

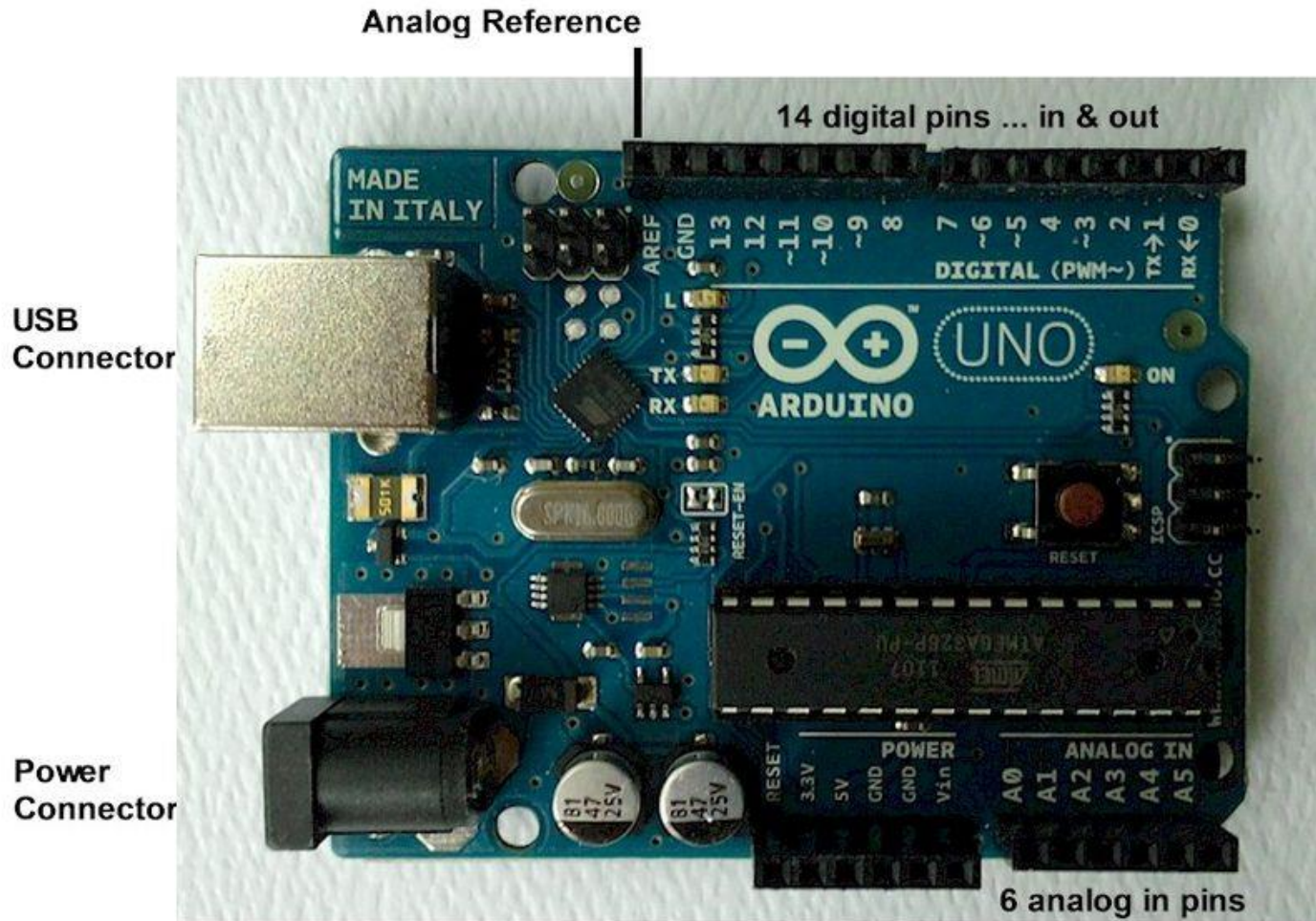
Aulas de Arduino uno

Book: Beginning Arduino-

Copyright © 2010 by Michael McRoberts

What is an Arduino?? (From Wikipedia, the free encyclopedia)

Arduino is a [single-board microcontroller](#) to make using electronics in [multidisciplinary](#) projects more accessible.



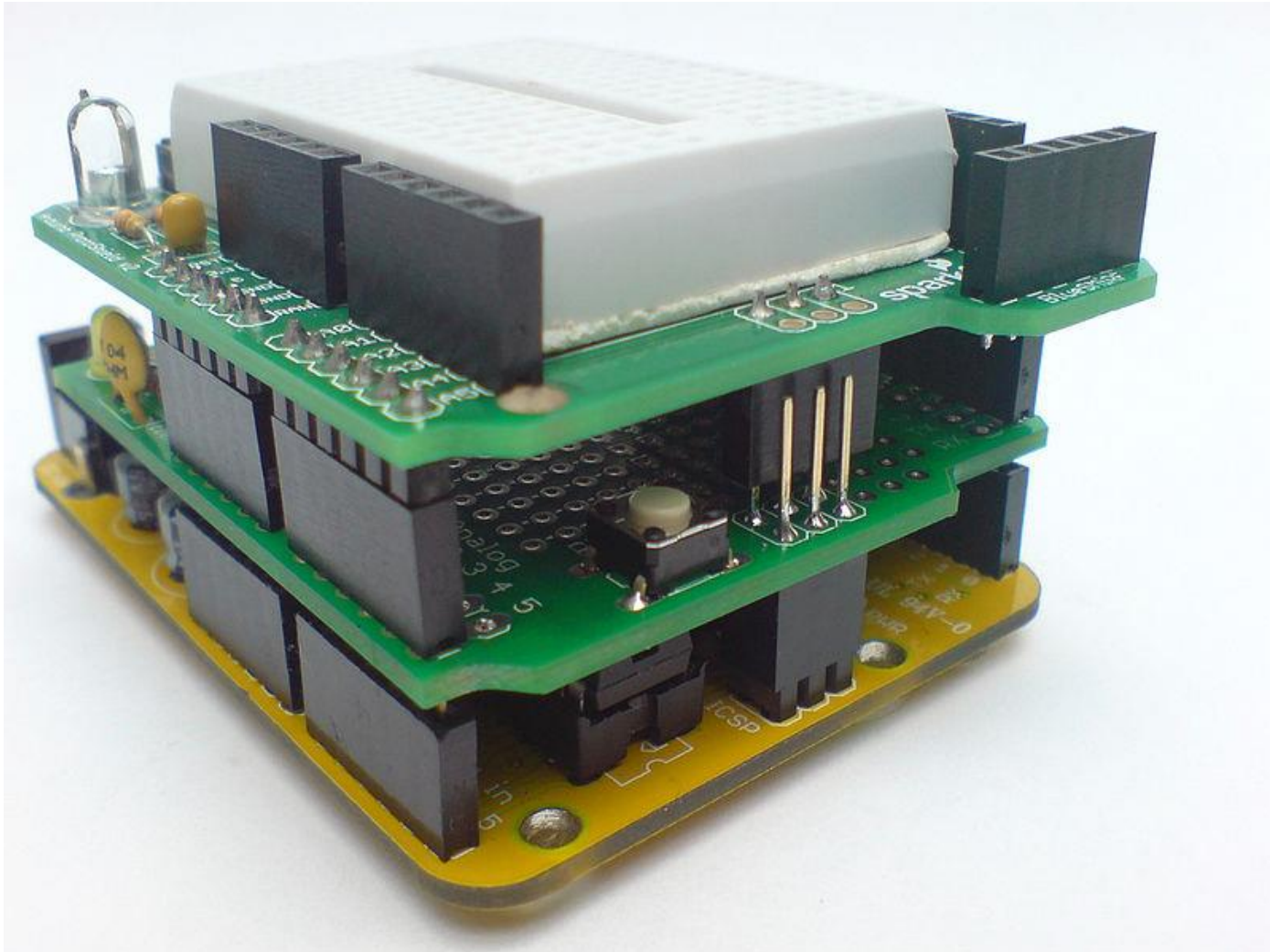
What is an Arduino?? (From Wikipedia, the free encyclopedia)

The hardware consists of a simple [open-source hardware](#) board designed around an 8-bit [Atmel AVR](#) microcontroller.

The software consists of a standard programming language compiler and a [boot loader](#) that executes on the [microcontroller](#).

Arduino boards can be purchased pre-assembled or as [do-it-yourself](#) kits. Hardware design information is available for those who would like to assemble an Arduino by hand.

Arduino and Arduino-compatible boards make use of *shields*—printed circuit expansion boards that plug into the normally supplied Arduino pin-headers. Shields can provide motor controls, [GPS](#), ethernet, LCD display, or [breadboarding](#) (prototyping)



Getting Started

How to set up your Arduino and the IDE (Integrated Development Environment) for the first time.



Arduino!!

- <http://www.youtube.com/watch?v=KZUrO9aXGh0>

Howto set up your Arduino and the IDE for the first time.

