LESSON 18

RANGE SENSING GAME

OBJECTIVE

In this lesson you will learn how to use a voltage divider to do simple signal level shifting, the absolute value function, and how to modify a file so it can be used as an import for another program. You will then move on to the final project for this course.

MATERIALS

- Raspberry Pi connected to a monitor, keyboard, and mouse
- Assembled Circuit from Lesson B-17
- 1 x Ultrasonic Range Sensor
- 2 x 1k-ohm Resistors
- 2 x Long Jumper Wires
- 5 x Short Jumper Wires

REVIEW CONCEPTS

If you do not feel comfortable with the following concepts, please review them before proceeding.

- Slide Switch (Lesson B-5)
- Voltage Dividers (Lesson B-9)
- Level Shifting Integrated Circuit (Lesson B-13)
- Ultrasonic Range Sensing (Lesson B-14)
- Running Modules as Imports vs. Directly (Lesson B-15)
- OLED Display (Lesson B-16)
- Capacitive Touch Sensors; GPIO High vs. Low (Lesson B-17)

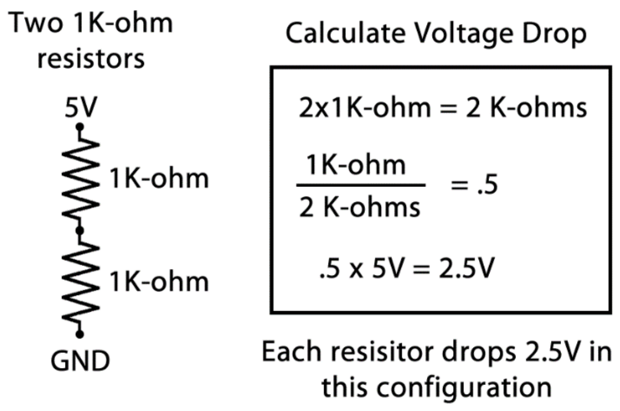
LESSON

In this lesson you will learn how to use a voltage divider to do simple signal level shifting, the absolute value function, and how to modify a file so it can be used as an import for another program. You will then move on to build the final project.

LEVEL SHIFTING WITH RESISTORS

In Lesson B-9 you learned how voltage dividers can be used to create new voltage levels. This same principle can also be used for level shifting from a higher voltage to a lower voltage.

The following voltage divider with two equal value 1K-ohm resistors will cut the supplied 5V signal in half:



The great thing about this circuit is that the 5V signal does not have to come from a constant supply. It can also be connected to the output from a 5V device, like the Ultrasonic Range Sensor, whose 5V output is not safe to connect directly to a GPIO pin. By using the output of the Ultrasonic Range Sensor to supply the input voltage, the output will be 2.5V which is safe to connect to a GPIO pin.

By watching the voltage divider output pin with the GPIO, you can determine if the 5V signal is present or not:

- 2.5V present means the Ultrasonic Range Sensor is applying 5V to the input
- 0V present means the Ultrasonic Range Sensor is <u>not</u> applying 5V to the input

You learned in Lesson B-17 that a GPIO pin will register as high for any voltage above 1.4V. So a 2.5V level will easily trigger a high in the GPIO pin:

- GPIO high means the Ultrasonic Range Sensor is applying 5V to the input
- GPIO low means the Ultrasonic Range Sensor is <u>not</u> applying 5V to the input

Using a voltage divider can be a very useful way to quickly reduce the voltage from a sensor, using only two resistors. When using several 5V sensors, it makes more sense to use a device like the 74LVC245 (Level Shifting) IC instead, due the amount and complexity of resistors that would be required to make a voltage divider for each channel.

ABSOLUTE VALUE IN PYTHON

An absolute value represents how far a value is away from zero. It essentially removes the positive or negative sign of a value, so the absolute value of -5 is 5. The same goes for the positive value. The absolute value of 5 is still just 5. The absolute value function in Python is:

```
abs()
```

The absolute value of will be taken of anything inside the parentheses:

`abs(-23)` becomes **`23`**

`abs(42)` becomes **`42`**

The second example doesn't seem to be very useful, but what if you don’t know what the value in the parentheses will be until your program starts running:

```
abs(x-y)
```

If x is greater than **`y`** then the **`abs()`** function will have no effect on the value that this function outputs. However, if **`y`** is greater than **`x`**, the result of **`x-y`** will be negative and **`abs()`** will strip off the sign from the negative number.

In the upcoming activity, you will build a game that will be taking the difference of two numbers, without knowing which will be larger. The **`abs()`** function will be used to strip the sign from the result, leaving only the positive value of the difference between the two values.

MODIFYING A FILE FOR IMPORT USE

When you import a file, that file is completely run from beginning to end. Any imports, variable assignments, function definitions, or anything else in that program, will run as if those lines were included in your program.

