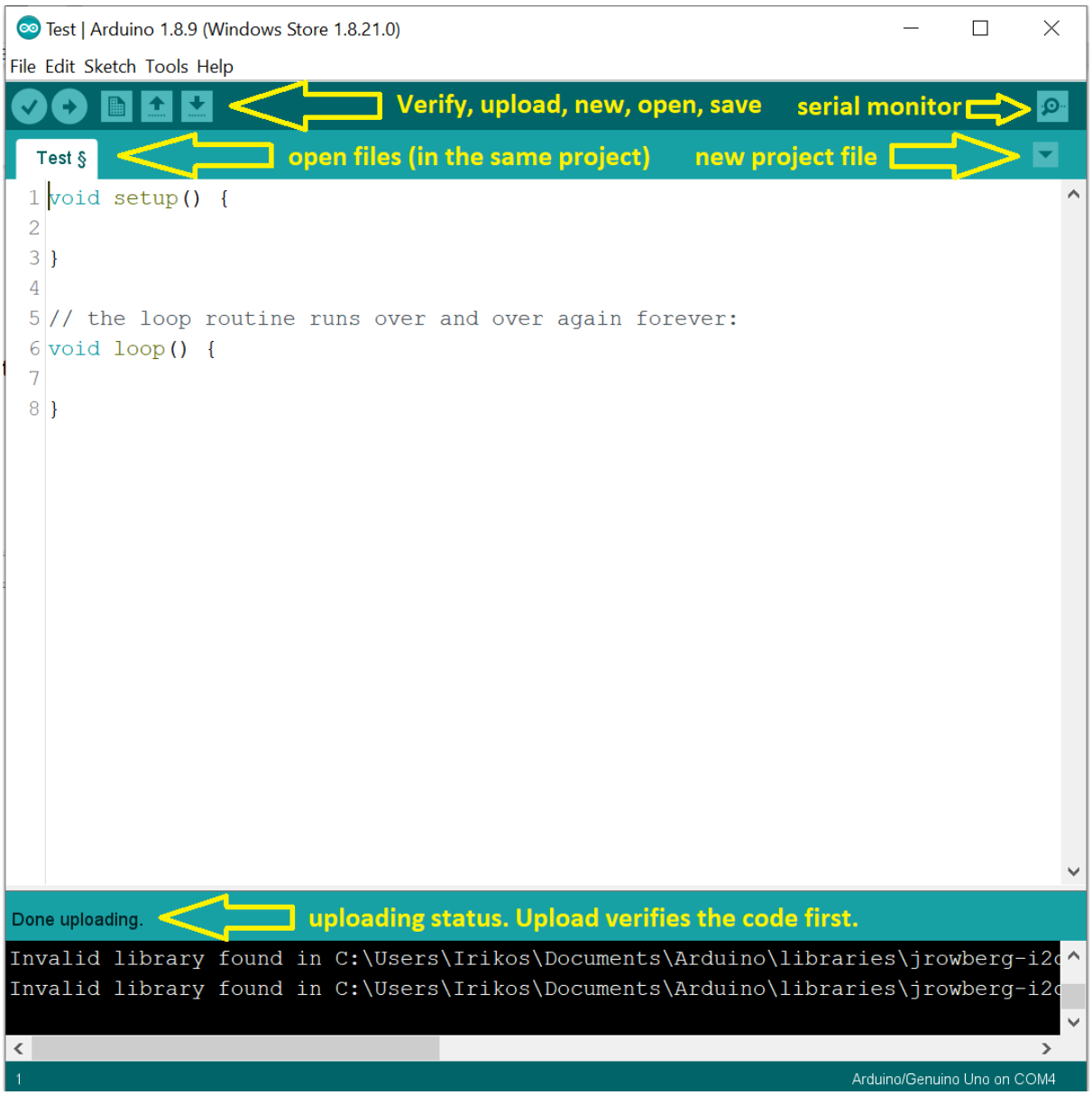
Introduction to robotics

2nd lab

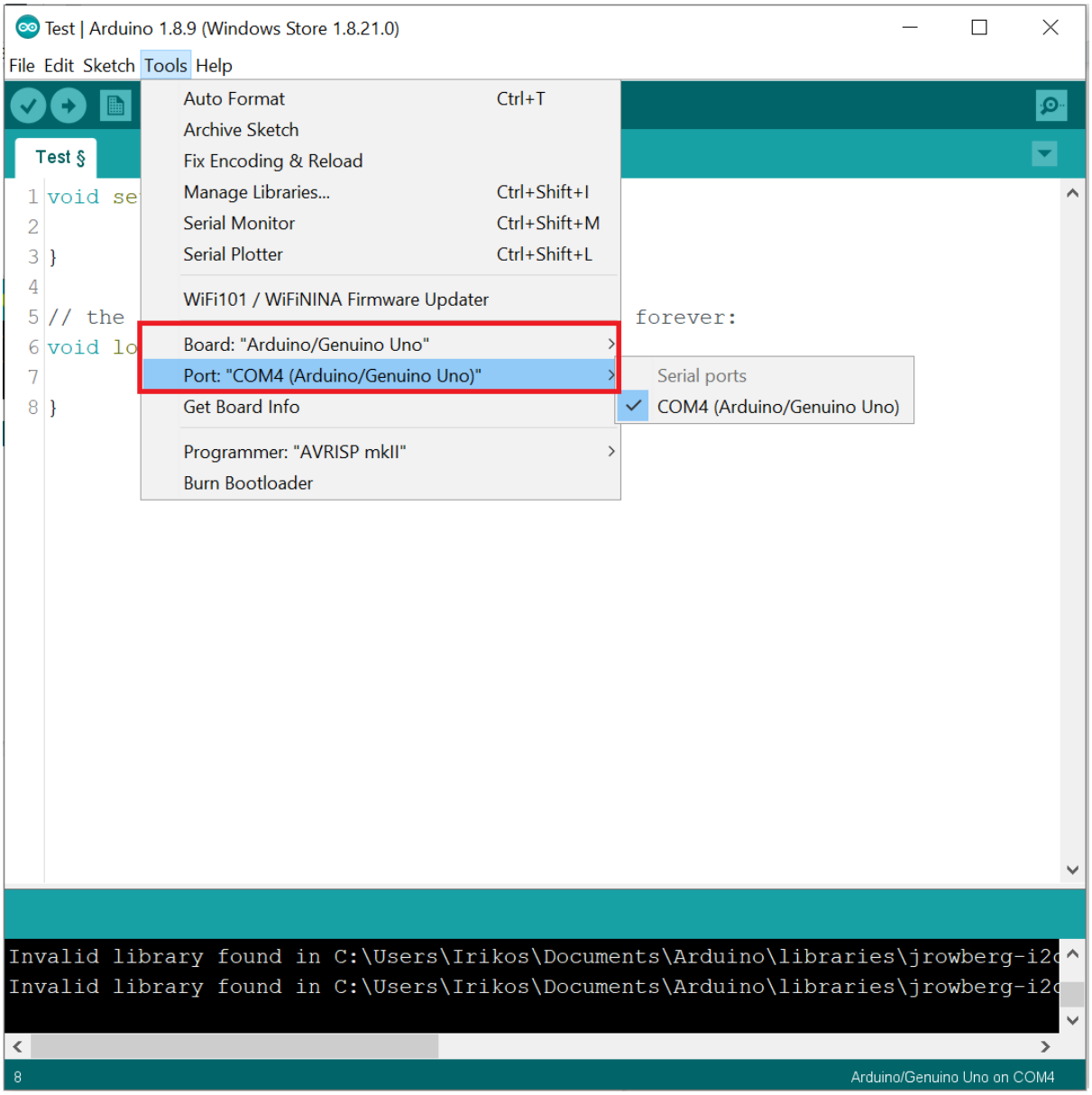
First of all, **homework check**.

