Every scientist or engineer begins life as a hacker. In order to discover something new, one must often BUILD something new. Fortunately for the “non-scientists” among us, that paradigm puts us on even ground!

For instance, temperature was once only a relative term: “Eh… it’s hotter than yesterday, isn’t it?” Finally someone with a workshop, some raw material, a bit of time on his hands, and a great bit of creativity invented the thermometer. Suddenly humanity had the ability to quantify “hot” and “cold” in a universal manner that could be understood across continents. Even more fantastic was the ability to record and compare these facts, year after year. Eventually, with a large enough data set, humanity was able to make reasonably approximate predictions.

All this from one man’s ingenuity: simple spheres of glass filled with various mixtures of oil and other liquids, suspended in a tall glass of water.

Fast-forward several hundred years. We now have the ability to measure so many phenomena that we can not only predict outcomes but also examine complex ecosystems, understand the cause and effects of changes within them, and have learned to reduce the negative effects—and sometimes eliminate them completely. More than any other technology, sensors (which provide the ability to quantify something) help scientists and everyday people save lives, save resources, and save the world.

It is with this premise that the book you now hold came about. By volunteering a small amount of their time and effort, normal people should be able to participate actively in scientific data gathering that benefits the greater good. If we can benefit ourselves along the way, even better!

The Arduino fits into the picture by positioning itself as the “bridge” between humans and sensors. Never has it been easier to learn about microcontrollers, understand sensor technology, and write code. The Arduino makes it all easy by providing a simple hardware and software platform that runs on any desktop or laptop computer. Furthermore, the programming language in which you write the code that is to run on the Arduino is an easy C-like language called Processing, which automates all of the difficult hardware tasks for you. Finally, a standard electronic interface based upon the “shield” concept makes working with complex hardware a simple matter of plugging in the optional boards. With some basic electronics knowledge, you can even build your own shields to serve customized purposes.

This book covers several sensor types. In addition, we will interface these sensors to the Arduino through a series of progressively complex methods. Initially, simple sensors will be connected directly to the Arduino inputs or via a breadboard. Once a circuit is verified, we will then build the interface circuits on prototyping shields or perf board.
With the primary circuitry complete, we will develop the project into its final form, adding support systems such as power supplies, switches, and jacks, as well as the all-important housing to protect the sensor system from environmental conditions.

It’s All About Sensors

The main theme of this book is constructing Arduino projects that focus on sciences. In particular, this book has a very strong “green focus.” What will make these projects possible are sensors, which are devices that respond electrically to a physical change. Often this response is a change in resistance. For example, a flex sensor will vary its resistance based on how much bend is applied to it. Essentially, the sensor converts one analog (physical) condition to another analog (electrical) condition, such as temperature to resistance or impact pressure to voltage.

By itself, a microprocessor (which lives in a digital world) cannot understand analog values. Resistance or voltage means nothing to a microprocessor. We need some way to convert these values into the ones and zeros of computer language.

At this point, I think we need to define how a microcontroller such as the one built into the Arduino board differs from a microprocessor. In fact, a microcontroller is a microprocessor. However, it has several key differences from the one lurking inside your laptop or desktop. A microcontroller has had several useful peripheral devices built inside the chip casing, along with the CPU.

A microcontroller has RAM, ROM, serial ports, and digital inputs and outputs. All these might be familiar to you already. After all, your personal computer has all the same devices. However, it is important to note that these peripherals are built into the chip instead of sitting on the side. Therefore, they are much more limited than their desktop PC counterparts. Where a traditional PC might have gigabytes of RAM, a microcontroller might have only a few kilobytes.

There is one peripheral device built into the microcontroller that we will focus on again and again throughout this book: the analog to digital converter, or ADC for short. As its name implies, the ADC connects the analog world to the digital world, converting the signals into something the CPU can understand and work with. Before moving on, let’s take a moment to look at the ADC more closely.

Arduino’s Analog to Digital Converter (ADC)

We will be using the analog to digital converter (ADC) extensively throughout the book. The Arduino has an ADC tied to six inputs (labeled Analog0–Analog5). A few of the projects might utilize all six inputs. We might even wish for more! It is the job of the ADC to sample a voltage at the specified input pin, transcribe that voltage to a byte value, and finally deposit that value into a variable you specify in ram.

Essentially, the ADC does nothing more than makes a comparison. It compares the voltage presented at the analog input to another voltage presented at a reference input.

Note The analog reference is considered the highest expected voltage that a signal will present to the analog input. The input will not be damaged by any voltage that is 5 volts or less, but anything above the reference voltage will be reported as the maximum value.