What if a file that you choose to import includes the following:

```
def thing1():
    x = 1

while True:
    print('Hello World')
```

On import, the imported file would define the function named **`thing1()`**, and then get stuck in the **`while True:`** loop, never returning to your program. This is obviously not ideal, and this is why back in Lesson B-15, you learned about the **`__name__`** variable that can be used to determine if a file has been imported, or has been executed directly:

```
if __name__=="__main__":
```

Any code indented below this **`if`** condition will only run when the file was run directly and will be ignored if the file is being executed as an import to another file. You can modify the earlier example to allow for both imported and direct execution:

```
def thing1():
    x = 1

if __name__=="__main__":
    while True:
        print('Hello World')
```

Importing the file will now result in the code inside the if block being completely ignored, and when the program is executed directly, it will run from top to bottom, including everything contained in the **`if`** block.

In the following activities, the ultrasonic range sensing program that you created in Lesson B-14 will be modified so its function definitions can be used for import, without executing the main program it contains.

ACTIVITIES

In the following activities you will modify the **`ultrasonic.py`** program you created in Lesson B-14, add the Ultrasonic Range Sensor to the circuit you built in Lesson B-17, and create a game program that uses the Ultrasonic Range Sensor, the OLED screen, the Capacitive Touch Sensor, the RGB LED, and the slide switch.

ACTIVITY #1 – MODIFYING ULTRASONIC.PY

In this activity, you will modify the **`ultrasonic.py`** program that you created in Lesson B-14 so its range sensing functions can be imported without running the main loop program that it contains.

STEP #1

The first step is to open **`ultrasonic.py`** from your Desktop in Thonny. Once open, scroll down to the **`try:`** loop. You want to enclose this entire **`try:`** loop inside of an **`if __name__`** condition to ensure that it does not run when the file is used as an import. Add the following line above the **`try:`** block and add another level of indentation to the **`try:`** block as well as everything below, including the **`except:`** block:

```
if __name__=="__main__":
    try:
        while True:
            distance = average()
            print('%.1f' % distance)
            if distance < 20:
                GPIO.output(red, GPIO.HIGH)
                GPIO.output(green, GPIO.LOW)
                GPIO.output(blue, GPIO.LOW)
            elif 20 <= distance < 25:
                GPIO.output(red, GPIO.LOW)
                GPIO.output(green, GPIO.HIGH)
                GPIO.output(blue, GPIO.LOW)
            else:
                GPIO.output(red, GPIO.LOW)
                GPIO.output(green, GPIO.LOW)
                GPIO.output(blue, GPIO.HIGH)
            time.sleep(.25)

    except KeyboardInterrupt:
        GPIO.cleanup()
```

All of the gray boxes above are additional spaces that were added to realign the code below the new **`if`** block.

STEP #2

Save your updated file so you can use it as an import later when building the game program in Activity #3.

ACTIVITY #2 – ADDING THE ULTRASONIC RANGE SENSOR

In this activity, you will add the ultrasonic range sensor to the capacitive touch circuit you built in Lesson B-17, Activity #3.

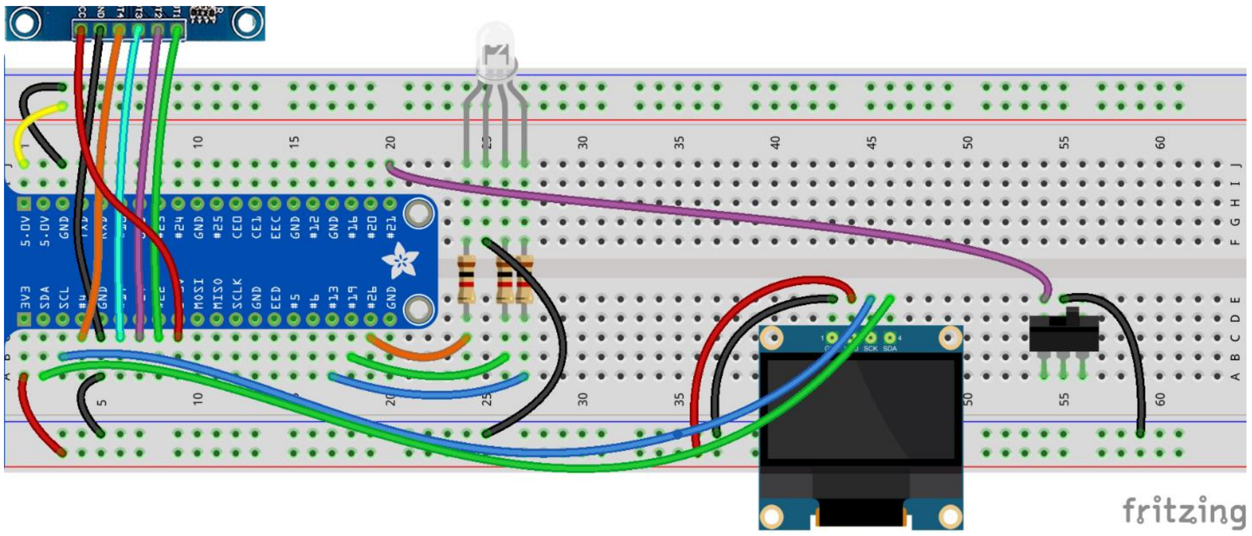
STEP #1

Shut down the Pi and disconnect power before proceeding.