1. Arduino IDE

Learn the Arduino IDE interface.



Obs: Don't forget to check if the port is selected!x§



Tips:

- opening serial monitor **RESTARTS** your program
- uploading also verifies your code, no need to verify and (re)upload when doing small changes
- when uploading, one of the most common error sources is not having the right port selected. Make sure you have the right board and port selected. This can change even if the arduino loses power for a brief second

2. Blinking led no. 13.

https://www.arduino.cc/en/tutorial/blink

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

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

Questions:

- What's with the delay?
- What happens if we change it (only one?)
- What happens if we remove it?

Remember, **LED_BUILTIN** is actually an arduino constant for the number **13**. You can easily replace LED_BUILTIN with the number 13. It is, however, bad practice to write the number of the pin directly. Still, since we will be using the PIN no. 13 for other purposes besides controlling an LED, it's best to create our own variable. (do that)

Pro tip: the built in examples are helpful, but do not get into the habit of opening them from File -> Examples in the Arduino IDE. Instead, go to https://www.arduino.cc/en/Tutorial/BuiltInExamples, understand the schematic and the code, then go to the IDE and write it yourself. The built in examples code can be confusing due to lots of comments and they also cover the basics of Arduino programming which you should master. There's a big difference between understanding how it's done and actually doing it yourself. Illusion of mastery is a real thing.

ATTENTION! Do not look directly perpendicular to the tip of the LED and do not look too much at the LED. It can and will affect your eyes - take this seriously!

```
const int ledPin = 13;

void setup() {
```

```
  // put your setup code here, to run once:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(100);
}
```

Now, as we've talked and as you can see, the built in led is actually connected to pin 13. So let's add an led to pin 13 as well.

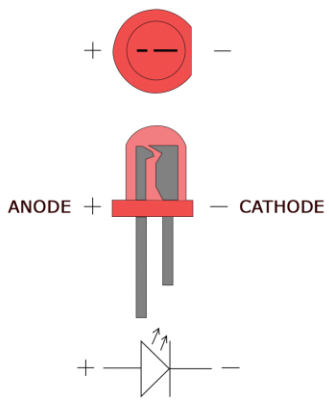
For this, we'll need 3 items:

- a LED (**green or maybe yellow. Blue and white ones are the brightest and will hurt your eyes**)
- a breadboard
- a 220 ohm resistor (330 works as well)

Take them all out.

Questions:

- What's a LED?
- Which one is + and which one is - and how do you recognize them? How are they called?
- What's a breadboard and how does it work?
- What's a resistor?

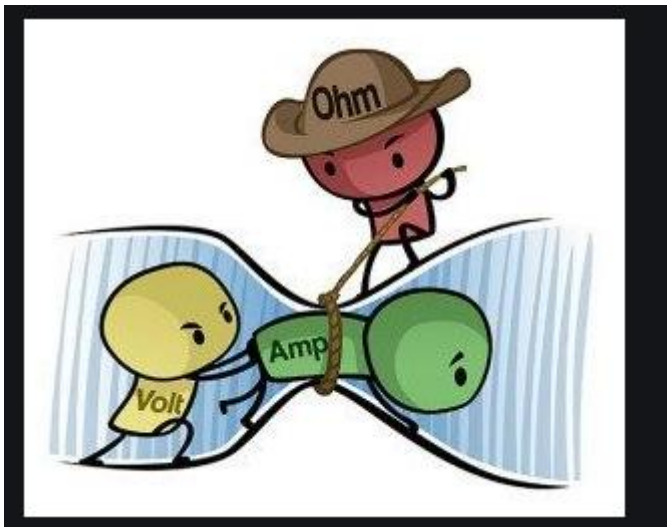


A light-emitting diode (LED) is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons. The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the band gap of the semiconductor.

Interesting facts: Unlike a laser, the light emitted from an LED is neither spectrally coherent nor even highly monochromatic. However its spectrum is sufficiently narrow that it appears to the human eye as a pure (saturated) color.

Resistors are electrical components used to limit current in a circuit. They are passive components, meaning they only consume power (and can't generate it). The resistor's resistance limits the flow of electrons through a circuit.

We'll study Ohm's Law and how to calculate the resistor value at the next course.



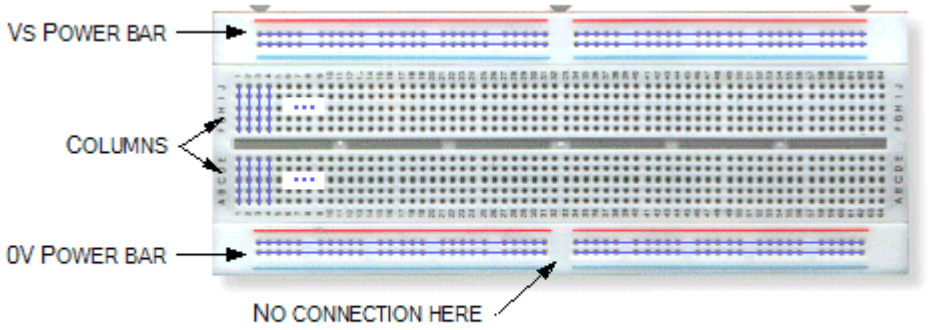
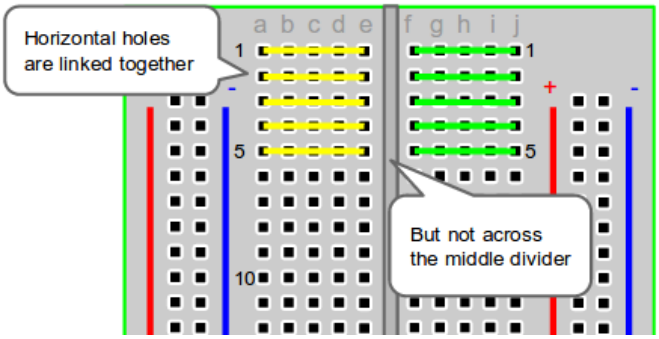


