Introduction To Device Art With The Arduino

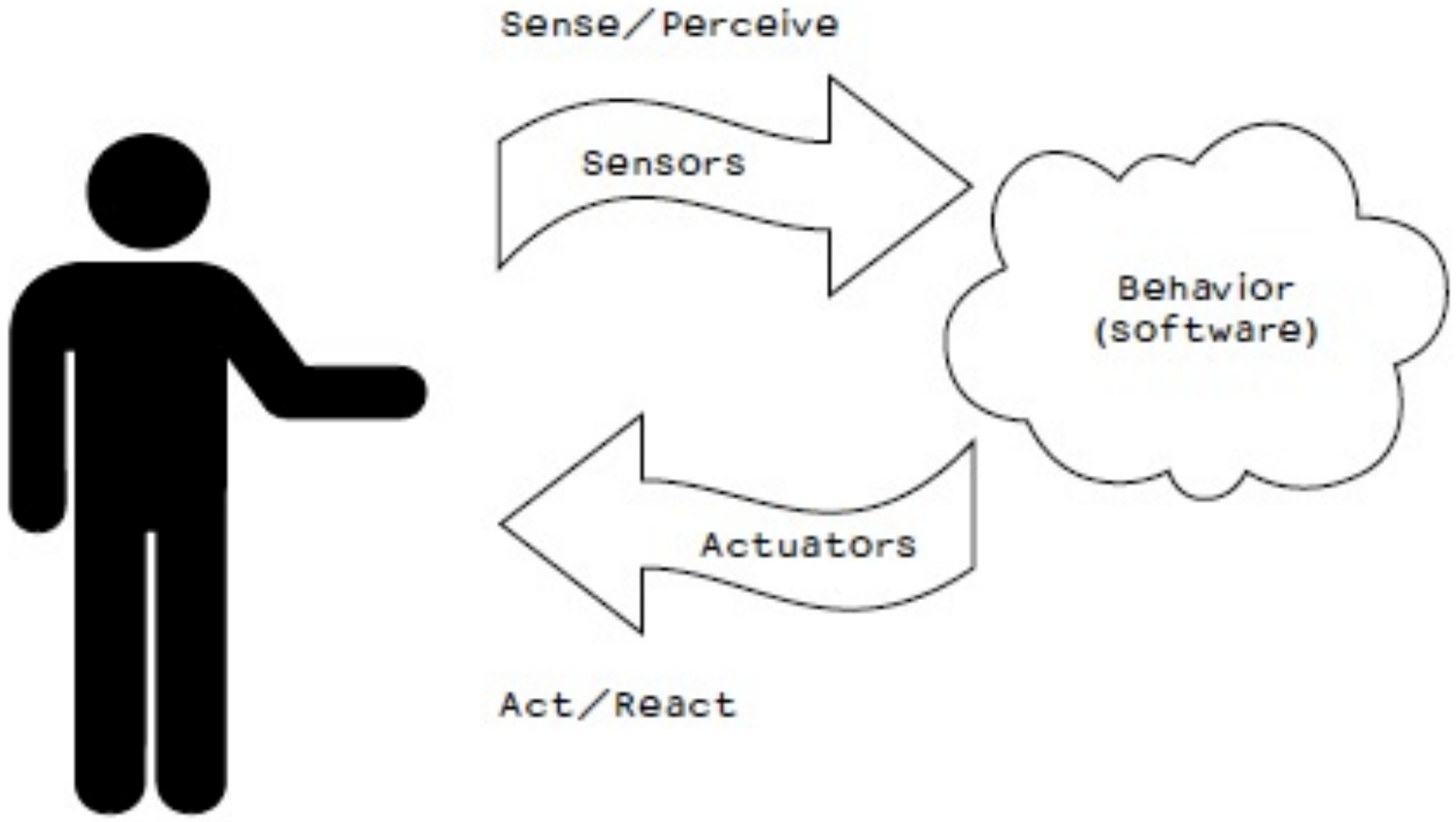
Arduino Is..

- An open-source/-hardware microcontroller device
- Designed to sense
- Designed to control
- Easy to program
- Easy to integrate into designs
- Developed by art-technologists to be practical for simple *and* complex designs with minimal hassle

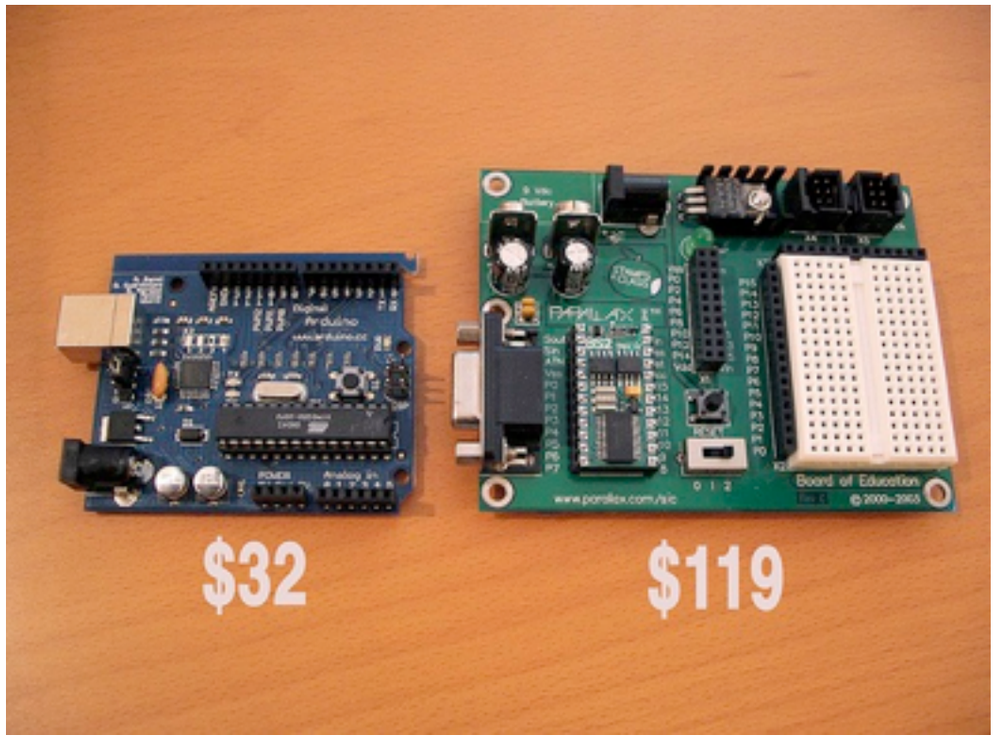
Now you'll learn how to build and program an interactive device.

