel refers to the channel number based on the set mode statement in previous setup.



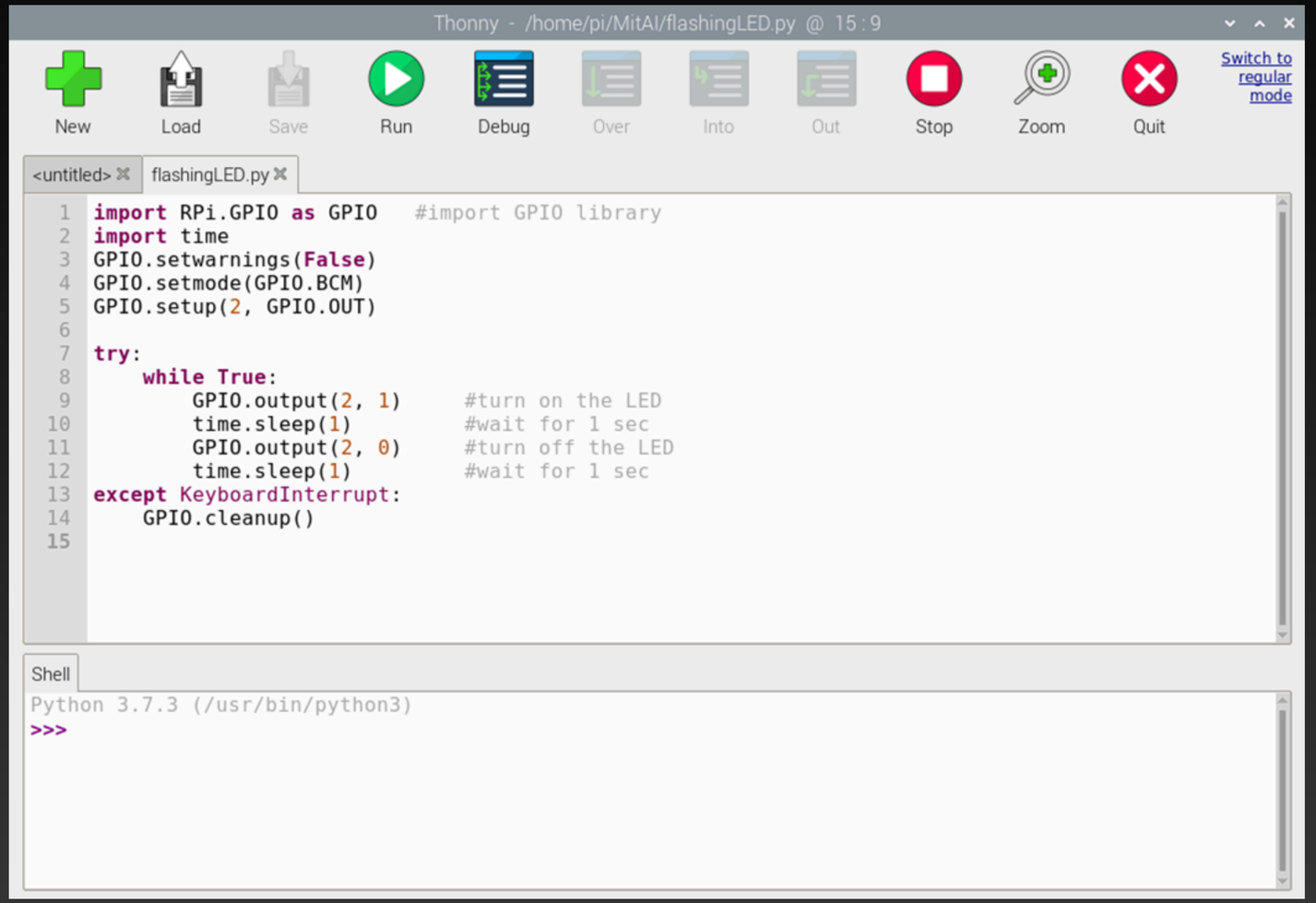
Physical Computing from Python - GPIO Library

Channel(I/O port pin) Config

Output configuration

```
GPIO.setup(channel,GPIO.OUT)
```

Channel refers to the channel number based on the set mode statement in previous setup.



