MATE MARSCHALKO

INTRODUCTION TO JAVASCRIPT ELECTRONICS

Brief introduction to building circuits and program electronics with JavaScript, Node.js and Arduino

INTRODUCTION TO JAVASCRIPT ELECTRONICS

Brief introduction to building circuits and programming electronics with JavaScript, Node.js and Arduino

Mate Marschalko

Web on Devices

Introduction to JavaScript Electronics,
Brief introduction to building circuits and programming electronics with JavaScript, Node.js and Ardunio
by Mate Marschalko

Published by Web on Devices Ltd, St Albans, UK
www.webondevices.com

Technical review (JavaScript): James Miller

ISBN 9781790516902
Second edition, January 2019

About the Author

Mate Marschalko

Mate is a Senior Creative Technologist and Web Developer currently working for top creative agencies and brands in London, UK as a freelance consultant. He has been working with the web for over 10 years and developed a passion for it. He loves exploring new technologies and devices and hacking them with JavaScript. On his workbench, you will always find new toys built with Arduinos, Raspberry PIs and other development boards.

About the Reviewer

James Miller

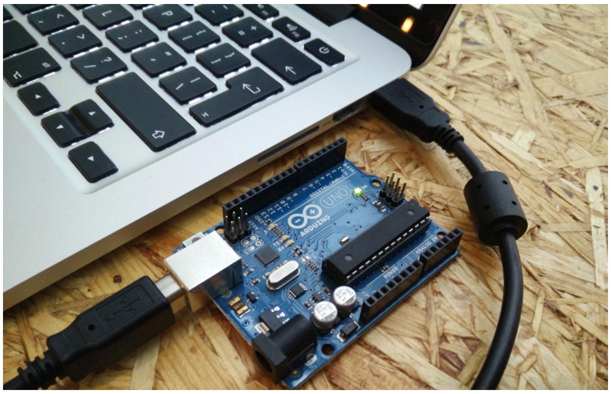
James is a freelance Technical Lead and Creative Technologist with over 10 years industry experience. He is a regular contributor to both Net Magazine and Smashing Magazine. He has a passion for Hybrid Apps, building his own hardware and his beloved football team - Luton Town.

Table of Contents

Introduction

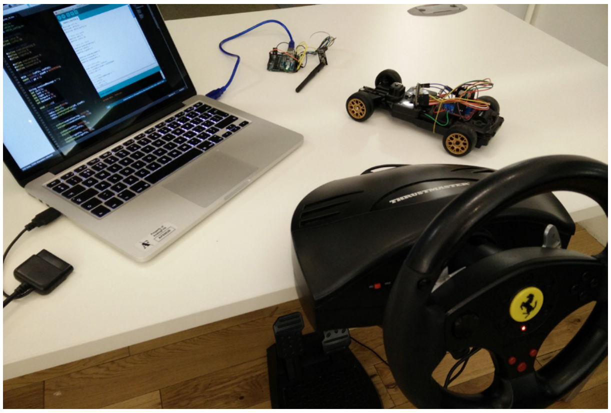
In recent years, JavaScript has evolved past merely being a tool for DOM manipulation and begun to expand its influence beyond the realms of the browser window. We now see JavaScript appearing on operating systems (Chrome OS, Firefox OS), on the server (Node.JS), and even on tiny microcontroller chips. It has also begun to become commonplace to see developers building hardware prototypes and electronic toys using JavaScript on prototyping platforms like the Arduino. These platforms and development boards made working with electronics possible for those without a background in electrical engineering.

The Internet of Things (IoT), a network of simple, connected devices, are paving new ways to market and advertise products



Arduino connected via the USB port

through physical experiences, as well as being a great financial investment for developers: they can expand their skill set towards building physical applications. Even today, forward-thinking companies have started hiring Creative Web Technologists. These developers are not only proficient in building websites but are also able to build interactive kiosks, virtual reality experiences or electronic hardware prototypes. They not only earn a lot more compared to regular developers, but they get to play with all the latest gadgets and toys in their day-to-day job!


Arduino RC car controlled with JavaScript

This book will help you take the first step into the world of Creative Technology, and start building electronic prototypes with the Arduino UNO development board and JavaScript. After the introduction, you will learn about the basics of electricity, the UNO and its components, controlling an LED light and reading light and temperature sensors. In the process, you will learn how to run JavaScript code outside the browser with Node.js, build simple circuits, and discover the Johnny-Five JavaScript library to control and communicate with the Arduino UNO and its components.

Restrictions of the browser

If we compare it to desktop and mobile applications, JavaScript running in the browser is quite limited. Access to communication ports, hardware components, and the file system are all restricted. This level of security is necessary on the Internet as not every website visited can be trusted, therefore, you need to ensure your computer can't be infiltrated by malicious code.

One way to work around the security blocks is to run your application on a server, from where you can stream all the required data to the front-end application in the browser.
The server can trust the code running on itself and will grant access to its hardware resources. There are many ways to write server applications, for example, running PHP on an Apache Linux server supported by a database (MySQL, Postgres, etc.), sometimes known as a LAMP stack. Instead of this setup JavaScript can also be used with our own computer as a server.

Communicating with the server traditionally meant sending requests through HTTP and waiting for the server to respond. Around 2005, AJAX appeared making this process a bit more dynamic. AJAX is still just polling the server to load new data, but this time without the need to reload the page. This allowed greater control as the connection and response can be both managed using JavaScript.

Using WebSockets provides a more efficient approach to retrieving data. The protocol makes it possible to open an interactive, two-way communication session between the user's browser and the server. With this API, you can send messages to a server and receive event-driven responses without having to poll for a reply.

This book will introduce you to the basics of electronics development on the Arduino platform writing JavaScript code as quickly as possible. This means we won't discuss more advanced techniques like setting up WebSocket connections or doing data logging.

My second book called *JavaScript Electronics* covers most of these advanced techniques including setting up HTTP, as well as, WebSocket servers. These servers will be used to power the Smart Talking Plant. The plant can read light, temperature and soil moisture sensors and is able to complain about its conditions by actually speaking for itself. The connection between the plant and the browser is set up to display sensor readings on real-time charts.

The next step before we can start building the circuits is to write the server application that will access the USB port and handle the Arduino; an ideal task for Node.js.

JavaScript on the server

Node.js is built on top of Google's V8 JavaScript engine which can run independently from the browser.

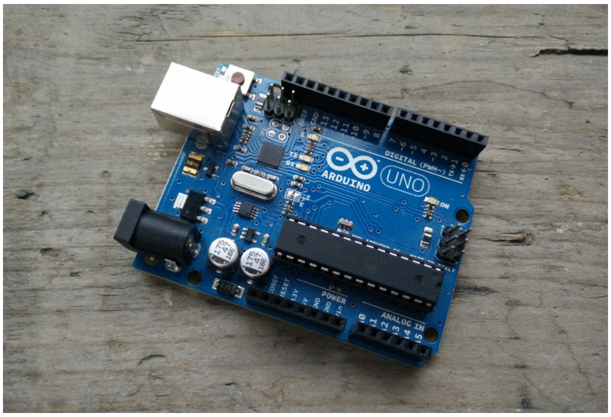
Drone controlled with JavaScript

Using JavaScript and Node.js for electronics hacking is not a new endeavour. The community is already using it to power successful projects like nodebots.io, Firmata, Cylon or Johnny-Five. The Node.js ecosystem is even stronger, with over a hundred thousand packages available via the node package manager (www.npmjs.org). In terms of speed and performance, JavaScript has also improved quite dramatically in the last decade making this aspect no longer a compromise.

The event-driven, asynchronous properties are also great for working with electronics and sensors, enabling applications that communicate through NFC, RFID or Bluetooth, all of which are currently impossible in the browser. Using Node.js opens up new possibilities for application development using devices like the Xbox Kinect, Leap Motion, midi controllers, drones and smart home accessories like the Nest thermostat or the Philips HUE light bulb.

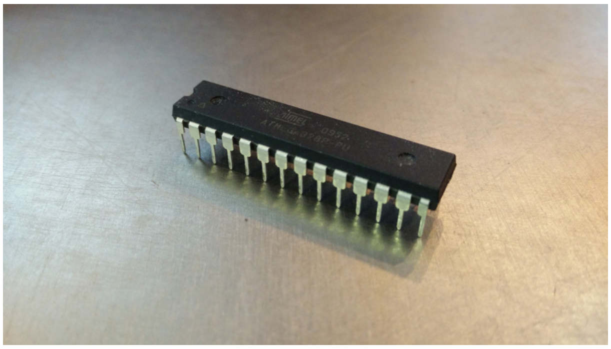
Development boards

The development board revolution started around 2005 when the first Arduino was released. The Arduino project was started by a few Italian university teachers trying to make it easy for their students to work with electronic components and program microcontroller chips. Arduino is still by far the most popular platform today.

