

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/>

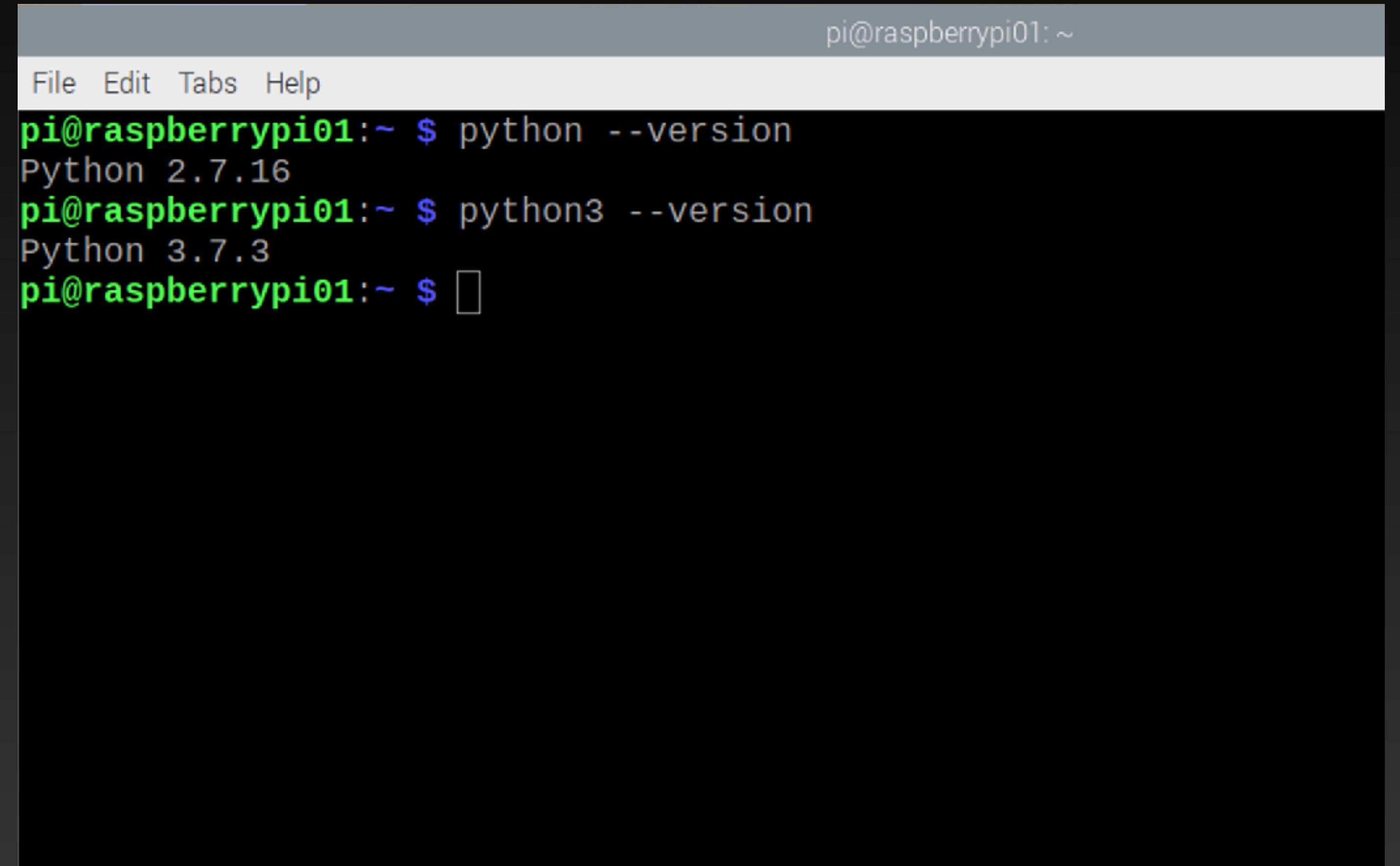
Python Programming Language

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



A screenshot of a terminal window titled "pi@raspberrypi01: ~". The window shows the following text:

```
File Edit Tabs Help
pi@raspberrypi01:~ $ python --version
Python 2.7.16
pi@raspberrypi01:~ $ python3 --version
Python 3.7.3
pi@raspberrypi01:~ $ █
```

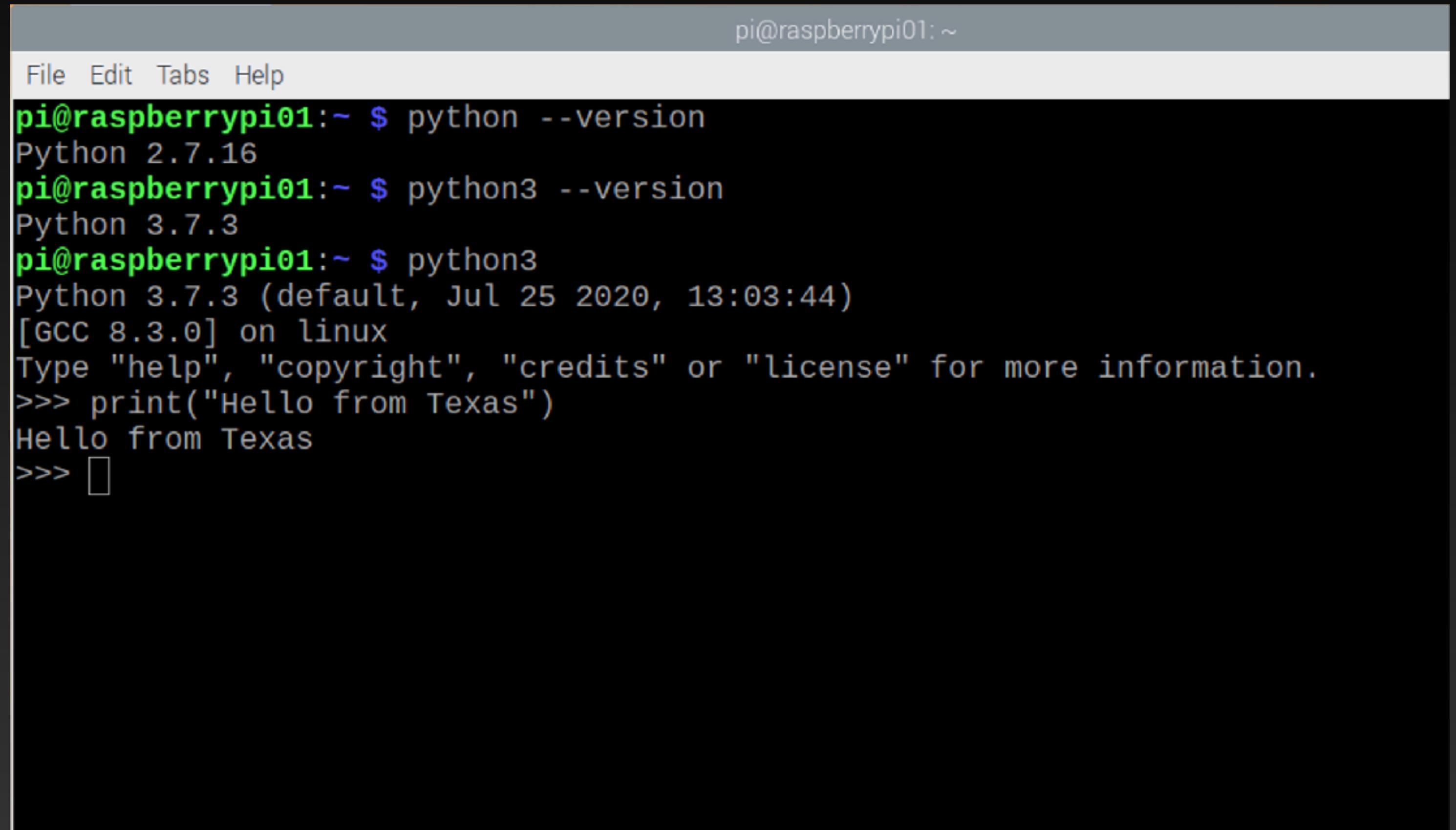
Python Programming Language

Run python program interactively.

In the terminal, type in command: python3

Type the program:

```
print("Hello from Texas")
```



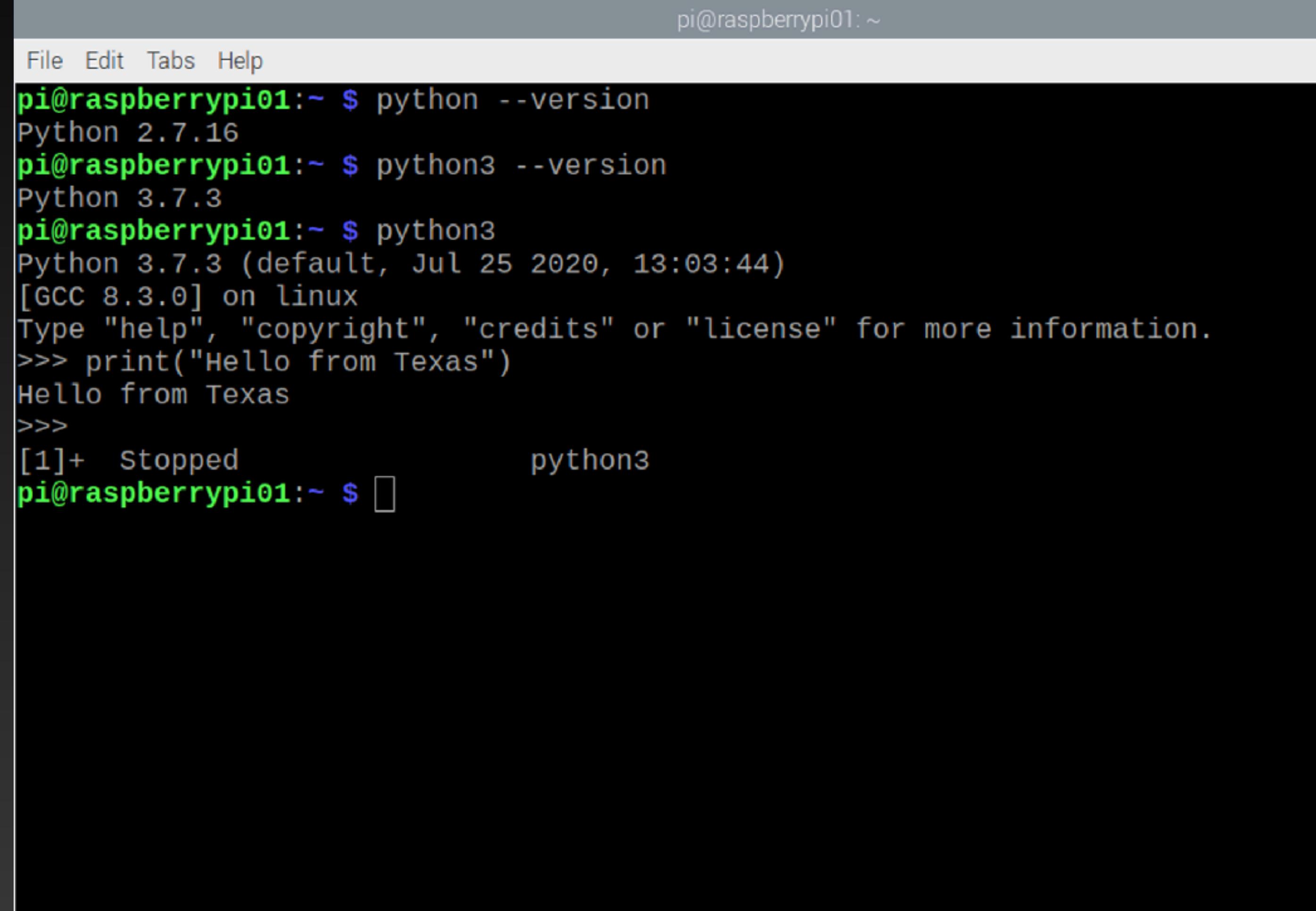
A screenshot of a terminal window titled "pi@raspberrypi01:~". The window shows the following text:

```
File Edit Tabs Help
pi@raspberrypi01:~ $ python --version
Python 2.7.16
pi@raspberrypi01:~ $ python3 --version
Python 3.7.3
pi@raspberrypi01:~ $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from Texas")
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



A screenshot of a terminal window titled "pi@raspberrypi01: ~". The window shows the following session:

```
pi@raspberrypi01:~ $ python --version
Python 2.7.16
pi@raspberrypi01:~ $ python3 --version
Python 3.7.3
pi@raspberrypi01:~ $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from Texas")
Hello from Texas
>>>
[1]+  stopped                  python3
pi@raspberrypi01:~ $ 
```

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.

```
pi@raspberrypi01:~
```

```
File Edit Tabs Help
```

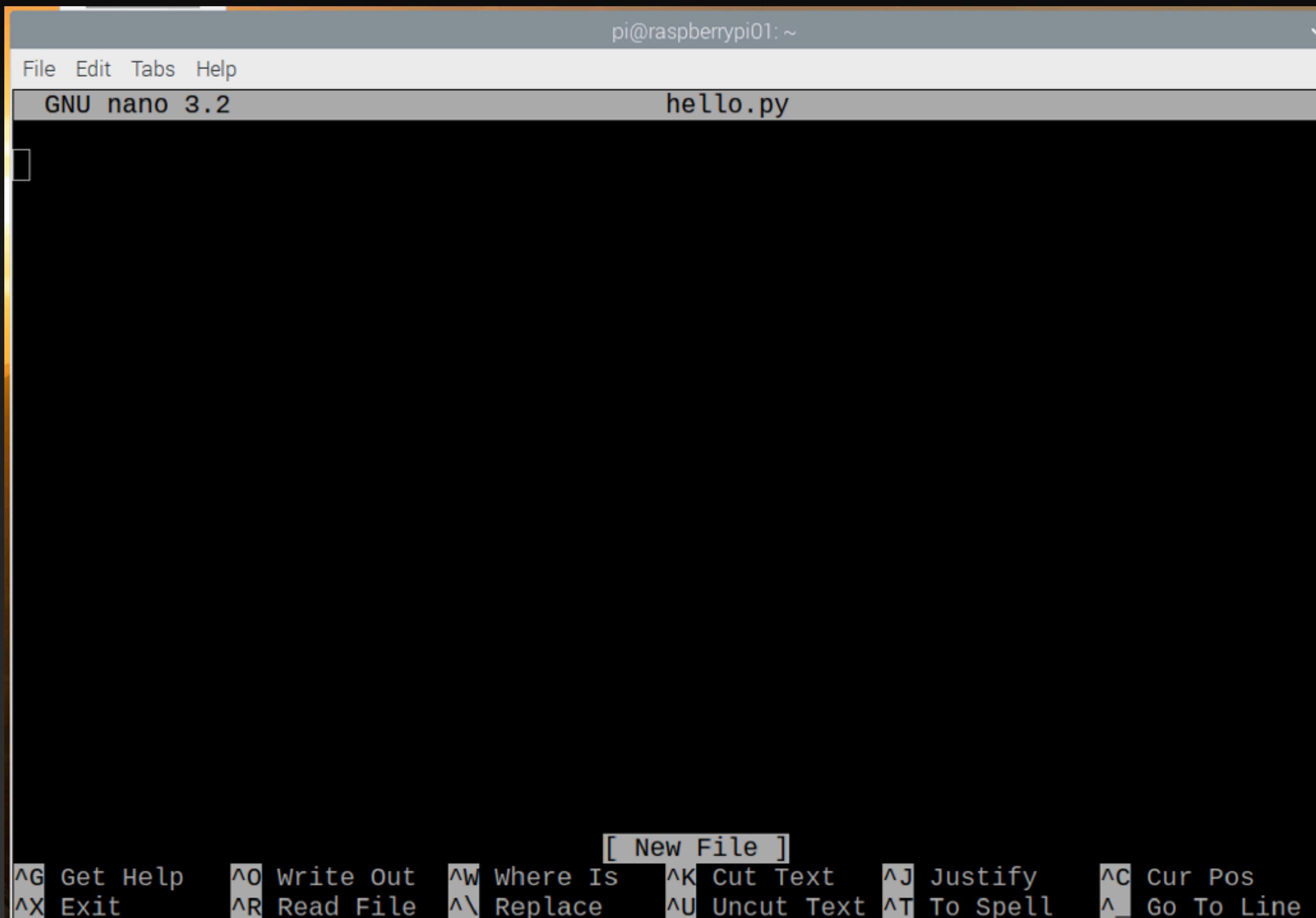
```
pi@raspberrypi01:~ $ python --version
Python 2.7.16
pi@raspberrypi01:~ $ python3 --version
Python 3.7.3
pi@raspberrypi01:~ $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from Texas")
Hello from Texas
>>>
[1]+  Stopped                  python3
pi@raspberrypi01:~ $ nano hello.py
pi@raspberrypi01:~ $
```

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