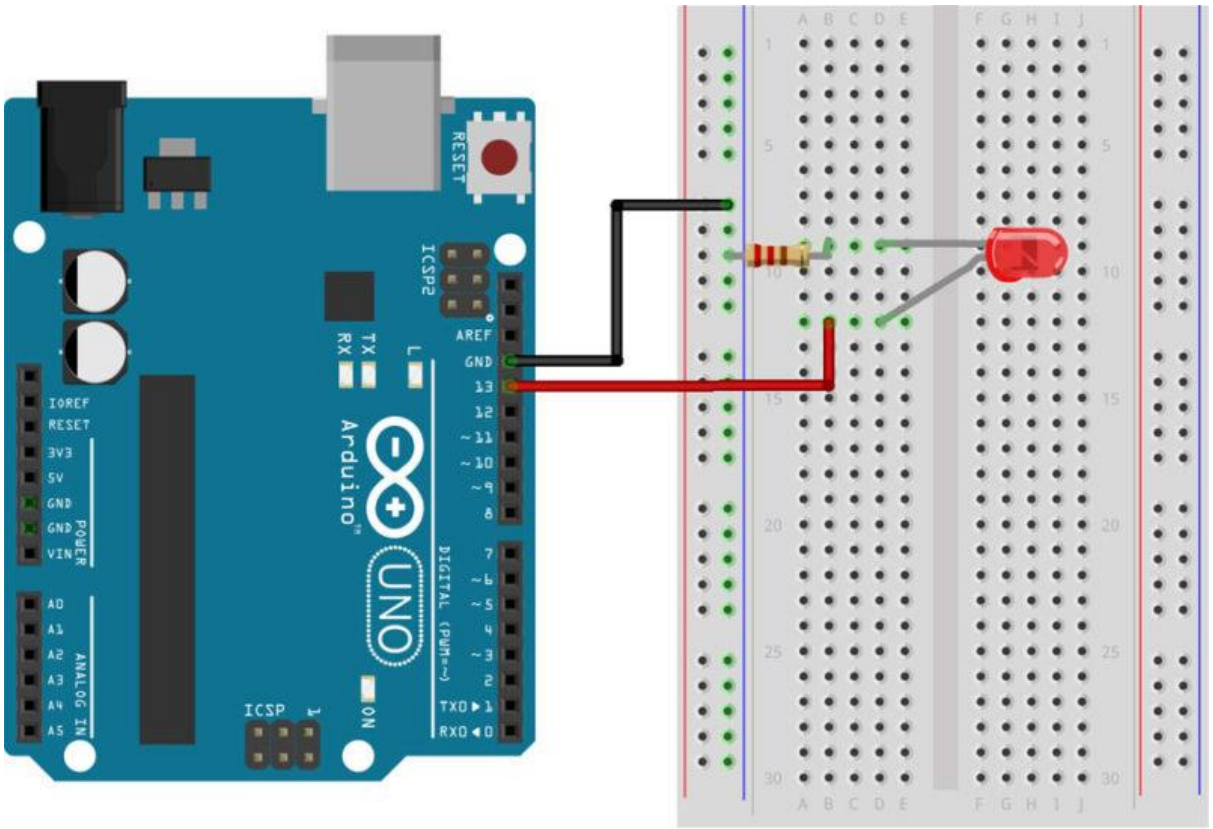
Figure 1: How the holes in a breadboard are connected electrically



Using the same code, let's add the LED. We can add the resistor to both the GND or the 13 PIN.

Remember, when possible, chose the wire color accordingly:

- **BLACK** (or dark colors) for **GND**
- **RED** (or colored) for **POWER (3.3V / 5V / VIN)**
- **Remember** than when you use digitalWrite or analogWrite, you actually send power over the PIN, so you can use the same color as for **POWER**
- **Bright Colored** for read signal
- We know it is not always possible to respect this due to lack of wires, but the first rule is **NOT USE BLACK FOR POWER OR RED FOR GND!**



3. Blink without delay

https://www.arduino.cc/en/tutorial/BlinkWithoutDelay

The example above works well, but mostly because it's simple. Can you identify a problem

with it?
The delay() function pauses the program for the amount of time specified as a parameter. While it is easy to create a blinking LED with the **delay()** function and many sketches use short delays for such tasks as switch debouncing, the use of delay() in a sketch has significant drawbacks. No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function, so in effect, it brings most other activity to a halt. More knowledgeable programmers usually avoid the use of **delay()** for timing of events longer than 10's of milliseconds unless the Arduino sketch is very simple.

But there's a better way.

"An analogy would be warming up a pizza in your microwave, and also waiting for some important email. You put the pizza in the microwave and set it for 10 minutes. The analogy to using delay() would be to sit in front of the microwave watching the timer count down from 10 minutes until the timer reaches zero. If the important email arrives during this time you will miss it.

What you would do in real life would be to turn on the pizza, and then check your email, and then maybe do something else (that doesn't take too long!) and every so often you will come back to the microwave to see if the timer has reached zero, indicating that your pizza is done." (source: arduino.cc)

```
const int ledPin =  13;
int ledState = LOW;

// Generally, you should use "unsigned long" for variables that hold
time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0;

const long interval = 1000;         // interval at which to blink
(milliseconds)

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // check to see if it's time to blink the LED; that is, if the
difference
  // between the current time and last time you blinked the LED is
bigger than
  // the interval at which you want to blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
```

```
      // save the last time you blinked the LED
      previousMillis = currentMillis;

      // if the LED is off turn it on and vice-versa:
      if (ledState == LOW) {
      ledState = HIGH;
      } else {
      ledState = LOW;
      }

      // set the LED with the ledState of the variable:
      digitalWrite(ledPin, ledState);
  }
}
```

Millis() returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero) after approximately 50 days.

```
unsigned long elapsedTime;

void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.print("Time: ");
  elapsedTime = millis();

  Serial.println(elapsedTime); //prints time since program started
}
```