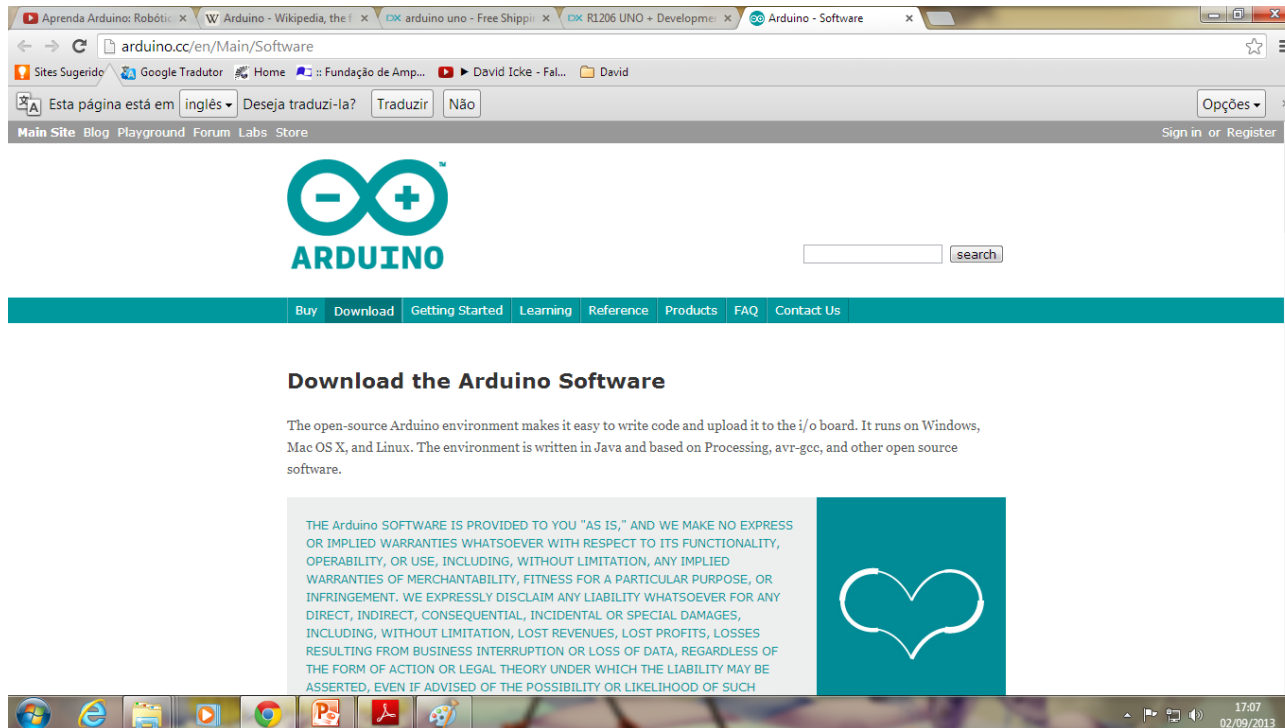
- Refer to the Getting Started instructions on the Arduino website at www.arduino.cc/playground/Learning

We need a USB cable (A to B plug type) which is the same kind of cable used for most modern USB printers.



Download the Arduino IDE. This is the software you will use to write your programs (or sketches) and upload them to your board.

For the latest IDE go to the Arduino download page at <http://arduino.cc/en/Main/Software>.



- + Arduino 1.0.3: [Windows](#), [Mac OS X](#), [Linux](#): (32 bit, 64 bit), source- hosted by [Google Code](#)
- + Arduino 1.0.2: [Windows](#), [Mac OS X](#), [Linux](#): (32 bit, 64 bit), source- hosted by [Google Code](#)
- + Arduino 1.0.1: [Windows](#), [Mac OS X](#), [Linux](#): (32 bit, 64 bit), source - hosted by [Google Code](#)
- + Arduino 1.0 (release notes): [Windows](#), [Mac OS X](#), [Linux](#) (32 bit) [64 bit](#), source - hosted by [Google Code](#)
Also available from [Arduino.cc](#): [Windows](#), [Mac OS X](#), [Linux](#) (32bit) (64bit), [source](#)
- + Arduino 0023 (release notes): [Windows](#), [Mac OS X](#), [Linux](#) (32 bit) [64 bit](#) - hosted by [Google Code](#)
Also available from [Arduino.cc](#): [Windows](#), [Mac OS X](#), [Linux](#) (32bit) (64bit)
- + Arduino 0022 (release notes): [Windows](#), [Mac OS X](#), [Linux](#) (32 bit) [64 bit](#) - hosted by [Google Code](#)
Also available from [Arduino.cc](#): [Windows](#), [Mac OS X](#), [Linux](#) (32bit) (64bit), [Source](#)
- + Arduino 0021 (release notes): [Windows](#), [Mac OS X](#), [Linux](#) (32 bit) (hosted by [Google Code](#))
Also available from [Arduino.cc](#): [Windows](#), [Mac OS X](#), [Linux](#) (32bit) (64bit), [Source](#)
- + Arduino 0020 (release notes): [Windows](#), [Mac OS X](#) (hosted by [Google Code](#))
Also available from [Arduino.cc](#): [Windows](#), [Mac OS X](#)
- + Arduino 0019 (release notes): [Windows](#), [Mac OS X](#), [Linux](#) (32 bit) (hosted by [Google Code](#))
Also available from [Arduino.cc](#): [Windows](#), [Mac OS X](#), [Linux](#) (32bit)

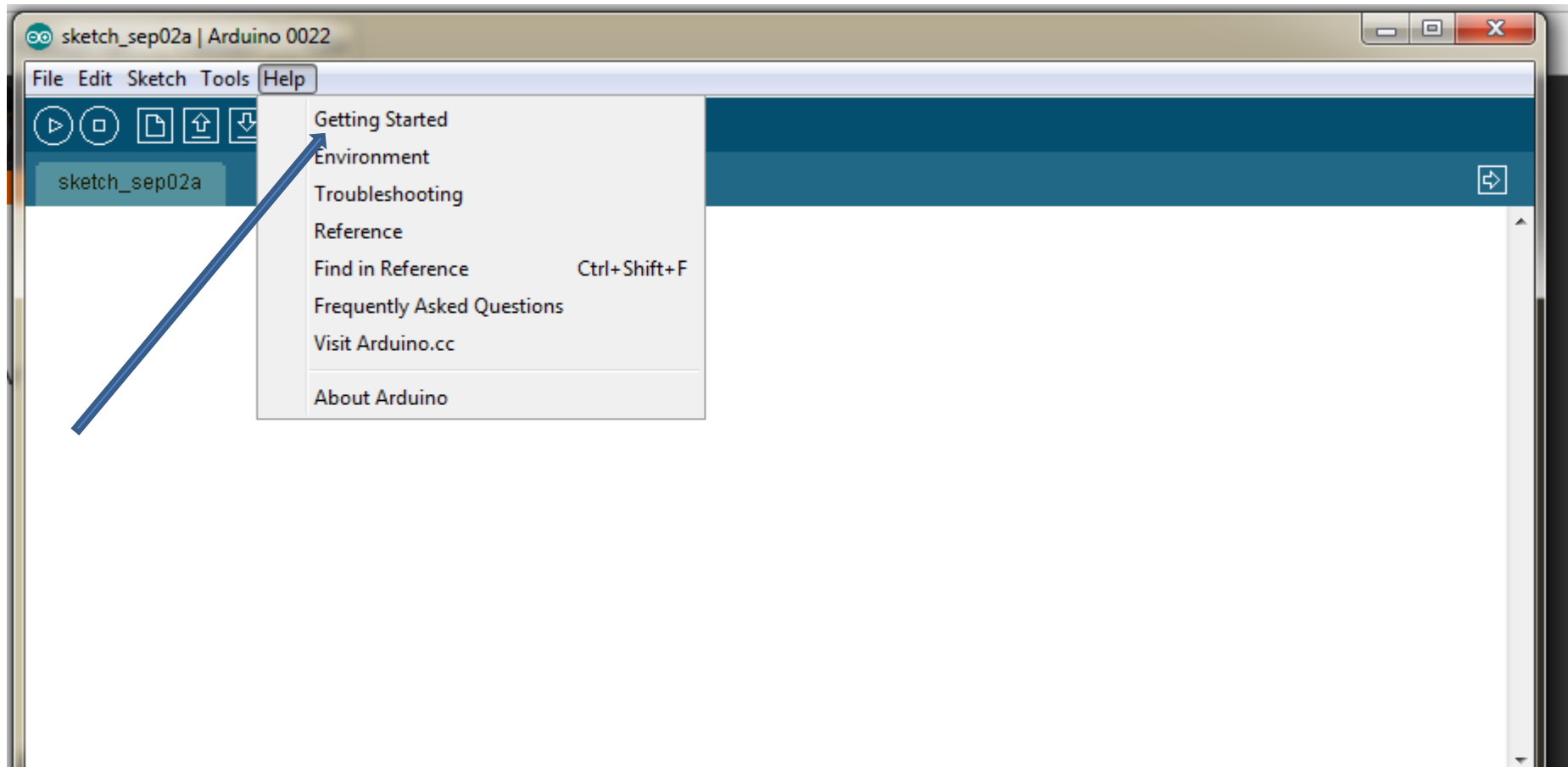




sketch_sep02a



Use of the file-Menu to install your Arduino.