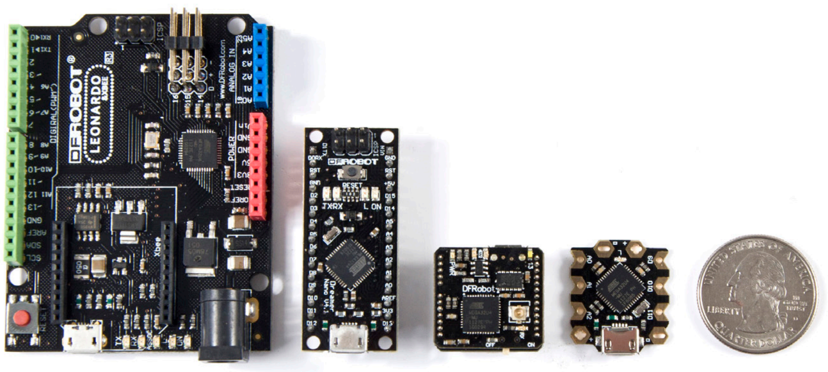


Arduino UNO development board

The microcontroller is the brain of the board that controls and manages everything. This is a low cost, low power, low-performance processor capable of running a single application at a time. The microcontroller is only responsible for managing low-level input and output electric signals and performing some basic calculations with them. This chip is capable of switching electronic components on and off, measuring electricity from sensors, and interfacing with other components.

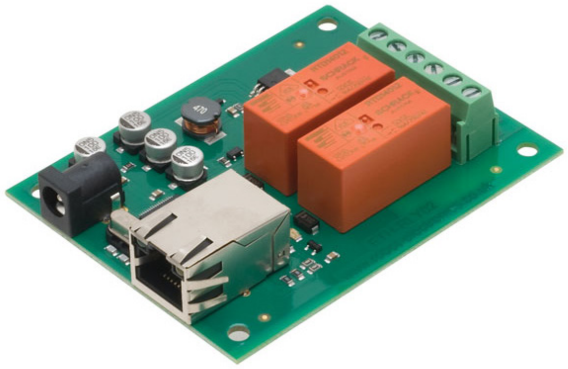

The microcontroller powering the Arduino UNO

These chips can work independently, however, development boards can streamline the building process. They help power, communicate, and interface with the microcontroller. The addition of a USB port to upload the code on the UNO and a barrel plug for powering the system from a 9V battery are both helpful. Currently, there's an overwhelming number of boards available on the market to choose from. Most of these are Arduino compatible, so how do they all differ from each other?


A variety of Arduio compatible boards from DFRobot

As a start, they greatly differ in size. Some of the smallest boards available are the DFRobot Beetle and the Tinyduino. These are great for wearable or drone projects where size and weight are important factors. With the smaller size, you have a stripped down feature set and fewer components too which means fewer things to power so they are perfect for battery-powered prototypes.

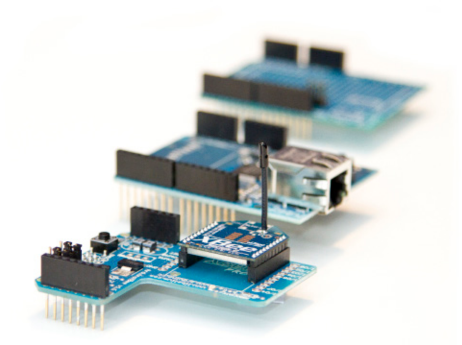
Other boards enable features like Wi-Fi, ethernet or bluetooth connection, while others can act as a mouse, keyboard or other USB controller when plugged into your computer. There are also boards designed for building home automation, and home security systems.



Development board for home automation with two relays to switch mains electricity

Building a circuit on the Arduino UNO

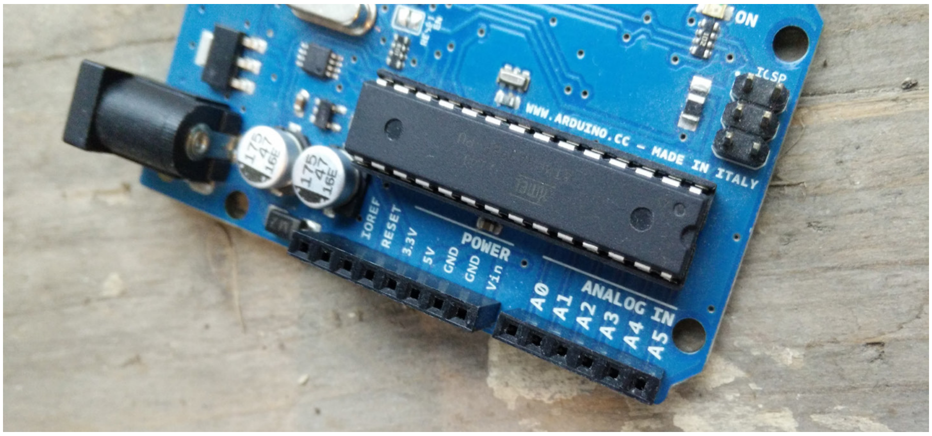
The boards previously discussed are both useful and affordable, but the Arduino UNO is still the number one, most popular board to start developing with. One of the reasons might be that its pin layout is clean and spacious, making it beginner friendly. The extra features mentioned previously, like wireless connection, are all possible with UNO projects, though these will require the addition of external modules or shields.



Extension shields for the Arduino UNO

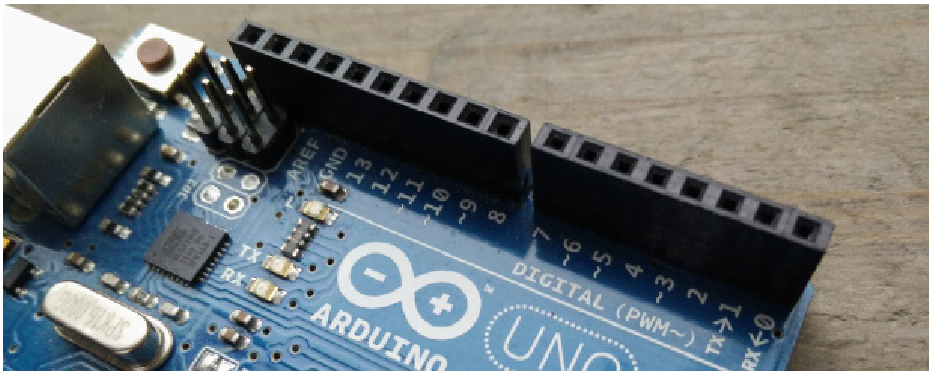
Anatomy of the Arduino UNO

The most important component of the Arduino UNO is the ATmega328 microcontroller chip. This chip manages the input and output pins (GPIO) of the UNO that are exposed through the header pins along the top and bottom edge of the board. In short, input pins (ANALOG IN) are designed to measure electricity from sensors like light, sound or temperature.



Power and input pins on the UNO
next to the long ATmega328 microcontroller chip

Digital pins (DIGITAL) on the other hand can switch electricity on and off for lights, motors, and other electronic components.



Digital pins

These pins can also be used as digital sensors which means they can detect if there is or there isn't incoming electricity. You can use these to work with binary (two-state) sensors like a button.

Both the input and output pins can be accessed and controlled from the application code you write and upload onto the board.

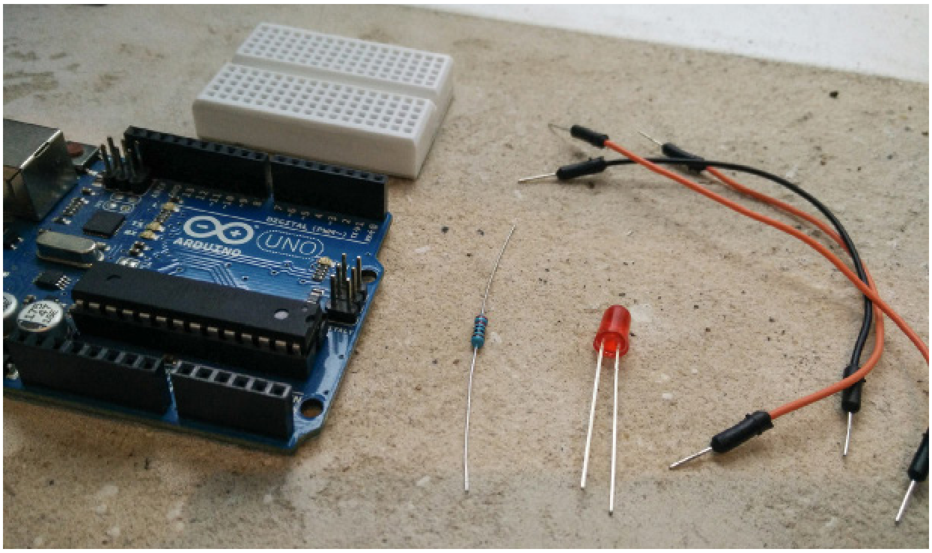
The header pins also allow connection to a constant 5V and a 3.3V (5 and 3.3 volts) power source. These will simply let you use the UNO like a battery to power your components. On regular batteries, you will find two poles: positive (+) and negative (-). The 5V and 3.3V pins on the Arduino are the positive poles, GND (ground) pins are the negative ones.

