



Co-funded by the
Erasmus+ Programme
of the European Union



Erasmus+ KA219 Project

ELECTRONIC COMPONENTS AND PROGRAMING ARDUINO

MODULE III

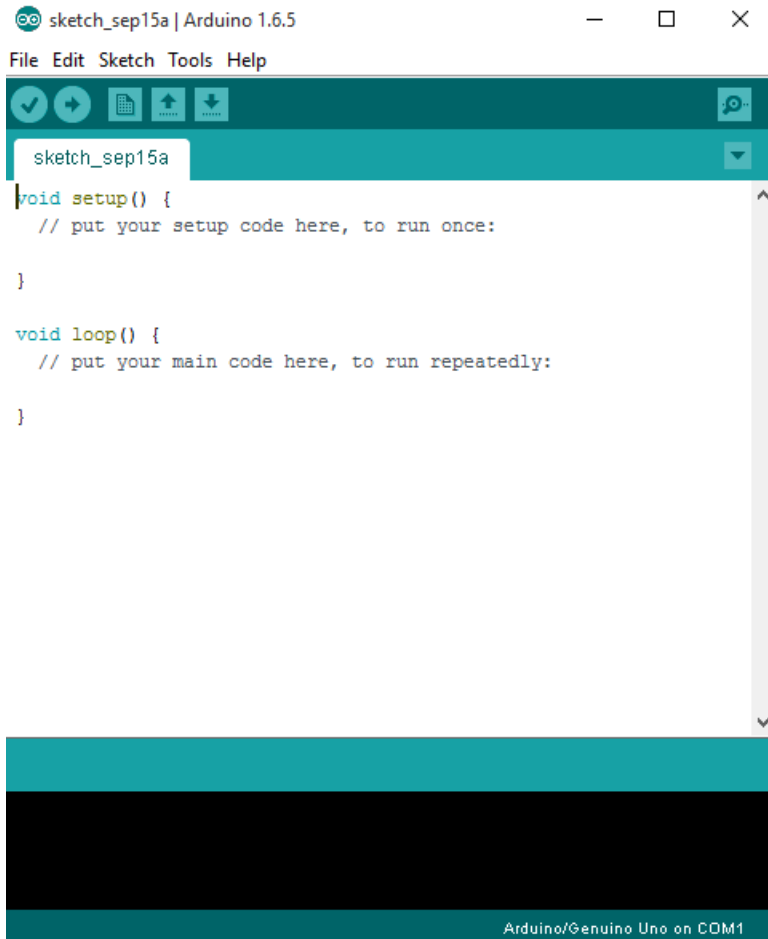


INDEX

| | |
|--------------------------------------------------------------------------|----|
| The Arduino IDE | 3 |
| Connecting your Arduino | 3 |
| Control an Output and read an Input | 5 |
| Controlling an output | 5 |
| 1. Blinking an LED, the simplest project | 5 |
| 1.1 Blinking an LED – taking it further | 8 |
| 1.2 Blinking an LED – with a pushbutton | 10 |
| 2. Traffic lights | 12 |
| 3. Controlling an LED with a potentiometer – reading analog values | 15 |
| 3.1 Controlling an LED with a Potentiometer + serial monitor | 16 |
| 3.2 Controlling 4 LEDs with a potentiometer | 17 |
| 4 Automatic lights with light sensor | 19 |

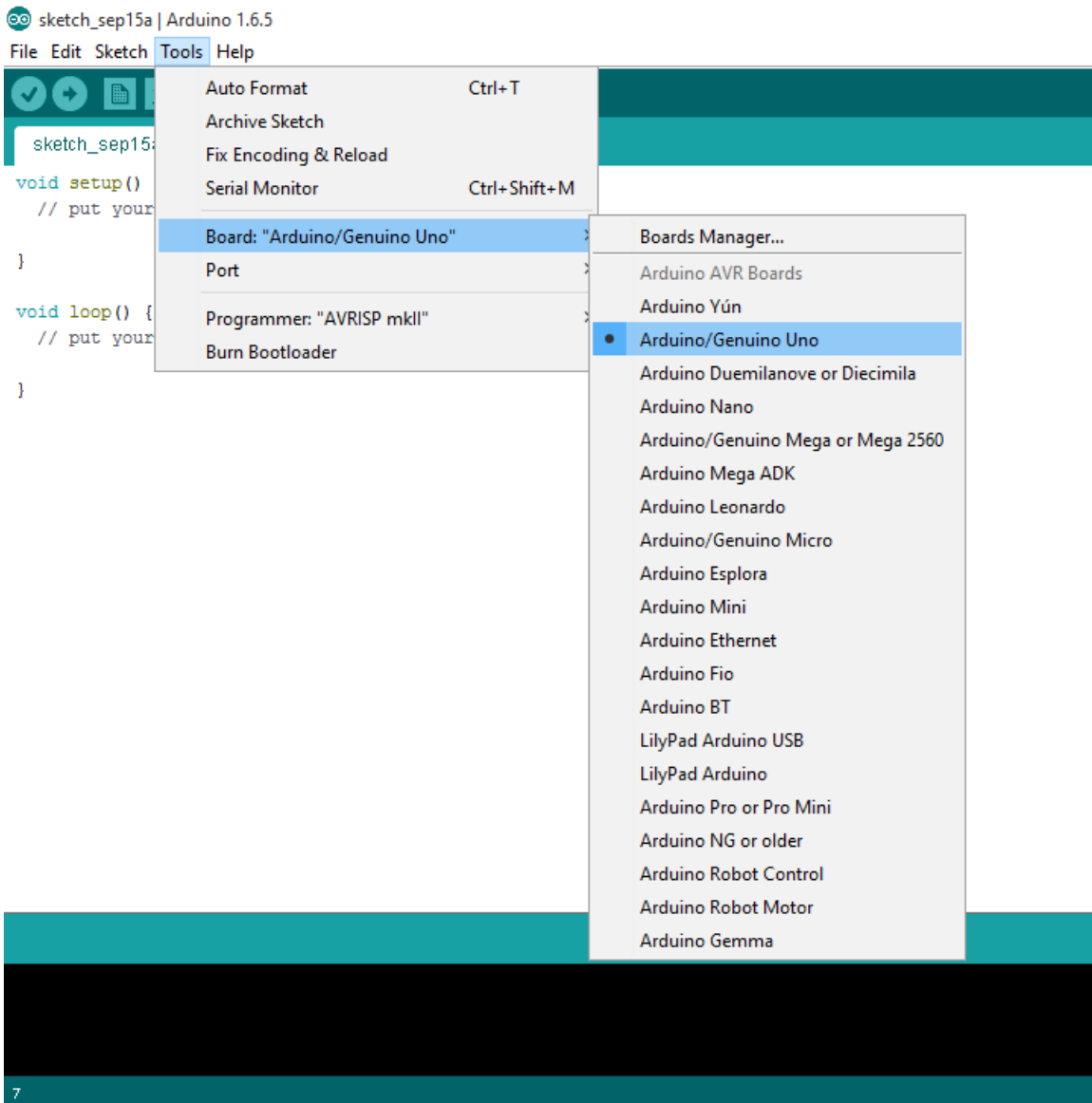
The Arduino IDE

The Arduino IDE (Integrated Development Environment) is where you develop your programs that will tell your Arduino what to do. You can load new programs onto the main chip, the ATmega328p, via USB using the Arduino IDE. To download your Arduino IDE, please click on the following link: <https://www.arduino.cc/en/Main/Software>. Select which Operating System you're using and download it. We won't go into much detail on how to install this software, since the official Arduino web site does a great job explaining how to do it in all three Operating Systems – Windows, Mac and Linux. When you first open the Arduino IDE, you should see something similar to the figure below:

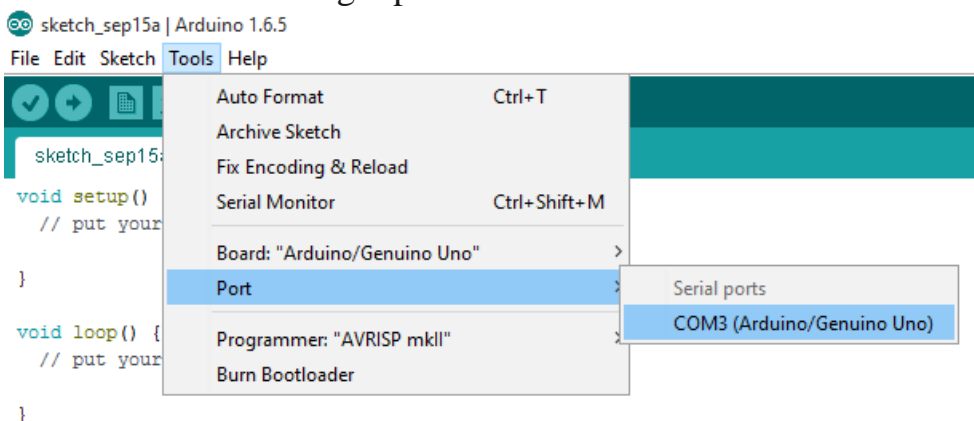


Connecting your Arduino

After connecting your Arduino with a USB cable, you need to make sure that Arduino IDE has selected the right board you are using. In our case, we're using Arduino Uno, so you should go to Tools > Board: "Arduino/Genuino Uno" > Arduino/Genuino Uno.



Then, you should select the right serial port where your Arduino is connected to. Go to Tools > Port and select the right port.



Control an Output and read an Input

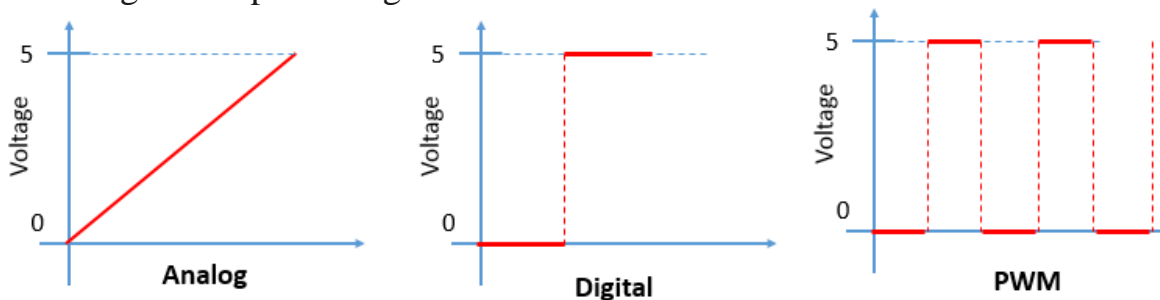
An Arduino board contains digital pins, analog pins and PWM pins.

Difference between digital, analog and PWM In **digital pins**, you have just two possible states, which are on or off. These can also be referred as High or Low, 1 or 0 and 5V or 0V. For example, if an LED is on, then, its state is High or 1 or 5V. If it is off, you'll have Low, or 0 or 0V.

In **analog pins**, you have unlimited possible states between 0 and 1023.

This allows you to read sensor values. For example, with a light sensor, if it is very dark, you'll read 1023, if it is very bright you'll read 0. If there is a brightness between dark and very bright you'll read a value between 0 and 1023.

PWM pins are digital pins, so they output either 0 or 5V. However these pins can output "fake" intermediate voltage values between 0 and 5V, because they can perform "Pulse Width Modulation" (PWM). PWM allows to "simulate" varying levels of power by oscillating the output voltage of the Arduino.



Controlling an output

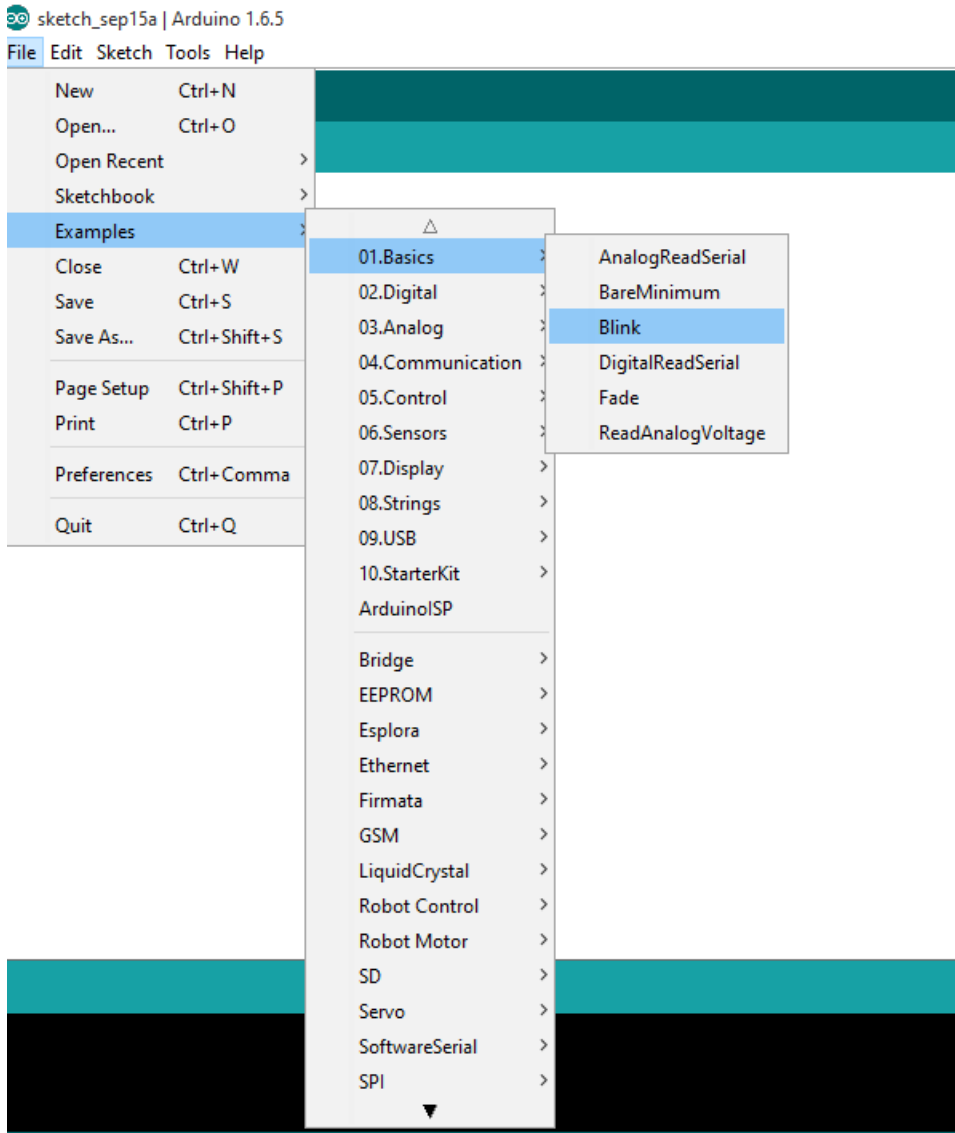
To control a digital output you use the function `digitalWrite()` and between brackets you write, the pin you want to control, and then HIGH or LOW if you want to turn something on or off. To control a PWM pin you use the function `analogWrite()` and between brackets you write the pin you want to control and a number between 0 and 255.