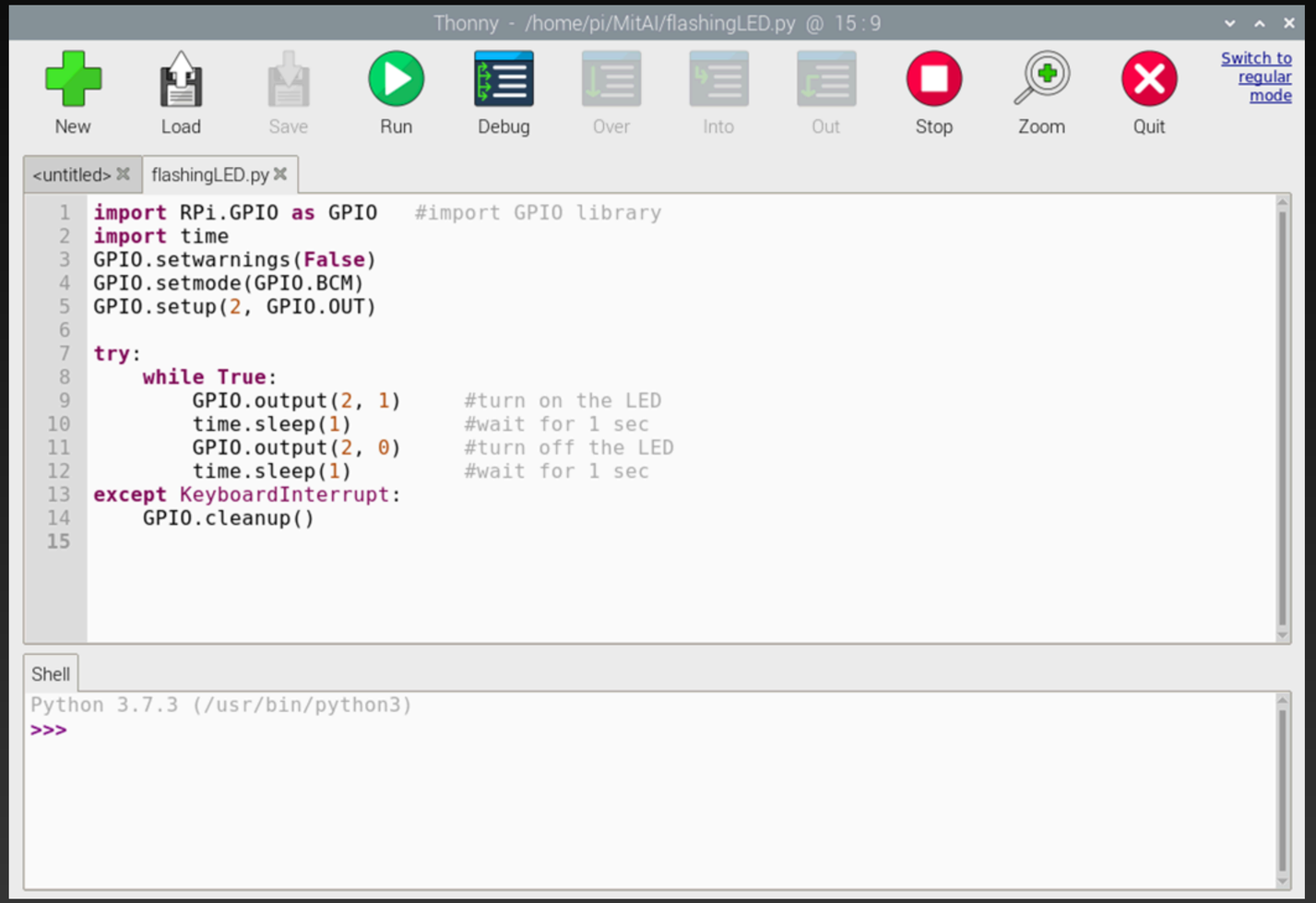
Physical Computing from Python - GPIO Library

Channel(I/O port pin) Config

Output configuration

We can specify a value for an output pin during it's setup. For example, we can config a channel as output and at the same time set its value to logic HIGH(3.3V)

```
GPIO.setup(channel,GPIO.OUT
, initial=GPIO.HIGH)
```



