GSOC Raspberry Pi - LED Lights - PROJECT #1

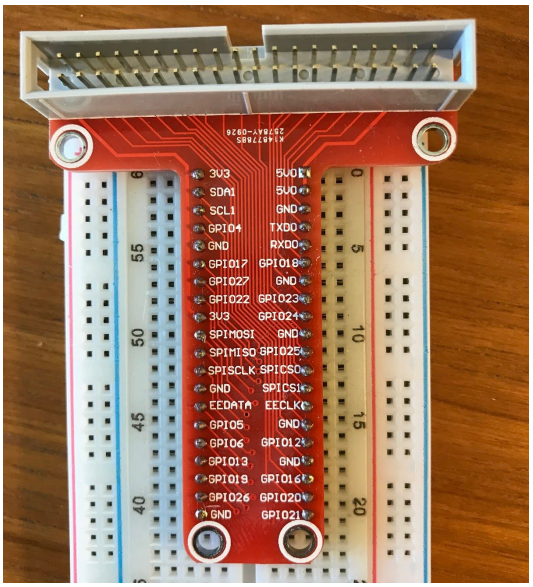
Last updated March 17, 2018

Congratulations on setting up the Raspberry Pi !!!

This next project will show why the Raspberry Pi is much more than just a “small computer” - we’ll illustrate how to use the Raspberry Pi to control physical objects.

Locate the following parts within the GSOC Raspberry Pi Box: Breadboard (white rectangle with lots of small pin holes), multi-colored ribbon cable, wire jumpers, resistors and a T-shaped “Raspberry Pi cobbler”, which connects the breadboard and ribbon cable as shown in the picture below.

STEP #1 Attach “cobbler” to the breadboard as shown below



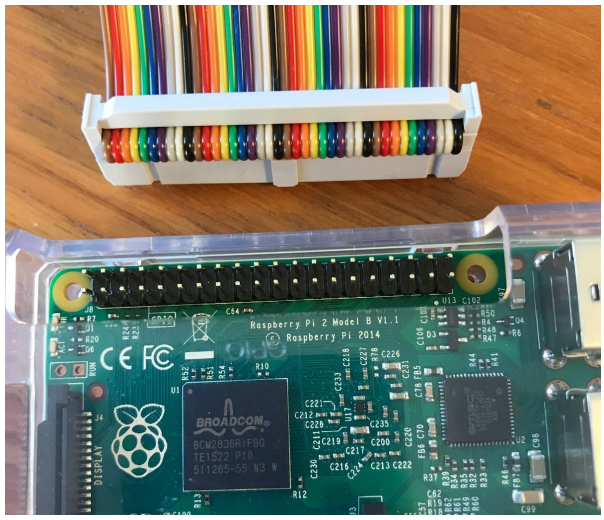
In the picture above, the bottom most cobbler pins (labeled GND and GPIO21) are inserted into the breadboard rows numbered 40 and 20 on the breadboard.

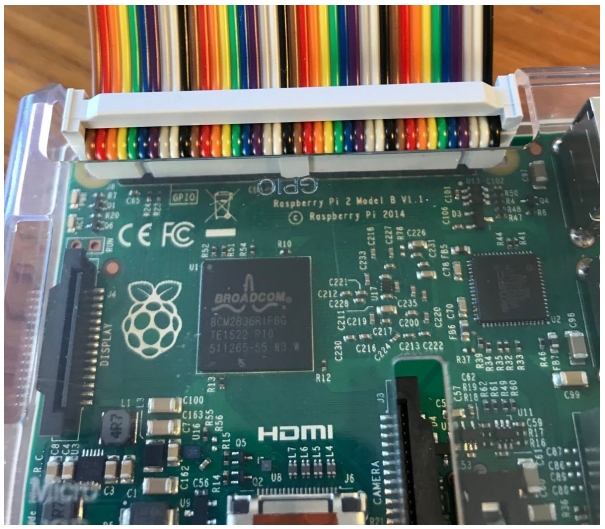
STEP #2 Attach the ribbon cable to the cobbler

Note that the notch in the cable matches the notch in the cobbler connector.

STEP #3 Attach the other end of the ribbon cable to the Raspberry Pi

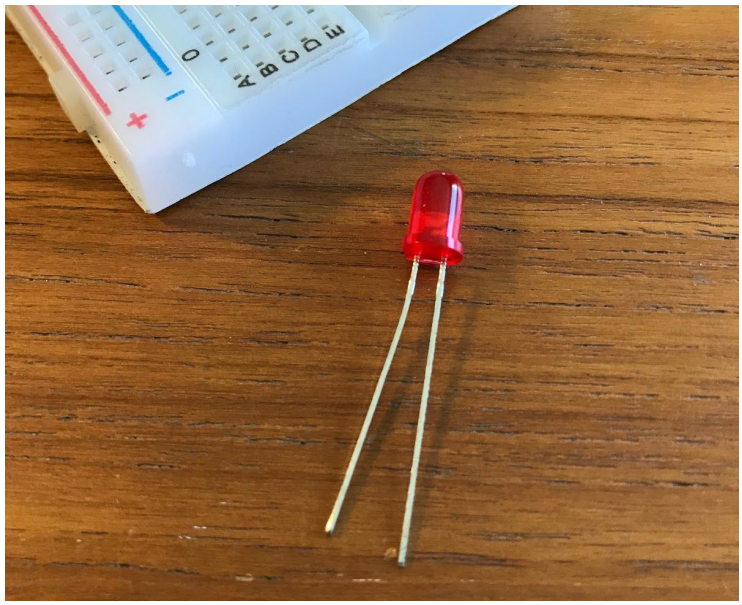
*** IMPORTANT * please POWER DOWN the Raspberry Pi** by removing the white micro USB power cord from Raspberry Pi, PRIOR TO connecting the ribbon cable as shown below.





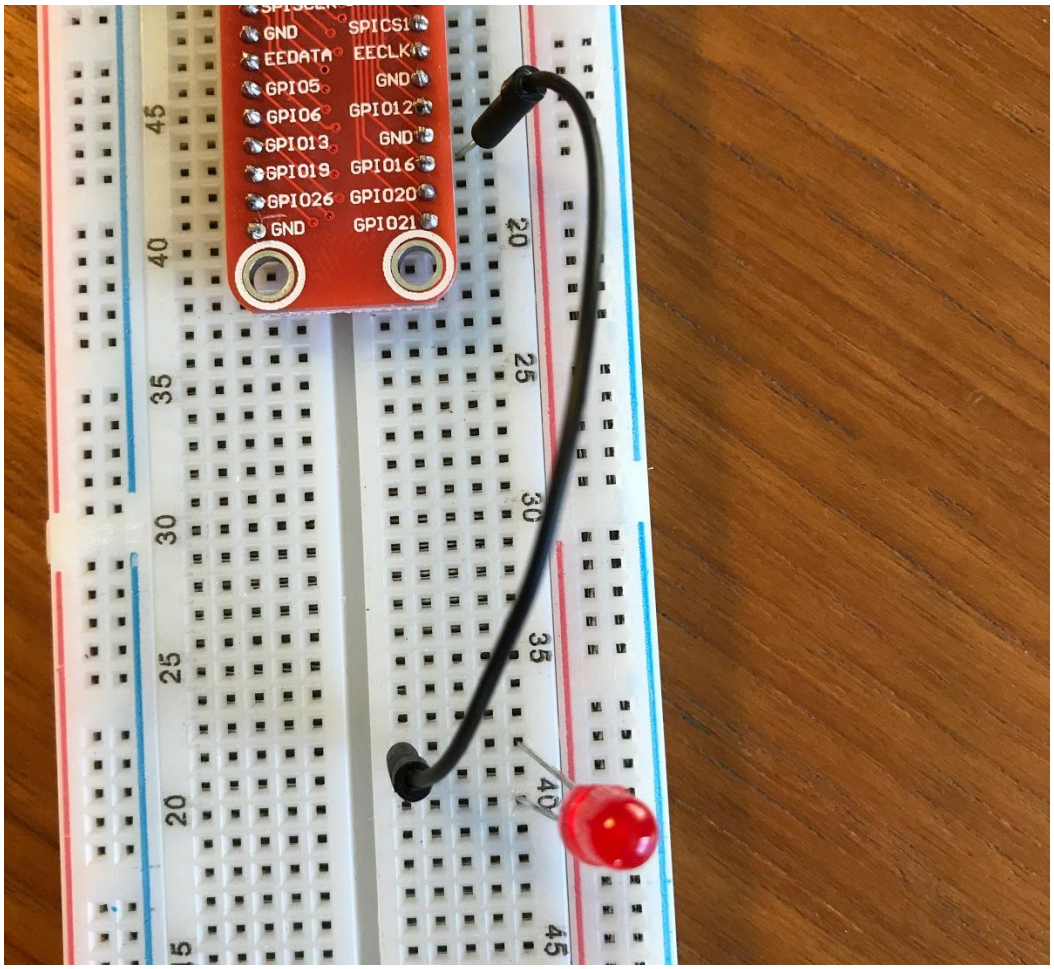
Note that the notch in the ribbon cable faces the center of the Raspberry Pi as shown in the picture above.