Next to the header pins is the large USB type-B port used to communicate with a PC and to upload code. The barrel plug on the bottom left is where you can connect a 9V battery or any other DC power source between 6 - 20V (the recommended is 7 -12V), this can be any mains adapter with the correct size of plug and amount of voltage. This time the Arduino won't be powered from an external power source, instead, it will be connected via USB to a computer which will, in turn, power the device.

The digital pins on the Arduino board are also power sources, just like the 5V pins. The main difference is that these digital output pins can be switched on and off by the microcontroller chip and the application we upload and run on it.

Building the first circuit

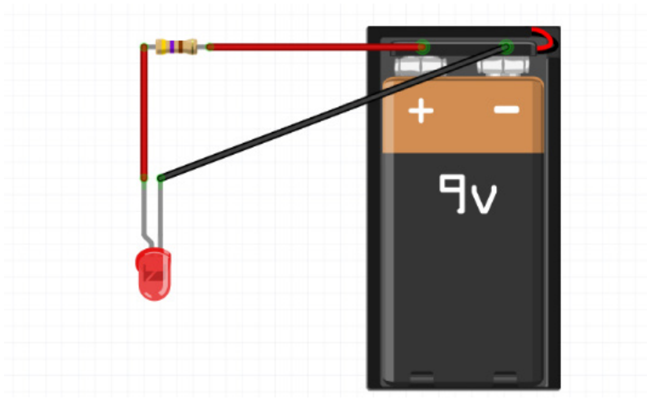
To demonstrate the behaviour of the output pins let's build a simple circuit and control it from JavaScript. The "Hello World" of hardware development is blinking an LED, so that's what we are going to start with. For this project, you will need an Arduino UNO, a regular LED, a resistor between 150 - 1kΩ, a breadboard, and a few jumper wires.



Components for the LED blink example project

Resistors can be identified from the coloured stripes on their body. There are many resistor calculators (http://www.dannyg.com/examples/res2/resistor.htm) and cheat sheets to help you with that.

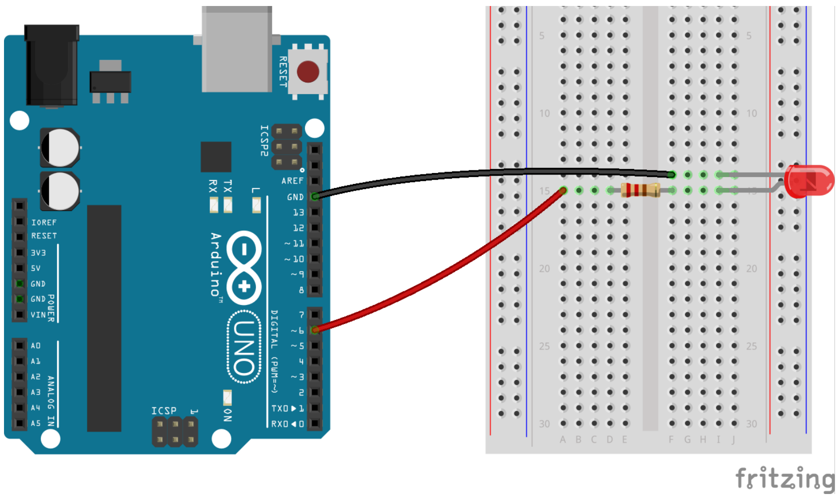
The simplest possible circuit we could think of would be an LED lit by a battery. This circuit connects the positive pole of the battery with the positive leg of the LED (long leg) and the



Battery and an LED (fritzing.org)

negative pole with the negative leg (short leg). Once they are connected, electricity and electrons start flowing through the LED which will inevitably cause it to light up.

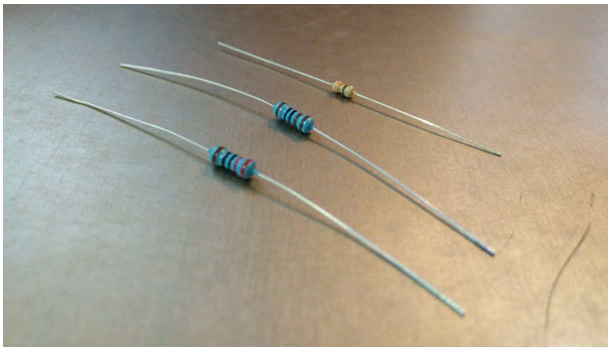
Rebuilding this simple circuit on the Arduino will essentially mean replacing the positive pole of the battery with the 5V pin or one of the output pins on the Arduino (DIGITAL) and the negative pole with GND.



LED circuit connected to a digital output pin

However, in our upcoming LED blink example we will use a digital output pin as these are the power source pins that we will be able to access from our application code and switch on and off. In this instance, I randomly picked output pin number 6. You can see the circuit diagram of this setup on the previous page.

You probably noticed the addition of a small resistor to the circuit and wonder why we need this. Well, there's one problem: LED lights don't limit the electric current in the circuit so without a resistor, they would let too much current across. With no resistor in our circuit, we would overdrive and damage our LED in a short amount of time. It would probably only work for a couple of minutes before burning out irreversibly.



Fixed rate resistors

To prevent this from happening, we can limit the flow of electric current through the LED by adding a resistor. Using Ohm's law (Resistance equals voltage divided by current or R = V / I) we can calculate the exact resistor needed in a certain setup.

In our circuit, the Arduino provides 5 volts but not all of that will fall to the resistor: the LED also has a 2-volt voltage drop which is the ideal supplied voltage for it to work. We also know that the ideal current for our LED is 20mA or 0.02 in Amperes (the datasheet of the LED will give you all this information). Now we need to choose a resistor which will drop that voltage at 20mA. The right value is given by Ohm's law:

$$R = V / I$$

$$R = (5V - 2V) / 0.02A$$

$$R = 150\Omega$$

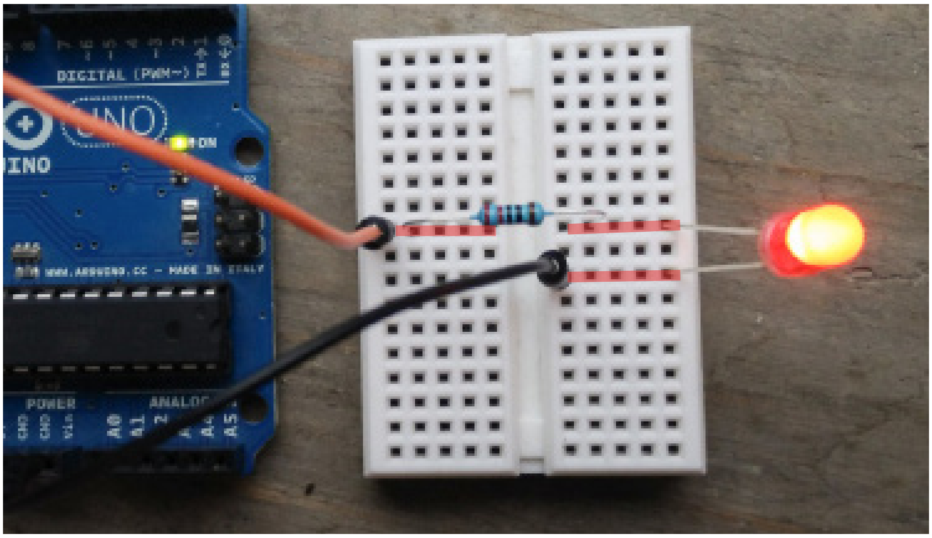
This means that we need a resistor around 150Ω. You don't want a resistor that's less than 150Ω but using a larger one, for example, 220Ω, 500Ω or 1000Ω, would simply resist the flow more which would make our LED dimmer but would extend its lifetime in return.