Once that's done, the first circuit modification will be to get 5V power and ground over to the P2/N2 power rails so it can be used to power the Ultrasonic Range Sensor. Make the following two connections using short jumper wires:

Short jumper wire – 5V – between J1 and P2-3

Short jumper wire – GND – between J3 and N2-3



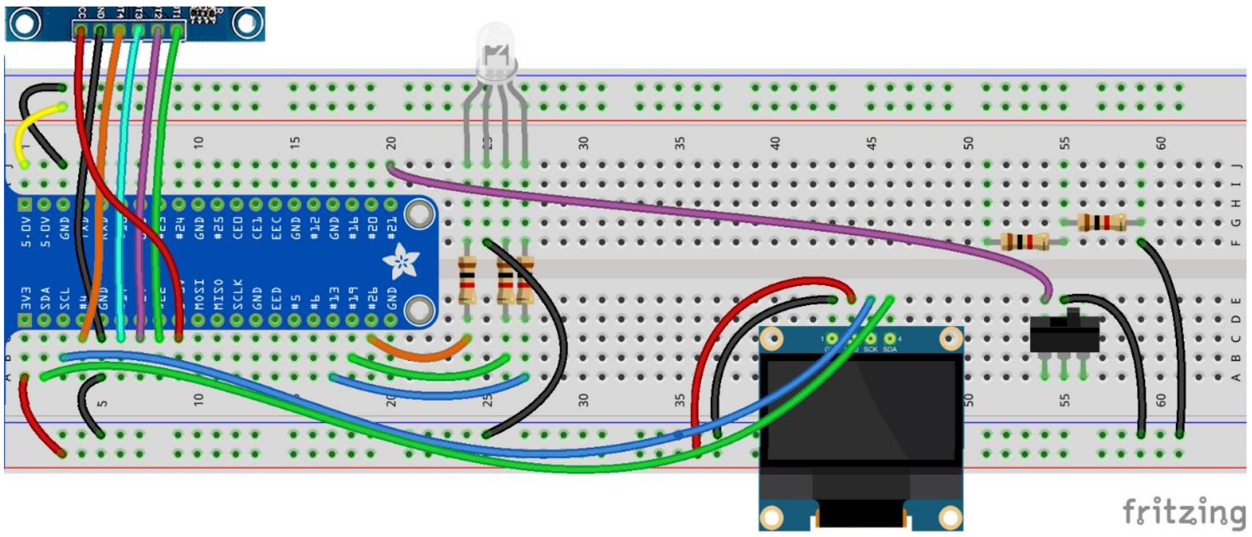
STEP #2

Next, add a voltage divider made of two 1K-Ohm resistors that will be used to level-shift the Echo output of the ultrasonic range sensor. Add two resistors and a short jumper wire between the points below:

1K-Ohm resistor – between G55 and G59

1K-Ohm resistor – between F51 and F55

Short jumper wire –GND - between F59 and N2-61



STEP #3

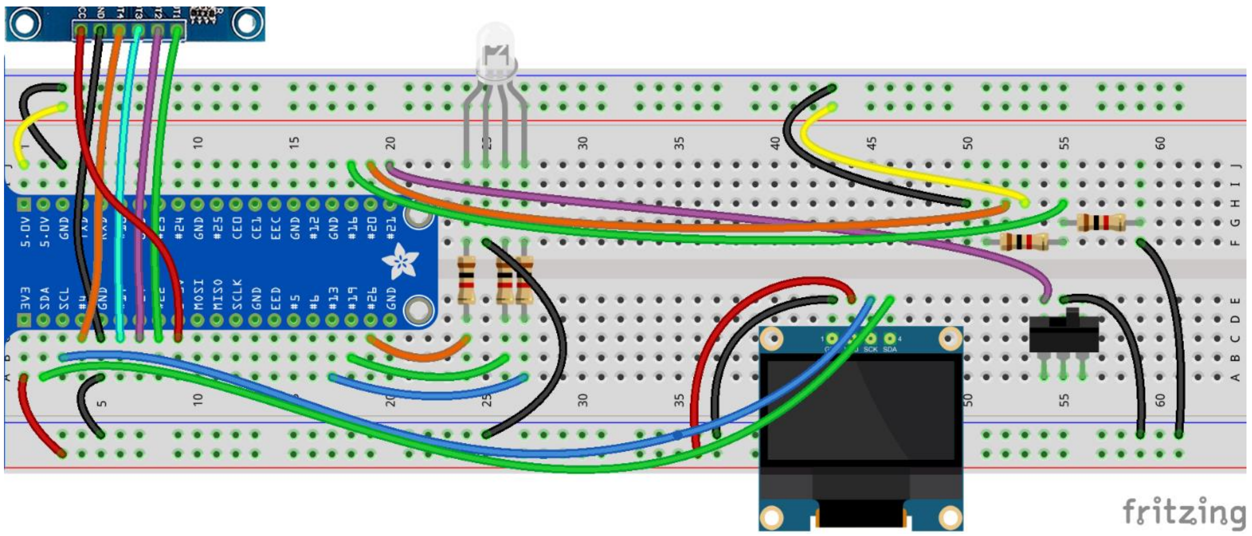
The voltage divider will take care of bringing the Echo line from the ultrasonic range sensor down to a safe level, but you will need to make a few more connections before you can use it for ranges. Make the following four connections between the points listed below:

Short jumper wire – 5V – between H53 and P2-43

Short jumper wire - GND – between H50 and N2-42

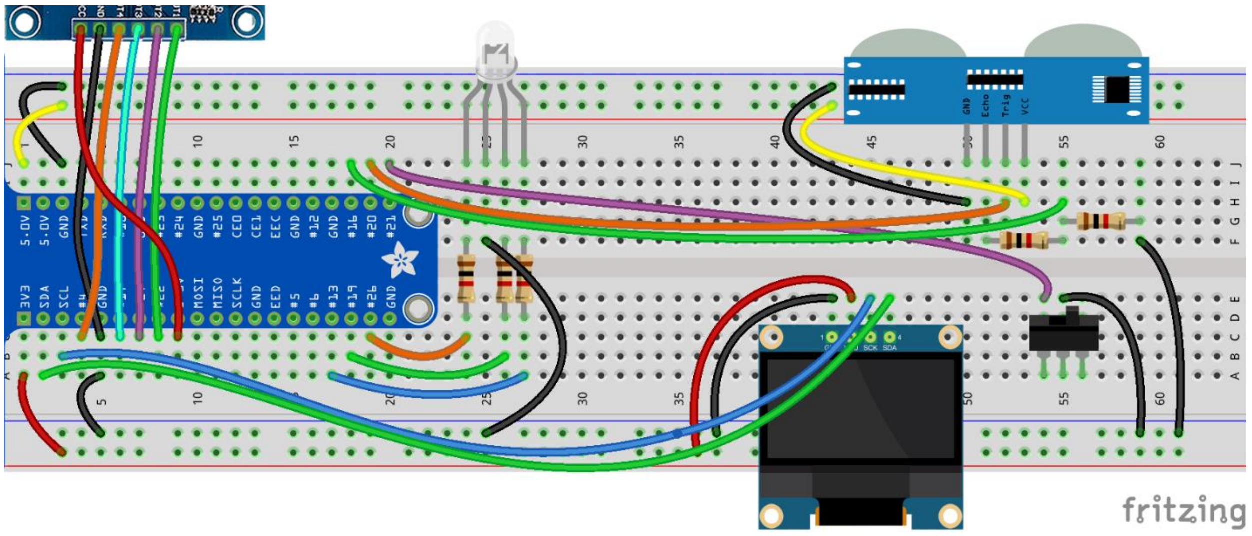
Long jumper wire – Trigger – between H52 and J19

Long jumper wire – Echo – between H55 and J18



STEP #4

The last step is to add the ultrasonic range sensor. Connect it to the breadboard at J50 through J53 with the sensor pointing away from the OLED display. Ensure the sensor is properly oriented and connected to the correct locations, or it could be damaged:



Power the Raspberry Pi on so it can be used to create a program to use with your new circuit.

ACTIVITY #3 – BUILDING THE RANGE GAME

In this activity, you will build a program to create the game outlined below:

The game will ask you to place your hand or another obstacle at a pre-determined distance from the Ultrasonic Range Sensor. The game will consist of the following actions:

- The user selects the desired difficulty level using the slide switch. Easy mode allows for 20cm of error while Hard mode only allows for 10cm. The program waits 2 seconds before the position of the slide switch determines which difficulty level will be used for that round.
- The screen displays a random target distance between 20cm and 100cm.
- The user is asked to press capacitive touch pad 1-4 to start the distance capture process. Each pad represents the number of seconds before the capture occurs, which can be used to add more difficulty. Less time to get ready before the capture makes it more difficult.
- The user range is captured and compared to the target distance. The RGB LED will turn green if the user distance was within 20cm in Easy mode, or 10cm in Hard mode. Errors above these amounts will result in the RGB LED turning red.
- The program loops back to the beginning.