STEP #4 Locate a Red LED light from the Raspberry Pi box as shown below



One of the LED's legs are LONGER than the other, the longer leg is known as the ANODE or POSITIVE (+) pole of the LED, the shorter leg is called the CATHODE or NEGATIVE (-) pole.

STEP #5 CAREFULLY plug the LED into the breadboard



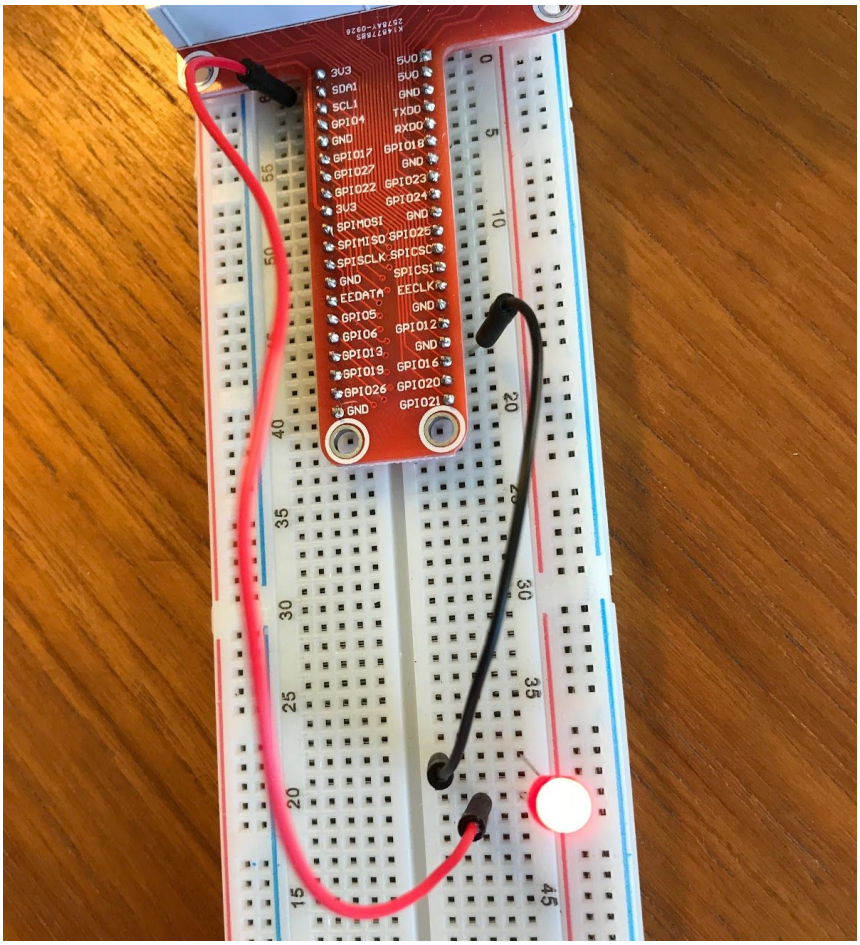
Plug the ANODE, or positive side of the LED (i.e. the longer leg) into the row numbered 40 and the opposite, or CATHODE side of the LED (-) into row 38. Next, locate a jumper wire from within the box and place one end of the jumper in row 38 and one in the row corresponding to the GND pin on the cobbler as shown in the picture above. (note: doesn't matter which of cobbler GND pins you use!)

When oriented as shown in the picture above, ROWS within the boardboard are interconnected, meaning that a current will flow between pins within the same row. Note that the grove in the center of the breadboard separates the connectivity between rows, meaning pins within the row labeled 20 and pins within the row labeled 40 are not connected together.

In the picture above we've connected ("ground") to the CATHODE (negative) side of the LED. We've used a black jumper wire as black is the color traditionally associated with ground. You can however use any color your like! Current flows through a wire regardless of its external color.

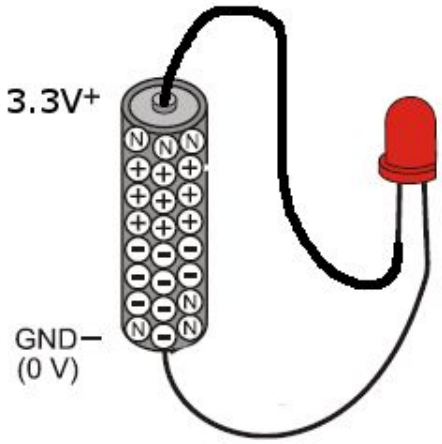
Now go ahead and power the Raspberry Pi up at this point by plugging the the white USB cable back into the Raspberry Pi.

STEP #5 Connect a jumper wire between 3v3 and the LED ANODE



MOMENTARILY connect a wire between 3v3 on the cobbler and the ANODE pin of the LED (row 40). The LED lights up! Don't leave the wire plugged in as shown above for very long as the LED will eventually "burn out".

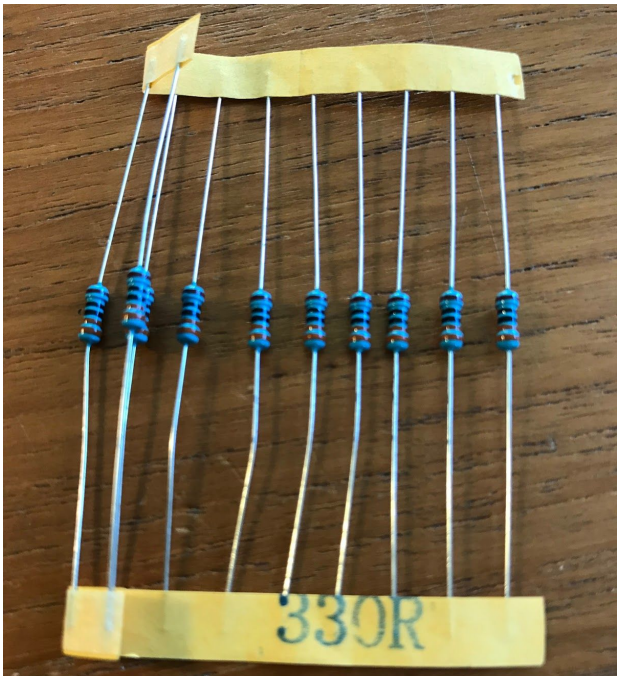
This circuit as shown above is identical to connecting the POSITIVE (+) pole of a battery to the ANODE of the LED and the NEGATIVE (-) pole of the battery to the CATHODE of the LED. Red is the traditional wire color to represent "positive", but once again you can use any color wire you'd like.



