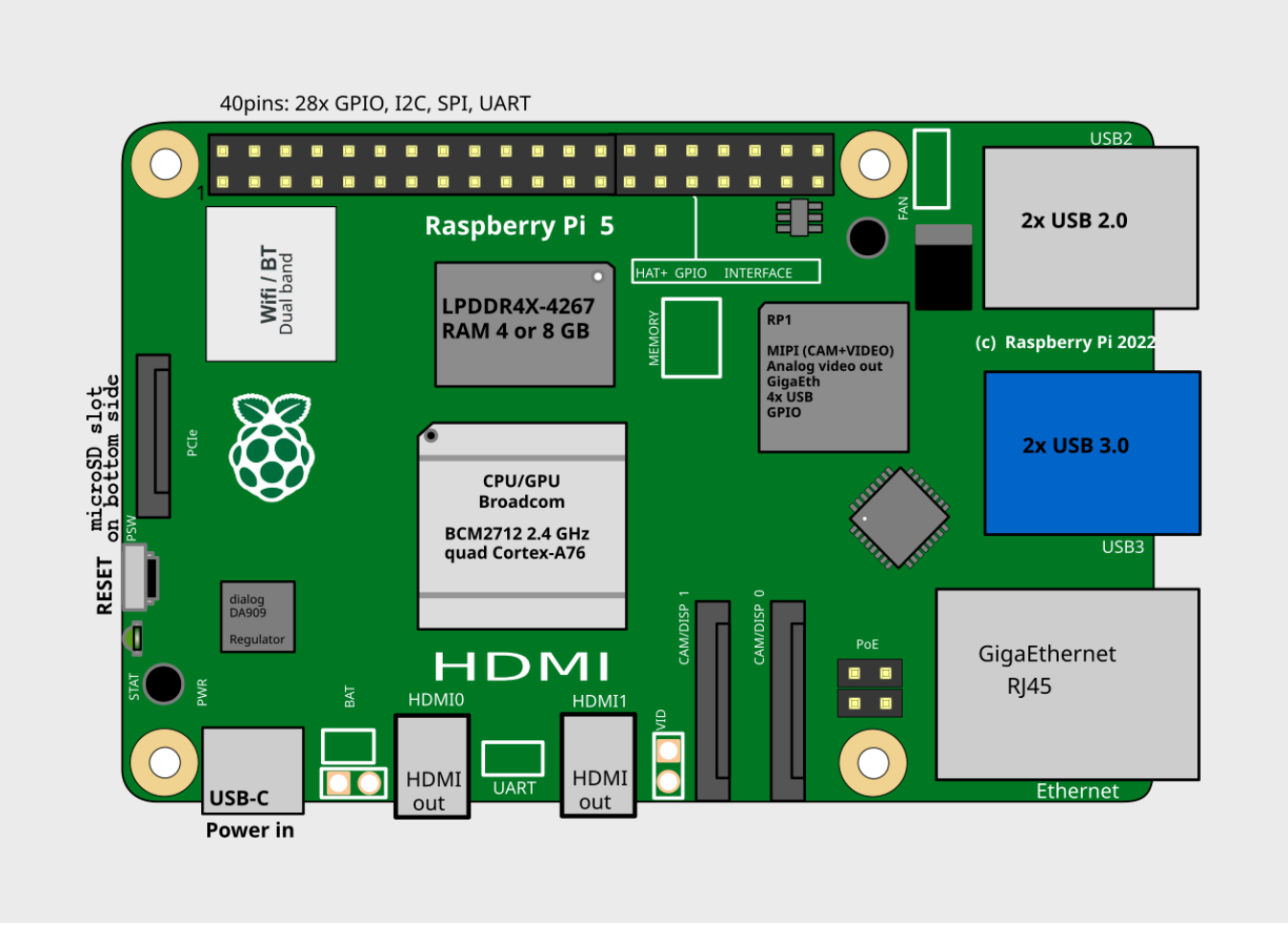
Lab 11: Raspberry Pi

There are several different varieties of “single-board” computers that enable digital connections to electronic components. The two most prominent platforms among tinkerers are **Arduino** and **Raspberry Pi**. Arduino systems are considered *single board microcontrollers*, whereas Raspberry Pi systems are full *single board computers*. Both have advantages and disadvantages. Arduinos tend to use less power and are generally “instant-on” systems. By contrast, Raspberry Pis are full computers, which come with larger power use and some “boot-up time,” but also allow for much easier interface and interactivity with sensors.