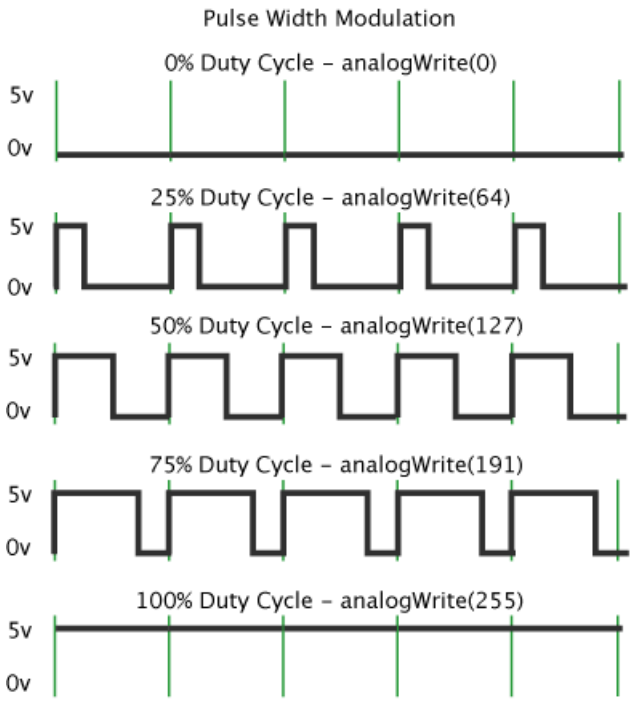
When checking Serial Monitor, check the “Check timestamp” for this example. What do you notice?

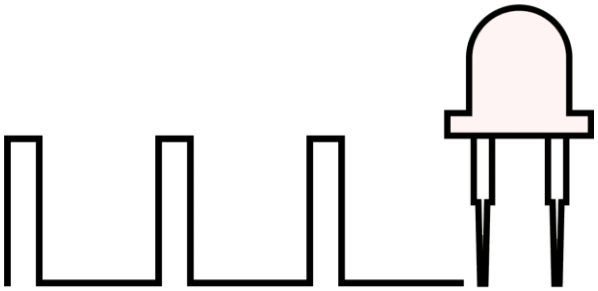
4. Fading an LED

Now that we know how to turn a light (LED) on and off, let’s see how we can choose the intensity of the light as well.

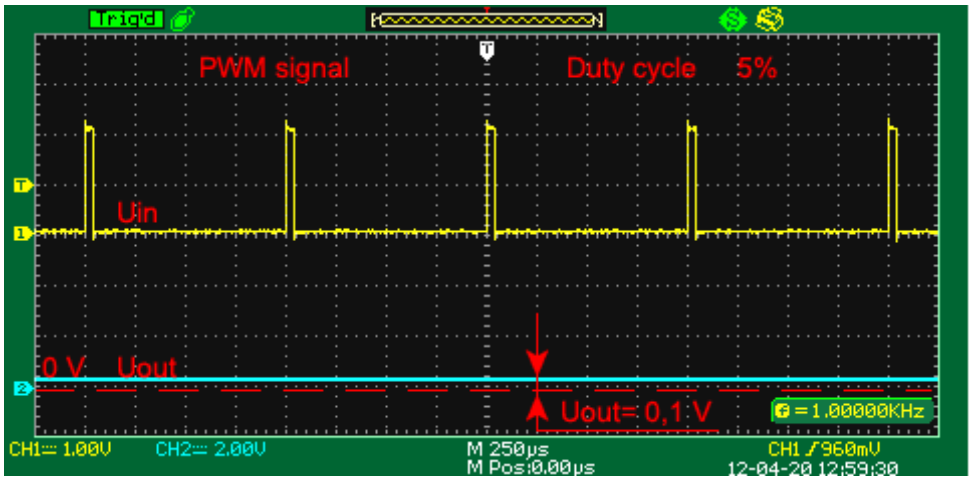
As you might remember from the course, the LED is a good example of a component that can be used both as a **digitalOuput** and an **analogOutput.** Basically, the intensity of the LED when using digitalWrite(ledPin, HIGH) = analogWrite(ledPin, 255), but you can set the LED value anywhere between 0 and 255.

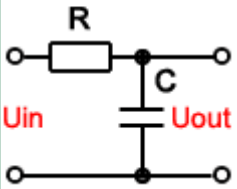
Let's recap the course a bit: PWM (pulse-width modulation) is a technique for getting analog results with digital means.





In practice, we use an RC low pass filter to convert the square wave into a continuous analog voltage (azzure line in the gif below)





That being said, let's change the code and play a bit with the LED intensity.

Since 255 is maximum, let's change it to 200.

```
const int ledPin = 13;
```

```
void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  analogWrite(ledPin, 128);
}
```

Hmm, we can't really see a difference. Let's change it to half then, **128.** Strange, we still can't see a difference. Let's change it to **127.** (it won't turn on anymore).

Questions:

- Why did it turn off?
- Why didn't it fade?

Ok, so always remember that the PINS that can use PWM and on which you can use analogWrite are the ones with **~ (3, 5, 6, 9, 10, 11)**

So, let's change it to **digital pin 9 (D9)** and try again. After we have connected the **ANODE (+)** wire to D9, we can easily see the advantage of keeping the pinNumber in a variable. In the code, we only have to change the **ledPin** value once, instead of changing it both in setup (pinMode) and in loop (analogWrite)

```
const int ledPin = 9;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  analogWrite(ledPin, 200); // change it to more values and test it's
                            // limits: 200, 128, 127, 100, 10 etc
}
```

Now let's fade it on and off.