Reading an input

To read an analog input you use the function `analogRead()` and for a digital input you use `digitalRead()`. The best way for you to learn about Arduino is by putting your hands to work. So, let's start building some projects

1. Blinking an LED, the simplest project

If you go to File > Examples, you'll find several sketches you can use. For this project select Basics > Blink.



Then, you'll see something similar to the figure below. The words in grey are comments, they are not commands, they are just notes that the creator of that sketch wrote for other people understand what the sketch does. If you want to write comments, you just need to type `//` before you write. If you read the comments, you'll understand the purpose of each line of code.

```
✓ ↻ 📄 ⬆️ ⬇️
Blink
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
  */

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```

With the function `setup()` you type the code that you want your Arduino to run only once. For example, you tell your Arduino if a certain pin will be an output or an input. In this case, we are setting pin13 as an output. If you want to change the pin, you just need to replace “13” with the number of the pin you want to control.

In the `loop()` function, we can write what we want our Arduino to do. The `loop()`, as the name says it’s a loop, and so, it will run forever, over and over again. You have already read previously about the function `digitalWrite()`, you can see that in brackets we have (13, HIGH), which means that we want the pin 13 to be HIGH. As we have an LED in pin 13, the LED will turn on.



Then, the function `delay()` waits for a number of milliseconds. In this case it’s 1000 milliseconds, which is 1 second. After this 1 second, we can see that the `digitalWrite()` function has “LOW” written, which means that the LED will turn off. Then, we’ll wait for another second, and the loop will start again. To upload the sketch to your Arduino, make

sure that your board is connected to your computer and that the right serial port is selected. Then you need to press the upload button that is highlighted below:








Now, you will see the magic happening. The LED turns on for 1 second and then off for 1 second, and then on for 1 second, and so on. To better understand how the code works, try to change it on your own. For example, change the number inside the `delay()` function, upload the new sketch to your Arduino and see what happens.

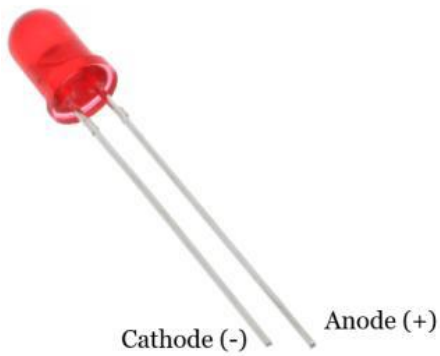
If you want to verify that your code has no errors, you can click the verify button (on the left side of the upload button).

1.1 Blinking an LED – taking it further

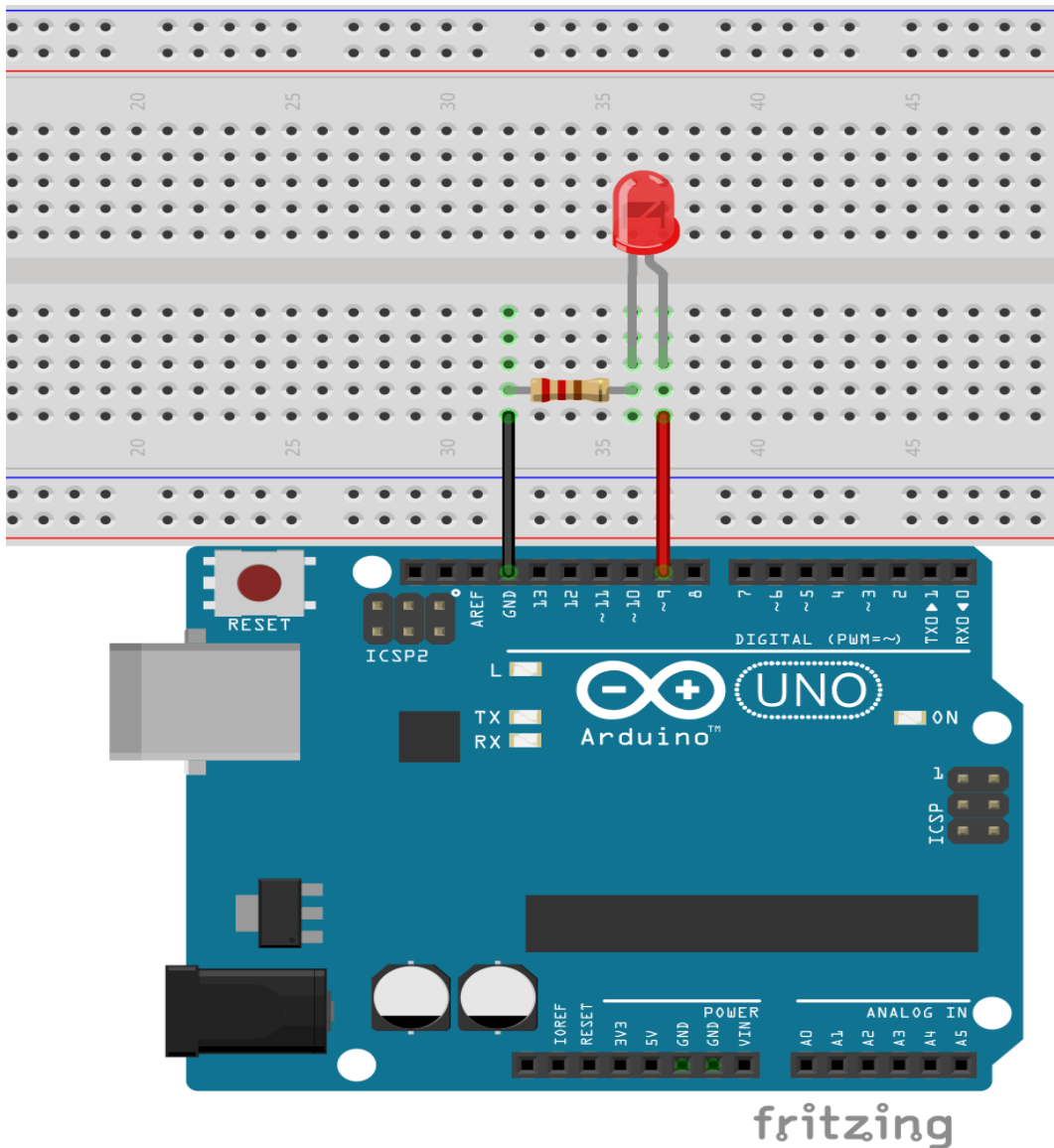
Now, let's do the same project but introducing some components. You'll need:

|  PARTS REQUIRED | |
|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| 1x LED |  |
| 1x 220Ω Resistor |  |
| Wires |  |
| 1x Breadboard |  |

You'll notice that your LED has two different legs, also called leads. One is longer than the other. The longest lead is the anode and the shortest is the cathode.



You should connect the anode to the positive supply, in this case to one of the digital pins, and the cathode to the ground pin. Let's place the components in the breadboard as shown in the schematics below:



Warning! You always need to have a suitable resistor when you connect LEDs to your Arduino. Arduino supplies 5V and an LED operates at a maximum of around 3V. If you connect the LED directly to your Arduino, you'll fry the LED. In general, a 220 Ω resistor is suitable for the LEDs you'll use.

Note that you have pin 9 connected, so we need to change our sketch and








replace 13 for 9. You could have chosen to connect your circuit to whatever pin you wanted. You just need to make sure that you change the sketch to include the proper pin.