You will be using a Raspberry Pi today, but most of the electronics lessons we will learn here translate trivially to other microcontrollers/single board computers.

Part A: Your First Slice

Consider this schematic of the Raspberry Pi board:

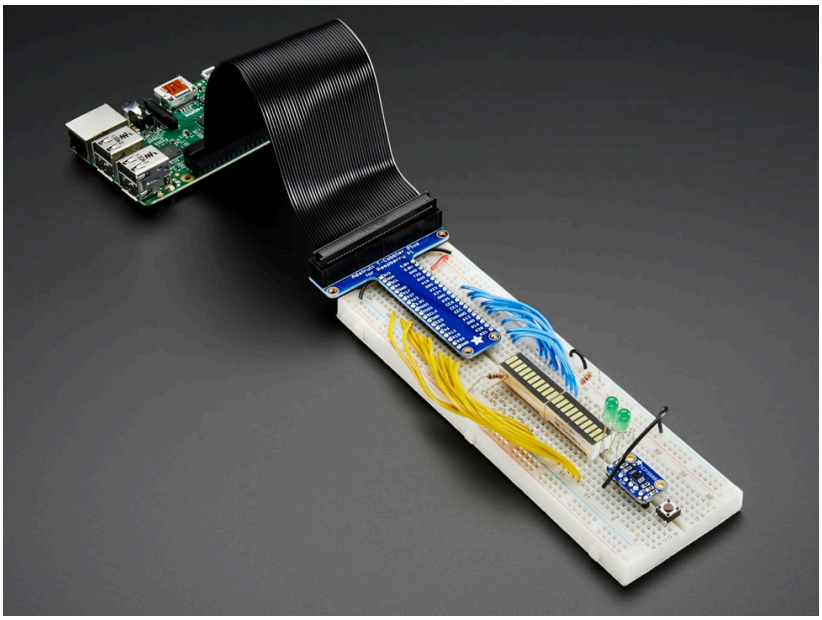


There are lots of different details on this schematic that show all of the chips that go in to making this mini-computer work. However, for our uses, we are only really concerned with the input/output ports here.

Let’s get set up to start working:

- Connect a keyboard and mouse into the USB ports on the right hand side of the board (you can use either the USB 2.0 or USB 3.0 ports; it doesn’t matter for this).
- Connect the provided HDMI to mini-HDMI cable into the *left* mini-HDMI port (“HDMI0”) on the bottom of the board. Connect the other end to an HDMI port on the back of the monitor.
- Using the provided “T-Cobbler,” connect the GPIO ports along the top of the Pi to your breadboard.

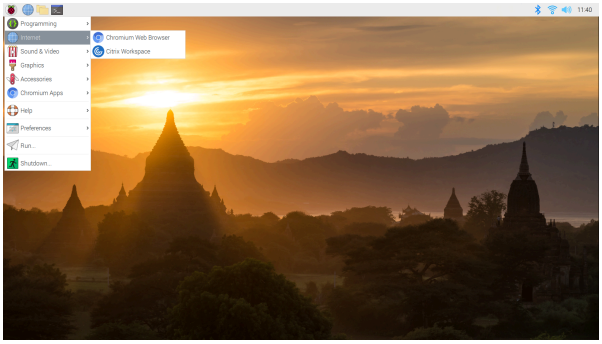
  Two important things:
  - The orientation of the T-cobbler definitely matters. Note the direction that the ribbon cable is installed in the picture below.
  - The pins are *easily bent*. Take great care to install both the ribbon cable and the T-Cobbler straight in so as you don’t bend any pins.



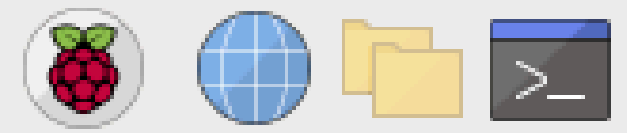
- Connect the provided power brick into the USB-C power port on the bottom left of the board.

Your Pi should display a boot-up screen and eventually load a desktop. You probably won’t need to login, but if you do, the login information is:

Username: elab
Password: Bennington!