Physical Computing from Python - GPIO Library

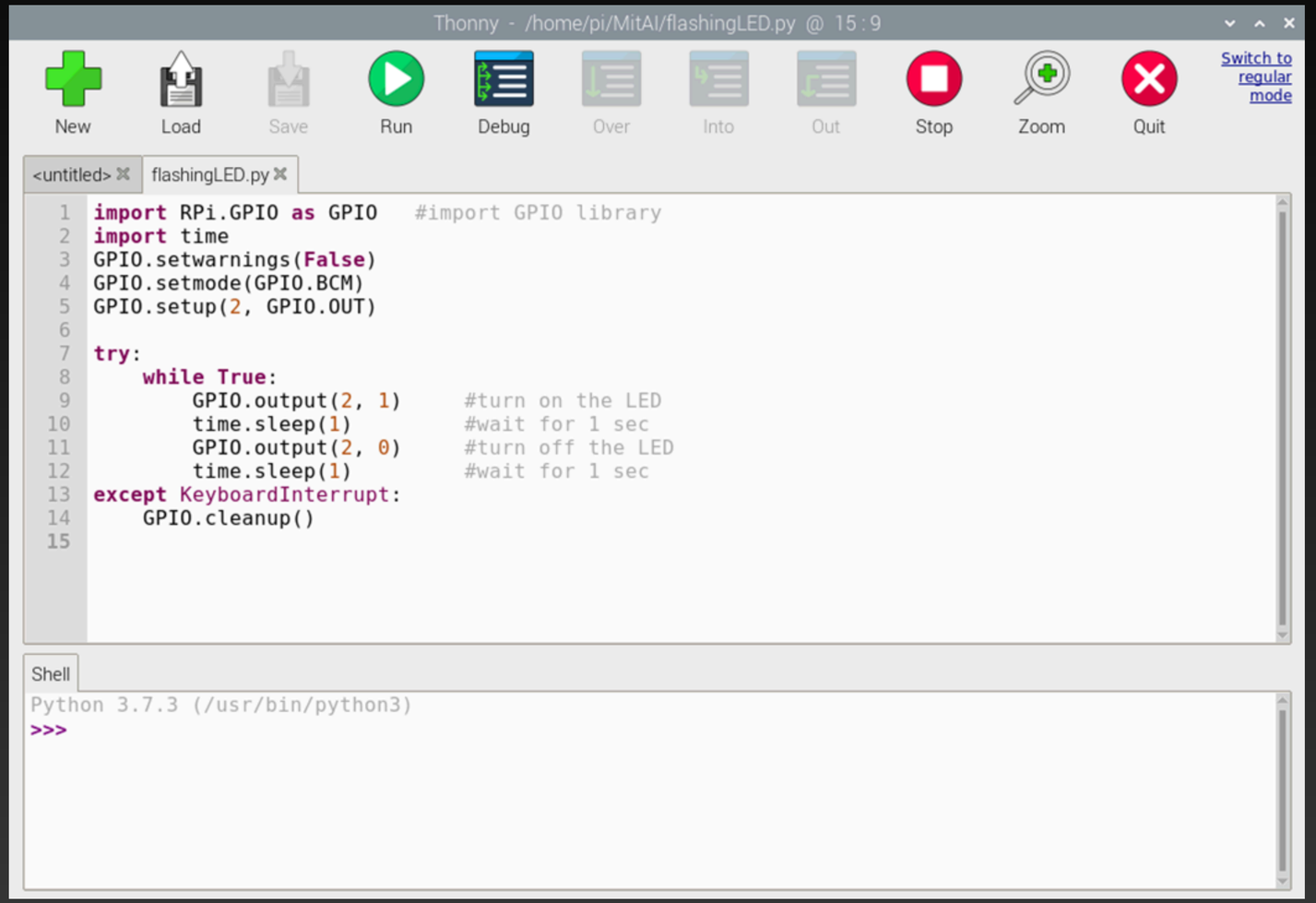
Channel(I/O port pin) Config

Output configuration

To send data to an output port pin we can use the following statement:

```
GPIO.setup(channel, value)
```

Where value can be 0(or GPIO.LOW, or False), or 1 (or GPIO.HIGH, or True)



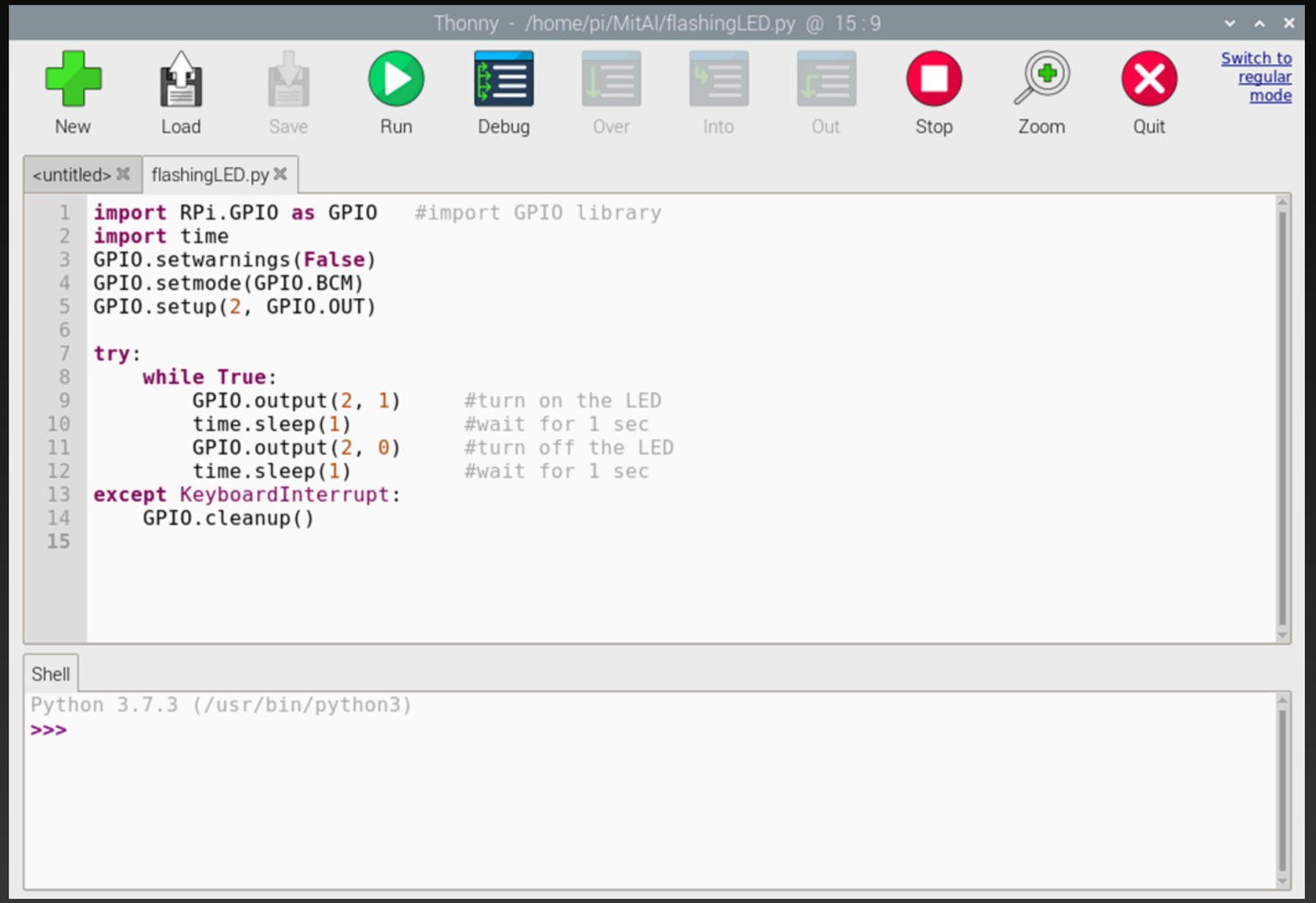
Physical Computing from Python - GPIO Library