```
// the setup function runs once when you press reset or power the board
void setup() {
// initialize digital pin 9 as an output.
pinMode(9, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
digitalWrite(9, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(9, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Now, you have a real LED blinking.

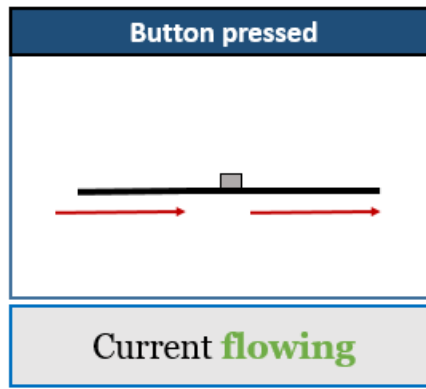
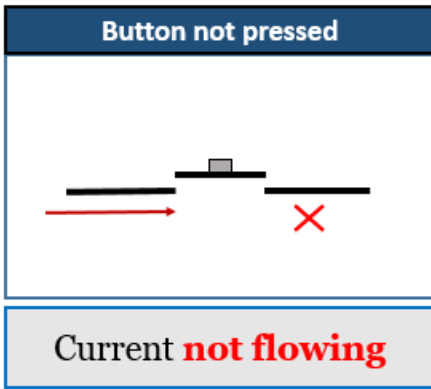
1.2 Blinking an LED – with a pushbutton

Here we will introduce a pushbutton. We will determine when the LED is on or off by pressing the pushbutton. When the pushbutton is pressed the LED will turn on, and when it isn't the LED will be turned off. We will also introduce the *if* and *else* statements.

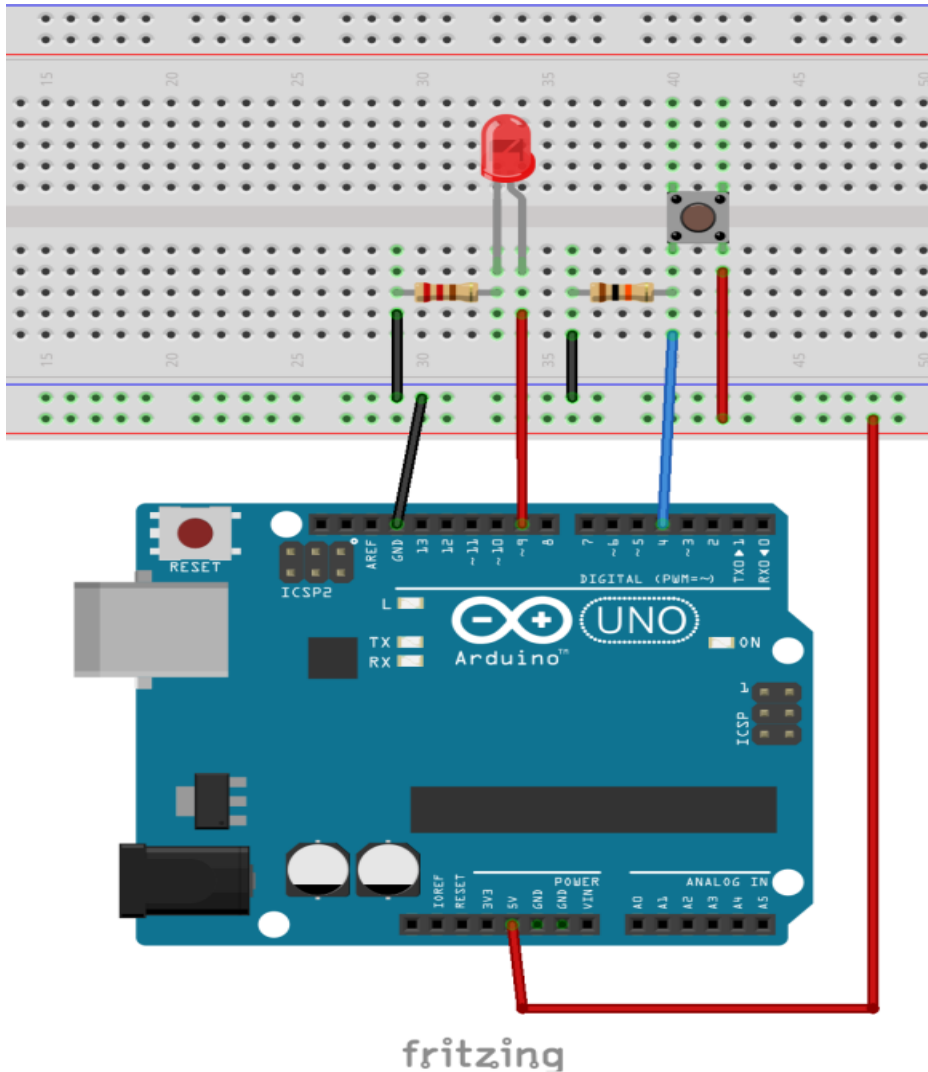
|  PARTS REQUIRED | |
|----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| 1x LED |  |
| 1x 220Ω resistor |  |
| 1x 10kΩ resistor |  |
| Wires |  |
| 1x Breadboard |  |
| 1 x Pushbutton |  |

