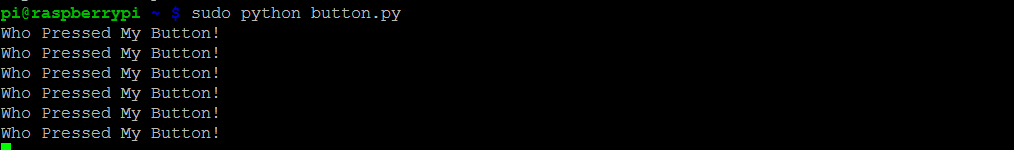
Buttons and switches are a fundamental part of ‘physical’ computing. This beginner’s tutorial is designed to teach the basics of physical operation with the Raspberry Pi using a simple momentary switch setup. In part one, we discussed and built our switch circuit. In this part, we’ll go through the steps of programming and interacting between the Pi and our components. The coding part of this tutorial will be completed on Python, a widely used general purpose language. It’s also very readable, so we can break down and explain the function of each line of code. The purpose of our code will be to read the I/O pin when the switch is pressed!



**Install the GPIO Python Library**

The first step is to install the RPi.GPIO Python library. There are several libraries available but we recommend you use the one found at **http://pypi.python.org/pypi/RPi.GPIO**.

Make sure you’re connected to the internet, open a terminal window on your raspberry pi, and download the library to your home directory. If a newer version has been released, simply replace the version number below with the current version.

***wget http://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.3a.tar.gz***

The downloaded file is compressed, so we use the tar command to extract it - “x” indicates extract, “v” indicates show output files, “z” denotes that we want to read from the file, and “f” specifies the output directory.

***tar xvzf RPi.GPIO-0.5.3a.tar.gz***

Change to the newly created directory (cd) and install the library:

**cd RPi.GPIO-0.5.3a**

**sudo python setup.py install**

We can now clean up our installation files by removing (rm) them from our home directory.

**cd**

**sudo rm -rf RPi.GPIO-0.5.3a/**

**sudo rm RPi.GPIO-0.5.3a.tar.gz**

**Making a Program in Python**

With the GPIO library installed, we’re all ready to start our Python project! Load up the Raspian GUI with **startx**, and load the program **IDLE 3** into which we’ll type our code. As we’re starting a new project, open up a new window **File>>New Window**. Remember that Python is case sensitive, and indentation is fundamental. Indentation, which is used to group statements, will occur automatically as you type commands, so make sure you stick to the suggested layout.

The first line of our code imports the Python library we’ve just downloaded into our project.

**import RPi.GPIO as GPIO**

We next need to set our GPIO pin numbering, as either the BOARD numbering or the BCM numbering. BOARD numbering refers to the physical pin numbering of the headers. BCM numbering refers to the channel numbers on the Broadcom chip. Either will do, but I personally prefer BCM numbering. If you’re confused, use a GPIO cheat sheet to clarify which pin is which!

**GPIO.setmode(GPIO.BCM)**

Now you need to define the GPIO pins as either Inputs or Outputs. In Part 1, we set BCM Pin 17:BOARD Pin 11 (***GPIO P17 [Pin 11])*** as our input pin. So our next line of code tells the GPIO library to set this pin as an input.

**GPIO.setup(17, GPIO.IN)**

In part 1 of the tutorial the input pin was tied high by connecting it to our 3.3V pin, and the purpose of our Python programme is to check to see if the if the input pin has been bought low e.g. when the button has been pressed. To check the high/low status of the pin we’re going to use a True or False statement running on an infinite loop.

We need to tie our true statement to the high value of our input pin. To do so we create a new variable called **input\_value** and set it the current value of ***GPIO P17 [Pin 11]***.

**while True:**

**input\_value = GPIO.input(17)**

Next we'll add some code so that the program will display a message when the button is pressed. For this, we’ll use the False statement e.g. when the input\_value (which is tied high) is no longer equal to the input\_value, which occurs when the button is pressed and it’s pulled low, execute a command.

**If input\_value == False**

**print(''Who pressed my button!)**

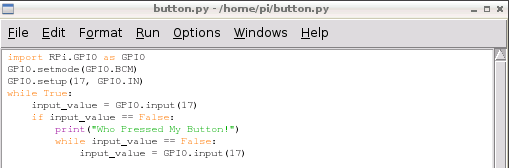
When the button is pressed the program will now display the text: “Who pressed my button!”, feel free the change this to anything you want.

**while input\_value == False:**

**input\_value = GPIO.input(17)**

The last two lines in the above code are very important, they create a loop that tells Python to keep checking the status of ***GPIO P17 [Pin 11]*** until it's no longer low (button released). Without this, the program would loop while the button is still being pressed meaning the message will be printed multiple times on the screen before you release the button.

The final program should look like this in python:



Save the file as button.py. In order to run the program, open a new terminal window on the Pi and type the following command:

**sudo python button.py**

At first nothing will happen, but if you press the button the program will print the defined message.



To exit a running Python script, simply hit CTRL+C on the keyboard to terminate.

If it hasn't worked don't worry. First check the circuit is connected correctly on the breadboard as defined in part 1, then that the jumper wires are connected to the correct pins on the GPIO port. If it still fails to work, double check each line on the program is correcting remembering that python is case-sensitive and indentation is correct. I find that typing the code out by hand will give better results than a simple copy/paste.

This is a deceptively simple program that can be used for many purposes. The same code could be used to read when the pins of separate devices, such as a sensor or external micro-controller, have been pulled high or low. Next time we’ll add an LED to our circuit, and make it flash on command from our button!