Anatomy of an Interactive Device

All of the objects we will build using Arduino follow a very simple pattern that we call the "Interactive Device". The Interactive Device is an electronic circuit that is able to sense the environment using sensors (electronic components that convert real-world measurements into electrical signals). The device processes the information it gets from the sensors with behavior that's implemented as software. The device will then be able to interact with the world using actuators, electronic components that can convert an electric signal into a physical action.



Microcontrollers



What is a Microcontroller?

- A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

- Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be communicate with software running on your computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

Why Use Arduino

- There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality.

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50

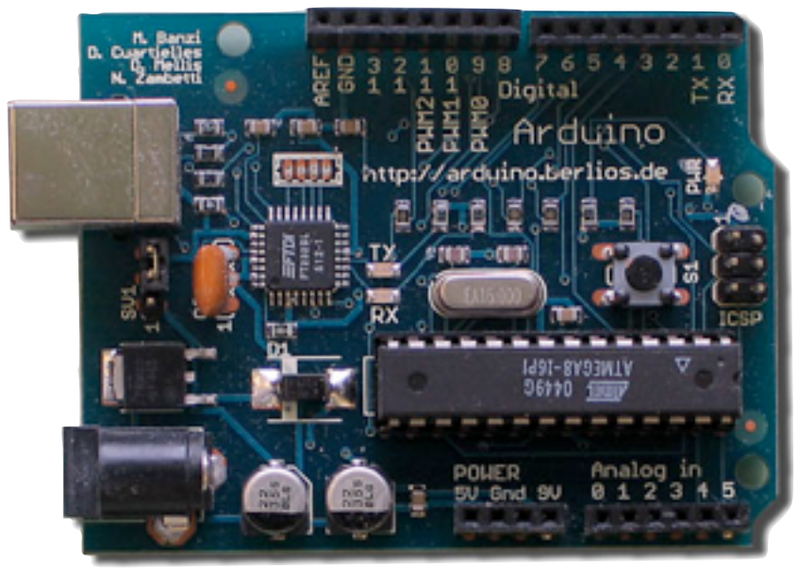
- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

- Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino

- Open source and extensible software- The Arduino software and is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. SImilarly, you can add AVR-C code directly into your Arduino programs if you want to.

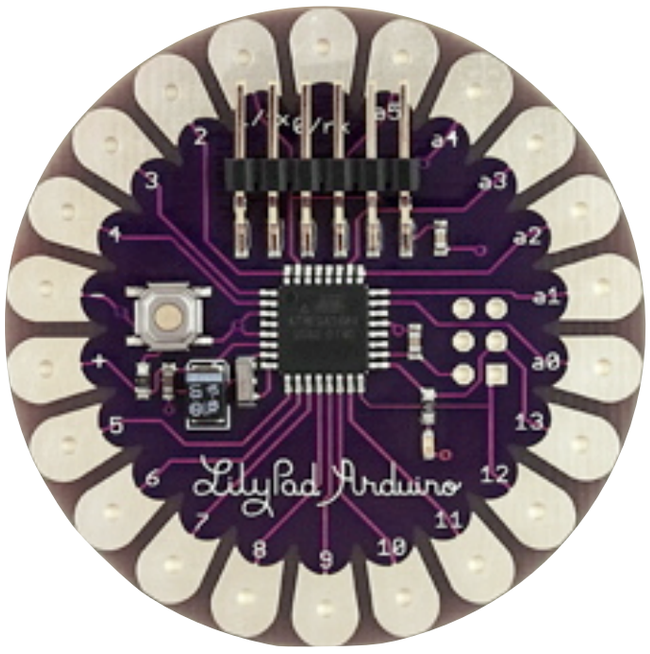
- Open source and extensible hardware - The Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

About the Arduino



14 Digital IO pins (pins 0–13)
These can be inputs or outputs, which is specified by the sketch you create in the IDE.

6 Analog In pins (pins 0–5)
These dedicated analogue input pins take analogue values (i.e., voltage readings from a sensor) and convert them into a number between 0 and 1023.



