PRACTICAL DEV COMPANION

A charrold303 companion guide

Chris Harrold:
@charrold303
www.charrold303.com
bit.ly/charrold303

TABLE OF CONTENTS

WELCOME!

Thanks for attending one of my workshops, An Introduction to Practical Development! I am very excited that you have chosen to invest your time with me today, and in the spirit of respecting that investment, I have prepared this guide to make sure you have as much information as possible at your fingertips. It is broken into three sections:

1. *The glossary of terms lists common terms I will use during the presentation that you should know*
2. *The Useful Things section has some materials to help you during the course of the workshop that we will be using – charts, diagrams, etc.*
3. *Important Links has links to everything we will be using in a shortened format, so you can type them in from a printed copy more easily (they will work automatically from the digital copy of course)*

Thank you again for joining me today, and for investing in this session. I have worked hard to make sure you walk away with some new knowledge and skills that you can use to get started with your own projects!

GLOSSARY

Technology is full of acronyms that are really important and often times poorly or completely misunderstood. This glossary of terms will be helpful for understanding the presentation and materials we will cover in our workshop, and just for general knowledge!

HARDWARE TERMS:

Single Board Computer - a complete computer built on a single circuit board, with microprocessor(s), memory, input/output (I/O) and other features required of a functional computer.

Raspberry Pi – an example of a Single Board Computer that was designed for hobbyists and enthusiasts to offer a low-cost, fully functional computer in a small form-factor.

Sensor – a piece of hardware that detects some sort of condition and provides a signal based on the condition. Our sensor detects sound.

PIN – A part on a circuit board where an external device, sensor, input or output may be connected.

GPIO – an acronym that stands for General Purpose Input/Output. It is used to refer to any pin that is not defined by the pin itself but is defined at runtime (see "runtime" under Software Terms).

Breadboard - A breadboard is a construction base for prototyping of electronics. TRIVIA SIDE NOTE: (Originally it was literally a bread board, a polished piece of wood used for slicing bread. This was popular with early electronics builders because they were cheap, sturdy, and did not conduct electricity!)

Male and Female – this refers to the type of connection that a wire, sensor, or pin can accept. Breadboards are "Female" connections and so require a "Male" connector. The GPIO pins on the Raspberry Pi are "Male" and thus require a "Female" connector.

LED – Light Emitting Diode – a small device that uses a lot of very deep scientific principles to emit light. For our purposes, it is a small, electric light that can operate at VERY low voltage.

Circuit – any complete electrical connection is a circuit. From the lightbulb and light switch you used this morning, to your phone charger, to the circuit you will build in this workshop. Electricity flows from + to – through a circuit. No circuit, no flow of electricity.

Resistance – a property in electronics that allows us to reduce and control the flow of electricity through a circuit.

Resistor - a passive two-terminal electrical component that implements electrical resistance as a circuit element. This is the fancy way of saying that it causes the amount of electricity flowing through it to be reduced. Think of it as an electricity traffic cop for your circuit.

Pinout – this is the map of the Pins on the SBC to their function. Pinouts are critical for wiring a circuit – without it you do not know what the pins do!

Potentiometer a "variable resistor" that allows you to control the resistance of electricity through a circuit. Often embedded on a sensor, but can also be added by itself

SOFTWARE TERMS:

Code – the underlying construct of any computer software, app website, or anything that has to do with computers. There are many, many types of code. Code is written in

different languages. Like all languages, all different code languages have their own syntax, grammar rules, and general flow.

Python – A type of code language. There are many, this one is the one we will use. It is good for things like hardware interfaces because of its support for libraries that make integrations possible with little to no additional code.

Runtime – the time during which your code is running. This is when all the things you are telling the computer through your code to do actually happen.

Bug – an undocumented feature of your code. This is a behavior that you did not expect based on a set of conditions that happen through the use of your cod.

Runtime Error (also debug error, compiler error) – happens because you have something incorrect in the code. Usually this is caused by poor syntax or trying to do things in the code that the code doesn't allow.

Loop – a flow in code that is controlled by executing a test for a condition, and then responding by doing something until the test is passed. Examples of loops are: FOR, WHILE, IF, DO, and INFINITE

Comment – the most important part of any code. Comments are your notes to future generations about your code; why it is what it is, what it does, and how you got it there. Comment early and often for best results! In Python we use the "#" to start a comment.

Comment Out – yes there is another type of comment! This one allows you to try different things without deleting code. By "commenting out" code you can keep it in the program and try different things. Same "#" with a slightly different meaning.