3 | Connect the board

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either the USB connection to the computer or an external power supply. If you're using an Arduino Diecimila, you'll need to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it's on the two pins closest to the USB port.

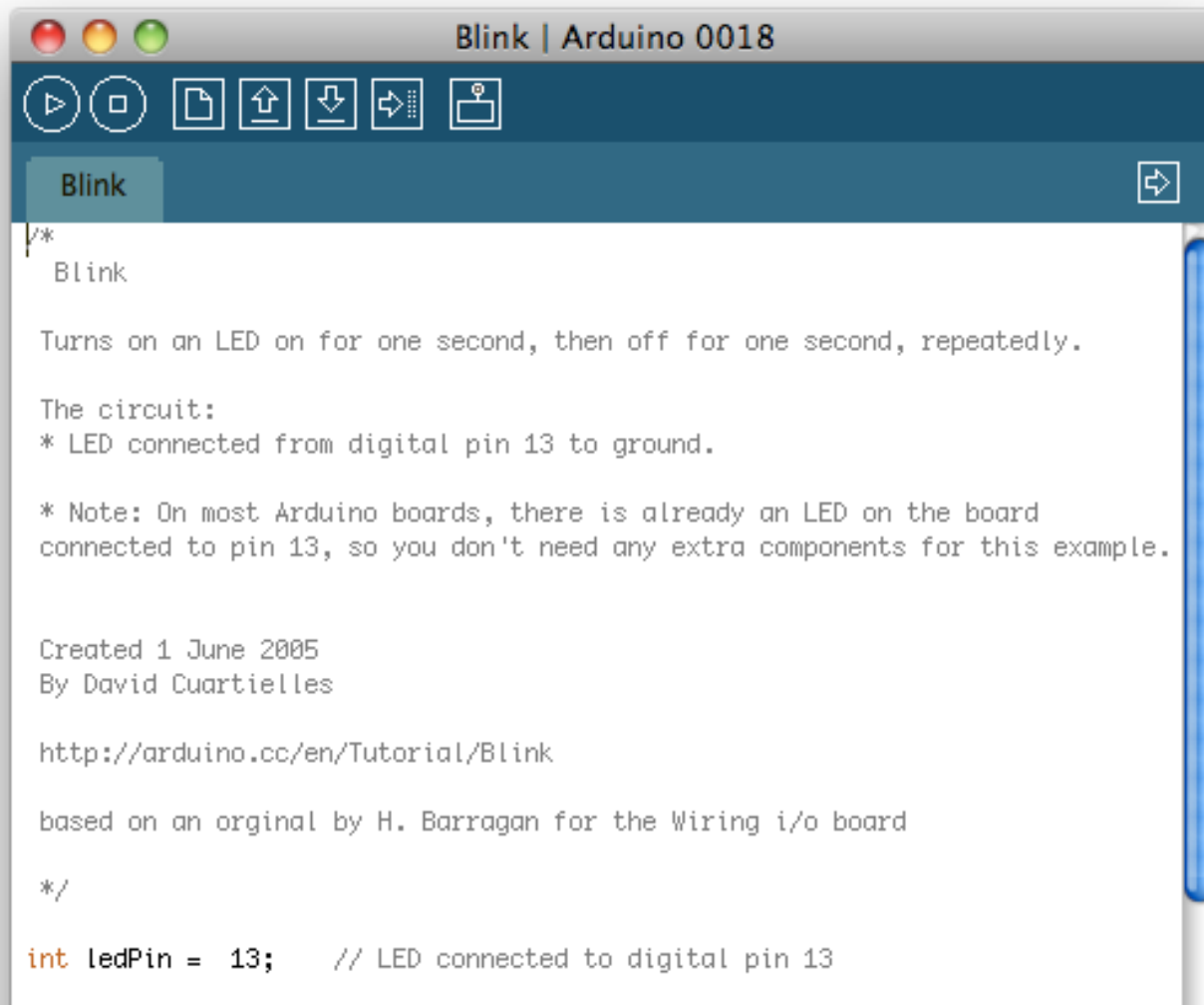
Connect the Arduino board to your computer using the USB cable. The green power LED (labelled **PWR**) should go on.

4 | Install the drivers

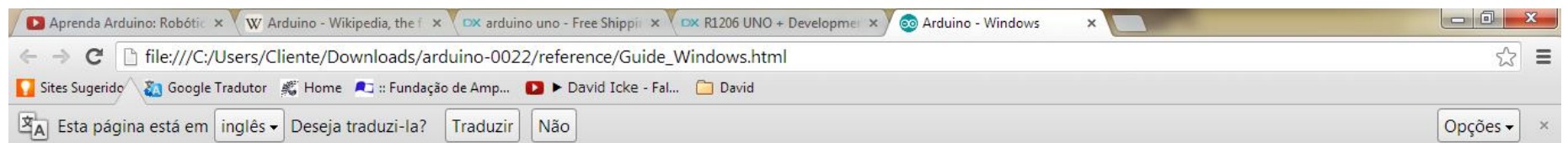
Installing drivers for the Arduino Uno with Windows7, Vista, or XP:

- Plug in your board and wait for Windows to begin its driver installation process. After a few moments, the process will fail, despite its best efforts
 - Click on the Start Menu, and open up the Control Panel.
 - While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
 - Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)"
 - Right click on the "Arduino UNO (COMxx)" port and choose the "Update Driver Software" option.
 - Next, choose the "Browse my computer for Driver software" option.
 - Finally, navigate to and select the Uno's driver file, named "**ArduinoUNO.inf**", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
-

Open the LED blink example sketch: **File > Examples > 1.Basics > Blink.**

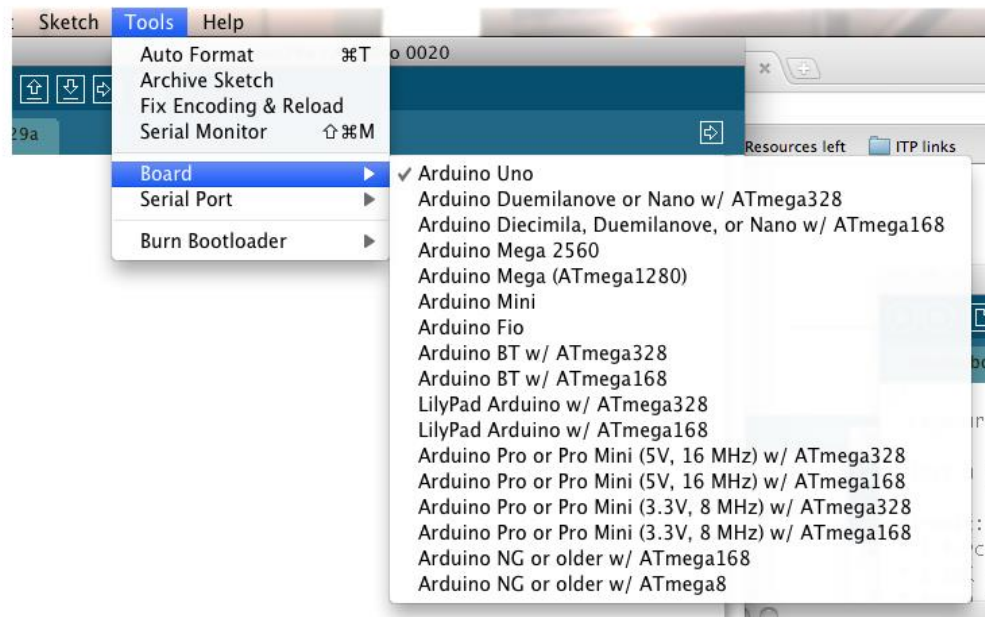


```
/*  
  Blink  
  
  Turns on an LED on for one second, then off for one second, repeatedly.  
  
  The circuit:  
  * LED connected from digital pin 13 to ground.  
  
  * Note: On most Arduino boards, there is already an LED on the board  
  connected to pin 13, so you don't need any extra components for this example.  
  
  Created 1 June 2005  
  By David Cuartielles  
  
  http://arduino.cc/en/Tutorial/Blink  
  
  based on an original by H. Barragan for the Wiring i/o board  
  
  */  
  
int ledPin = 13;    // LED connected to digital pin 13
```

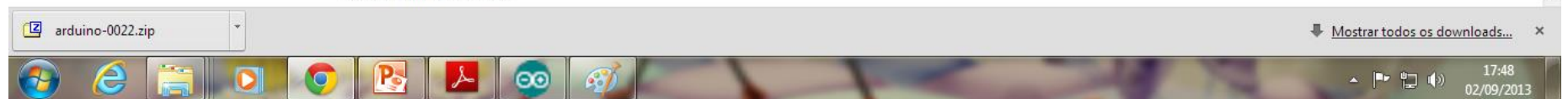


7 | Select your board

You'll need to select the entry in the **Tools > Board** menu that corresponds to your Arduino.



