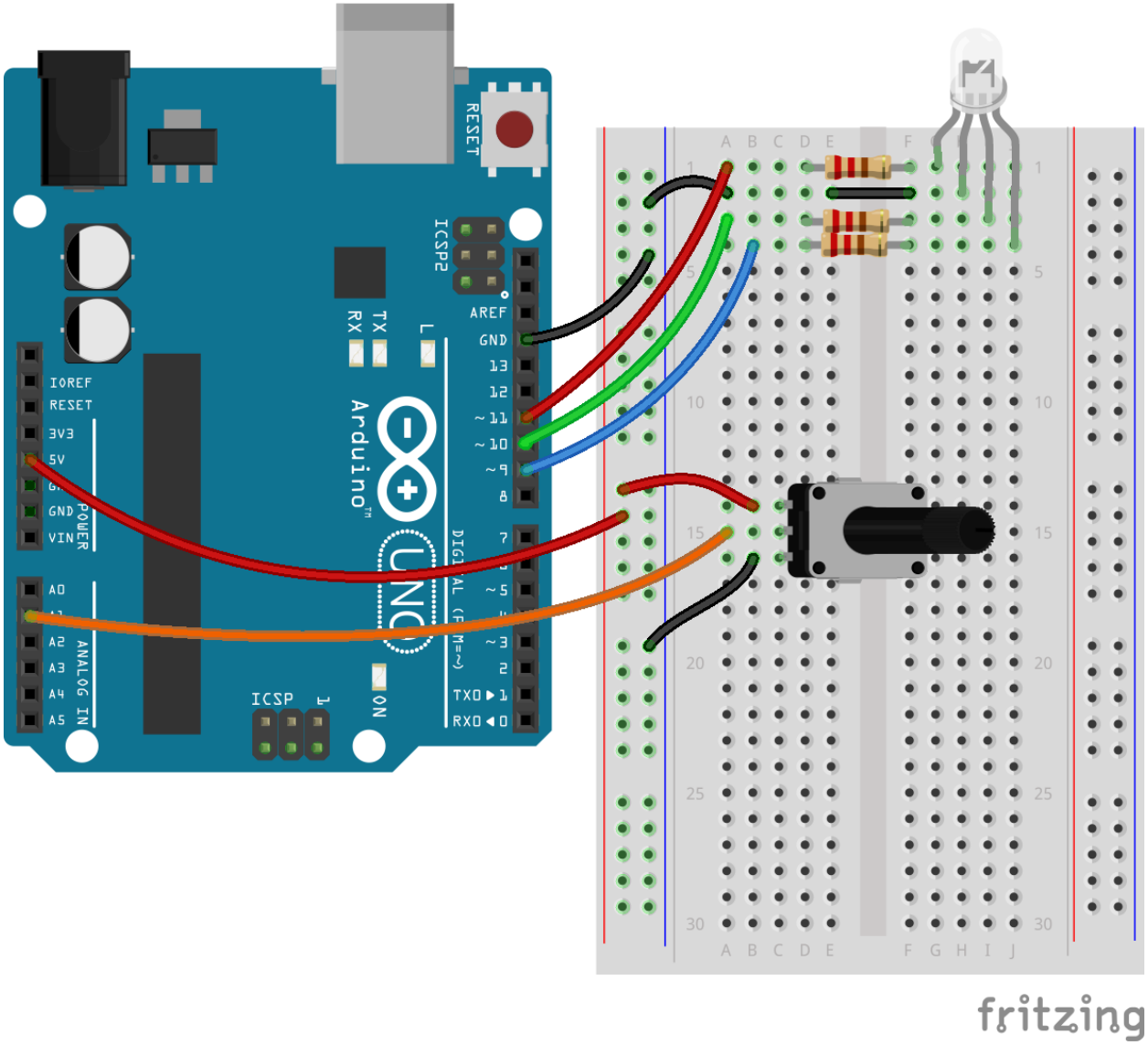
Rainbow colours

And I see your true colours…. shining through….
You're beautiful….. like a rainbow.

Yes, this project is all about 80's cheesy pop, and making rainbow colours.

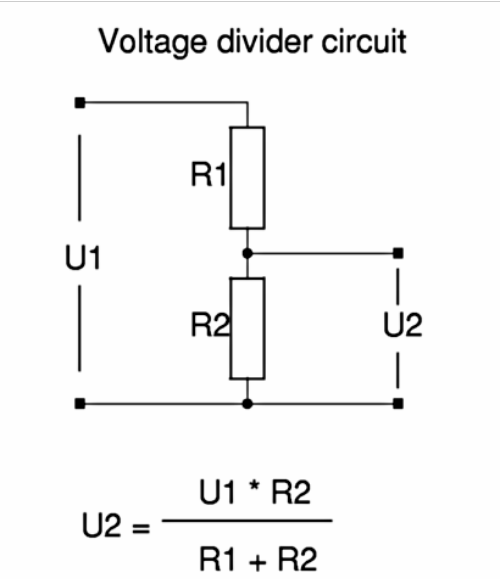


What this project will demonstrate:

- Using a voltage divider
- Reading analogue values
- Using simple function calls
- Simulating analogue (variable voltage) output using PWM
- Mixing colours using RGB

With this project, we're using a *potentiometer* to create a variable input voltage.
The changing input voltage directly relates to how far around the knob of the potentiometer has been turned. We're using a *voltage divider* to change a varying resistance into a variable voltage.

Now before we start to sound like a high school lesson, and start banging on about Ohm's Law, voltages, resistance and current, boring you to tears with maths equations let's back up a bit. Try to stay awake. It's really not *that* boring….