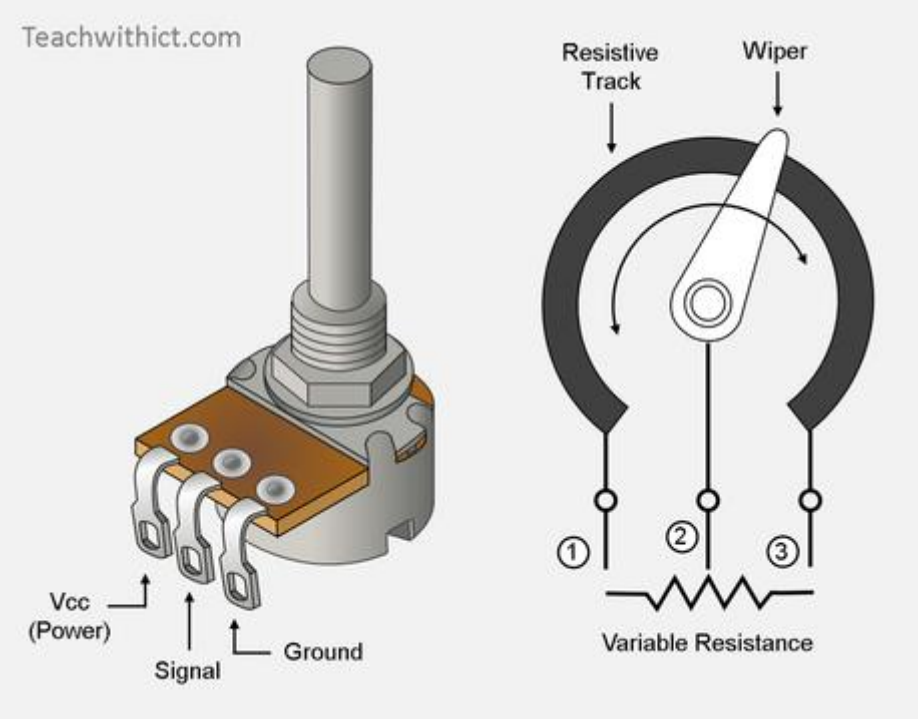
Example: https://www.arduino.cc/en/Tutorial/Fade
Let's write the code together.

```
const int ledPin = 9;    // the PWM pin the LED is attached to
int brightness = 0;      // how bright the LED is
int fadeAmount = 5;      // how many units to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(ledPin, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(ledPin, brightness);

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

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
      fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

5. Reading and printing a potentiometer value

Controlling the LED intensity is great, but changing the value and re-uploading the code is not practical. Let's add a component which we can use to control the LED intensity.

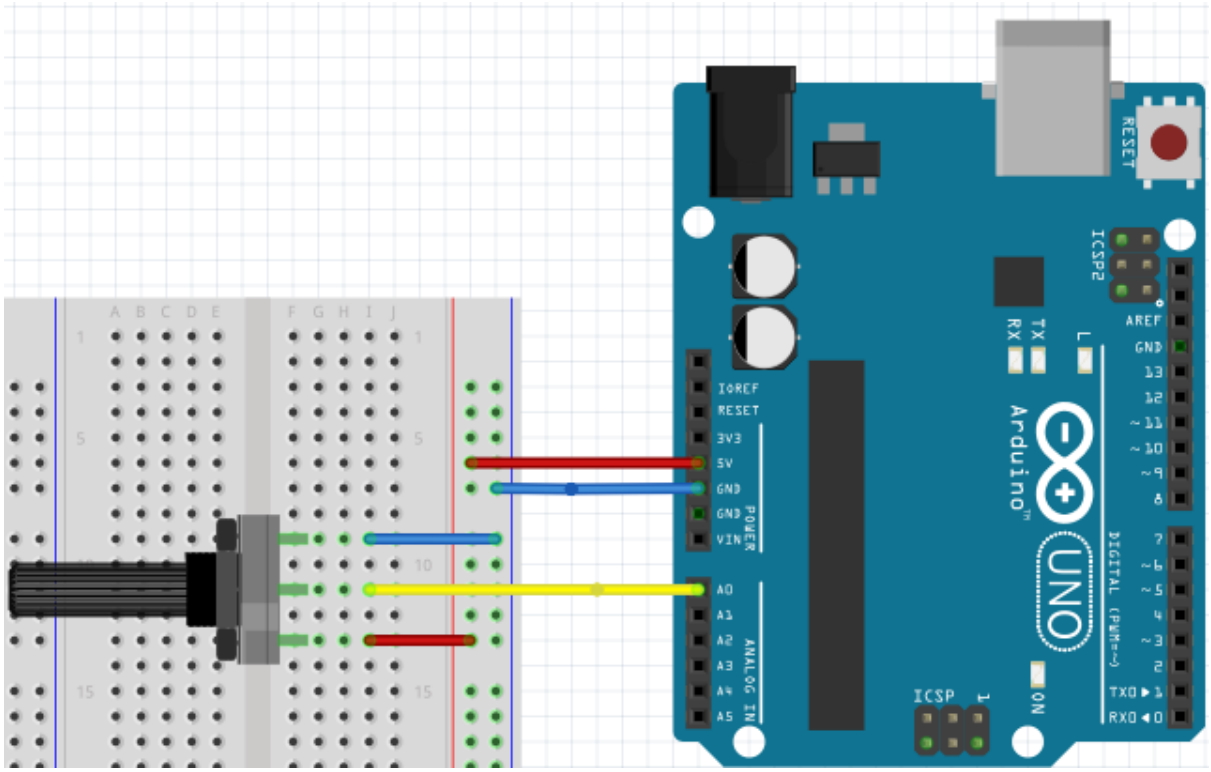
Meet the potentiometer:



A potentiometer is a simple knob that provides a variable resistance, which we can read into the Arduino board as an analog value.

Let's connect the potentiometer to the board as follows:

- the outermost pins go to **5V** and **GND**
- the middle pin goes to **A0**




```ino
const int potPin = A0;
int potValue = 0;

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  potValue = analogRead(potPin); // not the best practice
  // print out the value you read:
  Serial.println(potValue);
  delay(1);        // delay in between reads for stability
}
```

This is the example from https://www.arduino.cc/en/Tutorial/AnalogReadSerial, but slightly modified.

Questions:

- **Q:** Why do we use **const int** for pins but **int** for values?

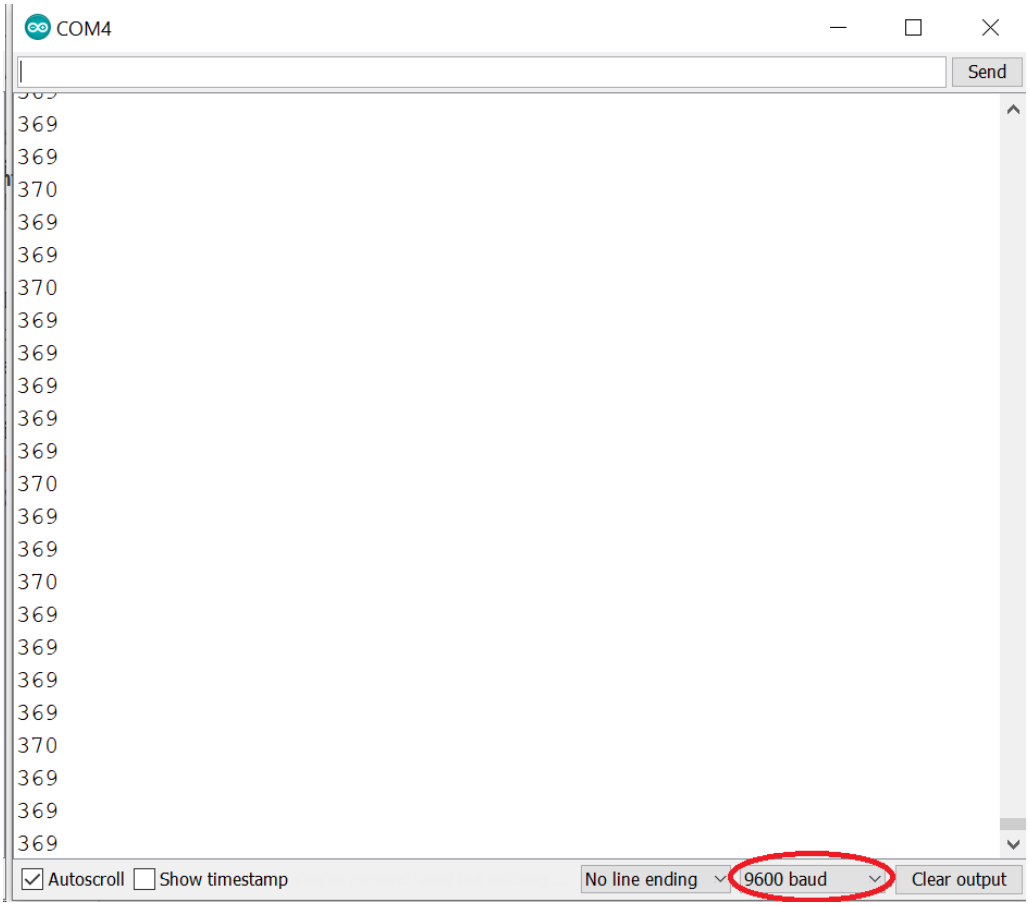
Run the program and open **Serial Monitor.**
On the bottom right, you will see **9600 baud.**

The **baud rate** is the **rate** at which information is transferred in a communication channel. In the serial port context, "9600 **baud**" means that the serial port is capable of transferring a maximum of 9600 bits per second.

At baud rates above 76,800, the cable length will need to be reduced. The higher the baud rate, the more sensitive the cable becomes to the quality of installation, such as how much of the wire is untwisted around each device.

Long story short, the number in Serial.begin(9600) must be the same as the baud rate.

Q: What do you think will happen if we change the baud rate in the serial monitor? Why does this happen?



6. Reading the analog voltage

https://www.arduino.cc/en/Tutorial/ReadAnalogVoltage

The microcontroller of the board has a circuit inside called an analog-to-digital converter or **ADC** that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, analogRead() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

Therefore, we can calculate the voltage read on the **potPin** by adding one line.