How a pushbutton works

Here, we are using a normally-open pushbutton. When the pushbutton is in its normal state (not pushed), the current doesn't flow. When you press the pushbutton, you allow the current to flow. It can help if you take a look at the figure below.



Let's assemble everything together as demonstrated in the schematics:



In this circuit we are supplying 5V to the pushbutton. As the pushbutton is connected to pin 4, we can read its state by using the function `digitalRead()`. When you press the button, the current will flow through it, and you will read HIGH in the pin 4. If there isn't current flowing, we will read LOW, because in this situation pin 4 is connected to the ground through the 10kΩ resistor. To write this Arduino sketch, we need to bear in mind two simple conditions:

- If there's current flowing – the pushbutton is being pressed – then, the LED will turn on;
- Else, it will be turned off.

```
int ledPin=9; // here we are calling ledPin to the pin 9
int buttonPin = 4; //here we are calling buttonPin to the pin 4
void setup() {
```

```

pinMode(ledPin, OUTPUT);
pinMode(buttonPin, INPUT) ; // the buttonPin is an INPUT because we
want to read its state
}
void loop()
{
// if the pushbutton is being pressed
if (digitalRead(buttonPin) == HIGH){
digitalWrite(ledPin,HIGH); // then, the LED will turn on
}
else{
digitalWrite(ledPin,LOW); // else, it will be turned down
}
}

```

In the first 2 lines, we are assigning names to our pins. This is useful because anytime you need to refer to a certain pin you can use its name instead. This is also useful if you want to change the pins where your LED or your pushbutton are connected to, you just need to change a single line of code. We are declaring our variables as integers by writing “int” before their names, which means that we are defining their type. Integer data types can store values from -32,768 to 32,767. There are several variable types. Take a look at the most often used:







- **char:** stores a character value
- **string:** stores a sequence of elements, usually characters
- **byte:** stores an 8-bit unsigned number, from 0 to 255
- **int:** store values from -32,768 to 32,767
- **unsigned int:** the same as int, but stores just positive values, from 0 to 25,535
- **long:** extended size variables for number storage, from -2,147,483,648 to 2,147,483,647
- **unsigned long:** the same as long but stores just positive numbers, from 0 to 4,294,967,295
- **float:** number that have a decimal point, the floating-point numbers. these can be as large as 3.4028235e+38 and as low as -3.4028235e+38.

2. Traffic lights

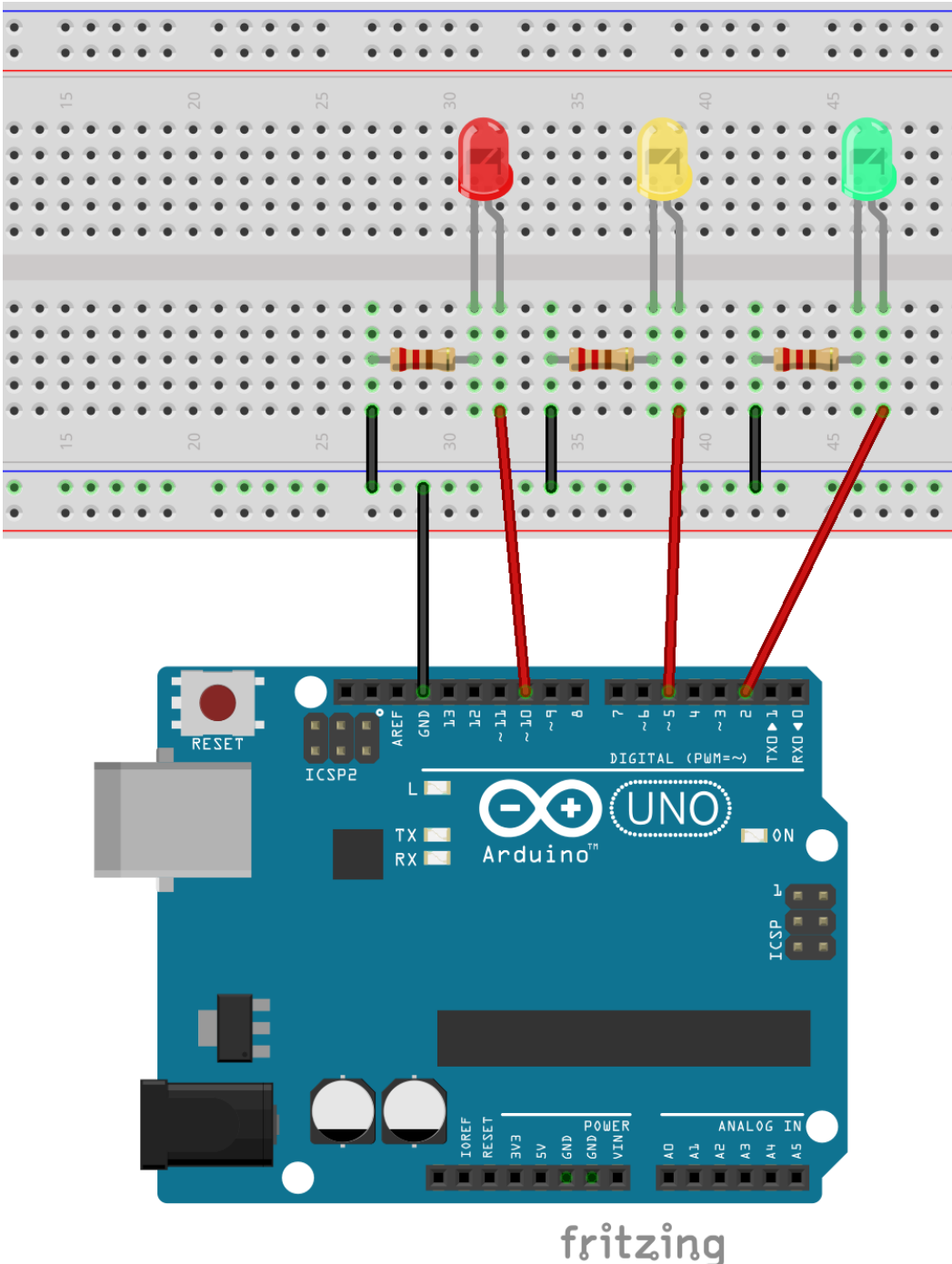
Now, let’s do something a little bit more advanced. Let’s mimic traffic lights. At this point you should be able to identify what components we’re going to need. Traffic lights have three colors, so we will need 3 LEDs, one of each color - red, green and yellow – and 3 x 220Ω resistors, one for each LED.