This circuit is now complete and by switching the 6th output pin on from our program code the LED would light up.

Prototyping breadboards

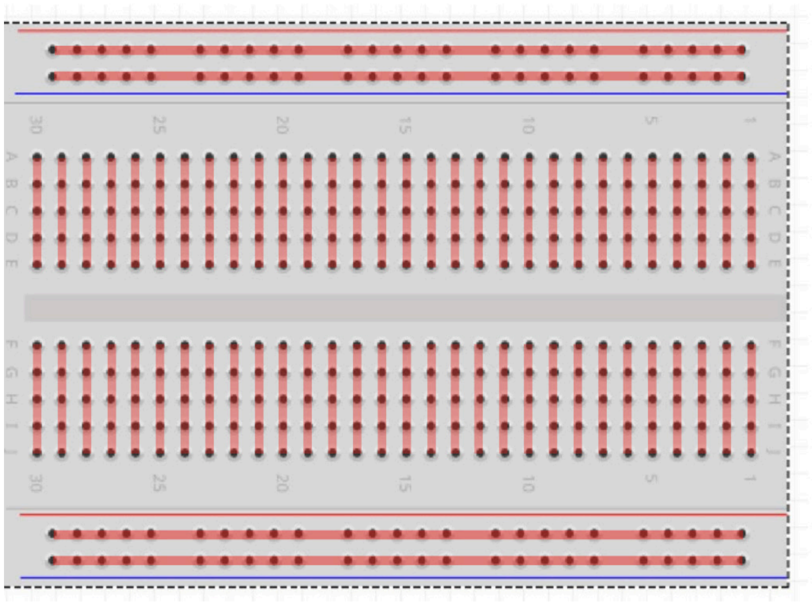
Breadboards make it easy and quick to build circuit prototypes. They are designed to connect regular male to male jumper wires or the legs of simple components like LEDs, resistors, diodes or capacitors.

Breadboards essentially help to connect these components together. When you stick the leg of an LED anywhere in the breadboard, components added to the same row of pins will be connected to the LED.



LED circuit connections highlighted inside the breadboard

Here is a larger breadboard with all its pin connections highlighted:



Getting started with Node.js

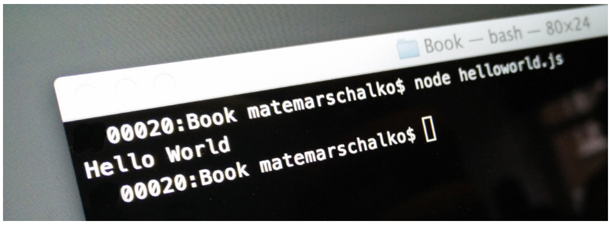
The LED is now ready to be switched on and off using JavaScript. To do this we will need to run JavaScript on the server computer in order to have access to the USB port that we will connect the Arduino to. The first step in this journey is to install the Node.js environment onto our computer.

Installing Node.js is a quick and easy process, and can be done with the official pre-built installers. You can download these from https://nodejs.org/download/.

Once installed, the new *`node`* command is available from the Terminal on MacOS or the Command Prompt on Windows. For a Hello World demonstration create a new file called helloworld.js and add this single line of JavaScript to it:

```
    console.log("Hello World");
```

Entering *`node helloworld.js`* to your command line tool (from the same folder that contains the helloworld.js file) will run the JavaScript file and output "Hello World" to the command line.



```
Book — bash — 80x24
00020:Book matemarschalko$ node helloworld.js
Hello World
00020:Book matemarschalko$ 
```

Node.js "Hello World" in terminal

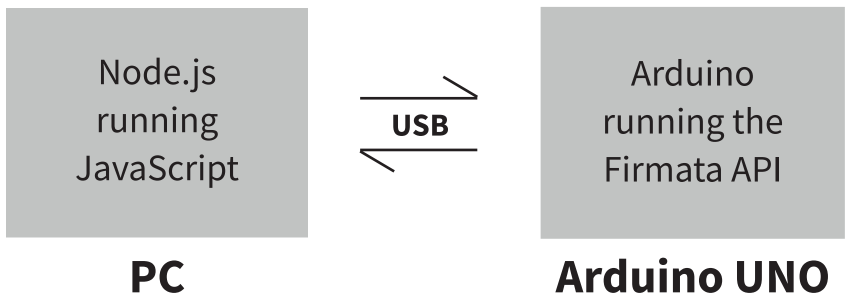
This might not seem much progress so far, but with Node.js set up we are ready to start building our hardware projects using JavaScript. From this environment, we now have access to the USB port, sensors and many other components of our computer.

Connecting Node.js to the Arduino

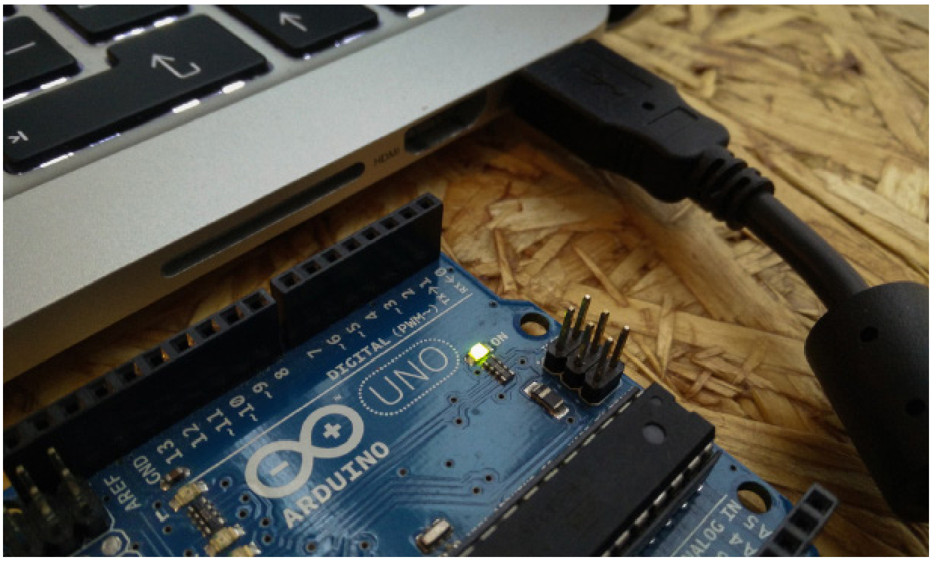
For this app, we will be using the Johnny-Five JavaScript library (http://johnny-five.io) designed for Node.js. This library was created to make hardware prototyping easy for web developers.

Running JavaScript code straight on development boards is only possible with a handful of models and, unfortunately, the Arduino is not one of them. Arduinos require you to write code in the Arduino language, and this can't be changed currently.

Johnny-Five, and many other Node libraries work around this limitation by requiring a fixed codebase installed onto the Arduino. This codebase essentially exposes an API to communicate with and control the Arduino from external devices. This codebase is called Firmata.



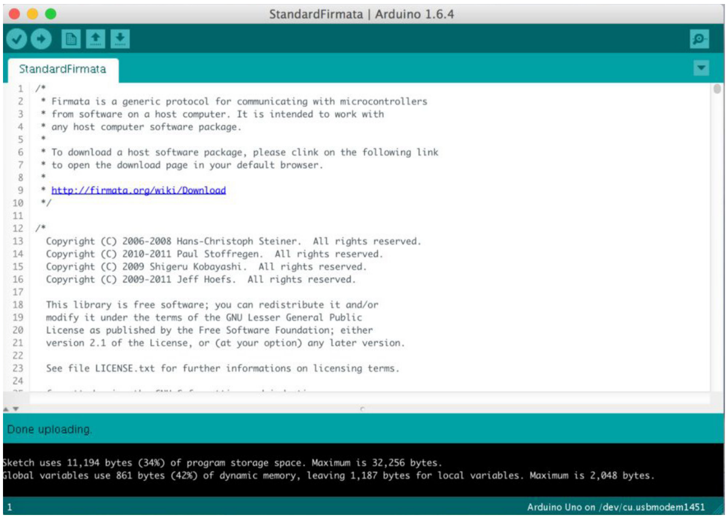
To make the LED blink, we first need to upload the Firmata codebase onto the Arduino UNO. To upload *any* code onto the board we need to download and install the official Arduino IDE from the Arduino website (https://www.arduino.cc/en/Main/Software). The Arduino IDE is just the name of the application you use to write and upload Arduino code to the board.