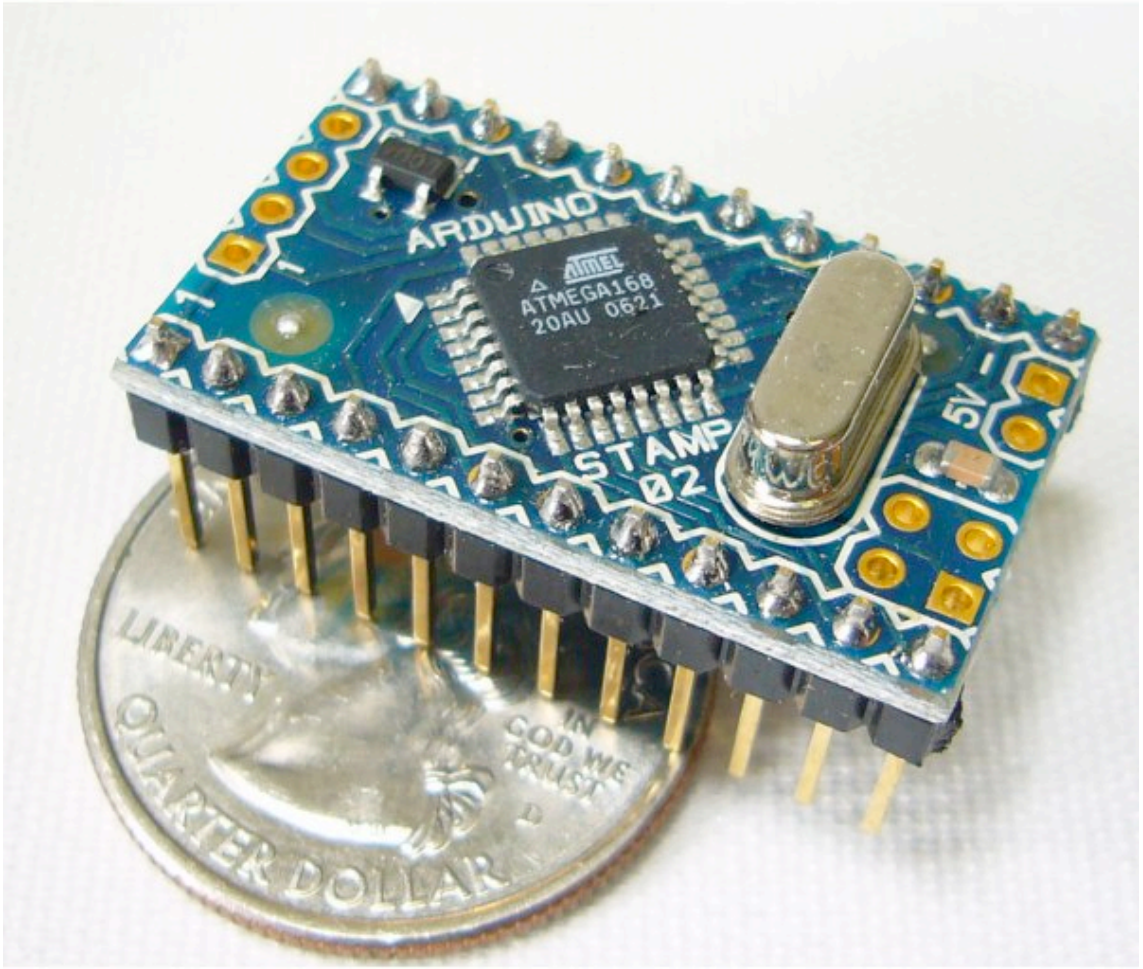
6 Analogue Out pins (pins 3, 5, 6, 9, 10, and 11)
These are actually six of the digital pins that can be reprogrammed for analogue output using the sketch you create in the IDE.

The board can be powered from your computer's USB port, most USB chargers, or an AC adapter (9 volts recommended, 2.1mm barrel tip, center positive).

If there is no power supply plugged into the power
socket, the power will come from the USB board, but as soon as you plug a power supply, the board will automatically use it.