PARTS REQUIRED

| | |
|-------------------------|------------------------------------------------------------------------------------|
| 1x LED - red |  |
| 1x LED - green |  |
| 1x LED - yellow |  |
| 3x 220Ω resistor |  |
| Wires |  |
| 1x Breadboard |  |

Let's put everything together as shown in the following schematics:



Usually, wires that are connected to ground are represented in black and the 5V's represented in red. However, you can use any color you want.

Let's take a closer look at the code.

```
int redPin = 10; // here we are calling redPin to the pin 10
int yellowPin = 5; // here we are calling yellowPin to the pin 5
int greenPin = 2; // here we are calling greenPin to the pin 2
void setup() {
  // here we are initializing our pins as outputs
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}
void loop() {
  digitalWrite(greenPin, HIGH); // turn the green LED on
  delay(5000); // the green LED will be on for 5 seconds
```





```
digitalWrite(greenPin, LOW); // the green LED will turn off
digitalWrite(yellowPin, HIGH); // the yellow LED will turn on for 2
second
delay(2000);
digitalWrite(yellowPin, LOW); // the yellow LED will turn off
digitalWrite(redPin, HIGH); // the red LED will turn on for 5
seconds
delay(5000);
digitalWrite(redPin, LOW); // the red LED will turn off
}
```

Inside the *loop()* function, we are just turning the LEDs on and off in a specific order during a period of time. If at this point this feels a little bit confusing, we advise you to pick up a piece of paper and try to write what is going on.

Write the code in the Arduino IDE, or just copy and paste it and then upload it to your Arduino.

3. Controlling an LED with a potentiometer – reading analog values

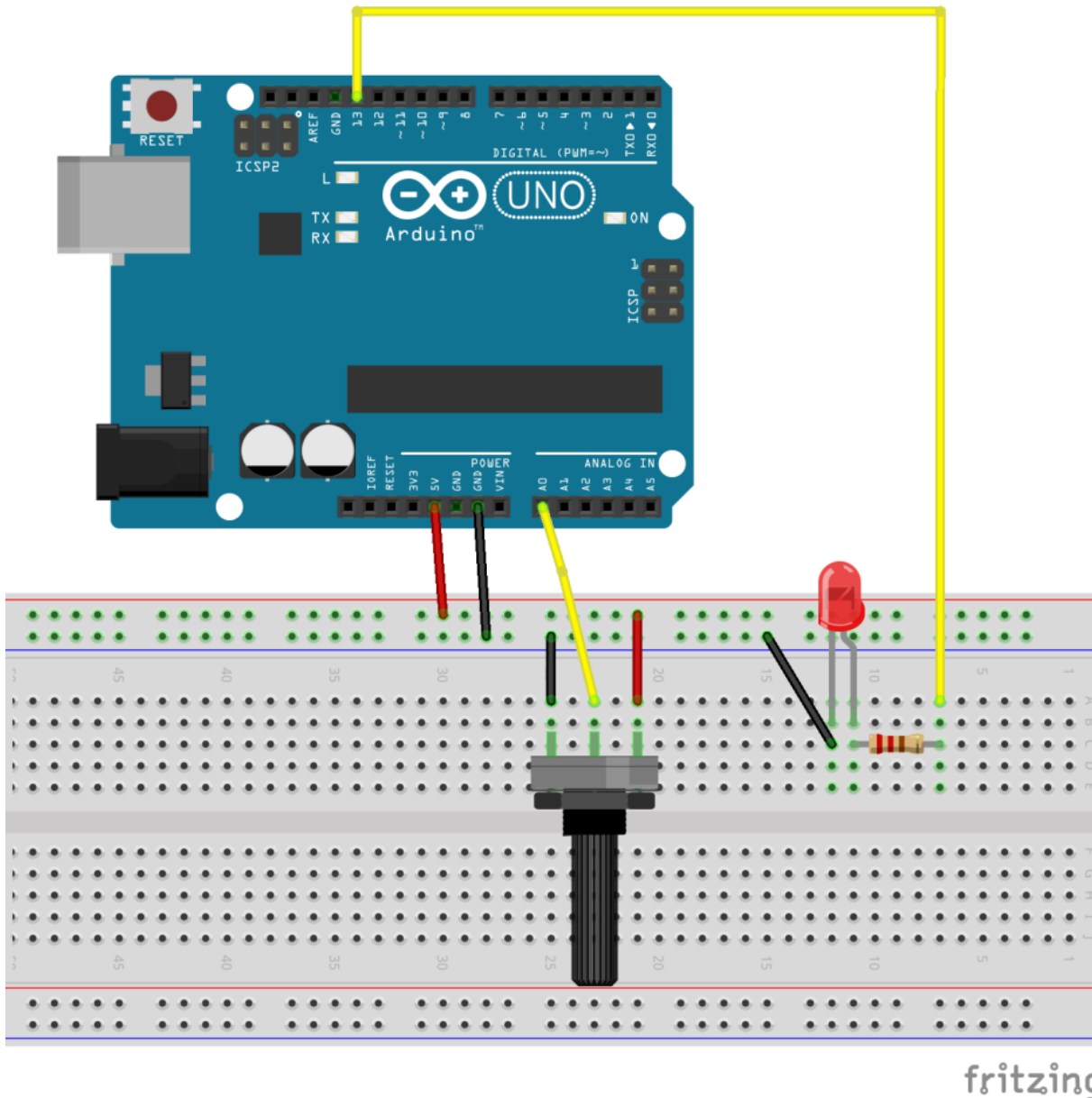
In this project we're going to control an LED with a potentiometer to learn how to read analog values with the Arduino. Besides those components, you'll also need:

|  PARTS REQUIRED | |
|---------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| 1x LED |  |
| 1x 220Ω resistor |  |
| 1x 10k potentiometer |  |

What is a potentiometer?

A potentiometer is a variable resistor. We can control its resistance rotating the shaft to one side or the other. It's like when you're increasing or decreasing the audio volume in a radio. If you turn the shaft to the right the volume will increase and when you turn it to the opposite side it will decrease - until it turns off. The level of resistance determines how much current is flowing into the circuit.