```
pi@raspberrypi01: ~
File Edit Tabs Help
GNU nano 3.2          hello.py          Modif
Hello from Texas

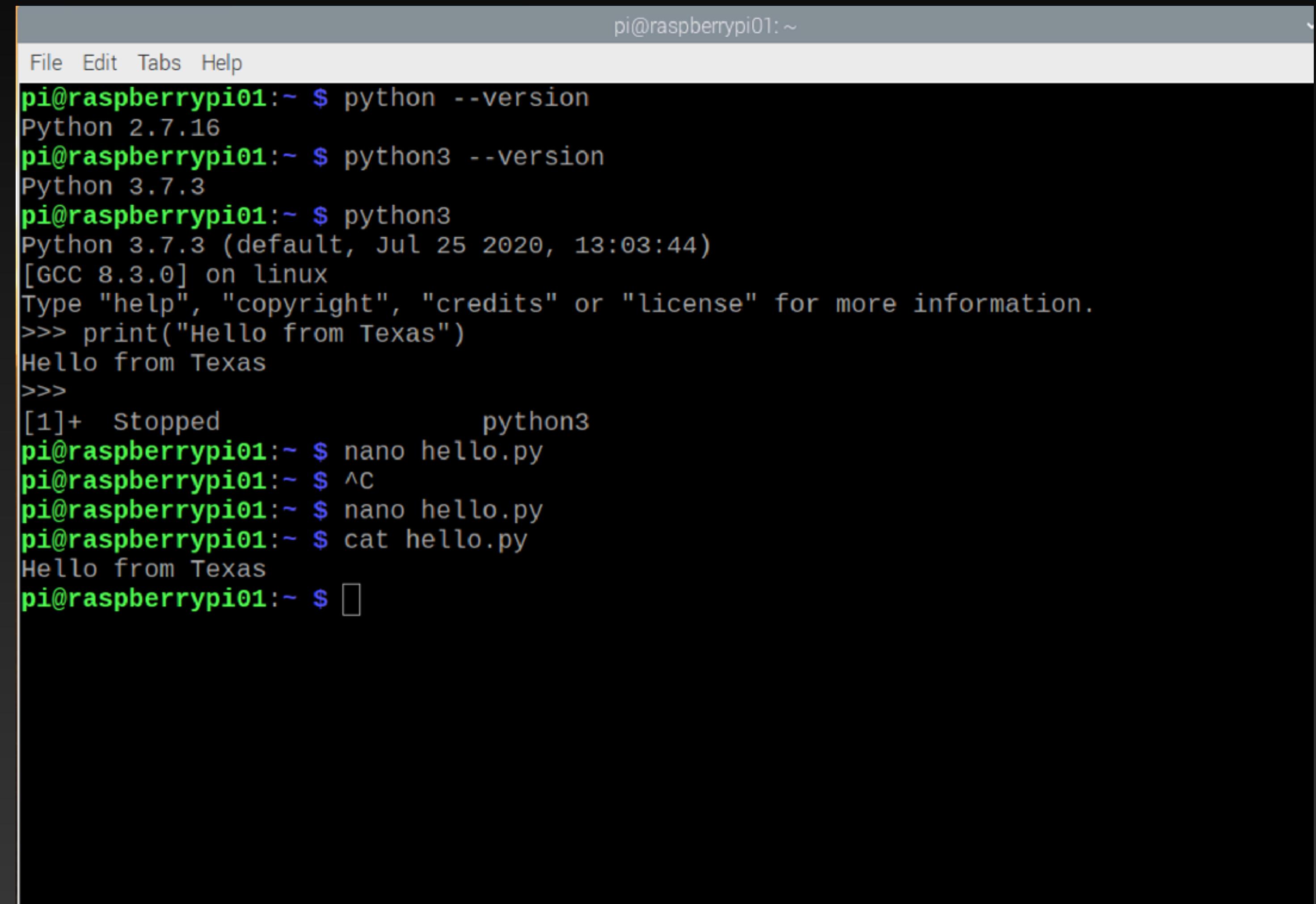
File Name to Write: hello.py
^G Get Help      M-D DOS Format      M-A Append      M-B Backup File
^C Cancel       M-M Mac Format      M-P Prepend      ^T To Files
```

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

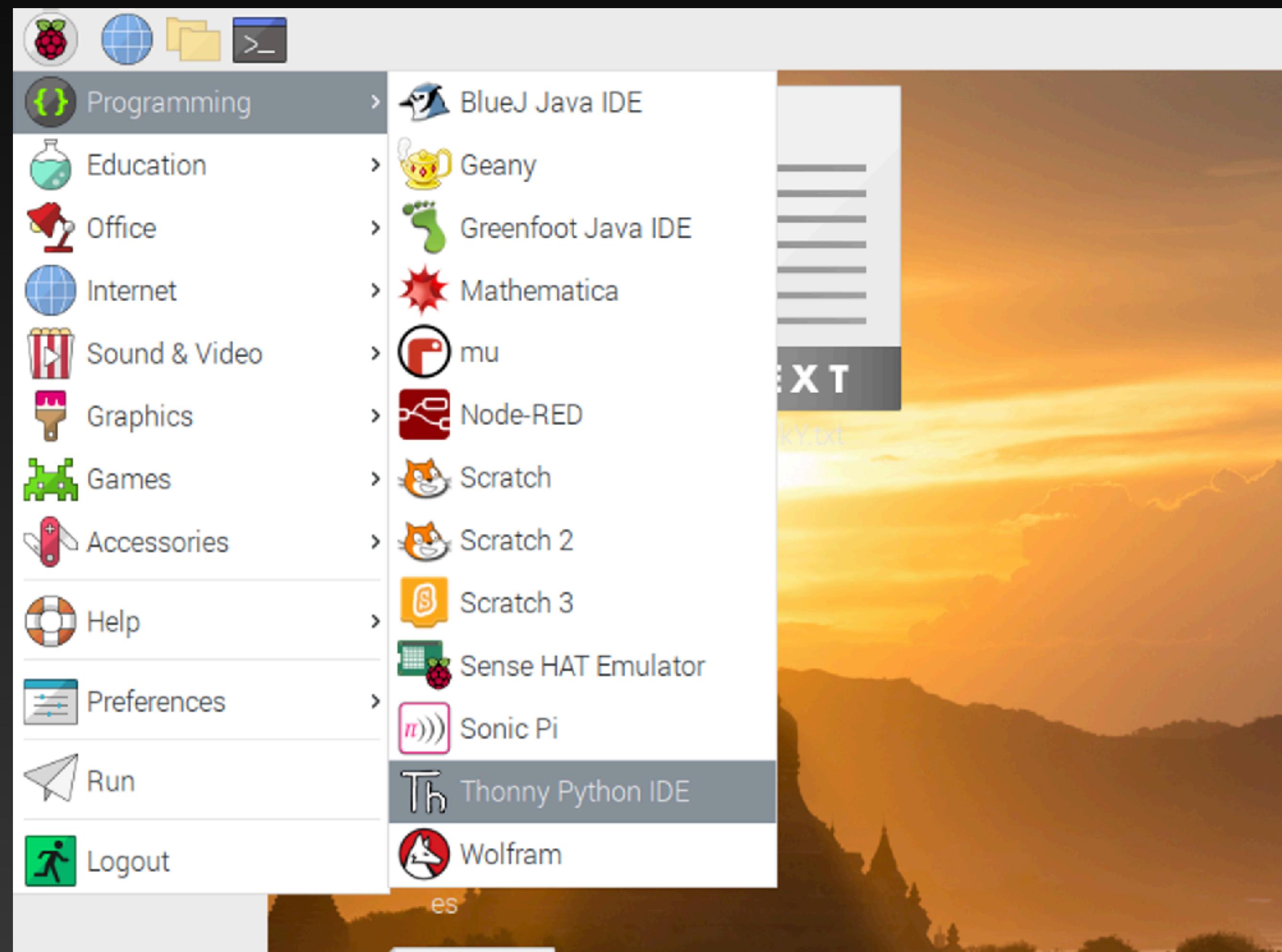


The screenshot shows a terminal window titled 'pi@raspberrypi01: ~'. The window contains the following text:

```
File Edit Tabs Help
pi@raspberrypi01:~ $ python --version
Python 2.7.16
pi@raspberrypi01:~ $ python3 --version
Python 3.7.3
pi@raspberrypi01:~ $ python3
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from Texas")
Hello from Texas
>>>
[1]+  Stopped                  python3
pi@raspberrypi01:~ $ nano hello.py
pi@raspberrypi01:~ $ ^C
pi@raspberrypi01:~ $ nano hello.py
pi@raspberrypi01:~ $ cat hello.py
Hello from Texas
pi@raspberrypi01:~ $ 
```

Creating Python File in GUI - Thonny

Access Thonny from
programming dropdown

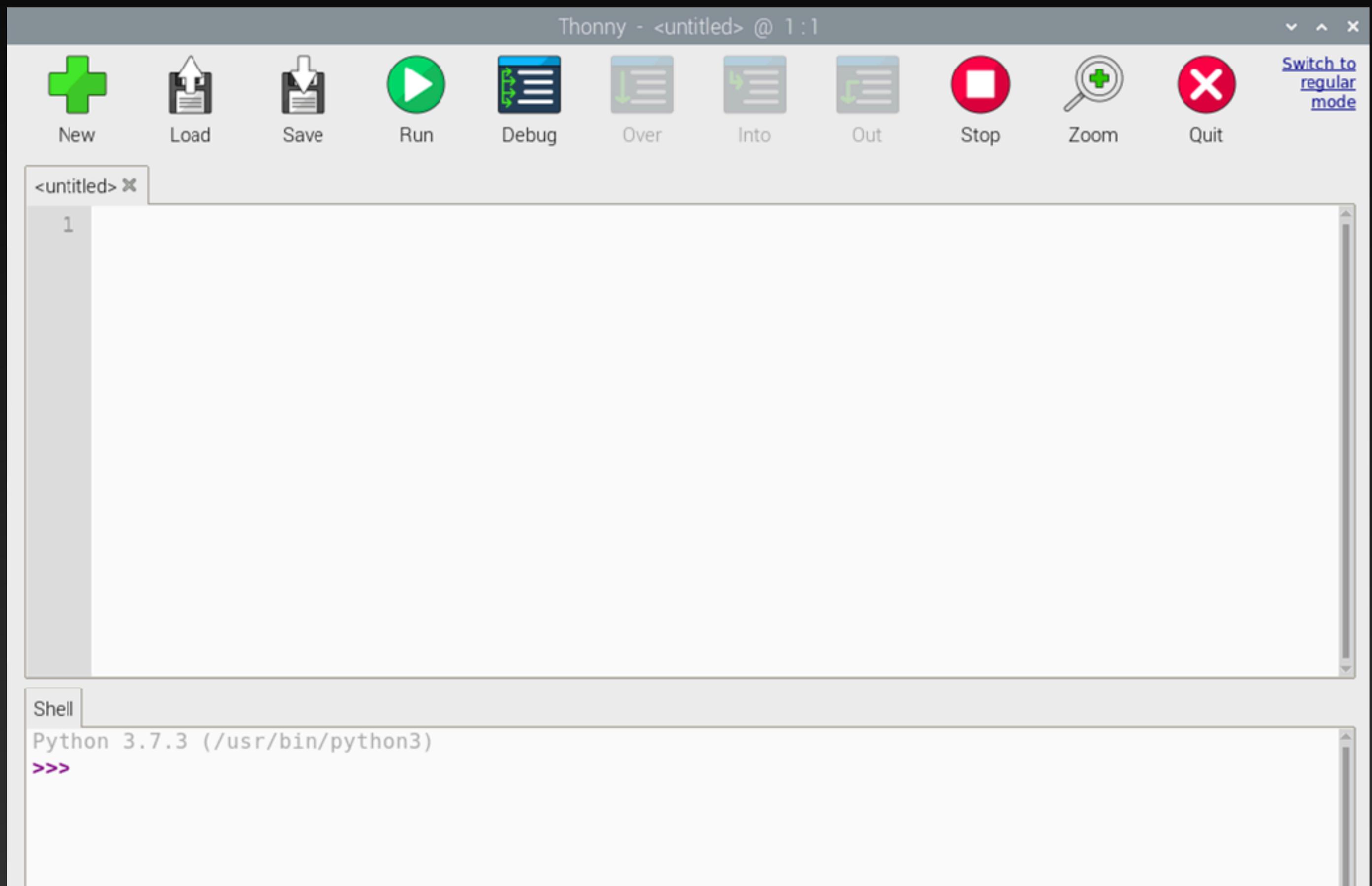


Creating Python File in GUI - Thonny

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

Which method to choose?

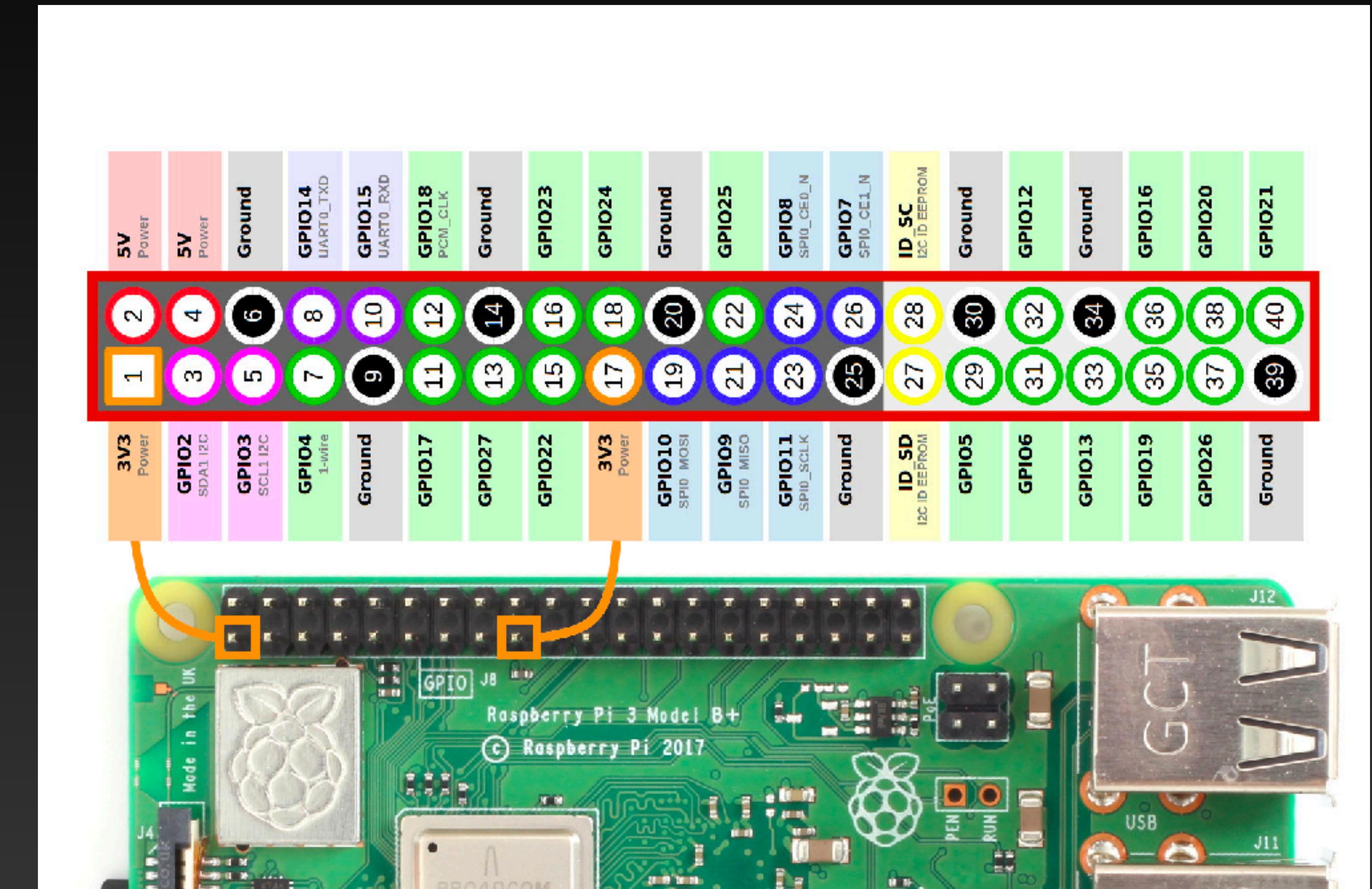
Interactively or program file?



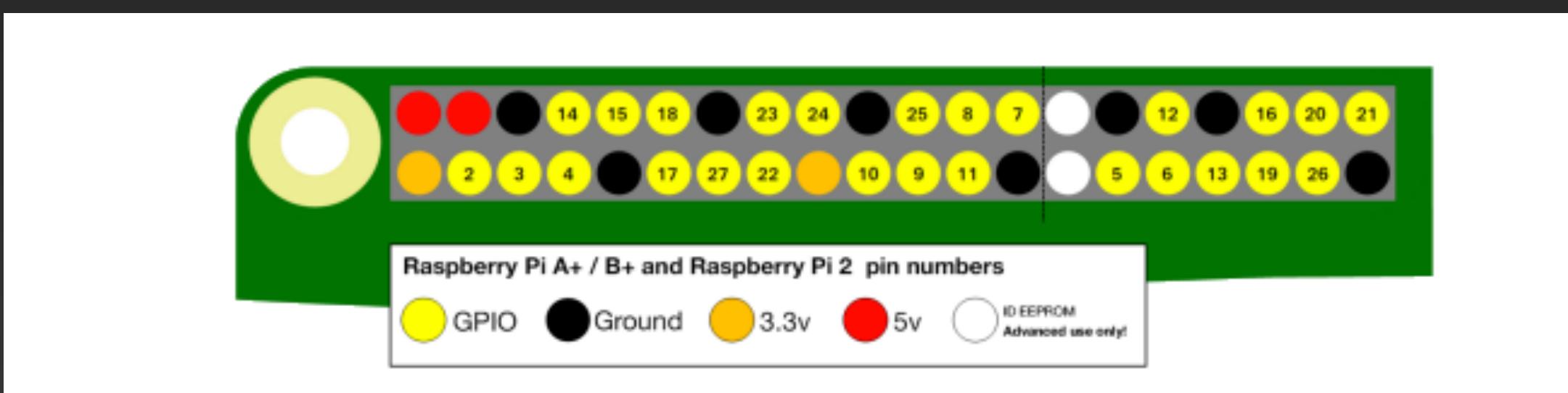
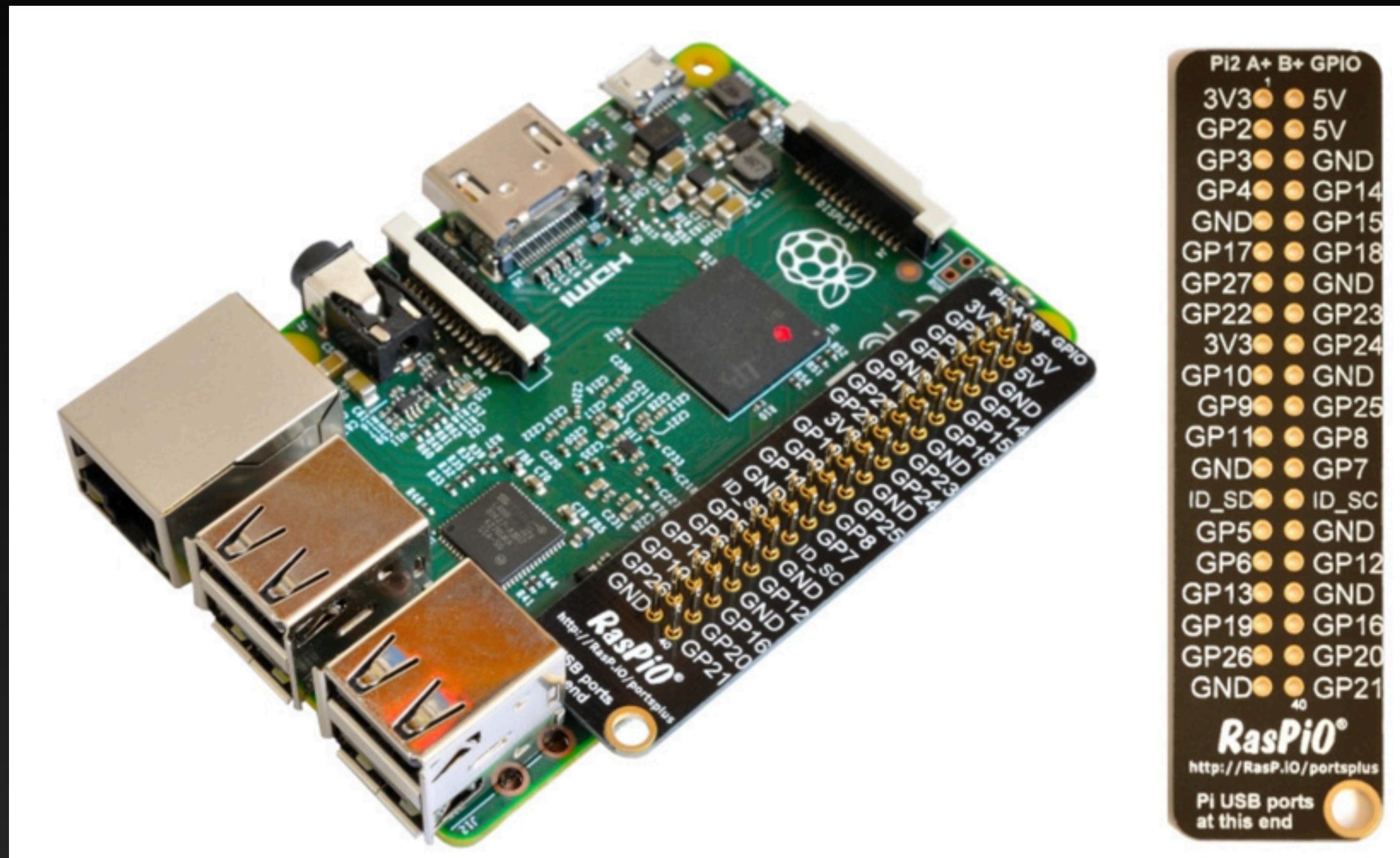
Physical Computing from Python - GPIO

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

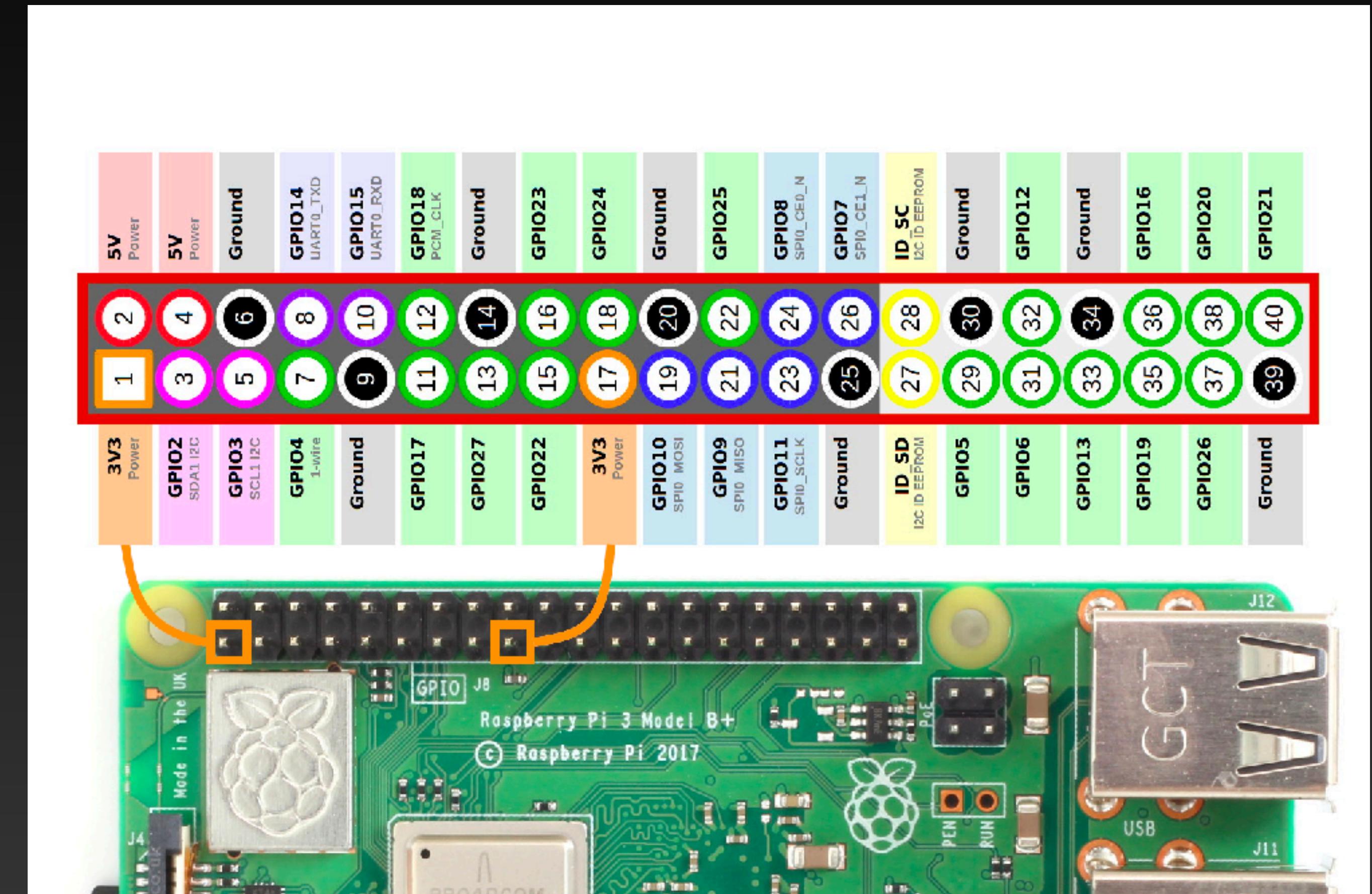


3V3	3.3 volts	Anything connected to these pins will always get 3.3V of power
5V	5 volts	Anything connected to these pins will always get 5V of power
GND	ground	Zero volts, used to complete a circuit
