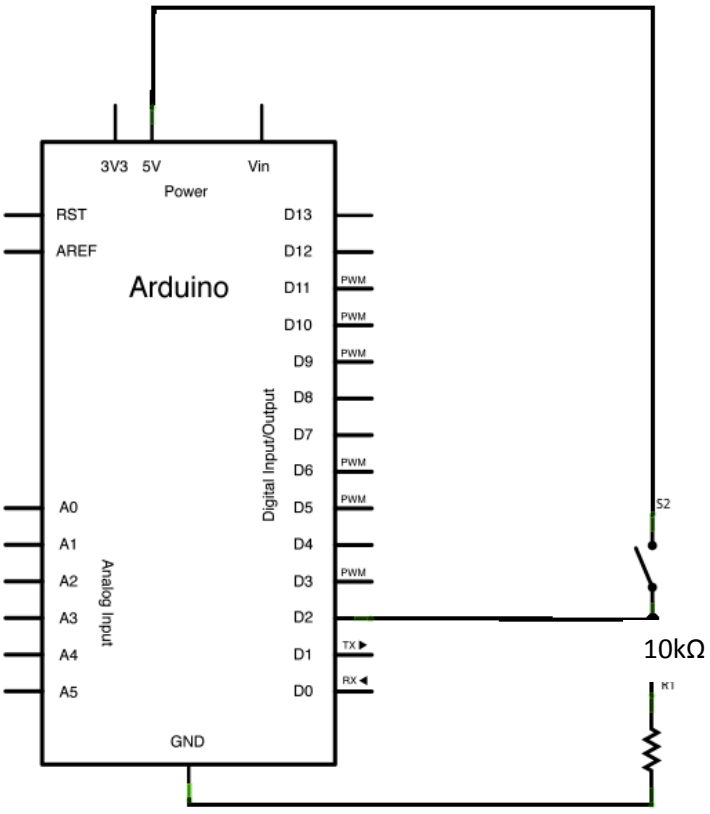
CSE 102 – Studio 2

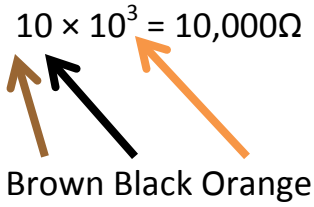
This week we are expanding the I/O capabilities of the Arduino board to include digital inputs and analog inputs. One of the interesting things with both of these two additional capabilities is that they both have aspects of how they are physically implemented that are frequently surprising to folks when they first learn about them. It is interesting how a conceptually simple thing might have pretty substantial details "under the hood" as it were, yet still provides a neat clean abstraction to users.

Digital Inputs

You can physically construct a digital input using a push-button switch and one resistor. Connect the push-button switch between the power supply (+5V) and the input pin, and connect a 10kΩ resistor between the input pin and ground (GND). The resulting circuit is illustrated in the schematic below:



A 10kΩ resistor will have a color code as follows:



Recall that the table of values is as follows:

0	1	2	3	4	5	6	7	8	9
black	brown	red	orange	yellow	green	blue	violet	grey	white

Do you like memory mnemonics? Here's one: **Better Be Right Or Your Great Big Venture Goes West**

Write a program called `mirror` that reads the state of the push-button input (using the function `digitalRead()`) and sends it out to an LED digital output. The program should repeat this operation continuously.

Extend this program to count the number of times the push-button input is pressed. You can detect a press either on the 0 to 1 or the 1 to 0 transition of the digital input port. Each time the count increases, send an output string that includes the current value of the count to the desktop machine for display.

Does the count seem high at times? Why might this be the case?

Ask an instructor or TA to show you the digital input waveform on an oscilloscope. How does this help explain the count?

Write a new program called `debounce` that reads the state of the push-button using proper debouncing techniques. How to do this is covered in this Tutorial from Arduino.

Check that your push-button is appropriately debounced by repeating the counting experiment above. What debounce delay is necessary for your push-button to be reliably debounced?

Stoplight Controller

Building on the stoplight controller from Studio 1, you will now extend it to include a pedestrian walk signals. First, reconstruct the controller from last week. If you didn't make it to the stoplight exercise, here is code that you can use. Run the program `stoplight` to ensure everything is in working order. Next, add two additional LEDs as digital outputs (a green one for "walk" and a red one for "don't walk"). They should be constructed in the same way as the original 6 LED digital outputs. Also include your push-button digital input from above.

Alter the `stoplight` code to meet the following specifications:

1. When the "walk" indicator is lit, both traffic signals should be red.

2. Conversely, when any traffic signal is not red (either green or yellow), the "don't walk" indicator should be lit.
3. When the push-button is momentarily pressed, the traffic light cycle should continue until the end of the yellow on the north/south street. At this point, both traffic signals should go to red and the "walk" indicator should be lit for an appropriate period of time. (Note: this corresponds to all pedestrians having the right-of-way and all traffic stopped.)
4. At the end of the "walk" signal, the traffic light cycle should resume where it left off (green for the east/west street).

Analog Inputs

We will be using a TMP36 temperature sensor manufactured by Analog Devices. The pinout is shown on Figure 4 of the datasheet, which is replicated below.

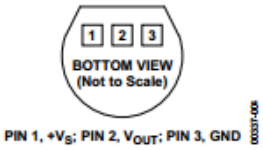


Figure 4. T-3 (TO-92)

You should connect pin 1 to +5V, pin 2 to an analog input pin, and pin 3 to GND.

Write a program called `temperature` that reads the temperature sensor and prints the value to the desktop computer. If using the `INTERNAL` analog voltage reference of 1.1V,

```
analogReference(INTERNAL);
```

a reasonably well calibrated temperature signal can be realized with the following:

```
temp = (float)(analogRead(analogInPin))*100*1.1/1024 - 50;
```

Using the PC for a display, show both unfiltered and filtered versions of the temperature. It also might be informative to show raw counts from the `analogRead()` function as well. How much filtering (i.e., how many values need to be averaged) is necessary to get a reasonably stable reading?

Change the ambient temperature of the sensor and take notice of how quickly the readings change (both the unfiltered and filtered values).

Using the graph of Figure 6 in the datasheet, can you recreate the calibration code we provided above?