```
float voltage = potValue * (5.0 / 1023.0);
```

So, our program should look like this:

```
const int potPin = A0;

int potValue = 0;
float voltage = 0;

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  potValue = analogRead(potPin); // not the best practice
// Convert the analog reading (which goes from 0 - 1023) to a voltage (0
- 5V):
  voltage = potValue * (5.0 / 1023.0);
  // print out the read voltage and comment value printing
  // Serial.println(potValue);
  Serial.println(voltage);
  delay(1);        // delay in between reads for stability
}
```

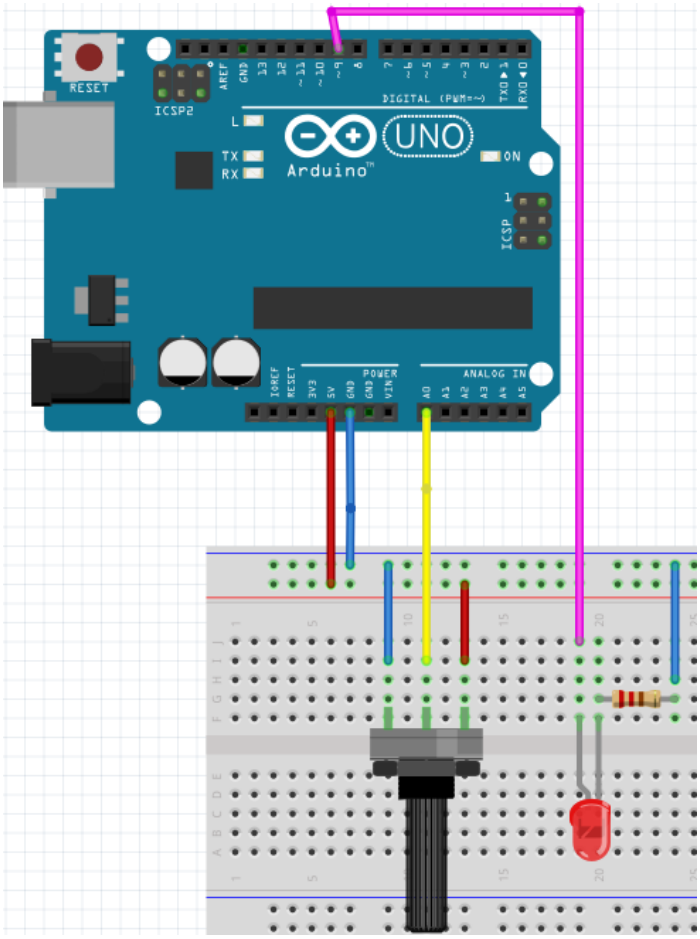
Let's recap the course a bit:

- we connected a **LED** to **D9**, which also has a **~**
- we used analogWrite(ledPin, ledValue), with ledValue = 0..255 to set the intensity of the LED

Now, let's use the potentiometer to control the intensity of the LED.

7. Fading an LED with the potentiometer

For this, we need to read the **potValue**, store it in a variable and write that variable to the LED.
https://www.arduino.cc/en/Tutorial/AnalogInOutSerial



The idea is simple, we have an analogInput and need to control an analogOutput with it

```
const int potPin = A0;
const int ledPin = 9;
// global variables are initialized to 0 by default
int potValue;

void setup() {
```

```
  pinMode(potPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  potValue = analogRead(potPin);
  analogWrite(ledPin, potValue);
}
```

Do that, see what happens.
Questions:

- why does it behave in this way?
- how can we fix it?

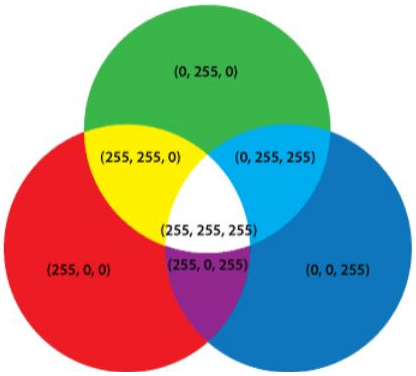
```
const int potPin = A0;
const int ledPin = 9;

int potValue = 0;
int ledValue = 0;

