Techshop EEE-105-Arduino-1: Basic programming

Chapter 3

Up to this point, we've seen how to use the serial monitor and how to control the output of a single digital pin. There isn't a whole lot we can do with just that, so let's learn some more cool stuff. In this chapter we're going to read inputs, as well as cover the idea of an analog value; we'll cover the following topics:

- Analog input
- Fading a LED with PWM
- variable type `int`
- `if` control structure

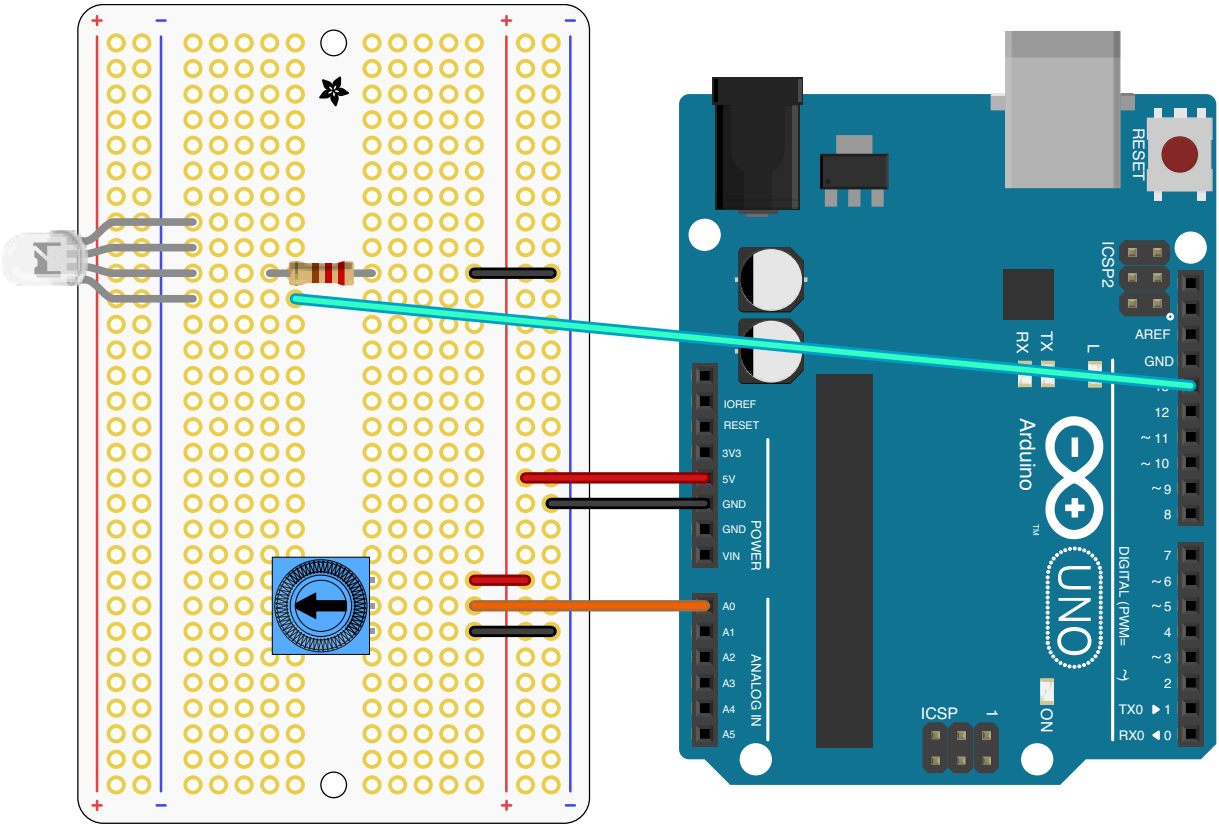
Table of contents

- Part 1 - analog input
- Part 2 - PWM digital output
- Part 3 - Putting it all together

Part 1 - analog input

In Chapter 2 we used the **output** of a digital pin to control the blinking of a LED; in this first part we are going to be looking at inputs. But not just any type of input, we're going to be using an analog input. Previously, we

see that we could set the digital pin to either `HIGH` or `LOW`; that is, it could be set to only two values. An analog pin is different in that it can have many values, in our case, 1024 to be exact! Let's begin with an example; the circuit we want is shown below, and the sketch your after is AnalogInput.ino.



Let's break the sketch down by looking at the top part first.

```
byte sensorPin = A0; // select the input pin for the trimpot
byte ledPin = 13;      // select the pin for the LED

// variable to store the value coming from the sensor
int sensorValue = 0;

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
  // because the analog pins are always INPUT, we
  // don't have to declare it as such
}
```

In the first three lines, we declare three variables, two `byte` types and one `int` type. Previously, we saw that a `byte` is a number that can be any value between 0 and 255 (inclusive); `int` (which stands for integer) is another type of number, but it's much bigger; it's a 16-bit number which equates to 2^16.

