**NCV2 Robotic Fundamentals Module 5**

**Module 5: Programming**

**(RaspberryPi – Scratch development environment for GPIO programming, and Pythin using IDLE and MU)**

## Connected5.1 Basic concepts of GPIO programming

Content:

* Computer programming principles
* GPIO Programming

*Learning Outcomes:*

*Students should be able to:*

### 5.1.1 Define the term Programming

**DEFINITION**

Programming is the implementation of logic to facilitate specified computing operations and functionality. It occurs in one or more languages, which differ by application, domain and programming model.

Source code is another name for computer program instructions, and program coding is another name for computer programming. An application is constructed using the semantics and syntax of a programming language. Programming requires knowledge of application domains, algorithms, and programming languages. There are hundreds of programming languages, which can be used to write computer programs and following are a few of them:

* Java
* C, C++ , C#
* Python
* PHP
* Perl
* Ruby
* Javascript
* Kotlin

Programming language logic differs by developer. At a high level, good code can be evaluated with factors such as:

* **Robustness**: Focuses on program continuation capability, regardless of errors or incorrect data
* **Reliability**: Focuses on correct design and algorithm implementation
* **Efficiency**: Focuses on memory, hardware or other properties used to optimize programs
* **Readability**: Proper documentation and indentation availability, which provides insight to other program developers or designers

### 5.1.2 Explain the term physical computing

Physical computing is the application of physical, embedded interactive systems with microcontrollers that can sense their environment and/or control outputs like lights, screens, and motors. It is creative and educational to put together a physical computer's hardware components and program it to perform the desired function. There are many established physical computing products on the market, including the BBC Microbit, Raspberry Pi, Arduino, and Circuit Playground. In this module, we are going to make use of the Raspberry Pi with Scratch and Python.

**eLink**

<https://www.youtube.com/embed/XYXLamqbl3k>

### 5.1.3 Describe the term electrical system in terms of inputs and outputs and basic signal processing

There are three fundamental components of electrical systems: resistors, capacitors, and inductors, which are correspondingly defined by resistance, capacitance, and inductance, which are generally deemed lumped parameters.

Electrical grids, which supply electricity to buildings and businesses over a wide area, are examples of power systems. Generators that produce electricity, transmission systems that move electricity from generating centres to load centres, and distribution systems that deliver electricity to surrounding residences and businesses make up the electrical grid. Additionally, smaller power systems can be found in residences, businesses, hospitals, and industries.

There are two main types of signals used in electronics: ***analogy*** and ***digital*** signals.

 Analog signals are time-varying and generally bound to a range (e.g., +12V to -12V), but within that range, there are infinite possibilities. An analogy signal is one that depends on the properties of a medium to convey its information, such as electricity moving through wires. Signals can be represented by voltage, current, or frequency variations. The analogy signal is often calculated in response to changes in sound, light, temperature, position, pressure, or other physical phenomena.  When plotted on a voltage vs. time graph, an analogy signal should produce a smooth and continuous curve as shown on Figure 5.1

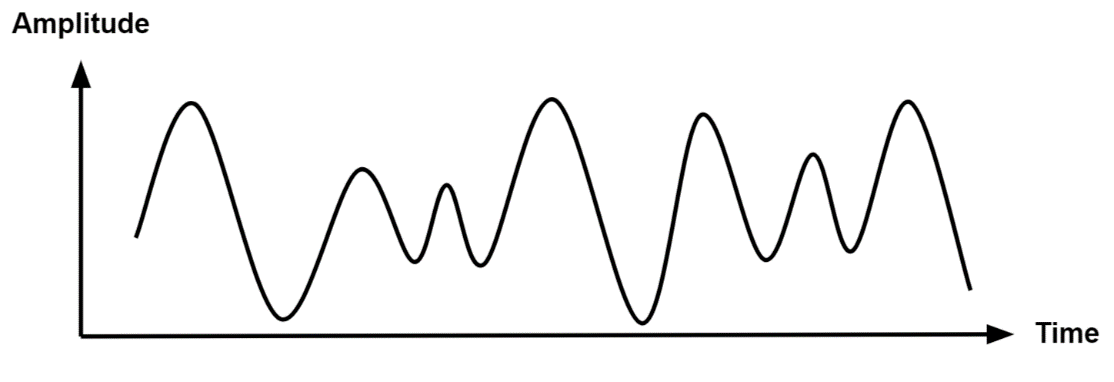


Figure 5. 1: Analogue signal

**Digital Signal**

Digital signals are signals that represent data as discrete values. Digital signals can only have one value at a time from a finite set of possible values. With digital signals, the physical quantity representing the information can be many things:

* Variable electric current or voltage
* Phase or polarization of an electromagnetic field
* Acoustic pressure
* The magnetization of a magnetic storage media

When plotted on a voltage vs. time graph, an analogy signal should produce a smooth and continuous curve as shown on Figure 5.2

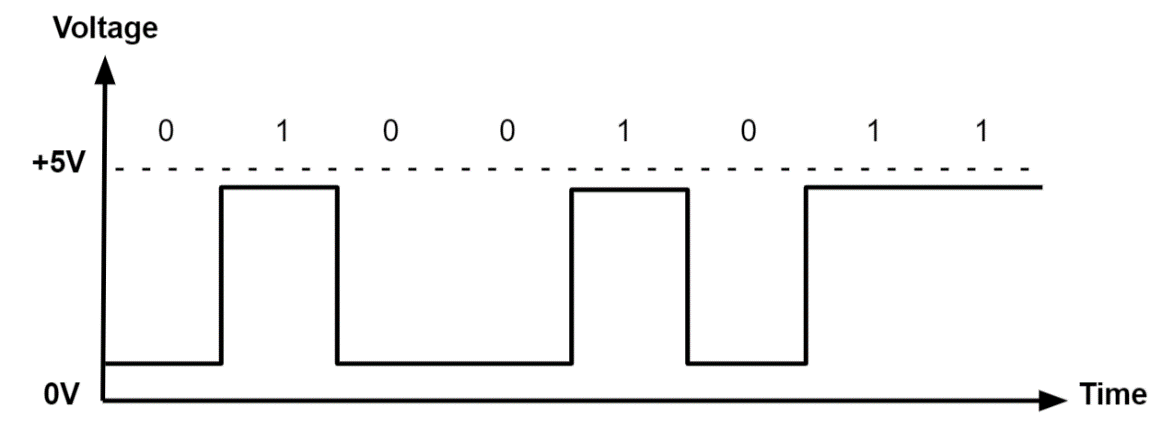


Figure 5. 2:Digital Signal

### 5.1.4 Expand the term GPIO

GPIO stands for General-Purpose Input/Output. A general-purpose input/output (GPIO) is an uncommitted digital signal pin on an integrated circuit or electronic circuit board that may be used as an input or output, or both, and is controllable by software.

GPIOs have no predefined purpose and are unused by default. Integrated circuit (IC) GPIOs are implemented in a variety of ways. Some ICs provide GPIOs as a primary function whereas others include GPIOs as a convenient "accessory" to some other primary function.

**What Is GPIO Used For?**

The most common use for GPIO is to operate custom electronics. Whether you are building your robot arm or a Do It Yourself (DIY) weather station, a GPIO interface lets you customize signals so that they operate your equipment correctly.

### 5.1.5 Location of GPIO pins

An important feature of the Raspberry Pi is the row of GPIO pins located at the top edge of the single-board computer. These pins connect the Raspberry Pi to the outside world physically. At their most basic level, they can be thought of as switches, which you can turn on or off (input) or the Pi can turn on or off (output).

The GPIO pins allow the Raspberry Pi to control and monitor the outside world by being connected to electronic circuits. The Pi can control LEDs, turn them on or off, run motors, and many other things. It is also able to detect whether a switch has been pressed, the temperature, and the light. We refer to this as "physical computing."

### 5.1.6 Explain the purpose of the GPIO pins

There are 40 pins on the Raspberry Pi4 (26 pins on early models), and they provide various functions.