Channel(I/O port pin) Config

Output configuration

At the end of the program, we should return all the used resources to the operating system. This is done by including the following statement at the end of our program:

```
GPIO.cleanup()
```



Flashing LED - Accessing the External World

Background info:

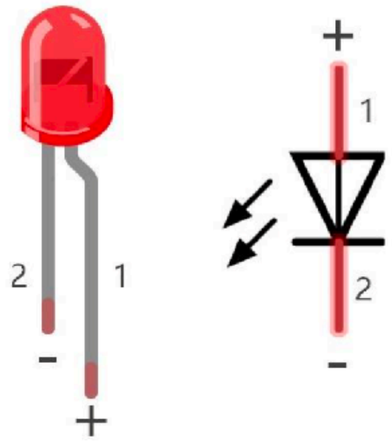
The forward voltage of an LED is around 1.8V

The current through the LED depends on the required light intensity and the type of LED used.

In general, 3mA should give enough visible light for small LEDs.

If we use output 3.3V, we can calculate the value of the current limiting resistor as:

3.3V - 1.8V / 3mA = 500 Ohm

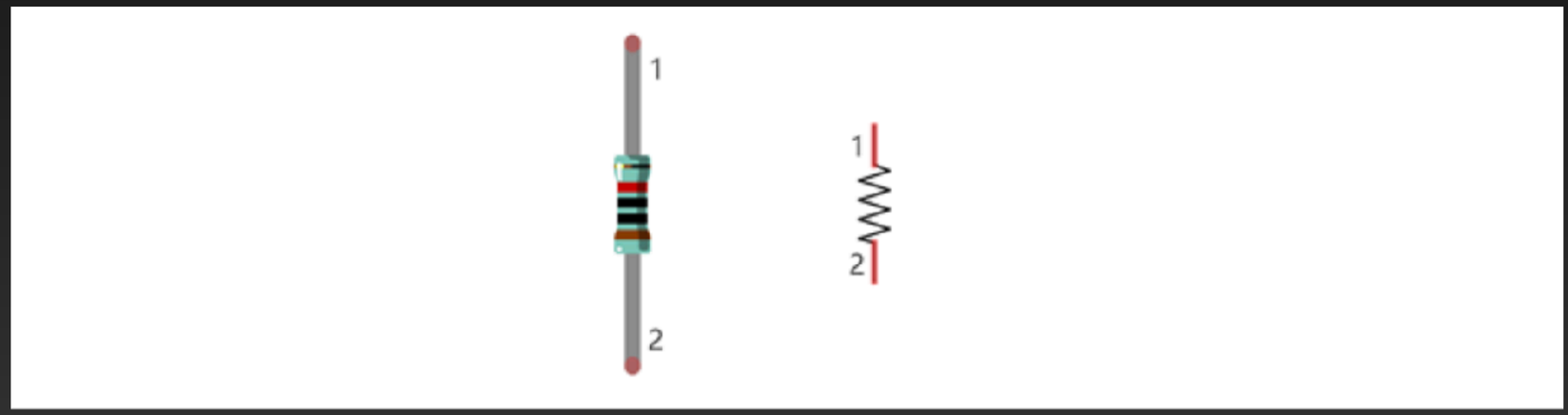


LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

Flashing LED - Accessing the External World

Background info:

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit.

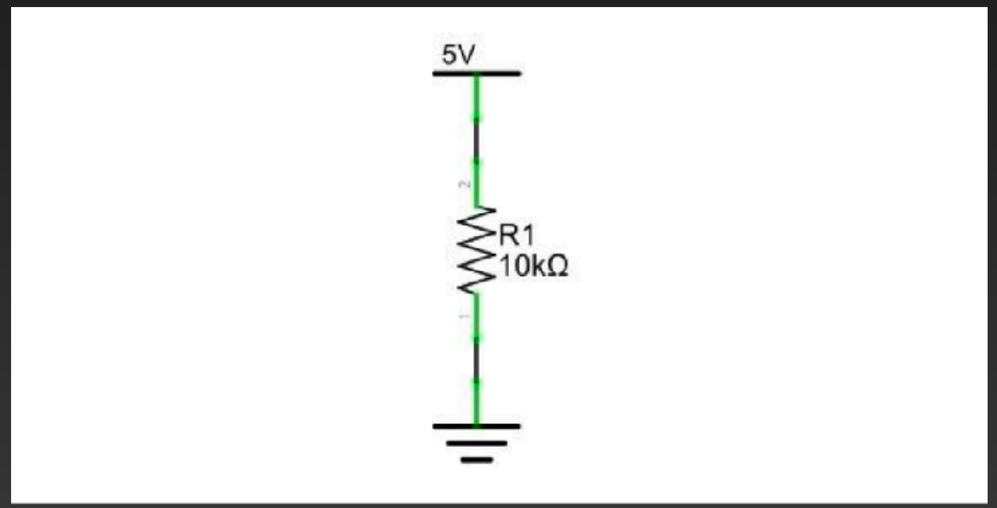


Flashing LED - Accessing the External World

Background info:

The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.
With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: I=V/R known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