Introduction To Device Art With The Arduino

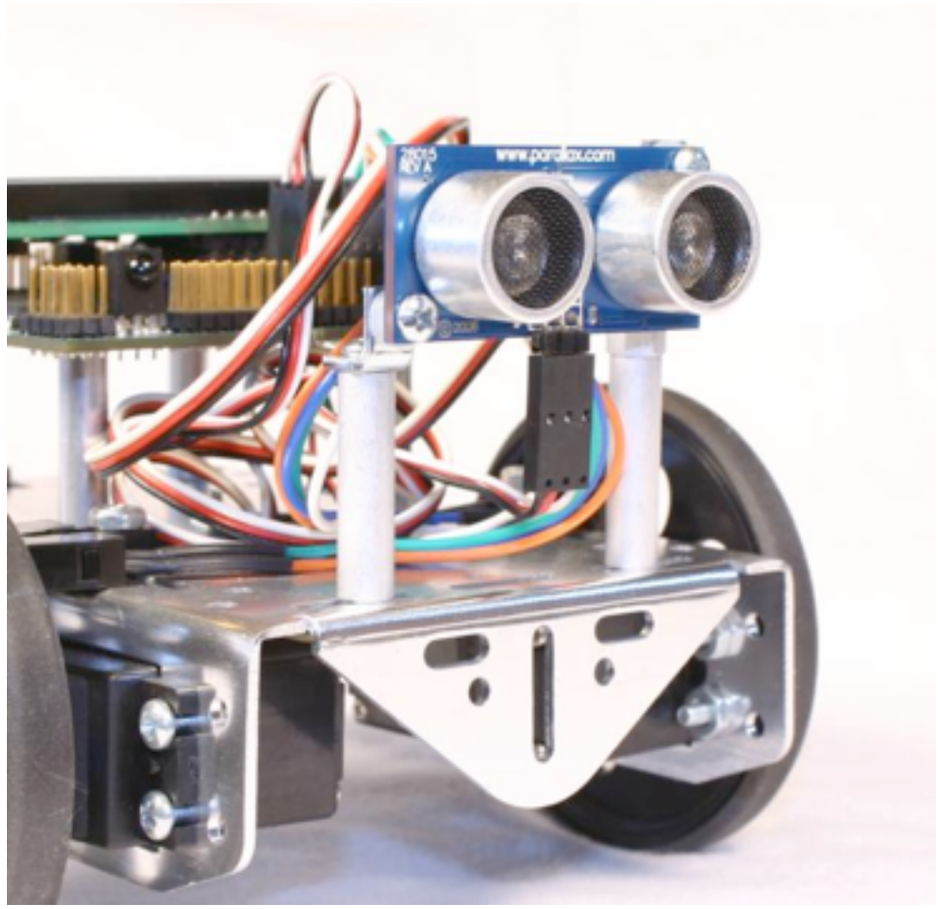
Arduino Mini



- Uses Atmega168
- Twice The Program Space As The Atmega8, Designed For Embedding In Projects
- Uses Same Arduino IDE

Sensors and Actuators

Sensors and actuators are electronic components that allow a piece of electronics to interact with the world.

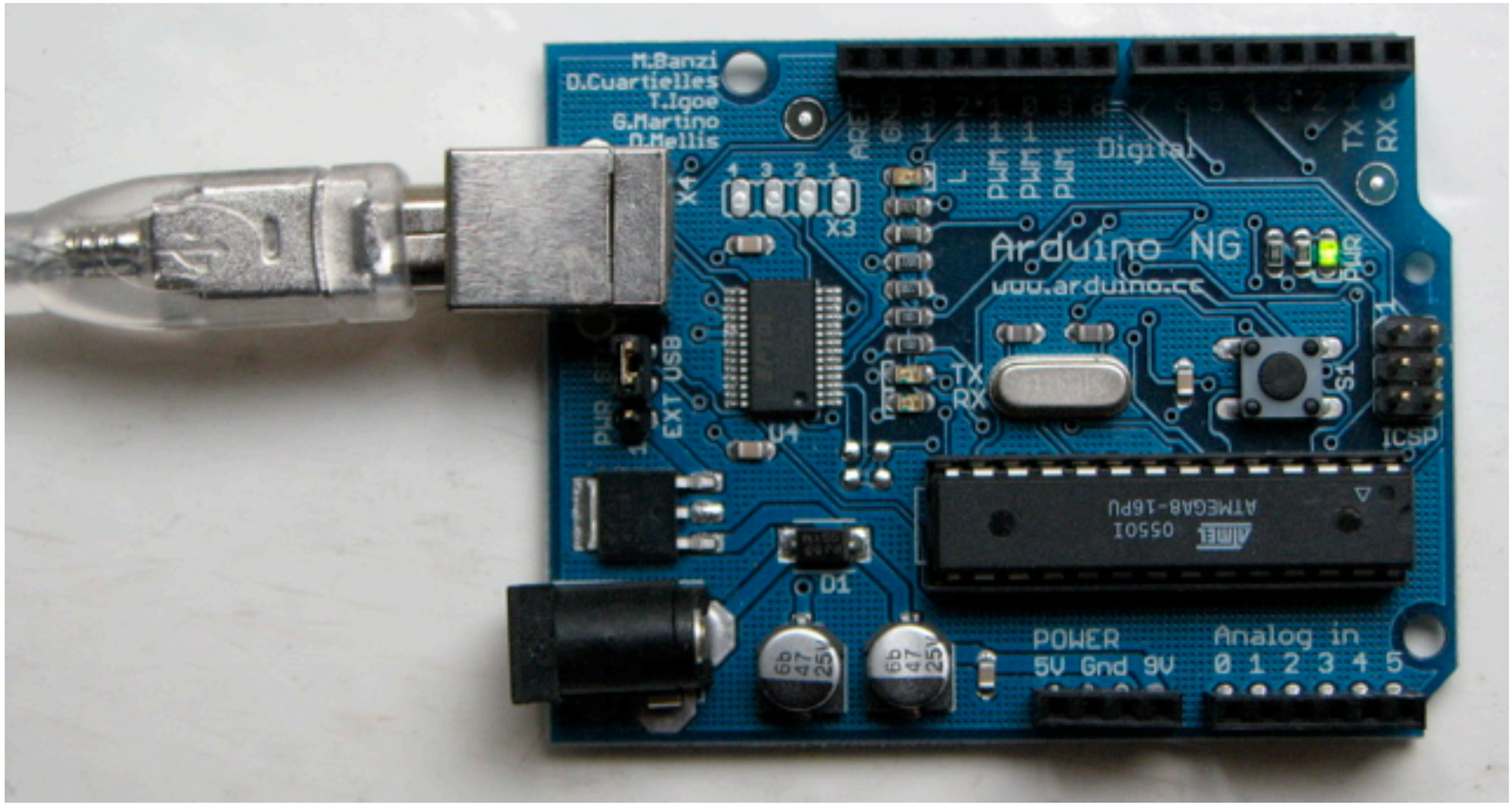


As the microcontroller is a very simple computer, it can process only electric signals (a bit like the electric pulses that are sent between neurons in our brains). For it to sense light, temperature, or other physical quantities, it needs something that can convert them into electricity.