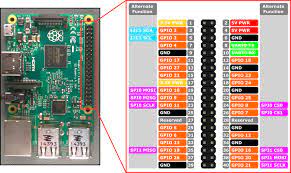


Figure 5. 3: GPIO pins on Raspberry Pi

GPIO interfaces are usually used in conjunction with a "breadboard". A breadboard is a rectangular plastic board with a bunch of tiny holes in it. These holes let you easily insert electronic components to prototype an electronic circuit. You can prototype circuits by adding, removing, or moving electronic components around. Many projects that involve devices like a Raspberry Pi allows you to assemble your device on a breadboard and then connect it to the GPIO pins using wires. GPIO-General Purpose Input/Output, aka "BCM" or "Broadcom". These are the big numbers, e.g., "GPIO 22". You will use these with RPi.GPIO and GPIO Zero. Physical or "Board" corresponds to the pin's physical location on the header. These are the small numbers next to the header, e.g. "Physical Pin 15". BCM: the number to be used in your Python or Scratch code to specify the GPIO to be used. BCM refers to the "Broadcom SOC channel" number, which is the numbering inside the chip that is used on the Raspberry Pi. These numbers changed between board versions.

Your microcontroller or computer must be able to understand the signals coming into the GPIO interface, as well as connect the GPIO pins to the appropriate connectors on your external circuit board. That means you need software, which you usually have to write! Python software is often used on Raspberry Pi systems to tell the GPIO controller what to send and what signals to listen for.

In some cases, the Raspberry Pi comes built into a keyboard. Such models do not require the purchase of a casing. The GPIO will be covered with a rubber enclosure to avoid dust and protect the pins from the outside world.

### 5.1.7 Basic commands in terms of communication to the various GPIO pins

To have the Raspberry Pi communicate with the outside world, it requires a programming language and the most common one is Python. We are also going to use Scratch as it allows us to learn development concepts easily.

**Scratch**

Scratch is an easy-to-use block-based visual programming software that can run on a Raspberry Pi. With Scratch on this Raspberry Pi, you can create animations, games, and more using a straightforward drag-and-drop interface. Programming with Scratch is a great way to get students interested in computing and get them started with programming. When running our Scratch program on a Raspberry Pi, we can make use of the "GPIO" pins to interface with the outside world.

**Starting Scratch**

Once Scratch 3 is installed on the Pi, go ahead and open up Scratch 3 by clicking the Raspberry Pi icon on the top left of the desktop, then Programming>Scratch3

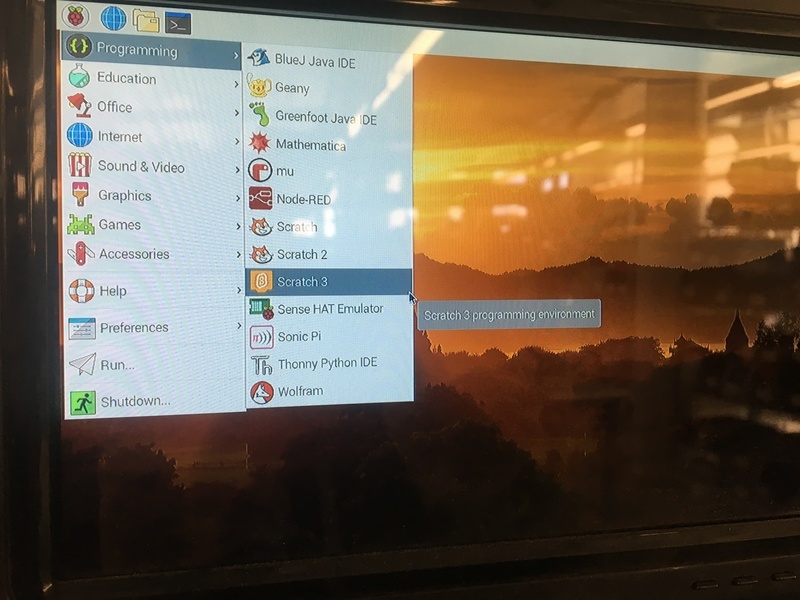


Figure 5. 4:Starting Scratch on Raspberry Pi

**Add the Raspberry Pi GPIO extension**

* Click the blue icon at the bottom left of the application to open the extensions.
* Select the extension entitled "Raspberry Pi GPIO". Now we have access to some awesome GPIO functionality.



Figure 5. 5: Four common blocks of GPIO on Raspberry Pi

The GPIO extension gives you the flexibility to connect and control a whole host of electronic devices.

**Python**

Python is an interpreted, high-level, general-purpose programming language that has been around since 1991. It is currently one of the programming languages with the quickest growth. The "Pi" in Raspberry Pi standards stands for "Python Interpreter," indicating that this is the platform's suggested language.

Being an interpreted language, Python allows you to type in commands and run them interactively without needing to compile the program. They undergo run-time compilation into an intermediate bytecode, which a virtual machine then runs.

Every Raspberry Pi installation comes with an **Integrated Development and Learning Environment**, which you’ll see shortened to IDLE or even IDE.  These are a class of applications that help you write code more efficiently. While there are many IDEs for you to choose from, Python IDLE is very bare-bones, which makes it the perfect tool for a beginning programmer.

The best place to experiment with Python code is in the interactive interpreter, otherwise known as a **shell**. The shell is a basic Read-Eval-Print Loop (REPL). It reads a Python statement, evaluates the result of that statement, and then prints the result on the screen. Then, it loops back to read the next statement.

Figure illustrates The Python IDLE ide code where two numbers are added and displayed on the shell.

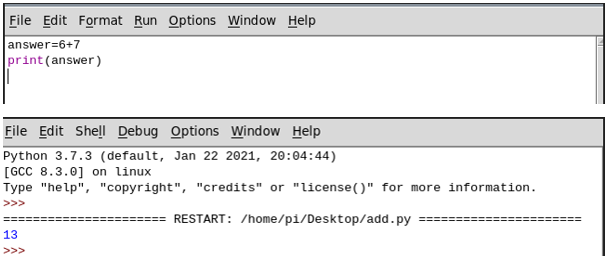


Figure 5. 6: Python IDLE screen

GPIO pins are used to connect the Raspberry Pi to electronic components. To interact with different physical components, Python has two libraries known as RPi.GPIO and gpiozero. By invoking these libraries, you can take control of the GPIO system and make it do your bidding. You have many different options for writing Python on Raspberry Pi. In this module, we will use the Mu editor to program our Raspberry Pi. There is not much of a difference with IDLE except that Mu has a better interface for both input and output.

### 5.1.8 Explain how programming is used to send and receive signals from pins

The Raspberry Pi has a 40-pin GPIO (General Purpose Input/Output) connection, which makes it very easy to connect to the outside world. To connect the GPIO to external sensors, you can:

* Connect the sensors directly to the GPIO pins using jumper wires
* Connect the GPIO pins to a ribbon cable, which in turn connects it to a breadboard. The Adafruit Pi T-Cobbler Plus - Breakout + Cable for Raspberry Pi A+/B+/Pi 2/Pi 4 is one such product. This option is ideal during the prototyping phase.

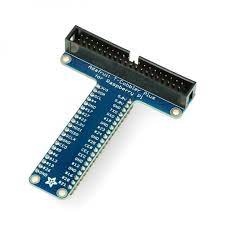


Figure 5. 7:The labels on the various pins on the Adafruit Pi T-Cobbler Plus

When using programming languages such as Python or Scratch, you will need to import libraries, which allows the computer to communicate with the external components attached to the computer through the GPIO. Among the easiest ways to control GPIO pins is by using the gpiozero library in Python. If you have written any Python code before, you will pick this up easy. Gpiozero is installed by default on Raspbian Desktop images. If you are using Raspbian Lite or a different operating system, you may need to install it.

### 5.1.9 Basic principles involved in programming

No matter how they are programmed, computers work through input, processing, and output. Data or commands input into a computer must be processed before output can be viewed.