Selecting an Arduino Uno



8 | Upload the program

Now, simply click the "Upload" button in the environment. Wait a few seconds - you should see the RX and TX leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar. (*Note:* If you have an Arduino Mini, NG, or other board, you'll need to physically present the reset button on the board immediately before pressing the upload button.)

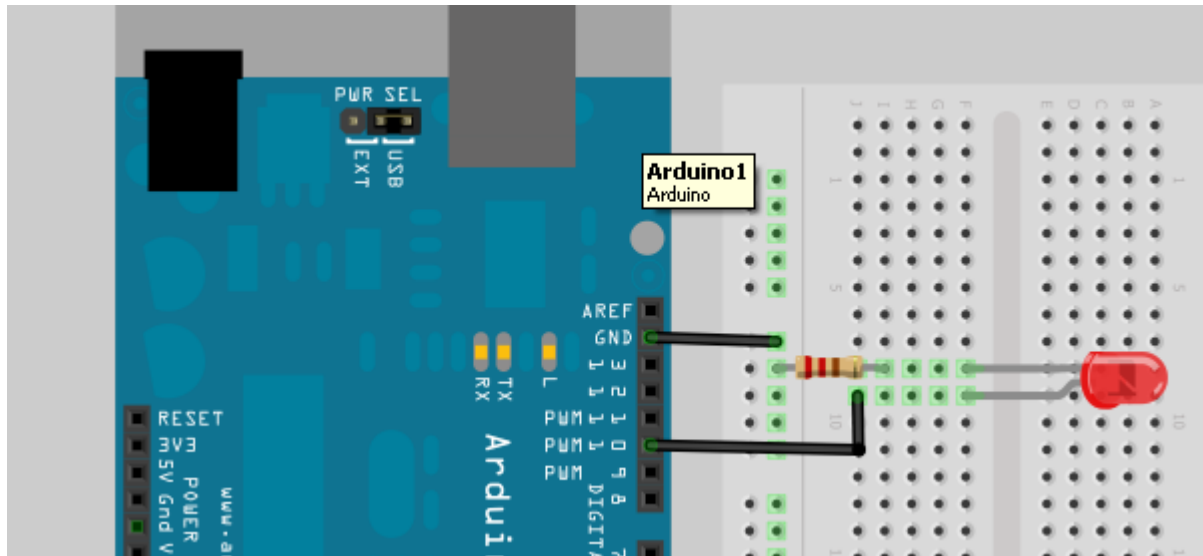


A few seconds after the upload finishes, you should see the pin 13 (L) LED on the board start to blink (in orange). If it does, congratulations! You've gotten Arduino up-and-running.

"Go Down the Rabbit Hole"

Aula 1 LED Flasher

Hardware:



Code for “LED flasher”

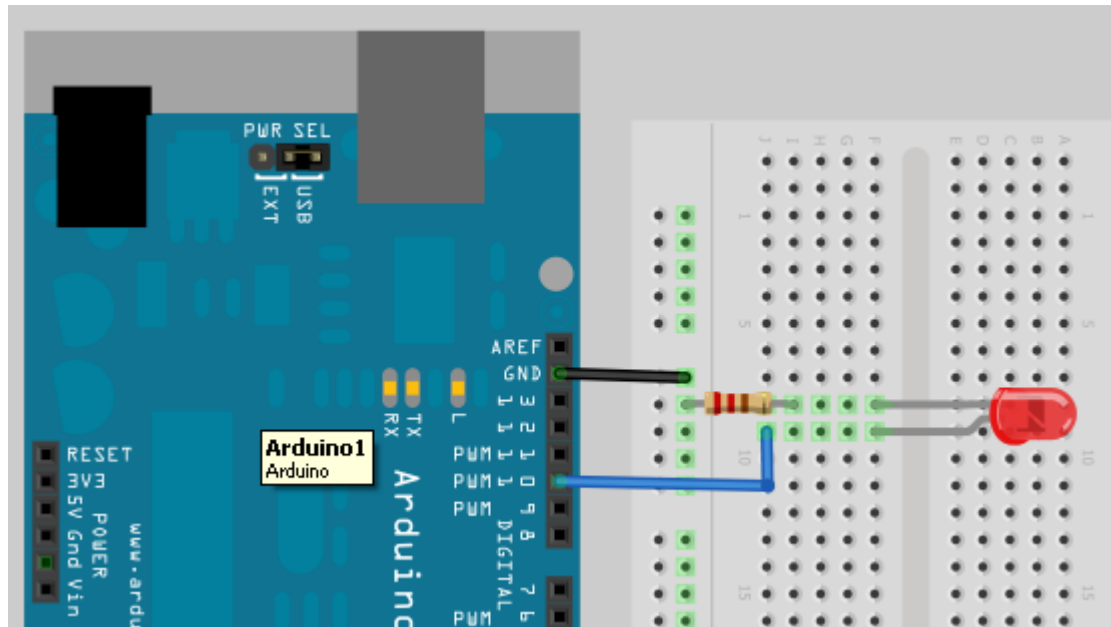
- **// Project 1 - LED Flasher**
- **int ledPin = 10;**
 - **void setup() {**
 - **pinMode(ledPin, OUTPUT);**
- **}**
 - **void loop() {**
 - **digitalWrite(ledPin, HIGH);**
 - **delay(1000);**
 - **digitalWrite(ledPin, LOW);**
 - **delay(1000);**
- **}**

- By varying the on and off times of the LED you can create any effect you want.
- For example, if you would like the LED to stay off for 5 seconds and then flash briefly (250ms), like the LED indicator on a car alarm, you could do this:

```
void loop() {  
digitalWrite(ledPin, HIGH);  
delay(250);  
digitalWrite(ledPin, LOW);  
delay(5000);  
}
```

Aula 2 S.O.S. Morse Code Signaler

Hardware:

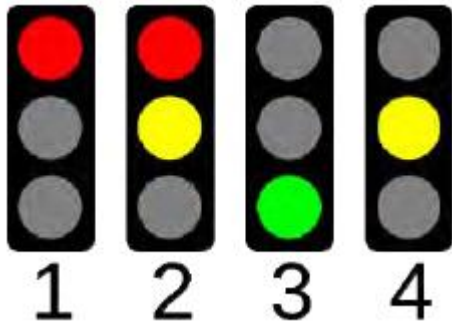


Code for “SOS”

```
// LED connected to pin 10
int ledPin = 10;
// run once, when the sketch starts
void setup()
{
  // sets the pin as output
  pinMode(ledPin, OUTPUT);
}
// run over and over again
void loop()
{
  // 3 dits
  for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH); // sets the LED on
    delay(150); // waits for 150ms
    digitalWrite(ledPin, LOW); // sets the LED off
    delay(100); // waits for 100ms
  }
  // 100ms delay between letters
  delay(100);
  // 3 dahs
  for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH); // sets the LED on
    delay(400); // waits for 400ms
    digitalWrite(ledPin, LOW); // sets the LED off
    delay(100); // waits for 100ms
  }
  // 100ms delay between letters
  delay(100);
  // 3 dits again
  for (int x=0; x<3; x++) {
    digitalWrite(ledPin, HIGH); // sets the LED on
    delay(150); // waits for 150ms
    digitalWrite(ledPin, LOW); // sets the LED off
    delay(100); // waits for 100ms
  }
  // wait 5 seconds before repeating
  delay(5000);
}
```

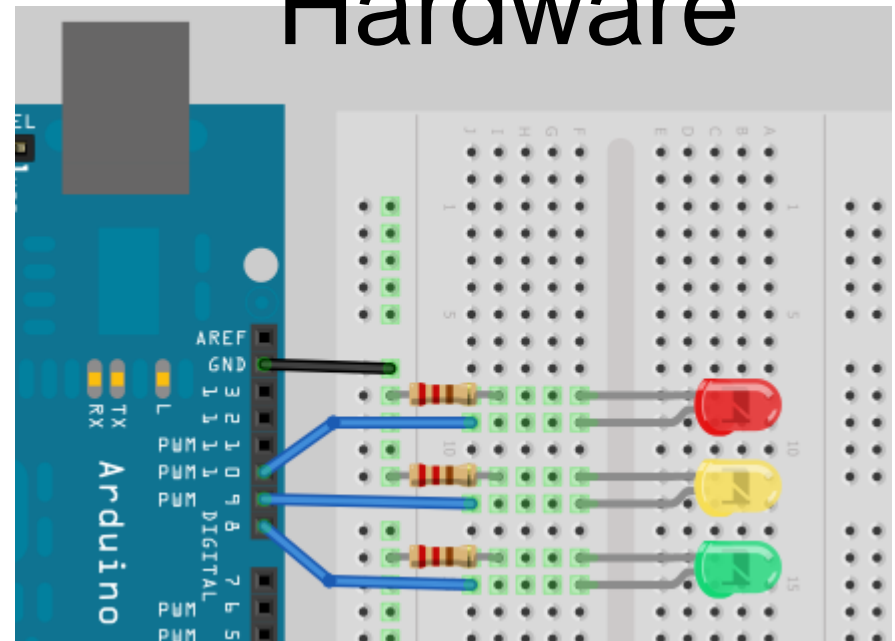
Project 3 – Traffic Lights

The project creates a set of traffic lights that will change from green to red, via amber, and back again, after a set length of time using the four-state UK system. This project could be used to make a set of working traffic lights for a model railway