The Arduino UNO is powered up from the USB port

Once the IDE is installed, plug in your Arduino into the USB port and the green power LED, labeled ON, should light up. Open up the IDE application you just installed and make sure under Tools/Board Arduino/Genuino UNO is selected. You will also need to select the USB port the Arduino is connected to under Tools/Serial Port. The port will only show up in the list if the board is actually connected. If you have problems getting the environment ready please refer to the official installation guide (https://www.arduino.cc/en/Guide/HomePage) or the troubleshooting page (https://www.arduino.cc/en/Guide/Troubleshooting).

This is how the Arduino IDE looks like:



StandardFirmata | Arduino 1.6.4
StandardFirmata
Done uploading.
1
Arduino Uno on /dev/cu.usbmodem1451

```
/*
 * Firmata is a generic protocol for communicating with microcontrollers
 * from software on a host computer. It is intended to work with
 * any host computer software package.
 *
 * To download a host software package, please clink on the following link
 * to open the download page in your default browser.
 *
 * http://firmata.org/wiki/Download
 */

/*
  Copyright (C) 2006-2008 Hans-Christoph Steiner.  All rights reserved.
  Copyright (C) 2010-2011 Paul Stoffregen.  All rights reserved.
  Copyright (C) 2009 Shigeru Kobayashi.  All rights reserved.
  Copyright (C) 2009-2011 Jeff Hoefs.  All rights reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  See file LICENSE.txt for further informations on licensing terms.

```

```
Sketch uses 11,194 bytes (34%) of program storage space. Maximum is 32,256 bytes.
Global variables use 861 bytes (42%) of dynamic memory, leaving 1,187 bytes for local variables. Maximum is 2,048 bytes.
```

Arduino IDE with the Firmata sketch loaded

At this stage, we won't be writing anything in the Arduino language. We only opened the IDE to upload the standard Firmata library. Select File/Examples/Firmata/StandardFirmata to open the sketch then hit upload (the green arrow pointing to the right), this will upload the Firmata library to your UNO.

If everything was successful you will see the "Done uploading". message in the green status bar. If you get an error message, make sure the Arduino is connected, has power, and that the correct board and port are selected in the Tool menu.

Blinking an LED

The wiring is now complet and the Arduino has the Firmata library loaded. The UNO is ready to take commands from Node.js. Since we are going to be creating an app for Node.js, it's a good practice to initialise an application in a new folder.

To do that, create a new folder and run `npm init` from the command line. This will run you through a series of questions about your app but you can just hit enter a couple of times to work with the defaults.

As we will be using the Johnny-Five library to blink our LED, let's now install it:

```
npm install johnny-five --save
```

Notice how a new node_modules folder has been created in your app folder containing your installed libraries. Let's also create a new file for our Node.js application and name it blink.js. The first thing we need to do in this file is load all the libraries into variables required by our application. In our case it's only Johnny-Five:

```
    const five = require("johnny-five");
```

Next, we initialise a new `Board` instance that will provide us with all the tools to interact with the Arduino:

```
    const arduino = new five.Board();
```

When working with jQuery we use the `$(document).on("ready", callback)` pattern to wait for the document to finish loading before we do anything. The Arduino and the USB connection also need some time to start up, so we will need to implement a similar event listener before we can send our commands:

```
    arduino.on("ready", function(){
        // The board is ready
        // We can now blink the LED

    });
```

To blink the LED, let's initialise a new LED instance inside the callback of the event listener:

```
    // Create an LED on pin number 6

    const led = new five.Led(6);
```

Then below this, we can simply call `blink()` on the instance:

```
    // Blink the LED every half second
    led.blink(500);
```

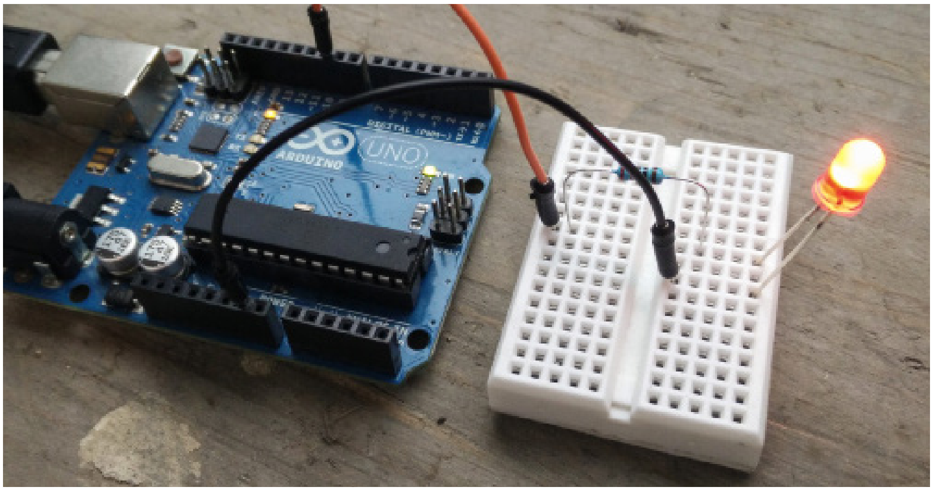
And that's all ther is to it! Here's the complete blink sketch:

```
    const five = require("johnny-five");

    const arduino = new five.Board();
```

```
arduino.on("ready", function(){
    const led = new five.Led(10);
    led.blink(500);
});
```

Now that the LED is set up we can try other methods too: `led.pulse()` will fade the LED in and out instead of just switching with no transition. Conveniently, fade in and out methods are also available:

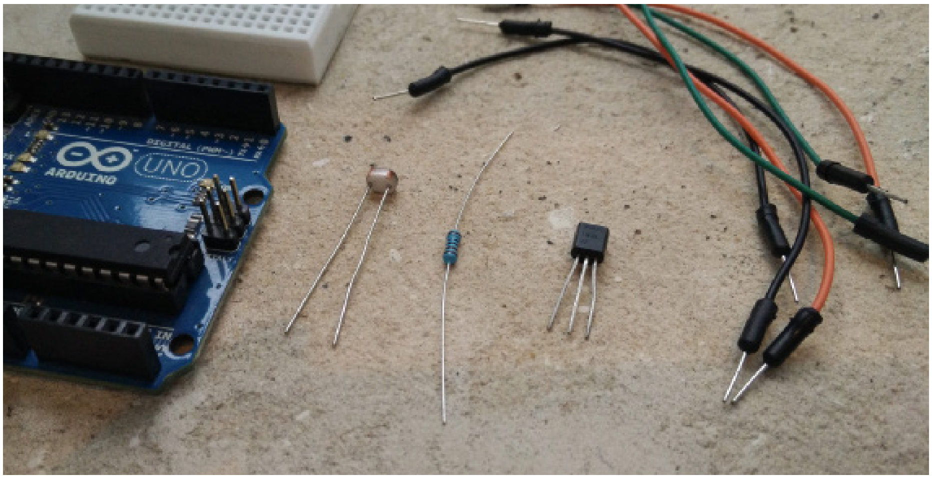
```
// fade LED in
led.fadeIn();

// wait 3 seconds then fade out
this.wait(3000, () => {
    led.fadeOut();
});
```



Blinking the LED

Sensing the world

In this chapter, we will extend our knowledge by exploring the input pins on the Arduino UNO. We already know that these pins can measure the change in voltage so we will use simple analog sensors that affect the voltage running across them depending on certain changes in the environment.



Components for the sensor projects

For this project, you will need an Arduino UNO, an LM35 or TMP36 temperature sensor, an LDR (light dependent resistor), a breadboard, two 1kΩ resistors, and a few male-to-male jumper wires.

Measuring temperature

For measuring temperature I picked the very common LM35 sensor. This low-cost sensor is rated to operate between -55 and 150°C, with a +/- 0.5°C accuracy (although it's a little bit tricky to measure negative temperatures with the LM35). The way this analog sensor works is really simple and in fact, most analog sensors work in a similar way. First, they are powered from a constant power source on two of their pins (+ and -), then, on a third pin, they output a lower voltage value that is proportional to the sensor reading.

Our temperature sensor will be powered from the Arduino UNO's 5V pin and will output voltage values between 0 and 2 volts, changing with the temperature. The LM35's scale factor is 0.01V/°C which means that a 1 Celsius degree change in the air temperature will result in a 0.01-volt change measured on the output pin.

