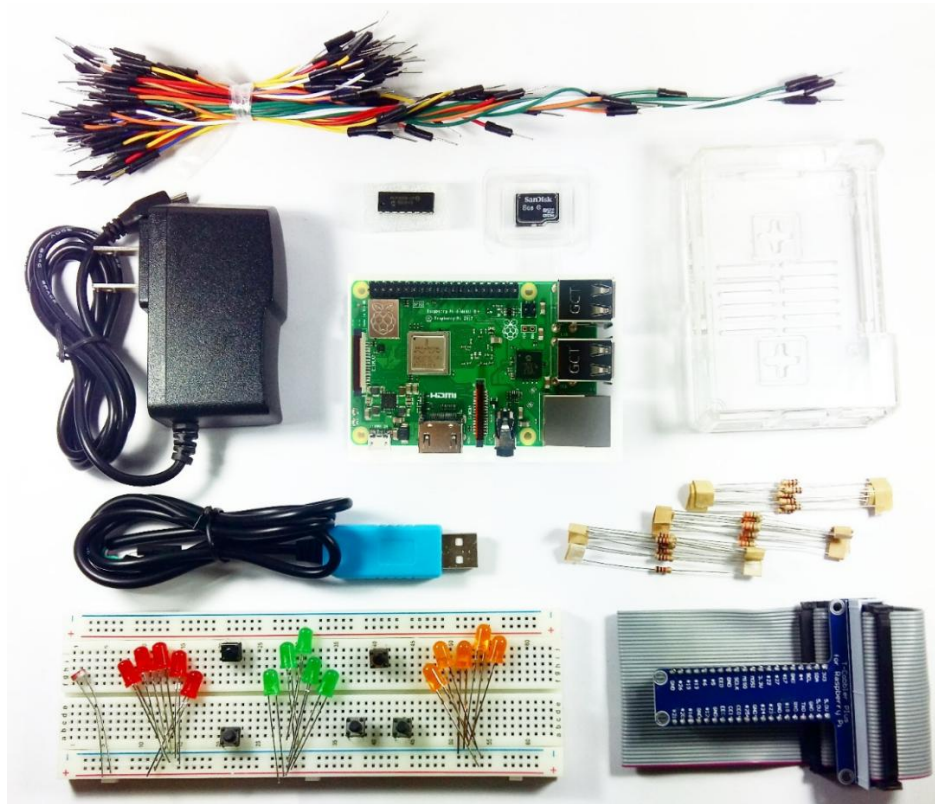


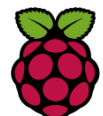
## Raspberry Pi Experiment Guide Part 2 - (GPIO) Example Inputs and Outputs



The Raspberry Pi is like an all-in-one small computer plus a microcontroller. You can use it as a hardware prototyping tool to connect electronic components and module boards, write some code and invent cool electronics!

If you came from the Arduino and other microcontroller platform, you are mostly familiar with the C programming and Windows Desktop computer environment. When working with the Raspberry Pi, we will also be able to run C program codes and the circuits we worked with Arduino.

There are a few programming languages to choose when programming the GPIO pins of the Pi but the really useful and easy to use is the **Python and C Programming** languages. We will also be getting more familiar with the **Linux Computer environment** thru the examples we will try out in the Examples section.



Be sure to check out Part 1 of this Experiment Guide to setup your Raspberry Pi computer.

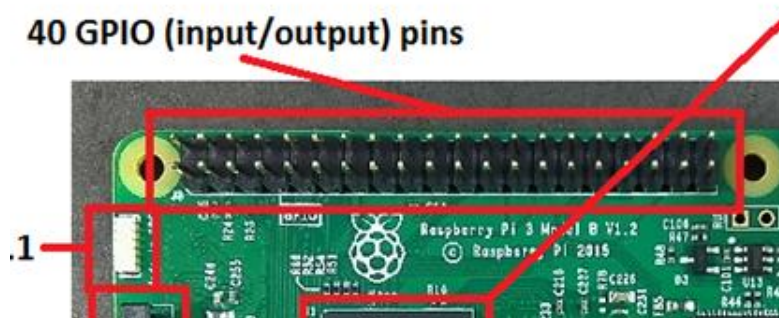
## 01 Tutorial Contents

In this Part 2 of the Experiment Guide we will be connecting electronics to the Pi's GPIO pins such as buttons, LED's and sensors. It is important to understand first how the Pi processes simple GPIO programming before we can explore to a more complex project. We compiled this experiment to learn the ins and outs properly. We will program and run codes on the Pi to interact with the electronics using **Python** and **C** programs. For this guide here's what we will explore.

- Raspberry Pi Hardware GPIO pins Overview
- Overview of the GPIO Pin extension Adapter
- Preparing the Hardware parts
- Prepare the Software and install the needed libraries (Python libraries and C libraries)
- Perform experiment circuits and run codes in Python and C program

## 02 GPIO Pins Overview

The Raspberry Pi has 40 GPIO pins which means we can connect lots of external hardware to our Pi. The 40 pins are composed of the Power pins, GPIO (General Purpose Input Output) where we can connect LED/buttons/sensors, UART Communication pins (Tx/RX), i2c interface pins (SDA, SCL), and also the SPI interface pins. That means we can interface lots of electronic components, sensor board and other modules.



Let's take a look at the Pi's each GPIO pins and know each of its uses and functionality so it will be easier for us to connect our electronic components. Below is a picture of the individual pins found in the Raspberry Pi's boards GPIO 40-pin header and its individual pin descriptions.



Function	WiringPi	GPIO. BCM and Scratch	GPIO. BOARD	GPIO. BOARD	GPIO. BCM and Scratch	WiringPi	Function	
3.3 VDC			1		2		5 VDC	
SDA1	8	2	3		4		5 VDC	
SCL1	9	3	5		6		GND	
GPCLK0	7	4	7		8	14	TxD	
GND			9		10	15	RxD	
GPIO	0	17	11		12	18	1	PCM_CLK PWM0
GPIO	2	27	13		14			GND
GPIO	3	22	15		16	23	4	GPIO
3.3 VDC			17		18	24	5	GPIO
MOSI	12	10	19		20			GND
MISO	13	9	21		22	25	6	GPIO
SCLK	14	11	23		24	8	10	CE0
GND			25		26	7	11	CE1
SDA0			27		28			SCL0
GPCLK1	21	05	29		30			GND
GPCLK2	22	06	31		32	12	26	PWM0
PWM1	23	13	33		34			GND
PCM_FS PWM2		19	35		36	16	27	GPIO
GPIO	25	26	37		38	20	28	PCM_DIN

## Pin Numbering System

The image on the right shows the 3 types of pin numbering system when using the Pi's GPIO pin. Although with different numbering system they function the same as described in the **Function** column. Let's explained below.

**GPIO.BOARD** – Raspberry Pi pin numbering sequence based on the 2 rows of 20 male pins on the Pi board. The numbering starts from top left then counting left to right. Thus, top left pin is number 1, next to the right is number 2 and the last pin at the lower right is number 40.

**GPIO.BCM** - Pin numbering system matching the 40 pins on the Pi board to where these pins are connected to main processor IC chip (BCM2837). BCM referred as Broadcom (the IC chip).

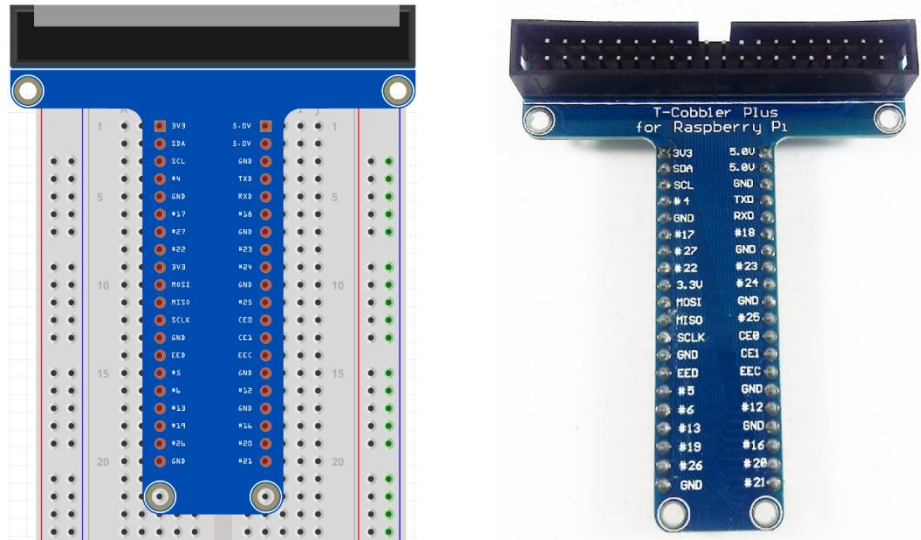
**WiringPi** - This numbering system could be used when we program the Pi with C programming language using the wiringPi library.

The selected numbering system we will use here is the **GPIO.BCM** for all the experiments since it is more specific to what pins on the main processor chip we are connecting. If you are adventurous you can try the other system also after we run example from the GPIO.BCM configuration.