The four states of the UK traffic light system

Hardware



Code for “Traffic Lights”

```
// Project 3 - Traffic Lights
int ledDelay = 5000; /* delay in
between changes*/
int redPin = 10;
int yellowPin = 9;
int greenPin = 8;
void setup() {
pinMode(redPin, OUTPUT);
pinMode(yellowPin, OUTPUT);
pinMode(greenPin, OUTPUT);
} void loop() {
digitalWrite(redPin, HIGH);
delay(ledDelay); // wait 5 seconds
```

```
digitalWrite(yellowPin, HIGH);
delay(2000); // wait 2 seconds
digitalWrite(greenPin, HIGH);
digitalWrite(redPin, LOW);
digitalWrite(yellowPin, LOW);
delay(ledDelay);
digitalWrite(yellowPin, HIGH);
digitalWrite(greenPin, LOW);
delay(2000); // wait 2 seconds
digitalWrite(yellowPin, LOW);
// now our loop repeats
}
```

Project 4 – “Interactive Traffic Lights”

In this project is included a set of pedestrian lights to the “traffic lights” program and a pedestrian push button to request to cross the road.

The Arduino will react when the button is pressed by changing the state of the lights to make the cars stop and allow the pedestrian to cross safely. We also interact with the Arduino and cause it to do something when we change the state of a button that the Arduino is watching. We learn how to create our own functions in code.

2 Red Diffused LEDs



Yellow Diffused LED



2 Green Diffused LEDs



150 ohm Resistor



4 Resistors



Pushbutton



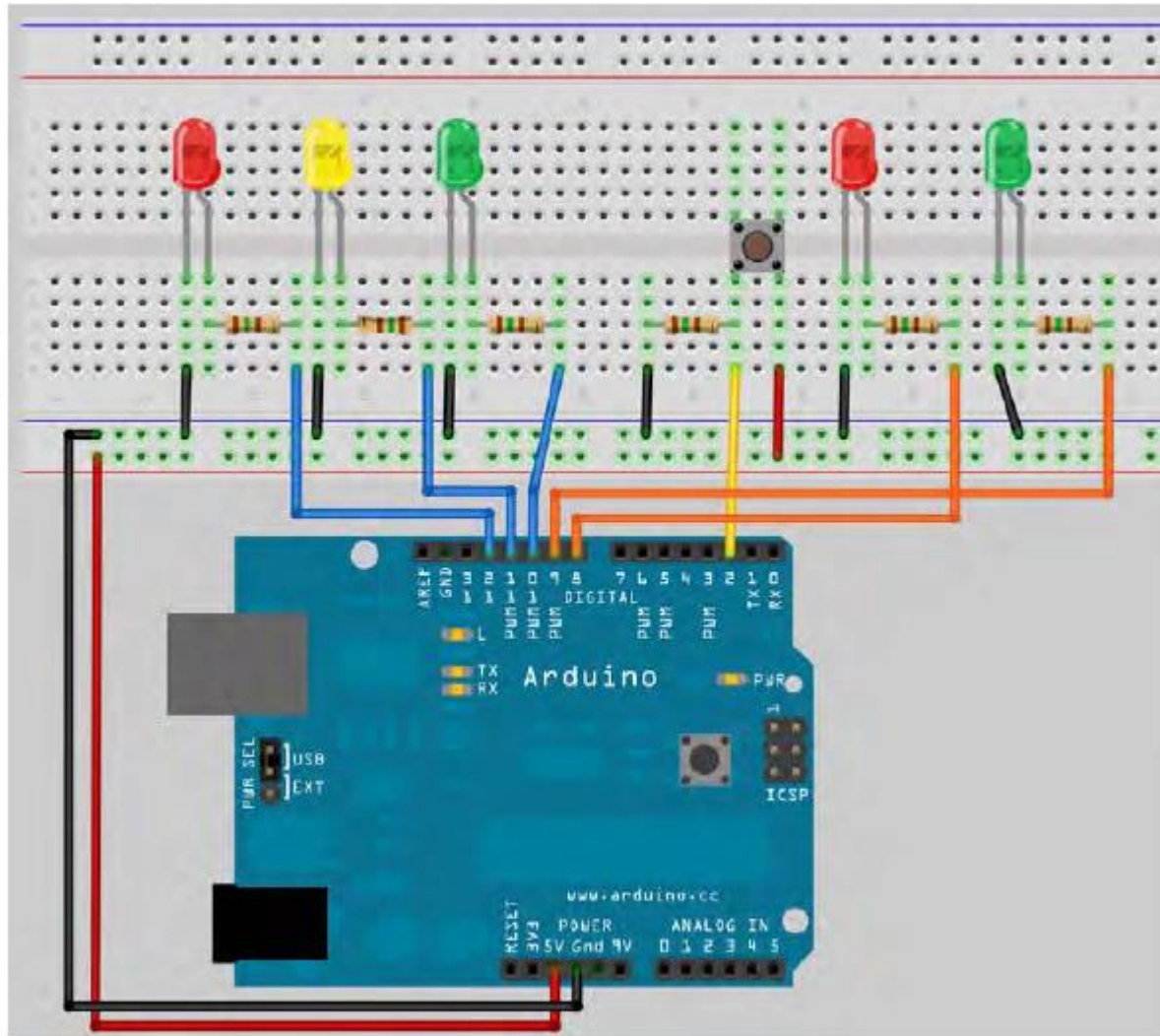
Interactive Traffic Lights

When you run the program, it begins with the car traffic light on green to allow cars to pass and the pedestrian light on red.

When you press the button, the program checks that at least 5 seconds have gone by since the last time the lights changed (to allow traffic to get moving), and if so, passes code execution to the function you have created: **changeLights()**.

In this function, the car lights go from green to amber to red, and then the pedestrian lights go green. After the period of time set in the variable `crossTime` (time enough to allow the pedestrians to cross), the green pedestrian light flash on and off as a warning to the pedestrians to hurry because the lights are about to change to red. Then the pedestrian light changes to red, the vehicle lights go from red to amber to green, and the traffic flow resumes.

Hardware



```
// Project 4 - Interactive Traffic Lights
int carRed = 12; // assign the car lights
int carYellow = 11;
int carGreen = 10;
int pedRed = 9; // assign the pedestrian
//lights
int pedGreen = 8;
int button = 2; // button pin
int crossTime = 5000; // time to cross
unsigned long changeTime; // time since
//button pressed
    void setup() {
        pinMode(carRed, OUTPUT);
        pinMode(carYellow, OUTPUT);
        pinMode(carGreen, OUTPUT);
        pinMode(pedRed, OUTPUT);
        pinMode(pedGreen, OUTPUT);
        pinMode(button, INPUT);
// button pin 2
// turn on the green light
digitalWrite(carGreen, HIGH);
        digitalWrite(pedRed, HIGH);
    }
```

```
void loop() {
    int state = digitalRead(button);
    /* check if button is pressed and it is over 5
seconds since last button press */
        if (state == HIGH && (millis() -
changeTime) > 5000) {
// Call the function to change the lights
                changeLights();
        }
    }
void changeLights() {
digitalWrite(carGreen, LOW); // green off
digitalWrite(carYellow, HIGH); // yellow on
delay(2000); // wait 2 seconds
digitalWrite(carYellow, LOW); // yellow off
digitalWrite(carRed, HIGH); // red on
delay(1000); // wait 1 second till its safe
digitalWrite(pedRed, LOW); // ped red off
digitalWrite(pedGreen, HIGH); // ped green
on
delay(crossTime); // wait for preset time
period
```

```
// flash the ped green
for (int x=0; x<10; x++) {
digitalWrite(pedGreen, HIGH);
delay(250);
digitalWrite(pedGreen, LOW);
delay(250);
}
// turn ped red on
digitalWrite(pedRed, HIGH);
delay(500);
```

```
digitalWrite(carYellow, HIGH); // yellow on
digitalWrite(carRed, LOW); // red off
delay(1000);
digitalWrite(carGreen, HIGH);
digitalWrite(carYellow, LOW); // yellow off
// record the time since last change of lights
changeTime = millis();
// then return to the main program loop
}
```

Variables take up space in memory; the larger the number, the more memory is used up for storing variables.