This program will be the largest you've written yet, but it will borrow large blocks from programs in previous activities. So, don’t be intimidated! Build the program one block at a time, just like you would with a shorter program.

STEP #1

The first step will be to open Thonny and create a new program called **`range_game.py`**. Save the file to your Desktop so it will have access to the **`ultrasonic.py`** file that you modified in Activity #1.

As you go through the steps below, save your program often to avoid losing an of your work.

STEP #2

The first area in the program will be the imports and there are quite a few. You will be using **time**, **RPi.GPIO**, **random**, **ultrasonic**, **Adafruit_SSD1306**, and the **PIL** imports you used for the OLED display. Here are the import lines to add:

```
import time, RPi.GPIO as GPIO, random, ultrasonic, Adafruit_SSD1306
from PIL import Image, ImageDraw, ImageFont
```

STEP #3

Next, you will assign the input and output pins using a lot of variables.

This will help you later if you want to modify this program and move an input or output to another pin. This list will contain pin assignments for the slide switch, Trigger and Echo lines, RGB elements, and Capacitive Touch inputs.

The lists for **rgb** and **cap** will allow those groups of pins to be configured as inputs and outputs as a group, using only one line each:

```
import time, RPi.GPIO as GPIO, random, ultrasonic, Adafruit_SSD1306
from PIL import Image, ImageDraw, ImageFont

slide = 21
trig = 20
echo = 16
rgb = [13,19,26]
red = 13
green = 19
blue = 26
cap = [22,27,17,4]
cap1 = 22
cap2 = 27
cap3 = 17
cap4 = 4
```

STEP #4

The next step will be the GPIO configuration. Here are the configuration steps that need to be completed:

- Set GPIO pin mode to **`BCM`**
- Configure **`slide`** as an input with pull-up
- Configure **`cap`** as an input (this will take care of all four touchpads)
- Configure **`rgb`** as an output (this will take care of all three RGB elements)
- Set the output of **`rgb`** to low or 0 (this will ensure the RGB LED is off initially in case your program has errors and exits improperly)

Here is the code to accomplish these five tasks:

```
cap2 = 27
cap3 = 17
cap4 = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(slide, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(rgb, GPIO.OUT)
GPIO.setup(cap, GPIO.IN)
GPIO.output(rgb, GPIO.LOW)
```

STEP #5

There is one more configuration step that must be accomplished but it can't be included with the GPIO configuration. It's the code to configure all the settings required for your OLED display to operate.

This block of code is pulled directly from the program you created in Lesson B-16, Activity #2, and each line of code is broken down in the section titled SSD1306 Display Driver. If you're unsure about anything below, please refer to that section for more information:

```
GPIO.setup(cap, GPIO.IN)
GPIO.output(rgb, GPIO.LOW)

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
width = disp.width
height = disp.height
image = Image.new('1', (width, height)) # '1' converts image to 1-bit color
draw = ImageDraw.Draw(image)
font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',18)
```

STEP #6

There are a couple lines of code that will be used every time the display needs to be updated. They are **`disp.image(image)`** and **`disp.display()`**.

Instead of using these two lines every time you need to update the display, create a function named **`update_display()`** that can be called every time an update is needed:

```
draw = ImageDraw.Draw(image)
font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',18)

def update_display():
    disp.image(image)
    disp.display()
```

You are now just over 30 lines into the program. Ensure your program matches the program on the next page before continuing to the next step.

```
import time, RPi.GPIO as GPIO, random, ultrasonic, Adafruit_SSD1306
from PIL import Image, ImageDraw, ImageFont

slide = 21
trig = 20
echo = 16
rgb = [13,19,26]
red = 13
green = 19
blue = 26
cap = [22,27,17,4]
cap1 = 22
cap2 = 27
cap3 = 17
cap4 = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(slide, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(rgb, GPIO.OUT)
GPIO.setup(cap, GPIO.IN)

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
width = disp.width
height = disp.height
image = Image.new('1', (width, height))
draw = ImageDraw.Draw(image)
font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',18)

def update_display():
    disp.image(image)
    disp.display()
```

STEP #7

Now that all the configuration steps have been completed, it's time to start building the main program loop. Start with a **`try:`** loop that contains a **`while True:`** loop, and an **`except KeyboardInterrupt:`** that contains the following:

disp.clear()	clears the image buffer
disp.display()	pushes the image buffer to display to so it's clear before shutdown
GPIO.cleanup()	resets all GPIO pins to their default state

The code to add these elements is listed below:

```
def update_display():
    disp.image(image)
    disp.display()

try:
    while True:

except KeyboardInterrupt:
    disp.clear()
    disp.display()
    GPIO.cleanup()
```

Any new code in upcoming steps will be inserted in the **`while True:`** loop.