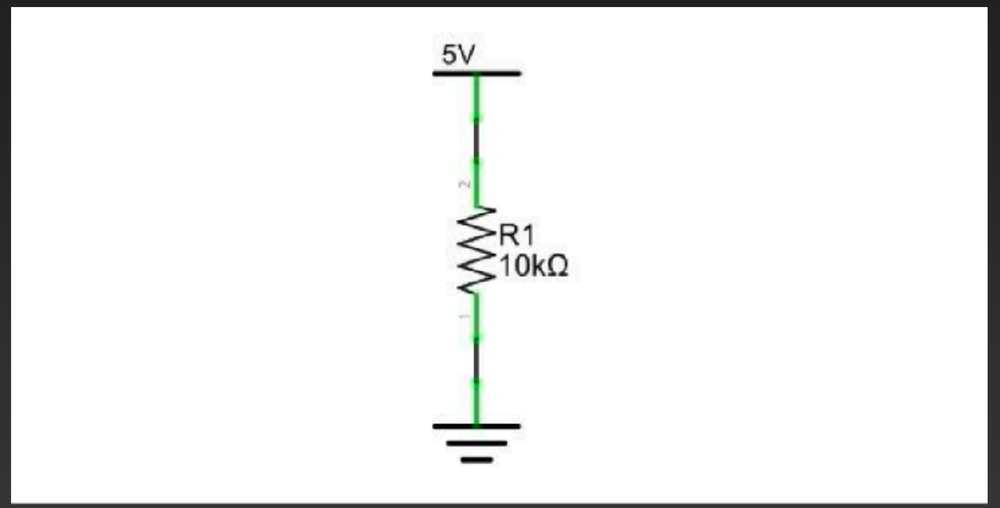
In the following diagram, the current through R1 is: I=U/R=5V/10kΩ=0.0005A=0.5mA.



Flashing LED - Accessing the External World

WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

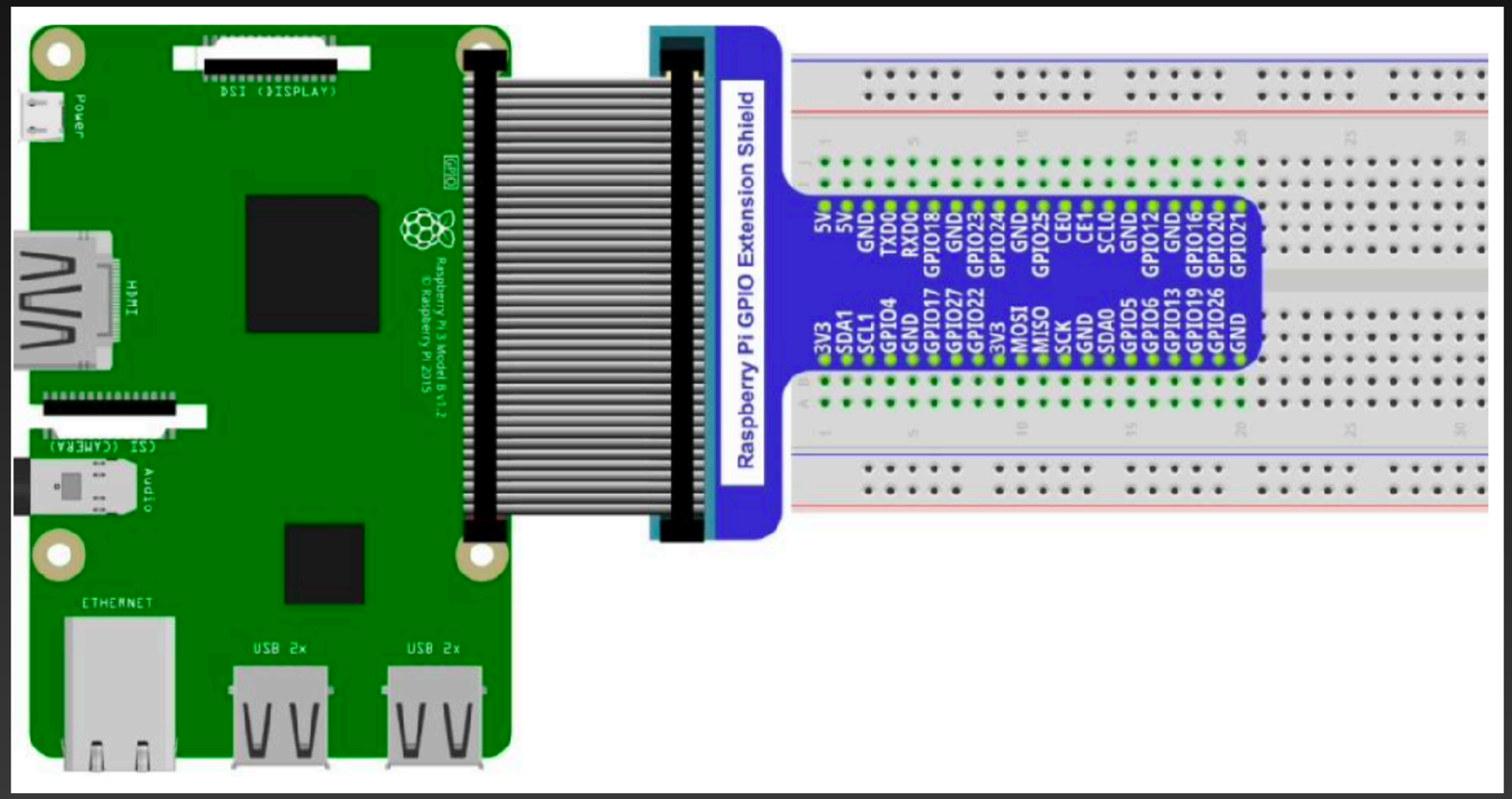
Note: Unlike LEDs and Diodes, Resistors have no poles and re non-polar (it does not matter which direction you insert them into a circuit, it will work the same)