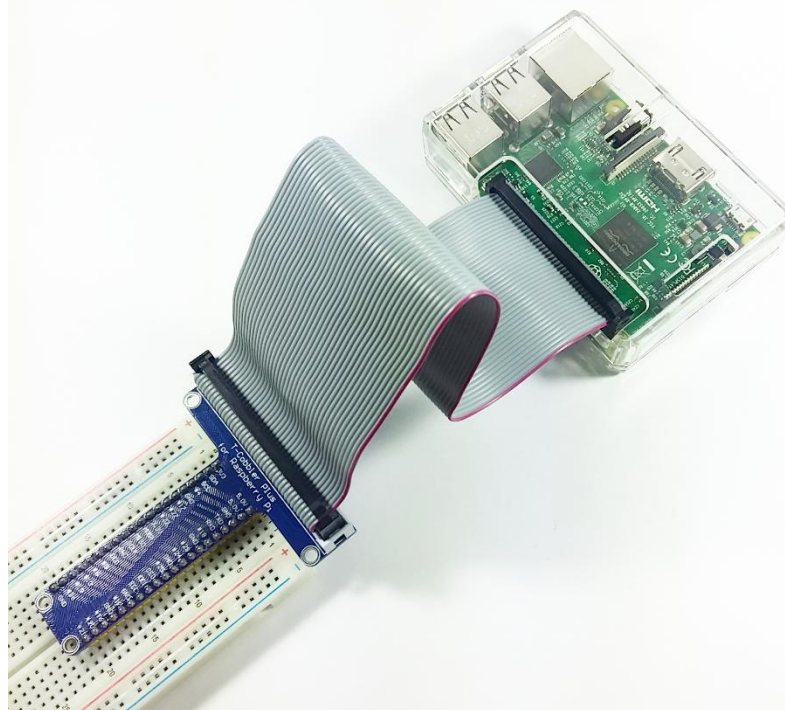
In the next part we will introduce an extension adapter where it becomes much easier to use our numbering system.

## 03 Raspberry Pi GPIO Pins Extension Adapter

In this kit we included a useful tool to even have an easier way to connect an electronics hardware to you Pi, the **Raspberry Pi GPIO Extension/Adapter - T-Cobbler (includes 40-pin ribbon cable)**. This Adapter gives a better visual of the Pi's pins plus it helps not damaging our Pi computer. This Adapter has printouts on the PCB of all the individual pin labels from the Pi GPIO. This pins labels on this GPIO Extension Adapter are based on the **GPIO.BCM pin** configuration which are discussed from the previous notes.



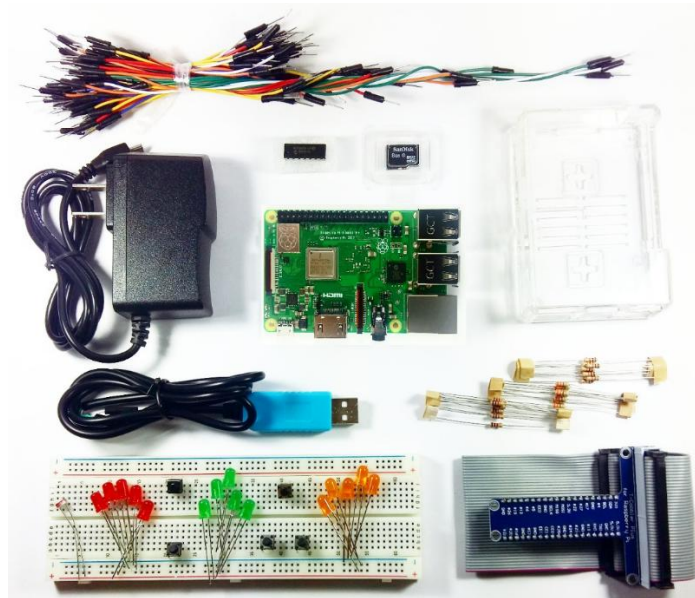
Insert the end cable of the Raspberry Pi T-Cobbler Adapter to the GPIO headers of the Pi following the same orientation below. The cable with red colored line will be in the corner part. Then connect the Extension Adapter board to the breadboard. For easier wiring place the first pins (5V and 3.3V) of the Adapter board to the first rows of the breadboard.





## 04 Prepare the Hardware parts

Prepare the hardware parts listed below for this experiment. All of these parts are already available if you got the Raspberry Pi 3 Model B+ Starter Kit.



- Raspberry Pi 3 Model B+ (B plus)
- Breadboard solderless - Full size 830points
- Raspberry Pi GPIO Adapter - T Cobbler (with ribbon cable)
- Connecting Wires - Male-Male, 65pcs (assorted-size)
- MicroSD Card with Raspberry Pi OS (8GB, class10)
- LED - Red (5mm)
- LED - Green (5mm)
- LED - Yellow (5mm)
- Push Button tactile switch (SPDT)
- Resistors (330 ohms 10pcs)
- Resistors (10k ohms 10pcs)
- Resistors (1k ohms 10pcs)
- LDR (Light Dependent Resistor)
- Power Supply Wall Adapter - 5V/2A (2000mA) (micro-USB)
- Raspberry Pi Case (Clear)
- MCP3008 - 8-Channel 10-Bit ADC With SPI Interface
- USB to RS232 Serial TTL Converter Cable (PL2303)

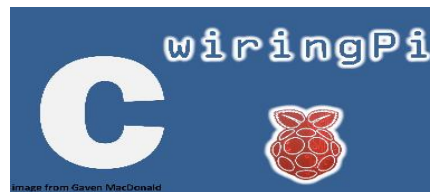
## 05 Download “git” software

We will download some software and library packages. We do this by using a software called “git”. Git serves as a software for creating, saving or getting some codes in a database on the internet. Install git software by typing the commands below. You may need an internet connection to install this.

```
pi@raspberrypi: ~ cd
pi@raspberrypi: ~ sudo apt-get install git-core
```

You will be prompted that you have successfully installed the git software.

## 06 Programming the Pi



The 2 most accessible and compatible programming languages to use for the Pi are:

- 1) Python Programming Language - using **RPi.GPIO library**
- 2) C Programming Language - using **wiringPi library**

In this experiment we will be trying out examples using the 2 programming languages to do tasks with the electronics that we will connect to the Pi.

**Python** (RPi.GPIO library) is already included in the Raspbian OS package installed on the SD card. It is a powerful scripting language. You may discover that with Python some tasks we do in C or other languages can be done in fewer code lines.

**C** (wiringPi library) is a software library written to allow C programs to be compiled and run on the Pi. Later we will see that wiringPi is written almost the same with Arduino code format. This package is not installed in any Raspberry Pi OS. You will need the Pi to be connected to the internet in order to download this software. To install, type the commands below.

```
pi@raspberrypi: ~ cd
pi@raspberrypi: ~ sudo git clone git://git.drogon.net/wiringPi
pi@raspberrypi: ~ cd wiringPi
pi@raspberrypi: ~ git pull origin
pi@raspberrypi: ~ cd wiringPi
pi@raspberrypi: ~ ./build
```

Some commands may take a moment to process since it will be downloading and installing.

## Python software version

If you like to know the version of the Python you are using you can type the command below (it has two “dash (-)” before the word “version”). A python code ends with a “.py” file.

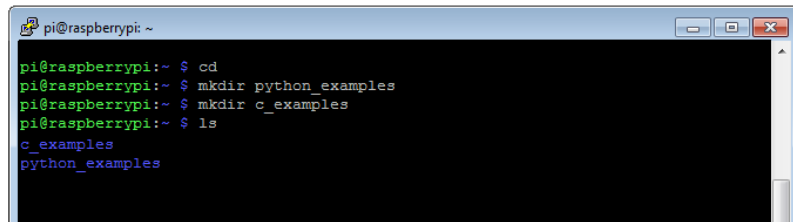
```
pi@raspberrypi: ~ python --version
```

## Create directory folders to store Python and C codes

Let us create a directory folder to save all our Python and C codes so it will be easier to locate the next time we use this codes. To create a folder we will use the command **mkdir**. Type the commands below and press Enter key after every line.

```
pi@raspberrypi: ~ cd
pi@raspberrypi: ~ mkdir python_examples
pi@raspberrypi: ~ mkdir c_examples
pi@raspberrypi: ~ ls
```

The first line directs as to the home directory since we will put the folder in our home directory to easily locate our files. The second and third line creates a folder with its name. The fourth line tells the system to list all the files and folders so we can check that we have successfully created the directory folders.

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The terminal shows the following commands and their outputs: 'cd' (no output), 'mkdir python\_examples' (no output), 'mkdir c\_examples' (no output), and 'ls' (output: 'c\_examples' and 'python\_examples'). The terminal has a black background with green and blue text. The window has standard Linux window controls (minimize, maximize, close) in the top right corner.

```
pi@raspberrypi:~ $ cd
pi@raspberrypi:~ $ mkdir python_examples
pi@raspberrypi:~ $ mkdir c_examples
pi@raspberrypi:~ $ ls
c_examples
python_examples
```

Now that we have created folders to save our python and C codes let's start forming a circuit and coding. Let's go ahead and test some projects!