STEP #8

The first step in the main program loop is to display the difficulty level based on input from the slide switch. You will control the timing of this portion of the program by looping 20 times with a 0.1 delay in each loop, for a total time of 2 seconds allowed for difficulty selection.

Inside this **`skill_selection`** loop, you will nest two **`if/else`** conditions:

If the slide switch is low or **`False`**, then draw a rectangle to clear the display, draw the strings **`"Difficulty?"`** and **`"Easy"`** to two lines of the image buffer, push the image buffer to the screen, and set a variable named **`skill`** equal to **`2`**. This variable will be used later to determine the size of the error window that will result in a green LED.

If the slide switch is not low, the **`else:`** block will execute, just like the block above but **`"Hard"`** will be printed to the second line of the display, and skill will be **`set`** equal to **`1`**.

Here is the code to accomplish the program functions outlined above:

```
try:
    while True:
        for skill_selection in range(0,20):
            if GPIO.input(slide) == False:
                draw.rectangle((0,0,width,height), outline=0, fill = 0)
                draw.text((0, 0), "Difficulty?",  font=font, fill=255)
                draw.text((0, 22), "Easy",  font=font, fill=255)
                update_display()
                skill = 2
            else:
                draw.rectangle((0,0,width,height), outline=0, fill = 0)
                draw.text((0, 0), "Difficulty?",  font=font, fill=255)
                draw.text((0, 22), "Hard",  font=font, fill=255)
                update_display()
                skill = 1
            time.sleep(.1)
```

Make sure your indentation matches the code above. This is crucial for each block of the program to operate as expected.

STEP #9

Now that the difficulty has been selected, the random target number can be selected and displayed on the screen to let the player know the target distance for this round. First, set a variable named **`target`** to a random integer between 20 and 100 by using the **`randon.randint()`** function.

Next, show the target distance on the display by using % notation to print **`'Target = %s" %target`** on the first line of the display. The second and third lines of the display should read **`"Press pad"`** and **`"to start"`**. An **`update_display()`** will be used to send this new information to the display:

```
                update_display()
                skill = 1
            time.sleep(.1)

        target = random.randint(20,100)

        draw.rectangle((0,0,width,height), outline=0, fill=0)
        draw.text((0, 0), "Target = %s" %target,  font=font, fill=255)
        draw.text((0, 22), "Press pad",  font=font, fill=255)
        draw.text((0, 44), "to start",  font=font, fill=255)
        update_display()
```

Be careful again with the indentation on this section of the program. It should be at the same indentation level as the **`for:`** loop. You want it to run after, and not as part of the **`for:`** loop.

STEP #10

The user has now been informed of the target distance and has been prompted to press a touchpad to start the capture process. Now hold the program in a **`while:`** loop until a touchpad is pressed and goes high.

This can be done by using a **`while:`** loop that requires **`cap1`**, **`cap2`**, **`cap3`**, and **`cap4`** all to be **`False`** to keep the loop running. As soon as any of the GPIO pins connected to those pads goes high, the loop will exit and continue with the rest of the program. A **`time.sleep`** of **`0.05`** will be added inside the loop to keep the program from using too many resources while waiting for input from a touchpad:

```
        draw.text((0, 44), "to start",  font=font, fill=255)
        update_display()

        while GPIO.input(cap1) == GPIO.input(cap2) == GPIO.input(cap3) ==
GPIO.input(cap4) == False:
            time.sleep(0.05)
```

Even though the **`while:`** condition didn't fit on one line in this document, it should be one continuous line in your code from **`while`** all the way to **`False`**:. The **`time.sleep(0.05)`** should be indented so it runs each time the **`while`** condition is met.

STEP #11

If the **`while:`** loop stops running, that means one of the capacitive touch pads had been triggered, but you don't know which one.

Create some **`if`** conditions following the **`while:`** loop that will check to see if each pad is high or **`True`**, and assign a variable named **`delay`** equal to that pad's number. If **`cap1`** was pressed then **`delay = 1`**, and if **`cap4`** was pressed then **`delay = 4`**, etc.

```
            time.sleep(.05)

        if GPIO.input(cap1) == True:
            delay = 1
        if GPIO.input(cap2) == True:
            delay = 2
        if GPIO.input(cap3) == True:
            delay = 3
        if GPIO.input(cap4) == True:
            delay = 4
```

Since the **`while:`** loop exited you know that one of the pads was pressed. This code will quickly check each pad and assign the value of delay based on which pad was pressed.