Bust out the calculator and you'll see that equals 65,536! However, an `int` is special in that it can hold negative values, in fact an `int` can hold the values -32,768 to 32,767; for those that are observant you can see that 32,767 + 32,768 = 65,535 (can anyone tell me what that isn't exactly equal to 2^16?). The reason we want to declare the variable `sensorValue` as an `int` is because the range we expect it to be reading from is 0 to 1023 (the range of the analog pins).

In the `setup()` function we see that we are setting the `ledPin` (pin 13) as an `OUTPUT`. Note that we don't have to define the analog pin as an input because it can only act asn an input. Let's move on to the interesting parts:

```
void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);

  // turn the ledPin on
  digitalWrite(ledPin, HIGH);

  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);

  // turn the ledPin off:
  digitalWrite(ledPin, LOW);

  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
```

```
}
```

In the `loop()`, we see a function we haven't seen before `analogRead()` - this is the function which will be reading the input values on our analog pin (pin A0). As we mentioned previously, we expect this value to be anywhere between 0 and 1023 (inclusive). The remaining lines of code we've seen in Chapter 2: we turn our LED on with `digitalWrite()`, then we pause the sketch by a value of `sensorValue` number of milliseconds, then we turn off our LED with `digitalWrite()` and then we pause again. The interesting part here is that, instead of pausing with a fixed amount, we can control the length of pause with the value that is read by `analogRead()` which we control with the trimpot. That means, we can directly control how fast our LED blinks just by twisting the trimpot to different values!

Explore: how would we change our sketch to see what value our variable `sensorValue` is set to?