On a small microcontroller like the Arduino's Atmega32, it's essential that you use the smallest variable data type necessary for your purpose.

unsigned long changeTime;

Integer data types, can store a number between -32,768 and 32,767 (2^{15}). Data type of *long*, can store a number from -2,147,483,648 to 2,147,483,647. An *unsigned long* means the variable cannot store negative numbers, so the range is from 0 to 4,294,967,295. Observe that:

4294967295 * 1ms = 4294967 seconds ($2^{32} = 4294967295$)

4294967 seconds = 71582 minutes

71582 minutes - 1193 hours

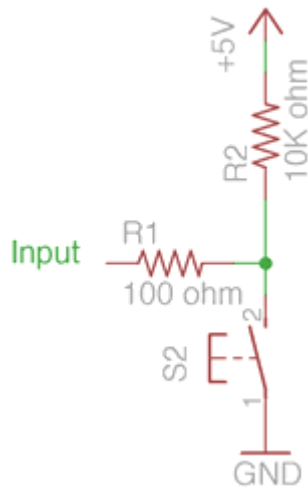
1193 hours - 49 days

Table 2-2. Data types

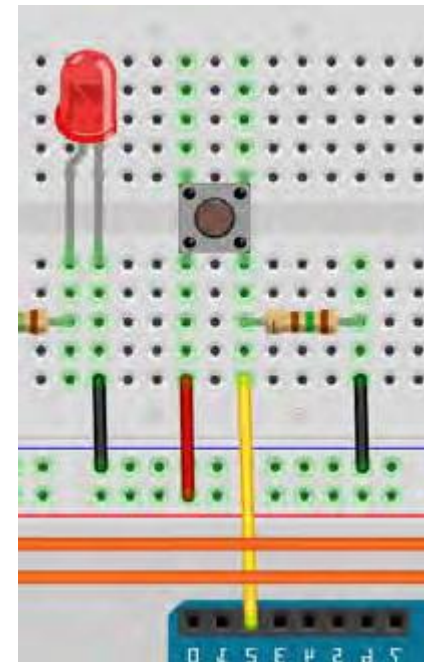
| Data type | RAM | Number Range |
|------------------|------------|---------------------------------|
| void keyword | N/A | N/A |
| boolean | 1 byte | 0 to 1 (True or False) |
| byte | 1 byte | 0 to 255 |
| char | 1 byte | -128 to 127 |
| unsigned char | 1 byte | 0 to 255 |
| int | 2 byte | -32,768 to 32,767 |
| unsigned int | 2 byte | 0 to 65,535 |
| word | 2 byte | 0 to 65,535 |
| long | 4 byte | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 4 byte | 0 to 4,294,967,295 |
| float | 4 byte | -3.4028235E+38 to 3.4028235E+38 |
| double | 4 byte | -3.4028235E+38 to 3.4028235E+38 |
| string | 1 byte + x | Arrays of chars |
| array | 1 byte + x | Collection of variables |

Pull-up and pull-down resistors

Pull-up resistor



Pull-down resistor



The Arduino's Internal Pull-Up Resistors

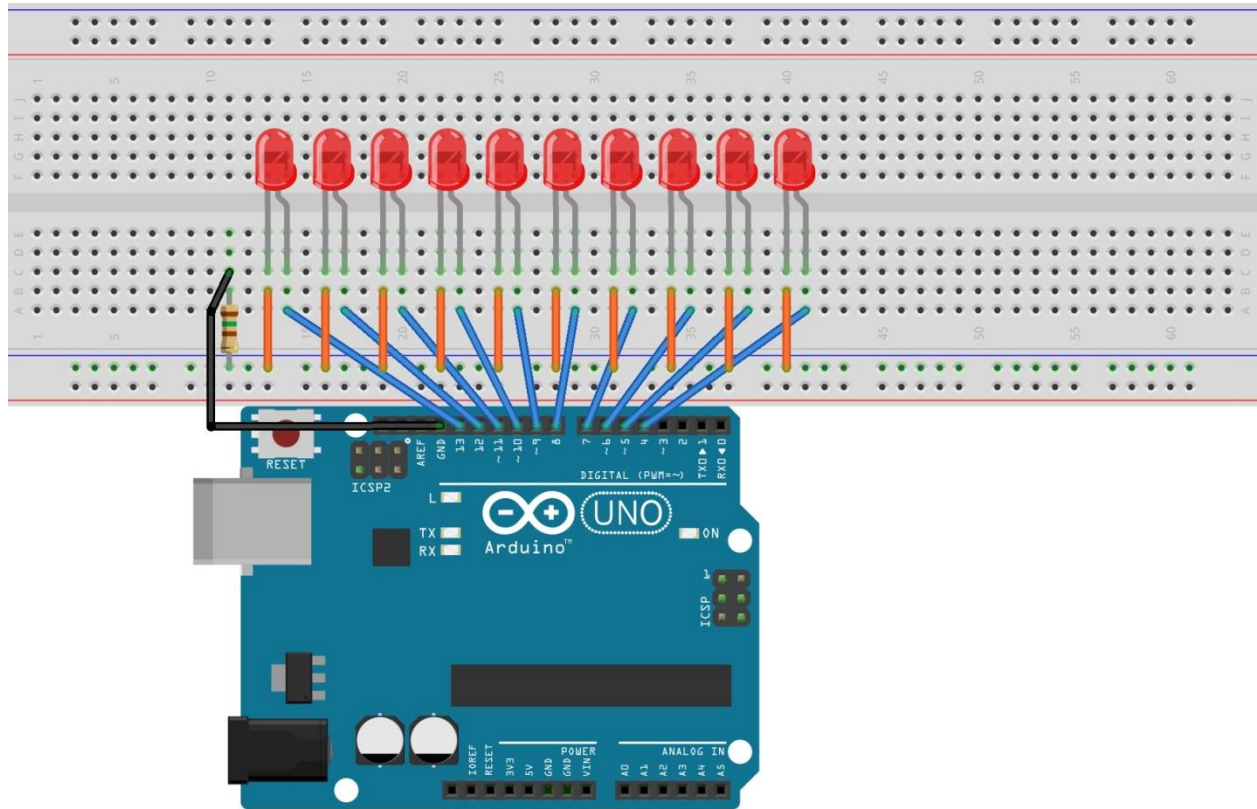
Conveniently, the Arduino contains pull-up resistors that are connected to the pins (the analog pins have pull-up resistors also). These have a value of 20K ohms and need to be activated within software to use them. To activate an internal pull-up resistor on a pin, you first need to change the **pinMode** of the pin to an INPUT and then write a HIGH to that pin using a **digitalWrite** command:

```
pinMode(pin, INPUT);  
digitalWrite(pin, HIGH);
```

If you change the pinMode from INPUT to OUTPUT after activating the internal pull-up resistors, the pin will remain in a HIGH state. This also works in reverse: an output pin that was in a HIGH state and is subsequently switched to an INPUT mode will have its internal pull-up resistors enabled.

Project 5 – LED Chase Effect

You're going to use a string of LEDs (10 in total) to make an LED chase effect. This project will introduce the concept of arrays.



Code for Project 5

```
// Project 5 - LED Chase Effect  
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12,  
13}; // Create array for LED pins  
int ledDelay(65); // delay between  
//changes  
int direction = 1;  
int currentLED = 0;  
unsigned long changeTime;  
    void setup() {  
        for (int x=0; x<10; x++) {  
            // set all pins to output  
            pinMode(ledPin[x], OUTPUT);  
        }  
changeTime = millis();  
}  
    void loop() {  
        if ((millis() - changeTime) >  
        ledDelay) { // if it has been  
        ledDelay ms since  
        //last change
```

```
        changeLED();  
        changeTime = millis();  
        }  
    }  
    void changeLED() {  
        for (int x=0; x<10; x++) { // turn  
off all LED's  
            digitalWrite(ledPin[x], LOW);  
        }  
        digitalWrite(ledPin[currentLED],  
HIGH); // turn on the current LED  
        currentLED += direction;  
        //increment by the direction  
        //value change direction if we  
        //reach the end  
        if (currentLED == 9) {direction = -  
        1;}  
        if (currentLED == 0) {direction =  
        1;}  
    }
```

Syntax (new)

`x += y; // equivalent to the expression x = x + y;`

`x -= y; // equivalent to the expression x = x - y;`

`x *= y; // equivalent to the expression x = x * y;`

`x /= y; // equivalent to the expression x = x / y;`

Parameters

`x`: any variable type

`y`: any variable type or constant

Examples

`x = 2;`

`x += 4; // x now contains 6`

`x -= 3; // x now contains 3`

`x *= 10; // x now contains 30`

`x /= 2; // x now contains 15`

Note that we have to stop the program, manually change the value of **ledDelay**, and then upload the amended code to see any changes.