Variable – much like the name implies, it is something that can have a variable definition. Variables are assigned to a value/values in your code and can be of many "types"

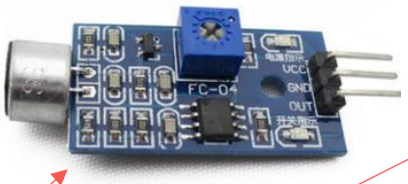
> **String** – just what it sounds like, a string is any string of text
>
> **Number** – numbers can have MANY types of values depending on the language. Python general deals in INT or integers. Our code will also use a Decimal number type

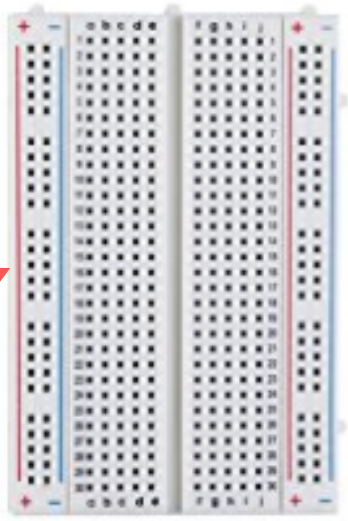
Keyword – keywords are reserved words that the language pre-defines and are key to the language itself. IF, ELSE, TRY, and IMPORT are all examples of keywords in Python

Code Block – a logical area of the code that is one part of the overall program. Blocks in Python are usually started with the "try:" keyword

USEFULL INFORMATION

What's In The Box!?





PARTS LIST:

1 x Sound sensor:

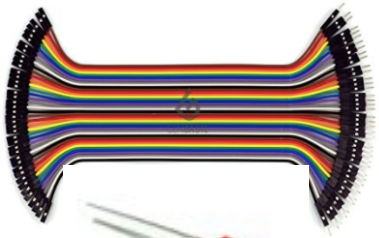
1 x Breadboard:

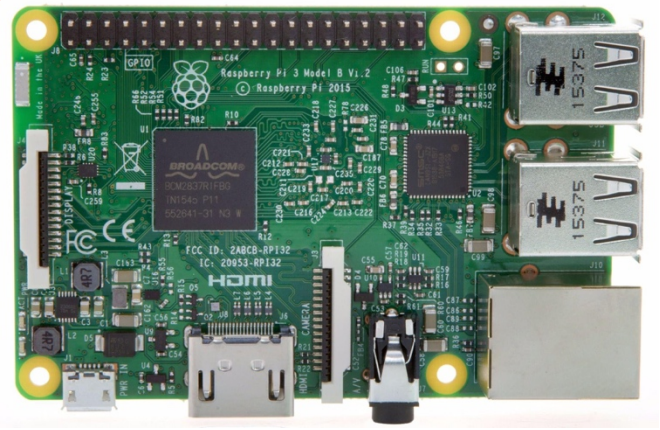
7 x Male to Female Wires:

1 x Red LED:

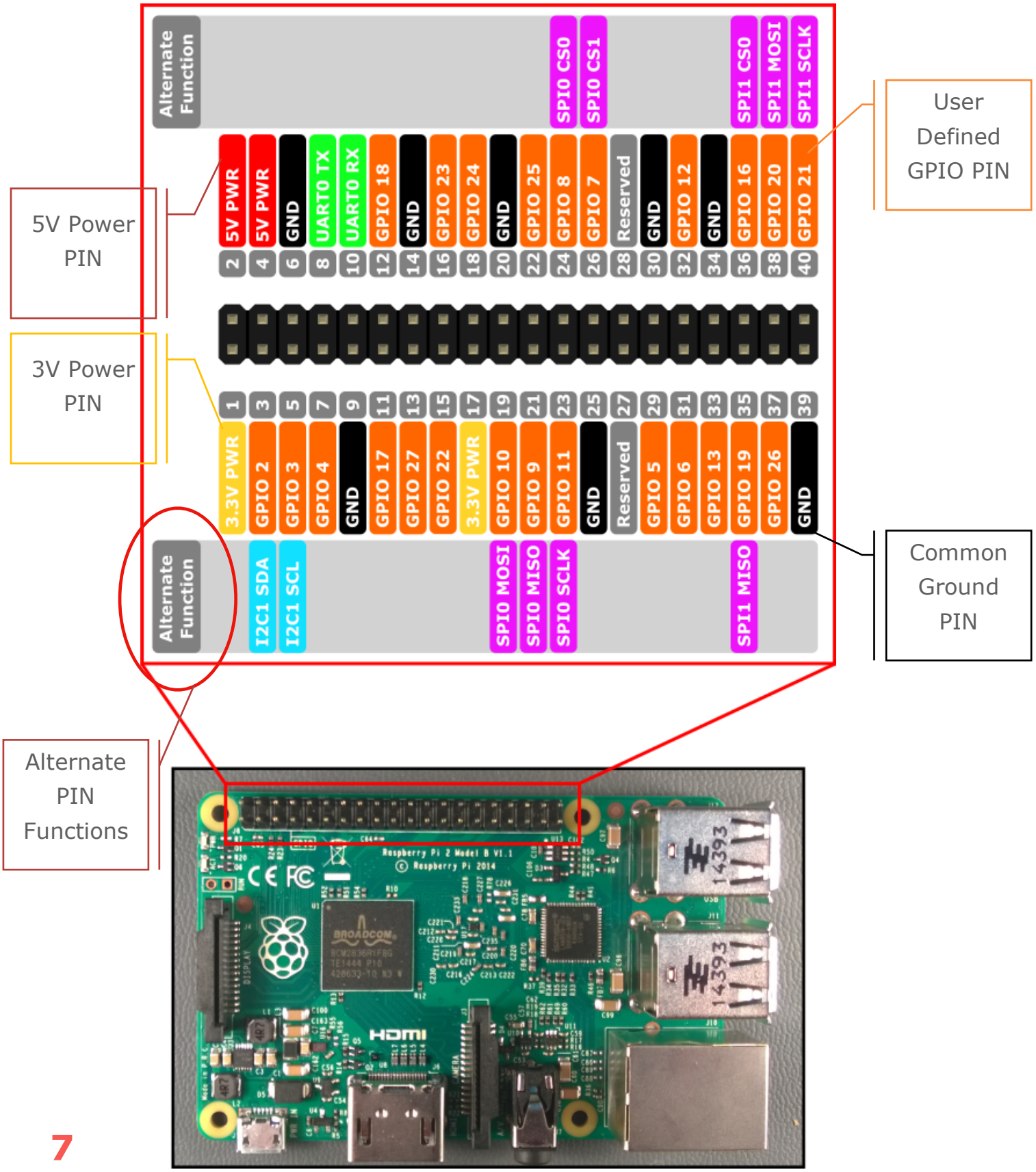
1 x Green LED:

1 x Raspberry Pi with power cord:

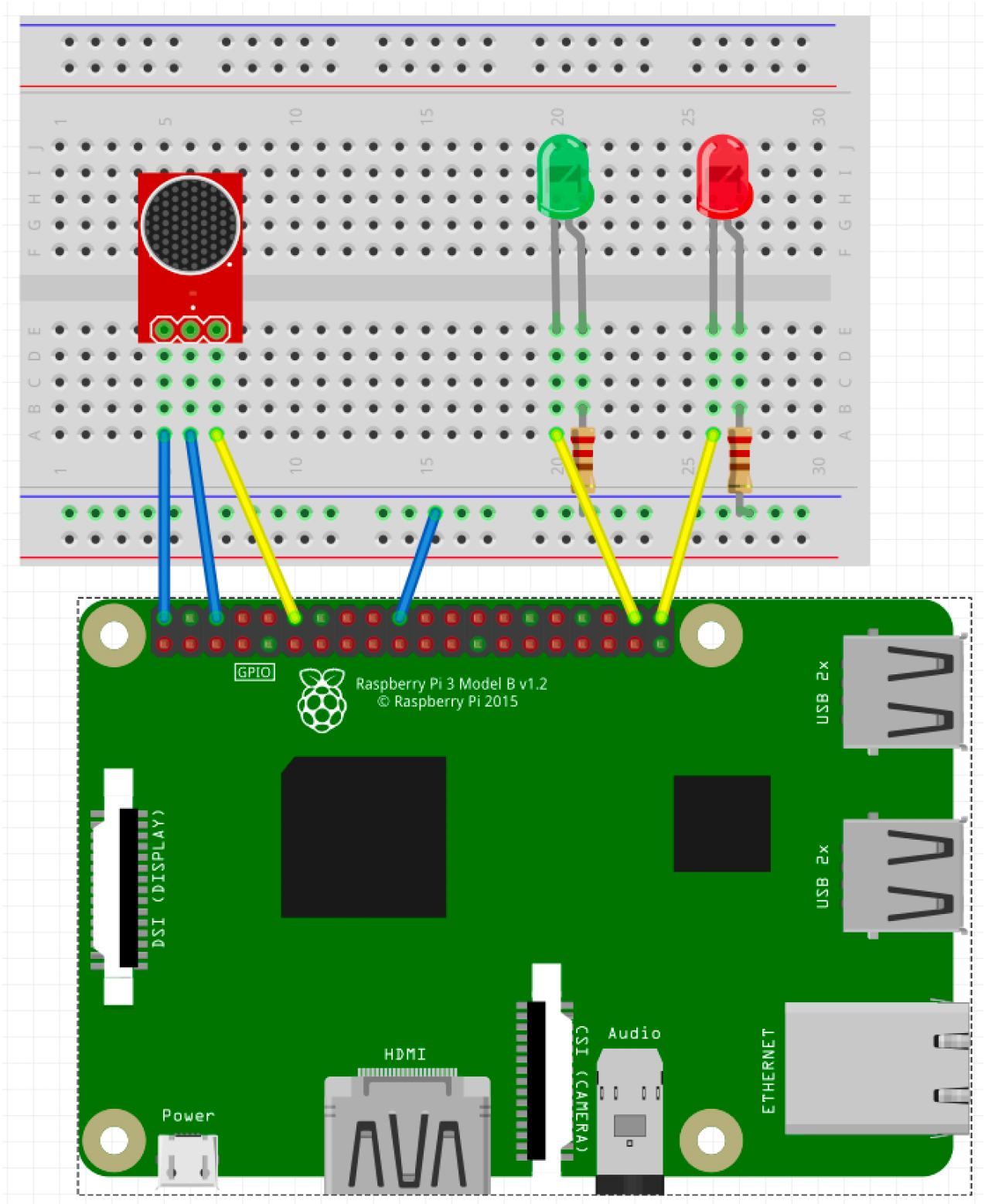


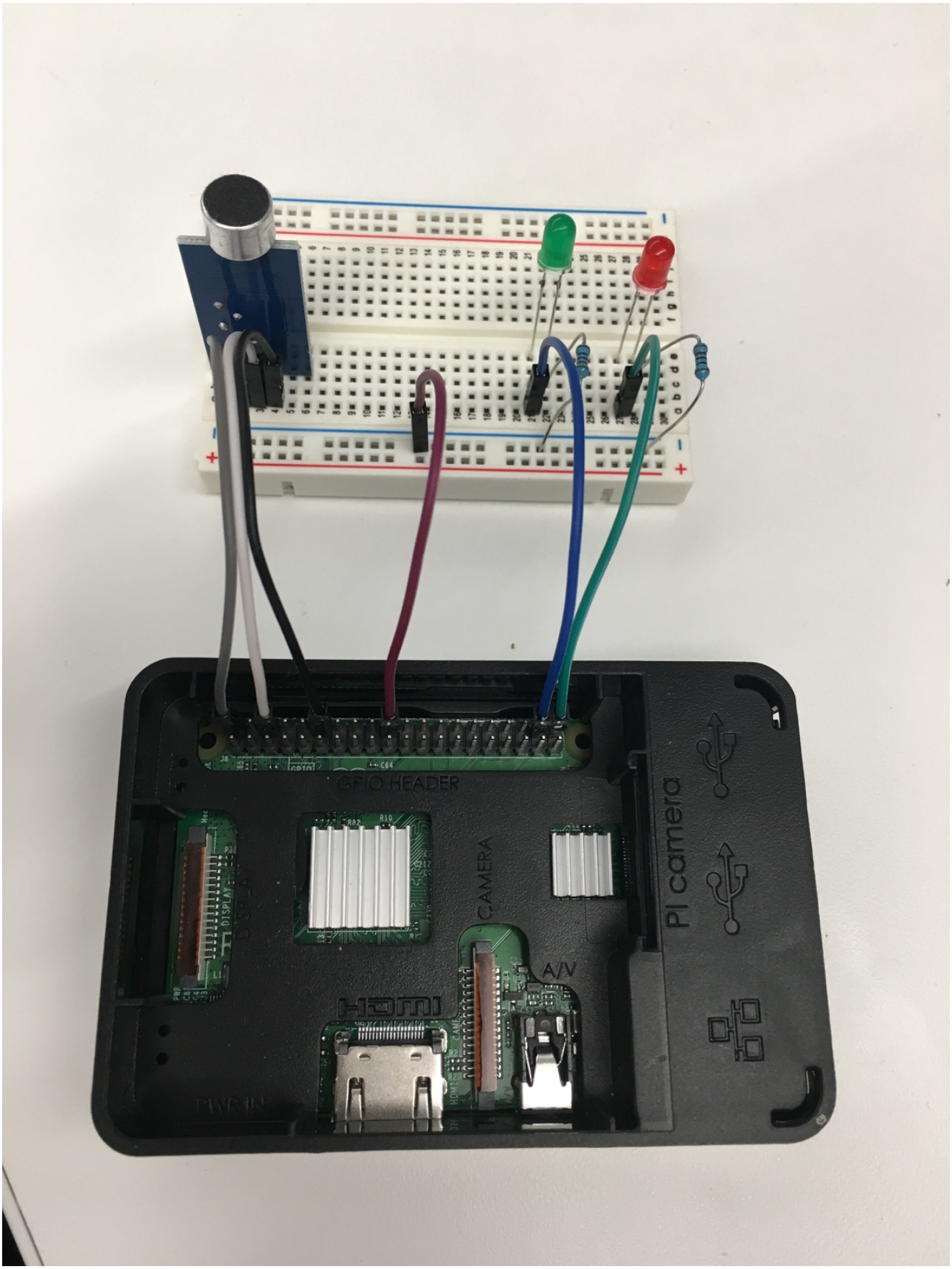


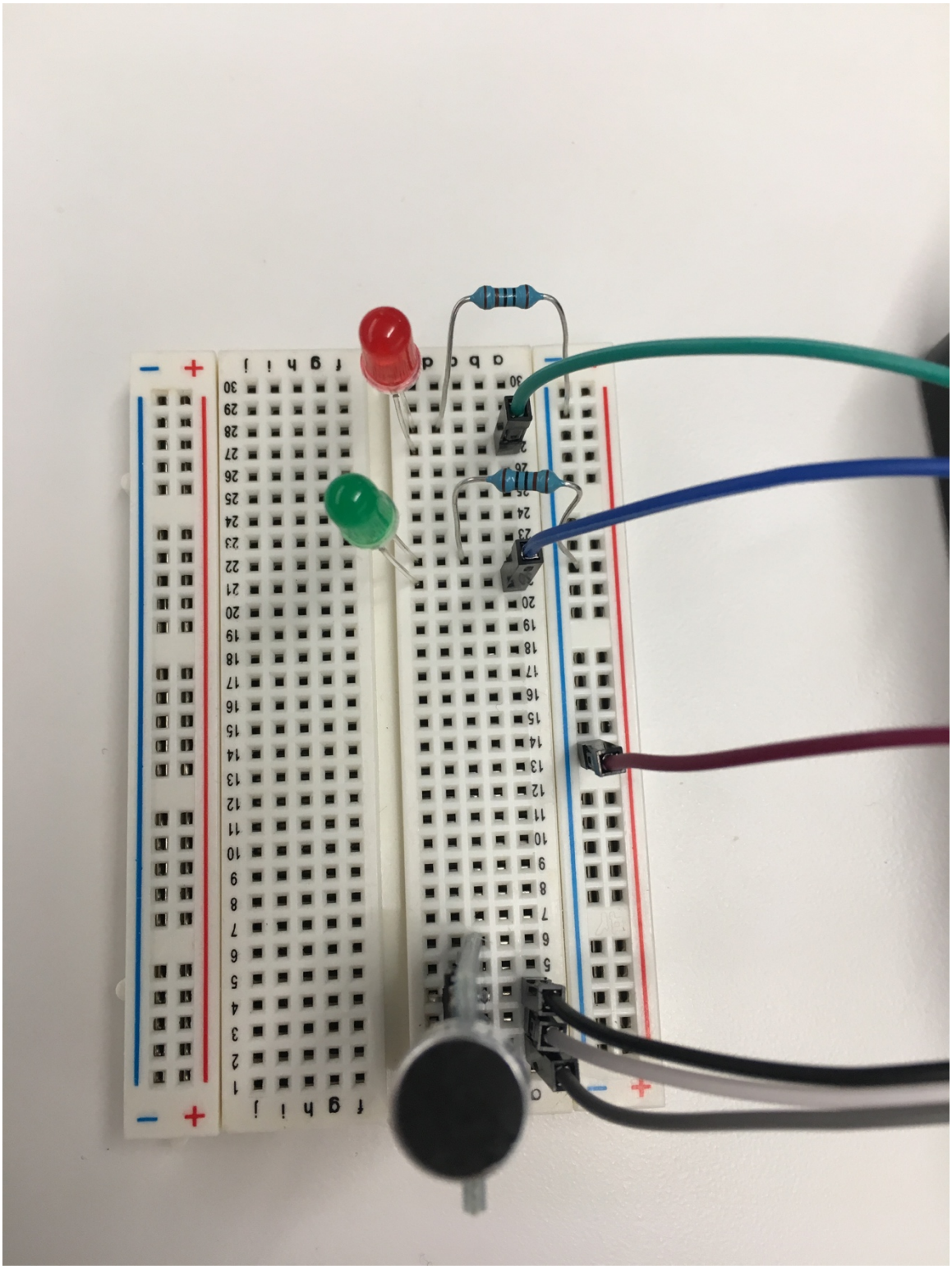
RASPBERRY PI GPIO PINOUT DIAGRAM:



WIRING PICTURES TO HELP YOU GET WIRED:




GPIO HEADER
CAMERA
DISPLAY
HDMI
A/V
PWR IN
PI camera



STEP-BY-STEP COMMAND REFERENCE

I will be walking the group through the following commands as we progress through the workshop, but this way you can get caught up if you get lost (or sprint ahead if we are going too slow). Also, this will be stored on the github repository, so you can repeat this workshop on your own without me telling you what to do!

