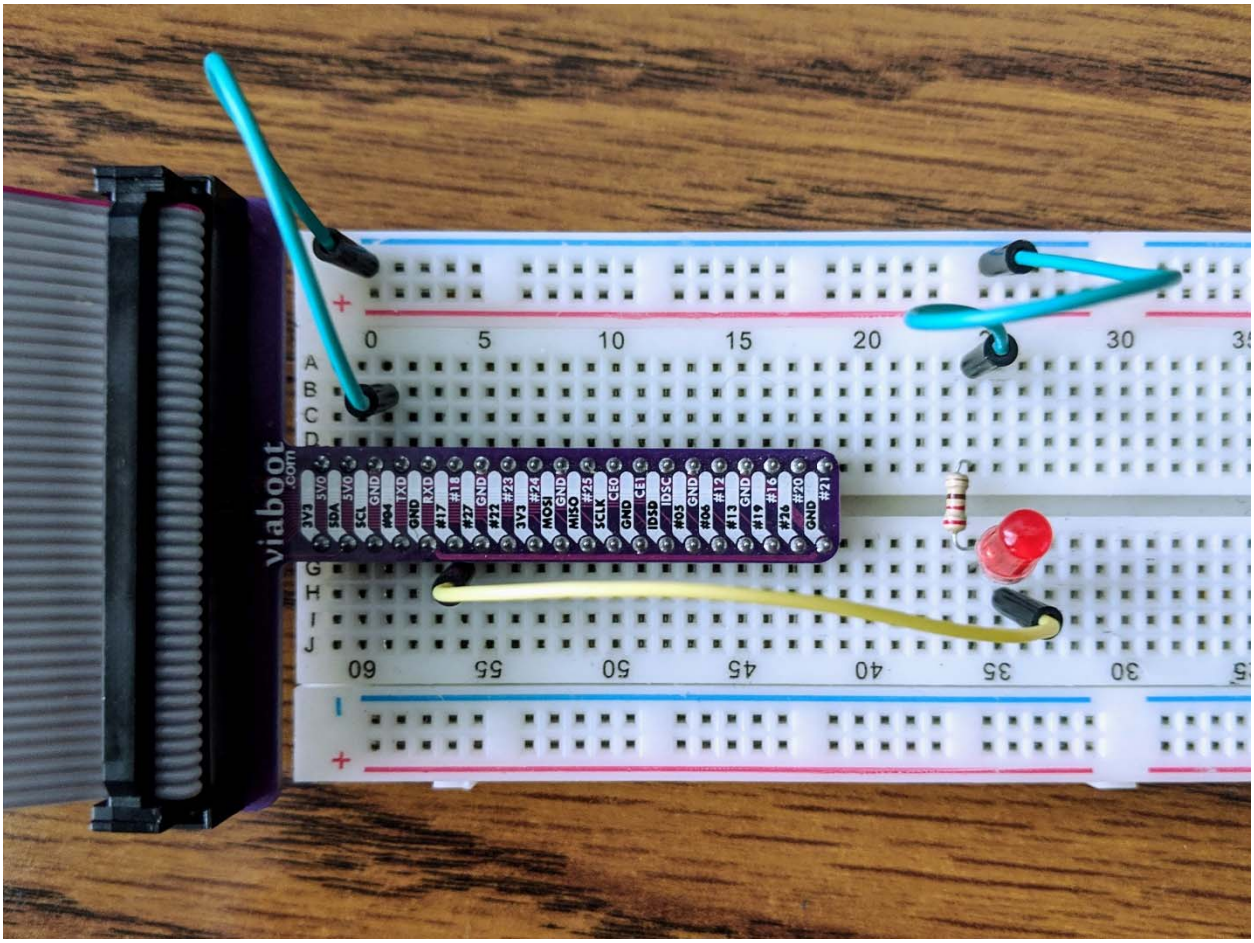
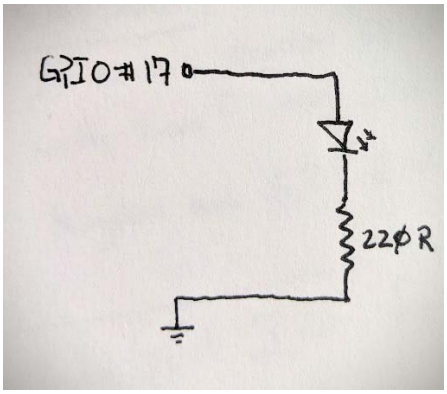
GPIO Python Iterative Scripting (Part 1):