A potentiometer has three legs. The first leg is connected to the source (5V), the third leg to ground and the middle leg (the output) to an analog pin. Assemble the following circuit:



Open your Arduino IDE and go to File > Examples > Analog > AnalogInput.

Read carefully all the comments in order to understand how the sketch works. Finally click the upload button.

After uploading the sketch, rotate the potentiometer for one side and then for the other side. Try to understand what is going on. The delay time is increasing as you turn the potentiometer to the right. That's because the variable *sensorValue* is increasing, this means that the current that is being read in analog pin 0 (A0) is increasing (the resistance is decreasing).

3.1 Controlling an LED with a Potentiometer + serial monitor

If you want to know what values are being read by the analog pin you need to do this (this is explained later inside the sketch of this example):

- Inside your *setup()* function you need to write *Serial.begin(9600)* to initiate serial connection;
- Then, you need to write *Serial.println(sensorValue)* inside the *loop()* function to write the values that the sensor is reading in the serial monitor.

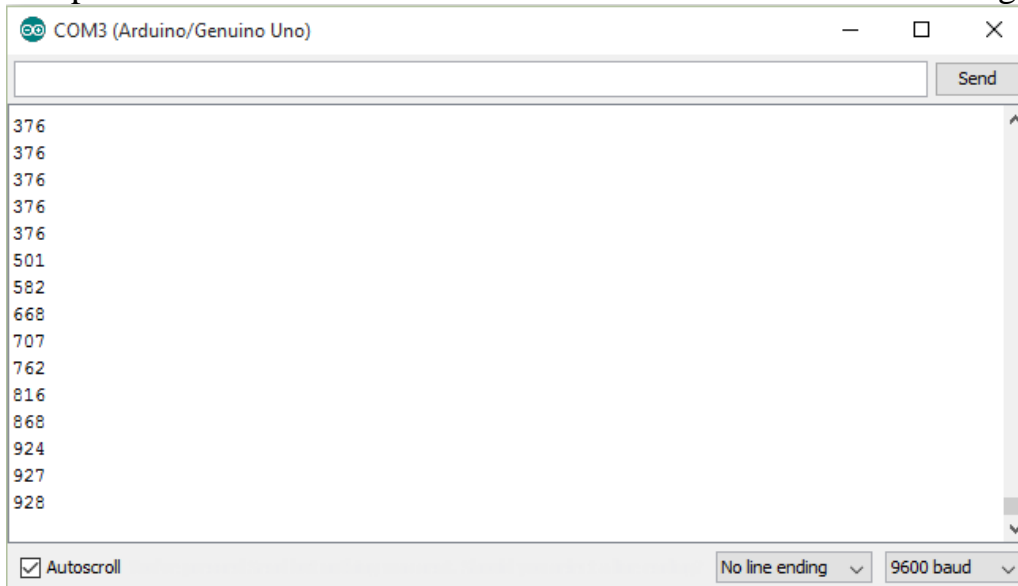
Here's how it should look like:


```

int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor
void setup() {
  Serial.begin(9600);
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}
void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  Serial.println(sensorValue);
  // 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);
}

```

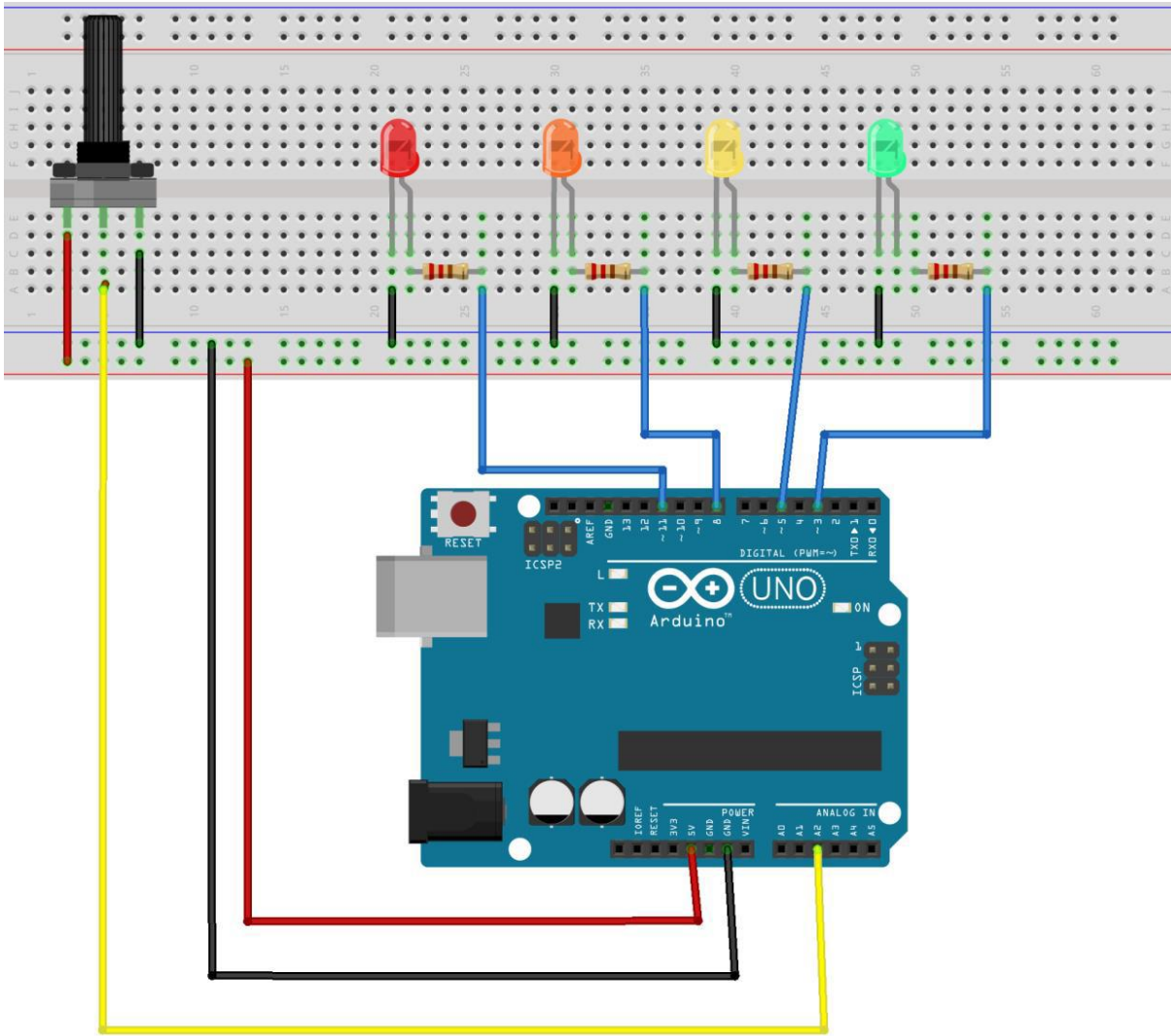
To open the serial monitor click on the icon shown on the following figure:



Turn your potentiometer for one side and for another. Your input values will vary between 0 and 1023.