**Input**

Data entered or received by a computer is referred to as input. This could involve the user tapping a touchpad, clicking a mouse to pick an item on the screen, or pressing a key on a keyboard. While some inputs give the computer instructions on what to perform, others give it data to process. It is not necessary for humans to always initiate input. As an illustration, a computer may receive a message from another device or get data from a temperature sensor.

**Processing**

The process determines what the computer does with the input. The same input can be processed in different ways. A program contains the set of instructions that define the process.

**Outputs**

The output is how the computer presents the results of the process. Outputs can be returned to the user in many ways such as text on a screen, printed materials, or as sound from a speaker.

Here is a full example of input output and processing. We will use an example of a camera.

* Input: pressing the button on the top
* Process: it captures and stores a picture
* Output: the image is shown on the screen

**Discussion Points**

* Think about your day. How many times have you used a computer? Describe some of these times.
* Look at some of the examples you listed in the question above: – What was the input? – What was the process? – What was the output?

### 5.1.10 Examples of programming languages suitable for GPIO programming

Raspberry Pi supports several programming languages such as C, C++, Ruby, Perl, Scratch, Python, Java and PHP but, to access GPIO, all of them require additional libraries.

### 5.1.11 Start an applicable programming IDE from the OS

An integrated development environment (IDE) is software for building applications that combines common developer tools into a single graphical user interface (GUI). An IDE typically consists of:

* **Source code editor**- A text editor that can assist in writing software code with features such as syntax highlighting with visual cues
* **Local build automation-** Utilities that automate simple, repeatable tasks as part of creating a local build of the software for use by the developer,
* **Debugger-** A program for testing other programs that can graphically display the location of a bug in the original code.

**How to start Python on Raspberry Pi**

Python can be programmed using a variety of IDE’s on the Raspberry Pi such as Mu, IDLE and Thorny. For the purposes of this module, we are going to use Mu.

**How to start Mu to program in Python**

Open Mu by selecting it from the Programming menu.

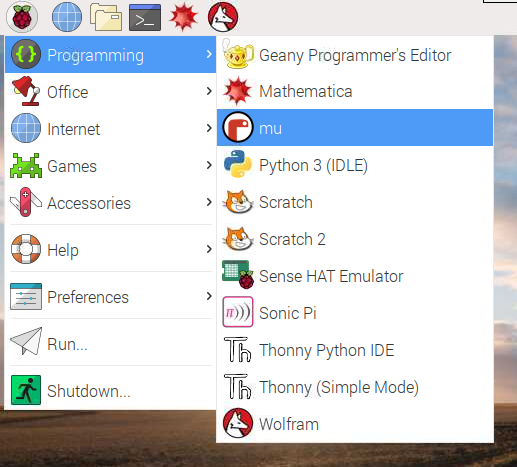


Figure 5. 8: Starting Mu

.

The interface looks like the one on Figure 5.8.

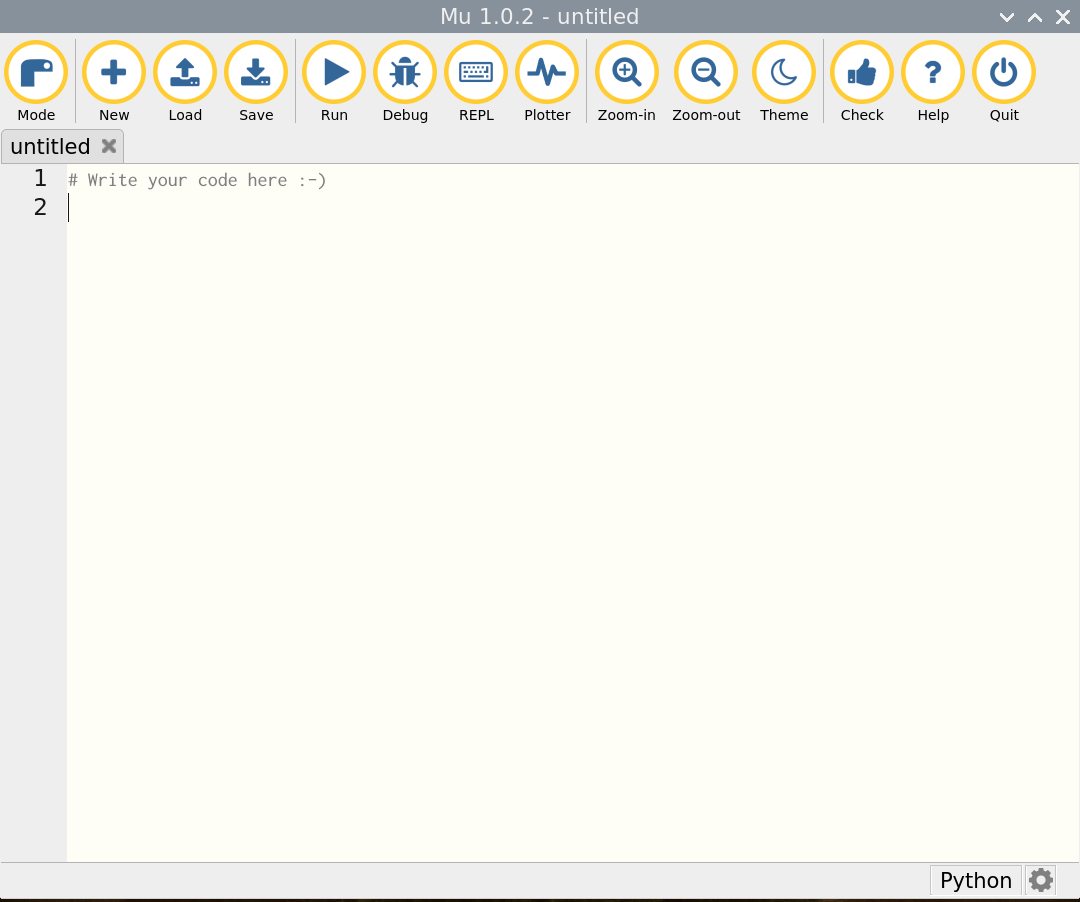


Figure 5. 9: Mu interface

### 5.1.12 Start or open an existing project (code project)

Once Mu IDE is open, let us create a simple Hello World. The program will print the text inside the parenthesis. Please take note that Python text can be in double quotes or single quotes. As a programmer, you just need to be consistent. Mixing single and double quotations generates a syntax error. See sample code in Figure 5.9.

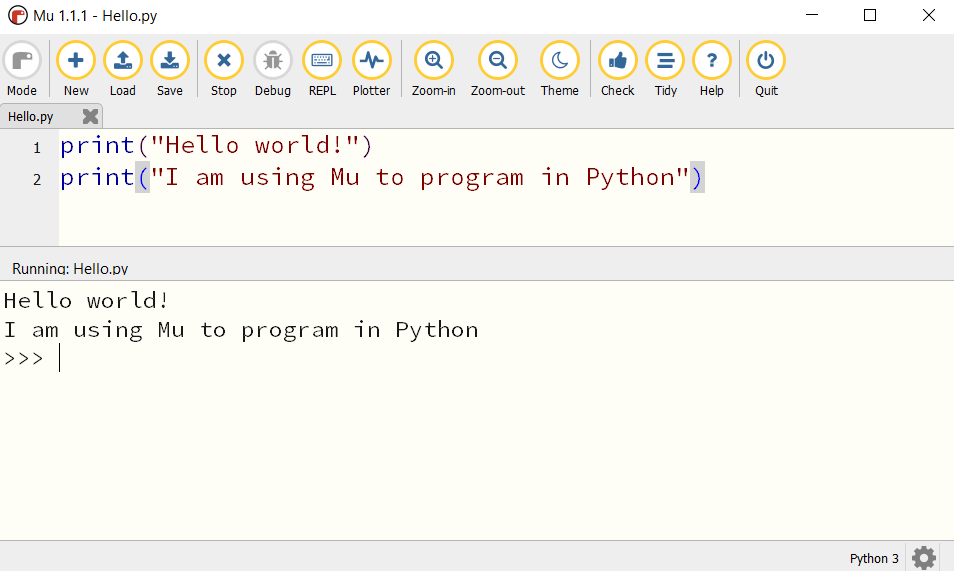


Figure 5. 10: Sample Python code

The keyword print allows whatever is inside the parenthesis to be printed as output.