First, connect your breadboard to the GPIO breakout cable and plug the cable into the GPIO. Then, connect the positive end of the LED to pin #17, a 220 ohm resistor to the negative terminal, bridging it to the other side of the board. Set the "-" strip as ground by connecting the top of it to a "GND" pin. It should look something like this: (remember that the long end of the LED is the + end)





This is about as simple of a circuit as they get. Basically if you moved the yellow wire to one of the 3.3v or 5v pins of the Pi, the LED would light up. We have it plugged into pin #17 though because we want to control the LED with the Pi. The circuit diagram for this is to the left.

So. To start out, let's create a file called blink.py and edit it. If we are doing this on the Pi itself we would:

```
touch blink.py
nano blink.py
```

(This blink.py is the same as the one on the Google Classroom. I'm going to explain what everything in it does before moving on to the next iteration of the script)

```
#Import libraries
import RPi.GPIO as GPIO
import time
```

Importing libraries should be very familiar to you at this point in your Computer Science Career, so I don't think that we need to get into it so much except to say that the RPi.GPIO library is pretty much THE library to use with the GPIO. We will be seeing a lot of it. The time library is necessary because we are going to be using the sleep function to make the light blink instead of just light up.

```
#Initialize the GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.OUT)
```

As the comment indicates, this initializes the GPIO, and assigns pin 17 as an output. This means that the Pi will put 3.3v out onto pin #17 when it is triggered.

```
#This function will make the light blink once
def blinkOnce(pin):
        GPIO.output(pin,True)
        time.sleep(1)
        GPIO.output(pin,False)
        time.sleep(1)
```

More accurately, this function makes the LED blink one time, with a 1 second duration on, and a 1 second duration off.

```
def blinkOnce(pin):
```

In this line, "def" indicates that we are defining a function. "blinkOnce" is the name of the function, and "(pin)" indicates that this function requires one argument named "pin"… easy peasy.

```
GPIO.output(pin,True)
```

Sets the pin assigned in the argument "pin" to the "True" state, which in turn supplies 3.3v to that pin.

```
time.sleep(1)
```

Was mentioned above. It waits 1 second before proceeding.

```
GPIO.output(pin,False)
```

Sets the pin assigned in the argument "pin" to the "False" state, which turns off the voltage to that pin.

```
time.sleep(1)
```

Was mentioned above. It waits 1 second before proceeding.

```
#Call the blinkOnce function above in a loop
for i in range(10):
       blinkOnce(17)
```

This one does what it says it does. The “range(10)” means that it is going to loop (and therefore blink the light) 10 times. “blinkOnce(17)” calls the blinkOnce function defined above, and passes it “17”, which is the pin that the positive end of the LED is hooked up to.

```
#Cleanup the GPIO when done
GPIO.cleanup()
```

This basically un-assigns the pins, and un-initializes the GPIO. This is important to do, otherwise when you run this or other scripts that use the GPIO in the future you will get errors telling you that the channel is already assigned.