GP2	GPIO pin 2	These pins are for general-purpose use and can be configured as input or output pins
ID_SD/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

Physical Computing from Python - GPIO

Pin Labels – Clear as Mud

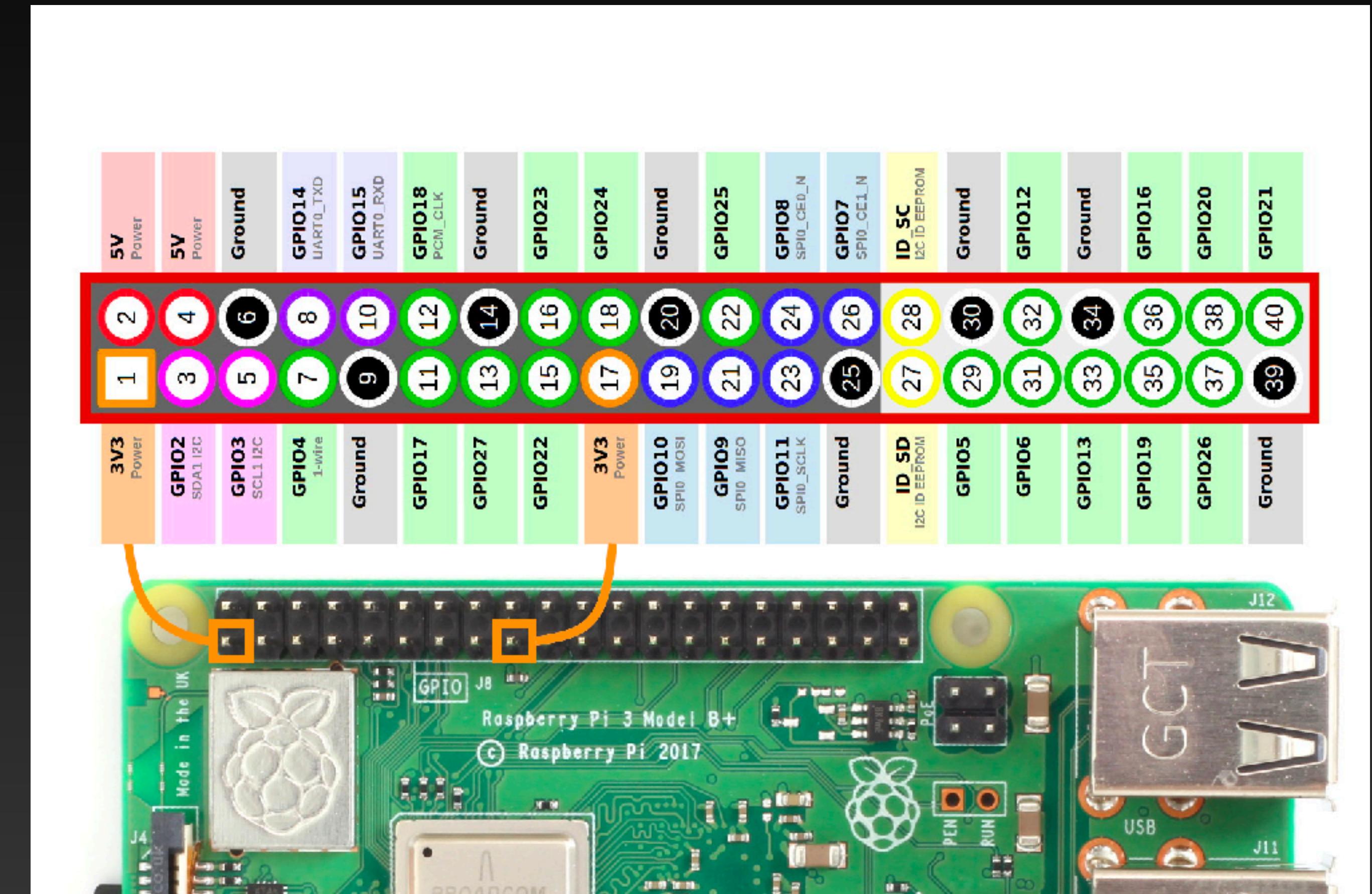
The labels above are the names of the pins on the Broadcom system chip to which the pin is physically connected. Much of the confusion around the GPIO is due to these labels, their relationship to the Broadcom labels and how they are referred to in your programs. To confuse things even more the GPIO 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.

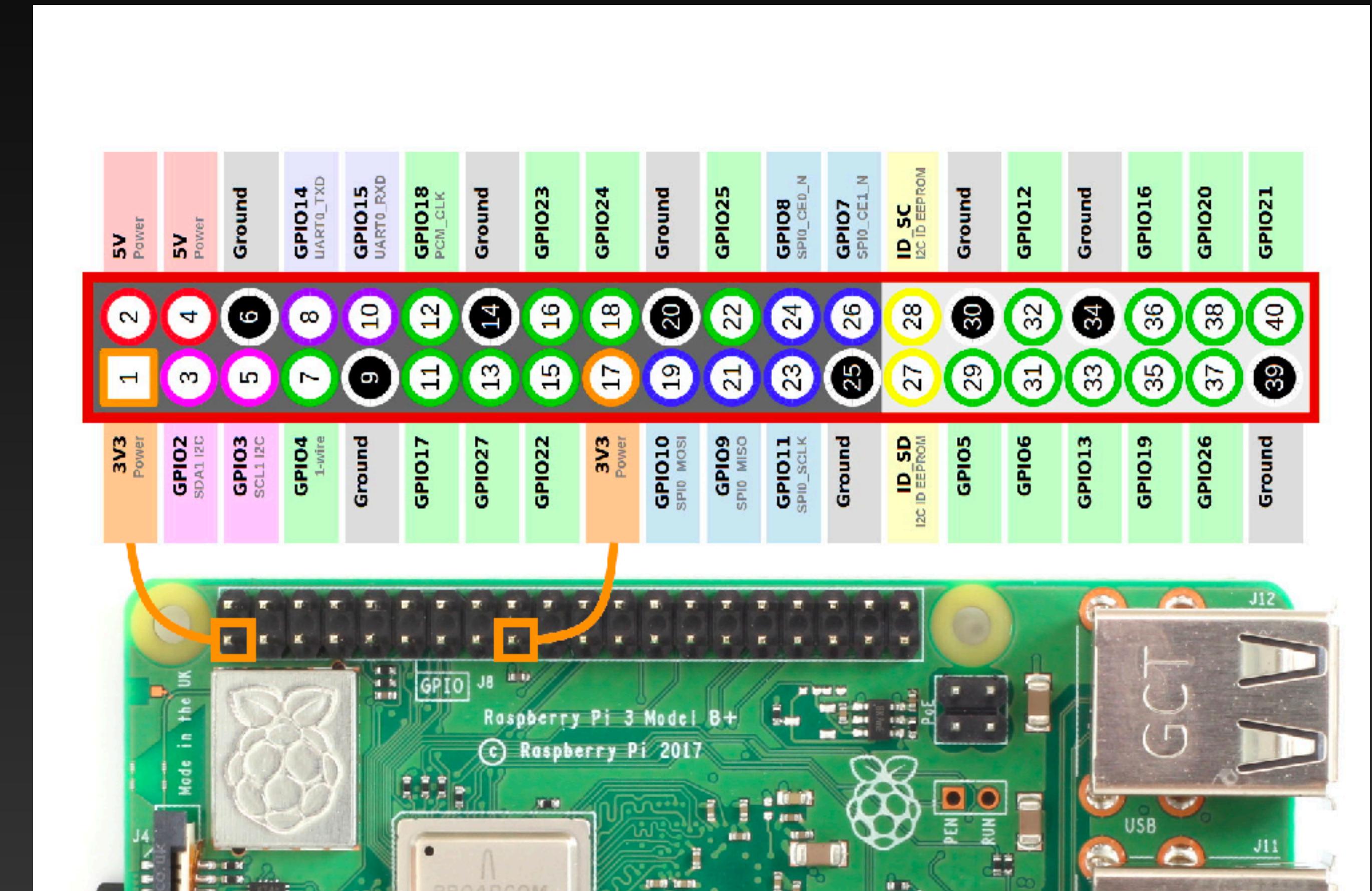


Physical Computing from Python - GPIO

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.

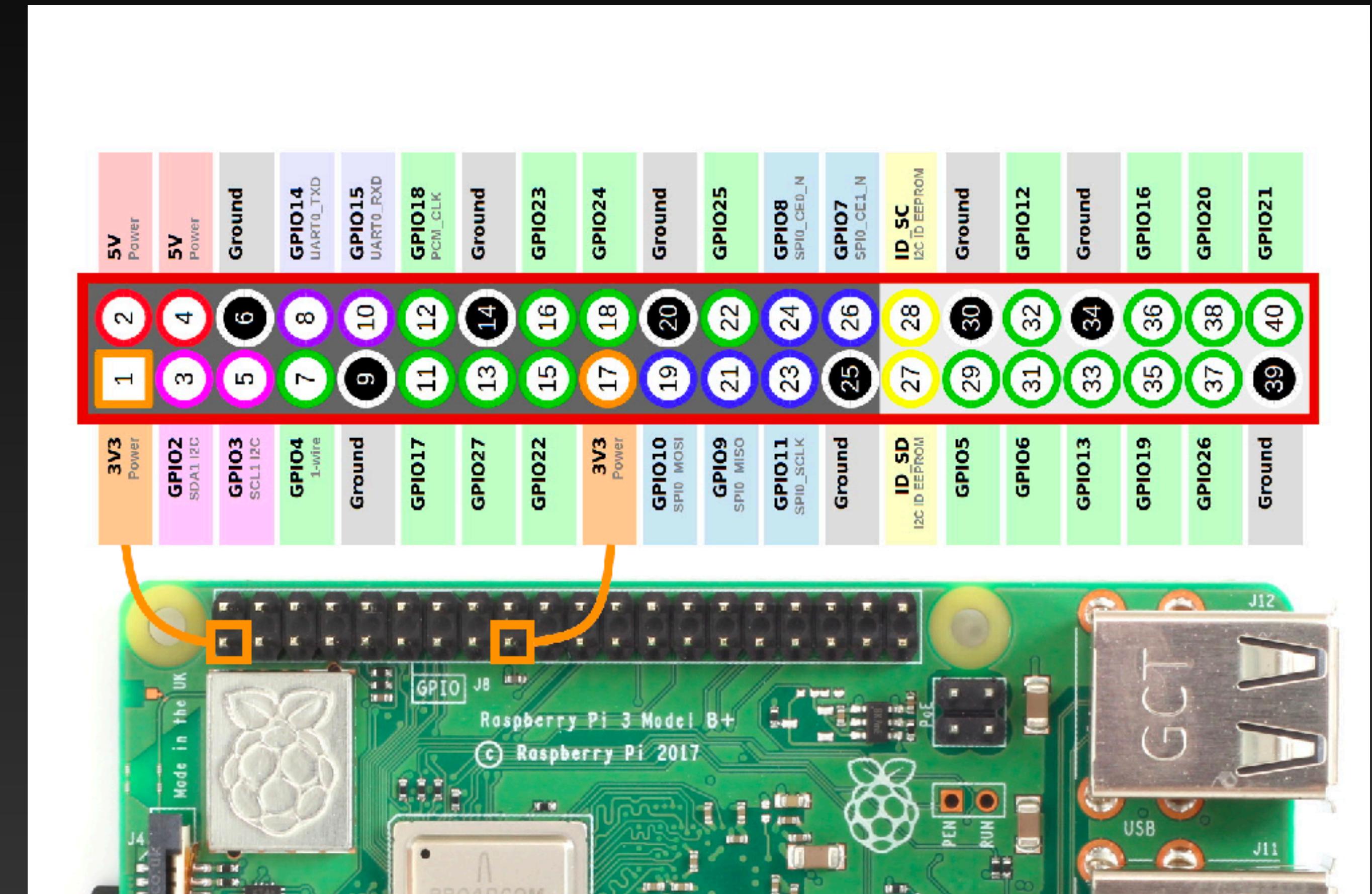
In order to use these pins you must tell the system whether they are inputs or outputs. This can be achieved a number of ways and it depends on how you intend to control them.



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.



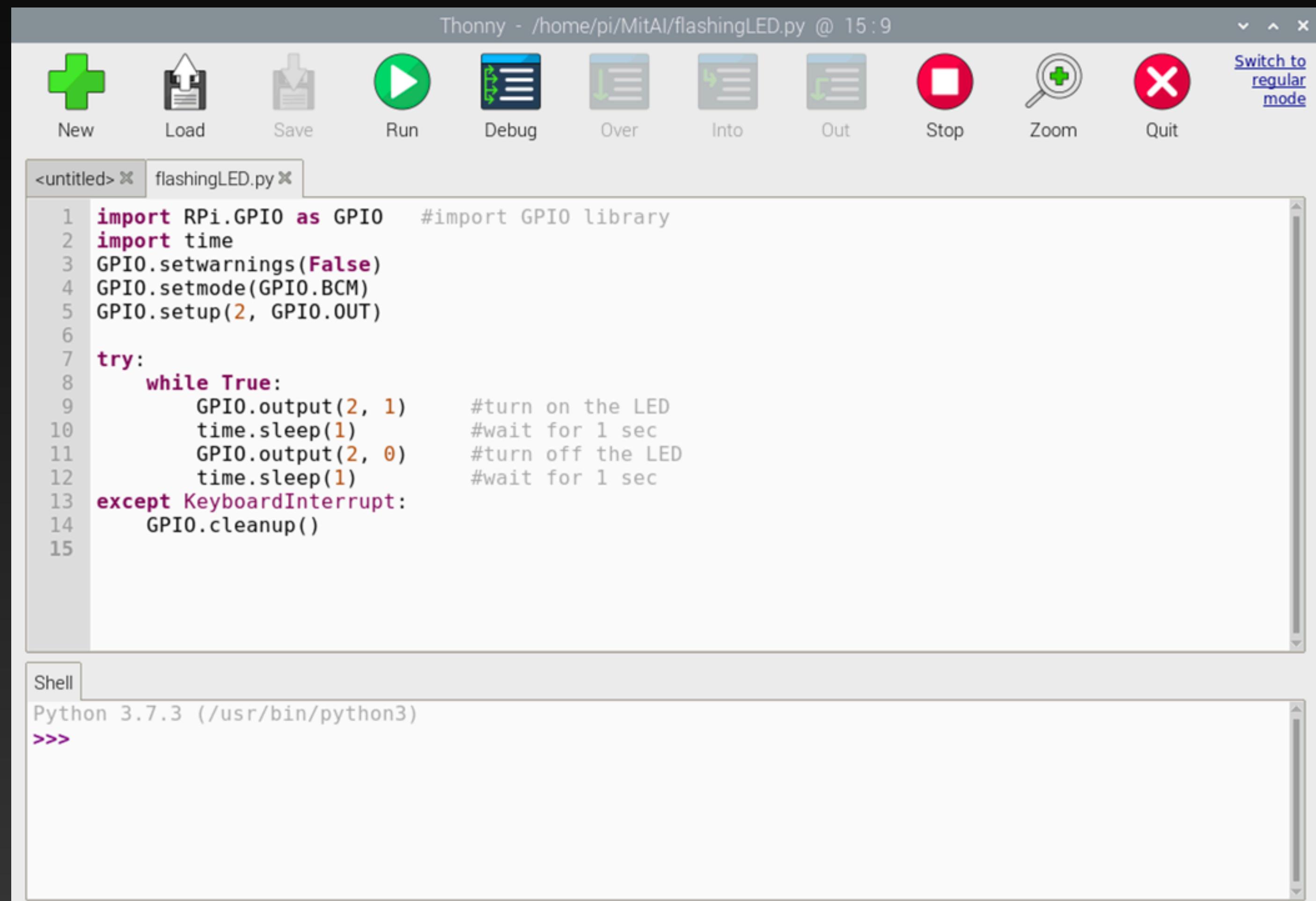
Physical Computing from Python - GPIO Library

The GPIO Library

The GPIO library is called RPi.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
```