Introducing Extension Board

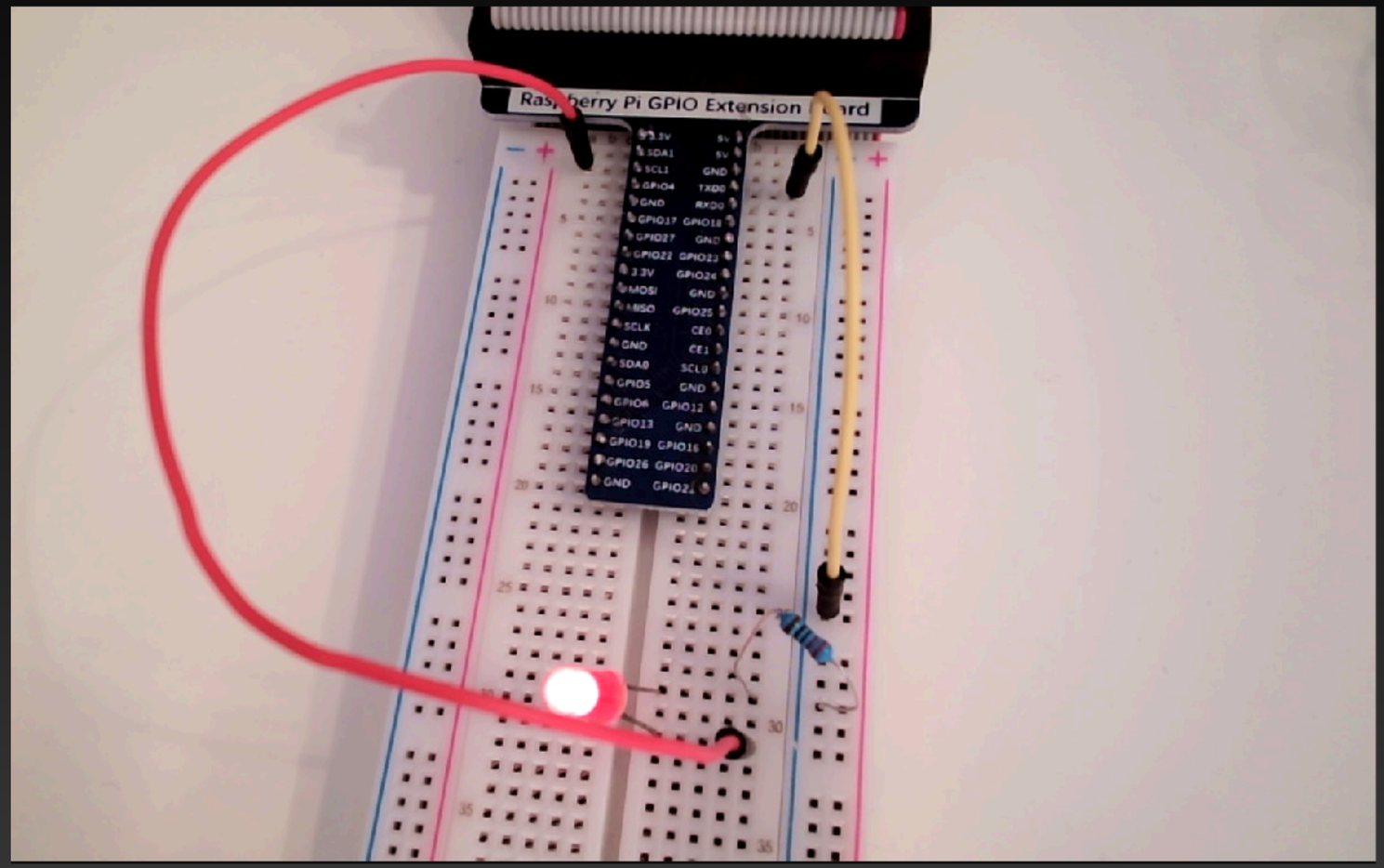
GPIO Extension Board

GPIO board is a convenient way to connect the RPi I/O ports to the breadboard directly. The GPIO pin sequence on Extension Board is identical to the GPIO pin sequence of RPi.