Wow. That looks incredibly nerdy. And a bit like maths or physics or some other boring lesson we left behind a long time ago. But here's what the diagram is basically saying:

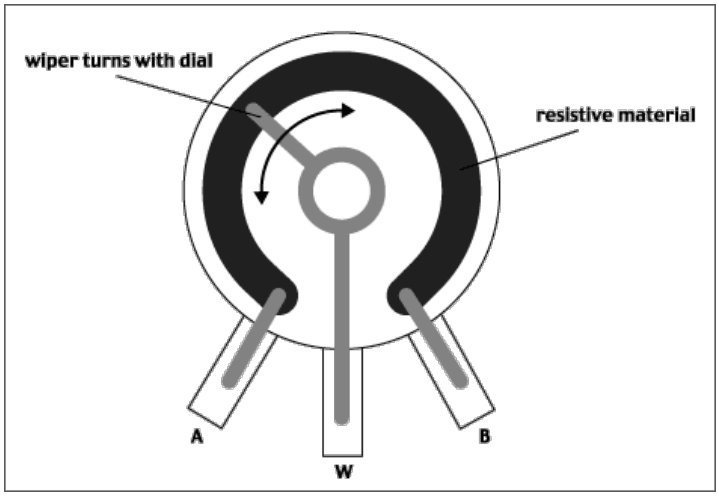
The voltage (in our case the supply voltage) is 5v from the supply line to the ground rail.
We already know this. If we were to put a single resistor between power and ground, we'd not only drain our battery or power source (and, unless it's a pretty high value resistor, we'd probably make smoke and maybe even some flames!) but we could measure the voltage *across* the resistor. It would be 5v on one side, and 0v on the other.

So if we put two resistors across the rails, we'll still have a 5v difference between the power and ground rails. That's just the nature of how the circuit works - so across *both* resistors, we've got a 5v difference. But at the mid-point, we'll have a reduced voltage. It won't be the full 5v (because it's a mid-point between the two-rails) and it won't be all the way down at 0v, but it'll be somewhere in between.

And this is where those boring equations come in.
To work out the voltage at some point in the middle of two resistors, we need to look at the *ratio* between them. By some amazing laws of dark magic (or it might be something to do with physics and Ohm's Law) the ratio between the resistance just so happens to be the ratio between the voltages.

Simply put - if we had a 1k resistor for both R1 and R2, the ratio between the voltages is 1:1
The ratio between the output voltage is also 1:1 - or, in real words, the voltage at the midpoint is half the supply voltage.

Wake up! We're getting to the good bit now!
A potentiometer is like having two resistors



It has an A and a B point, and a "moving midpoint" called a wiper.
So if we set the dial to the centre of a 10k linear pot (sometimes we get all cool and groovy call potentiometers pots) then the resistance between A and W is 5k, and the resistance between W and B is also 5k.

If we connect our power supply to A and ground to B, we've recreated the voltage divider circuit from earlier. R1 is the resistance between A and the wiper, R2 is the resistance between the wiper and B. But the great thing is, as we move the knob clockwise (which controls the wiper) the value of R1 goes up and the value of R2 goes down.
Turing the knob the other way reverses this effect.

As we aleady know, changing the two resistances in a voltage divider changes the output voltage (the voltage at the mid-point). So on our breadboard, we've connected the wiper to an analogue input pin.

These are special pins on the Arduino, which convert a variable voltage into a number between 0 and 1024 (don't ask why it's such a crazy number - it's to do with 2s and ADC convertors being 10-bits wide, and 2 to the power of 10 is 1024….. really, don't ask).

Anyway, we've got a way of turning a knob, and getting the Arduino to work out how far it's been turned. Our first analogue input!

We'll use this to choose a colour from the rainbow.
You can easily imagine how a face-place for this knob might look, with red all the way on one side, green somewhere in the middle, and violet over on the far side of the dial.
Let's make some colours!

You do know your primary colours, right?
red + yellow = orange
yellow + blue = green
blue + red = purple
red + yellow + blue = sludge

Here's where it gets a bit weird. You'll just have to trust us on this one:
The primary colours, when playing with lights (not paint) are:

red, **green** and **blue**.

Yup. Green.
And here's where you might need a sit down:
To make yellow, we mix red and green!

Once you've got over that bombshell, the rest is pretty similar as to the Dulux colour chart:
To make orange, for example, we still mix red and yellow.
But yellow is made up of red and green.
So orange is more like 2-parts-red and 1-part-green.
Trippy huh?

In fact, if you've ever done anything with web coding or HTML, you might already be familiar with this concept. If you play about with Photoshop or any imaging software, quite often you'll find colours mixed down into their red, green and blue components.
If all this is a bit much, just fire up your favourite image editing software and pick a colour, then read the HTML/RGB colour values off the screen. They won't be a close match, but they'll be near enough!

We're going to play with an RGB led to make different colours.
Using the Arduino's ability to simulate a variable output voltage (it actually just flickers an output pin really really fast, at different rates to emulate different intensities) we can mix colours by adding, say, a bit of red, to a lot of green, and a smidge of blue.

You might notice a resistor on each leg of the RGB LED.
You can just put one resistor on the ground leg, but you'll get slightly different results (and some colours might appear brighter, and overshadow the others). We're treating the RGB LED as three separate LEDs of different colour (just all mashed together into one package).

By hooking up each colour in the RGB to a different output pin, we can vary the intensity of each colour, and thanks to the diffused lens over the LED, the three colours appear as one - ranging from red, through orange and yellow, into green, and finally blue and purple.

The Arduino sketch simply looks at the position of the potentiometer knob and decides which “band” the resulting analogue value falls in. Which band the result is in determines which colour we display on the RGB LED.

Have fun!