### 5.1.13 Modify an existing code project based on new requirements

We are going to modify the program in section 5.1.12 to allow the user to input two numbers and add the two. The program will print the output. Here is the modified code:

print("Hello world!")

#accepting user input and casting the value to integer

number1=int(input("Please enter value of first number: \t"))

number2=int(input("Please enter value of second number: \t"))

#processing the addition of number1 and number2 storing result in answer

answer=number1+number2

#displaying the output

print(number1, "+", number2 , "=" , answer)

Output

Hello world!

Please enter value of first number: 6

Please enter value of second number: 9

6 + 9 = 15

On the above example, we introduced variables. Variables are spaces in memory location whose values can change. We use three variables in our program and these are number1, number2 and answer. We use an addition operator to do our addition. The “Run” button execute current script. When this happens the textual **input** and **output** of the program is displayed in a panel between the text editor and Mu’s footer. For further understanding, we refer you to the programming module.

### 5.1.14 Save an existing project

To save an existing project, select the save button  and type in the file name. make sure to select the directory of choice.

5.1.15 Import applicable libraries or extensions

We demonstrated how we can import Raspberry Pi extensions in Scratch (refer to section 5.1.7).

For Python, its easy to import libraries- we make use of the keyword *import <libraryname>*. In this section, we present four Raspberry Pi GPIO programming libraries: WiringPI, Pigpio, Gpiozero, RPI.GPIO. One of the libraries which we use so often in Python programming with Raspberry is the gpiozero. Officially endorsed by the Raspberry Foundation, gpiozero is a Python-only library that not only accesses the GPIO pins, but also provides direct hardware support for working with actuators and sensors. For connecting other devices, only SPI is supported — I2C was announced, but is not included yet.GPIO Zero is installed by default in the Raspberry Pi OS desktop image, and the Raspberry Pi Desktop image for PC/Mac, both available from raspberrypi.org.

**Rpi.GPIO**

The RPI.GPIO library is another Python-only library. It provides basic interactions with the GPIO pins, but no implementation of any connection protocol yet.

**Pigpio**

The pigpio is an actively developed library with an impressive set of features: All GPIO pins of the Raspberry Pi can be read, written to, attached to interrupt handlers, and output PWM signals at the same time.

**Wiring Pi**

The WiringPi library is an often used, functional rich library that supports I2C and SPI connections. It is written in C and provides Python bindings. The library is unfortunately deprecated.

Here is an example of importing gpiozero library to power three LED lights which resemble traffic lights. We use three LEDs of different colours: Red Orange, and Green.

*#Program to turn ON/OFF the lights using Raspberry Pi***from** gpiozero **import** LED  
**from** time **import** sleep  
*#declaring the 3 variables -red, amber and green*red = LED(22)  
amber = LED(27)  
green = LED(17)  
*#Turning On/Off the lights using a sequence*red.on()  
sleep(1)  
amber.on()  
sleep(1)  
green.on()  
sleep(1)  
red.off()  
sleep(1)  
amber.off()  
sleep(1)  
green.off()

5.1.16 Differentiate between inputs and outputs from a GPIO perspective

GPIO pins consists of two buffers: input buffer and Output buffer. The other important part is an enable line with an inverter on the output buffer. If the enable line is set to 0 in software, the output buffer is set to enable which is a 1 and the pin is set to output. If the enable line is set to 1, input buffer gets activated and the pin gets configured into input mode. Generally, GPIO can be used in multiple ways. [GPIO Output Mode](https://embetronicx.com/tutorials/tech_devices/understanding-the-microcontroller-gpio-gpio-working-explained/#GPIO_Output_Modes)

1. GPIO Input Mode
2. Analog Mode
3. Alternate function Mode

Figure 5.10 shows the simple implementation of a GPIO pin in a microcontroller.

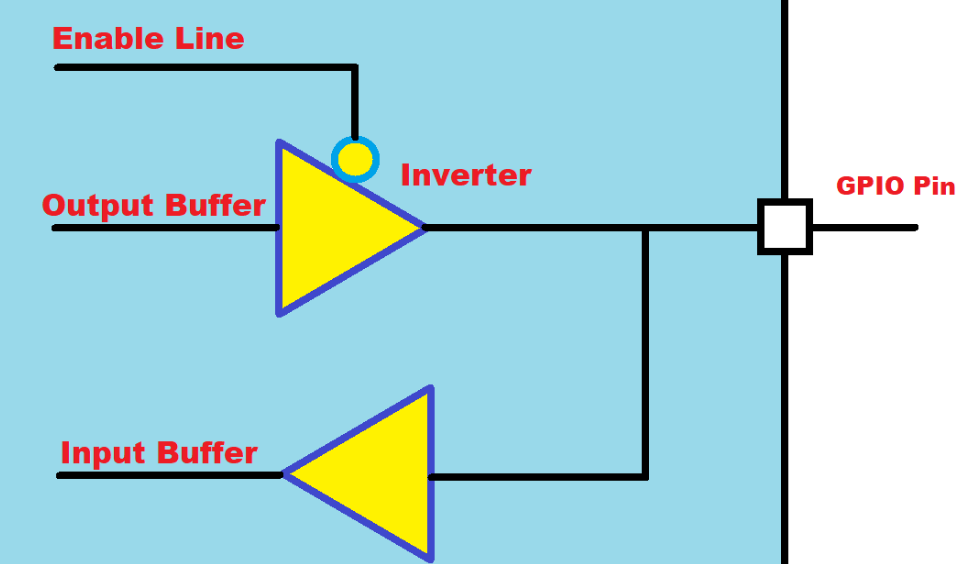


Figure 5. 11:Microcontroller for GPIO

Each GPIO pin consists of two buffers – Input buffer and Output buffer along with one enable line. When the enable line is **0**, the output buffer gets activated and the input buffer will be deactivated. When the enable line is **1**, then the output buffer gets deactivated, and the input buffer gets activated.

**GPIO Input Modes**

The GPIO is used to read the electrical signal in the pin when it is configured as an input. In general, GPIO inputs are primarily configured in one of three ways:

* High-impedance or Floating- keeping the pin floating by not connecting to either **HIGH (Vcc)** or **LOW (GND)** Voltage levels
* Pull-up - The internal pull-up resistor is connected to the pin. So, the state will be **HIGH** unless an external pull-down resistor is used
* Pull-down -The internal pull-down resistor is connected to the pin. So, the state will be Low unless an external pull-up resistor is used.

**GPIO Output Modes**

The GPIO is used to drive the electrical signal (high or low) to the pin when it is configured as an output. There are primarily two configuration options for GPIO outputs:

1. [Push-pull](https://embetronicx.com/tutorials/tech_devices/understanding-the-microcontroller-gpio-gpio-working-explained/#Push-pull)- This state is the default state of the GPIO output mode
2. [Open-drain](https://embetronicx.com/tutorials/tech_devices/understanding-the-microcontroller-gpio-gpio-working-explained/#Open-drain)

**Analog Mode**

You can configure the input as an analogy input. This mode connects the pin to an internal ADC (analogy-to-digital converter) and allows you to read a value that represents a given voltage in a pin.

When a GPIO is configured in analogy mode, the input pull-up/pull-down resistors are disconnected (floating).