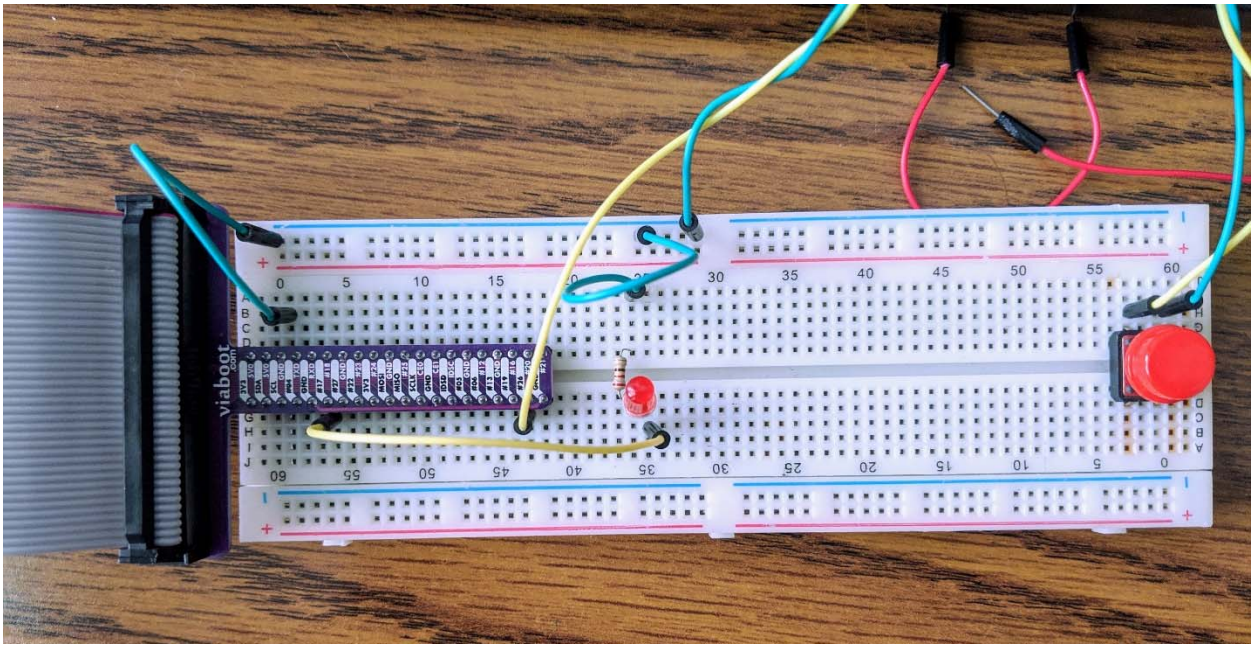
So when we run this script:

```
python3 blink.py
```

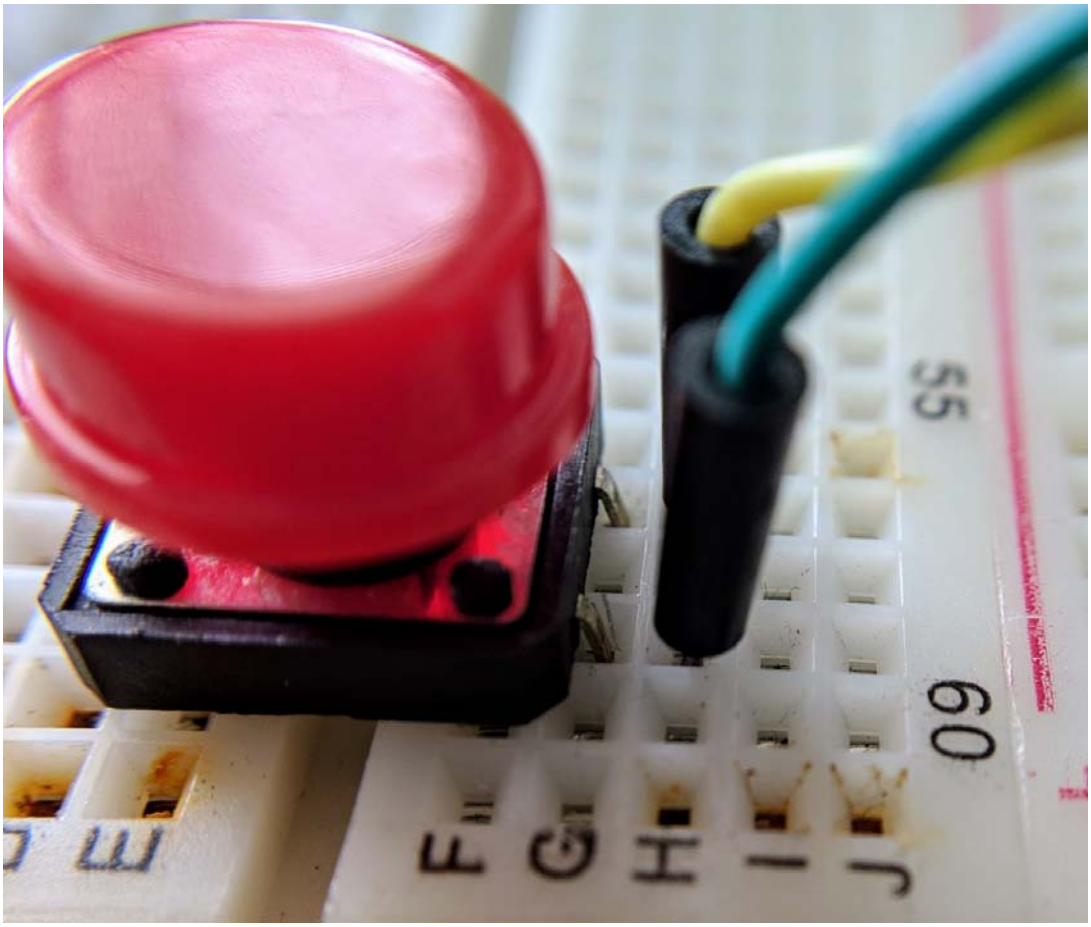
The LED should blink 10 times, one second on, and 1 second off. We could change those to say “.1” to make it blink quickly.