In our body, for example, the eye converts light into signals that get sent to the brain using nerves. In electronics, we can use a simple device called a light-dependent resistor (an LDR or photoresistor) that can measure the amount of light that hits it and report it as a signal that can be understood by the microcontroller.

Once the sensors have been read, the device has the information needed to decide how to react. The decision-making process is handled by the microcontroller, and the reaction is performed by actuators. In our bodies, for example, muscles receive electric signals from the brain and convert them into a movement. In the electronic world, these functions could be performed by a light or an electric motor.

Plug in your Arduino and then plug in the other end of the cord into one of your serial ports. Now go back to your Arduino Environment window...



About the Arduino Software

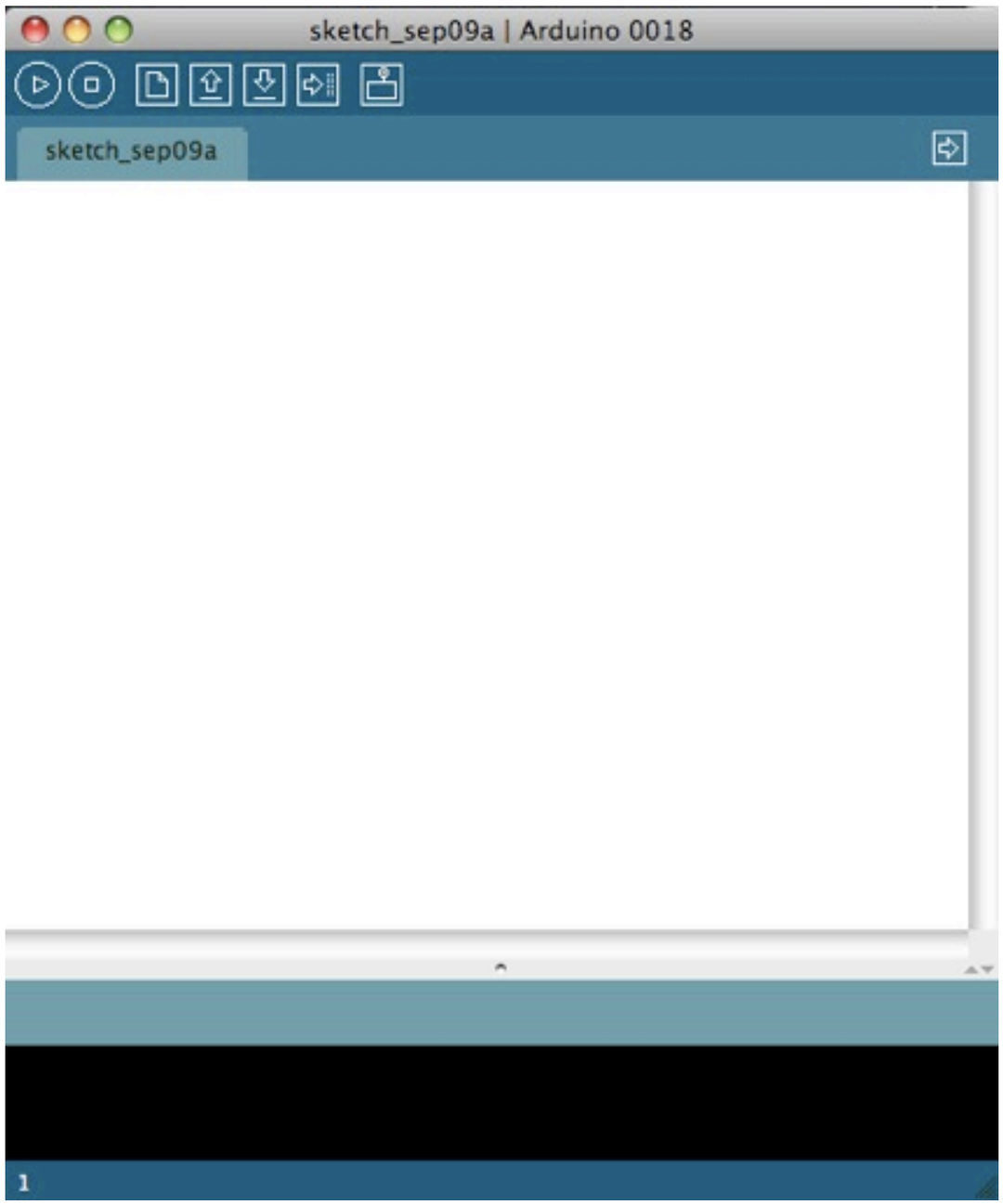
The Software (IDE)

The IDE (Integrated Development Environment) is a special program running on your computer that allows you to write sketches for the Arduino board in a simple language modeled after the Processing www.processing.org) language. The magic happens when you press the button that uploads the sketch to the board: the code that you have written is translated into the C language (which is generally quite hard for a beginner to use), and is passed to the avr-gcc compiler, an important piece of open source software that makes the final translation into the language understood by the microcontroller. This last step is quite important, because it's where Arduino makes your life simple by hiding away asmuch as possible of the complexities of programming microcontrollers.