### 5.1.17 Read and interpret a Pi GPIO Pin guide

The easiest way to read and interpret the GPIO pin guide is first seeing the structure of the GPIO. By typing the command pinout from the raspberry Pi terminal, you will get the pin labels and also how your raspberry pi is built. The output will look as follows

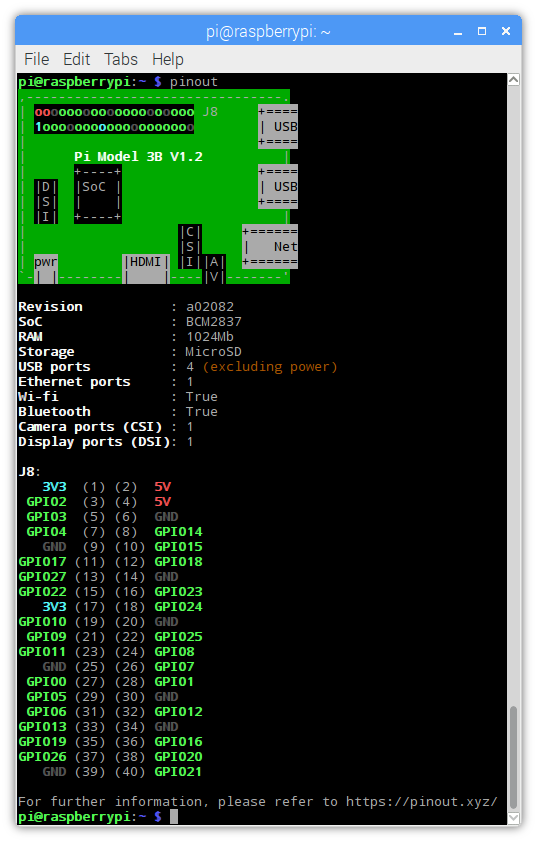


Figure 5. 12: GPIO Layout

### 5.1.18 Explain what the terms 3V3, 5V, GND GP2 means on the GPIO board

Using the Raspberry Pi as the example again, you will find a few types of pin:

* Pins that provide power at typical voltages such as 3.3V or 5V. This is to power connected devices that do not have their own power source, such as a simple LED.

3V3 (3.3 volts) - Anything connected to these pins will always get 3.3V of power.

5V (5volts) - Anything connected to these pins will always get 5V of power.

* Ground pins that do not output power, but are necessary to complete some circuits. GND - Zero volts, are used to complete a circuit.
* GPIO pins, which can be configured to send or receive electrical signals.
* Special purpose pins, which vary based on the specific GPIO in question.

### 5.1.19 Write basic I/O code to represent a solution to a problem

In this section, we are going to demonstrate how to represent a solution to a given problem using Scratch and using Python. Given any problem, programmers start by making sure they understand the problem which needs to be solved. After, they identify a design tool which can represent how they will resolve the problem as part of planning phase. There are quite several design techniques which can be implemented. Some of them includes:

* IPO tables- The IPO table will consist of three columns representing Input, Process and Output.
* Flowchart- A flowchart is a diagram that shows the logic of the program using symbols.
* Algorithms- An algorithm is a step-by-step description of how to arrive at a solution in the easiest way.
* Pseudocodes- It is a methodology that allows the programmer to represent the implementation of a problem using English statements.

We are going to solve the following problem:

Write a program which accepts the radius of a circle during development and compute the area. Remember the area of a circle is computed using this formula, pi \* r 2

We are going to use IPO table as part of the planning phase.

|  |  |  |
| --- | --- | --- |
| **Input** | **Processing** | **Output** |
| Enter radius | pi=3.14  Area= pi x radius x radius | Display Area |

We are going to start by solving the problem using Scratch

**Steps**

Add a when clicked from the event block

Add the ask… wait block from the sensing category

**Input**

Select radius and click make a variable- name its radius.

Add a set…. to and set radius to answer. The answer block is under the sensing block.

**Processing**

Add the set…to. Add two join blocks together. Add the variable radius in the empty spaces and add the value of pi in the third empty space.

**Output**

Add the say…for …. seconds. Insert two join blocks joined together. The first empty block holds text to be displayed “Area of a circle is”. The other block stores the result and lastly the metric (cm2)

Your code should look as the one below.

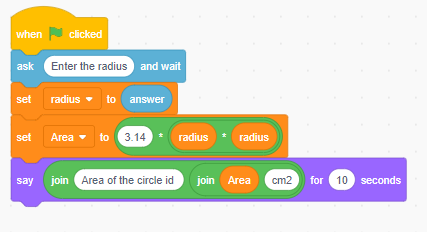


Figure 5. 13:Scratch solution for area of a square

To run the code, click the green flag  just above the stage area. The output will be displayed in the stage are next to the cat default sprite.

Now we are going to use Python to solve the same problem. We will be using Mu IDE.

**Steps**

Open Mu and save the project on to the desktop.

In our program we have a constant pi. A constant is a place in memory location whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later. We can just declare the constant and save the file with constants. Alternatively, we can declare it in our current program. For the purposes of this discussion, we are going to declare Python in the same program.

**Python**

#declaring variables and constants.

radius=int(input("Please enter the radius of the circle"))

pi=3.14

#processing

area\_of\_circle=pi\*radius\*radius

#displaying the area of the circle

print(f"The area of a circle is {area\_of\_circle}")

Please enter the radius of the circle5

The area of a circle is 78.5

>>>

### 5.1.20 Deploy/Run a code script (/block of code/code fragment)

**Run/Execute the source code**

The “Run” button does exactly what you’d expect. It runs the current script. When this happens the textual **input** and **output** of the program is displayed in a panel between the text editor and Mu’s footer.

The same program can be run from the terminal. You will need to navigate to the directory where the python program is saved from the Raspberry Pi terminal. Use the cd <directoryname> to navigate to the folder. cd is short for change directory.

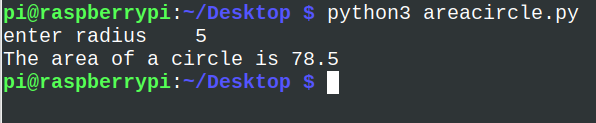


Figure 5. 14:Running Python script from the Raspberry Pi terminal

**Deploy Python Script**

After all the hard work developing a project in Python, we want to share our project with other people. It can be your friends or your colleagues. Maybe they are not interested in your code, but they want to run it and make some real use of it. There are quite a lot of platforms which you can use to deploy your Raspberry Pi scripts developed using Python such as:

* Repl.it
* Docker
* Heroku
* py2exe
* AWS

We are not going to discuss this as it is outside the scope of our syllabus. A link to deploy your project on replit is provided below.

**eLink**

<https://www.youtube.com/embed/WDyru4AAH-M>

### 5.1.21 Debug a code script

An unexpected problem in your program is called a bug. Some are more difficult to fix than others, and they can take many forms. By just reading through your program, you won't be able to catch some bugs. A Python IDE such as Mu or IDLE provides some basic tools that will assist you in debugging your programs.

**Interpreter DEBUG Mode**

Turning on the built-in debugger will allow you to run your code with it. To do so, select *Debug → Debugger* from the Python IDLE menu bar. In the interpreter, you should see [DEBUG ON] appear just before the prompt (>>>), which means the interpreter is ready and waiting.

you can inspect the values of your local and global variables as your code executes. This gives your insight into how your data is being manipulated as your code runs. four checkboxes appear in the debug window, and these are:

* **Globals:** your program’s global information
* **Locals:** your program’s local information during execution
* **Stack:** the functions that run during execution
* **Source:** your file in the IDLE editor

### 5.1.22 Use various programming constructs in the development or modification of a solution

Programs are designed using common building blocks, known as programming constructs. These programming constructs form the basis for all programs. There are three basic programming constructs, and these are:

* sequence
* selection
* iteration

Sequence is the order in which instructions occur and are processed. An example of sequence construct is the one in section 5.1.19 where instructions are executed step by step. In the program, we also made use of input command, used variables and an arithmetic calculation. The program also utilised constant pi to calculate the area of a circle whose input is captured from the keyboard.

Conditionals (Branching)

Another term used for conditionals is selection constructs. Selection determines which path a program takes when it is running. In programming, there are occasions when a decision needs to be made. Selection is the process of making a decision. The result of the decision determines which path the program will take next.