## Example 1: Digital Output - Blink LED

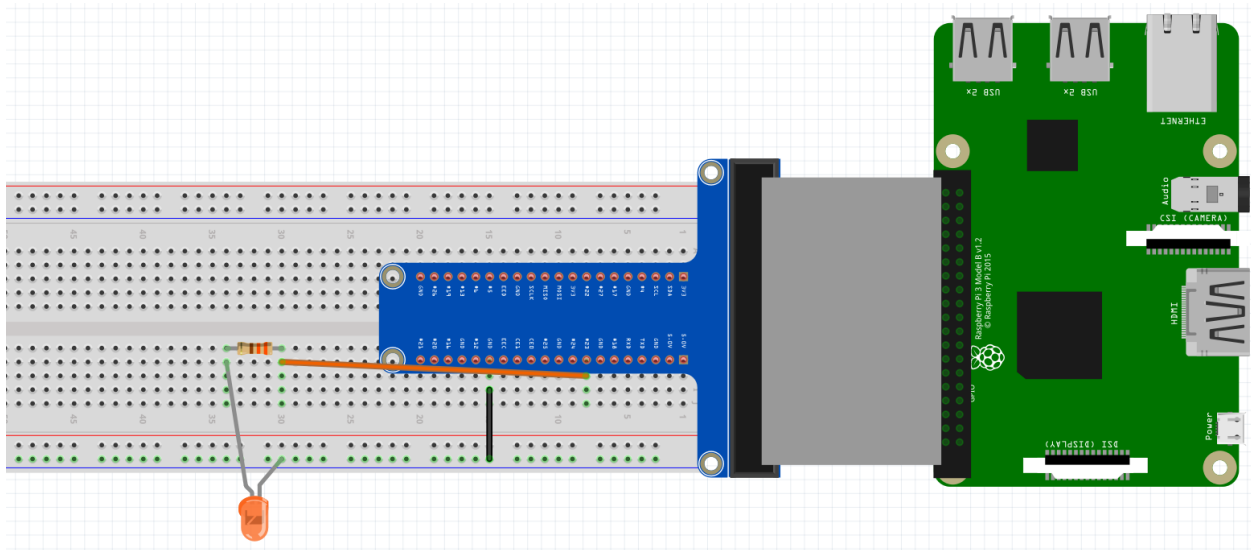


Fig. 1A

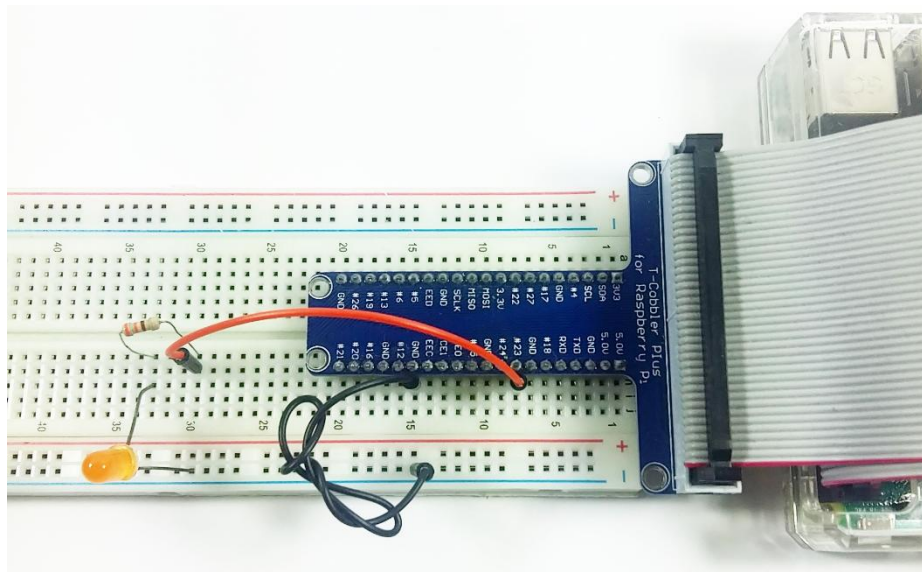


Fig. 1B

Our first experiment is to connect a button (Digital Input) to one of the pins of our Raspberry Pi.

### Parts List:

- \*Raspberry Pi 3 Model B
- \*Raspberry Pi GPIO Adapter - T-Cobbler (with ribbon cable)
- \*Breadboard - full size 830-points
- \*LED (Red)
- \*Resistor - 330 ohms
- \*Connecting Wires



Assemble the circuit in Figure 1A. Your actual circuit would look like in Fig. 1B. You may zoom in your view to see the wiring connections clearly. In this setup we connect an LED to **PIN#23** of the Raspberry Pi GPIO pin as extended by the GPIO Adapter board. **PIN#23** is connected to one end of the 330 ohm resistor and the other end of this resistor is connected to the long leg of the Orange LED. We connect a wire from the Adapter board to the common Ground (GND) of the breadboard and then connect the short leg of the LED to this Ground.

**Note:** As explained earlier for the Pin Numbering System we will be using the **GPIO.BCM** configuration which means that what pin number we use in our code will base on the numbers written in the GPIO Extension Adapter.

## Python code (using RPi.GPIO library)

Let us now create a python script to blink the LED. We will store the python file in the directory folder created earlier. First go to the created directly folder by issuing the commands below.

```
pi@raspberrypi: ~ cd
pi@raspberrypi: ~ cd python_examples
```

Now we are in the directory folder **python\_examples**. Issue the commands below to create a blank text file and name it **Blink\_GPIO\_BCM.py**

```
pi@raspberrypi: ~ sudo nano Blink_GPIO_BCM.py
```

You will then be directed to a blank text editor called “**nano**” text editor. Here, type the script found on the Sample Python code below.

*Remember that there are “white” spaces in the editor so be sure to type in correctly every letter, lines with space and tabs. You can only use your keyboard arrow keys to navigate to the blank space when you are in the nano text editor.*

## Sample Python code

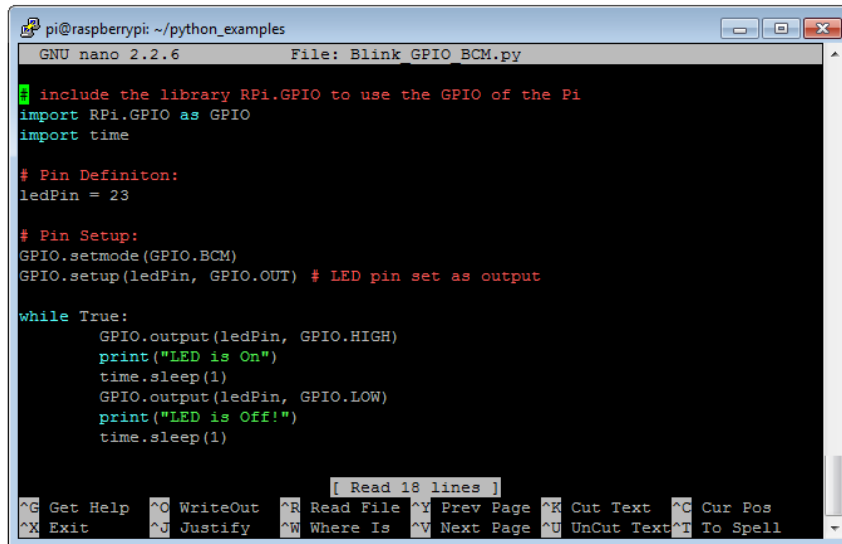
```
# include the library RPi.GPIO to use the GPIO of the Pi
import RPi.GPIO as GPIO
import time

# Pin Definiton:
ledPin = 23

# Pin Setup:
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output

while True:
    GPIO.output(ledPin, GPIO.HIGH)
    print("LED is On")
    time.sleep(1)
    GPIO.output(ledPin, GPIO.LOW)
    print("LED is Off!")
    time.sleep(1)
```

Your code would look like this below.



```
pi@raspberrypi: ~/python_examples
GNU nano 2.2.6 File: Blink_GPIO_BCM.py

include the library RPi.GPIO to use the GPIO of the Pi
import RPi.GPIO as GPIO
import time

# Pin Definiton:
ledPin = 23

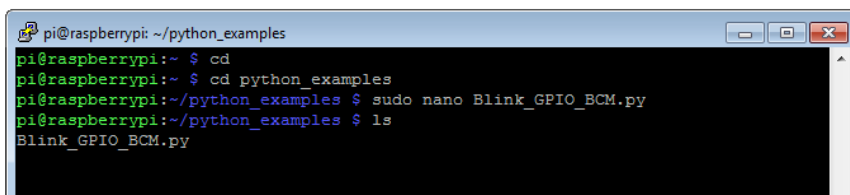
# Pin Setup:
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output

while True:
    GPIO.output(ledPin, GPIO.HIGH)
    print("LED is On")
    time.sleep(1)
    GPIO.output(ledPin, GPIO.LOW)
    print("LED is Off!")
    time.sleep(1)
```

Now save the file by pressing in your keyboard **Ctrl+x**, then **Y**, then **Enter**.

To check if we have saved the python file in the directory folder, type the command below. This will list all the files that is inside the directory.

**pi@raspberrypi: ~ ls**



```
pi@raspberrypi: ~/python_examples
pi@raspberrypi:~ $ cd
pi@raspberrypi:~ $ cd python_examples
pi@raspberrypi:~/python_examples $ sudo nano Blink_GPIO_BCM.py
pi@raspberrypi:~/python_examples $ ls
Blink_GPIO_BCM.py
```

## Run the Python code!

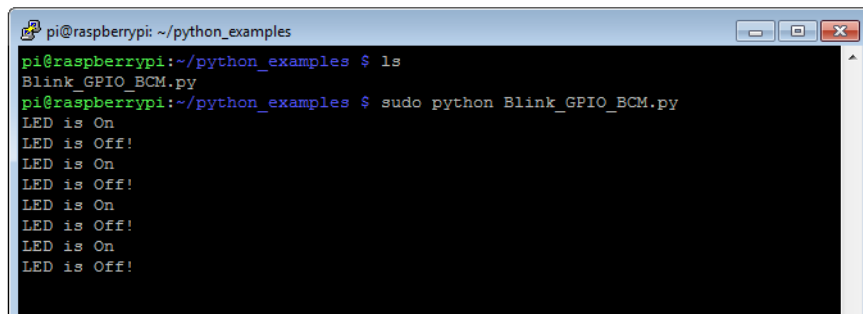
To run the python script, issue the commands below. This command tells the computer to run a python code with its assigned filename.

**pi@raspberrypi: ~ sudo python Blink\_GPIO\_BCM.py**

## The Output

Congratulations! Your first python script is running!

You should see that the LED is blinking every second. At the same time, a text will also appear on the screen that tells the LED is either On or Off.



```
pi@raspberrypi: ~/python_examples
pi@raspberrypi:~/python_examples $ ls
Blink_GPIO_BCM.py
pi@raspberrypi:~/python_examples $ sudo python Blink_GPIO_BCM.py
LED is On
LED is Off!
LED is On
LED is Off!
LED is On
LED is Off!
LED is On
LED is Off!
```

You just had your first python script! To cleanly exit simply press your keyboard's **Ctrl+C**.

## Code Explained

The second and third line of your code tells the system to use the **RPi.GPIO** library and also use a **time** library which we use to set a 1 second delay. The sixth line assigns PIN#23 (using GPIO.BCM pin numbering system, in the T Cobbler adapter) as the output pin for the LED. The ninth and tenth line use GPIO.BCM Pin Numbering System and assign the pin for the LED as an output pin.