STEP #6 Add a protective RESISTOR

Did you note how brightly the LED lit up? The LEDs will not last long when receiving the full voltage provided in our last circuit. Therefore, we need to add a protective RESISTOR which will lower the voltage to the LED.

Locate a set of 330 ohm resistors within the box. If the 330 ohm resistors are all used up, then it's OK to use 220 ohm resistors instead. The resistors may be loose within the box or attached to the breadboard, so search around! Note that the stripes on the resistors are different depending upon the resistance rating (ohms is a measure of the amount of resistance).

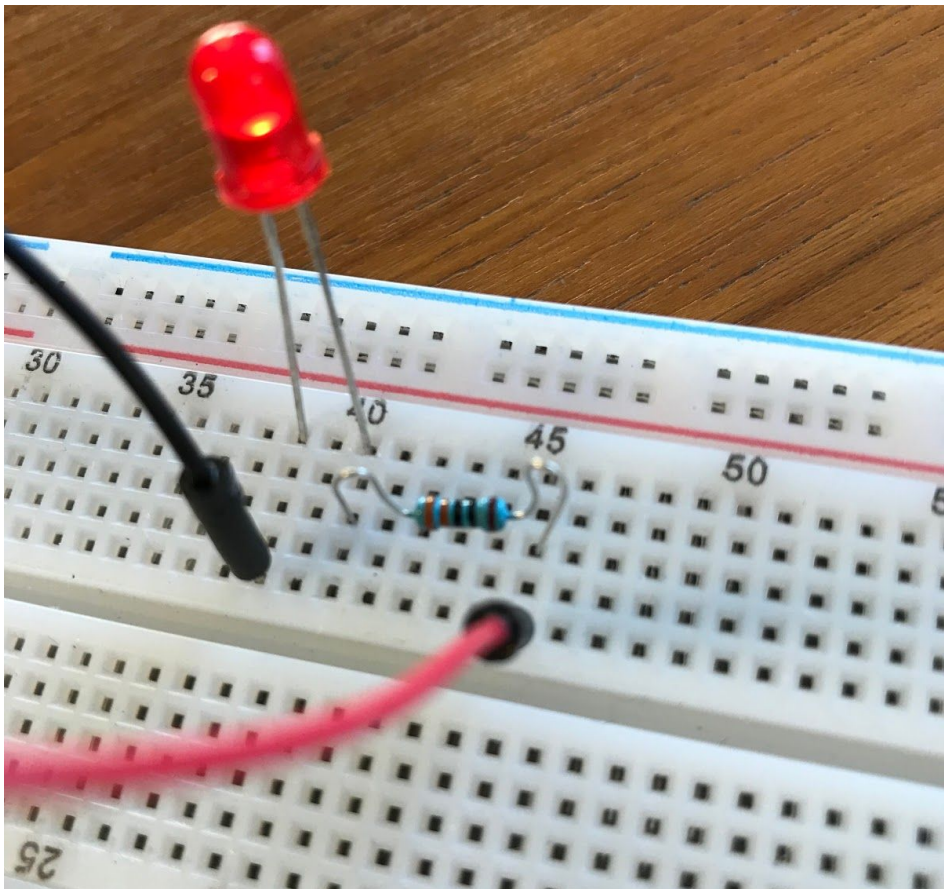


Carefully remove the resistor and bend the resistor's legs as shown below.

Hard to see here, but the 5 colored stripes are orange, orange, black, black, and brown.

STEP #7 Add the resistor to our breadboard circuit

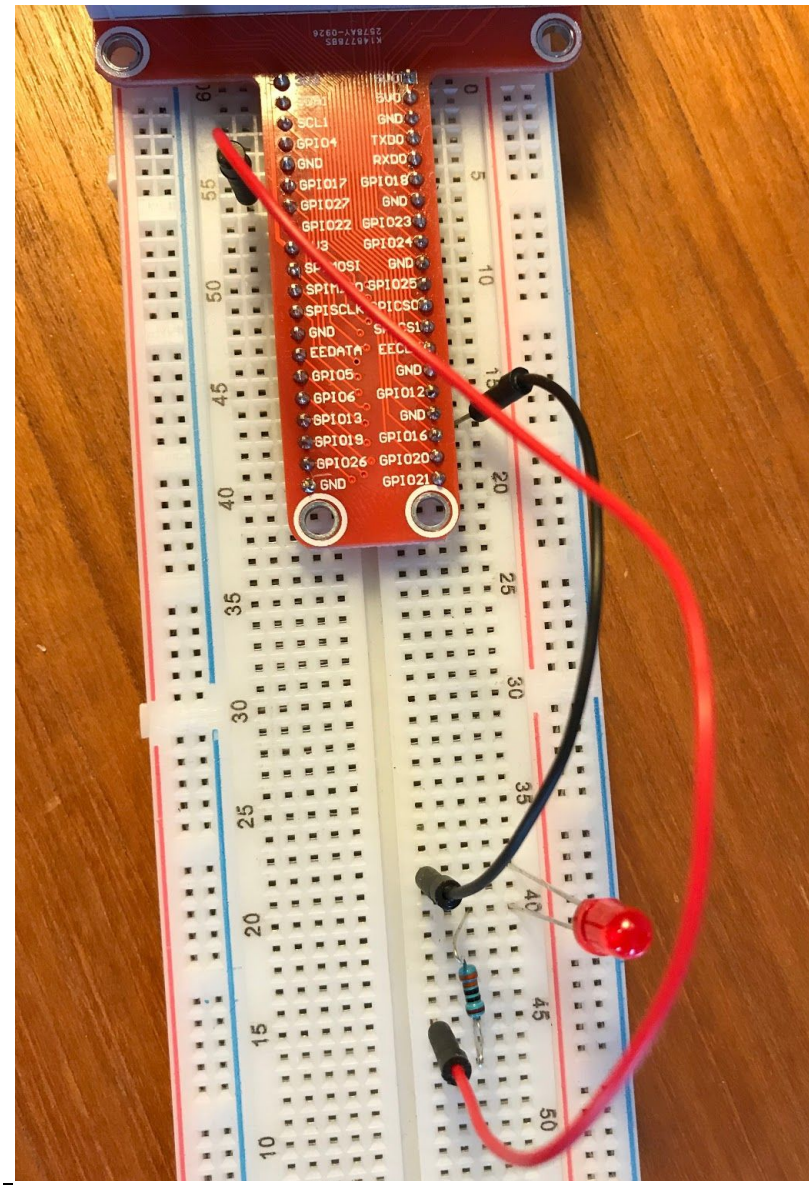
Connect one end (doesn't matter which end) of the resistor in row 40 and the other in row 45 (or whatever row works best for your bend). Now reconnect the jumper wire between row 45 and 3v3 on the cobbler. The LED lights, but not as brightly! Why? Because we have lowered the voltage to the LED using the resistor.



You can "experiment" by moving the red jumper (as shown above) between rows 45 and 40 as shown in the example above. By moving the red jumper to row 40, we are bypassing the resistor. Returning the red jumper to row 45 adds the resistor back into our circuit. The LED light should be glow brightly when the resistor is bypassed (wire in row 40) and less brightly when the resistor is included within the circuit (wire in row 45). Be sure to return the wire to row 45 as shown above so that we are including the resistor in the circuit.

Now let's use the Raspberry Pi to control the LED!