Python makes use of if statements in achieving decisions for program path.

if <expr>:

<statement>

In the form shown above:

* <expr> is an expression evaluated in a Boolean context. The outcome of the expression evaluates to either True or False.
* <statement> is a valid Python statement, which must be indented. (You will see why very soon.)

If <expr> is true, then <statement> is executed. If <expr> is false, then <statement> is skipped over and not executed.

Sometimes, you want to evaluate a condition and take one path if it is true but specify an alternative path if it is not. This is accomplished with an else clause:

Syntax

if <expr>:

<statement(s)>

elif <expr>:

<statement(s)>

elif <expr>:

<statement(s)>

...

else:

<statement(s)>

#Accepting mark

mark=int(input("Enter the average mark \t"))

#checking for condition

if (mark>89) and (mark <101):

print("You got an A grade")

elif (mark>79) and (mark <90):

print("You got an B grade")

elif (mark>59) and (mark <80):

print("You got an C grade")

elif (mark>49) and (mark <60):

print("You got an D grade")

elif (mark>39) and (mark <50):

print("You got an E grade")

elif (mark>=0) and (mark <40):

print("You got an F grade")

else:

print("Mark is outside the range")

Output

Enter the average mark 45

You got an E grade

>>>

**Iteration**

Iteration is often referred to as looping, since the program ‘loops’ back to an earlier line of code. Iteration is also known as repetition. Iteration allows programmers to simplify a program and make it more efficient. Instead of writing out the same lines of code again and again, a programmer can write a section of code once, and ask the program to execute the same line repeatedly until no longer needed. When a program needs to iterate a set number of times, this is known as definite iteration and makes use of a FOR loop.

Syntax

for i = 1 to 10

<loop body>

Example

Write a Python loop to print numbers 1 to 10.

for i in range(1,11):

print(i)

Output

1

2

3

4

5

6

7

8

9

10

>>>

Post-test loops

A post-test loop can be implemented using a do ….while loop. To create a do while loop in Python, you need to modify the while loop a bit in order to get similar behaviour to a do while loop in other languages. A do while loop will run at least once even if the condition is not met. If the condition is met, then it will run again.

Example

i =9

while True:

print(i)

i = i + 1

if(i > 5):

break

Output

9

Pre-test loops

This can be implemented using a while loop. The condition is checked first before executing any statements. Syntax

While test\_expression:

Body of while

Example

Write a do loop to print numbers from 0 to 10

i=0

while (i<10):

print(i)

i=i+1

Output

0

1

2

3

4

5

6

7

8

9

10

>>>

If for any reason, we initialise our i to a number greater than 10, nothing will ever be printed

Challenge

Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).

mylist=[]

#define the loop

for x in range(1500, 2701):

#add a condition

if (x%7==0) and (x%5==0):

#add each value meeting the condition to the list

mylist.append(str(x))

#displaying output

print (','.join(mylist))

* Instantiate applicable objects to reference components and basic sensors
* Invoke a subroutine or method (Calls)

## 5.2 GPIO programming, practical projects and physical computing

Content

* GPIO Programming projects
* Component input and output
* gpiozero
* Importing classes/libraries

*Learning Outcomes:*

*Students should be able to:*

5.2.1 Design simple prototypes using components connected directly to the GPIO pins

**Control LED Using Raspberry Pi GPIO**

In this project, we are going to use Scratch programming to connect an LED and configure it to blink. This is a basic project to connect an LED to RPi GPIO and control using a Scratch.

**Step 1: Components**

You will need the following components to connect the circuit.

* Raspberry Pi
* LED
* Resistor - 330 ohm
* Breadboard
* 2 Male-Female Jumper Wires

**Step 2: Connecting the Circuit**

Please keep the Raspberry Pi turned off until the circuit is connected to avoid accidentally shorting any components. The LED has 2 legs. The longer leg, 'anode', is always connected to positive supply. The shorter leg, 'cathode', is always connected to ground. You need a 330-Ohm resistor is the circuit to limit the amount of current in the circuit. Without the resistor the current flowing through the LED will be much larger and lead to a short damaging the circuit.

1. Use a jumper wire to connect the ground ( Pin 3) of GPIO to rail marked in blue on the breadboard.
2. Connect the resistor from the same row on the breadboard to a column on the breadboard.
3. Connect the LED with the cathode in the same row as the resistor. Insert the anode in the adjacent row.
4. Use another jumper cable to connect the GPIO Pin 21 ( 3.3 V) in the same row as the anode of LED.

**Scratch Code to control the Raspberry Pi**

Start Scratch and add the Raspberry Pi GPIO extension by clicking on the bottom lower corner button on the Scratch window. Immediately you will see three Raspberry Pi blocks added.

Add a green flag from the Events block.

Add the repeat block from the control blocks. Set it to 10 seconds

Add the set gpio … to output high block. Set the gpio to 16 (Pin 36).

Add the Wait for 1 seconds block

Add the set gpio … to output high block. Set the GPIO to 16. Instead, set the output to low.

Add the Wait for 1 seconds block

Click the flag and you will see your LED blinking

Congratulations. We have set out the first project where hardware is interacting with software.

If you decide to use Python, the connection on the breadboard and GPIO remains the same.

The code on the Mu editor would be as follows.

#importing the gpiozero and time library

from gpiozero import LED

from time import sleep

green = LED(16)

#Turning On/Off the LED using

green.on()

sleep(1)

green.off()

sleep(1)

When you run the above code, the light will only turn on once and switch off. This is simply because the code is sequential execution. We will discuss about repetition later in this module.

5.2.2 Incorporate the use of input components and control output components using a single board computer

Now we want to accept input from the Raspberry Pi and let our the pi to respond accordingly.

**Challenge**

We are going to modify the grading program in section 5.1.22 to include physical computing. In this exercise, we want to do the following:

Turn the green LED light on as long as the student got a B grade or higher, turn the orange light when a student gets a C or D grade, turn red if the pass mark is between 0 and 49 otherwise turn all three lights on if the mark is outside the range.

You will need:

* A breadboard
* 3 LEDs (red, green and orange)
* 6 female-to-male jumper cables
* 3x 330 Ohm resistors

Connect the cathode to the the following pins on the Raspberry Pi.

* Green to pin (22)
* Orange to pin (27)
* Red to pin (17)

The Python Program will look as below:

from gpiozero import LED

from time import sleep

green=LED(22)

orange=LED(27)

red=LED(17)

try:

#Accepting mark

mark=int(input("Enter the average mark \t"))

#checking for condition

if (mark>89) and (mark <101):

green.on()

print("You got an A grade")

sleep(5)

green.off()

elif (mark>79) and (mark <90):

green.on()

print("You got an B grade")

sleep(5)

green.off()

elif (mark>59) and (mark <80):

orange.on()

print("You got an C grade")

sleep(5)

orange.off()

elif (mark>49) and (mark <60):

orange.on()

print("You got an D grade")

sleep(5)

orange.off()

elif (mark>39) and (mark <50):

red.on()

print("You got an E grade")

sleep(5)

orange.off()

elif (mark>=0) and (mark <40):

red.on()

print("You got an F grade")

sleep(5)

red.off()

else:

red.on()

green.on()

orange.on()

print("Mark is outside the range")

sleep(5)

red.off()

green.off()

orange.off()

5.2.3 Incorporate and use different polarized components as part of the design (Range: Batteries, LED’s. Diodes, Relays and Capacitors)

5.2.4 Build simple prototypes using components connected directly to the GPIO pins

5.2.5 Plan design and code simple programs to read data from analogue and digital inputs and control analogue and digital outputs

* + 1. Use basic programming constructs to create simple programs
    2. Initiate and setup Python program that imports gpiozero
    3. Write Python code to support reading from and control digital and analogue inputs using
       1. pause from the signal module
       2. sleep from the time module
       3. uniform from the random module
       4. Selection structures
       5. Repetition structures

So let us implement loops with physical computing.

We want to turn an LED using a while loop.