In the **while** loop, the code repeats showing ON and OFF text the Putty serial terminal and turning ON and OFF of the attached LED every 1 second.

## Example #2: Digital Input - Push button

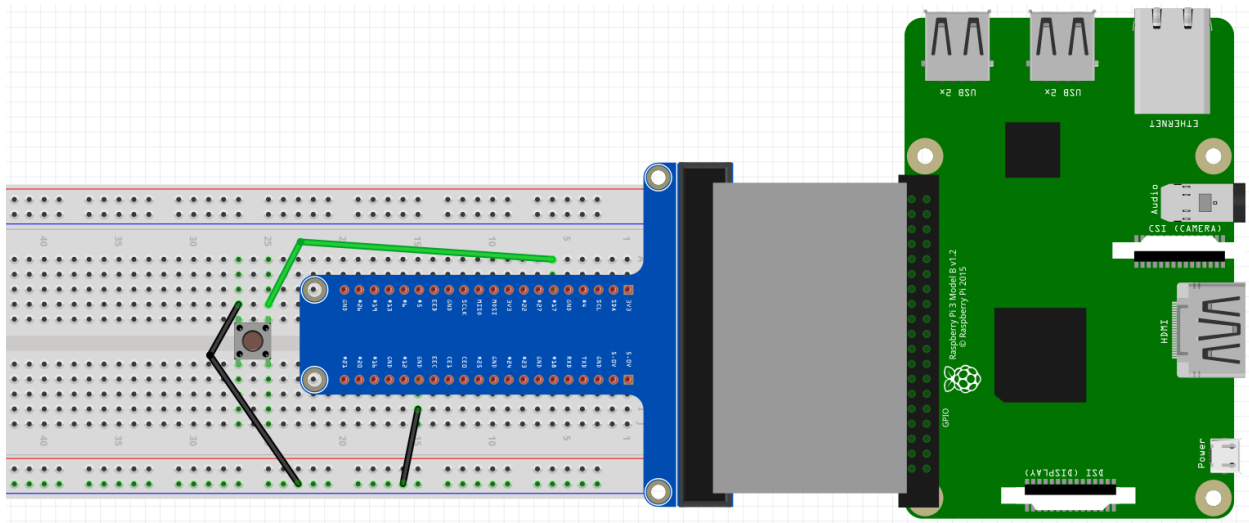


Fig. 2A

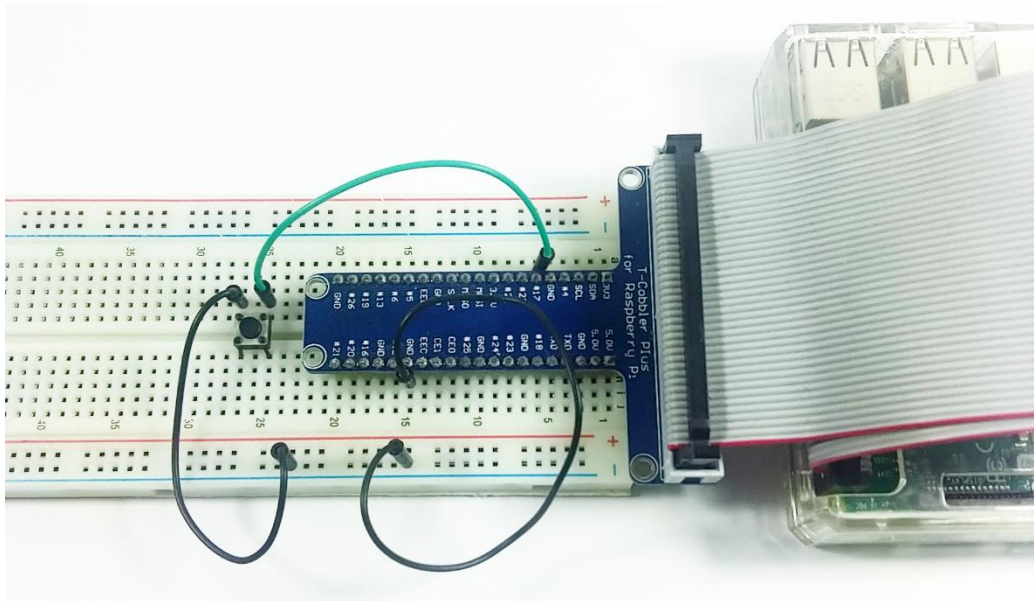


Fig. 2B

Our second experiment is to connect a Push button tactile switch (Digital Input) to one of the pins of our Raspberry Pi and display a text when the button is pressed (Switch Closed) or not pressed (Switch Open).

### Parts List:

- \*Raspberry Pi 3 Model B
- \*Raspberry Pi GPIO Adapter - T Cobbler (with ribbon cable)
- \*Breadboard - full size 830-points
- \*Push Button tactile switch (SPDT)
- \*Resistor - 330 ohms
- \*Connecting Wires

Assemble the circuit in Figure 2A. Your actual circuit would look like in Fig. 2B. In this setup we connect one leg of the push button to **PIN#17** of the Raspberry Pi GPIO pin through the GPIO Adapter board. The other leg of the button is connected to the Ground.

Just like the previous experiment we will use Pin Numbering System **GPIO.BCM** configuration and on the rest of the experiments in the following pages.

## Python code (using RPi.GPIO library)

Now create a python script to read button presses. Since we are still in the directory folder **python\_examples** that we created from the previous experiment let's create a new blank text file and name it **ButtonPress\_GPIO\_BCM.py** by typing the command below.

```
pi@raspberrypi: ~ sudo nano ButtonPress_GPIO_BCM.py
```

In the blank text editor type the script found on the Sample Python code below. Use your keyboard arrow keys to navigate to the blank spaces.

## Sample Code

```
# include the library RPi.GPIO to use the GPIO of the Pi
import RPi.GPIO as GPIO

# Pin Definiton
inputpin = 17

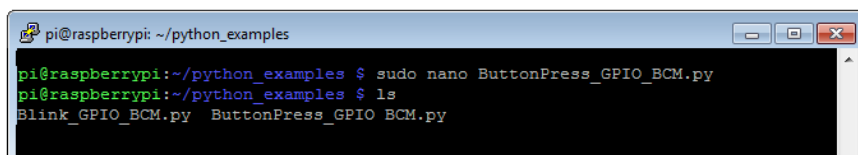
# Pin Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(inputpin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    if GPIO.input(inputpin):
        print ("Switch Open")
    else:
        print ("Switch Closed")
```

Now save the file by pressing in your keyboard **Ctrl+x**, then **Y**, then **Enter**.

Check if we have saved the python file in the directory folder, type the command below.

```
pi@raspberrypi: ~ ls
```





## Run the Python code!

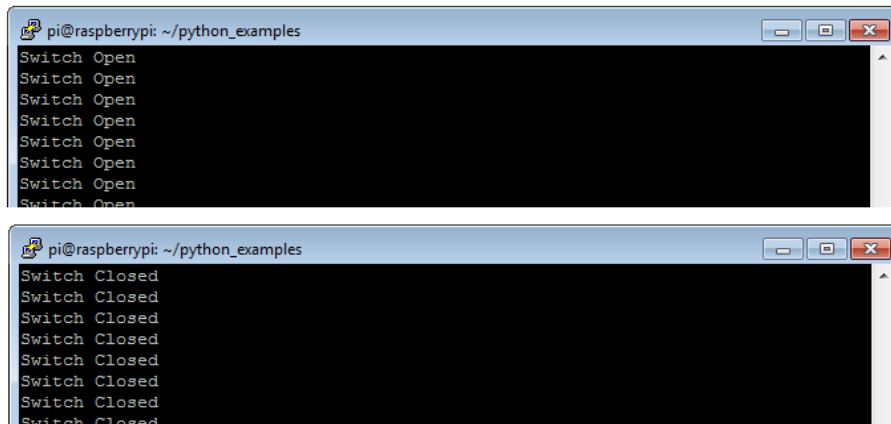
Run the python script with the command below.

```
pi@raspberrypi: ~ sudo python ButtonPress_GPIO_BCM.py
```

## The Output

Your python script is running.

You should see that in the screen a text “Switch Open” will appear since the button is not pressed. Now press the push button and you will see a “Switch Closed” text in the screen.



To exit press your keyboard **Ctrl+C**.

## Code Explained

The second line of the code means to use the **RPi.GPIO** library. The fifth line assigns PIN#17 (using GPIO.BCM pin numbering system, in the T Cobbler adapter) as the input pin for the Push Button. The eighth line means that we will use the GPIO.BCM Pin Numbering System. The ninth line tells the system that will use inputpin (PIN#17) as an input and use the Pi’s built-in internal Pull-up resistor mechanism.

In the **while** loop, the code repeats in showing “Switch Open” if the button is not pressed and “Switch Closed” if the button is pressed.