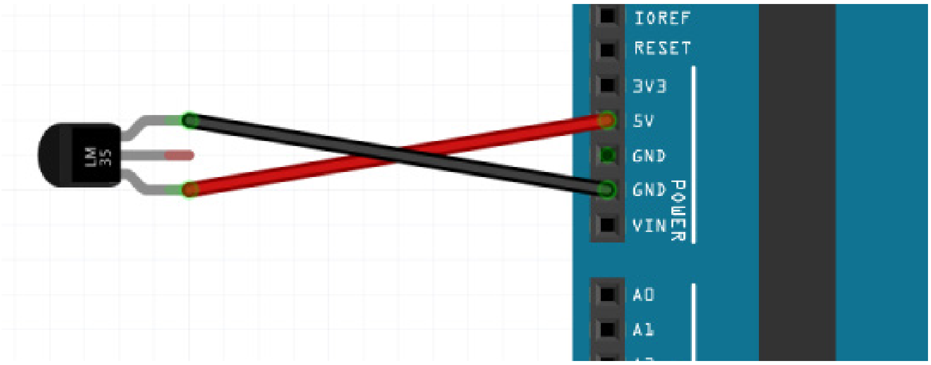
Conveniently, the Arduino's analog input pins are designed to measure and convert voltage into data we can use. In actual numbers, the full range of 0 to 5 volts that can be measured by the analog pins of the Arduino UNO will be mapped to a 0 - 1024 scale in our program code.

Let's say your sensor returns only 1.5 volts from the supplied 5 volts to the analog input pin you connected it to. The reading in your application will be 307 on the 0 - 1024 scale because (1.5 / 5) * 1024 is 307.2.

You have the option to retrieve this raw number in your application, but the Johnny-Five library maps the 0–1024 measurement scale to the sensor's -55 to 150°C range and calculates the resulting temperature from this information for us.

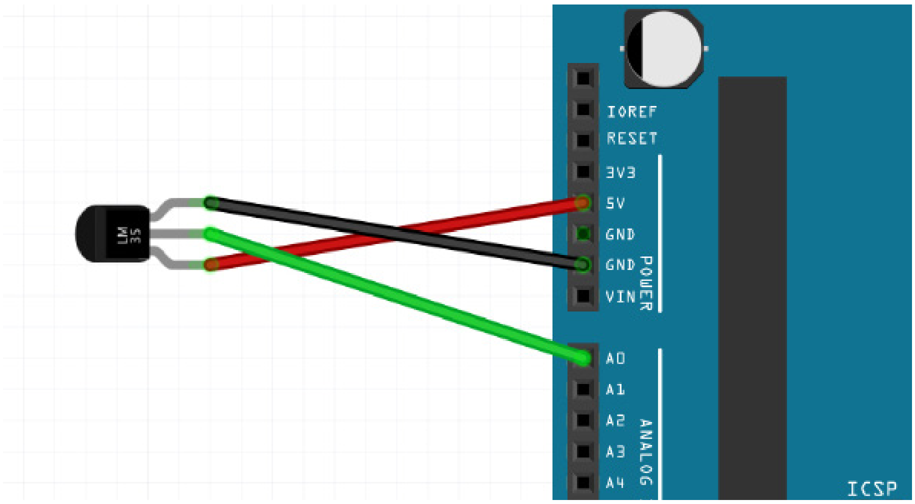
Let's go back to our temperature sensor and wire it up. Before you start wiring, always make sure you have unplugged your Arduino from the USB or any other power source. Otherwise, you could accidentally stick a wire into the wrong header pin and damage your board or the sensor.

From the three legs of the sensor, the one on the left is the positive, and the one on the right is the negative pole (the flat side of the sensor is the front). We will feed the power in from the 5V (positive) and GND (negative) pins of the Arduino UNO as you can see on this diagram:



Temperature sensor powered up (fritzing.org)

The make the connections you might want to use a breadboard at this point! The sensor is now powered up and the only pin left to be connected is the output pin in the middle. This will need to be connected to one of the five analog pins on the UNO. These are the ones lined up along the bottom right edge of the board, marked A0 to A5. Let's connect it to pin A0:



Temperature sensor connected to an analog pin (fritzing.org)

Now that the wiring is finished, let's write the JavaScript code that will read the sensor. Firstly, create a new project (*`npm init`*) and name the main file temp.js, starting it off the same way as the LED blink sketch:

```
const five = require("johnny-five");

const arduino = new five.Board();

arduino.on("ready", function(){
    // The Arduino is ready
});
```

Next, we need to create a new `Thermometer` sensor instance with a few settings. They include the name of the sensor, and the pin number it's connected to.

```
const tempSensor = new five.Thermometer({
    controller: "LM35",
    pin: "A0"
});
```

If you have a TMP36 sensor that can still be used the same way. Simply change the controller setting to TMP36 and the wiring remains the same.

After initialising the temperature sensor we can start using the "on data" event listener to catch sensor readings as they arrive via the USB port. The Johnny-Five library leverages the asynchronous capabilities of Node.js which means that sensor readings will immediately appear in the "on data" event listener's callback function. Let's write this event listener:

```
tempSensor.on("data", function(){
    console.log(this.celsius + "°C");
    console.log(this.fahrenheit + "°F");
});
```

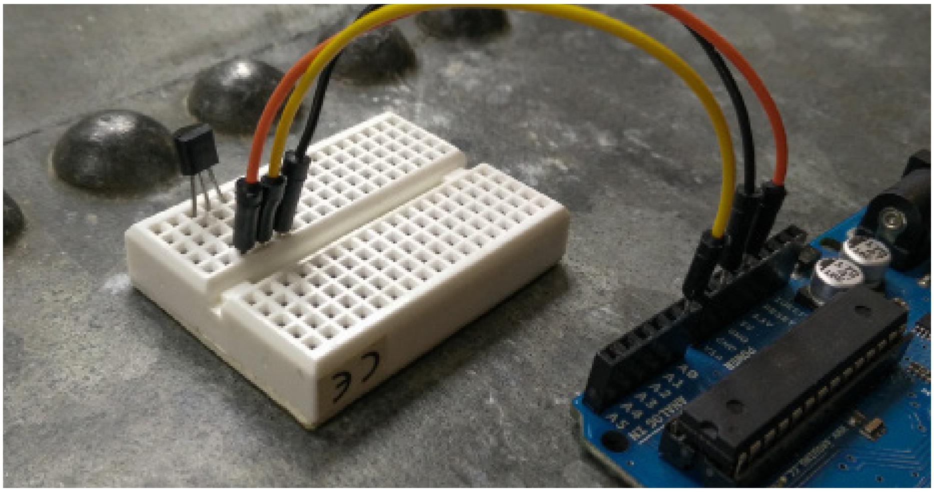
Again, we need to wait for the board to be initialised so both of these blocks will need to go inside the `arduino.on("ready", callback)` code block.

In the final version of the code, both the Celsius and the Fahrenheit sensor readings are rounded up to one decimal places to make the numbers easier to read.

```
const five = require("johnny-five");
const arduino = new five.Board();
```

```
let celsius = 0;
let fahrenheit = 0;

arduino.on("ready", function(){

    const tempSensor = new five.Thermometer({
        controller: "LM35",
        pin: "A0"
    });

    tempSensor.on("data", function(){
        celsius = this.celsius.toFixed(1);
        fahrenheit = this.fahrenheit.toFixed(1);

        console.log(celsius + "°C");
        console.log(fahrenheit + "°F");
    });
});
```

Here's the LM35 temperature circuit added to a small breadboard:

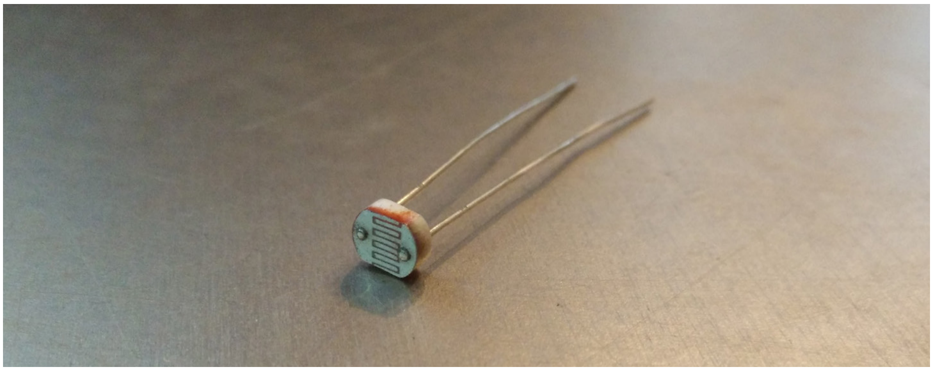


Temperature sensor circuit on a breadboard

Run your code by typing `node temp.js` into your command line. Check the readings, and if there are anomalies, double check your wiring, make sure the Arduino is connected to the USB port, and the green power light is on. If all is good, we can move on to the light sensor!

