Arduino for Biologists
Class Assignment #1

Digital Output

1) LED Blink
If you haven't done so already, install the Arduino software on your laptop. Next, download the program named *Basic_Blink* from the class website: https://github.com/HMS-RIC/ArduinoNanocourse
Now upload it to your Arduino and make sure it runs.

The *Basic_Blink* program makes the Arduino's onboard LED blink on for one second, then off for one second, repeatedly. Modify the code to change the blinking pattern, e.g., make it blink twice in quick succession (250 ms on, 250 ms off, 250 ms on) and then pause for 2 seconds before repeating.

Now upload this new program to the Arduino and confirm that it works as you intended.

2) External LED
The program in part 1 made use of the Arduino's onboard LED. Now we will build a simple circuit to drive a second, external LED. Use your breadboard to wire up the following circuit:

To build the circuit, all you will need are one of the colored 5mm LEDs (the kind with two leads), a 330 Ω resistor, and some jumper wire.

Test that your circuit works by modifying the *Basic_Blink* program to use pin 12 (rather than 13). Once you have demonstrated that your LED circuit works, modify the program so that the two LEDs (on pins 12 and 13) blink on alternate cycles. Save this program as *ExternalBlink*.

USB communication with a PC

3) Writing to the PC over USB
One powerful feature of the Arduino is its ability to communicate over a USB connection to a computer. In many projects, an Arduino will send data (sensor values, timing info) to a PC, or the PC will send commands and parameter values to the Arduino.

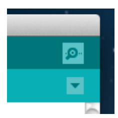
Just as important is the ability to use the USB connection for debugging. By now you may have encountered a situation where the Arduino did not behave as you expected. In these situations it's often useful to have the Arduino report its internal state to help you debug what went wrong.

To simulate a debugging situation, try modifying *ExternalBlink* so that it writes the string "LED on" or "LED off" to the USB port every time the LED changes state. You will find the following commands helpful:

```
Serial.begin(9600);  // this should be run once, in setup()

Serial.println("Some Text"); // this goes anywhere you want
                             //  to output text
```

Open up the 'Serial Monitor' on your computer to confirm that the messages are being sent. (You can open the Serial Monitor by clicking the magnifying glass button on the top right corner of the Arduino window.):



Next, we will output the value of a numeric variable over USB. This functionality is particularly important for debugging variables whose values might change during the course of the program's run.

Add the following lines within `loop()` to initialize a variable called `elapsed_time` and use the `millis()` command to set its value to the elapsed time, measured in milliseconds, since the Arduino started running:

```
unsigned long int elapsed_time;  // declare a variable
                                 // to store time
elapsed_time = millis(); // get the elapsed time
                         // in ms since startup
```

Now, with the help of the Arduno reference pages, figure out how to print out the current elapsed time ever time an LED turns on. Try to format it nicely with some explanatory text, all on the same line if possible. E.g.,:

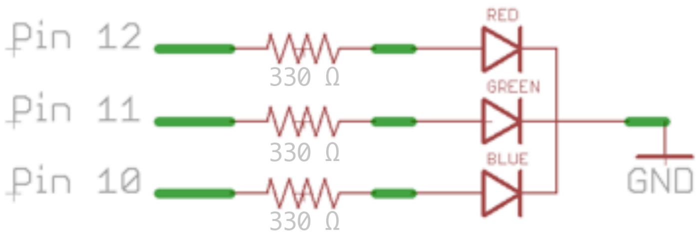
```
"Elapsed Time: XXXXX ms"
```

You should refer to the Arduino reference site to learn more about the commands `Serial.print()` and `Serial.println()`:

www.arduino.cc/en/Serial/Print
www.arduino.cc/en/Serial/Println

Functions

4) RGB LED
Let's make things even more fun by using an RGB LED. This is really just three LEDs — a red, green, and blue — squeezed together in one package. To keep the wiring simple, the three LEDs are packaged with a common cathode (negative terminal). Try to wire up the circuit below and then use pins 10–12 to drive the three colors independently. Make a new blink cycle: red-green-blue-white-off. Save this program as *BlinkRGB.*



5) Functions
The code for generating different colors from the RGB LED was probably starting to look a bit messy with all the different `digitalWrite()` statements. For example, suppose you wanted to reverse the color order or to add yellow (red + green) in between each of the existing colors. How easy would that be? Would it be faster to start again from scratch?