### Example #3: Analog Output (PWM) plus Digital Output/Input .

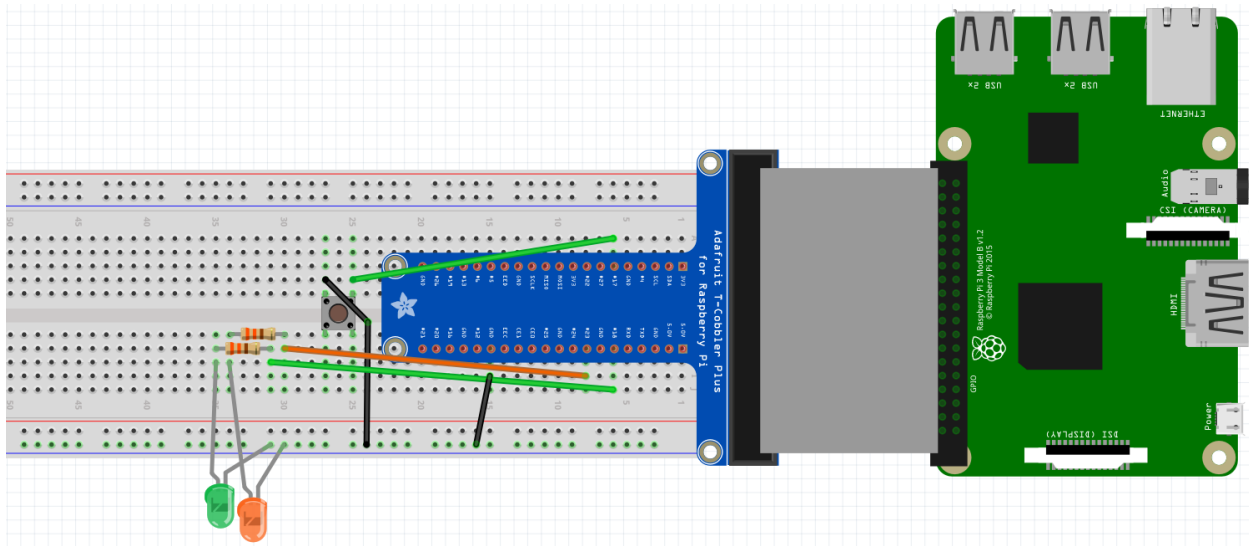


Fig. 3A

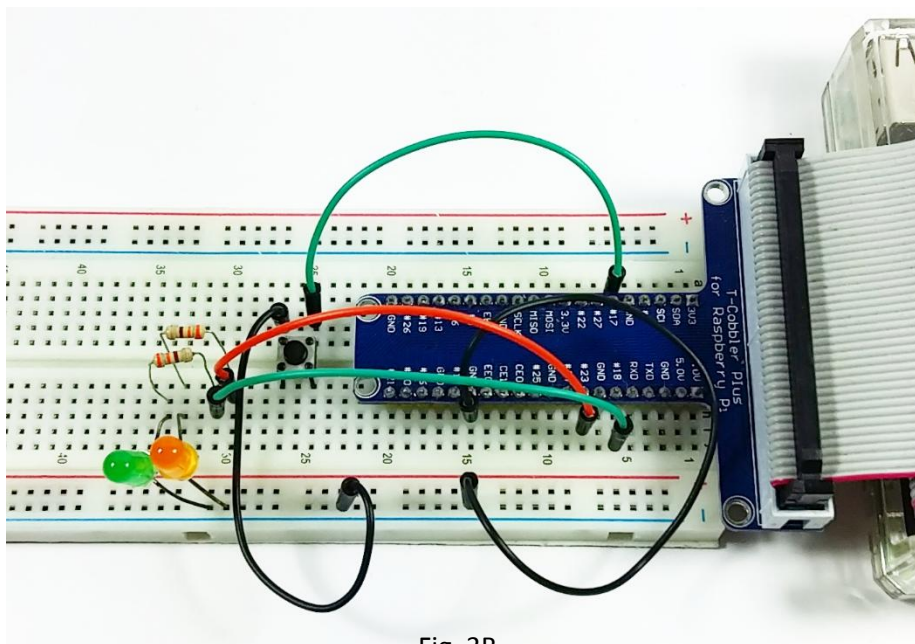


Fig. 3B

Our third experiment is to combine blinking LED (Digital Output), pressing a push button (Digital Input) and demonstrate how to create a PWM output (Analog Output).

#### Parts List:

- \*Raspberry Pi 3 Model B
- \*Raspberry Pi GPIO Adapter - T Cobbler (with ribbon cable)
- \*Breadboard - full size 830-points
- \*Push Button tactile switch (SPDT)
- \*LED (Red)
- \*Resistor - 330 ohms
- \*Connecting Wires

Assemble the circuit in Figure 3A. Your actual circuit would look like in Fig. 3B. In this setup we have combined the circuit from Experiment 1 and Experiment 2 and added additional Green LED connected to PIN#18 as output to show a PWM example. PIN#18 connects to one end of the 330 Ohm resistor and the other end of this resistor connects to the long leg of the Green LED. The short leg of the Green LED is connected to Ground.

## Python code (using RPi.GPIO library)

Now create a python script to read button presses, blink an LED, output a PWM signal and display some text. Still in the directory folder create a blank text file and name it **Blink\_Button\_PWM.py** by typing the command below.

```
pi@raspberrypi: ~ sudo nano Blink_Button_PWM.py
```

On the blank editor type the script on the Sample Python code below and use your keyboard arrow keys to navigate to the blank spaces.

## Sample Code

```
# Install external library to use using import command
import RPi.GPIO as GPIO
import time

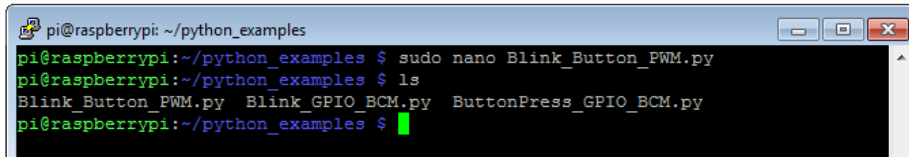
# Declare pin Variables
LED_pin = 23 # LED pin, BCM pin 23, BOARD pin 16
BUTTON_pin = 17 # Push Button pin, Active-low button, BCM pin 17, BOARD pin 11
PWM_pin = 18 # PWM LED, BCM pin 18, BOARD pin 12
dutyCycle = 80 # Duty cycle (from 0 to 100) for PWM pin

# Setup pin configurations
GPIO.setmode(GPIO.BCM) # (Broadcom) BCM pin-numbering system
GPIO.setup(LED_pin, GPIO.OUT) # LED pin set as Output
GPIO.setup(BUTTON_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Push Button pin set as Input w/ Pull-up
GPIO.setup(PWM_pin, GPIO.OUT) # PWM pin set as Output
pwm = GPIO.PWM(PWM_pin, 50) # Initialize PWM on PWM_pin 100Hz frequency

# Initial state for the LED's
GPIO.output(LED_pin, GPIO.LOW)
pwm.start(dutyCycle)
print("Python script running... Press the Button to start PWM. Press CTRL+C to exit")
try:
    while 1:
        if GPIO.input(BUTTON_pin): # Push Button is not pressed
            pwm.ChangeDutyCycle(dutyCycle)
            GPIO.output(LED_pin, GPIO.LOW)
        else: # Push Button is pressed:
            print("Button is pressed, PWM running at PWM pin!")
            pwm.ChangeDutyCycle(100-dutyCycle)
            GPIO.output(LED_pin, GPIO.HIGH)
            time.sleep(0.05)
            GPIO.output(LED_pin, GPIO.LOW)
            time.sleep(0.05)
except KeyboardInterrupt: # Exit in the code cleanly when CTRL+C is pressed
    pwm.stop() # stop the PWM
    GPIO.cleanup() # Cleanup all the GPIO states
```

Save the file by pressing in your keyboard **Ctrl+x**, then **Y**, then **Enter**.  
Check if we have saved the python file in the directory folder with the command below.

```
pi@raspberrypi: ~ $ ls
```

A terminal window on a Raspberry Pi showing the command 'ls' being executed in the directory ~/python\_examples. The output lists three files: Blink\_Button\_PWM.py, Blink\_GPIO\_BCM.py, and ButtonPress\_GPIO\_BCM.py.

```
pi@raspberrypi: ~/python_examples
pi@raspberrypi:~/python_examples $ sudo nano Blink_Button_PWM.py
pi@raspberrypi:~/python_examples $ ls
Blink_Button_PWM.py  Blink_GPIO_BCM.py  ButtonPress_GPIO_BCM.py
pi@raspberrypi:~/python_examples $
```

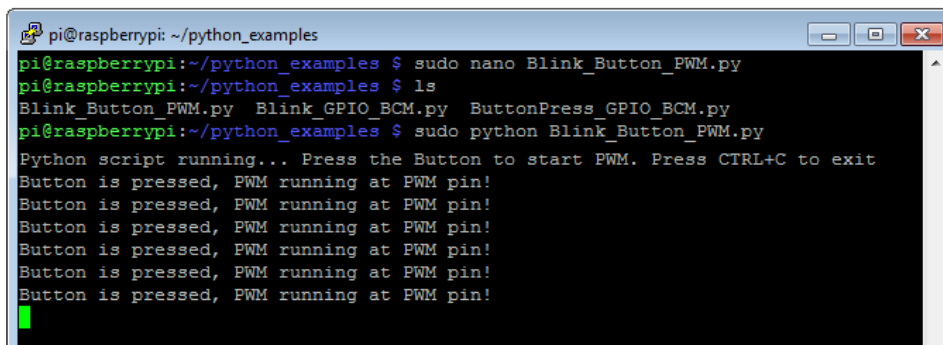
## Run the Python code!

Run the python script with the command below.

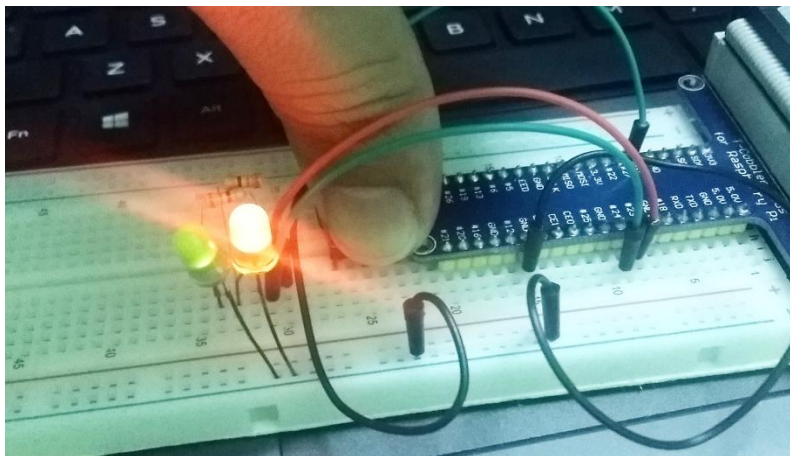
```
pi@raspberrypi: ~ $ sudo python Blink_Button_PWM.py
```

## The Output

A text will display that your python script is now running. The Green LED will also light up. If you press the push button in your circuit a message will display that it has been pressed and the PWM is running in the PWM output pin PIN#18.