The screenshot shows the Thonny Python IDE interface. The title bar reads "Thonny - /home/pi/MitAI/flashingLED.py @ 15 : 9". The toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit, along with a "Switch to regular mode" button. The main window displays a code editor with the file "flashingLED.py" open. The code is as follows:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

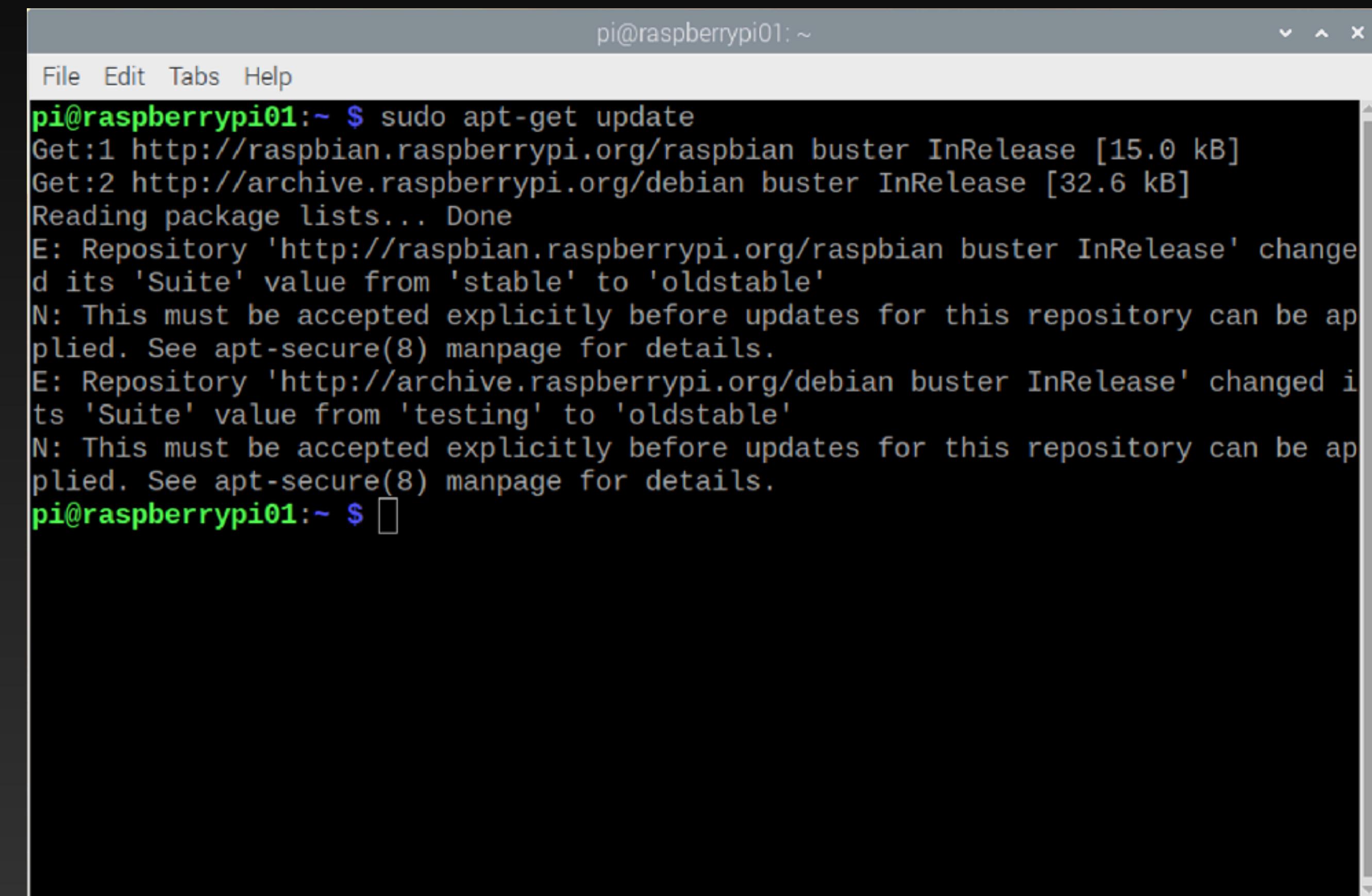
6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
```

Below the code editor is a "Shell" tab showing the Python interpreter prompt: "Python 3.7.3 (/usr/bin/python3) >>>".

Physical Computing from Python - GPIO Library

Install GPIO Library

sudo apt-get update



A screenshot of a terminal window titled "pi@raspberrypi01: ~". The window has a standard OS X-style title bar with icons for minimizing, maximizing, and closing. The menu bar includes "File", "Edit", "Tabs", and "Help". The main area of the terminal shows the command "sudo apt-get update" being run. The output indicates that the Raspbian repository's 'Suite' value was changed from 'stable' to 'oldstable', which requires explicit acceptance before updates can be applied. The terminal prompt "pi@raspberrypi01: ~ \$ " is visible at the bottom.

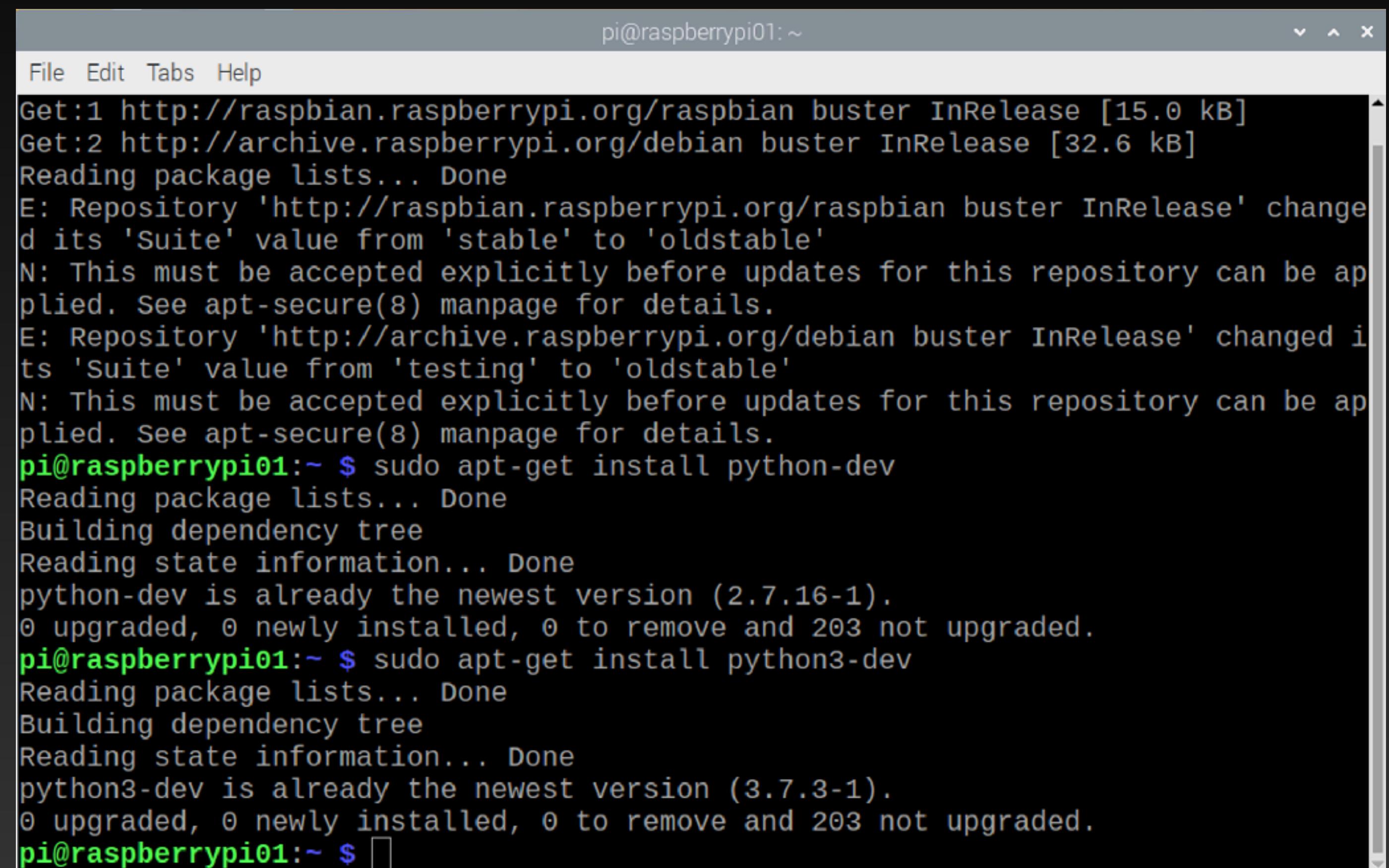
```
pi@raspberrypi01:~ $ sudo apt-get update
Get:1 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:2 http://archive.raspberrypi.org/debian buster InRelease [32.6 kB]
Reading package lists... Done
E: Repository 'http://raspbian.raspberrypi.org/raspbian buster InRelease' changed its 'Suite' value from 'stable' to 'oldstable'
N: This must be accepted explicitly before updates for this repository can be applied. See apt-secure(8) manpage for details.
E: Repository 'http://archive.raspberrypi.org/debian buster InRelease' changed its 'Suite' value from 'testing' to 'oldstable'
N: This must be accepted explicitly before updates for this repository can be applied. See apt-secure(8) manpage for details.
pi@raspberrypi01:~ $
```

Physical Computing from Python - GPIO Library

Install GPIO Library

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

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



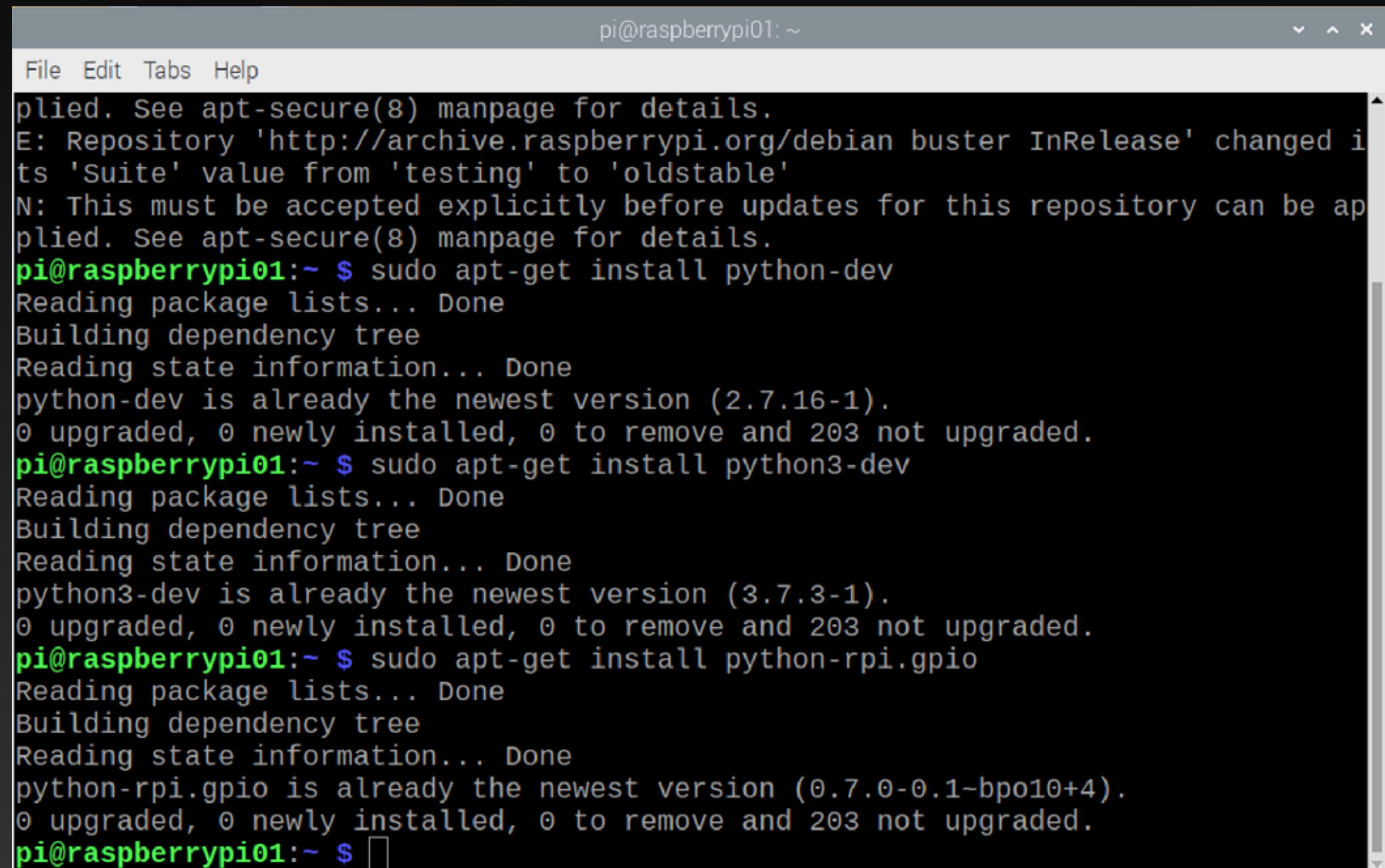
The screenshot shows a terminal window titled "pi@raspberrypi01:~". The window contains the following terminal session:

```
pi@raspberrypi01:~  
File Edit Tabs Help  
Get:1 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]  
Get:2 http://archive.raspberrypi.org/debian buster InRelease [32.6 kB]  
Reading package lists... Done  
E: Repository 'http://raspbian.raspberrypi.org/raspbian buster InRelease' changed its 'Suite' value from 'stable' to 'oldstable'  
N: This must be accepted explicitly before updates for this repository can be applied. See apt-secure(8) manpage for details.  
E: Repository 'http://archive.raspberrypi.org/debian buster InRelease' changed its 'Suite' value from 'testing' to 'oldstable'  
N: This must be accepted explicitly before updates for this repository can be applied. See apt-secure(8) manpage for details.  
pi@raspberrypi01:~ $ sudo apt-get install python-dev  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
python-dev is already the newest version (2.7.16-1).  
0 upgraded, 0 newly installed, 0 to remove and 203 not upgraded.  
pi@raspberrypi01:~ $ sudo apt-get install python3-dev  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
python3-dev is already the newest version (3.7.3-1).  
0 upgraded, 0 newly installed, 0 to remove and 203 not upgraded.  
pi@raspberrypi01:~ $
```

Physical Computing from Python - GPIO Library

Install GPIO Library

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



A screenshot of a terminal window titled "pi@raspberrypi01: ~". The window shows the command-line interface for installing the GPIO library. The user runs three commands: "sudo apt-get install python-dev", "sudo apt-get install python3-dev", and "sudo apt-get install python-rpi.gpio". The terminal output indicates that "python-dev" and "python3-dev" are already at their newest versions, while "python-rpi.gpio" is also at its newest version (0.7.0-0.1-bpo10+4). The process involves reading package lists, building dependency trees, and checking state information.

```
pi@raspberrypi01:~ $ sudo apt-get install python-dev  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
python-dev is already the newest version (2.7.16-1).  
0 upgraded, 0 newly installed, 0 to remove and 203 not upgraded.  
pi@raspberrypi01:~ $ sudo apt-get install python3-dev  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
python3-dev is already the newest version (3.7.3-1).  
0 upgraded, 0 newly installed, 0 to remove and 203 not upgraded.  
pi@raspberrypi01:~ $ sudo apt-get install python-rpi.gpio  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
python-rpi.gpio is already the newest version (0.7.0-0.1-bpo10+4).  
0 upgraded, 0 newly installed, 0 to remove and 203 not upgraded.  
pi@raspberrypi01:~ $
```

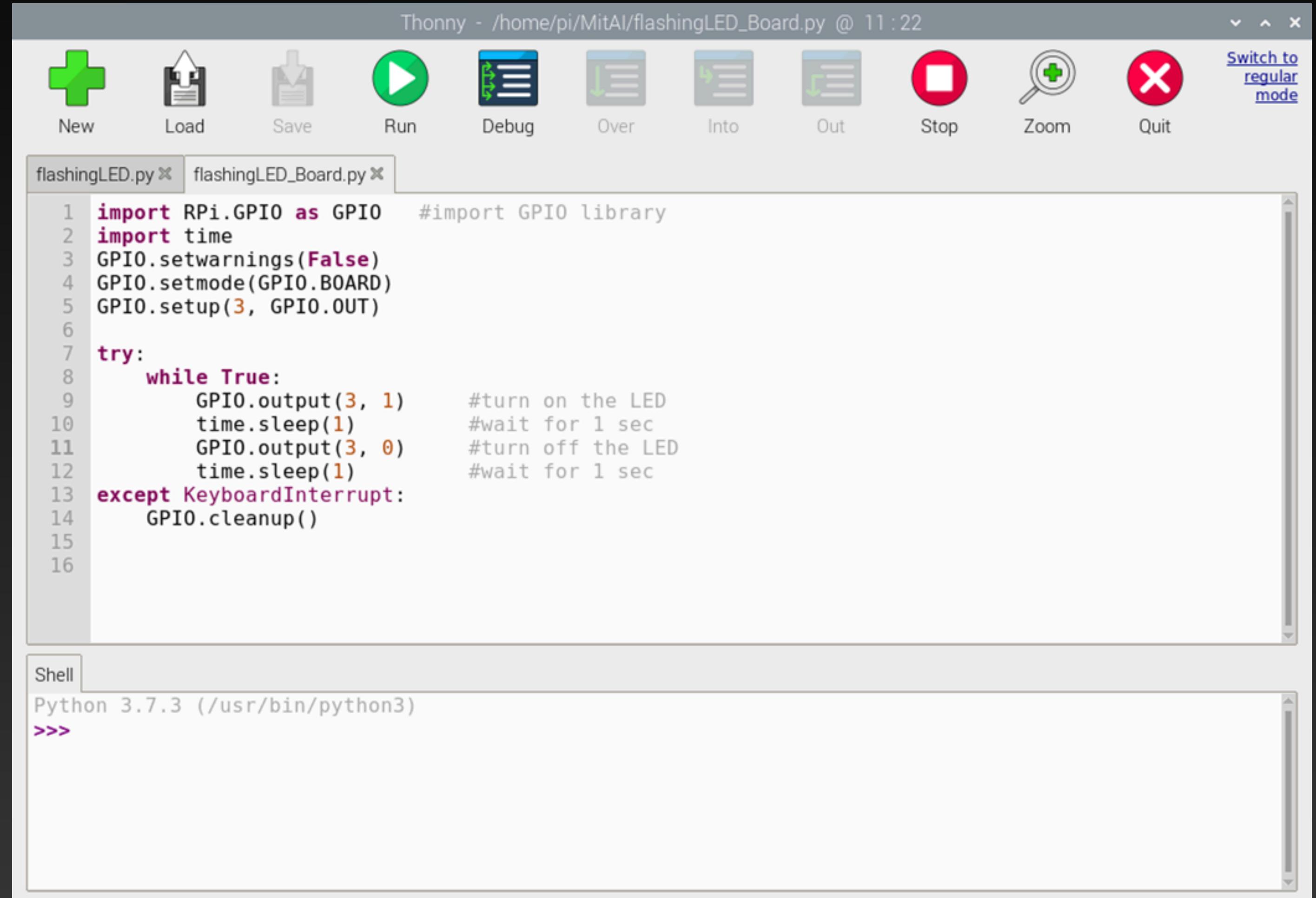
Physical Computing from Python - GPIO Library

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.BUILD)`



The screenshot shows the Thonny IDE interface with the following details:

- Toolbar:** Includes New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit buttons.
- File Tabs:** Two tabs are open: "flashingLED.py" and "flashingLED_Board.py".
- Code Editor (flashingLED_Board.py):**

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BOARD)
5 GPIO.setup(3, GPIO.OUT)

6 try:
7     while True:
8         GPIO.output(3, 1)      #turn on the LED
9         time.sleep(1)          #wait for 1 sec
10        GPIO.output(3, 0)     #turn off the LED
11        time.sleep(1)          #wait for 1 sec
12 except KeyboardInterrupt:
13     GPIO.cleanup()
```
- Shell:** Shows the Python version and prompt:

```
Python 3.7.3 (/usr/bin/python3)
>>>
```