The Arduino Environment

The Arduino development environment contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Software written using Arduino are called sketches. These sketches are written in the text editor. It has features for cutting/pasting and for searching/ replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino environment including complete error messages and other information. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor:



Toolbar

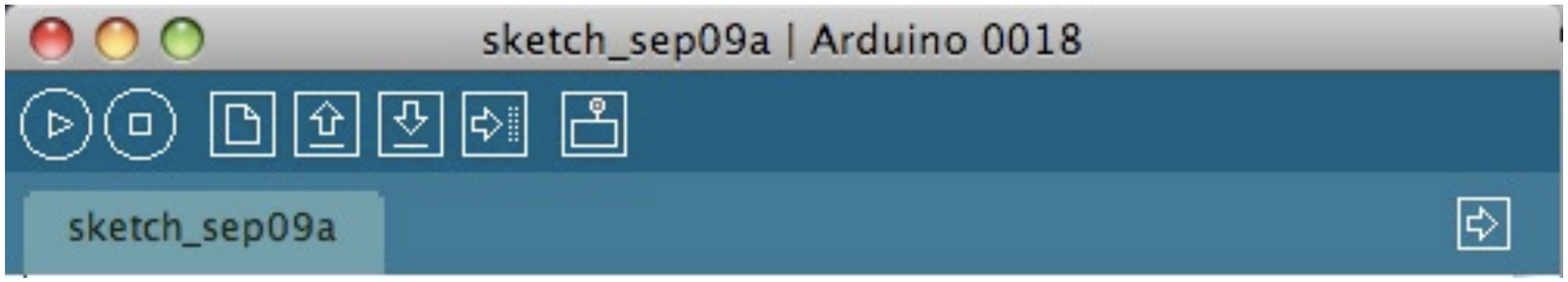
Tabs

Text Editor

Message Area

Text Area

The Arduino Environment



Verify/Compile
Checks your code for errors.

Stop
Stops the serial monitor, or unhighlight other buttons.

New
Creates a new sketch.

Open
Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File | Sketchbook** menu instead.

Save
Saves your sketch.

Upload to I/O Board
Compiles your code and uploads it to the Arduino I/O board. See uploading below for details.

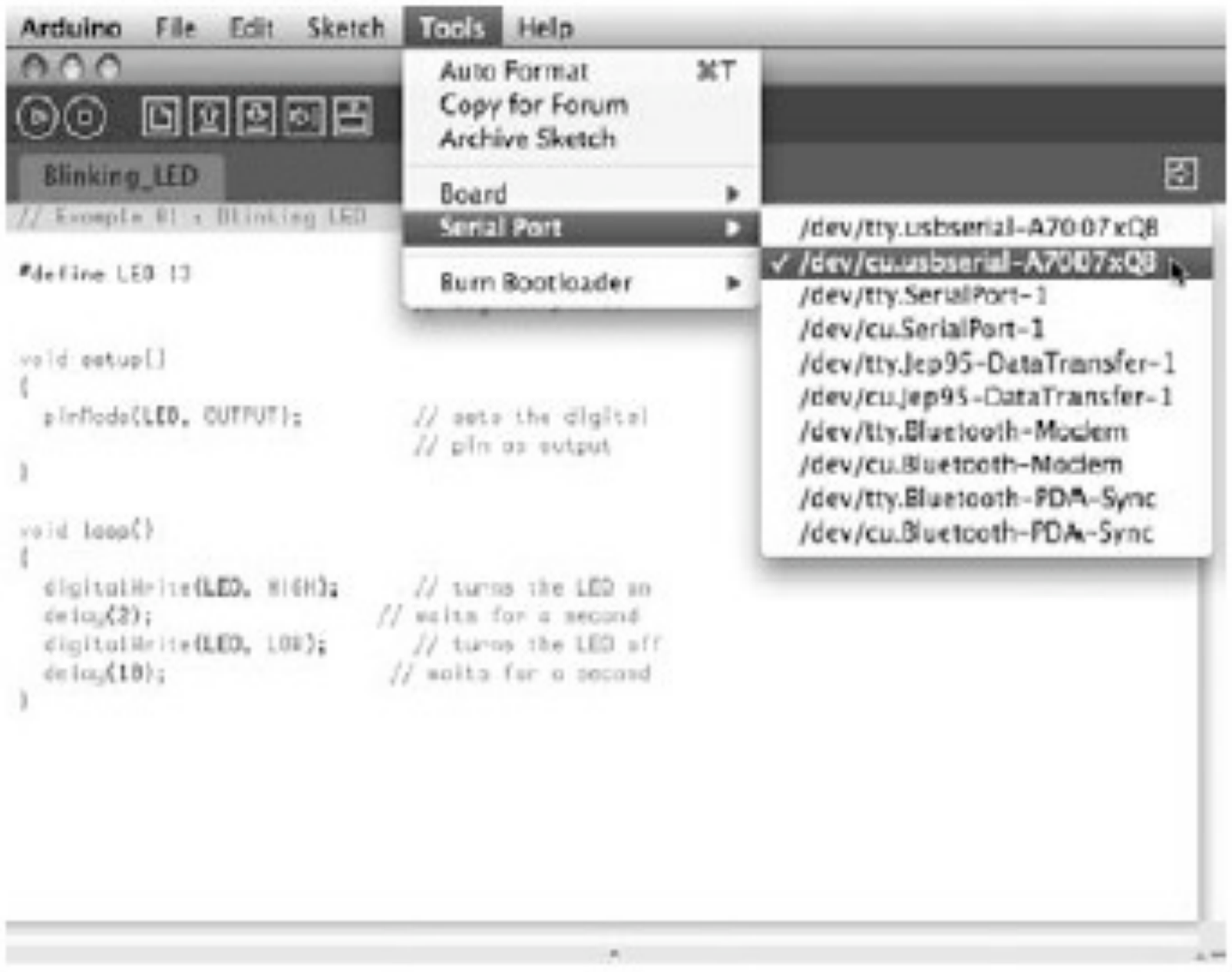
Serial Monitor
Opens the serial monitor.