To use these steps:

- the line in the `preformatted font` tells you the command string to type.
- After you type in the command, press ENTER or Return on your keyboard
- If you get an error, you probably just had a typo. Pressing the up arrow will repeat the command in the command prompt and you can read it and find the typo and retry the command

COMMANDS

Here are the commands we will perform, in order, with the place the command is performed:

On your laptop - either inside the putty application (windows) or in terminal (Mac):

ssh pi@YOUR.IP.GOES.HERE

(in the workshop your IP is written on the box the parts are in)

When prompted for the password, it is: raspberry

Your new prompt will be: charrold303_pixx <- where xx is a number. If you see that at the front of your command prompt you are ready to proceed to the next step.

To input our code into the pi, we will be using a built-in program called nano. I like nano better than vi or vim and that's why we are using it. Don't @ me. To enter code mode, the command is:

Sudo nano my_app.py

You will now be in the nano application and can copy the code from the next section.

THE CODE:

PLEASE NOTE! The copy-paste below has wrapped lines – you should **NOT wrap the lines** when you type them in. I tried to make the margins as big as I could, but there are still a few wraps so be aware of them (I will call them out during the walkthrough).

You can also skip the lines that start with **print** if you wish, but your application will not perform any notifications without them. This is OK if you trust it to run clean all the time. You will notice I am not the trusting sort…