STEP #12

Now that you have the selected delay time stored as a variable, you can display a message letting the user know the target distance, and a message about when the range will be captured. There is, however, a small problem with this plan. The string "**`Capturing in X seconds`**" works for selections **`2`**, **`3`**, and **`4`**, but not for **`1`**. Since you don't want to display "1 seconds", the second line will have to be customized based on the value of **`delay`** to maintain correct grammar.

The first two lines of the message will be very similar to the **`Press pad to start`** block of code form Step #9. The first line will display the target distance and the second will display **`"Capturing range"`**. The third line of the display will need to be customized using an **`if/else`** block.

If delay is 1 then the third line should be **`"in %i second" %delay`** to maintain proper grammar for the single second. If **`delay`** is anything else, then the third line should be **`"in %i seconds" %delay`** to properly display multiple seconds.

At the end of this block you will update the display and insert a delay equal to the value of the **`delay`** variable by using **`time.sleep(delay)`**:

```
        if GPIO.input(cap4) == True:
            delay = 4

        draw.rectangle((0,0,width,height), outline=0, fill=0)
        draw.text((0, 0), "Target = %s" %target,  font=font, fill=255)
        draw.text((0, 22), "Capturing range",  font=font, fill=255)
        if delay == 1:
            draw.text((0, 44), "in %i second" %delay,  font=font, fill=255)
        else:
            draw.text((0, 44), "in %i seconds" %delay,  font=font, fill=255)
        update_display()
        time.sleep(delay)
```

STEP #13

The delay is over, and you are now ready to capture the range to the user. You can do this by accessing the **`average()`** function inside your **`ultrasonic.py`** file, and setting the result equal to a variable named **`distance`**.

Next, you will create a variable named **`diff`** that will hold the absolute value of the random **`target`** variable minus the **`distance`** returned from the **`average()`** function. This will be expressed as **`diff = abs(target-distance)`**.

Now you will calculate the range of values that will get a green LED based on the difficulty that's been selected. Create a variable named **`window`** that equals **`10 * skill`**. This means:

Easy mode, **`skill = 2`** so **`window`** will equal 10 X 2 or **`20`**

Hard mode, **`skill = 1`** so **`window`** will equal 10 X 1 or **`10`**

Add these program tasks with the following lines of code:

```
        update_display()
        time.sleep(delay)

        distance = ultrasonic.average()
        diff = abs(target-distance)
        window = 10 * skill
```

STEP #14

It's now time to light the LED using the **`diff`** and **`window`** variables. If **`diff`** is smaller than or equal to **`window`**, the user distance is good, so the LED should turn green, If not, the user distance that round was larger than the window, so the LED should turn red. This can be accomplished by using a simple **`if/else`** block:

```
        distance = ultrasonic.average()
        diff = abs(target-distance)
            window = 10 * skill

        if diff <= window:
            GPIO.output(green, GPIO.HIGH)
        else:
            GPIO.output(red, GPIO.HIGH)
```

STEP #15

The program is almost done.

Finally, you will display the target and player distances, along with the difference between the two, add a 5 second delay so there is time to view the results, and turn off the LED so it's ready for the next round.

The display block is exactly like other blocks above it:

- Blank the image buffer with a black rectangle
- Line 1 will display "Target =" along with the value of **`target`**
- Line 2 will display "Player =" along with the value of **`distance`**
- Line 3 will display "Diff =" along with the value of **`diff`**
- Update the display

You will use **`time.sleep(5)`** for the delay and the **`rgb`** list to turn off all LED elements:

```
        else:
            GPIO.output(red, GPIO.HIGH)

        draw.rectangle((0,0,width,height), outline=0, fill=0)
        draw.text((0, 0),  "Target = %s" %target,  font=font, fill=255)
        draw.text((0, 22),  "Player = %.0f" %distance,  font=font, fill=255)
        draw.text((0,44),   "Diff = %.0f" %diff,  font=font, fill=255)
        update_display()
        time.sleep(5)
        GPIO.output(rgb, GPIO.LOW)
```

