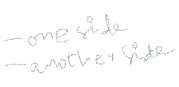
Raspberry Pi-Push Button Tutorial (with Python 3)

Push button Pi

One side Pin 6 GND

Another side Pin 36 GPIO16 (bottom right 3rd pin)





We are selecting the internal pull-up pull-down resistors.

Open Thonny Python IDE on pi.

Click New.

import RPi.GPIO as GPIO

BUTTON\_PIN=16 #variable to store the GPIO pin number

GPIO.setmode(GPIO.BCM) #since we are using GPIO number, mention BCM

#set the pin as input with pull-up resistor. Button not pressed=HIGH(3.3V) and Button pressed=LOW(0V)

GPIO.setup(BUTTON\_PIN, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

#Read the button state. Based on it, print something

if GPIO.input(BUTTON\_PIN) == GPIO.LOW:

print(“Button is pressed”)

else:

print(“Button is not pressed”)

GPIO.cleanup() #its a good practice to do this. Bcoz, it will set the GPIO pins to its default configuration

Program2: Same program but put the if..else condition in while loop

import RPi.GPIO as GPIO

import time

BUTTON\_PIN=16 #variable to store the GPIO pin number

GPIO.setmode(GPIO.BCM) #since we are using GPIO number, mention BCM

#set the pin as input with pull-up resistor. Button not pressed=HIGH(3.3V) and Button pressed=LOW(0V)

GPIO.setup(BUTTON\_PIN, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

#after this select the all code presnt in if..else then press tab so that this part becomes the part of while loop.

#I am not going to use this for full speed. As it will consume CPU. I am going to add time.sleep for 0.1 so that this basically reads the button status 10 times per second and prints button pressed or not.

#since we are using while loop i.e. infinite loop, we have to press ctrl+c from that while loop we will not be able to execute the gpio.cleanup because the program is going to exit from one of those lines in the while loop. So, what we can do is to handle the exception. So, put the entire while loop in “try:”

try:

while True:

time.sleep(0.1)

#Read the button state. Based on it, print something

if GPIO.input(BUTTON\_PIN) == GPIO.LOW:

print(“Button is pressed”)

else:

print(“Button is not pressed”)

except KeyboardInterrupt:

GPIO.cleanup()

#its a good practice to do this. Bcoz, it will set the GPIO pins to its default configuration

Save the above program with the name “button.py”

Then run the program by clicking the play button.

#Actually you can click on the “play” button. Then just make sure to click inside the “shell” window to select it and press ctrl+c.

Alternatively, open the terminal and go to the program file path.

pi@raspberrypi: ~$ cd tutorials/

pi@raspberrypi: ~/tutorials $ ls

lsit of files

pi@raspberrypi: ~/tutorials $ python3 button.py

Then the output starts in the terminal.

When button is not pressed, the appropriate message will be there. When pressed then accordingly.

It keeps on going.

To stop it, press CTRL+C

Program3: lets improve the program to indicate and detect when the button state is changed.

Same program. But changes in the while loop.

import RPi.GPIO as GPIO

import time

BUTTON\_PIN=16 #variable to store the GPIO pin number

GPIO.setmode(GPIO.BCM) #since we are using GPIO number, mention BCM

#set the pin as input with pull-up resistor. Button not pressed=HIGH(3.3V) and Button pressed=LOW(0V)

GPIO.setup(BUTTON\_PIN, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

Previous\_button\_state = GPIO.input(BUTTON\_PIN) #after setup the button pin, we are going to save the status of button into the previous\_button\_state variable.

try:

while True:

time.sleep(0.01) #read the button state 100 times per second

button\_state = GPIO.input(BUTTON\_PIN)

if button\_state != previous\_button\_state: #if there both variables are not equal, then it means there is change of state

previous\_button\_state=button\_state

if button\_state == GPIO.HIGH: #LOW means pressed, HIGH means released

print(“Button has just been released”)

except KeyboardInterrupt:

GPIO.cleanup()

#its a good practice to do this. Bcoz, it will set the GPIO pins to its default configuration

In this program, inside the while loop 3rd line we will know that button state has changed and in the 5th line we understands that the signal is HIGH. Therefore, we can confirm that the button is released. Coz, LOW means pressed, HIGH means released.