Writing a function allows you to simplify your program in two significant ways:
- It segregates away low-level details of a particular action (i.e., which pins should be modified in order to get red light).
- It allows you to reuse the same code in multiple locations of your program.
Write five functions (for red, green, blue, white, off) in the following form:

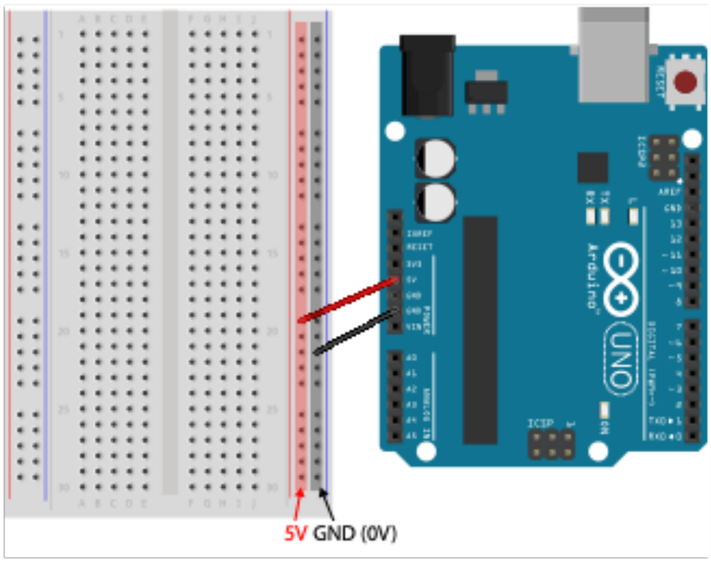
```
void LED_red() {
     // configure the three pins to output Red
     digitalWrite ....
}
```

Now rewrite `loop()` so that it calls these functions. Your `loop()` should have no more direct calls to `digitalWrite()`. Re-save this new and improved version of *BlinkRGB.*

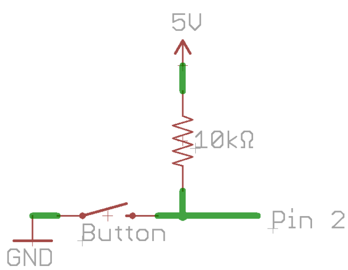
Now reverse the order of the colors. Easy, right?

Digital Input

Going forward, you will be making multiple connections to ground and power (5V). To simplify the wiring (and make debugging easier), please wire up the power (red) and ground (black/blue) busses on the side of your breadboard, as shown below:



6) Pushbutton
Let’s try to add a user control so that the RGB LED only lights up after a button is pressed. Wire up a pushbutton to pin 2 by following this schematic:



In the next lecture, we will learn that at the button's natural state (open circuit), this circuit will “pull up” the value on pin 2 to 5 V. But when the button is pressed, the switch brings the voltage on pin 2 to ground (0 V). For the Arduino to detect this change, we first need to define pin 2 as a digital input pin. Add the following lines to *BlinkRGB*:

Insert this line *before* the `setup()` function:

```
int pushbutton_pin = 2;
```

Insert this line inside the `setup()` function:

```
pinMode(pushbutton_pin, INPUT);
```

Now try to modify the `loop()` function so that the LED remains off by default and cycles through the colors (red-green-blue-white) when a button press is detected. You will need to use the `if` statement and the `digitalRead()` function; You can read how to use this function on the Arduino reference website:

    www.arduino.cc/en/Reference/DigitalRead

The syntax for the `if` statement is here:

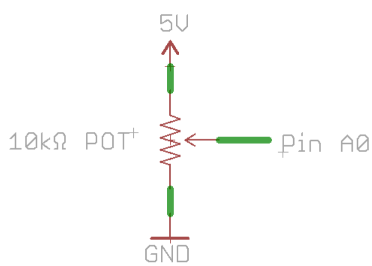
    www.arduino.cc/en/Reference/If

Save this new program as *RGBButton.*

Analog I/O:

7) Potentiometer Input

Connect a potentiometer (variable resistor) to your Arduino and wire it up to pin A0 as in the following schematic.



The central pin of the pot should be connected to A0, the two outer pins can be wired to 0 and 5 V in either order. This circuit provides an input voltage between 0–5 V, depending on the position of the knob. Now write a new program that reads in the analog value on A0 and then outputs the voltage to the USB bus once a second:

First read the help page for the `analogRead()` command to learn how the command works and get sample code that will print out the raw values:

    www.arduino.cc/en/Reference/AnalogRead

(You might need to make a small change to the sample program to get it to work with your circuit.)