STEP #8 Move the jumper wire from the cobbler's 3v3 to the GPIO17 pin as shown

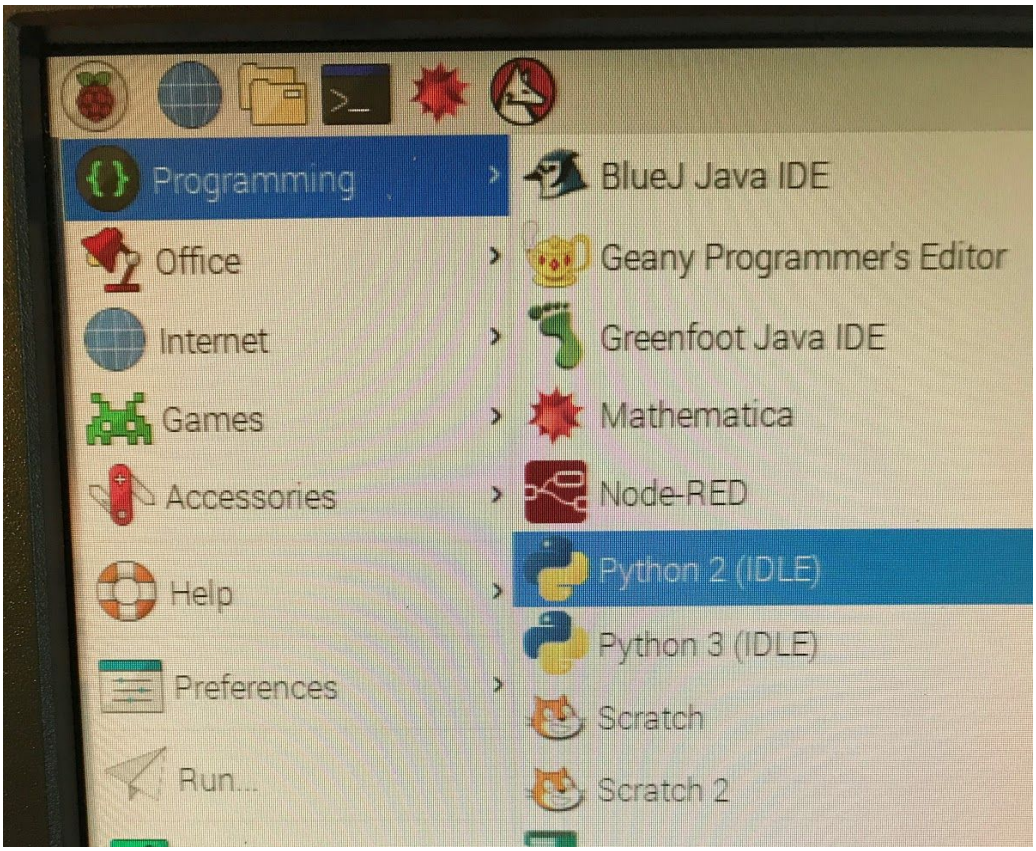


The LED will go out once we disconnect the ANODE from 3v3+. In the following steps, we will write a program that will bring 3.3v power to the GPIO17 pin, and this will light the LED back up.

GPIO stands for "General Purpose Input and Output bus". "17" is the port number of the bus that is connected to our LED. Our program will reference this port number.

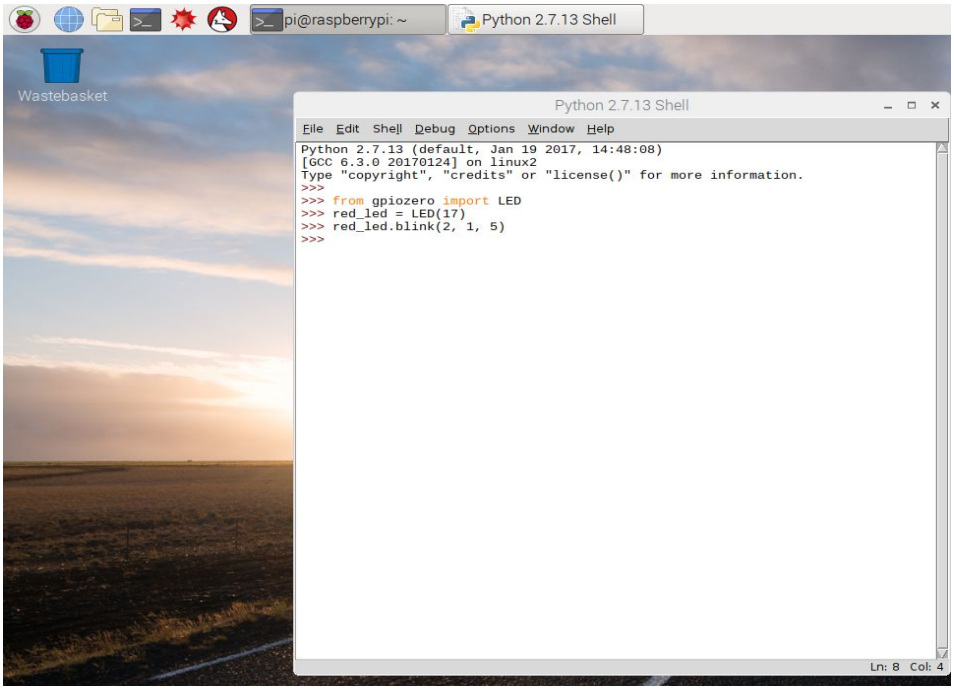
Note that there are several other port numbers called out on the cobbler (e.g. GPIO27, GPIO22, GPIO6, etc.). Each port allows us to control (i.e. "write to") an object like a light or a motor, or alternatively "read" a sensor like a motion detector, temperature sensor, or button.

STEP #9 Move back to the Raspberry Pi's screen, keyboard and mouse

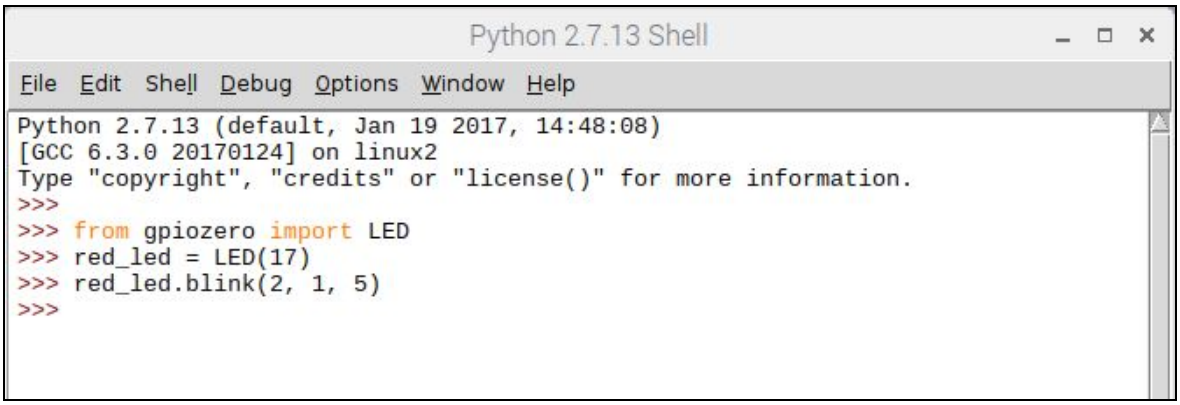


Click the "Raspberry" icon on the top left corner of the screen, which will expose a menu. Select the programming menu item and then Python 2 (IDLE) as show above.

This will expose the Python programming language IDLE (integrated development environment) application window as shown below.