A terminal window showing the execution of the Blink\_Button\_PWM.py script. It displays the initial file listing, followed by the command to run the script. The output shows the script running and responding to button presses with messages like 'Button is pressed, PWM running at PWM pin!'.

```
pi@raspberrypi: ~/python_examples
pi@raspberrypi:~/python_examples $ sudo nano Blink_Button_PWM.py
pi@raspberrypi:~/python_examples $ ls
Blink_Button_PWM.py  Blink_GPIO_BCM.py  ButtonPress_GPIO_BCM.py
pi@raspberrypi:~/python_examples $ sudo python Blink_Button_PWM.py
Python script running... Press the Button to start PWM. Press CTRL+C to exit
Button is pressed, PWM running at PWM pin!
Button is pressed, PWM running at PWM pin!
Button is pressed, PWM running at PWM pin!
Button is pressed, PWM running at PWM pin!
Button is pressed, PWM running at PWM pin!
Button is pressed, PWM running at PWM pin!
```



When the button is pressed you will also notice that the Green LED will slightly dim since we programmed the PWM duty cycle to decrease. Also, the orange LED will blink continuously.

To exit press your keyboard **Ctrl+C**.

## Code Explained

This experiment combines the circuit and code of Experiment 1 and 2 and added one feature the Analog Output (PWM). To demonstrate a PWM output we declared PIN#18 as the PWM pin.

The first group code lines calls the libraries to use. The second code group assigned variables for use. The third group sets up the hardware configuration and assigned them as input, output and PWM pins. The last code group executes the code and runs repeatedly for which when the button is pressed or not and displays a message in the screen.

To understand more on what the code does, check back on the Sample Code above and check out comments on each code lines.



## Example #4: Analog Input with MCP3008

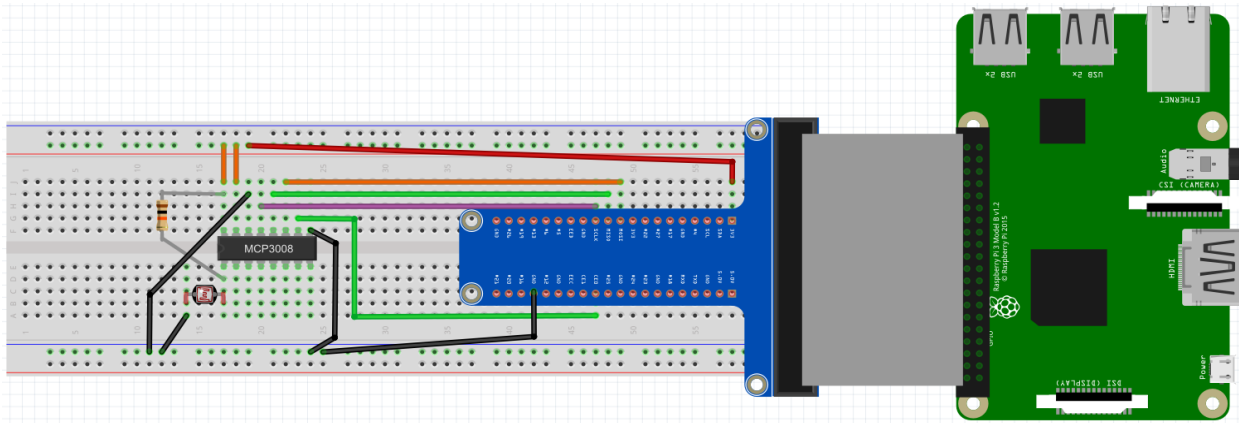


Fig. 4A

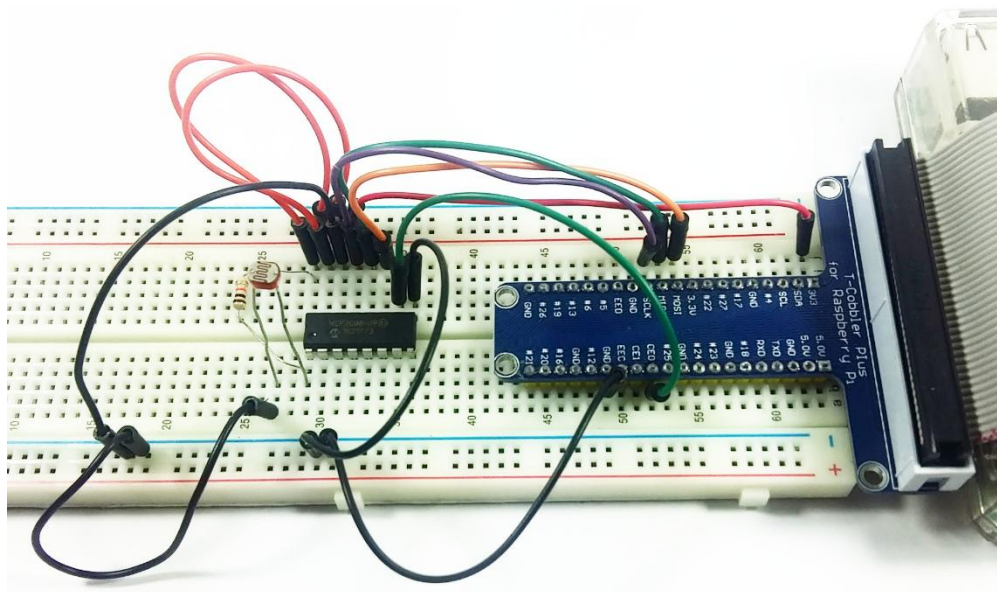


Fig. 4B

In this Example number 4 we will use the MCP3008 Analog-to-Digital Converter IC (with SPI Interface) to read analog input sensors and send them digitally to the Raspberry Pi (since the Pi does not have Analog inputs). The Raspberry Pi's SPI pins will connect to the MCP3008's SPI pins to get those digitized data that the Pi can now use. The chip can read up to 8 channels of Analog inputs so now we can get analog data's from sensors like the LDR light sensor or LM35 temperature sensor.

### Parts List:

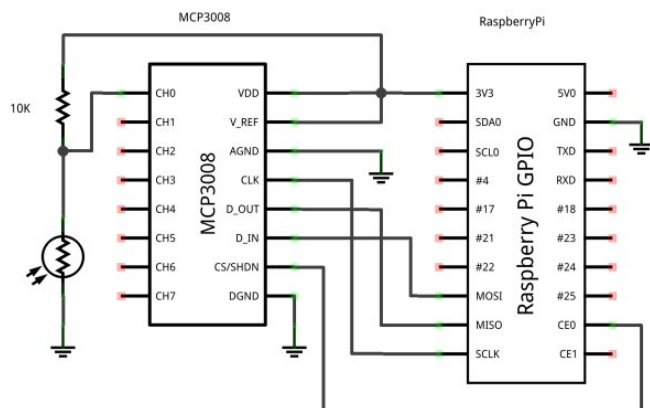
- \*Raspberry Pi 3 Model B
- \*Raspberry Pi GPIO Adapter - T Cobbler (with ribbon cable)
- \*Breadboard - full size 830-points
- \*LDR (light sensor, photocell)
- \*Resistor – 10k Ohms
- \*Connecting Wires

Assemble the circuit on Figure 4A and your actual circuit should look like the one in Fig. 4B. You may zoom in the image to clearly see the wiring connections.

Here we placed the MCP3008 chip on our breadboard and wire the SPI pins of the chip to the SPI pins of the Raspberry Pi through the Pi GPIO Extension Adapter. A 10k Ohm resistor is connected to 3.3V power rail and the other end of this resistor is connected to Channel 0 (CH0) of the MCP3008 chip. One leg of the LDR is connected also in the CH0 of the MCP3008 and the other leg of the LDR is connected to Ground. Review your MCP3008 and RPi pin wiring with the matching and the image below.

<u>MCP3008</u>		<u>RPi (extension adapter board)</u>	
VDD	to	3.3V	
VREF	to	3.3V	
AGND	to	GROUND	
CLK	to	SCLK	
DOUT	to	MISO	
DIN	to	MOSI	
CS/SHDN	to	CE0	
DGND	to	GROUND	

The CH0-CH7 pins are the 8 Analog inputs.



## Python code

Our circuit is ready. Let us now create our python script. We will store the python file in the same directory folder created earlier. In your **python\_examples** directory, create a blank text file named **rpi-analog-input-mcp3008.py** by typing,

```
pi@raspberrypi: ~ sudo nano rpi-analog-input-mcp3008.py
```

In your nano text editor type in the python code below on the Sample Code section

## Sample Code

```
# import libraries to use
import spidev
import time

# initialize SPI bus
spi = spidev.SpiDev()
spi.open(0,0)

# function to read SPI data from MCP3008 IC, 8 channels (channel 0 to 7)
def readChannel(channel):
    adc = spi.xfer2([1,(8+channel)<<4,0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

# Function to convert data to voltage level,
# rounded to specified number of decimal places.
def convertVolts(data,place):
    volts = (data * 3.3) / float(1023)
    volts = round(volts,place)
    return volts

# Define LDR sensor pin channel
ldr_pinChannel = 0

while True:

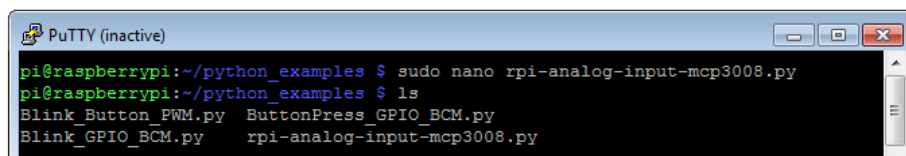
    # Read LDR light sensor data and convert to voltage
    ldr_value = readChannel(ldr_pinChannel)
    ldr_volts = convertVolts(ldr_value,2)

    # Print the data on the screen
    print "-----"
    print("LDR Light : {} ({} Volts)".format(ldr_value,ldr_volts))

    # Put short delay for each reading
    time.sleep(1)
```

Save the file by pressing in your keyboard **Ctrl+x**, then **Y**, then **Enter**.