Connect the LED and resistor on to the breadboard. Use pin 18 (GPIO 24) for the positive connection. The anode can be connected to any ground pin on the raspberry pi. Import 2 libraries: gpiozero and LED to allow accessing the gpio pins and LED for the lights.

Here is the code

from gpiozero import LED

from time import sleep

led=LED(24)

while True:

led.on()

sleep(2)

led.off()

sleep(5)

gpiozero.cleanup()

Please take note on the above code, we implemented the cleanup( ) method so that whenever you press control +c on the keyboard an interrupt is activated and stops the LED.

**Challenge**

Use a for loop to allow the LED to blink five times only.

Solution

from gpiozero import LED

from time import sleep

led=LED(24)

for x in range(0,5):

led.on()

sleep(2)

led.off()

sleep(2)

gpiozero.cleanup()

* + 1. Python code to read from / control digital and analogue inputs using gpiozero classes

#### 5.2.9.1 LED

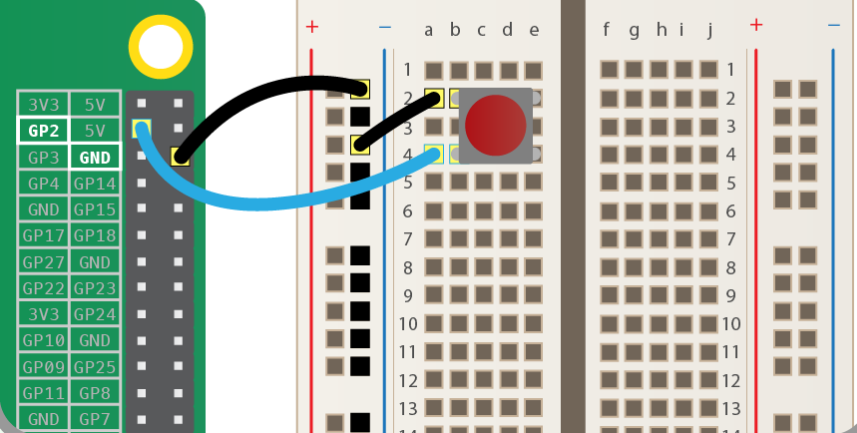
LEDs are delicate little things. If you put too much current through them, they will pop (sometimes quite spectacularly). To limit the current going through the LED, you should always use a resistor in series with it. Section 5.2.1 outlined how you can connect an LED light to the Raspberry Pi and turn the LED on and off.

#### 5.2.9.2 Button

As well as controlling the physical world, you can react to it using an input device such as a button. Connect your button to a breadboard, and then connect one pin to a ground pin and the other to a numbered GPIO pin. In this example pin 2 has been used.

* Create a new file by clicking New.
* Save the new file by clicking Save. Save the file as gpio\_button.py.
* This time you’ll need the Button class, and to tell it that the button is on pin 2. Write the following code in your new file:

The connection should look as shown on the diagram below.



Python code

from gpiozero import Button

button = Button(2)

button.wait\_for\_press()

print('You pressed the button”)

Output

You pressed the button

* + - 1. RGBLED

An RGB LED contains four pins: one for each colour (Red, Green, and Blue) and one for the common cathode. It features three separate colour-emitting diodes that may be combined to produce a wide range of colours! Depending on how light each diode is, any colour is possible. This guide will teach you how to utilize an RGB LED with a Raspberry Pi to generate unique colour combinations.

Step 1: What You Will Need

For this tutorial you will need:

* [GPIO Breakout(optional)](https://grobotronics.com/t-cobbler-plus-gpio-breakout-for-raspberry-pi-3-2-b.html)
* [Breadboard](https://grobotronics.com/breadboard-1360-tie-point-medium.html)
* [RGB LED](https://grobotronics.com/led-clear-5mm-rgb-common-anode.html)
* [3x220 Ohm resistors](https://grobotronics.com/resistor-1-2w-carbon-5-220ohm.html)
* Stripline pin or [some jumper cables](http://amzn.to/2ekI7r3) for Raspberry Pi



Connections

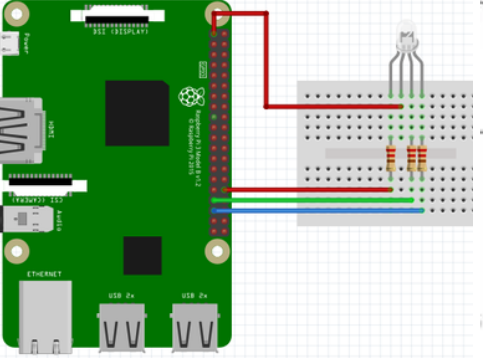
Plug the circuit on GPiO in this position. We connected the VCC (Cathode) to pin number 1 on the breadboard.

Red Pin number 32 (GPIO12)

Green -Pin number 33 (GPIO13)

Blue – Pin number 35 (GPIO19)

Each of the three pins (RGB) have a resistor on the breadboard as shown on the diagram.



Pay attention of the numbers of pin. In this project we use a GPIO pin number. Please double-check before start your system. You can burn Raspberry Pi.

The combination of the three colours is made to produce one colour and they are each either set to low or high. For example, to produce the white colour, the GPIO pins are set to output mode and the following combinations are applied:

* GPIO.output(redPin,GPIO.LOW)
* GPIO.output(greenPin,GPIO.LOW)
* GPIO.output(bluePin,GPIO.LOW)

Run the program and you will notice different colours (more than 3 vc m o

Solution in Python

#libraries

import RPi.GPIO as GPIO

from time import sleep

#disable warnings (optional)

GPIO.setwarnings(False)

#Select GPIO Mode

GPIO.setmode(GPIO.BCM)

#set red,green and blue pins

redPin = 12

greenPin = 19

bluePin = 13

#set pins as outputs

GPIO.setup(redPin,GPIO.OUT)

GPIO.setup(greenPin,GPIO.OUT)

GPIO.setup(bluePin,GPIO.OUT)

def turnOff():

GPIO.output(redPin,GPIO.HIGH)

GPIO.output(greenPin,GPIO.HIGH)

GPIO.output(bluePin,GPIO.HIGH)

def white():

GPIO.output(redPin,GPIO.LOW)

GPIO.output(greenPin,GPIO.LOW)

GPIO.output(bluePin,GPIO.LOW)

def red():

GPIO.output(redPin,GPIO.LOW)

GPIO.output(greenPin,GPIO.HIGH)

GPIO.output(bluePin,GPIO.HIGH)

def green():

GPIO.output(redPin,GPIO.HIGH)

GPIO.output(greenPin,GPIO.LOW)

GPIO.output(bluePin,GPIO.HIGH)

def blue():

GPIO.output(redPin,GPIO.HIGH)

GPIO.output(greenPin,GPIO.HIGH)

GPIO.output(bluePin,GPIO.LOW)

def yellow():

GPIO.output(redPin,GPIO.LOW)

GPIO.output(greenPin,GPIO.LOW)

GPIO.output(bluePin,GPIO.HIGH)

def purple():

GPIO.output(redPin,GPIO.LOW)

GPIO.output(greenPin,GPIO.HIGH)

GPIO.output(bluePin,GPIO.LOW)

def lightBlue():

GPIO.output(redPin,GPIO.HIGH)

GPIO.output(greenPin,GPIO.LOW)

GPIO.output(bluePin,GPIO.LOW)

while True:

turnOff()

sleep(1) #1second

white()

sleep(1)

red()

sleep(1)

green()

sleep(1)

blue()

sleep(1)

yellow()

sleep(1)

purple()

sleep(1)

lightBlue()

sleep(1)

MotionSensor

Motion Sensor

In this circuit, you will be connecting a **passive infrared (PIR) motion sensor** to the Raspberry Pi. A passive infrared motion sensor detects any motion in its field of vision and sends a signal back to the Raspberry Pi. The motion sensor consists of three pins:

1. VCC for voltage
2. OUT for communicating with the Raspberry Pi
3. GND for ground

Adjusting the Sensor

When using a motion sensor, you may need to adjust how sensitive it is to motion and how long it will send out a signal after motion is detected. You can make adjustments using two dials on the side of the sensor. See the image of the motion sensor shown below.