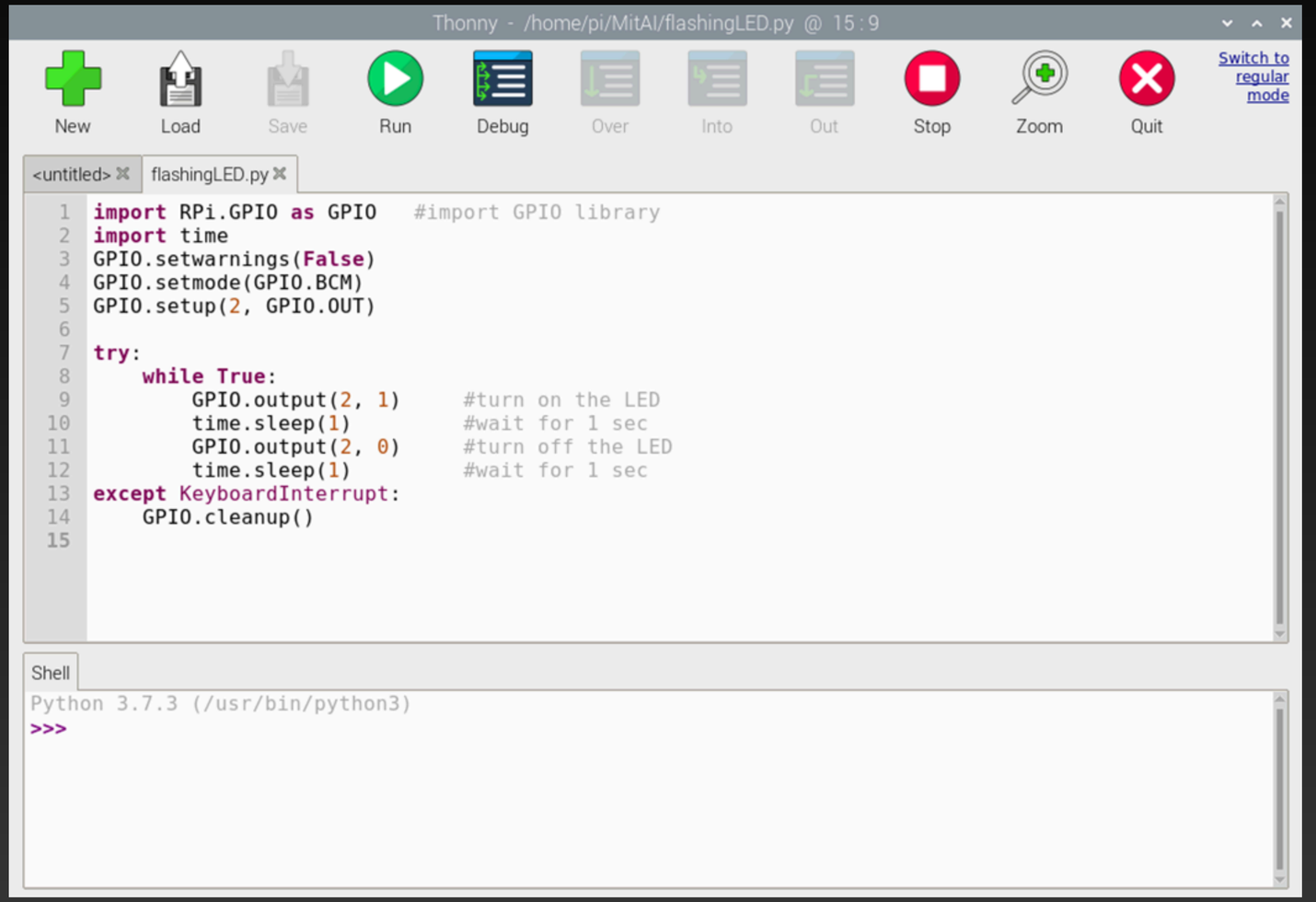
Flashing LED - Accessing the External World

Connect the wires use GPIO2 as output.



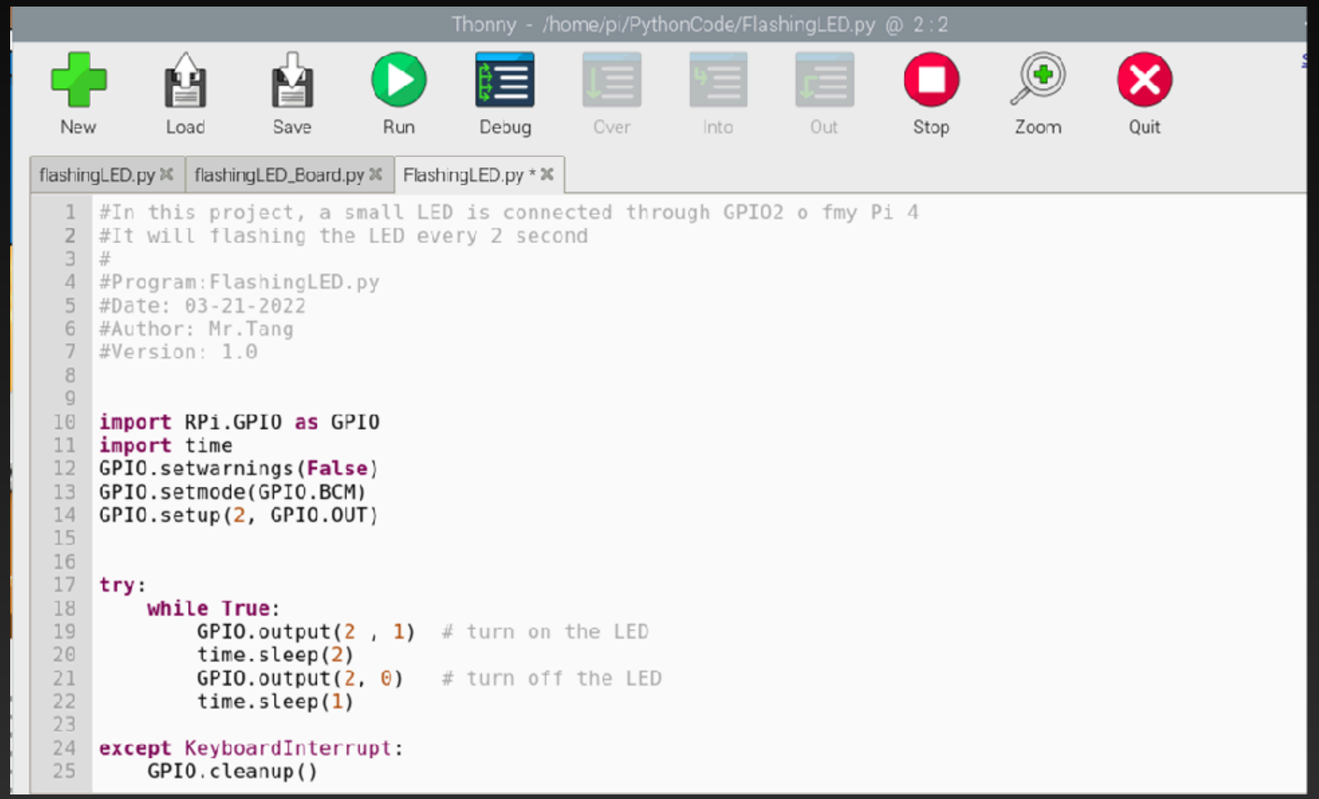
Flashing LED - Accessing the External World