void setup() {
  pinMode(potPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  potValue = analogRead(potPin);
  ledValue = map(potValue, 0, 1023, 0, 255);
  // ledValue = potValue / 4; // this works just as well
  analogWrite(ledPin, ledValue);
}
```

Extra exercise: connect the LED (which goes to **D9**) to the middle pin of the potentiometer. As you can see, it controls the fade, much in the same way Arduino did. We did not need an Arduino to do this, as it is a simple exercise. However, with the help of Arduino we can increase the complexity of the project using the code, while keeping the complexity of the hardware fairly low.

DE ADAUGAT SCHEMA FRITZING

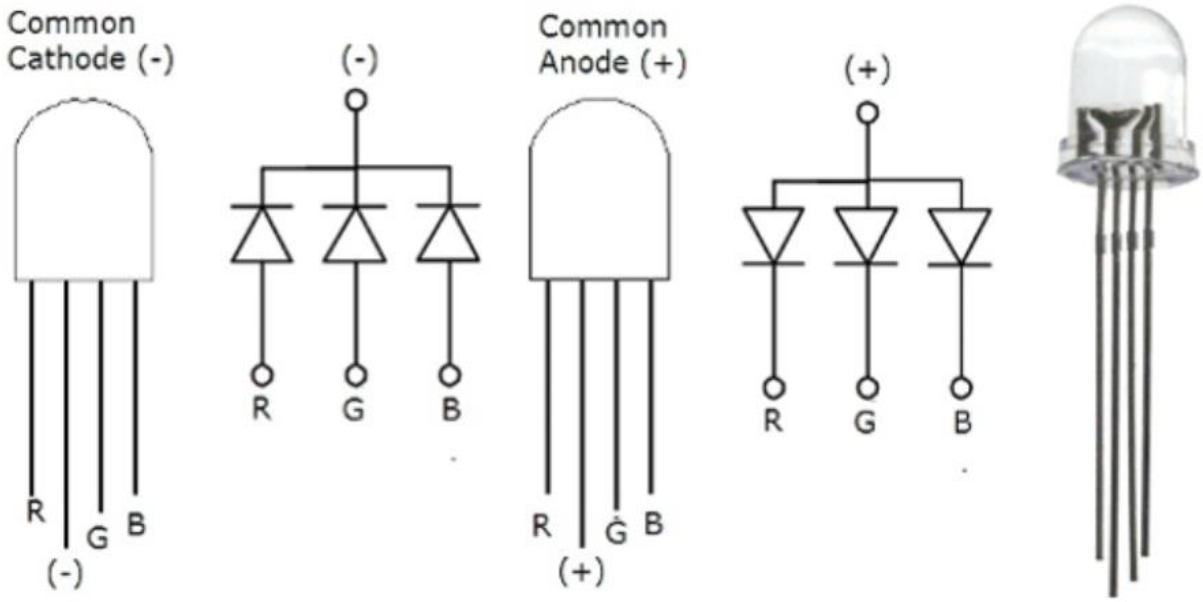
8. Controlling a RGB LED

These LEDs have three tiny LEDs of 3 primary colors (**red, green and blue**) where a terminal is common for all. Some have common positive terminal (anode) and some have common negative terminal (cathode). When different voltage is applied to different LEDs, they make a mixture and produce several thousands of colors.



There are actually two types of RGB LEDs; the common cathode one and common anode one.
With a **common anode** you connect the anode to the +5v and each individual LED to a resistor each. Connect that resistor to an output pin.
Then a write LOW to that pin will turn the LED on and a HIGH will turn it off. This is called **current sinking**.

With a **common cathode** you connect the cathode to ground and connect each LED's anode through a resistor to the output pin.
Then a HIGH turns it on. This is called **current sourcing**.

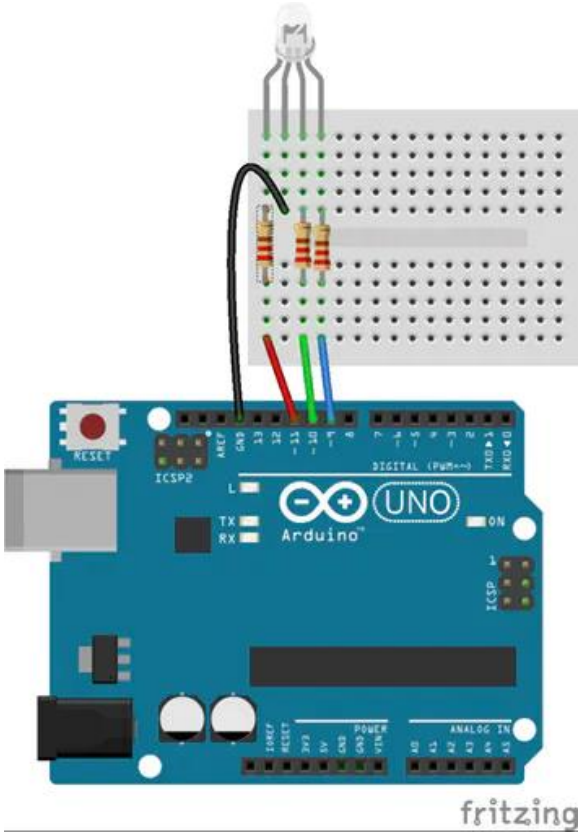


To check if you have a common cathode or common anode you can use a multimeter. Set it to diode and first use **- on the long pin** and **+ on a small one**. If it lights up, it's **common cathode**. If not, use **+ on the long pin and - on a small one**; if it lights up now, then it's **common anode**.

Either way, after finding out which type you have (most likely common cathode), keep the long pin and keep switching between the smaller ones with the multimeter, in order to change between **R** - Red, **G** - Green and **B** - Blue

We will do the examples for common cathode. If you encounter the common anode one, there are 2 practical differences:

1. Instead of GND, connect the long pin to 5V
2. When coding, you must sink current, thus
    a. if on common cathode you write HIGH, on common anode you write LOW
    b. if on common cathode you **analogWrite(pin, value)** on common anode you **analogWrite(pin, 255 - value)**




```
const int redPin = 11;
const int greenPin = 10;
const int bluePin = 9;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
```

```
// play with LOW and HIGH on each of them. Change the pinValues to match
the colors if they don't
  digitalWrite(redPin, HIGH);
  digitalWrite(greenPin, HIGH);
  digitalWrite(bluePin, HIGH);
}
```

However, using digitalWrite only gives us a small fraction of the possible colors combination. Let's switch to analogWrite instead.