**Connecting the Motion Sensor**

We are also going to allow an LED light to turn on if a motion has been detected and turn off if no motion has been detected. So, we will stick to the earlier connection for our LED.

Using these pins, you need to take the following steps:

1. Connect a **female-to-female** jumper wire from the Raspberry Pi’s **5V** pin to the sensor’s **VCC** pin. (Pin 2)
2. Connect a **female-to-female** jumper wire from the Raspberry Pi’s **GPIO4** pin (Pin 7) to the sensor’s **OUT** pin.
3. Connect a **female-to-female** jumper wire from the Raspberry Pi’s **GND** pin to the sensor’s **GND** pin.(Pin 6)

Our connection for the sensor and the GPIO header will be as shown on the figure below.

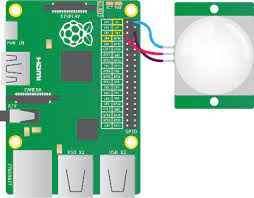


Figure 5. 15: Motion Sensor

With the motion sensor adjusted and wired to the Raspberry PI, let us look at the Python code for detecting motion.

from gpiozero import LED

from time import sleep

from gpiozero import MotionSensor

pir=MotionSensor(4)

green=LED(16)

green.off()

while True:

pir.wait\_for\_motion()

print(Motion detected”)

green.on()

pir.wait\_for\_no\_motion()

green.off()

print(“Motion stopped”)

**NOTE**

Make sure to move the sensor as far from the Raspberry Pi as possible to avoid occasional false positives.

* + - 1. Buzzer

There are two main types of buzzer: active and passive. A passive buzzer emits a tone when a voltage is applied across it. It also requires a specific signal to generate a variety of tones. In this module, we will cover an active buzzer. An *active* buzzer can be connected just like an LED, but as they are a little more robust, you won’t be needing a resistor to protect them.

The diagram below illustrates the simple connection you will need to do for an active buzzer.

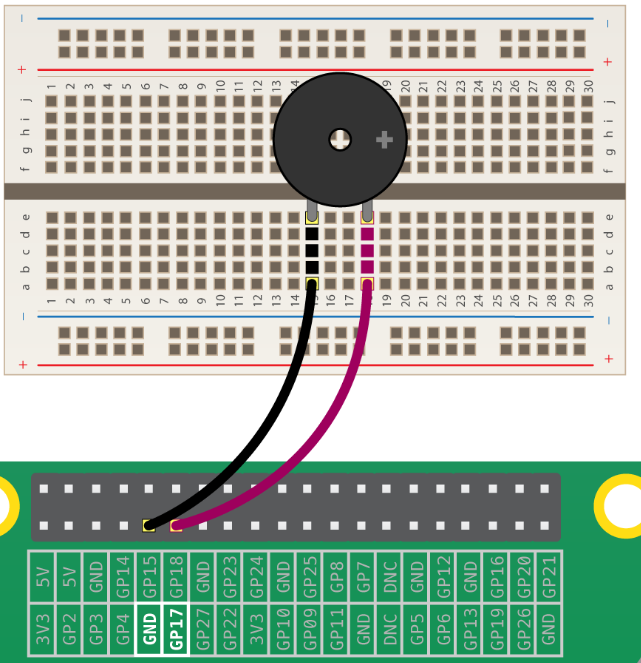


Figure 5. 16: Connecting a buzzer

**Python Code**

from gpiozero import Buzzer

from time import sleep

buzzer=Buzzer(17)

while True:

buzzer.on()

sleep(1)

buzzer.off()

sleep(1)

When the program is run, the buzzer begins to buzz: one second on, and one second off. If we had used a passive buzzer, you would hear a brief click every second instead of a continuous buzz.

* + - 1. LightSensor

A light sensor is a kind of photodiode. These components convert light into energy. The output from a photodiode can be measured and used to trigger specific functions.

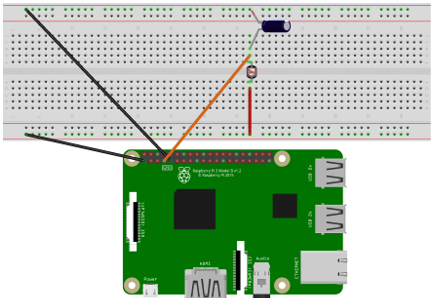
In basic terms, these are diodes that detect light and give an output based on the level of light they receive. We can use the output in custom programs and projects. Light sensors can be used to trigger night lights or even help you monitor the light levels within a given space.

We are going to calculate how long it takes to charge a capacitor in a circuit containing a variable resistance resistor. The resistor being used is a photoresistor, so depending on how much light is present, the resistance changes. When light is present, the resistance is lower, making it faster to charge the capacitor. When light is not present, the resistance is higher, making it slower to charge the capacitor. We use the Raspberry Pi to constantly read voltage values to determine how long it takes to charge the capacitor. Since the Pi does not have any analog pins, we can not read the resistance from the photoresistor directly without a Analog-to-Digital converter.

**What you need**

* Photoresistors
* Breadboards
* Breadboard Jumper Wires
* Capacitors
* Red Dot Laser Head/Diode

Here is the diagram which shows the connection which we are emulating



Connections

* Connect the capacitor to the resistor through the breadboard. The positive leg of the positive leg of the capacitor to the photo
* Connect the negative leg of the capacitor to ground
* connect the 3.3 volt pin as well as the ground pin on the Raspberry Pi to the breadboard
* the positive leg of the capacitor connect to one of the PI's GPIO pins using a jumper
* the leg of the photo resistor wire
* connect the other leg of the photo resistor to power with another jumper wire

**Python Code**

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BOARD)

resistorPin = 7

while True:

GPIO.setup(resistorPin, GPIO.OUT)

GPIO.output(resistorPin, GPIO.LOW)

time.sleep(0.1)

GPIO.setup(resistorPin, GPIO.IN)

currentTime = time.time()

diff = 0

while(GPIO.input(resistorPin) == GPIO.LOW):

diff = time.time() - currentTime

print(diff \* 1000)

time.sleep(1)

**Run the program**

Our program will show the difference in times it takes the capacitor to charge between when the light intensity is high when the light intensity is low by using the laser diode.

* + - 1. MCP3008 (Analogue to digital converter)
      2. LEDBarGraph

|  |  |
| --- | --- |
| 👀 | PLEASE see:  <https://gpiozero.readthedocs.io/en/v1.2.0/recipes.html> |

Design, develop and test a project using the GPIO

COMPULSORY PROJECTS

|  |  |
| --- | --- |
| ***Range*** | ***Pedagogical purpose*** |
| Programmatically switching an LED on and off | Basic commands, circuits and the importance of resistance |
| Traffic light simulation | Introduce iteration constructs  Introduce the invocation of built-in commands, routines and methods |
| Reaction time button. | Input signal processing   * Capture the lapsed time between two button presses * Switch an LED on and off with a button * Add a green and red LED and two buttons (the button who is pressed first lights up the corresponding LED) * Basic events |
| Day night switch with light sensor | Signal processing. Incorporate a light sensor to switch on or switch off a circuit based on light intensity.  Conditional logic / Circuit design  Introducing Capacitors / and LDR (light dependant resistor) sensors. |
| LED thermometer | Temperature sensor  Display the temperature indication using LED lights where green represents cold, yellow represents moderate and red, hot temperatures based on user parameters and sensor input. Converting analogue to digital |

5.2.2 Design and develop a basic user defined GPIO project utilising the constructs and components presented in 4.2.4 and 5.1.17. This also in conjunction with 6.1

|  |
| --- |
| For LO 5.2.1 and 5.2.2, each of the various projects and examples presented should be done in such a manner that the student presents an understanding of:   * The basic design principles followed in the design * The composition and use of the various electronic components * Use of the BASIC sensors (Range See 4.2.12) |

References

Lobontiu, N., 2017. System dynamics for engineering students: Concepts and applications.