Code Flashing LED program in Thonny.



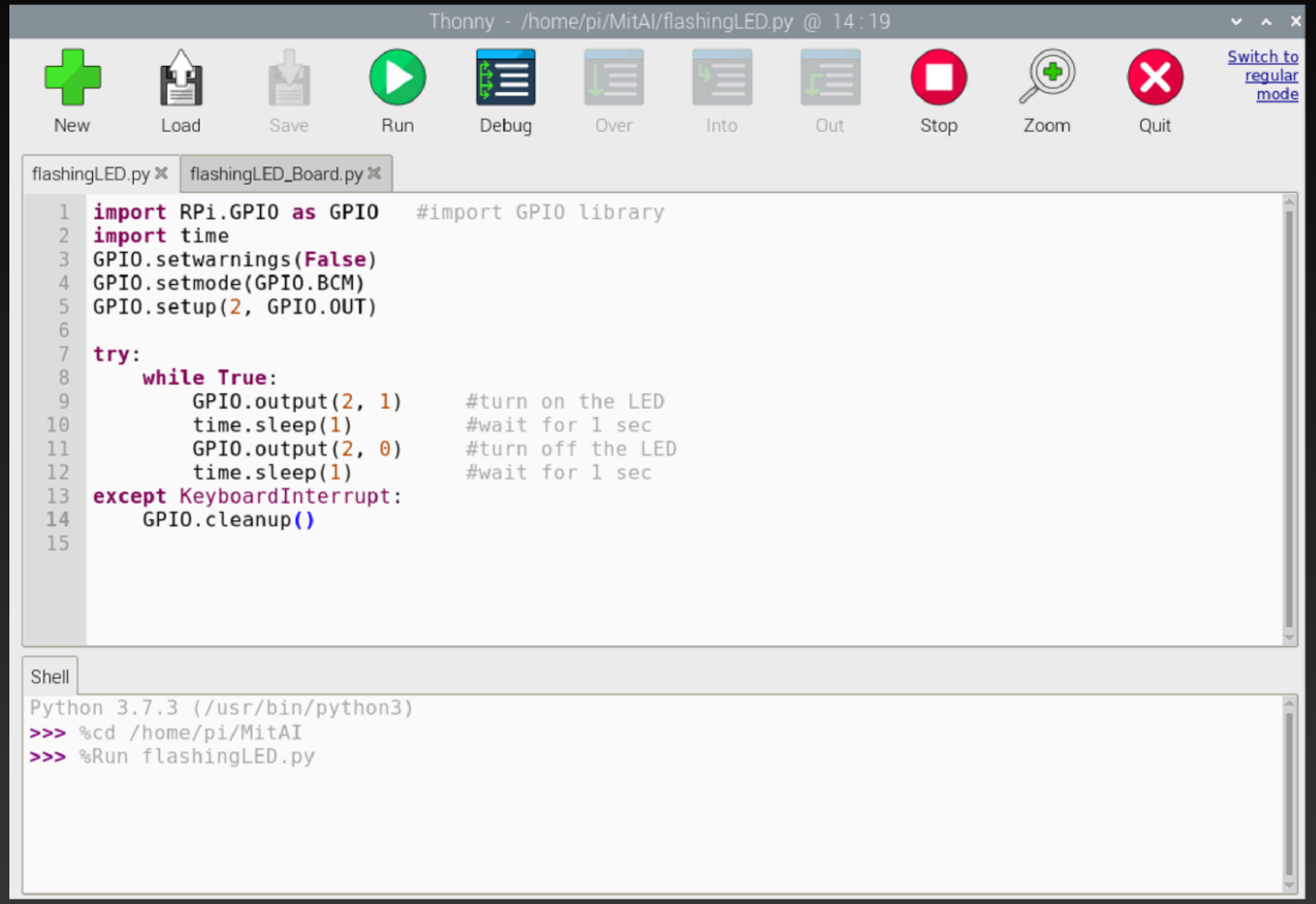
Flashing LED - Accessing the External World

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(2, GPIO.OUT)


try:
    while True:
        GPIO.output(2 , 1)  # turn on the LED
        time.sleep(2)
        GPIO.output(2, 0)   # turn off the
LED
        time.sleep(1)

except KeyboardInterrupt:
    GPIO.cleanup()
```



Flashing LED - Accessing the External World

Run the code.



GPIO control in python on Raspberry Pi

Home project - Develop python program to simulate traffic lights.

Hint: Use three GPIO pins as input and modify the Python program learned from class to simulate traffic lights.

IoT with MIT App Inventor

Office hour