STEP #16

Double check your program and indentation against the code below:

```
import time, RPi.GPIO as GPIO, random, ultrasonic, Adafruit_SSD1306
from PIL import Image, ImageDraw, ImageFont

slide = 21
trig = 20
echo = 16
rgb = [13,19,26]
red = 13
green = 19
blue = 26
cap = [22,27,17,4]
cap1 = 22
cap2 = 27
cap3 = 17
cap4 = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(slide, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(rgb, GPIO.OUT)
GPIO.setup(cap, GPIO.IN)

disp = Adafruit_SSD1306.SSD1306_128_64(rst=None)
disp.begin()
width = disp.width
height = disp.height
image = Image.new('1', (width, height)) # '1' converts image to 1-bit color
draw = ImageDraw.Draw(image)
font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeSans.ttf',18)

def update_display():
    disp.image(image)
    disp.display()

try:
    while True:
        for skill_selection in range(0,20):
            if GPIO.input(slide) == False:
                draw.rectangle((0,0,width,height), outline=0, fill = 0)
                draw.text((0, 0), "Difficulty?",  font=font, fill=255)
                draw.text((0, 22), "Easy",  font=font, fill=255)
                update_display()
                skill = 2
            else:
                draw.rectangle((0,0,width,height), outline=0, fill = 0)
                draw.text((0, 0), "Difficulty?",  font=font, fill=255)
                draw.text((0, 22), "Hard",  font=font, fill=255)
                update_display()
                skill = 1
            time.sleep(.1)
        target = random.randint(20,100)

        draw.rectangle((0,0,width,height), outline=0, fill=0)
        draw.text((0, 0), "Target = %s" %target,  font=font, fill=255)
        draw.text((0, 22), "Press pad",  font=font, fill=255)
        draw.text((0, 44), "to start",  font=font, fill=255)
        update_display()


        while GPIO.input(cap1)==GPIO.input(cap2)==GPIO.input(cap3)==GPIO.input(cap4))==False:
            time.sleep(.05)

        if GPIO.input(cap1) == True:
            delay = 1
        if GPIO.input(cap2) == True:
            delay = 2
        if GPIO.input(cap3) == True:
            delay = 3
        if GPIO.input(cap4) == True:
            delay = 4

        draw.rectangle((0,0,width,height), outline=0, fill=0)
        draw.text((0, 0), "Target = %s" %target,  font=font, fill=255)
        draw.text((0, 22), "Capturing range",  font=font, fill=255)
        if delay == 1:
            draw.text((0, 44), "in %i second" %delay,  font=font, fill=255)
        else:
            draw.text((0, 44), "in %i seconds" %delay,  font=font, fill=255)
        update_display()
        time.sleep(delay)

        distance = ultrasonic.average()
        diff = abs(target-distance)
        window = 10 * skill

        if diff <= window:
            GPIO.output(green, GPIO.HIGH)
        else:
            GPIO.output(red, GPIO.HIGH)

        draw.rectangle((0,0,width,height), outline=0, fill=0)
        draw.text((0, 0),  "Target = %s" %target,  font=font, fill=255)
        draw.text((0, 22),  "Player = %.0f" %distance,  font=font, fill=255)
        draw.text((0,44),   "Diff = %.0f" %diff,  font=font, fill=255)
        update_display()
        time.sleep(5)
        GPIO.output(rgb, GPIO.LOW)

except KeyboardInterrupt:
    disp.clear()
    disp.display()
    GPIO.cleanup()
```

STEP #17

Now that your program is complete, run the program. Select the difficulty by using the slide switch and start the capture process by pressing one of the Capacitive Touch pads.

If your program is not working as expected, identify which area of the program needs to be checked. Each block of code is performing a very specific function, so identify what part of the game is not working properly and check out the block of code that is controlling that behavior.

If you still can't get your program to work, you can download a copy of this program from the Level B Resource Page.

QUESTIONS FOR UNDERSTANDING

1. Does signal level-shifting require an IC or can it be done with two resistors?

2. Can every file be used an import without causing any problems in your main program?

3. What function can be used to take the absolute value of a variable or expression?

Answers can be found on the next page.

ANSWERS TO THE QUESTIONS FOR UNDERSTANDING

1. *Does signal level-shifting require an IC or can it be done with two resistors?*

   ANSWER: Signal level shifting can be done with two resistors but if multiple 5V devices are being used, using an IC for level shifting is recommended.

2. *Can every file be used an import without causing any problems in your main program?*

   ANSWER: No. Everything in the imported file will run on import unless it's inside an **`if __name__=="__main__":`** condition. Loops or other types of code in the imported file could cause problems when imported and must be enclosed in this **`if:`** condition to isolate it during your import.

3. *What function can be used to take the absolute value of a variable or expression?*

   ANSWER: The **`abs()`** function will return the absolute value of a variable or expression.

CONCLUSION

Congratulations! You have completed Level B of this course. You have learned a great number of new skills and you should be proud of yourself!

What's next?

- Order a copy of Level C of this course to continue to learn to work with more electronic components and coding commands (available Spring 2019).

- The skills you have gained and the components you have amassed working with Level A and Level B, have put you in a great position to tackle beginner and intermediate projects online. We recommend searching for projects that use both the Raspberry Pi and Python. You are likely to find that you have worked with much of the code and components that will be called for in the projects, and those you haven't, you likely have the skills to figure out.

Intro to Robotics Course of Study:

Level A: Building Circuits and Beginning Programming

Level B: Working with Sensors and Intermediate Programming

Level C: Audiovisual and Advanced Programming

NEXT STEP

Level D: Working with Motors and Taking It Mobile