Next, modify the code so that it outputs voltages, rather than raw values. Remember that the result of an integer division is an integer; you may want to store the raw value as a floating point type before converting to voltage.

Save this program as *AnalogToSerial*.

8) Pseudo-analog Output (Pulse Width Modulation)
The Arduino can only output digital signals (5V or 0V), but it can approximate analog output signals by rapidly switching between 0 and 5 V. Read the help page for `analogWrite()` to understand how to make use of this on the Arduino:

    www.arduino.cc/en/Reference/AnalogWrite

You may also be interested in a better explanation of PWM (pulse width modulation):

    www.arduino.cc/en/Tutorial/PWM

Only pins marked with a '~' are capable of generating PWM outputs. Connect an LED to one of these PWM-enabled digital pins (remember to use a 330Ω current-limiting resistor). Modify your code from the previous exercise so that the brightness of the LED is modulated by the analog potentiometer input.

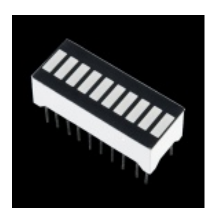
Build Your First Instrument:

9) Primate Reaction Time Experiment
Congratulations, you now have enough experience to start making scientific instruments with the Arduino!

Behavioral scientists often need to measure the reaction time of their experimental subjects. For your final exercise, you will build a device that delivers a flash of light every 30 seconds. The device should measure the time it takes a human subject to detect the flash and press a button in response. Finally, the device should print out the exact response time to a computer and also give a coarse visual indication to the subject if his/her response was fast or slow.

You can build this however you like. You may want to look at the timing-related commands on the Arduino reference website.

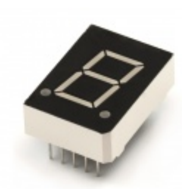
BONUS: 10) Multi-segment LEDs



Now wire up a 10-segment bar LED to your Arduino. You can use a ten-channel 330Ω resistor net to avoid wiring up 10 separate resistors (ask one of us to explain how they work). Next write a function called

```
void BarDisplay(int numSegments) {...}
```

that turns on the first 'numSegments' segments of the bar. Finally, call this function every 100 ms based on the potentiometer's voltage reading (scaled from 0-10).



Alternatively, instead of the bar LED, try outputting the digits 0–9 on a 7-segment numeric LED display. (Unfortunately, you can't use resistor nets for this circuit, so you will need ~7 Ω resistors.)

Finally, add the multi-segment LED (bar or numeric) to your reaction time device to provide a higher resolution indicator of response speed.

BONUS: 11) If you're still having fun:
Try out some of the examples provided by Arduino. Open an example sketch in *File->Examples* and wire up the described circuit. Then run the sketch and try to understand how the code works.

* * *

List of parts needed to complete this assignment at home:

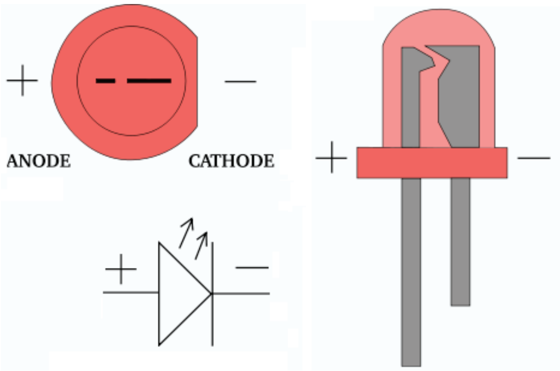
- Arduino with breadboard and USB cable
- Jumper wires (~10x)
- 10k Ω Resistor
- 330 Ω Resistors (~4x)
- Single-color LED
- RGB LED
- Pushbutton switch
- Potentiometer

For Bonus Exercise:

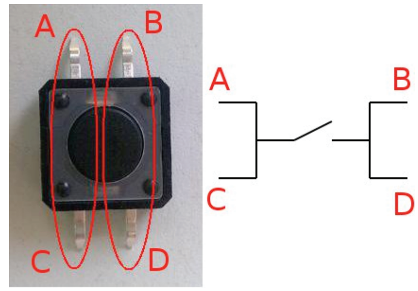
- 10-Segment Bar LED
- Resistor net

  OR
- 7-Segment Numeric LED
- 330 Ω Resistors (8x)