3.2 Controlling 4 LEDs with a potentiometer

This is another simple example of what you can do with a potentiometer. In the previous project, we learnt that a potentiometer can read a value between 0 and 1023. What we are going to do is to divide that range in four equal ranges, and each range will turn on one specific LED. The parts for this project are the same as the previous, but you will need 3 more LEDs and 3 more resistors of 220Ω.



fritzing

Upload the following code and see the magic happening:

```
//initializing our variables
int potPin = A2; // select the input pin for the potentiometer
int ledPin1 = 3; // select the pin for the LEDs
int ledPin2 = 5;
int ledPin3 = 8;
int ledPin4 = 11;
int sensorValue = 0; // variable to store the value coming from the
sensor
int ledValue = 0;
void setup() {
Serial.begin(9600); // initializing serial connection
// declare the ledPins as OUTPUTS:
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
```

```

pinMode(ledPin4, OUTPUT);
}
void loop() {
// read the value from the sensor:
sensorValue = analogRead(potPin);
Serial.println(sensorValue);
if (sensorValue > 100)
digitalWrite(ledPin1,HIGH);
if (sensorValue < 100)
digitalWrite(ledPin1,LOW);
if (sensorValue > 407)
digitalWrite(ledPin2,HIGH);
if (sensorValue < 407)
digitalWrite(ledPin2,LOW);
if (sensorValue > 615)
digitalWrite(ledPin3,HIGH);
if (sensorValue < 615)
digitalWrite(ledPin3,LOW);
if (sensorValue > 922 )
digitalWrite(ledPin4,HIGH);
if (sensorValue < 922)
digitalWrite(ledPin4,LOW);
}






```

4 Automatic lights with light sensor

In this project we are going to read data from a light sensor (photoresistor). With this sensor we can know whether there's light or not. This is useful in automatic lights. When the night is coming, the lights turn on automatically.

In our project:

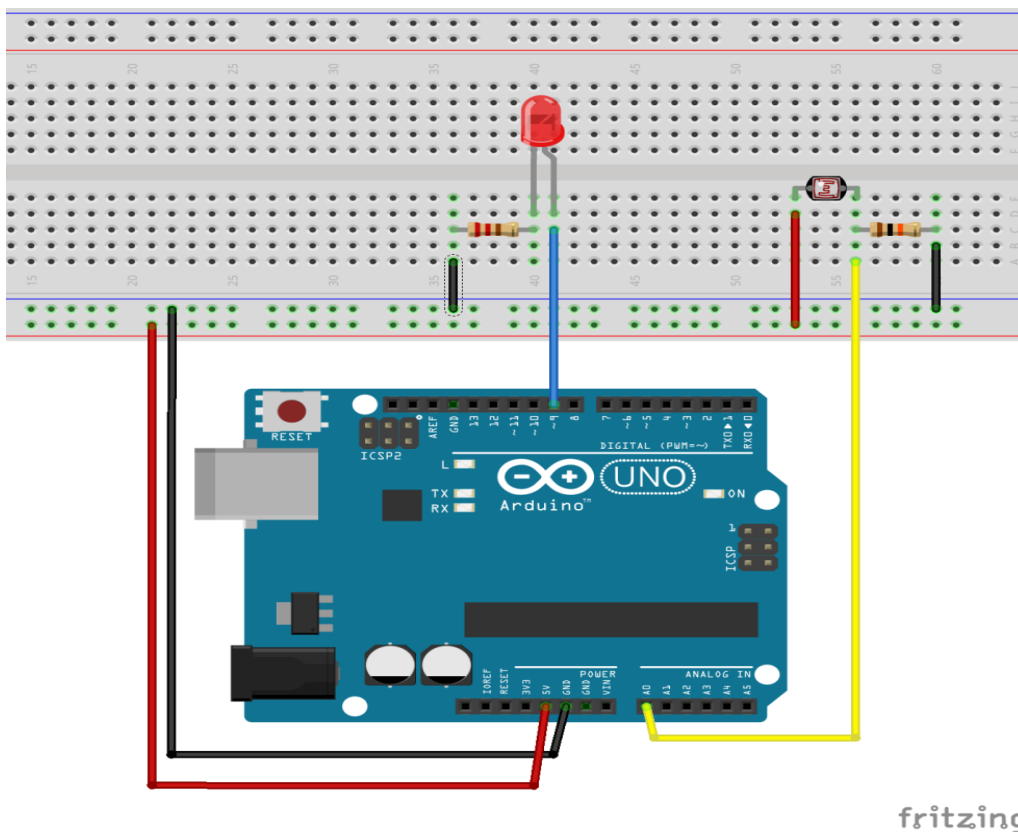
- If there isn't light, the LED will turn on
- If there is light, the LED will turn off

|  PARTS REQUIRED | |
|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| 1x LED |  |
| 1x 220Ω resistor |  |
| 1x photoresistor |  |
| 1x 10kΩ resistor |  |

What is a photoresistor?

A photoresistor is a light-dependent resistor. The resistance of a photoresistor decreases with increasing of light intensity. So:

- When there is light, the resistance decreases, we will have more current flowing.
 - When there is no light, the resistor increases, we will have less current flowing.
- Put everything together as the schematics below:



Then, upload the following code:

```
int ledPin = 9;
int ledBrightness = 0;
int sensorPin = A0;
int sensorValue = 0;
void setup(void) {
  pinMode(ledPin, OUTPUT);
  // We'll send debugging information via the Serial monitor
  Serial.begin(9600);
}
void loop(void) {
  sensorValue = analogRead(sensorPin);
  Serial.print("Sensor reading: ");
  Serial.println(sensorValue);
  // LED gets brighter the darker it is at the sensor
  // that means we have to -invert- the reading from 0-1023 back to 1023-0
  sensorValue = 1023 - sensorValue;
  //now we have to map 0-1023 to 0-255 since that's the range analogWrite uses
  ledBrightness = map(sensorValue, 0, 1023, 0, 255);
  analogWrite(ledPin, ledBrightness);
  delay(50);
}
```

In the code we use the *map()* function. This function should be used as follows: *Map(value, fromLow, fromHigh, toLow, toHigh)*.

We use this function when we have a value within a certain range, and we want to fit that value into another range. In our case, we are reading an analog value that is between 0 and 1023. Then, we want to output that value.

However, the output pins, output a range between 0 and 255, so we need to adjust that value to this new range.