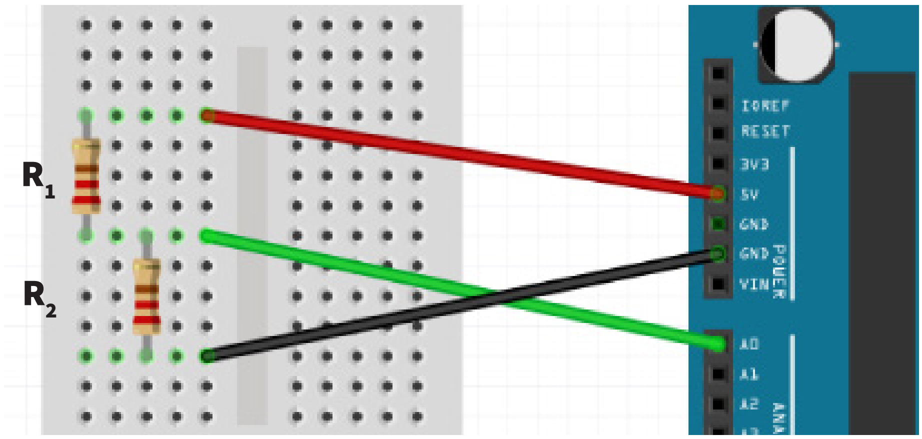
Measuring light

Our light sensor is one of the simplest sensors available, known as an LDR, which stands for Light Dependent Resistor. An LDR acts like a regular resistor when added to a circuit, however, the main difference is that the LDR changes its resistance depending on the light conditions.



A Light Dependent Resistor

Unfortunately, resistance isn't easy for the analog input fields of the Arduino to measure. To convert this resistance change to voltage change, that the input fields are more comfortable measuring, we need to build a simple voltage divider circuit. A voltage divider circuit splits a larger voltage into a smaller one as per the ratio of the two resistors included in the circuit:



Voltage divider circuit (fritzing.org)

The 5-volt input of this circuit comes from the Arduino through the red wire. The output voltage through the green wire is directly proportional to the input voltage and the ratio of the resistors (R_1 , R_2).

$$V_{out} = V_{in} * (R_2 / (R_1 + R_2))$$

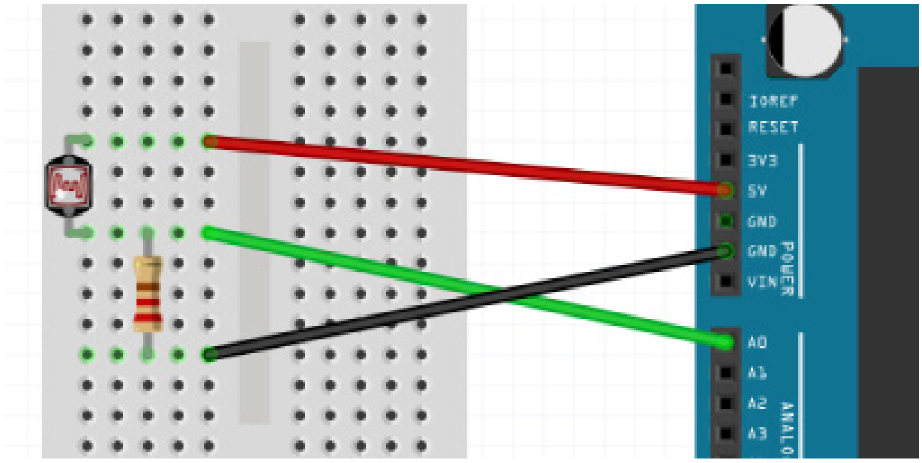
In this circuit we use two 1000Ω resistors so here's how the equation looks like after substituting these values in:

$$V_{out} = 5V * (1000 / (1000 + 1000))$$
$$V_{out} = 5V * 0.5$$
$$V_{out} = 2.5V$$

This means that if two of the same resistors are used in a voltage divider circuit the output voltage will be half of the input voltage. 2.5 volts in our case.

Using the equation we also see that changing only one of the resistors will change the output voltage up or down. This here is the key learning for our light sensor circuit!

Let's now change one of the regular resistors to the light dependent resistor. The resistance of the LDR will change with the light conditions in the voltage divider circuit which will constantly change the output voltage for our input pin in return.

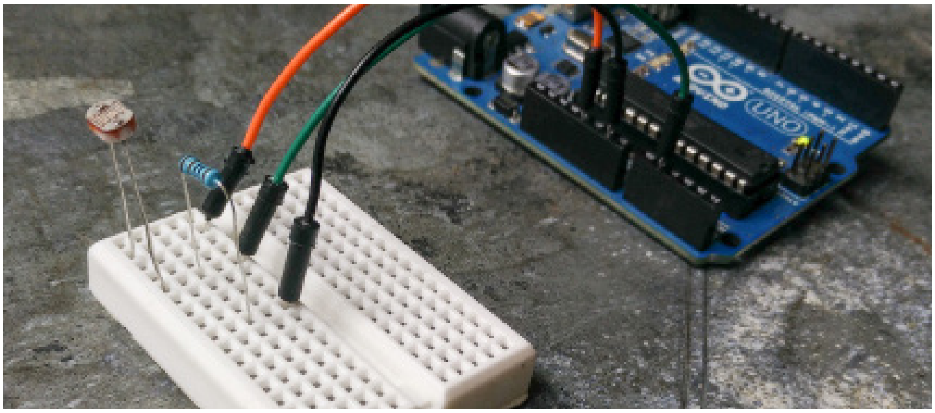


Light sensor circuit (fritzing.org)

When the photoresistor is exposed to light its resistance decreases so the voltage reading will be higher. Conversely, with less light, the voltage reading will be lower. The changing voltage value is then what the analog input pins of the Arduino factor in to calculate the light measurements.

The light dependent resistor changes its resistance between 150Ω in bright light and 9000Ω in dark. Keeping the original fixed 1000Ω resistor in and using the voltage divider equation we can tell that our output voltage can change between 4.35V and 0.5V which is a good range for the Arduino's input pin.

Here's how this circuit looks like on a small breadboard:



Light sensor circuit on breadboard

The circuit is now complete so let's add the light sensor to our JavaScript app into the Arduino's "on ready" callback function. We initialise a new `lightSensor` then add an "on change" event listener:

```
const lightSensor = new five.Sensor({
    pin: "A0",
    freq: 250
});

lightSensor.on("change", function(){
    console.log(this.value);
});
```

This piece of code is very similar to the way the temperature sensor is handled by the Johnny-Five library. First, a new sensor needs to be initialised with a few settings, then the "on change" event listener is used to wait for readings to arrive.

Here's the final version of the light sensor code:

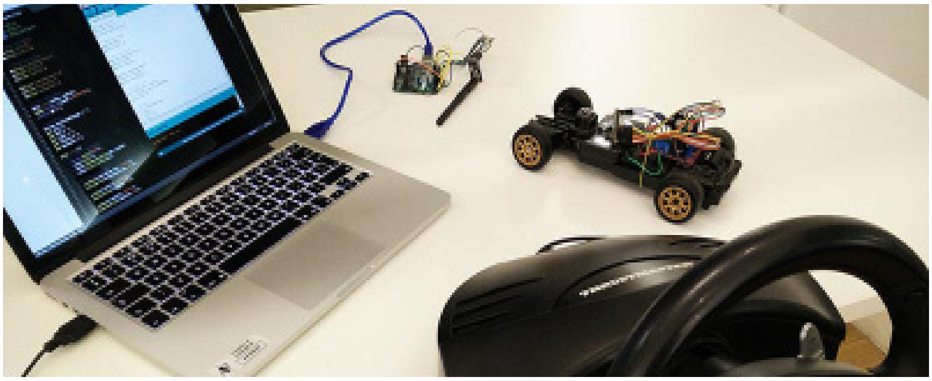
```
const five = require("johnny-five");

const arduino = new five.Board();

let light = 0;

arduino.on("ready", function(){

    const lightSensor = new five.Sensor({
        pin: "A0",
        freq: 250
    });

    lightSensor.on("change", function(){
        light = this.value;
        console.log(light);
    });
});
```

Saving this into a new light.js file and entering *`node light.js`* into the command line will result in light sensor readings appearing every quarter of a second. The `freq` (frequency) property is where we can change this behaviour by inputting a number in milliseconds.