But, it is possible to adjust the speed while the program is running. We will interact with the program and adjust the speed using a potentiometer. See project 6.

Project 6 – Interactive LED Chase Effect

4.7K Ω Rotary Potentiometer



Project 6

The code for this Project is almost identical to the previous project. We simply added a potentiometer to the hardware and the code additions enable it to read the values from the potentiometer and use them to adjust the speed of the LED chase effect.

int potPin = 2;

potentiometer is connected to Analog Pin 2. The Arduino has six analog input/outputs with a 10-bit analog to digital converter. This means the analog pin can read in voltages between 0 to 5 volts *in integer values* between 0 (0 volts) and 1,023 (5 volts). This gives a resolution of 5 volts / 1024 units or 0.0049 volts (4.9mV) per unit.

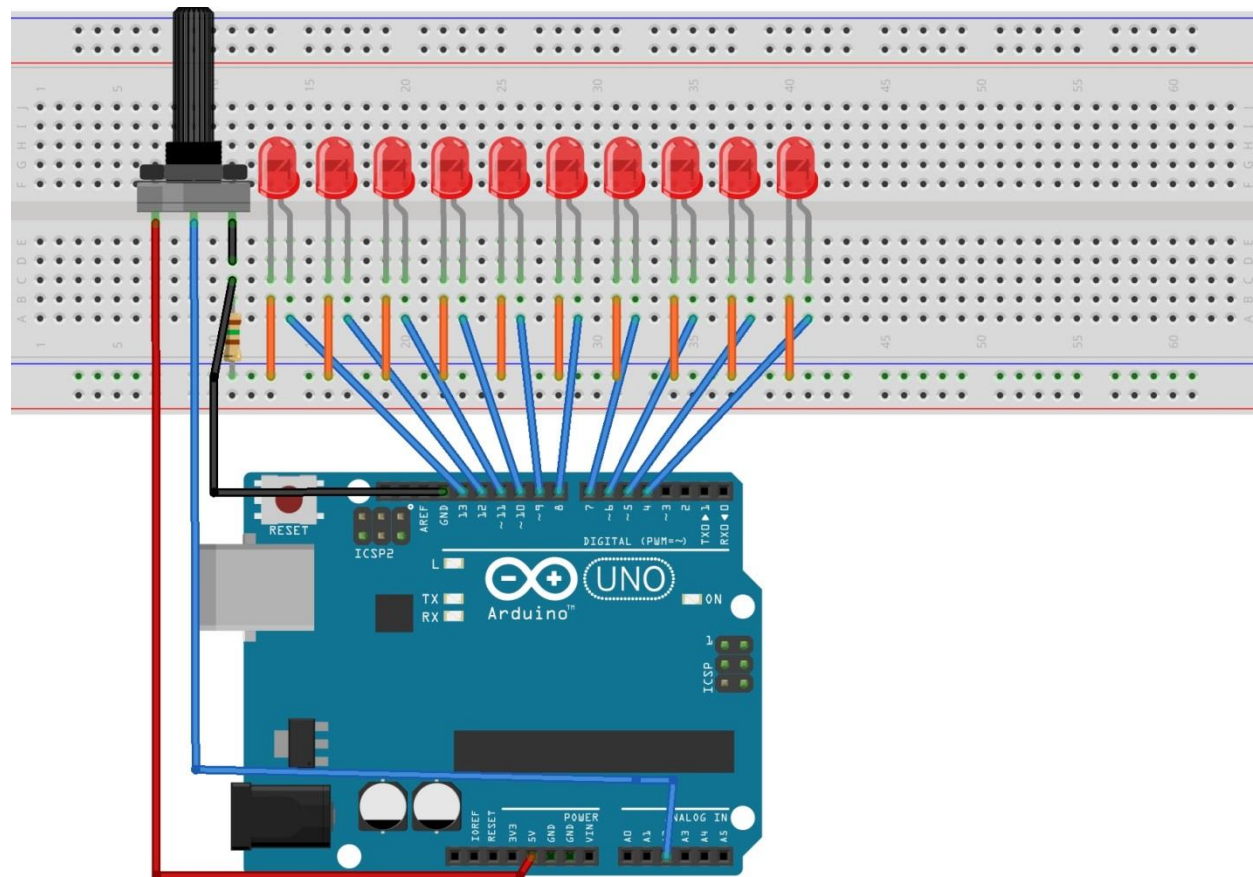
We can use the direct values read in from the potentiometer to adjust the delay between 0 and 1023 milliseconds (or just over 1 second).

ledDelay = analogRead(potPin);

This is done during the main loop and therefore it is constantly being read and adjusted. By turning the knob, we can adjust the delay value between 0 and 1023 milliseconds (or just over a second) and thus have full control over the speed of the effect.