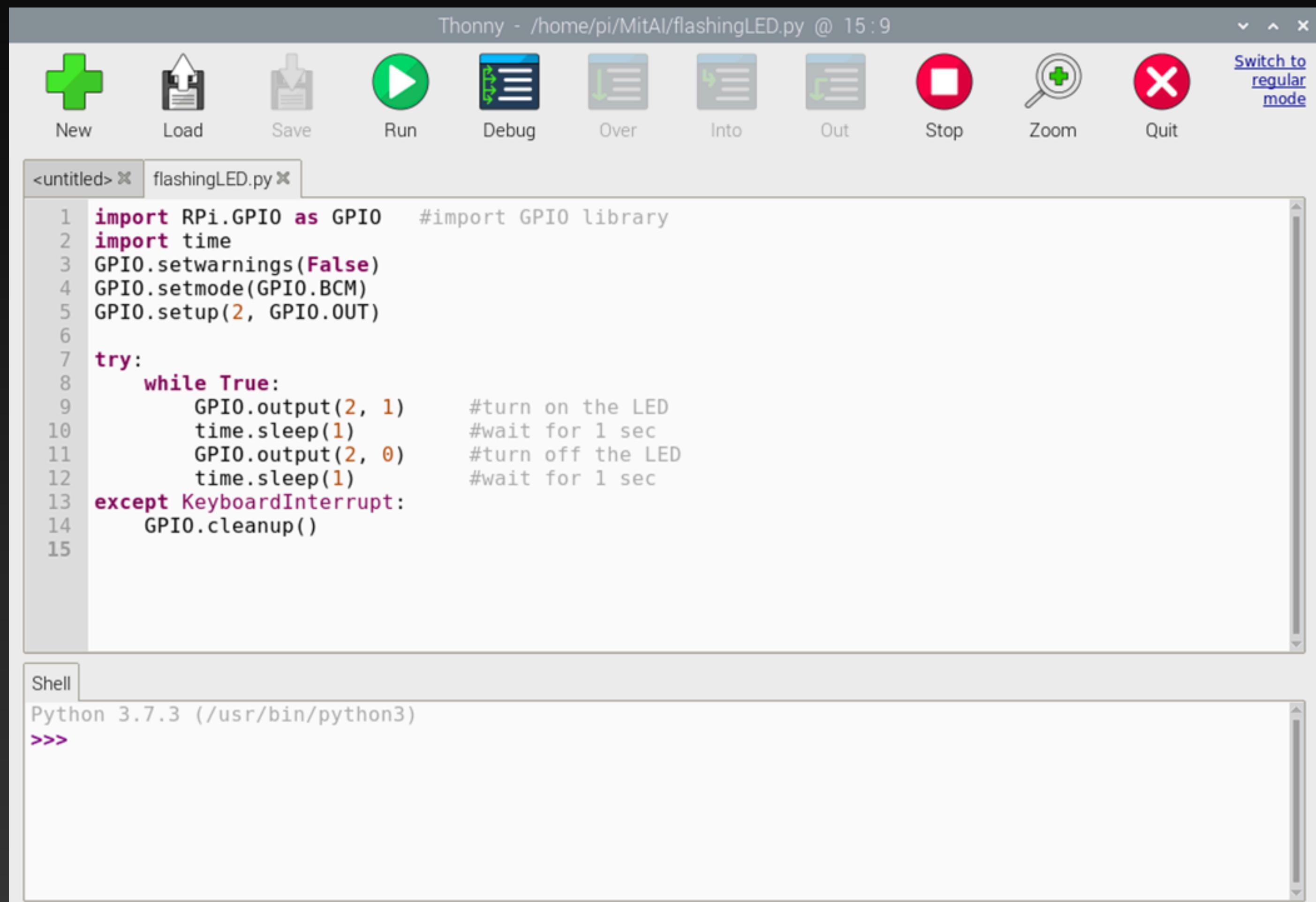
Physical Computing from Python - GPIO Library

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)`



The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - /home/pi/MitAI/flashingLED.py @ 15 : 9". The menu bar includes "File", "Edit", "Run", "Debug", "Tools", "Help", and "Switch to regular mode". The main window has two tabs: "flashlingLED.py" (active) and "<untitled>". The code in "flashlingLED.py" is:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
```

The "Shell" tab at the bottom shows the Python interpreter prompt:

```
Python 3.7.3 (/usr/bin/python3)
>>>
```

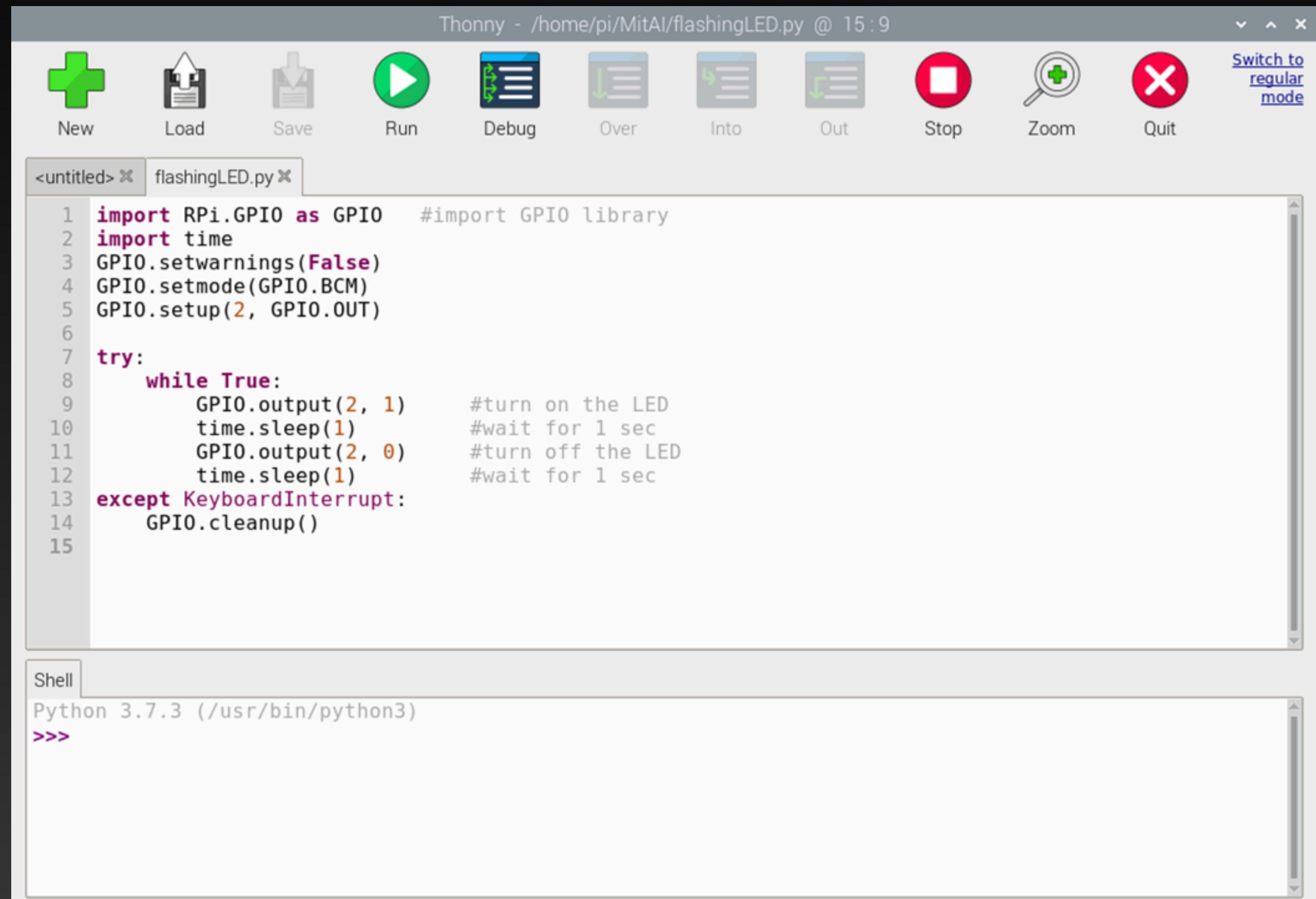
Physical Computing from Python - GPIO Library

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.



The screenshot shows the Thonny IDE interface with a Python script named 'flashingLED.py' open. The script code is as follows:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
15
```

The Thonny interface includes a toolbar with icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. A status bar at the top right indicates 'Thonny - /home/pi/MitAI/flashingLED.py @ 15 : 9'. Below the code editor is a shell window showing the Python version and a prompt: 'Python 3.7.3 (/usr/bin/python3) >>>'. On the far right of the Thonny window, there is a vertical scroll bar and a 'Switch to regular mode' button.

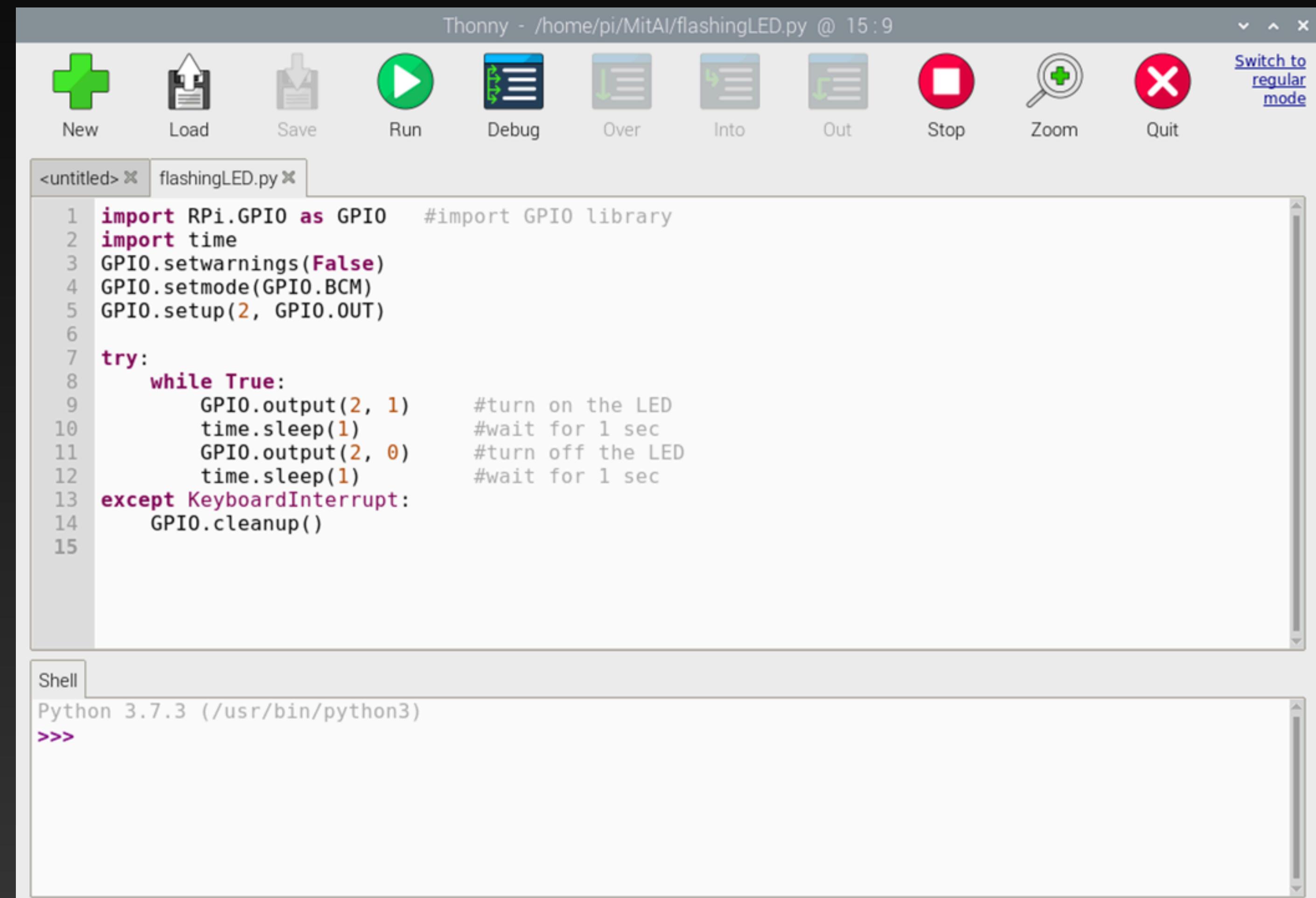
Physical Computing from Python - GPIO Library

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.



The screenshot shows the Thonny IDE interface with a Python script named 'flashingLED.py' open. The script contains the following code:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
15
```

The Thonny interface includes a toolbar with icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. A status bar at the top indicates 'Thonny - /home/pi/MitAI/flashingLED.py @ 15 : 9'. Below the code editor is a shell window showing the Python version: 'Python 3.7.3 (/usr/bin/python3)' and a prompt '>>>'. There is also a 'Switch to regular mode' button in the top right corner.

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)
```

The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - /home/pi/MitAI/flashingLED.py @ 15 : 9". The menu bar includes "File", "Edit", "Run", "Debug", "Tools", "Help", and "Switch to regular mode". The main window has two tabs: "<untitled>" and "flashingLED.py". The "flashingLED.py" tab contains the following code:

```
1 import RPi.GPIO as GPIO      #import GPIO library  
2 import time  
3 GPIO.setwarnings(False)  
4 GPIO.setmode(GPIO.BCM)  
5 GPIO.setup(2, GPIO.OUT)  
6  
7 try:  
8     while True:  
9         GPIO.output(2, 1)      #turn on the LED  
10        time.sleep(1)        #wait for 1 sec  
11        GPIO.output(2, 0)      #turn off the LED  
12        time.sleep(1)        #wait for 1 sec  
13 except KeyboardInterrupt:  
14     GPIO.cleanup()  
15
```

Below the code editor is a "Shell" window showing the Python interpreter prompt:

```
Python 3.7.3 (/usr/bin/python3)  
>>>
```

Physical Computing from Python - GPIO Library

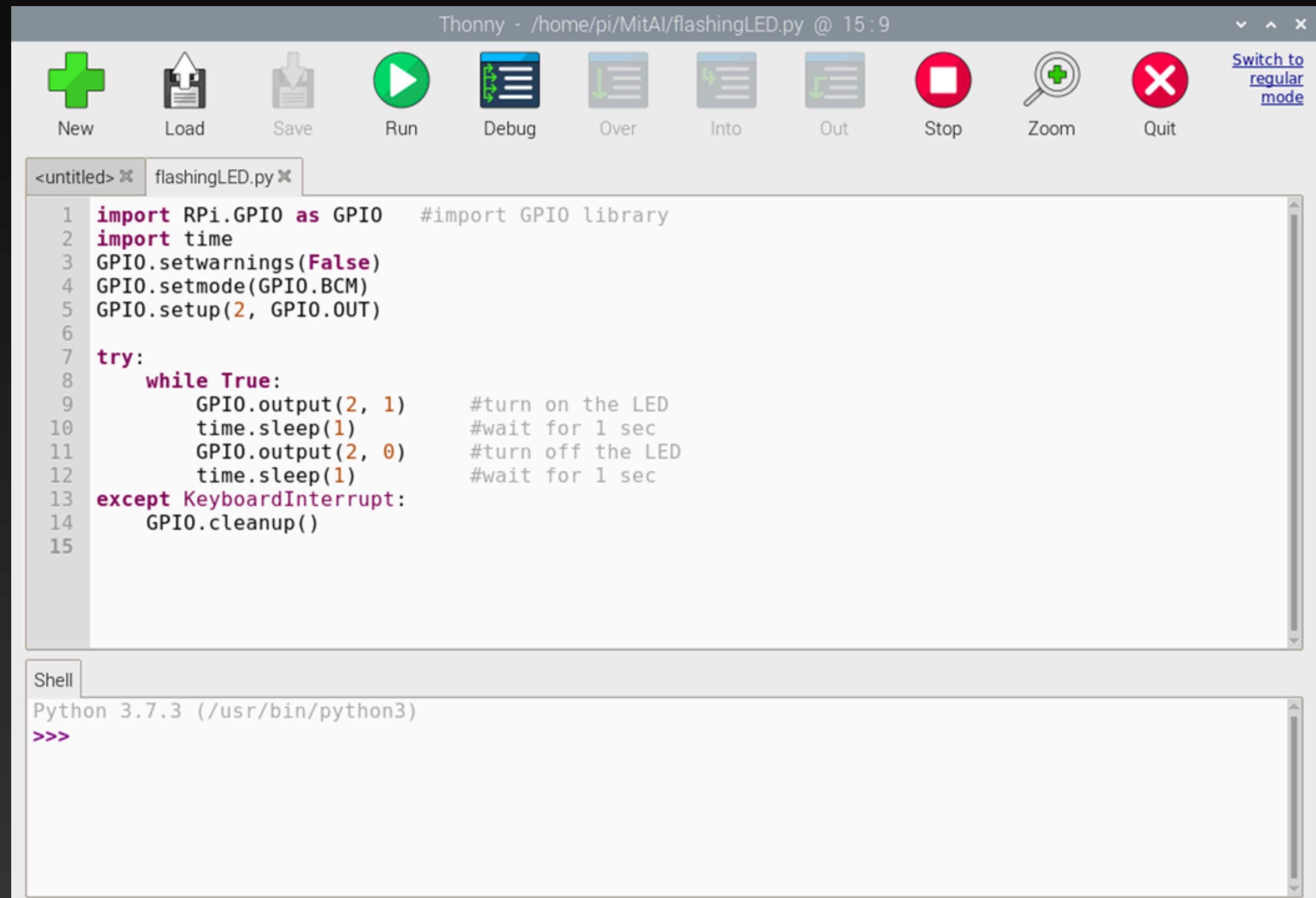
Channel(I/O port pin) Config

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`)