STEP #10 Enter this 3 line "program" into the Python IDLE as shown below

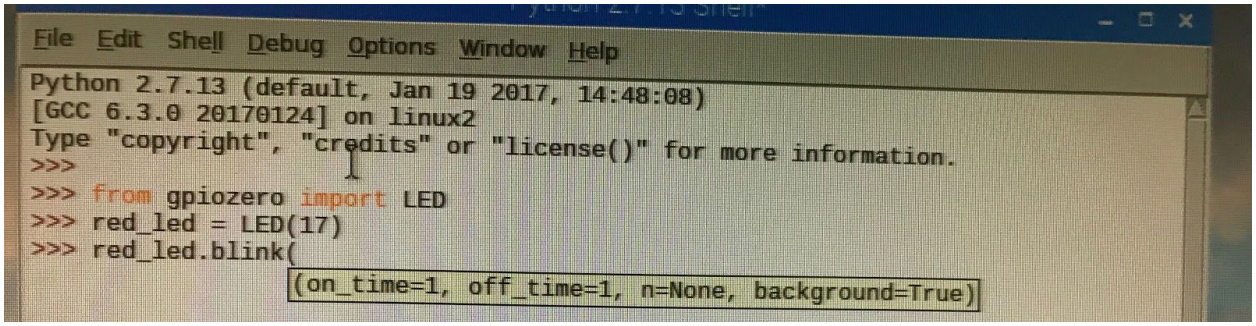


In this "interactive mode" of the IDLE, each line we enter is immediately executed. The first line imports a "module" which is a previously written program that we will use within our program. In this case we are importing the module "gpiozero" which supports lighting LED's, reading buttons, etc. We are specifically only importing the LED method from the gpiozero module within this first line when we enter: "`from gpiozero import LED`".

Remember GPIO port #17? In the second line we create a variable (more correctly, an "object") that represents an LED light connected to the Raspberry Pi's GPIO port 17. We'll call it `red_led`.

Our third line uses the "blink" method of the "`red_led`" object we created in line 2. The blink method's arguments are: # seconds ON, # seconds OFF, # number of repeats. We will be turning our LED ON for 2 seconds, then off for 1 second, and we will repeat that sequence 5 times. Be sure you're looking at your breadboard when you hit ENTER after typing this line, as the LED should start blinking away!

You may have noticed that the the IDLE helps you identify arguments to methods as you enter your code. Note the picture below shows how to understand the arguments of the method "blink" (e.g. on_time=1, off_time=2, etc.).



The actual values shown above are the "default" values that the blink method would use if you didn't enter any specific values between the blink method's parentheses.

Each time the LED "blinks on", the GPIO pin 17 is "turned up" to + 3.3v by our program, thereby creating the same basic circuit we'd previously used.

STEP #11 Create a "Reaction Time" program using this same circuit

Now let's learn how to write and save a program with the Python IDLE. Select the menu item File from the Python IDLE application's menu followed by the New File menu item. This will create a new blank "Untitled" window on the screen.

Carefully enter the program as shown below, which we review line by line below.

File Edit Format Run Options Window Help

```
from gpiozero import LED
from time import time, sleep
from random import randint

red_led = LED(17)

print "Get ready to press the ENTER KEY when the red light comes on!!!"
sleep(1)
best = 9999

for attempt in range(1, 11):
        sleep(randint(1,10))
        red_led.on()
        start = time()
        raw_input()
        end = time()
        red_led.off()
        print "Attempt nbr: ", attempt, " Seconds: ", end-start
        if (end-start) < best:
                best = (end-start)

print "Your best was: ", best, " seconds in ", attempt, " tries."
```

The first line is the same as our initial three line program; we are importing the gpiozero's module's LED method. The next two lines are similar; we import methods time and sleep from the "time" module and randint (return a random integer) from the "random" module.

The next line is the same as our initial program; we create a `red_led` object which is connected to our breadboard's LED wired to GPIO port 17.

Next we will a print a message to our "user" reminding the user to be ready to press the keyboard's ENTER key when the red LED light turns on.

We let the program sleep for 1 second so the user has a moment to read the message. Next we create a new variable called "`best`" and set it to a value of 9999, a very high (meaning here "very slow") reaction time. Try to figure out why we initialized `best` to this very high value as we review the rest of this program.

Now we create a LOOP, or a block of code that will repeat until some condition is met. In this case our look repeats 10 times, within the range of numbers 1 through 11. This line will set the variable "`attempt`" to the value of 1, 2, .. up to 10, counting each iteration of the loop.

Within our loop, the first thing we do is let the program sleep a random amount of time between 1 and 10 seconds prior to turning on our light. This makes it harder for the user to "guess" when

the LED is about to light up. Note that you need to INDENT lines within the loop using the keyboard's Tab key; lines that indented execute within the loop.

Now we turn the LED on just as we did in our initial program, except we use the `red_led` object's ON method instead of BLINK method we'd used previously. The "on" method just turns on the light, and we'll need a separate way to turn it back off.

As soon as the LED is turned on, we record the current time in the variable "start".

The next line, "`raw_input()`" simply waits for the user to press the ENTER key on the keyboard. When pressed, the program continues to the following line.

The next line records the current time in the variable "end". We then turn the LED off using the `red_led` object's OFF method.

In the next line:

```
print "Attempt nbr: ",  attempt, " Seconds: ", end-start
```

… we "print" (send to the screen) some text (e.g. "Attempt nbr: ") and the current value of the variables `attempt, end` and `start`. `Attempt` contains the loop iteration count, and the `start` time subtracted from the `end` time represents the user's "reaction time"; the time it took for the user to press the ENTER key after the LED lit up.

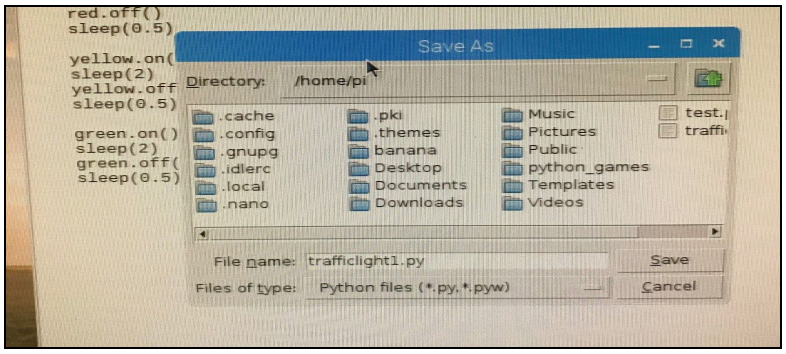
Lastly, we check to see if this loop iteration "reaction time" was the best overall. We compare the variable `best` to the reaction time, and if the reaction time was "faster" (i.e. less than) then the prior value of `best`, then variable `best` is set equal to the new record reaction time.

You will notice that the last line:

```
print "Your best was: ",  best, " seconds in ", attempt, " tries."
```