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive which means only those items relevant to the work currently being carried out are available.

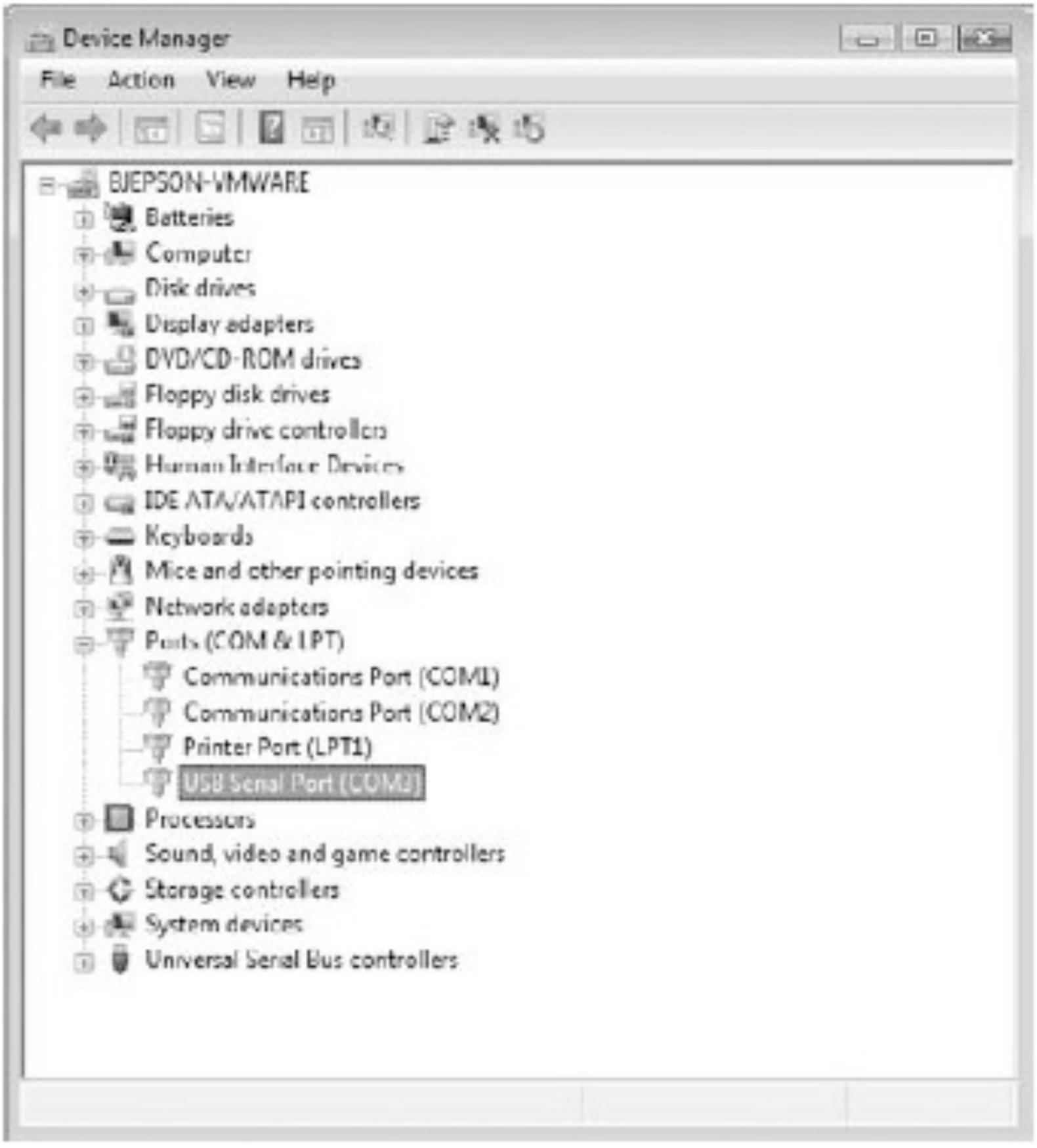
Selecting the Correct Serial Port for Mac