Enter the command **ls** in the Putty terminal and verify that the file **rpi-analog-input-mcp3008.py** is listed.



The screenshot shows a PuTTY terminal window titled 'PuTTY (inactive)'. The terminal output shows the user at the prompt 'pi@raspberrypi:~/python\_examples' running the command 'sudo nano rpi-analog-input-mcp3008.py'. After pressing 'ls', the terminal lists the files: 'Blink\_Button\_PWM.py', 'ButtonPress\_GPIO\_BCM.py', 'Blink\_GPIO\_BCM.py', and 'rpi-analog-input-mcp3008.py'.

## Run the Python code!

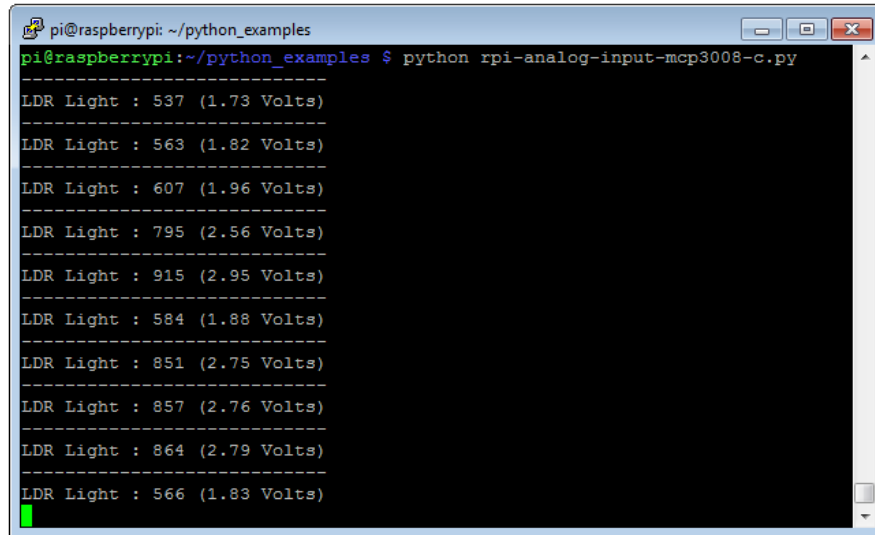
To run the python script, issue the commands below in your Putty terminal.

```
pi@raspberrypi: ~ $ sudo python rpi-analog-input-mcp3008.py
```

## The Output

Your python script for MCP3008 analog input is now running. A text will print out in the terminal the reading of the LDR's analog voltage in 0 to 1024 range and the equivalent 0 to 3.3 Volts range.

Now place your hand near and away from the LDR light sensor and you will notice the LDR light values on your terminal will change. If you totally cover the LDR the value will approach to 1024 (3.3 Volts). If you put a very bright light in the LDR the value will approach to 0 (0 Volts).



```
pi@raspberrypi: ~/python_examples
pi@raspberrypi:~/python_examples $ python rpi-analog-input-mcp3008-c.py
-----
LDR Light : 537 (1.73 Volts)
-----
LDR Light : 563 (1.82 Volts)
-----
LDR Light : 607 (1.96 Volts)
-----
LDR Light : 795 (2.56 Volts)
-----
LDR Light : 915 (2.95 Volts)
-----
LDR Light : 584 (1.88 Volts)
-----
LDR Light : 851 (2.75 Volts)
-----
LDR Light : 857 (2.76 Volts)
-----
LDR Light : 864 (2.79 Volts)
-----
LDR Light : 566 (1.83 Volts)
```

To exit simply press your keyboard's **Ctrl+C**.

## Code Explained

Check out the comments on each of the code line groups in our python code. Each code groups has its own comments to describe what it does. The first parts is calling out the libraries to use. The second initializes the SPI bus to use. Then we created a code function to read the data from the channel of the MCP3008 Analog to Digital chip. We also created a function to convert the analog reading from 0-1023 into its voltage equivalent. In our **while** loop we read the analog data from the LDR pin (CH0) and convert it to Volts and display them in the terminal. We also included a 1 second delay per analog reading sequence.

Next up, we will try out how to run C programs on our Raspberry Pi using the **wiringPi** C program library.

## Example #5: Running C programs on the Pi

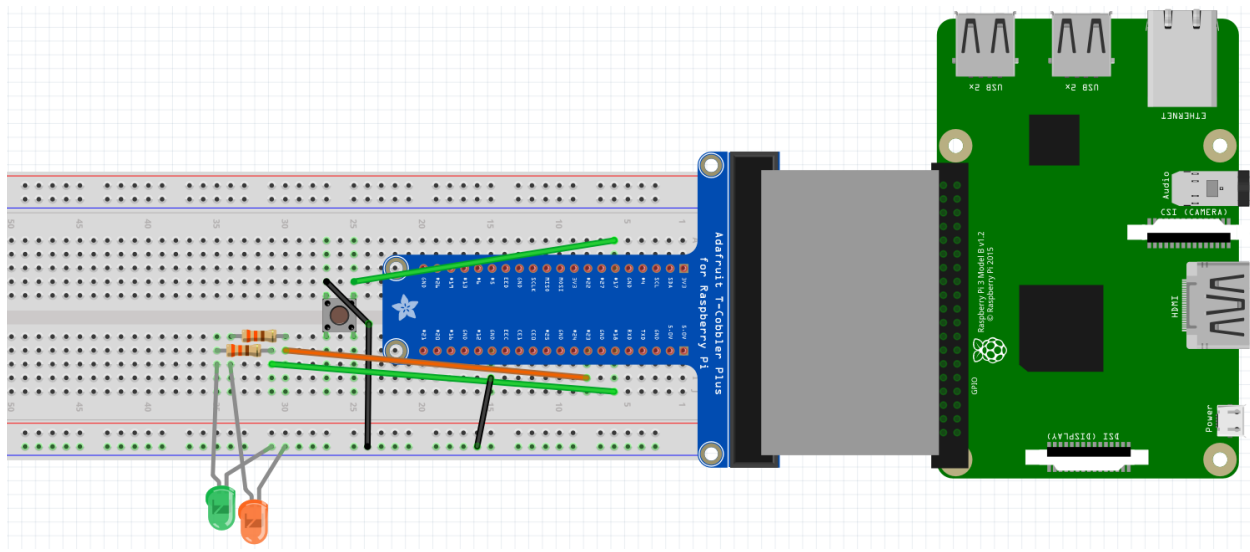


Fig. 5A

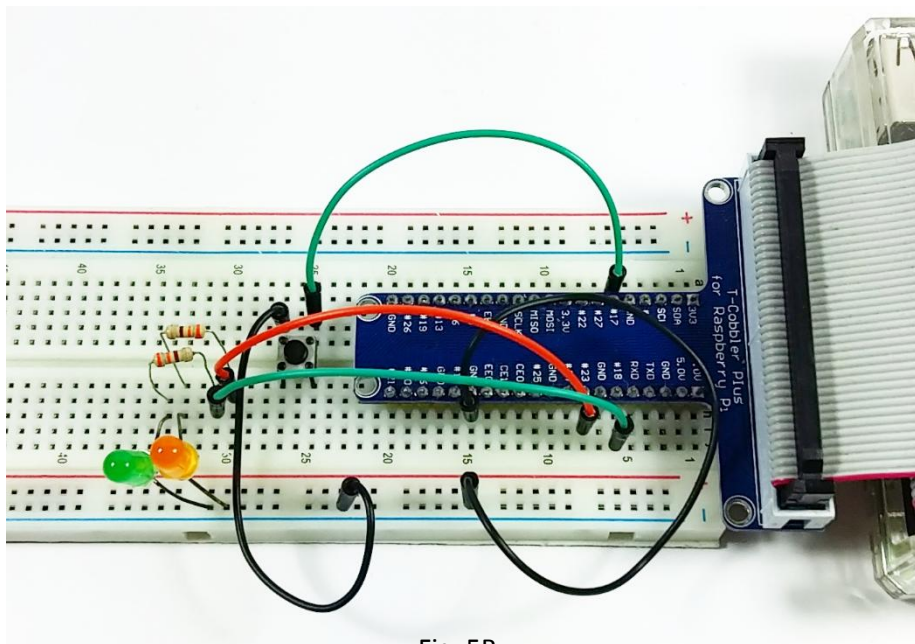


Fig. 5B

The fifth experiment is the same circuit with Experiment #3. We combine blinking LED (Digital Output), pressing a push button (Digital Input) and demonstrate how to create a PWM output (Analog Output) but this time using the **wiringPi** C program library to run in our Raspberry Pi.

### Parts List:

- \*Raspberry Pi 3 Model B
- \*Raspberry Pi GPIO Adapter - T Cobbler (with ribbon cable)
- \*Breadboard - full size 830-points
- \*Push Button tactile switch (SPDT)
- \*LED (Red)
- \*Resistor - 330 ohms
- \*Connecting Wires



Assemble the circuit in Figure 5A and your actual circuit would look like in Fig. 5B. In this setup we have the circuit from Experiment 1 and Experiment 2 combined and added additional Green LED connected to PIN#18 to serve as PWM output LED. PIN#18 connects to one end of the 330 Ohm resistor and the other end of this resistor connects to the long leg of the Green LED. The short leg of the Green LED is connected to Ground.