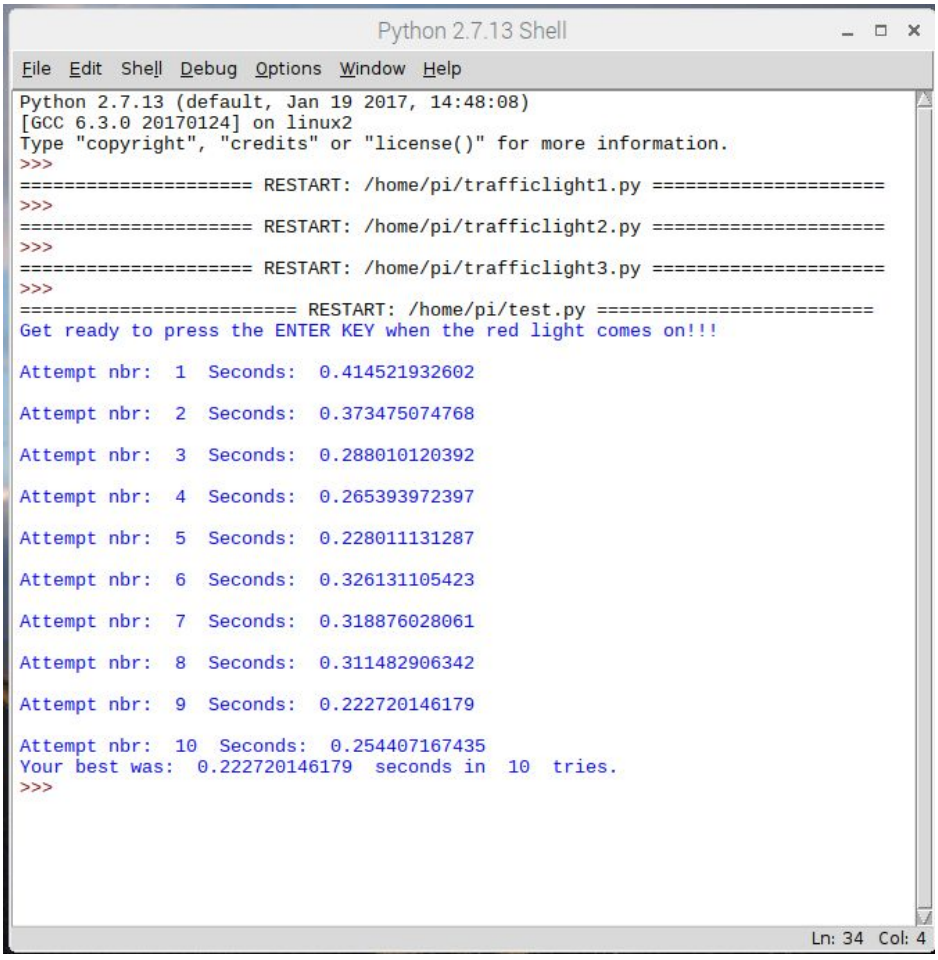
… is not indented as our previous lines were. Python uses indenting to determine which lines belong within a loop or within a conditional statement like "`if (end-start) < best:`".

You should save this program after you enter it. You might very well have made a mistake in entry, or you may wish to modify it further later. Use the Python IDLE's File Save command to save your program, using any name you wish, however, please do use the file extension .PY which is the standard for python programs. (Here we used filename trafficlight1.py).



STEP #12 Run your reaction time program!

After entering and saving your program, press the F5 key to run the program, or alternatively select the menu commands Run and then Run Module from the Python IDLE application window. When a program runs within the Python IDLE application, it will open a separate window to display the output of the program. Your output window will look similar to the one shown below:



Our attempt number and reaction time is shown for each loop iteration, followed by our overall best reaction time and total number of loop iterations.

Congratulations if this is your very first Python program! You learned how to enter and save a program using the Python IDLE, and how to use a loop and a conditional statement!

If your program returned an error, don't forget that you can use File Open to reopen your saved work. Recompare your code to the lines on the prior page and try again! Be sure you are indenting correctly. Programs rarely work right the very first time, so don't be discouraged, try again!

Did you have the fastest reaction time within your troop? Why did we choose to initially set the value of variable `best` to 9999?

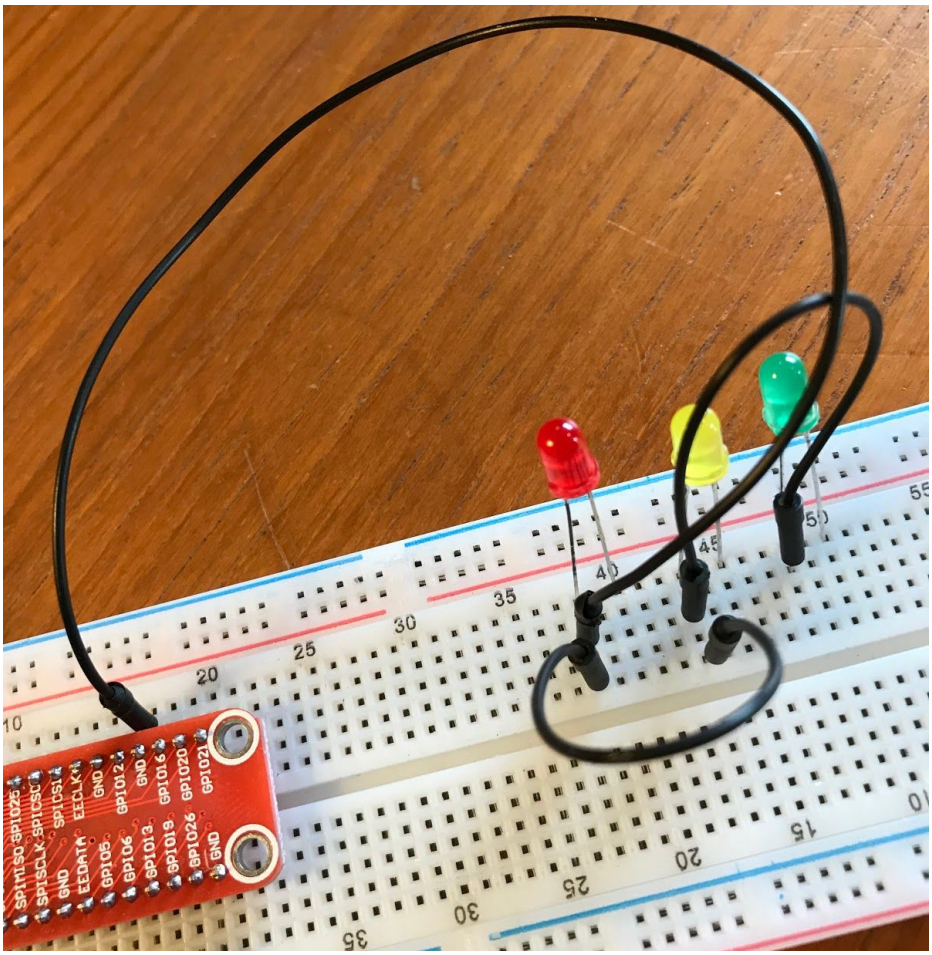
GSOC Raspberry Pi - LED Lights - PROJECT #2

Congratulations on completing your first breadboard circuitry and programming project!

Now let's expand on what we've learned so far and create a three LED "traffic light" project.

STEP #1 Locate yellow and green LEDs, two more resistors, and four more wire jumper wires.

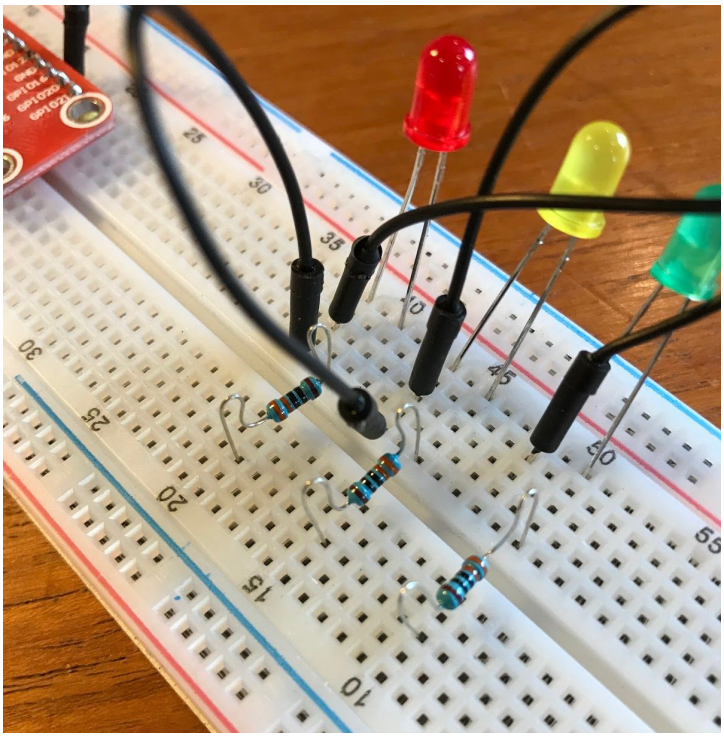
Why stop at just 1 LED when 3 is way more fun! We'll wire the yellow and green LEDs exactly the same way as our red LED. Place the ANODE (longer lead) of the yellow LED in row 45 and the CATHODE in row 43. Place the ANODE of the green LED in row 50 and the CATHODE in row 48.