You can skip the comments – lines that start with a # they are there to tell you what the actual commands do. The total of uncommented lines is around 50. We will input them together with the class, but you can skip ahead if you wish.

```
001: # ------------------------------------------------------------------------------
--
002: # Introduction
003: # ------------------------------------------------------------------------------
--
004: # Welcome to my sound sensor project! This program will do 3 basic things:
005: # 1) configure your RPi to use a sound sensor (based on the DAOKI sound sensor
from Amazon)
006: # 2) Monitor the sensor for a change in state
007: # 3) Output both a visual alert (Red LED) and update a webpage with detection
information
008: #
009: # You can grab this project from github with the following command:
010: # git clone https://github.com/ChrisHarrold/PiWorkshop.git
011: # this project is in the Sounds folder
012: # Happy Pi-ing
013:
```

```
014: # ------------------------------------------------------------------------------
--
015: # Libs, Variables, and Program Header
016: # ------------------------------------------------------------------------------
--
017:
018: # Import libraries used in this program
019: # the RPI.GPIO library is used by Python to interface with the RPi Hardware
020: import RPi.GPIO as GPIO
021: # the OS module allows us to use the file system
022: import os
023: # time is an incedibly useful python library that gives you access to commands
for time
024: import time
025: # decimal allows you to work with decimal notation for numbers - very useful for
high-precision
026: from decimal import Decimal
027: # the math library allows you to perform standard math functions
028: import math
029:
030: # Startup message - will print to the console only
031: print("Preparing to monitor sound levels")
032: print("You can gracefully exit the program by pressing ctrl-C")
033:
034: # We will be outputting to a website for a real-time view of noise detection
events.
035: # for debugging you can leave this section commented out:
036: print("Readying Web Output File")
```

```
037: # Web output file definition - this file is called by the sound.html webpage and used to
038: # display the status of the sound detection
039: web_file = "/var/www/html/table.shtml"
040:
041: # Opens and preps the HTML file for the first time. Will remove anything it
042: # finds in the file and prep it with this default entry - the replaces old
043: # data so definitely collect that info somewhere else if you want to keep it!
044: with open(web_file + '.new', 'w') as f_output:
045: f_output.write("")
046: os.rename(web_file + '.new', web_file)
047:
048:
049: # various counters used for determining the thresholds for sensitivity and detection
050: # as well as the time of the loop and frequency for debugging
051: Loud_Count = 0 # Count of trigger events from the sensor total since the program started running
052: louds_per = 0 # Count of trigger events from the sensor in this time interval
053: per_detected = 0 # The percent of loops where sound is detected versus not detected (math performed on this value later)
054: time_loop = 5 # The numeric value is how many seconds you want the timestamps to be spaced by
055: stime = time.time() # The time right now (in UNIX Datetime format - that is to say a really long string of seconds)
056: etime = stime + time_loop #etime is the end time of our loop - the difference between right now and the time_loop value
057: ptime = time.ctime() #the "pretty" version of the current time - suitable for charts and graphs!
```

```
058:
059: # loop count and max_loop are used for math on the number of detection loops within the time threshold
060: Loops_Tot = 0
061: loop_count = 0
062: max_loop = 10000000
063:
064: # This value is the number of times loud sound was detected
065: # versus the number of times the loop ran. this number will likely be
066: # very small - something on the order of .00000001
067: # You determine this number by running the program a few times and seeing the
068: # ratio of detections to loops (It is printed out later in the program)
069: a_threshold = .00000001
070:
071:
072: # ----------------------------------------------------------------------------
073: # Setup Area
074: # ----------------------------------------------------------------------------
075:
076: # Set our GPIO pin assignments to the right pins
077: sensor_in = 18
078: red_led = 21
079: green_led = 20
080: # Setup GPIO commands and pins
081: GPIO.setmode(GPIO.BCM)
```

```
082: GPIO.setup(red_led, GPIO.OUT)
083: GPIO.setup(green_led, GPIO.OUT)
084: GPIO.setup(sensor_in, GPIO.IN)
085:
086: # Make sure the pins start off in the LOW state
087: GPIO.output(green_led, GPIO.LOW)
088: GPIO.output(red_led, GPIO.LOW)
089:
090: # Then turn on the green - no noise light - and confirm system is online.
091: GPIO.output(green_led, GPIO.HIGH)
092: print("GPIO set. Service ready. Initiating Detection Protocol.")
093:
094: # Add an event detection setting to the sensor pin - when it changes state it will
trigger an alert condition
095: GPIO.add_event_detect(sensor_in, GPIO.RISING, bouncetime=300)
096:
097:
098: # ----------------------------------------------------------------------------------
--
099: # Functions Area
100: # ----------------------------------------------------------------------------------
--
101:
102: #this is our main work function - if the sensor is triggered, we will do work here
103: def dowork(sensor_in):
104:  # because Python implements loose variables - we have to make sure it knows
we are not redefining these
```

```
105:  # but reusing the global versions. This is not strictly "good code", but there is not a good alternative
106: global Loud_Count, loop_count, per_detected, max_loop, louds_per
107:
108:  # did we detect something?
109: if GPIO.input(sensor_in):
110:         # YUP! Turn on that red light!
111:         GPIO.output(red_led, GPIO.HIGH)
112:
113:         # We have NOISE! Add it to the count of Loud events