Project 6

Hardware

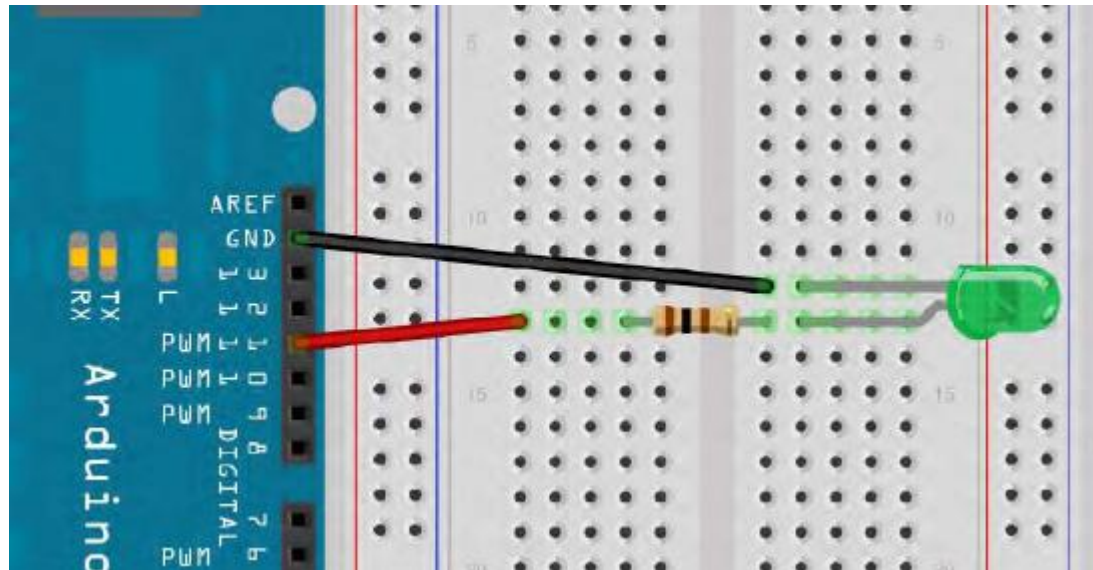


Project 6 – Interactive LED Chase Effect

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Create array for LED pins
int ledDelay; // delay between changes
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2; // select the input pin for the potentiometer
void setup() {
    for (int x=0; x<10; x++) { // set all pins to output
        pinMode(ledPin[x], OUTPUT); }
    changeTime = millis();
}
void loop() {
    ledDelay = analogRead(potPin); // read the value from the pot
    if ((millis() - changeTime) > ledDelay) { // if it has been ledDelay ms since last change
        changeLED();
        changeTime = millis();
    }
}
void changeLED() {
    for (int x=0; x<10; x++) { // turn off all LED's
        digitalWrite(ledPin[x], LOW);
    }
    digitalWrite(ledPin[currentLED], HIGH); // turn on the current LED
    currentLED += direction; // increment by the direction value
    // change direction if we reach the end
    if (currentLED == 9) {direction = -1;}
    if (currentLED == 0) {direction = 1;}
}
```

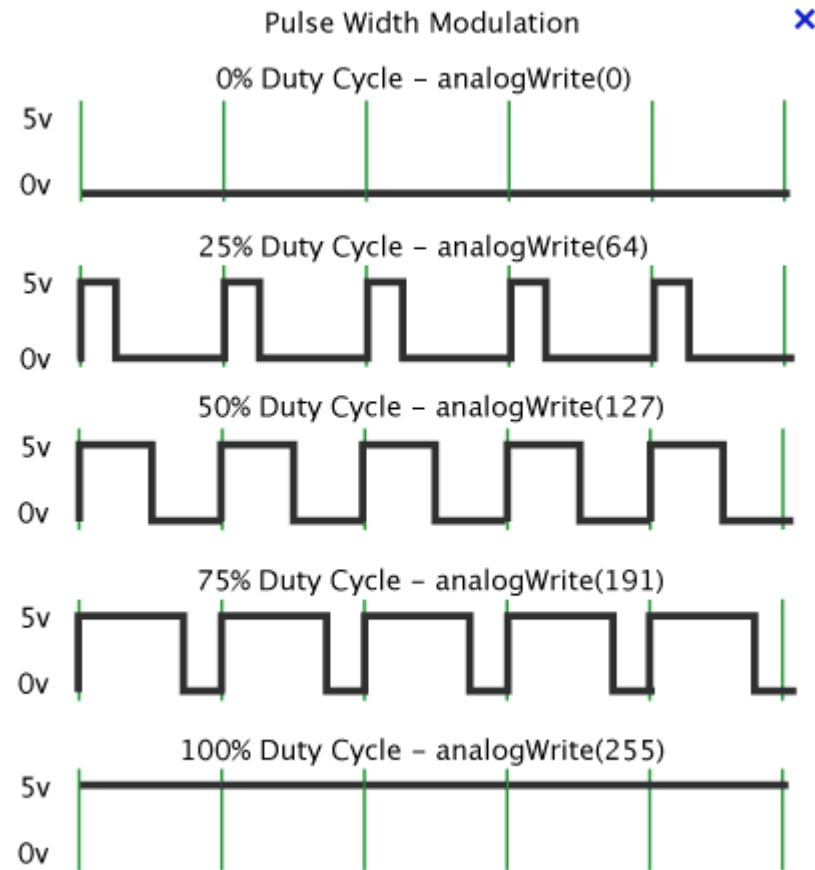
Project 7 – Pulsating Lamp

Hardware



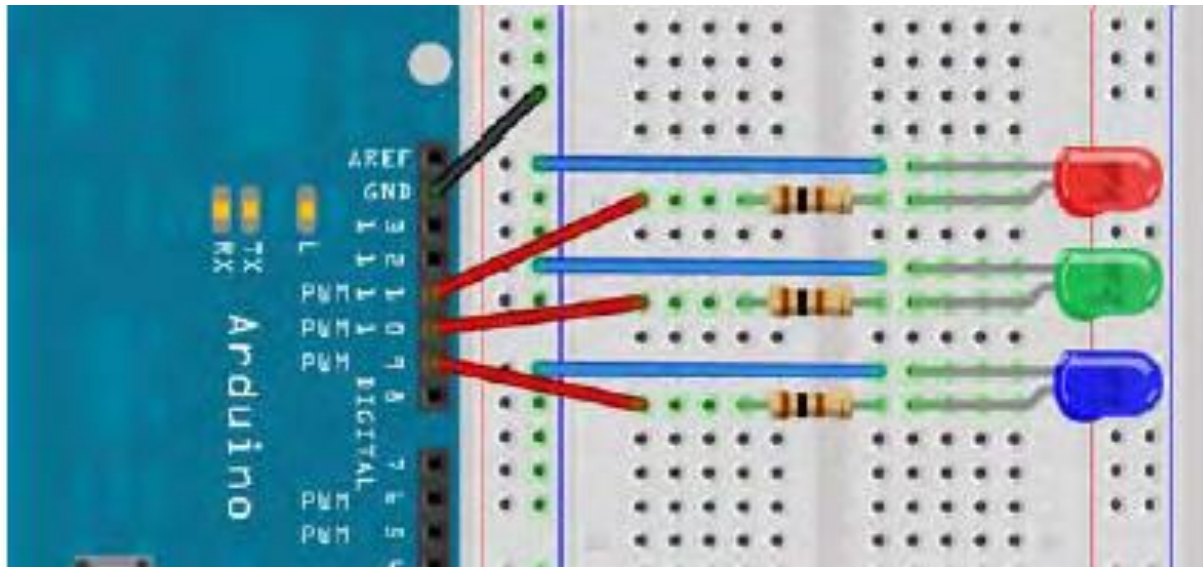

```
// Project 7 - Pulsating lamp
int ledPin = 11;
float sinVal;
int ledVal;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  for (int x=0; x<180; x++) {
    // convert degrees to radians then obtain sin value
    sinVal = (sin(x*(3.1412/180)));
    ledVal = int(sinVal*255);
    analogWrite(ledPin, ledVal);
    delay(25);
  }
}
```

Six of the Arduino pins (3, 5, 6, 9, 10 & 11) have PWM written next to them. These pins differ from the remaining digital pins in that they are able to send out a PWM signal. PWM stands for Pulse Width Modulation, which is a technique for getting analog results from digital means. On these pins, the Arduino sends out a square wave by switching the pin on and off very fast. The pattern of on/offs can simulate a varying voltage between 0 and 5v. It does this by changing the amount of time that the output remains high (on) versus off (low). The duration of the on time is known as the *pulse width*.



Project 8 – RGB Mood Lamp

Hardware



Code for Project 8

// Project 8 - Mood Lamp

float RGB1[3];

float RGB2[3];

float INC[3];

int red, green, blue;

int RedPin = 11;

int GreenPin = 10;

int BluePin = 9;

void setup()

{

randomSeed(analogRead(0));

RGB1[0] = 0;

RGB1[1] = 0;

RGB1[2] = 0;

RGB2[0] = random(256);

RGB2[1] = random(256);

RGB2[2] = random(256);

}

void loop()

{

randomSeed(analogRead(0));

for (int x=0; x<3; x++) {

INC[x] = (RGB1[x] - RGB2[x]) / 256; }

for (int x=0; x<256; x++) {

red = int(RGB1[0]);

green = int(RGB1[1]);

blue = int(RGB1[2]);

analogWrite (RedPin, red);

analogWrite (GreenPin, green);

analogWrite (BluePin, blue);

delay(100);

RGB1[0] -= INC[0];

RGB1[1] -= INC[1];

RGB1[2] -= INC[2];

} for (int x=0; x<3; x++) {

RGB2[x] = random(556)-300;

RGB2[x] = constrain(RGB2[x], 0, 255);

delay(1000);

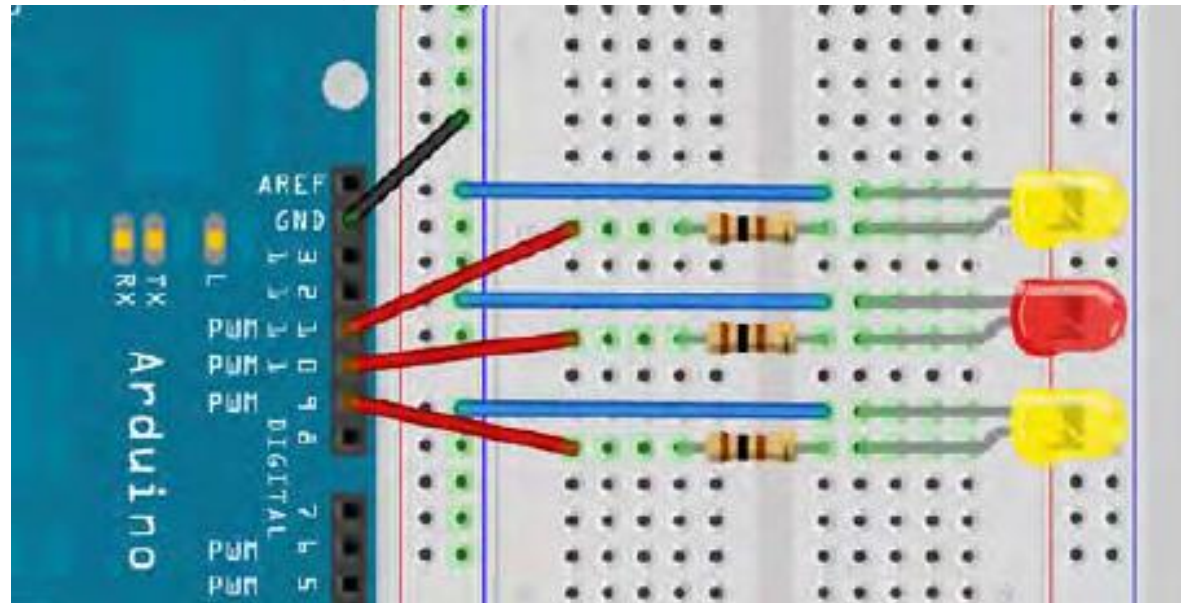
}

}

Project 9 – LED Fire Effect

Project 9 will use LEDs and a flickering random light effect, via PWM again, to mimic the effect of a flickering flame. This simple example show how LEDs can be used to create special effects for movies, stage plays, model dioramas, model railways, etc.

Hardware



```
// Project 9 - LED Fire Effect
```

```
int ledPin1 = 9;
```

```
int ledPin2 = 10;
```

```
int ledPin3 = 11;
```

```
void setup() {
```

```
    pinMode(ledPin1, OUTPUT);
```

```
    pinMode(ledPin2, OUTPUT);
```

```
    pinMode(ledPin3, OUTPUT);
```

```
}
```

```
void loop() {
```

```
    analogWrite(ledPin1, random(120)+135);
```

```
    analogWrite(ledPin2, random(120)+135);
```

```
    analogWrite(ledPin3, random(120)+135);
```

```
    delay(random(100));
```

```
}
```