Next, locate a short jumper wire and connected the CATHODES (rows 43 and 48) of our yellow and green LEDs together. Add a jumper wire jumper connecting row 43 and our red LED's CATHODE which we'd placed in row 38. Now all three LED's CATHODES are connected together, and in turn, connected to the cobbler's GND pin as shown above.

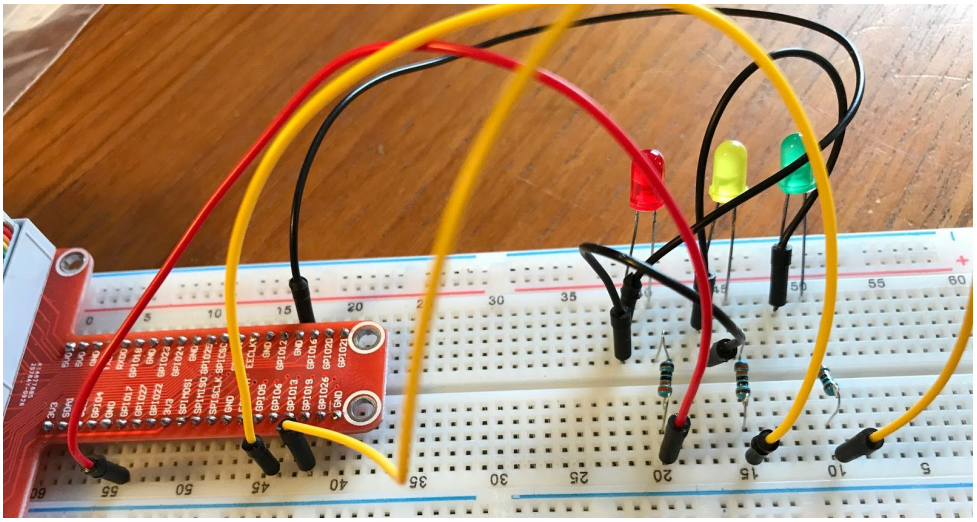
Now let's connect the resistors and the ANODE pins of our 3 LEDs. Remember that the groove within the breadboard separates the right (rows like 40) and left (rows like 20) sides of the

breadboard. We can use that to our advantage here and “cross the groove” with our three resistors. Place one end of each resistor in rows 40, 45 and 50, or each row containing the ANODE pins of each of our three LEDs.



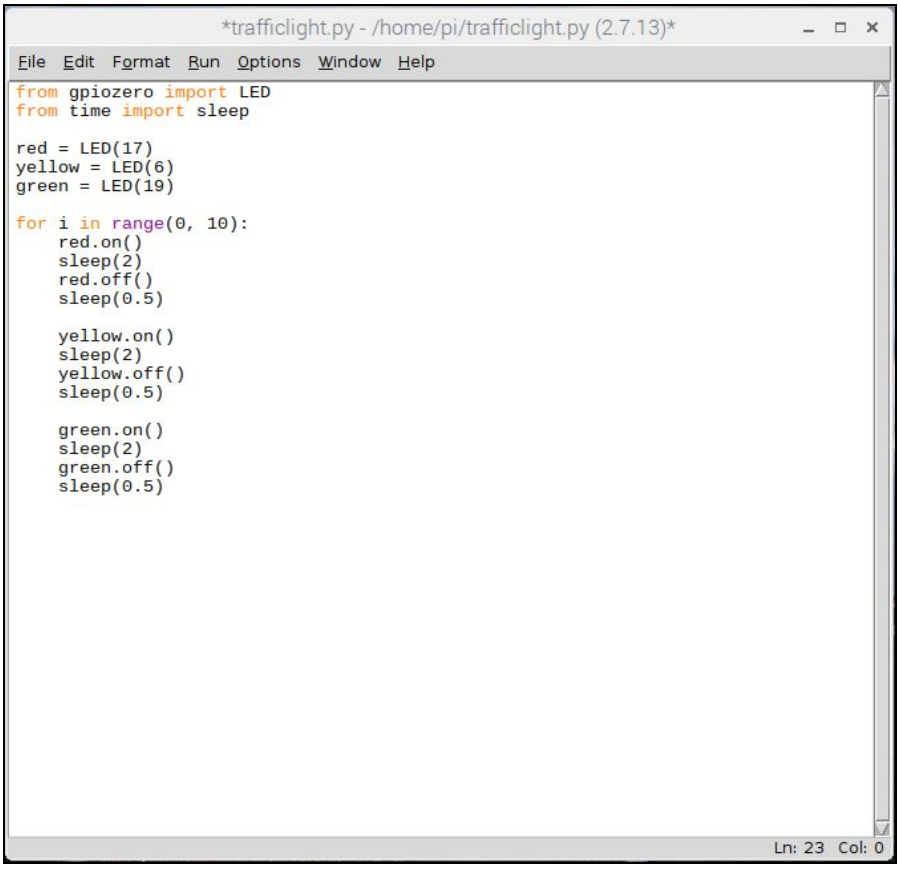
Place the other end of the resistors in the corresponding rows on the other side of the breadboard’s groove; rows 20, 15 and 10 as shown above.

Now we need to connect this end of our resistors (rows 20, 15 and 10) to our Raspberry Pi’s GPIO pins. Let’s reconnect our red LED’s resistor (row 20) to GPIO 17, and we’ll add two new jumper wires to connect GPIO 6 to the yellow LED’s resistor (row 15) and GPIO 19 to the green LED’s resistor (row 10) as shown below. I ran out of red colored wires, so I used yellow jumper wires instead!



STEP #2 Write a new Python IDLE program to turn all three LEDs on and off

Open the Python IDLE as we had in Project #1, and select the menu item File from the Python IDLE application's menu followed by the New File menu item. This will create a new blank "Untitled" window on the screen. Enter the program as shown below, which we will review line by line.



In our first two lines we import the gpiozero and time module that we will use within this program. Note that we only need the sleep method from the time module for this program as we're not trying to capture time or to "time" anything like we did within our Reaction Game program.

Next, we enter three lines which create three new objects, `red`, `yellow` and `green`, and we assign each variable to their appropriate Raspberry Pi GPIO ports (GPIO ports 17, 6 and 19) based upon our circuit wiring.