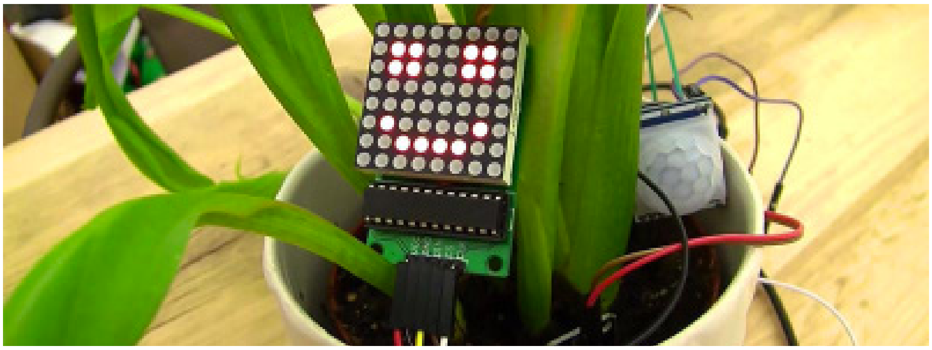
Going forward

In this book, you learned the basics of working with the Arduino UNO using Node.js. On Web on Devices (www.webondevices.com) you will find other interesting projects built using similar methods.

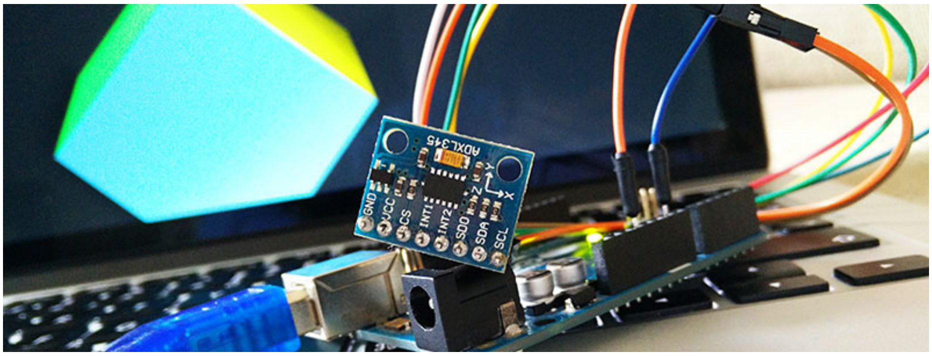
Check out the radio controlled car that was rebuilt from scratch using Arduino boards. It connects to a computer wirelessly and with the JavaScript Gamepad API allows the user to drive the car with a USB steering wheel.



Or there's George, the talking plant. Through his sensors, he can detect temperature, light, motion and soil moisture changes. He complains when he is not happy with any of the sensor readings, and can also answer basic questions. He talks and listens using the WebSpeech API. This smart talking plant project is the final example in my second book called JavaScript Electronics where everything is discussed from the very beginning.



There's also an Arduino gyroscope project that lets you rotate 3D CSS objects on the screen using a physical controller.



Using Node.js to communicate with the Arduino is just one of many ways to use JavaScript for building electronic projects. In these examples, the JavaScript code was running on your computer processor, and we've been sending commands to the Arduino.

Other boards like the Raspberry PI, Arduino Yún, Tessel, and the Espruino can actually execute JavaScript without a computer. The Arduino compatible Particle boards expose a RESTful API, and there's a Node.js library to work with them too. Particle boards connect to the Internet wirelessly, so the Node library doesn't have to rely on a USB connection.

The Web on Devices project is dedicated to keep pushing the limits of what's possible with development boards and smart devices leveraging mainly web technologies. In the upcoming projects, we will explore all the mentioned boards and techniques!

Follow Web on Devices on Facebook, Twitter or Instagram so you don't miss any of the cool upcoming projects:

www.facebook.com/webondevices
www.twitter.com/web_on_devices
www.instagram.com/web_on_devices

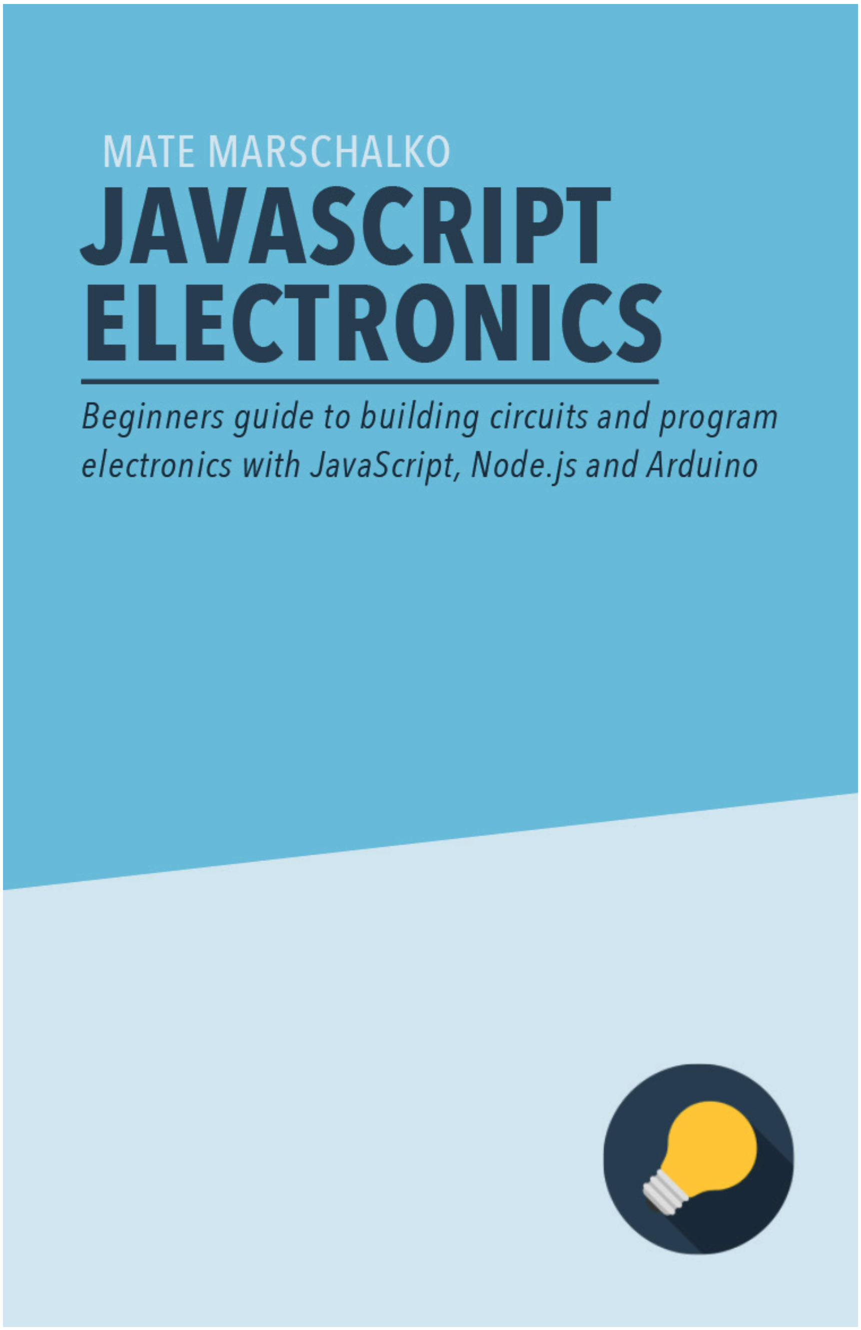
JavaScript Electronics

Now that you have learned the basics of working with the Ardunio, its input and output pins and Node.js you are ready to start working on more complex projects. My second book called JavaScript Electronics discusses all areas covered in this introductory book in a lot more detail and contains many exciting projects to put all that knowledge into use!

There will be many example projects to help you understand some of the key principles in hardware development. These projects include: controlling an LED light, reading electronic sensors, monitoring the soil moisture levels of a desk plant and alerting the owner with tweet and SMS messages and spoken words with speech synthesis, data logging and data displaying on charts, controlling the volume on a computer with a physical potentiometer, sensing motion with an infrared sensor and sounding an alarm.

In the final chapters of the book, we will look at a few other development boards with different sizes and set of features and compare them to the Arduino UNO.

Get the book now and get started building your very own projects with JavaScript and the Ardunio UNO!


MATE MARSCHALKO

JAVASCRIPT ELECTRONICS

Beginners guide to building circuits and program electronics with JavaScript, Node.js and Arduino

Introduction to JavaScript Electronics,
Brief introduction to building circuits and programming electronics with JavaScript, Node.js and Arduino by Mate Marschalko

Published by Web on Devices Ltd, St Albans, UK
www.webondevices.com

ISBN 9781790516902