The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - /home/pi/MitAI/flashingLED.py @ 15 : 9". The menu bar includes "File", "Edit", "Run", "Debug", "Tools", "Help", and "Switch to regular mode". The main window has two tabs: "<untitled>" and "flashingLED.py". The code in "flashingLED.py" is as follows:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6 try:
7     while True:
8         GPIO.output(2, 1)      #turn on the LED
9         time.sleep(1)          #wait for 1 sec
10        GPIO.output(2, 0)     #turn off the LED
11        time.sleep(1)          #wait for 1 sec
12 except KeyboardInterrupt:
13     GPIO.cleanup()
```

Below the code editor is a "Shell" tab showing the Python interpreter prompt:

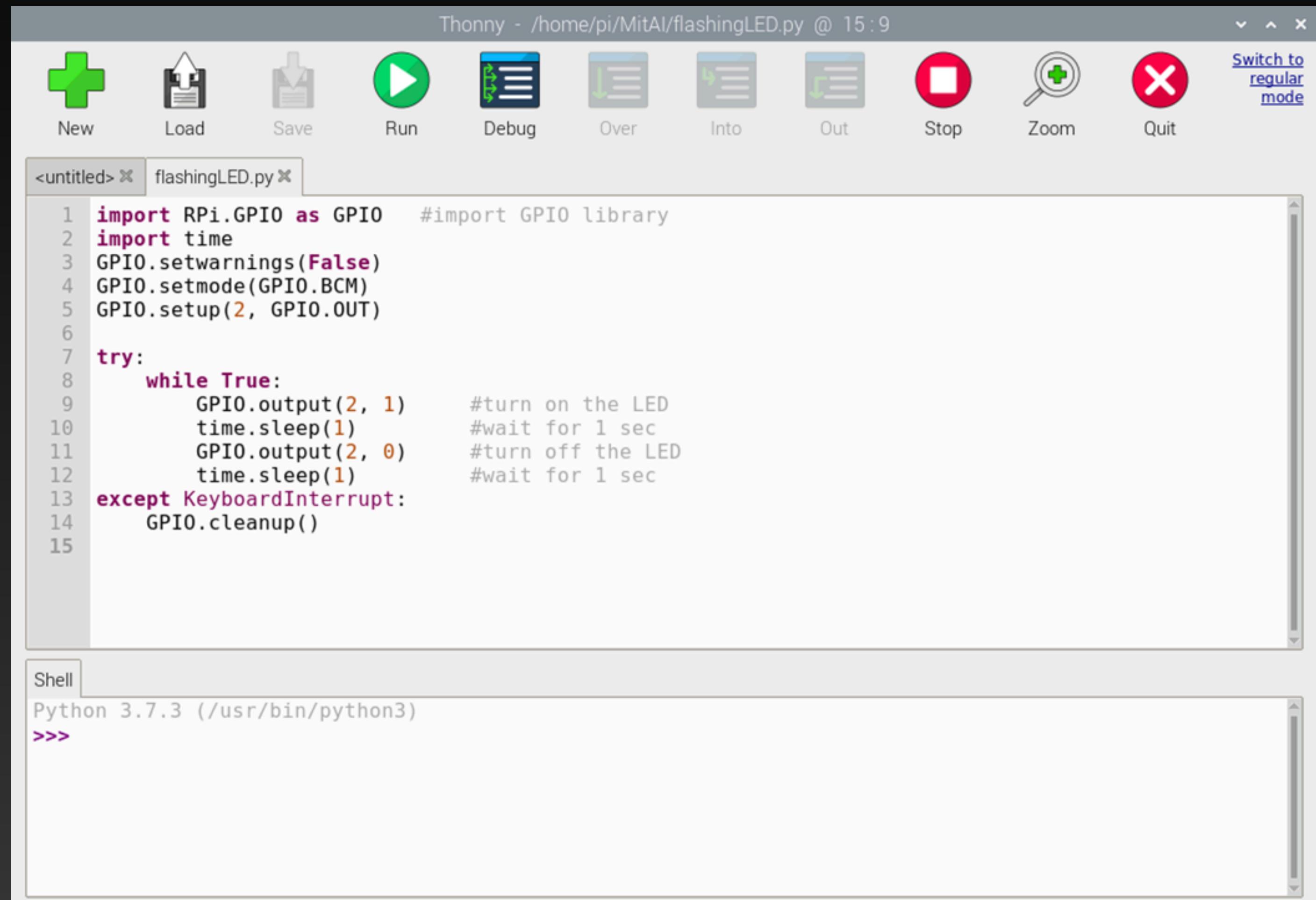
```
Python 3.7.3 (/usr/bin/python3)
>>>
```

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()`



The screenshot shows the Thonny IDE interface with a Python script named `flashingLED.py`. The script code is as follows:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
```

The Thonny interface includes a toolbar with icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. A status bar at the top right indicates the file path `/home/pi/MitAI/flashingLED.py` and the line number `@ 15 : 9`. A "Switch to regular mode" button is also visible.

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:

$$3.3V - 1.8V / 3mA = 500 \text{ Ohm}$$

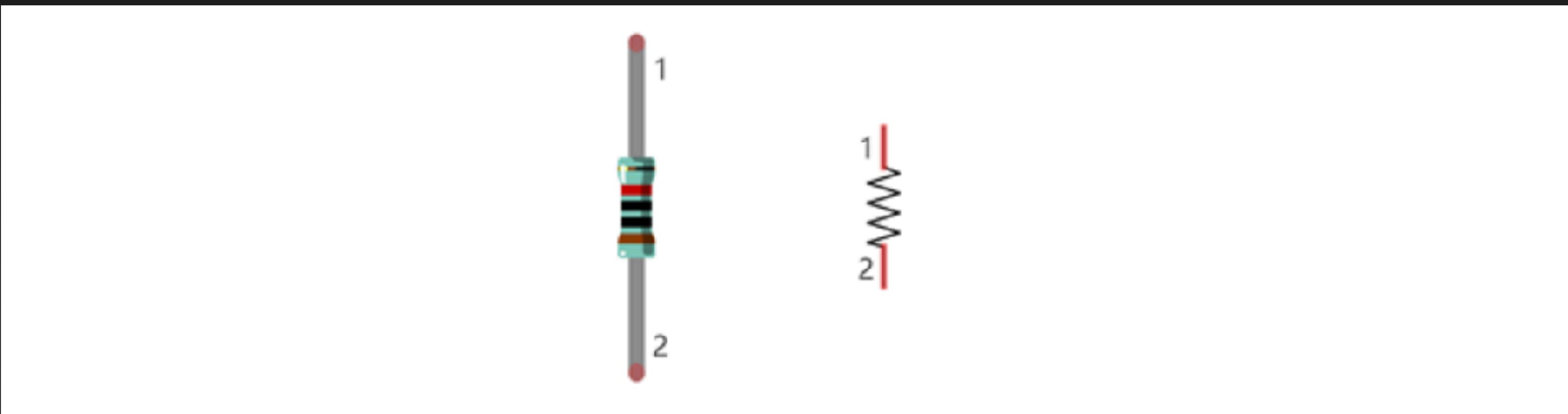


Flashing LED - Accessing the External World

Background info:

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit.



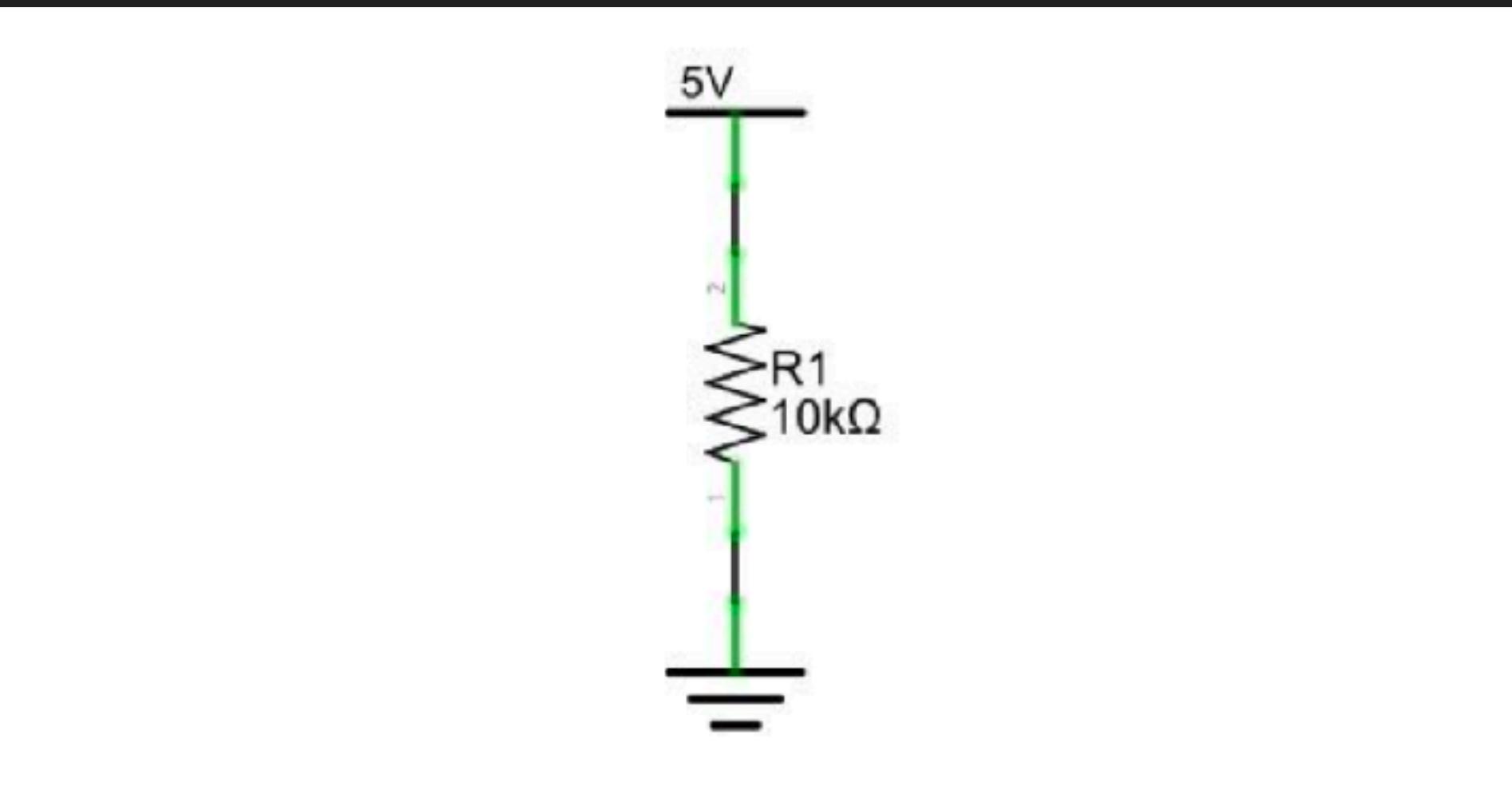
Flashing LED - Accessing the External World

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.

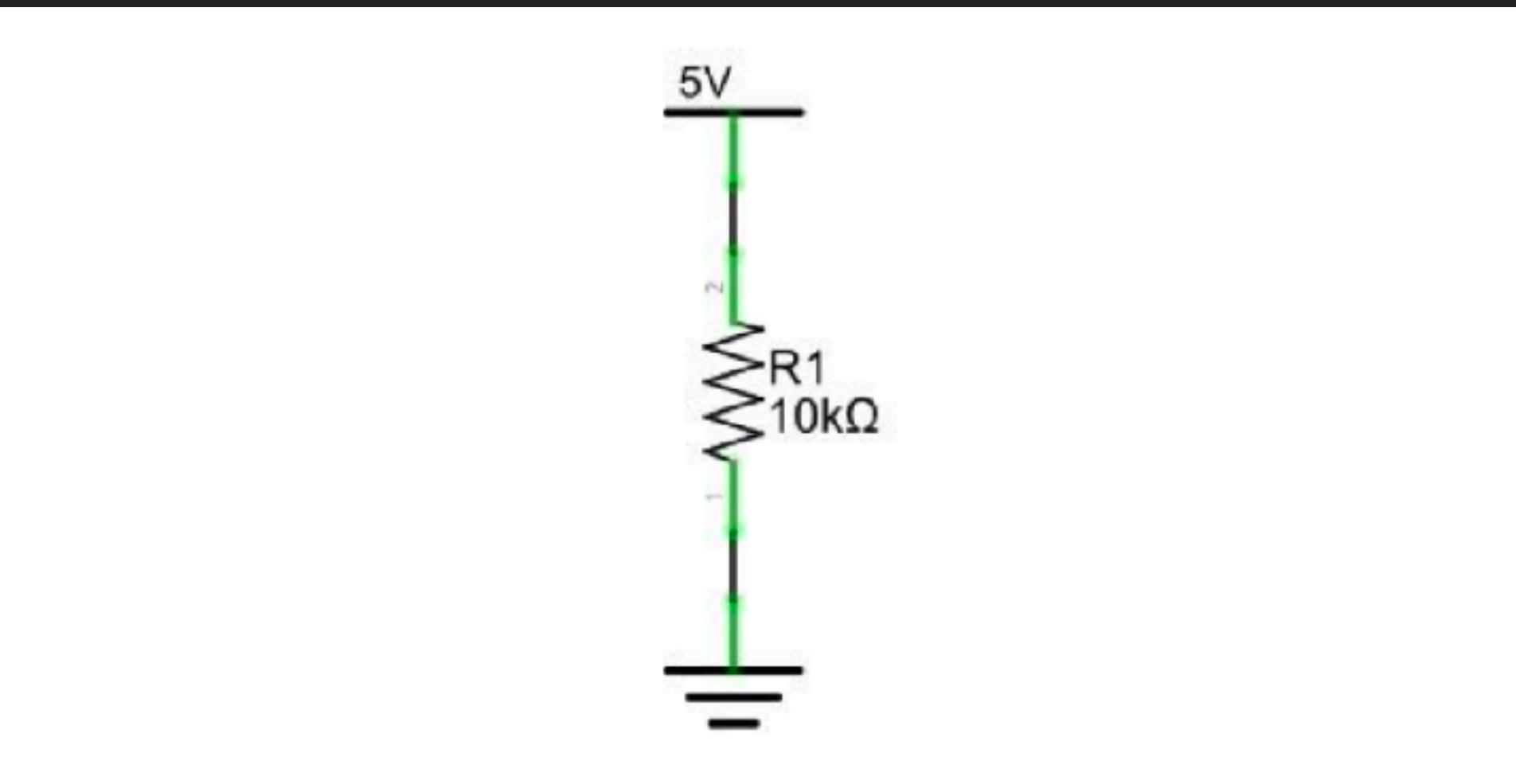
In the following diagram, the current through R1 is: $I=U/R=5V/10k\Omega=0.0005A=0.5mA$.



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.

