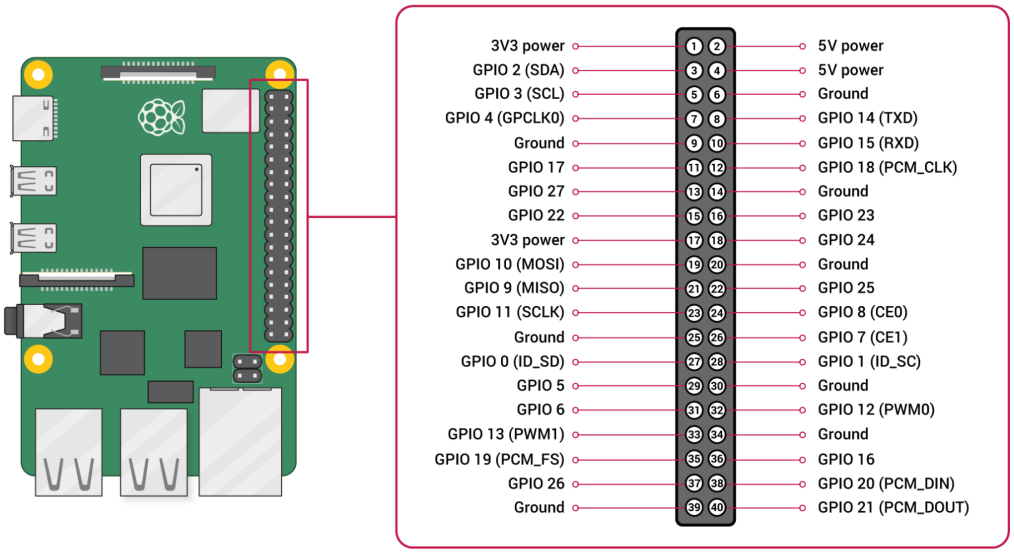
Outputting to Raspberry Pi Pins

Raspberry Pi contains GPIO (General-Purpose Input Output) pins that allow the Raspberry Pi to interact with external features through electronic circuits. The gpiozero module allows Python to use these GPIO pins with a number of supported components (e.g. LEDs, buttons).

These pins fall into a number of categories: 3V3, 5V, Ground, or GPIO. The chart below shows which pin is in what category on a Raspberry Pi.