The desktop includes a menu along the top with a bunch of different software packages (the little Raspberry menu shown clicked above), a link to a web browser, a link to a file browser, and a terminal window:



Click the terminal window button  to open up a window where you can type commands into the Pi.

1. Try typing “date”, followed by “Enter,” which will give you the current date and time.

We will be working in “interactive mode” for much of what we do today. This may be a little different for those of you with more programming experience, who are likely used to working in a text editor and/or IDE. If you want to do that today, you are welcome to, although you might find the tinkering easier to do in the “interactive mode.”

Part B: Blink the LED

A classic “first project” for a Raspberry Pi is to make an LED blink. To interface with external electronic components, you have to use the GPIO pins on the Raspberry Pi. GPIO stands for **G**eneral **P**urpose **I**nput/**O**utput and, as the name suggests, can be used to send a signal *out* to something like an LED or take a signal *in* from something like an environmental sensor. Of the 40 pins on the “header” of the Raspberry Pi, there are GPIO pins. A map of those pins is shown to the right, on the T-Cobbler interface to the breadboard, and a helpful interactive tool at: https://pinout.xyz/

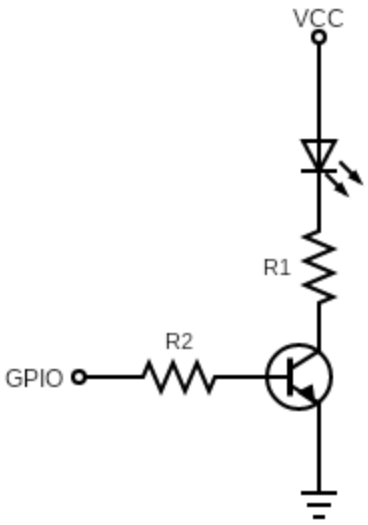
	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

2. Using the interactive tool above, write down in your lab notebook what the other pins are for: “3.3V”, “GND”, “DNC”, “5V.” Also, using the webpage above, which pins are used for “SDA” and “SCL” (digital communication pins we’ll use later)?

The signals that come out of the GPIO pins are *binary* signals, in that the pins can only be **on** or **off**. You need an additional component to receive (or send) analog signals. We’ll see an example in part D.

We are going to use one of the GPIO pins to turn an LED on and off. A GPIO pin is capable of sending out +3.3V, but only at a small current. For this reason, it is “best practices” to use a transistor with our LED[7] and to power our LED from an external source.

[7] Note: A Raspberry Pi GPIO pin can probably power an LED – the limit is something like 50 mA. However, if you want to power anything more than a single LED, you need some kind of transistor. We’re going to use one here because it’s good to get into that habit.



3. Build the circuit shown here on your breadboard. The "GPIO" should be connected to a GPIO pin of your choice. The GPIO pin, while in its "high" state, will hold a voltage of +3.3V. Your "VCC" should be an external voltage source (i.e. your benchtop power supplies) with some chosen voltage – +5V works well. Make sure that the grounds of the power supply and the raspberry pi are "tied together" – it is important to have a "common ground" for all electronic components. Any NPN transistor works here – you could use a 2N3904, since you're familiar with its specs from last week. You will have to do a little math. For R1, use the specs of the LED to determine the maximum current that can flow. To ensure that your transistor is fully saturated, it is often a rule of thumb to give 1/10 of the collector current to the base. Calculate R2 based on that, the expected voltage for the GPIO pin (i.e. +3.3V), and the 0.7V drop between base and emitter.

4. Verify that the circuit at the left works by applying +3.3V from the Raspberry Pi Pin #1 (marked "3V3" on the T-Cobbler) into the GPIO input part of the circuit. If the LED turns on, you built your circuit correctly. After verifying that your LED circuit works, connect the control pin back to an GPIO pin of your choice.

5. On the Raspberry Pi, type:

   ```
   python
   ```

   This will load the interactive python coding environment (called REPL for "Read-Evaluate-Print Loop") that we will work in for this project.