114:         Loud_Count = Loud_Count + 1
115:         louds_per = louds_per + 1
116:
117:         # Now we can see if we are detecting a lot of events or not?
118:         # By getting the ratio of events to the number of times we looked for one, we can see if it was
119:         # a spike or actually a really noisy time. This is why the "max_loop" variable matters
120:         # so you can set the a_threshold value and see if you have consistent noise or just spikes
121:         per_detected = Decimal(louds_per) / Decimal(loop_count)
122:         per_detected = round(per_detected, 10)
123:
124:         # compare the percent of detection loops to the overall threshold for loudness - that is the ratio
125:         # of non-detection loops to detection events - and then respond accordingly:
126:         if per_detected > a_threshold:
```

```
127:                print("REALLY PRETTY LOUD! Detect vs Threshold: " +
str(per_detected) + " / " + str(a_threshold))

128:                print(str(loop_count) + "loops vs " + str(louds_per) + "
events")

129:        else:

130:                print("Meh. Some noise. Detect vs Threshold: " +
str(per_detected) + " / " + str(a_threshold))

131:                print(str(loop_count) + "loops vs " + str(louds_per) + "
events")

132:

133:

134: # ----------------------------------------------------------------------------
--

135: # Main Program Area

136: # ----------------------------------------------------------------------------
--

137:

138: # try block to handle exception conditions and run the program loop

139: try:

140:  # This syntax will lock the loop into a time window (5 seconds

141:  # by default as definied by the time_loop variable)

142:  # This is extremely useful for debugging, and for setting the max_loops value

143: etime = time.time() + time_loop #etime is the end time of our loop - the
difference between right now and the time_loop value

144: while(True): #time.time() < etime:

145:

146:        # Count the number of iterations

147:        loop_count = loop_count + 1

148:        Loops_Tot = Loops_Tot + 1
```

```
149:        if GPIO.event_detected(sensor_in):
150:
151:                # Now we also need to go do our work to update our hardware and
software:
152:                dowork(sensor_in)
153:
154:                # Because we detected sounds, we need to turn on and off our event
collector to avoid
155:                # "duplicate assignment" errors
156:                GPIO.remove_event_detect(sensor_in)
157:                # an extra pause of sleep cycle to make sure everything is cleared
out before re-enabling our event detection
158:                # if you want to slow down to overall sound detection event, you can
raise this number (in seconds)
159:                time.sleep(0.25)
160:                # Turn back on the detection and start "listening" to our pin again
161:                GPIO.add_event_detect(sensor_in, GPIO.RISING,
bouncetime=300)  # lets us know when the pin is triggered
162:
163:        # Lastly for the main body, we catch our loop count when it exceeds the
end time
164:        # and reset everything to keep everything running, and our display and
math accurate:
165:        if time.time() > etime:
166:                # first we update our output to the web for display:
167:                with open(web_file, 'a') as f_output:
168:                        if louds_per > 5:
169:                                if louds_per > 10:
170:                                        f_output.write("<tr><td align=center
bgcolor=red><font color=white>On " + str(ptime) + ", it was
```

```
Loud!!</td><td align=center bgcolor=red><font color=white>" +
str(louds_per) + "</font></td></tr>")

171:                            else:

172:                                f_output.write("<tr><td align=center
bgcolor=orange><font color=white>On " + str(ptime) + ", it was a little
loud.</td><td align=center bgcolor=orange><font color=white>" +
str(louds_per) + "</font></td></tr>")

173:                        else:

174:                            f_output.write("<tr><td align=center
bgcolor=green><font color=white>On " + str(ptime) + ", it was pretty
quiet.</td><td align=center bgcolor=green><font color=white>" +
str(louds_per) + "</font></td></tr>")

175:

176:            print("Reseting Counters")

177:            loop_count = 0

178:            louds_per = 0

179:            etime = time.time() + time_loop # etime is the end time of
our loop - the difference between right now and the time_loop value

180:            ptime = time.ctime(etime) # the "pretty" version of the
current time block - suitable for charts and graphs!

181:            # turn off the RED LED since we are starting a new detection loop -
maximum time it would stay on is 5 seconds by default

182:            GPIO.output(red_led, GPIO.LOW)

183:

184:

185: except (KeyboardInterrupt, SystemExit):

186:

187:  #If the system is interrupted (ctrl-c) this will print the final values

188:  #so that you have at least some idea of what happened

189: print("----------------------------------------")
```

```
190: print(" ")
191: print("System Reset on Keyboard Command or SysExit")
192: print(" ")
193: print("Total Noises Detected: " + str(Loud_Count))
194: print(" ")
195: print("Total loops run: " + str(Loops_Tot))
196: print(" ")
197: print("----------------------------------------")
198:
199:  # Having this command is a best practice for all Raspberry Pi programs!
200:  # It ensures that the GPIO pins are all reset to defaults (off) state
201:  # when the program exits. Important so that you don't have errors on the
202:  # next program run!
203: GPIO.cleanup()
204:
205: else:
206:
207:
208:  # You can remove this entire block once you go to "production" mode
209:  # but these values are critical for the initial tuning phase.
210: print("----------------------------------------")
211: print(" ")
212: print("System Reset for some reason")
213: print(" ")
214: print("Total Noises Detected: " + str(Loud_Count))
215: print(" ")
```