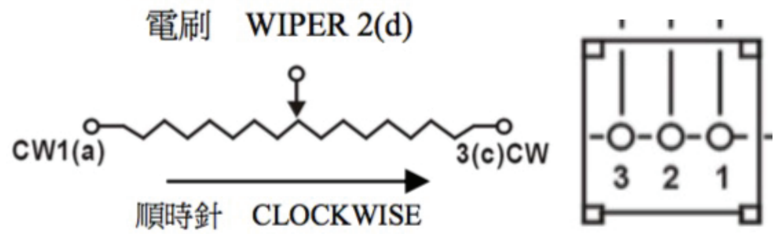
LED



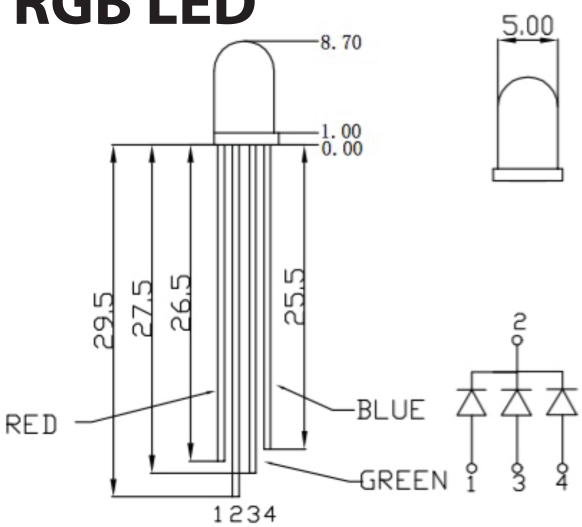
Pushbutton



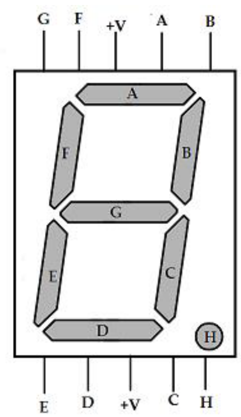
Potentiometer



RGB LED

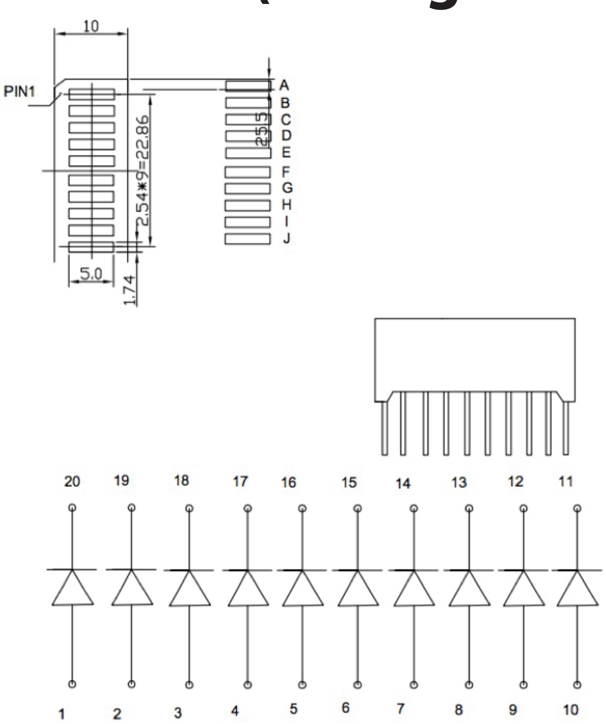


Numeric LED (7-segment)

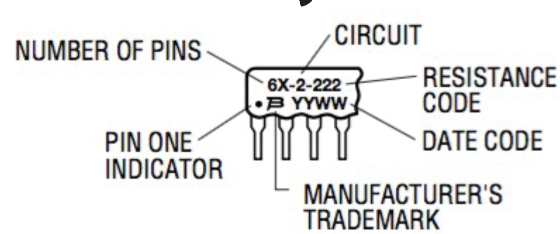


These actually have 8 functional segments, but are still called "7-segment LEDs" for historic reasons. The right decimal point is a newly-added feature. (And the left decimal point is not wired up.)

Bar LED (10-segment)



Resistor Array (or Net)



Bussed Resistors (101 Circuit)

Model 4600X-101-RC
4 through 14 Pin