6. Type the following commands to set up the raspberry pi to output to the GPIO pins, pressing "Enter" after each command. The first line of code loads the necessary libraries, the second tells the number scheme to use for identifying GPIO pins (which is the set of numbers identified on the T-Cobbler and the "GPIOXX" numbers identified on the image above – this is the numbering scheme we will be using throughout this lab), while the third sets up the pin of your choice (replace "##" with the pin you've selected) for *output:*

   ```
   import RPi.GPIO as GPIO
   GPIO.setmode(GPIO.BCM)
   GPIO.setup(##, GPIO.OUT)
   ```

   Note: These basic setup commands will persist for as long as you're in python, but you will need to re-type the setup commands if you exit out of the python REPL environment.

7. Turn on the LED by changing the state of the GPIO pin to "HIGH":

   ```
   GPIO.output(##, GPIO.HIGH)
   ```

   This sets the pin of your choice to "HIGH", which is +3.3V.

Now, turn off the GPIO pin:

```
GPIO.output(##, GPIO.LOW)
```

(Pro tip: Press the up arrow repeatedly to go back through past commands so you don't have to type whole commands out again)

We will now blink the light. We first import the "sleep" command, which we use to momentarily pause the loop that we right next. The "while(True)" loop is a loop that will run continuously until we interrupt it.

```
from time import sleep
while(True):
     GPIO.output(##, GPIO.HIGH)
     sleep(1)
     GPIO.output(##, GPIO.LOW)
     sleep(1)
```

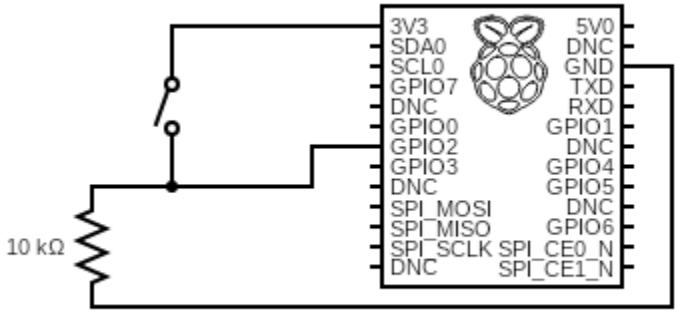
Press "Enter" twice at the end of this code to make the loop that you've created run. Observe the LED. Type Ctrl-C to exit out of the loop. You should still be in the python REPL environment.

Keep your LED circuit wired up because we'll use it in the next parts.

Part C: Button Control of LED

We will now use *two* GPIO pins at the same time – one for output and one for input.

8. Wire up a switch so that it connects one of the GPIO pins (other than the one you're using for the LED) to +3.3V when the switch is closed. Note which GPIO pin you use in your lab notebook. Next wire the same pin you chose through a 10kΩ resistor to ground. This is called a "pull-down resistor." Google "pulll-down resistor" and write a brief explanation of its function in your lab notebook. How does its function differ when the switch is open vs. when the switch is closed? How does it differ from a "pull-**up** resistor"?



You now are using two different GPIO pins – one as *output* to the LED and one as *input* from the switch. We can now write some code so that we can use button pushes to turn the LED on and off.

9. In the python command window, type the commands to set up the input and output pins, where "##" is the pin connected to the LED and "XX" is the pin connected to the switch:

```
GPIO.setup(##, GPIO.OUT)
GPIO.setup(XX, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

   Note the pull_up_down variable when we set up the input for the switch. This is telling the Pi that we are using a pull-**down** resistor and that the pin will be **low** (i.e. 0V) when the switch is open.

10. Next, create the loop that will continually check the switch status and determine whether the switch is closed (such that the pin is "high" at 3.3V) or open (such that the pin is "low" at 0V):