216: print("Total loops run: " + str(Loops_Tot))

217: print(" ")

218: print("--")

219: GPIO.cleanup()

COMMANDS CONTINUED:

When you have finished entering your code in nano, you will need to save it with the following commands:

control-o – this will ask you to confirm the file name – press enter to save it.

control-x – this will exit the nano application

Now you can run your program! EXCITE!

sudo python3 my_app.py

At this point, one of two things will happen:

1. *It will run perfectly and work exactly as we expect*
2. *We will get an error and a line number and have to go debug our typos*

Spoiler alert – it's number 2 in almost every case.

Not to worry though, simply repeat the commands to get into and out of nano and run the application until it runs without error. I will help you debug in the workshop! If we just cannot figure it out, there is a secret shortcut. The code is already on the Pis, and I will help you use that.

If you want to do this workshop later at home, and do not want to retype the code from scratch, you can clone my git repository to your RaspberryPi with this command:

git clone https://github.com/ChrisHarrold/PiWorkshop.git

The code we are using is in the "Sounds" directory under the PiWorkshop folder.

IMPORTANT RESOURCES

Some good RPi books for your ongoing education!

Programming the Raspberry Pi, Second Edition: Getting Started with Python – by Simon Monk

Raspberry Pi Cookbook: Software and Hardware Problems and Solutions – also by Simon Monk

Raspberry Pi 3: From Noob to Master; Simple Step By Step Guide to Setting up Your Raspberry Pi 3 and Using It for a Wide Variety of Cool Projects – By Steve Ora

Quick Survey: How did I do? Was this worthwhile? Anything else?
http://bit.ly/charrold

GitHub Repository (code, this doc, other materials):
https://www.github.com/ChrisHarrold/PiWorkshop/Sounds

Pimoroni: A good source for beginner friendly hardware kits and project materials:
https://pimoroni.com

Putty: A windows-based ssh client (if you use a windows machine you will need this for the workshop and should just have it in general!):
http://bit.ly/_putty

SCP: The preeminent file copy tool for Raspberry Pi! Not really, it's a UNIX command that you will want to know for copying your code from your computer to the Pi
http://bit.ly/scp_command

GitHub: The preeminent code repository system for sharing and collaborating on code (yes, really it is)
www.Github.com

ImgFlip: Because memes are life
https://imgflip.com/memegenerator

Fritzing: Great little tool for creating your own very simple and easy to read/share/use wiring and circuit diagrams:
www.fritzing.org

My Website (updates, other projects, blog, etc.):
https://www.charrold303.com