Ok. That was about as simplistic as this gets. Let’s add a little more to it.

Next, let’s add a switch, and put one end of the switch to pin #26, and the other end of the switch to ground. When you are done, it should look like this:



One thing to note here is how the switch works. This particular switch should get hooked up like this, with the green wire going to ground, and the yellow going to pin #26:



Now, change the imports to look like this:

```
#Import libraries
import RPi.GPIO as GPIO
import time
import os
```

The 'os' library we are importing allows you to call Bash commands from a Python script. It is very useful.

Then we need to add the switch to out GPIO setup section:

```
#Initialize the GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(26, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Ok, so basically we are telling the Pi that it needs to listen to port 26 for a signal. We are setting it up as an input, hence the 'IN'. The second portion of that line is saying that the signal it should be listening to is if the pin is 'pulled up' or 'pulled down'. In this case we assign the GPIO pin as 'UP', so we are listening to see if it is pulled down. If it were 'DOWN', and we wanted to see if it was pulled up, the basic state of the GPIO pin would be 0v, and it would be waiting until it was presented with voltage (probably 3.3v, but don't quote me on that). When it starts 'UP' it starts at 3.3v, and is listening for a voltage drop. Since we attached the other side of the switch to ground, when the switch is pushed, it will connect the 3.3v pin (#26) to ground, and the voltage will momentarily drop to nothing.

Lets change the blinkOnce function to flash the LED more quickly:

```
#This function will make the light blink once
def blinkOnce(pin):
        GPIO.output(pin,True)
        time.sleep(.1)
        GPIO.output(pin,False)
      time.sleep(.1)
```

So all we did was change the sleep times from '1' to '.1'.

Now we have to build a constant loop to listen for the switch:

```
try:
        while True:
                input_state = GPIO.input(26)
                if input_state == False:
                        for i in range(10):
                                blinkOnce(17)
                        time.sleep(.2)
```

So this is a while loop. It reads the input state of pin 26. When that pin's voltage is high, it returns 'True'. When that pin is grounded, it will return 'False'. Once the switch is pressed, it will trigger 'False' by grounding pin 26. That will invoke the blinkOnce function 15 times before waiting .2 seconds and returning to a listening state.

There is one thing left to do. We want this script to exit cleanly and cleanup the GPIO, but as it stands, it will just run forever. We are going to add a keyboard interrupt so the script exit:

```
#Clear the shell, print your goodbyes, and clean up the GPIO
except KeyboardInterrupt:
        os.system('clear')
        print('Thanks for Blinking and Thinking!')
        GPIO.cleanup()
```

And there we have it! Try it out with:

```
python blink.py
```