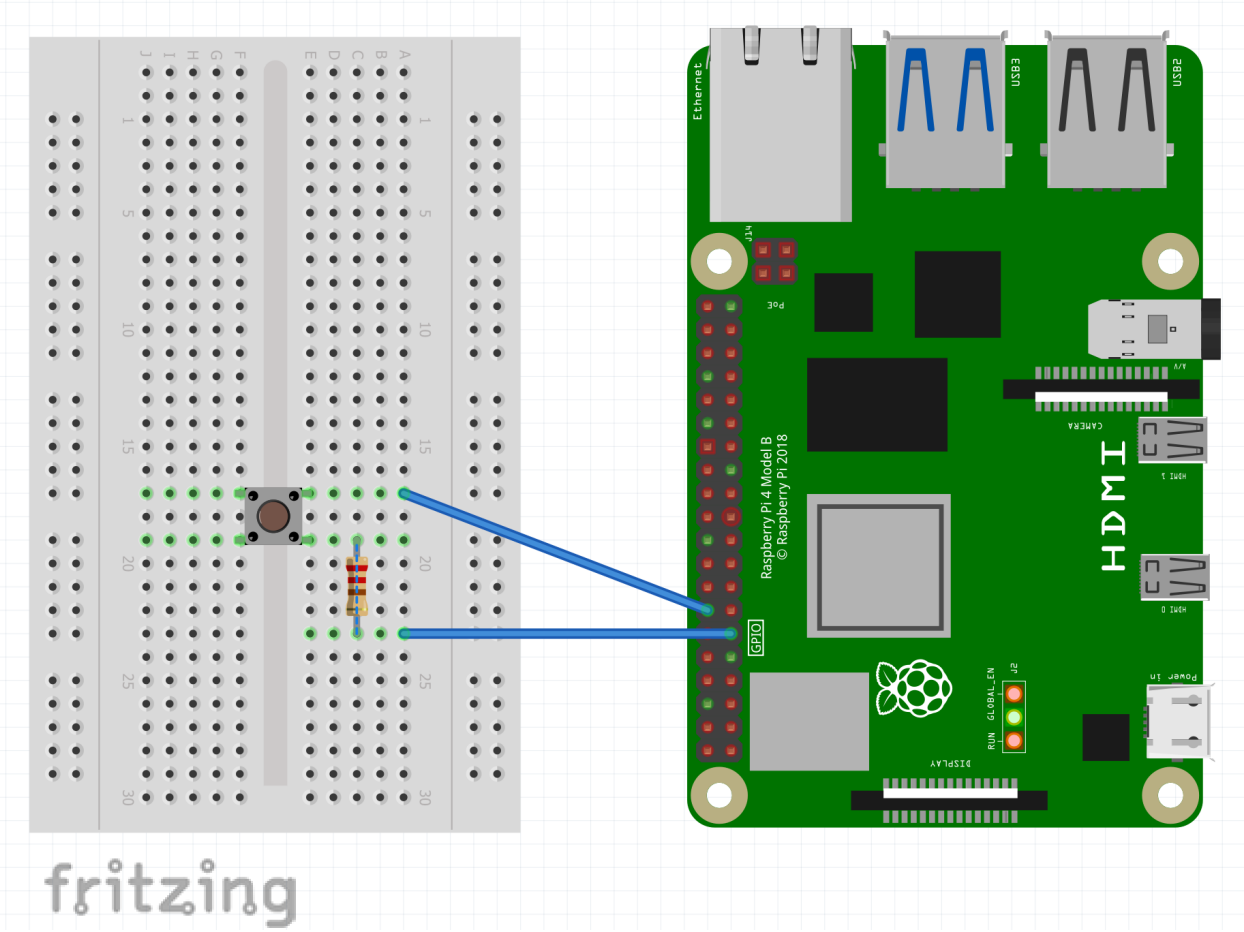


https://www.raspberrypi.org/documentation/usage/gpio/

Using a Button

This activity uses a button connected to a Raspberry Pi's GPIO pins to instruct the program when to collect data. Each time the button is pressed, the sensors will collect a single data point. The program will end when the button has been held down for 5 seconds.

1. Gather the following equipment:
    a. Two jumper cables with a plug on one end and a socket on the other,
    b. A breadboard,
    c. Aa Raspberry Pi,
    d. A button,
    e. A resistor of 100 ohms or more.



2. Create the circuit illustrated above.
    a. Create a series circuit that connects GPIO pin 17 to the button and resistor in a series circuit. The circuit is completed by connecting the free end to a ground (GND) pin.You should have a jumper wire connecting a GND (ground) pin to the resistor and one that connects a GPIO pin to the button. The program will be written using GPIO 17.

Open your Python editor and create a new Python file. Note: if you have not installed the gpiozero module on your Raspberry Pi, you will need to access the terminal and pip3 install it. Type `pip3 install gpiozero` in the terminal prompt to install this module.

Create the following code. Add comments to help you remember what the code accomplishes. We recommend skipping a line between these sections of code to make it easier to read.

Code	Explanation

`from gpiozero import LED`
`from gdx import gdx`
`gdx = gdx.gdx()`
	Import the required modules.
	Create a variable gdx to streamline your code

`gdx.open_ble() or gdx.open_usb()` `gdx.select_sensors()`	Only specify one type of “open” code - either USB or BLE. This opens the GDX device. The user is prompted to select a sensor, and then starts collecting data at a specified rate (100 milliseconds). This does not store any data, simply connects the sensor, and starts sampling data.
`button = Button(17)`	Creates a constant to represent the button on pin 17
`saved_data = []`	Creates a list to save data. This opens up opportunities for extensions and is optional.
`button.hold_time = 5`	hold_time is a command in gpiozero. This will be used to stop the program.

```
running = True
while running:
    if button.is_pressed:
        gdx.start(period=100)
        data_list = gdx.read()
        if data_list == None or len(data_list) == 0:
            break
        print(data_list[0])
     saved_data.append(data_list[0])
        gdx.stop()
```

This while loop starts data collection if the button is pressed, prints and stores the sensor reading, and stops data collection.

`    while button.is_pressed:`
`        if button.is_held:`
`            running = False`
`            Break`
`gdx.close()`
	This while loop is embedded in the previous while loop. As long as the button is pressed the program will loop through this section of code.
	When button.is_held is True (when the button.hold_time is met) the initial while loop is set to False and this loop is escaped.
	This ensures a single data point will be collected and provides a means of escaping the while loop.
	gdx.close() stops the sensor data collection.

This program is fully functional. Save and run the program now.

You will notice that the screen is full of output stating that the gdx sensor has started and stopped. The following code will allow you to eliminate this text. Linux contains a special file, /dev/null, that is a location that will receive information without cluttering the IDLE shell.

To access this, you will need to import two more modules. Follow these instructions to send these messages to the null location.

Code	Explanation
`import os` `import sys`	Add these two modules after the code to import other modules.
`out_to_null = open('/dev/null', 'w')` `out_og = sys.stdout`	out_to_null constant will direct output to the /dev/null location out_og constant saves the default path should you want to direct the output to the IDLE shell.
`sys.stdout = out_to_null`	Insert this before the `gdx.start(period=100)` and before the `gdx.stop()` lines of code. This sets the output to the /dev/null location
`sys.stdout = out_og`	Insert this code before `print(data_list[0])` and `gdx.close()` The addition of these lines of code toggles whether the output goes to the IDLE shell or to the /dev/null/ location.

Run the program and make sure it is working properly. You should now be able to press the button and have one data point printed to the screen, and end the program by holding the button for 5 seconds.

Adjusting the Program

Once you have successfully run the program make changes as suggested below to customize your program.

This activity saved the sensor readings in the `saved_data` list. Change this program to integrate a graph of these data values using the VPython module.