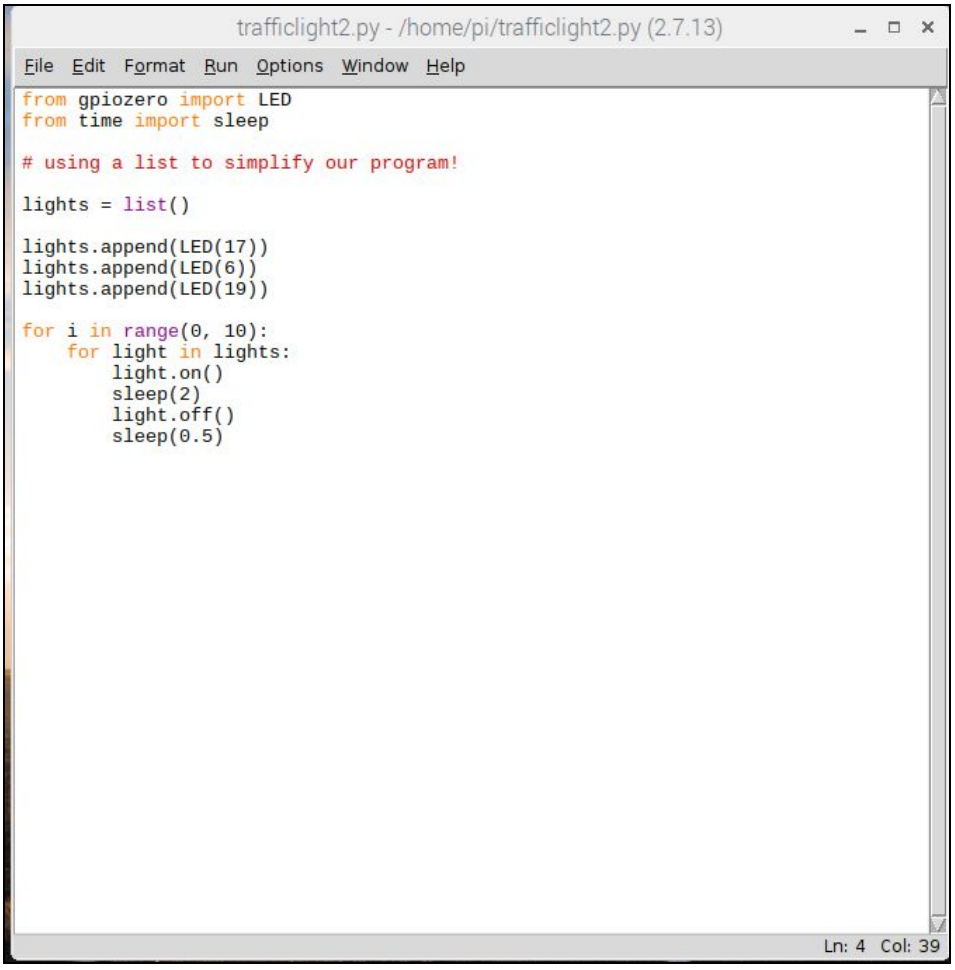
Now we start a loop that runs ten times, from 0 to 10. Within that loop (and don't forget to INDENT using the keyboard's Tab key) we:

a) Turn a LED on
b) Wait 2 seconds
c) Turn that LED off
d) Wait ½ of a second (0.5 seconds)

We wait a little bit after turning the LED on and prior to turning if off so we can be sure we see the LED lit up! Note that we repeat each of the four steps (a) - (d) for each of our three red, yellow and green LEDs.

Once we've entered the program as above, select the F5 key to run the program. Your breadboard should act like a "traffic light", changing from red to yellow and to green.

We mentioned above that we needed to repeat these same four lines of code for each of our three LEDs. Programmers often look for ways to make their code more readable and efficient, and to eliminate "redundant" or repetitive code. Let's modify our program to include a new structure called a LIST and show how we can shorten this program.



In the program above we add a new line:

```
lights = list()
```

This line creates a new list object that we'd named "`lights`". We then append our three LEDs to list `lights` using the subsequent three lines (e.g. `lights.append(LED(17))`).

Now we can "loop over" each of the three LED items within list `lights` and execute the same four lines of code. We start by adding an "inner loop", or an additional loop inside of our existing loop.

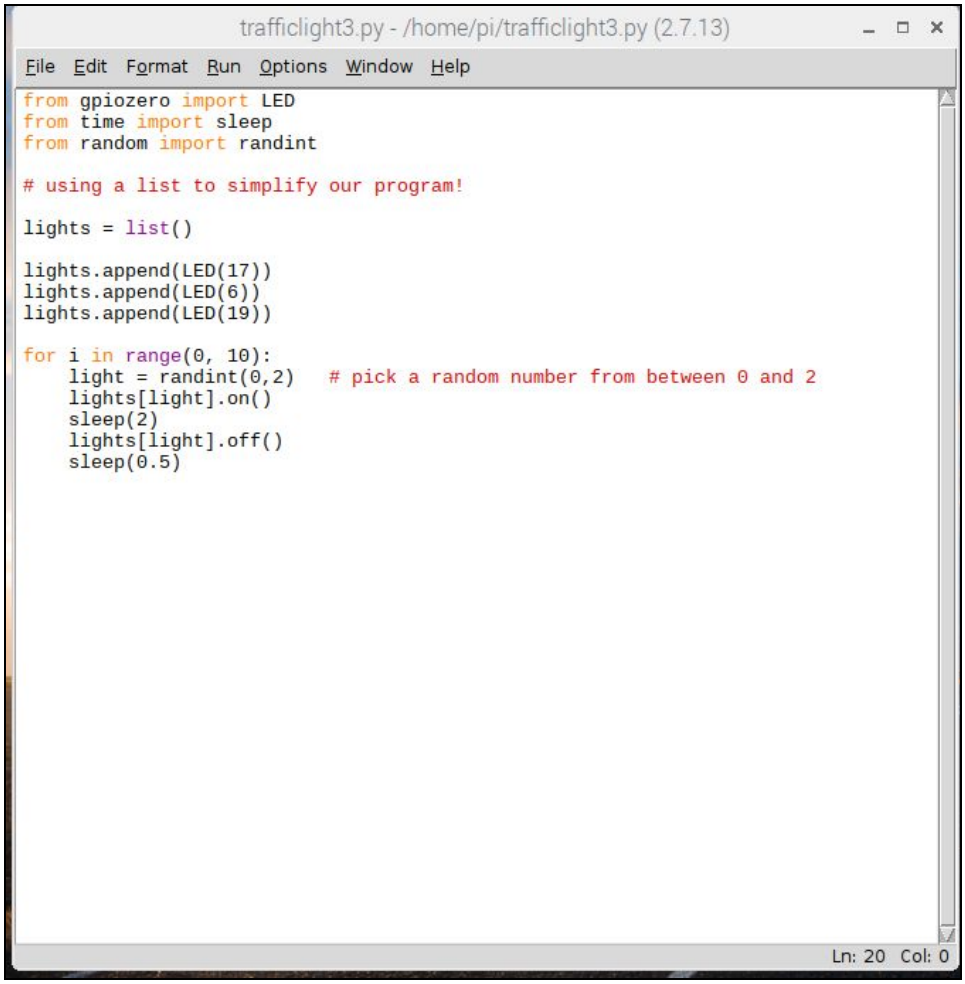
The INDENTED line:

```
    for light in lights:
```

… means “for each `light` within the list `lights`”, execute the indented lines found within that loop, which are the four lines that turn on the LED, wait for 2 seconds, turn off the LED, and then wait another ½ second.

Use F5 to run this program, and note that our “traffic lights” should behave identically to our prior program.

Now let’s make a simple addition to the program above that will randomly flash our lights on and off, instead of the red -> yellow -> green ordering we have been using so far.



Can you spot the changes to our program? First, note at the top of the code that we’ve added a reference to module “random” which we used in Project #1’s reaction time program.

Next, we’ve replaced our inner loop with the line: `light = randint(0,2).` This line sets a random value of variable `light` to a value of either 0, 1, or 2.

The remaining four lines are slight modifications of our previous four lines. We use square brackets after list variable names in order to refer to specific items within the list. For example, `lights[0].on()` means “reference the first item in list lights and then execute the “on” method of this item”. In Python, the first item within a list is the ZEROth item; as counting starts at 0 in Python. (Remember, you were “0” years old before you were 1 year old).

In the code above we have the line `lights[light].on()`. Since we don’t know exactly what the value of variable `light` will be (only that the value of `light` will be between 0 and 2) we don’t know which of our three lights will light up!

Go ahead and run your program using F5 as you had previously and see what happens!