```
while(True):
     if GPIO.input(XX)==GPIO.HIGH:
          GPIO.ouput(##, GPIO.HIGH)
          print('led turned on >>>')
     else:
          GPIO.output(##,GPIO.LOW)
          print('led turned off <<<')
```

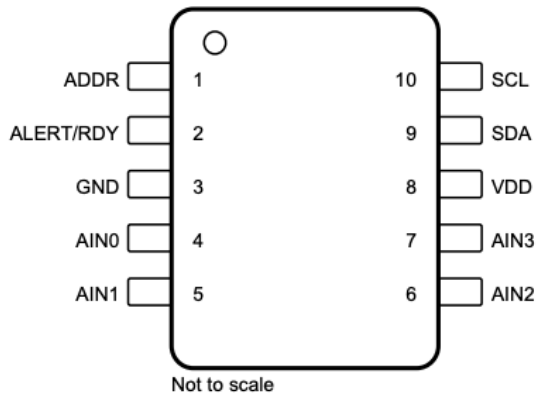
   *Syntax note: when you are finished with the **if** part of the loop, press backspace to go one level up to type the **else** part of the loop.*

11. Reflect in your notebook: Does your circuit work as expected? What is the advantage to a system like this, where the button push is running through a computer, versus directly wiring the switch to the LED?

You can un-wire the switch, if you like – we won't use it again in this lab.

Part D: Reading Analog Signals into the Raspberry Pi

As you saw above, GPIO pins are dual function – they can either read signals that come *in* or send a signal *out.* However, they are only capable of sending ***digital** signals* – the pins can either be *on* or *off*, but can't be in an inbetween state. In other words, the input can only read 0V or +3.3V, but can't distinguish between +2V and +3.3V. As you learned from last week, the signal that comes out of the environmental sensors that we have used so far are ***analog*** signals. This means that they are *continuous* in their values. These values can be something like a *resistance* or *capacitance* (depending on the type of sensor) or a *voltage* if the sensor is integrated with a voltage divider. So, an analog signal can be 1.2V or 1.21V or 1.32V or anything in that range.



In order to *convert* the signal from *analog* to *digital*, you need an Analog-to-Digital Converter – an "ADC." We will be using the very common ADS1115 ADC, a 10 pin integrated circuit that takes up to four analog inputs and converts them into a digital signal for the Raspberry Pi to read. To do this, the Raspberry Pi will use the I2C protocol (more on this below).

The ADS1115 ADC is an integrated circuit (basically a collection of electronic components in one package) that can take analog signals and convert them to digital signals that the Raspberry Pi can read. The ADC layout is shown here, although we use a version of this ADC that is packaged onto a "breakout board" that makes it easier to access the pins on a breadboard.

12. Look up the data sheet for the ADS1115 and, to the best of your ability, write in your notebook what each of the pins do.

In order to let the ADS1115 communicate with the Raspberry Pi, they need to speak the same protocol – essentially the same "digital language." There are a few different common ones, but we will be using the "I2C" protocol. But first, we need to connect four wires between the ADC to the Pi – two for power and two for communication. Place the ADC on your breadboard (straddling the center, probably) and:

- Connect the +3.3V pin from the Pi to the VDD wire in the ADC.
- Connect a GND pin on the Pi to the GND wire on the ADC.
- Connect the SCL Pi pin to the SCL ADC pin (this is the "serial clock" pin and is part of the data communication).
- Connect the SDA Pi pin to the SDA ADC pin (this is the "serial data" pin and is part of the data communication).

Within your Python REPL environment, run the following code to import the necessary modules for communication over I2C and initialize the I2C communication bus:

```
import board
i2c = board.I2C()
```

Next, import the module for the specific ADC we're using – the ADS1115:

```
from adafruit_ads1x15 import ADS1115, AnalogIn, ads1x15
```

Next, we will create an "object" (a programming structure) that will allow us to communicate with the ADC:

```
ads = ADS1115(i2c)
```

Now, we're ready to start getting data. Tell your Pi which pin on the ADC you want to read (replace the "A0" below if you are going to use a different analog input pin on the ADC):

```
chan = AnalogIn(ads, ads1x15.Pin.A0)
```

Next, print out the reading from the pin:

```
print(chan.value, chan.voltage)
```

13. Record the readings, both the "value" (which is the number from 0 to 65535) and the "voltage," which is the voltage that the ADC measures.

14. Power on your benchtop power supply and set it to around +2V. (**VERY IMPORTANT: DO NOT FEED MORE THAN +3V into the ADC. YOU MAY FRY IT).** Make sure the benchtop power supply shares a common ground with the other components and tie its positive side into the analog input on the ADS1115. Take another measurement and record the results in your lab notebook:

    ```
    print(chan.value, chan.voltage)
    ```

    Does the reading match what you are expecting?

Pick out an environmental sensor of your choice. The ones that work well for this lab include:

- Light sensors (a.k.a. photodiodes)
- Pressure sensors
- Temperature sensors (although you may have to get a little creative about what temperatures you're sampling).

Wire up your environmental sensor in a voltage divider such that the voltage goes up with more stimulus (i.e. higher temperature, higher pressure, higher light) and down with less stimulus. You will power the voltage divider with the benchtop power supply set to 5V. Think carefully whether you should put your sensor in the upper location or the lower location. Remember to use a common ground! Choose your voltage divider resistor so that your output voltages from the voltage divider range from 0V to **no higher than +3V** over the reasonable range of the voltage divider.

15. Draw a sketch in your lab notebook of your voltage divider circuit and explain briefly why you put the environmental sensor in the position you placed it. Show your work in calculating the resistor to use and the range of output voltages expected.

Verify with a multimeter that the output voltages range over the range expected with the full range of external stimuli – this is a very important step so that you don't send too high of a voltage into the ADC and fry it.

16. Now wire the output of your voltage-divider circuit into your chosen input GPIO pin. From the terminal interface, print the voltage again (remember the up arrow trick). Does it match what you measure with a multimeter?

17. Vary the range of external stimuli, taking voltage measurements each time. Make a table in your lab notebook recording the voltage of several different external stimuli.

Leave your sensor circuit wired up for the next part of the lab.

Part E: Input and Output Together

Conventional incandescent lightbulbs can be dimmed by reducing the voltage across them, thereby reducing the current through the filament. Because LEDs work at a fixed voltage, they cannot be dimmed in this way. Dimming in LEDs is accomplished by "Pulse Width Modulation" – effectively blinking the LED faster than your eye can sense. The longer that the LED is on in each cycle, the brighter it appears to our eyes.

The Raspberry Pi is capable of sending a PWM signal through its GPIO pins, although it only works through pins 12, 13, 18, and 19.

18. In the python REPL environment, ensure that all necessary libraries are loaded and the code knows how we're referring to the GPIO pins:

    ```
    import RPi.GPIO as GPIO
    from time import sleep
    import board
    from adafruit_ads1x15 import ADS1115, AnalogIn, ads1x15
    GPIO.setmode(GPIO.BCM)
    ```

19. Choose one of the PWM-enabled pins and wire it up to the base of the transistor in your LED circuit. Using the same setup that you did in part B, verify that you can turn on the LED. Turn off the LED.

20. Next, set up the pin for output and to enable PWM:

    ```
    pwm = GPIO.PWM(##, 1000)
    pwm.start(50)
    ```

    The first line enables PWM on that pin and sets a flash rate (1000 Hz; well above the frequency that a human can see), while the second line starts the flashing and defines the "duty cycle" – what percentage of the time the light should be on (i.e. 50% in this case).

21. Change the duty cycle with the command:

    ```
    pwm.ChangeDutyCycle(75)
    ```

    What happened to the light's brightness?

Turn the PWM off all together with the line:

```
pwm.stop()
```

Finally, we will use the input signal from the environmental sensor and the output to the pulse width modulated LED, to create a circuit where the brightness of the LED depends on the strength of the external stimulus. We have to do a bit of math in the python code first.

22. Write down in your notebook the *full range* of the voltage your sensor works over. This is probably something close to 0V to 3V, but look back to the table you made in the previous part to give you some idea. Assign that maximum voltage to the variable v_max on the Pi:

    ```
    vmax = XXX
    ```

23. Create a variable that changes from 0 to 100 based on the fraction of the maximum voltage currently being measured:

    ```
    vfrac = (chan.voltage/vmax)*100.0
    ```

24. Turn on the LED with PWM and set the "duty cycle" of the PWM to this fractional voltage:

    ```
    pwm = GPIO.PWM(##, 1000)
    pwm.start(v_frac)
    ```

25. Change the external stimulus (i.e. shine a brighter light, put on more pressure, change the temperature), update the value of v_frac, and change the duty cycle of the PWM:

    ```
    v_frac = (chan.voltage/v_max)*100.0
    pwm.ChangeDutyCycle(v_frac)
    ```

    What happens to the light?

Finally, put it in a loop so that the light is continuously being updated:

```
while(True):
     v_frac = (chan.voltage/v_max)*100.0
     pwm.ChangeDutyCycle(v_frac)
```

26. Verify that the brightness of the light changes as you vary the conditions of the sensor. Take a short video of your circuit working and email it to Hugh (hcrowl@bennington.edu).

27. Shut down your Pi and disconnect all your components, placing them back where you got them.