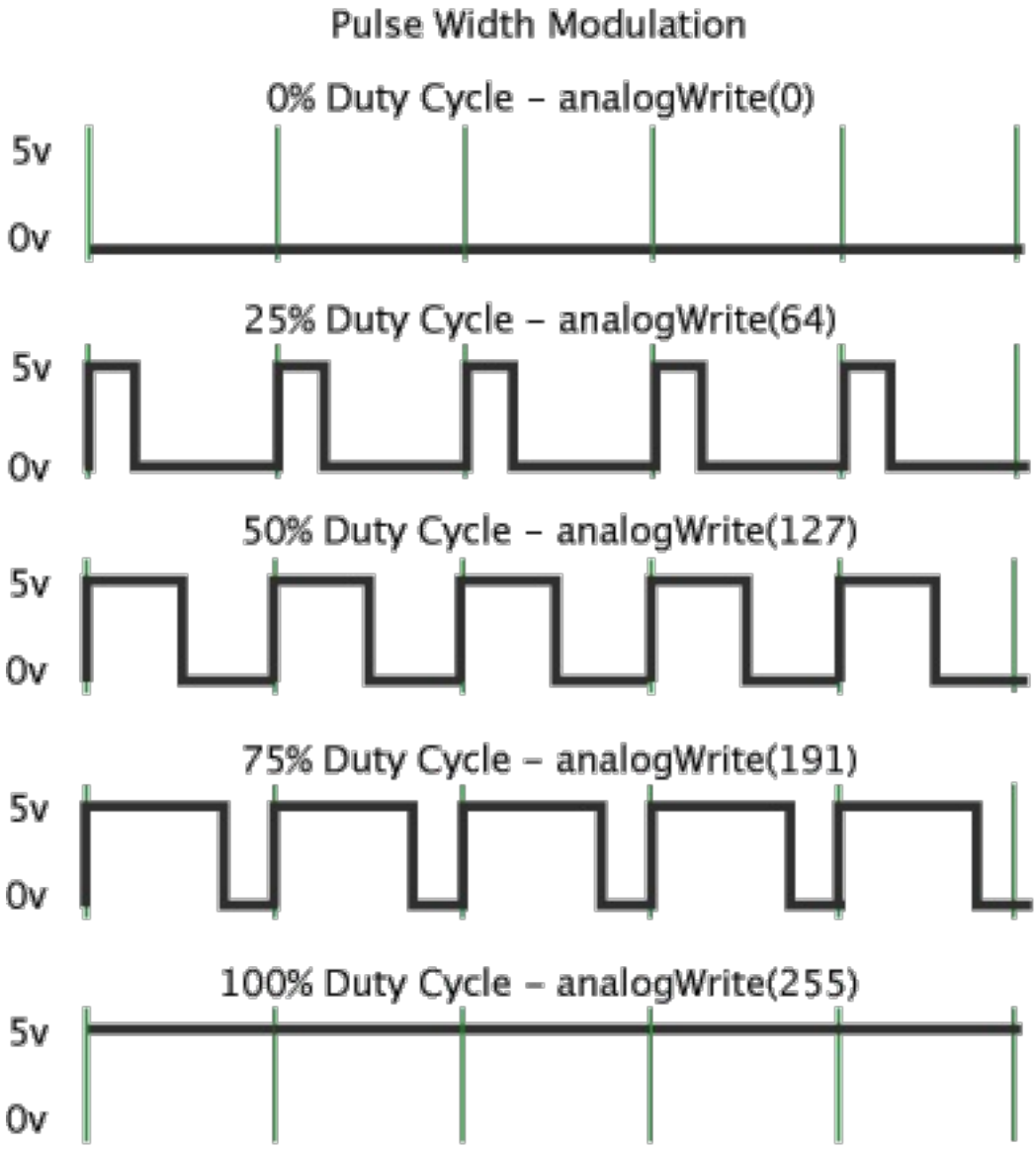
Part 2 - PWM digital output

In part 1 we covered analog inputs, but what about analog outpus? Unfortunately the Arduino can't directly do analog outputs, however we can fake it through the use of the Arudino software through a technique called pulse width modulation (PWM):

> Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called

the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.



Enough with the theory, let's take a look with a hands on example, upload the Fade sketch to your Aruino.

The only new part in this sketch is the use of the `analogWrite()` function (*make sure you set the* `ledPin` *variable to a digital pin that support PWM such as 10 or 11*):

```
void loop() {
  analogWrite(ledPin, 0); // update analog value on ledPin
  delay(500); // delay 500 miliseconds

  analogWrite(ledPin, 50); // update analog value on ledPin
  delay(500); // delay 500 miliseconds

  analogWrite(ledPin, 100); // update analog value on ledPin
  delay(500); // delay 500 miliseconds

  analogWrite(ledPin, 150); // update analog value on ledPin
  delay(500); // delay 500 miliseconds

  analogWrite(ledPin, 200); // update analog value on ledPin
  delay(500); // delay 500 miliseconds

  analogWrite(ledPin, 255); // update analog value on ledPin
  delay(500); // delay 500 miliseconds
}
```

All we're doing is generating an analog output through PWM at various frequencies. We start it at 0% duty cycle (LED is off), and then increase the frequency slowly, pausing for half a second at each frequency change. What you should be seeing is that the LED slowly increases in brightness as the PWM frequency is increased.

Explore:

- how would you make the LED increase in brightness faster?

Part 3 - Putting it all together

So far we've completely ignored the fact that the LED we've been using is actually 3 LEDs built into one. So, for this last part of this class, we are going to put everything we learned together in order to control all three colors of the LED. But before we do that, we need to learn one more thing: the `if` control statement.

An `if` statement allows us to run certain parts of code only *if* a condition, or multiple conditions, are true. An example might be: *turn LED on, if button 1 is pressed down*; the general format looks like this:

```
if (condition is true)
{
  // do something
}
```

The condition statment uses *comparison operators* and *boolean operators* to evalute whether a condition is true. Let's take a look at those.

Comparison operators

These allow us to check a single condition in multiple ways; the following operators are available in Arduino:

> x == y (x is equal to y)
>
> x != y (x is not equal to y)
>
> x < y (x is less than y)

> x > y (x is greater than y)
>
> x <= y (x is less than or equal to y)
>
> x >= y (x is greater than or equal to y)

So, for example, if we were reading an analog value and storing it to the variable `analogVal`, and we wanted to turn a LED on only if the `analogVal` was larger than 500, we'd write something like:

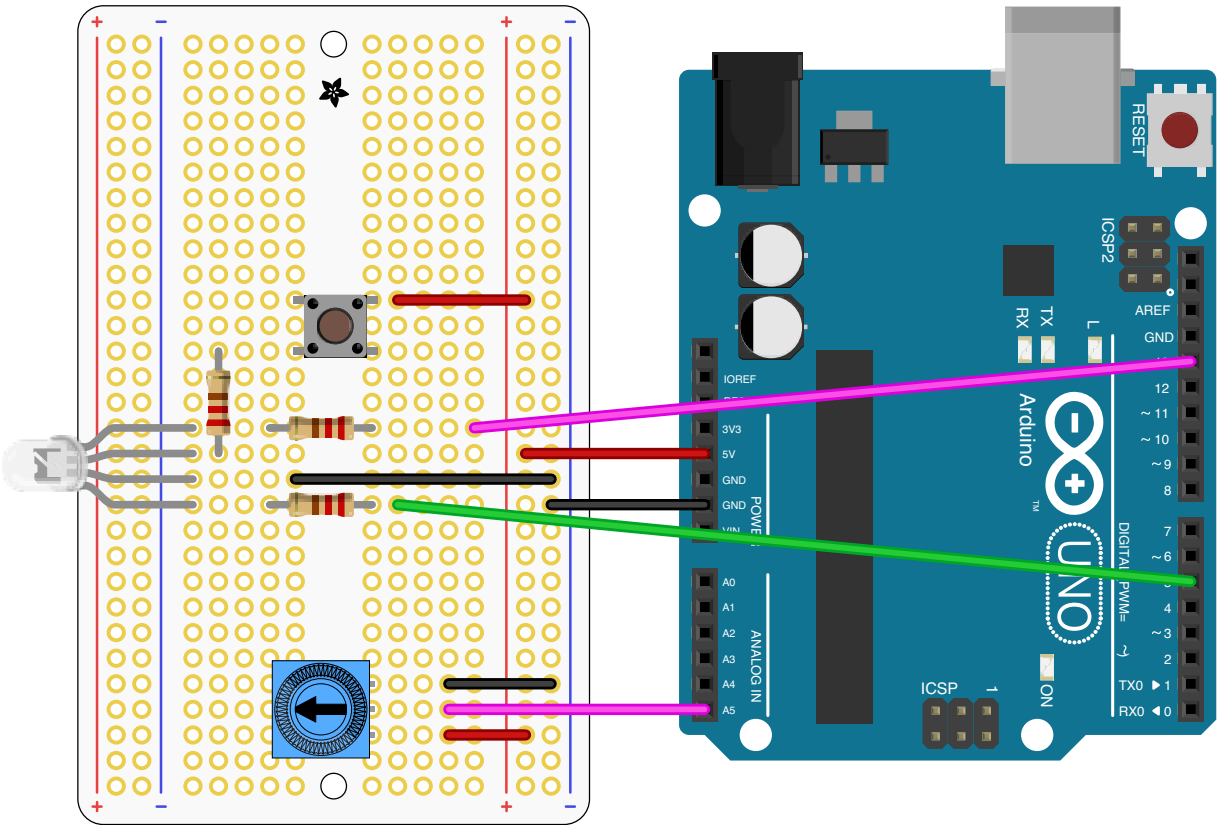
```
if (analogVal >= 500)
{
    digitalWrite(ledPin, HIGH); // turn on LED
}
```

Boolean operators

These allow us to chain multiple comparison operators together through the use of a boolean `AND`, `OR`, or `NOT`. To exend the above example, let's say we only want to turn the LED on when the `analogVal` is between 500 and 700, we'd write:

```
if (analogVal >= 500 && analogVal <= 700)
{
    digitalWrite(ledPin, HIGH); // turn on LED
}
```

Let's get to it! The circuit you need to build is seen below and the sketch we're going to use is the Fadelf.ino sketch.



As you can see in the circuit, we've now hooked up all three color components of the LED:

- green (G) is connected to a digital pin (13)
- blue (B) is connected to the trimpot
- red (R) is connected to a digital PWM pin (5)

The top part of the sketch should be pretty self explanatory by now, so let's go through the `loop()`:

```
// the loop routine runs over and over again forever:
void loop() {

  analogVal = analogRead(A5); // read analog value

  // turn green LED on if analogVal is greater than 300,
  // otherwise turn if off
  if (analogVal >= 300) {
    digitalWrite(ledG, HIGH); // turn green LED on
  } else {
    digitalWrite(ledG, LOW); // turn green LED off
```

```
  }


  // set the brightness of the red LED
  analogWrite(ledR, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading
  // when brightest or dullest
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }


  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

There's a lot going on here, so let's break it down; the code is controlling the state of both the green (`ledG`) and red (`ledR`) LEDs:

Green LED

We begin by reading the analog value on pin A5 and setting it to the variable `analogVal`; you'll notice it's hooked up to the trimpot, so as you adjust it, the `analogVal` will change. Then we encouter our `if` statement; it's setup to turn the green led (pin `ledG`) ON if the `analogVal` is larger than or equal to 300; otherwise it will turn it off. So far so good.

Red LED

The red LED is a bit more complicated. We are using PWD (`analogWrite()`) to set the brightness with a variable named

`brigthness`. On our first loop, `brightness` is 0; however, on each loop we adjust the `brigthness` value by adding `fadeAmount` (which equals 5) to it. BUT! If the `brigthness` is larger than 255 (or less than 0) then we change `fadeAmount` to be the negative of itself. So, when `fadeAmount` is 5, it would change to -5 and *vice a versa*. What this means, is that on every loop, we change the `brightness` value and thus the PWM duty and thus the brightness of the red LED.

Practically, what this all means is, you should see the red LED slowly getting brighter and then fading AND depending on the value for `analogVal` the green LED will be on or off. This way you can mix colors! AND! the blue LED is hooked up to the push button switch so you can manually control it.

Explore:

- How can we see what voltage is being set by the trimpot?
- How can we make the LED show a white color? How about purple?
- How could we turn all three colors on/off with a single pin?