In order to run C programs in the Raspberry Pi we will use a library called **wiringPi**. This wiringPi library was written to compile C codes and translate it to run on the Pi. This library was designed to be “Arduino-like” coding style and we will see this later in our example code.

### C code (wiringPi library)

We have already installed and compiled our **wiringPi** library. If not, go back to Section 06 and come back here to proceed.

Let's go back to the home directory by typing **cd** in the terminal and press Enter. Next type in **cd c\_examples** and press Enter to go to the directory folder where we will save our C program file.

Now create a blank text file and name it **Blinker\_PWM.c** by typing the command below. Notice that the last letter is now “.c”.

```
pi@raspberrypi: ~ sudo nano Blinker_PWM.c
```

On the blank editor type the code found on the Sample C code below and remember to use your keyboard arrow keys to navigate to the blank spaces.

## Sample C code

```
#include <stdio.h> // Use this for printf() statement
#include <wiringPi.h> // Use the wiringPi library

// Assign pin number variables. We will use GPIO.BCM pin numbering system
const int PWMpin = 18; // PWM LED, BCM pin 18, BOARD pin 12
const int LEDpin = 23; // LED pin, BCM pin 23, BOARD pin 16
const int BUTTONpin = 17; // Push Button pin, Active-low button, BCM pin 17, BOARD pin 11
const int PWMvalue = 250; // Set this for PWM LED brightness

int main(void)
{
    // Setup wiringPi GPIO pins
    wiringPiSetupGpio(); // Initialize wiringPi, this is using GPIO.BCM pin system

    pinMode(PWMpin, PWM_OUTPUT); // Set PWM LED as PWM Output
    pinMode(LEDpin, OUTPUT); // Set the normal LED as Output
    pinMode(BUTTONpin, INPUT); // Set Push Button as Input
    pullUpDnControl(BUTTONpin, PUD_UP); // Enable internal pull-up resistor on the Button

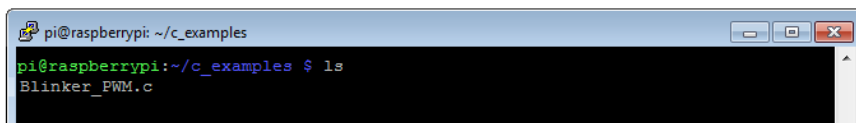
    printf("blinker is now running! Press CTRL+C to cleanly exit.\n");

    // repeating loop, while(1)
    while(1)
    {
        if (digitalRead(BUTTONpin)) // Button is not pressed, initially HIGH
        {
            pwmWrite(PWMpin, PWMvalue); // PWM LED is dim, low dutycycle
            digitalWrite(LEDpin, LOW); // normal LED is Off
        }
        else // Push button is pressed, digitalRead is now LOW
        {
            // PWM LED is now brighter, dutycycle up at around 77%
            pwmWrite(PWMpin, 1024 - PWMvalue);
            // we will blink the normal LED
            digitalWrite(LEDpin, HIGH); // normal LED is ON
            delay(75); // Wait for 75 ms
            digitalWrite(LEDpin, LOW); // normal LED is OFF
            delay(75); // Wait for 75 ms
        }
    }

    return 0;
}
```

Save the file by pressing in your keyboard **Ctrl+x**, then **Y**, then **Enter**.

Enter the command **ls** in the Putty terminal and verify that the file **Blinker\_PWM.c** is listed.



## Compile the C code!

We first have to compile the code before we can run it, issue the commands below to do it.

Note: -o is (dash and small letter "O"), -l is (dash and small letter "L")

```
pi@raspberrypi: ~ gcc -o blinker Blinker_PWM.c -l wiringPi
```

Then wait for 1 second to allow code compiling.

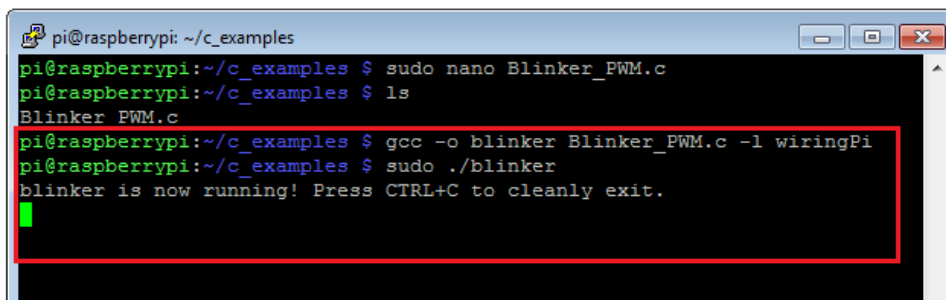
Note: The -l command is used to load the wiringPi library. The "**blinker**" is the C file to run in the system (shown in the next step) and the **Blinker\_PWM.c** is the C file that is compiled.

Now run the C code by typing below.

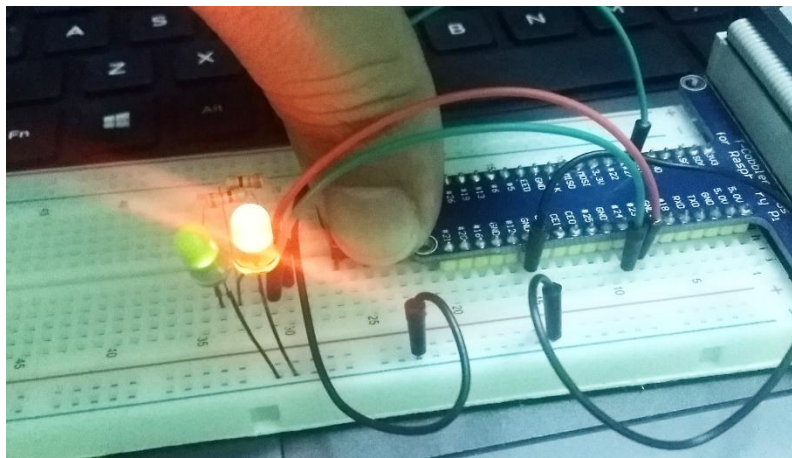
```
pi@raspberrypi: ~ sudo ./blinker
```

## The Output

A text will display that your blinker C code is running. The Green LED (as the PWM output LED, PIN#18) will slightly light up since the PWM output dutycycle is initially low. If you press the button in your circuit the PWM dutycycle will increase (about 77%) so the Green LED will increase in brightness and the normal LED (orange) will blink continuously.



```
pi@raspberrypi: ~/c_examples
pi@raspberrypi:~/c_examples $ sudo nano Blinker_PWM.c
pi@raspberrypi:~/c_examples $ ls
Blinker_PWM.c
pi@raspberrypi:~/c_examples $ gcc -o blinker Blinker_PWM.c -l wiringPi
pi@raspberrypi:~/c_examples $ sudo ./blinker
blinker is now running! Press CTRL+C to cleanly exit.
```



To exit press your keyboard **Ctrl+C**.

## Code Explained

This experiment is the same with Experiment 3 but instead of running Python code we use C programming codes with the use of **wiringPi** library. This experiment combines the circuit of Experiment 1 and 2 with Digital Input/Output with an additional Analog Output (PWM). We declared PIN#18 as the PWM pin and the PWM value will vary from 0 to 1024.

In our code we included comments on each code lines so we can check at what each of this line does.

In the first 2 lines we included library to allow printing text using printf() statement and also to install the wiringPi C programs library. In the second code group we assigned some variables to use. In the third code group where we have our main code activity we initialized the wiringPi library, assigned the variable to be Input/Output and in the repeating **while()** loop is our PWM, LED blinking and push button code.

## Going Further and What's next!

We have just learned some examples on how to use the Raspberry Pi's GPIO ports by connecting buttons, blinking LED's and connect a sensor. And we have tried running C codes too!

The Raspberry Pi is a powerful all-in-one computer. It can do some basic and complex programs too!

In the upcoming guides we will learn more about other applications involving the Pi that may tackle Raspberry Pi's SPI interface, i2c protocol, UART, and web applications. We will also try to discover on how to interface with other sensor boards, a web server, connection to Arduino/ESP8266, IoT (internet of things) application, robotics, interfacing with other language like Javascript and much more!

## Links and Resources

You will find more information and explore on some of the links below.

<https://www.raspberrypi.org/> - Raspberry Pi Organization official site and info

<http://wiringpi.com/> , <https://projects.drogon.net/raspberry-pi/wiringpi/> - information page about the wiringPi c programming library for Pi

[http://elinux.org/RPi\\_Low-level\\_peripherals](http://elinux.org/RPi_Low-level_peripherals) - Raspberry Pi low level peripheral information

<https://pypi.python.org/pypi/RPi.GPIO> - more Raspberry Pi information about the GPIO

## Hardware parts for Raspberry Pi computer

- [Raspberry Pi 3 Model B+ \(B plus\)](#)
- [Breadboard solderless - Full size 830points](#)
- [Raspberry Pi GPIO Adapter - T Cobbler \(with ribbon cable\)](#)
- [Connecting Wires - Male-Male, 65pcs \(assorted-size\)](#)
- [MicroSD Card with Raspberry Pi OS \(8GB, class10\)](#)
- [LED - Red \(5mm\)](#)
- [LED - Green \(5mm\)](#)
- [LED - Yellow \(5mm\)](#)
- [Push Button tactile switch \(SPDT\)](#)
- [Resistors \(330 ohms 10pcs\)](#)
- [Resistors \(10k ohms 10pcs\)](#)
- [Resistors \(1k ohms 10pcs\)](#)
- [LDR \(Light Dependent Resistor\)](#)
- [Power Supply Wall Adapter - 5V/2A \(2000mA\) \(micro-USB\)](#)
- [Raspberry Pi Case \(Clear\)](#)
- [MCP3008 - 8-Channel 10-Bit ADC With SPI Interface](#)
- [USB to RS232 Serial TTL Converter Cable \(PL2303\)](#)