Execute the program.

Go to terminal.

pi@raspberrypi: ~/tutorials $ python3 button.py

After pressing enter following the above command, nothing will be seen on the terminal. Whenever you press the button and release it, then it will give the message.

Press CTRL+C to stop the program.

======End=======

**Power on LED when Button is Pressed**

Aim: Button press ---- LED ON; Button not pressed ----- LED OFF

from gpiozero import LED

from gpiozero import Button

or you can also write the following

from gpiozero import LED, Button

led=LED(17)

button=Button(26)

while True:

if button.is\_pressed:

led.on()

else:

led.off()

Execute the program. Press Run.

When you press the button LED must be ON otherwise OFF.

Need to improve this code. Because, in case of above program, if you open “Task manager”(go to prefernces->Task manager). If you search for ‘Python3” in the task list. You’ll observe lots of CPU% is being utilized. This is the main disadvantage. This will consume more CPU. In order to rectify this just include delay in the while loop.

Program2:

Same code, slight changes in while loop.

from gpiozero import LED, Button

import time

led=LED(17)

button=Button(26)

while True:

time.sleep(0.01) #10ms. This loop would be running for much much lower speed i.e. 100 times per second

if button.is\_pressed:

led.on()

else:

led.off()

Save the program. Press Stop icon. And press Run icon.

Now observe the task manager.

Program3:

Record/Register the button state and pause.

from gpiozero import LED, Button

import time

from signal import pause

led=LED(17)

button=Button(26)

button.when\_pressed=led.on

button.when\_released=led.off

pause()

In this program, the libraries are imported. Pins are initialized. Dictating, if button pressed, LED must be ON and if button not pressed, LED mist be OFF.

Real life, analogy, you asked somebody some information. They will tell you, Okay I’ll tell you when I have this information. You give them your phone number to register your phone number so that when they go that information they will call you back.

Therefore, there are external events happening in the program. The events are button pressed or not pressed. So, you are telling that if the button is pressed register that LED must be ON.

You can write like this also:

button.when\_pressed = led.on() #if you write like this you are calling the function led.on.

But we are writing like this:

Button.when\_pressed = led.on #here, there are no parenthesis. Coz, we are registering its functions so that you can call it later.

If you run the code, same thing happens.

**Toggle LEDs when Button is Pressed**

New Task:

Aim: Three LEDs. Single Push button.

< Create a function to switch on the LED

def switch\_led():

button.when\_pressed=switch\_led

< Now, we need to keep track of which led is On or OFF

led\_index=0

def switch\_led():

led\_index=1

from gpiozero import LED, Button

from signal inport pause

led1=LED(17)

led2=LED(27)

led3=LED(22)

button=Button(26, bounce\_time=0.05) #the second argument, bounce time will tell the program to wait for 50ms. i.e. whenever you read a value from push button, we are gonna ignore next few values over the next 50ms. Coz, in that 50ms you might have some physical bounces of button. So, it will stop to monitor for next 50ms. That means if you press the button for 20 times a sec it will work correctly. As a human we cannot press a button 20 times/sec.

led1.off()

led2.off()

led3.off()

led\_index=0

def swtich\_led():

global led\_index

#we are turning ON one Led only based on the led\_index. Then we switch off other leds. Then we increment the led\_index

If led\_index == 0:

led1.on()

led2.off()

led3.off()

led\_index +=1

elif led\_index==1:

led1.off()

led2.on()

led3.off()

led\_index +=1

#elif led\_index==2 #this is not required. Coz, we need not to go after 2. This is going to be last one. That’s why

else

led1.off()

led2.off()

led3.on()

led\_index =0 #we need to make it to default value

button.when\_pressed=switch\_led

pause()

Explanation: with the above code, when the button is pressed, it will call the function “switch\_led”. Then, it will execure the function. Which checks led\_index is equal to 0 or not. If it is 0 then it will switch ON led1 and so on. We are mentioning “global led\_index” because of this line “led\_index+=1”. Cause, in python, you cannot manipulate the global variable inside the function otherwise it will give error.