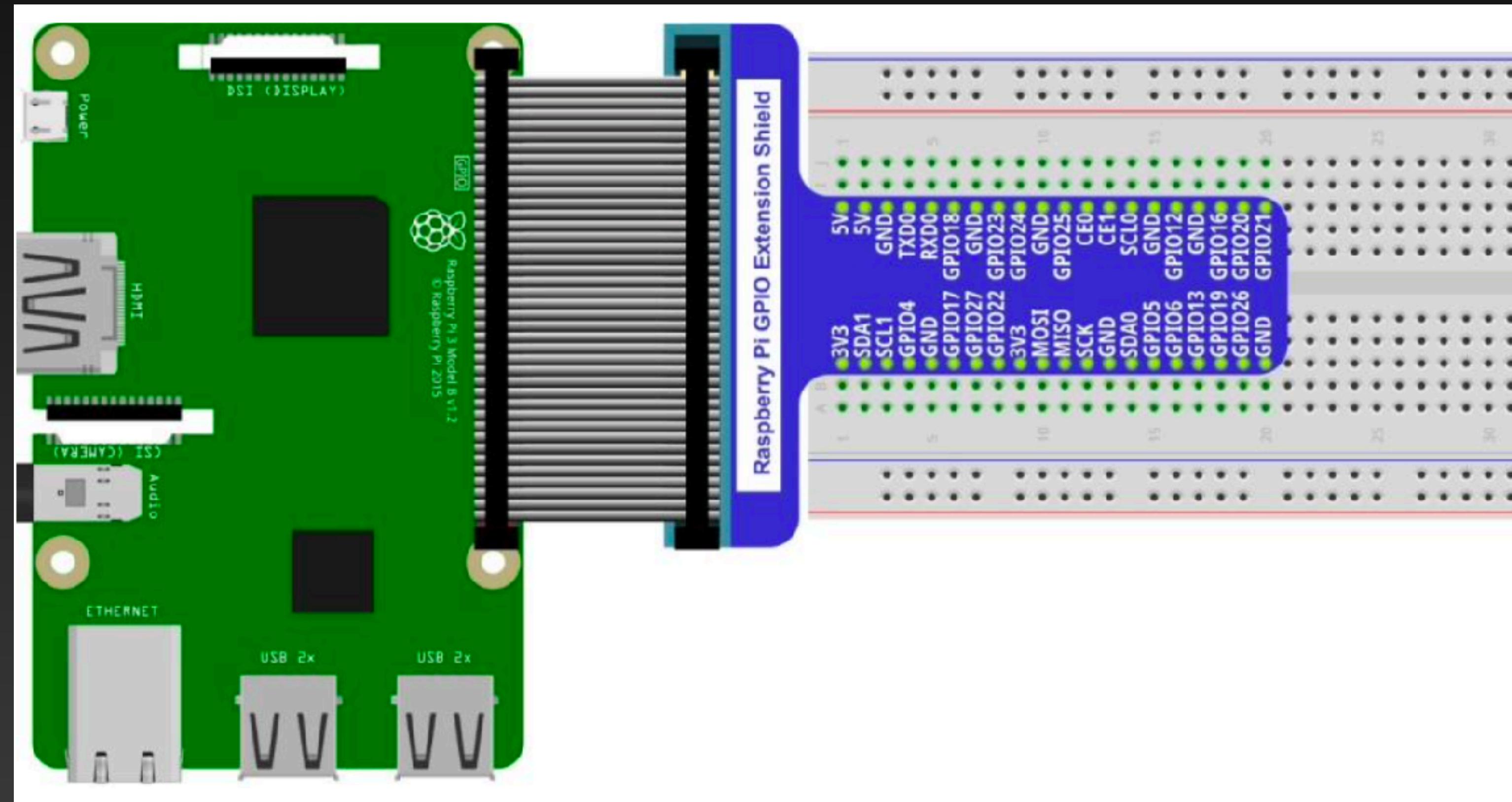
Note: Unlike LEDs and Diodes, Resistors have no poles and re non-polar (it does not matter which direction you insert them into a circuit, it will work the same)



Introducing Extension Board

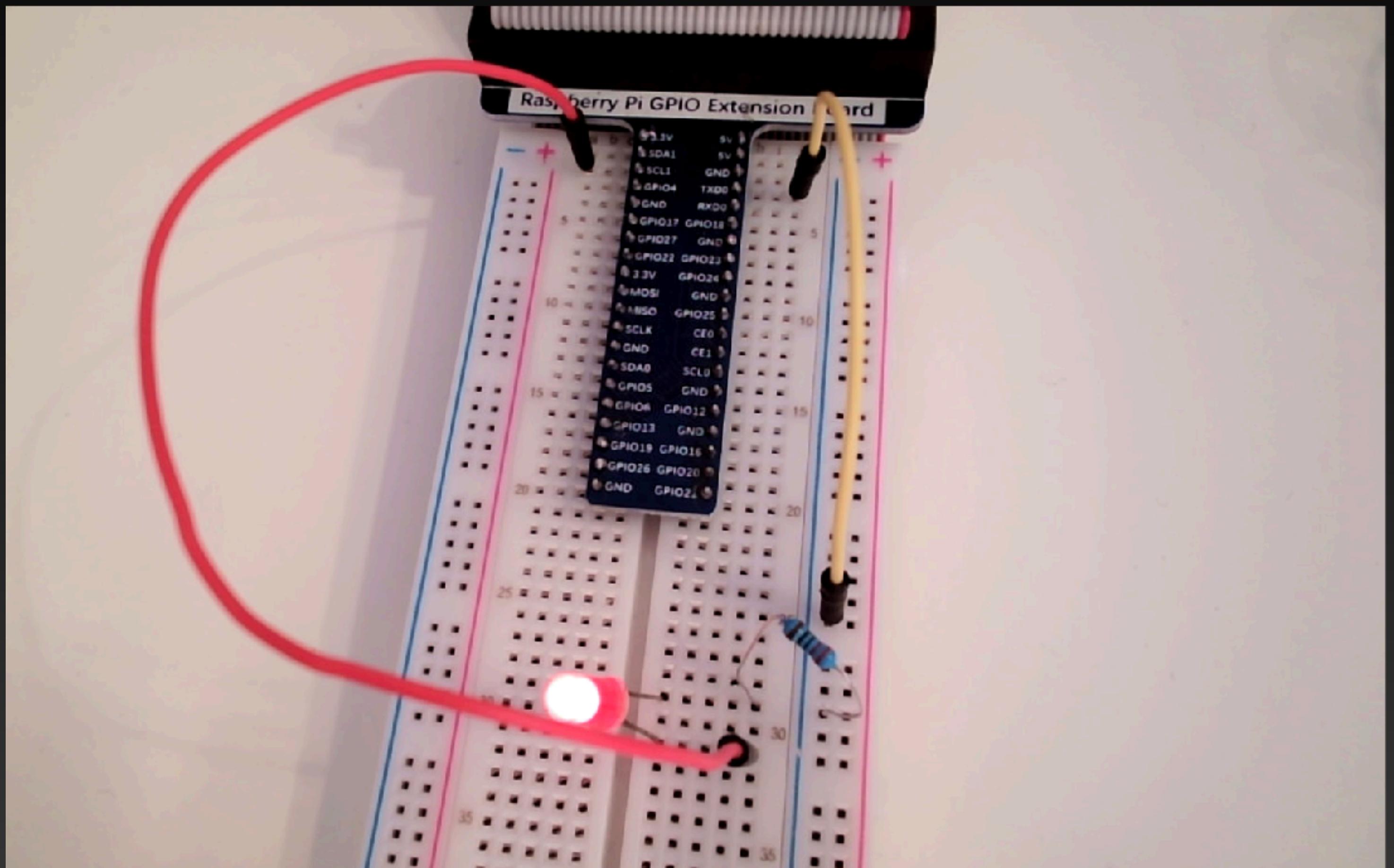
GPIO Extension Board

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.

The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - /home/pi/MitAI/flashingLED.py @ 15:9". The toolbar contains icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. Below the toolbar, there are two tabs: "<untitled>" and "flashingLED.py". The code editor displays the following Python script:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
```

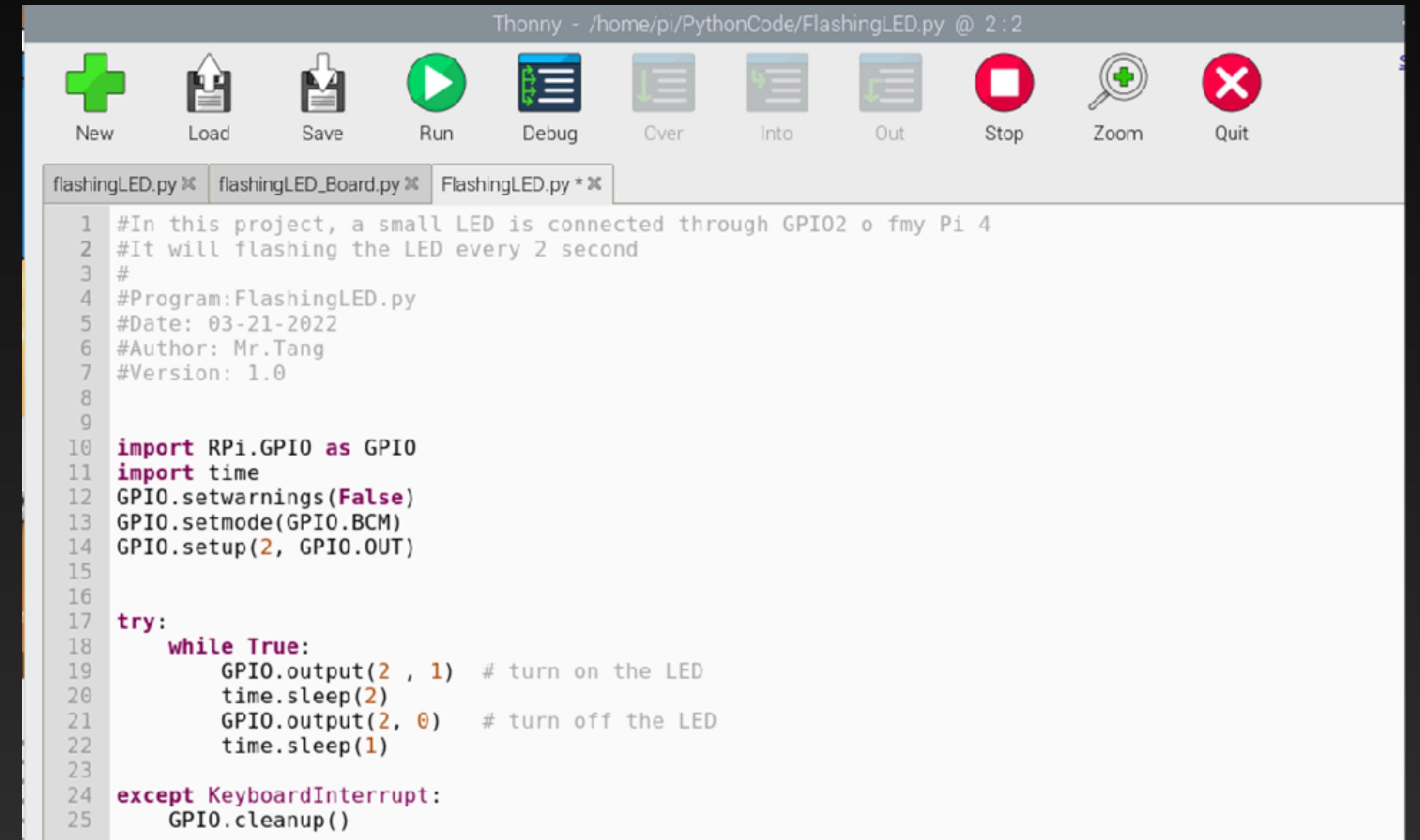
Below the code editor is a "Shell" tab with the text "Python 3.7.3 (/usr/bin/python3)" and a prompt ">>>".

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()
```

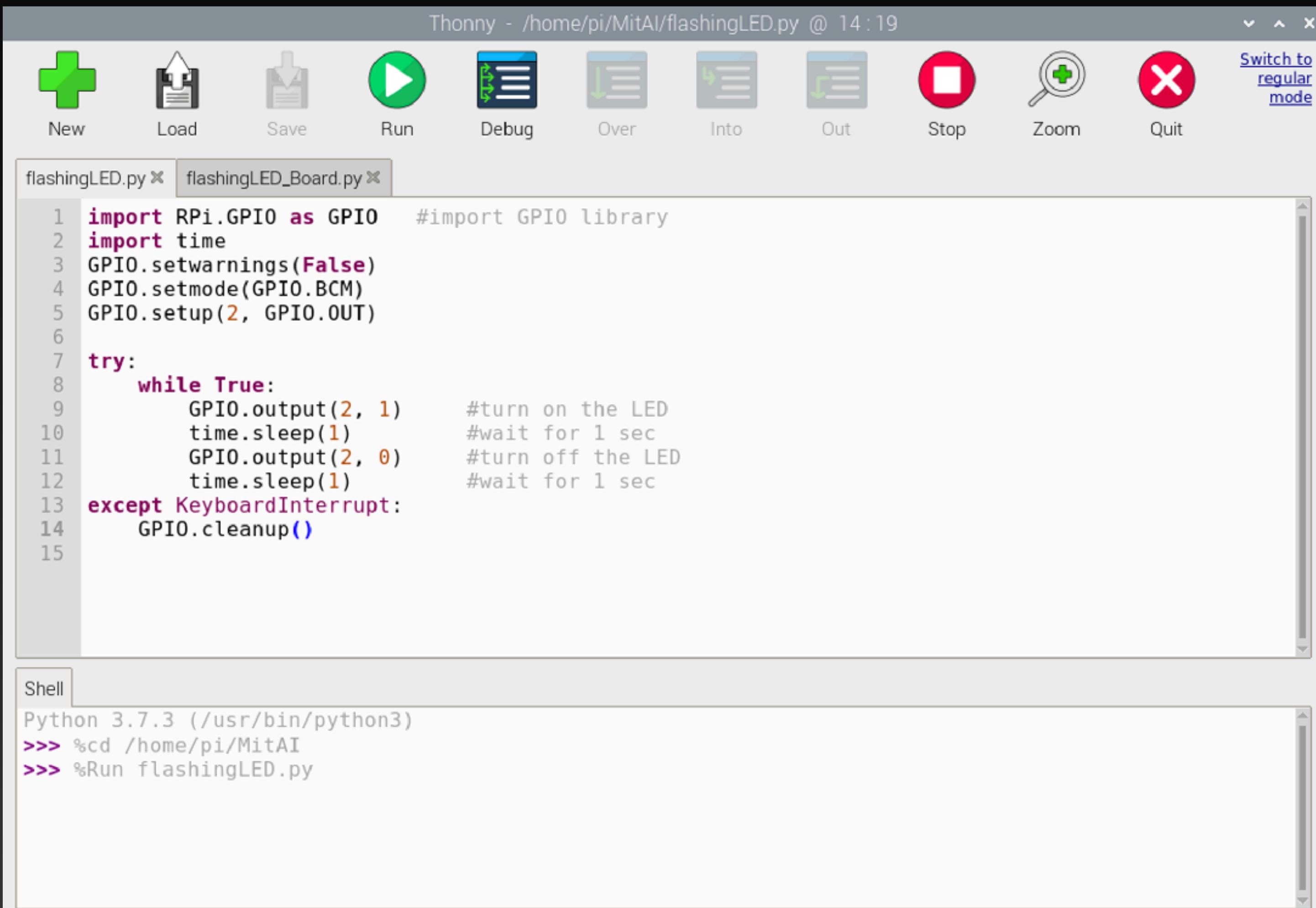


The screenshot shows the Thonny Python IDE interface. The title bar reads "Thonny - /home/pi/PythonCode/FlashingLED.py @ 2 : 2". The toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The code editor window displays the following Python script:

```
flashingLED.py ✘ flashingLED_Board.py ✘ FlashingLED.py * ✘
1 #In this project, a small LED is connected through GPIO2 o fmy Pi 4
2 #It will flashing the LED every 2 second
3 #
4 #Program:FlashingLED.py
5 #Date: 03-21-2022
6 #Author: Mr.Tang
7 #Version: 1.0
8
9
10 import RPi.GPIO as GPIO
11 import time
12 GPIO.setwarnings(False)
13 GPIO.setmode(GPIO.BCM)
14 GPIO.setup(2, GPIO.OUT)
15
16
17 try:
18     while True:
19         GPIO.output(2 , 1) # turn on the LED
20         time.sleep(2)
21         GPIO.output(2, 0) # turn off the LED
22         time.sleep(1)
23
24 except KeyboardInterrupt:
25     GPIO.cleanup()
```

Flashing LED - Accessing the External World

Run the code.



The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - /home/pi/MitAI/flashingLED.py @ 14:19". The toolbar contains icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. A "Switch to regular mode" link is also present. The code editor window displays two files: "flashingLED.py" (selected) and "flashingLED_Board.py". The code in "flashingLED.py" is:

```
1 import RPi.GPIO as GPIO      #import GPIO library
2 import time
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(2, GPIO.OUT)

6
7 try:
8     while True:
9         GPIO.output(2, 1)      #turn on the LED
10        time.sleep(1)        #wait for 1 sec
11        GPIO.output(2, 0)      #turn off the LED
12        time.sleep(1)        #wait for 1 sec
13 except KeyboardInterrupt:
14     GPIO.cleanup()
```

The shell window at the bottom shows the command line:

```
Shell
Python 3.7.3 (/usr/bin/python3)
>>> %cd /home/pi/MitAI
>>> %Run flashingLED.py
```

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