```
const int redPin = 11;
const int greenPin = 10;
const int bluePin = 9;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
// play with LOW and HIGH on each of them. Change the pinValues to match
the colors if they don't
  analogWrite(redPin, 133);
  analogWrite(greenPin, 244);
  analogWrite(bluePin, 13);
}
```

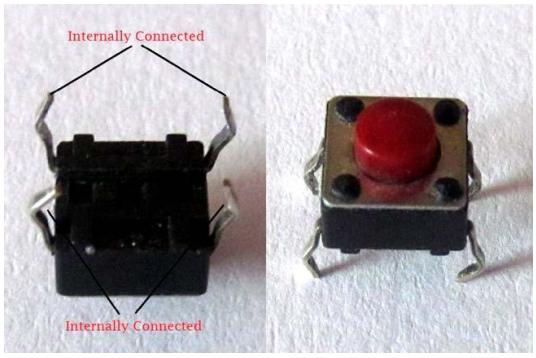
Even better, let's create a function to simplify using it.

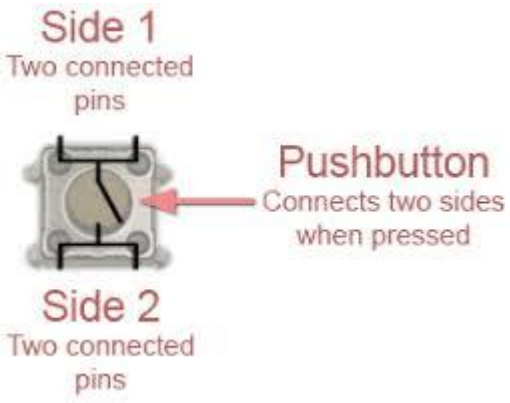
```
const int redPin = 11;
const int greenPin = 10;
const int bluePin = 9;

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop() {
  setColor(255, 0, 0); // red
  delay(1000);
  setColor(0, 255, 0); // green
  delay(1000);
  setColor(0, 0, 255); // blue
  delay(1000);
  setColor(255, 255, 0); // yellow
  delay(1000);
  setColor(255, 0, 255); // purple
  delay(1000);
  setColor(0, 255, 255); // acqua
  delay(1000);
}

void setColor(int red, int green, int blue) {
  analogWrite(redPin, red);
  analogWrite(greenPin, green);
  analogWrite(bluePin, blue);
}
```

9. Pushbuttons

Let's recap the course a bit: What is a push button?

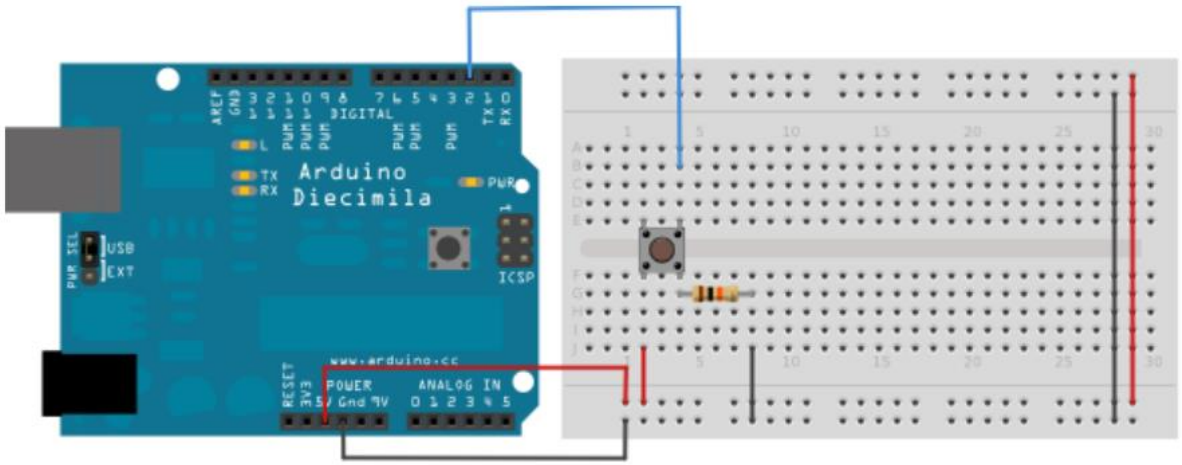




Always remember that the oppositely oriented pins are connected.

https://www.arduino.cc/en/Tutorial/DigitalReadSerial

Pushbuttons or switches connect two points in a circuit when you press them. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and reads as LOW, or 0. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that the pin reads as HIGH, or 1.



5k or 10k ohm resistor works well

```
const int pushButton = 2;
int buttonState = 0;

void setup() {
  Serial.begin(9600);
```

```
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

void loop() {
  // read the input pin:
  buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);       // delay in between reads for stability
}
```

10. Controlling an LED with the pushbutton

https://www.arduino.cc/en/Tutorial/Button

The idea is simple: reading the button value returns **HIGH** or **LOW** which we can write directly to the ledPin. For this, we can use ledPin 13 (and maybe add an LED directly on pin 13 and the GND next to it)

```
const int pushButton = 2;
const int ledPin = 13;

int buttonState = 0;

void setup() {
  Serial.begin(9600);
  pinMode(pushButton, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  buttonState = digitalRead(pushButton);
  digitalWrite(ledPin, buttonState);
}
```

Works great, but let's check something out. Let's do a button press counter.

```
const int pushButton = 2;
const int ledPin = 13;

int lastButtonState = 0;
```

```
int buttonState = 0;
int buttonPressCount = 0;

void setup() {
  Serial.begin(9600);
  pinMode(pushButton, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the input pin:
  buttonState = digitalRead(pushButton);
  digitalWrite(ledPin, buttonState);
  if (buttonState == HIGH) {
      buttonPressCount++;
  }
  Serial.println(buttonPressCount);
}
```

Press the button, removing the finger as fast as possible.
Questions:

- how does the counter grow? (more times per press)
- why? (because it does many readings in a fraction of a second)
- how can we fix this?

11. Debounce

https://www.arduino.cc/en/Tutorial/Debounce

Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program. This example demonstrates how to debounce an input, which means checking twice in a short period of time to make sure the pushbutton is definitely pressed. Without debouncing, pressing the button once may cause unpredictable results. This sketch uses the millis() function to keep track of the time passed since the button was pressed.

```
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin = 13;       // the number of the LED pin

int ledState = HIGH;         // the current state of the output pin
int buttonState;             // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin

// the following variables are unsigned longs because the time, measured
in
// milliseconds, will quickly become a bigger number than can be stored
in an int.
unsigned long lastDebounceTime = 0;  // the last time the output pin was
toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if
the output flickers

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);
}

void loop() {
  int reading = digitalRead(buttonPin);

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH), and you've waited long
enough
  // since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
```

```
    // whatever the reading is at, it's been there for longer than the
debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }

  // set the LED:
  digitalWrite(ledPin, ledState);

  // save the reading. Next time through the loop, it'll be the
lastButtonState:
  lastButtonState = reading;
}
```

12. Extra

(in case you finish earlier)
https://www.arduino.cc/en/Tutorial/StateChangeDetection

13. Homework

V1, if you didn't get to do debounce:
Control the color of an RGB LED with 3 potentiometers, one connected to each of the color pins.
Daca le dati tema asta, mai dati-le si cate un potentiometru din cutia de la usa. Trebuie sa aiba 3 in total.

V2, if you did debounce as well:
Control the color of an RGB LED with 1 potentiometer and 1 button. Each time you press the button, it changes the color PIN the potentiometer controls.