Port Identification: Macintosh

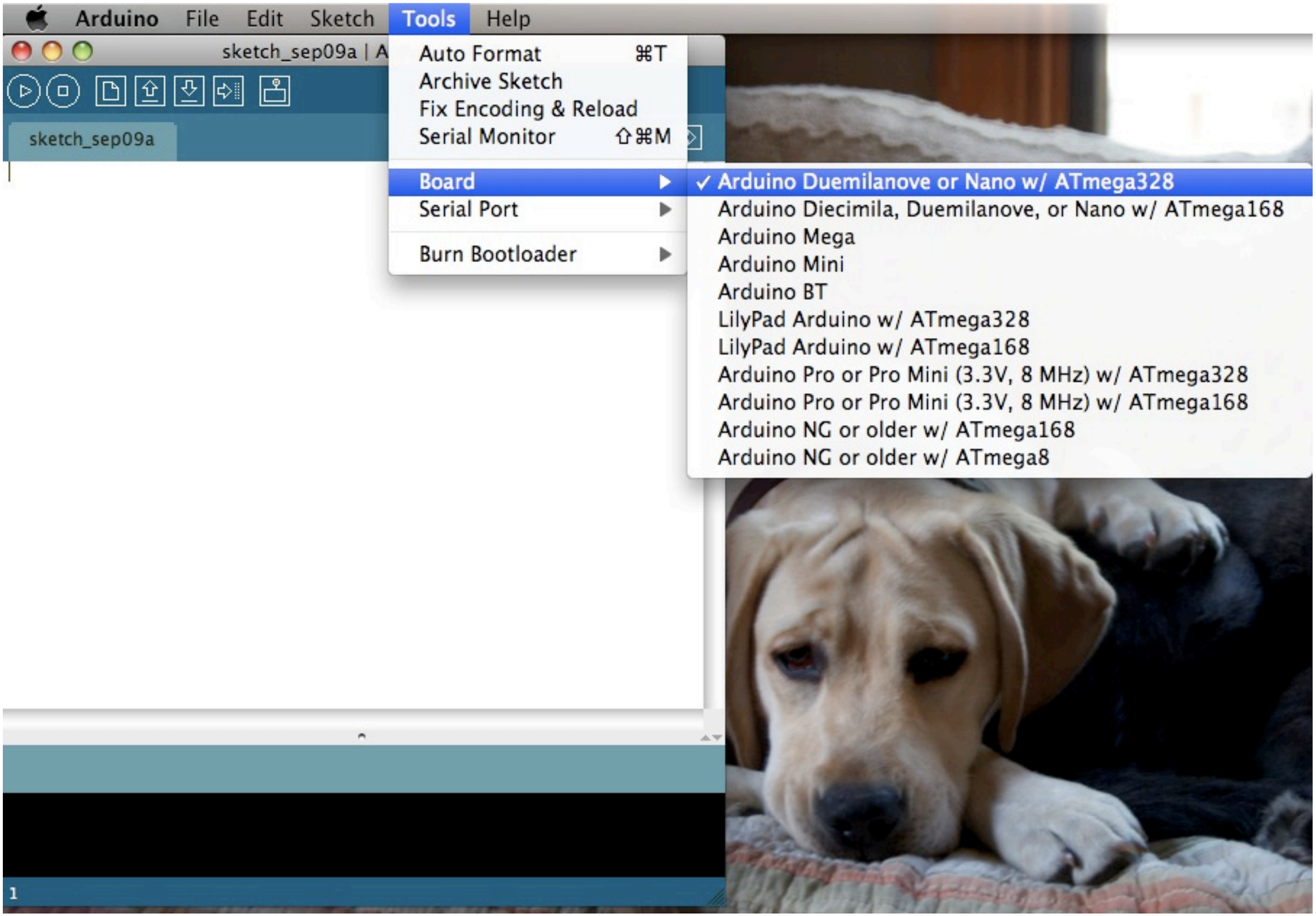
From the Tools menu in the Arduino IDE, select "Serial Port" and select the port that begins with */dev/cu.usbserial-*; this is the name that your computer uses to refer to the Arduino board. Figure 3-3 shows the list of ports.



Selecting the Correct Serial Port for Windows

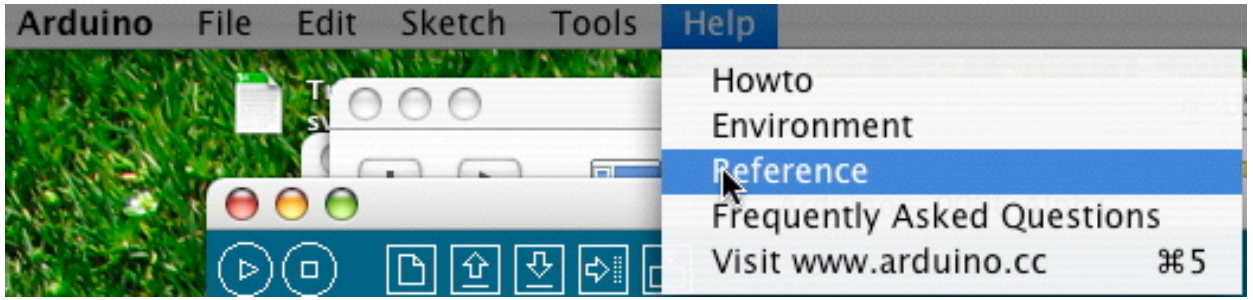


Selecting the Correct Arduino Board

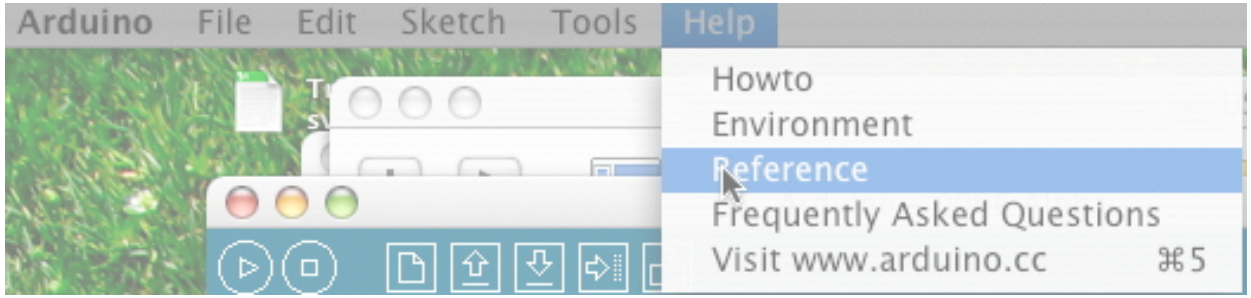


Arduino IDE

Built In Help



Arduino IDE



Some Basic Functionality

Digital I/O

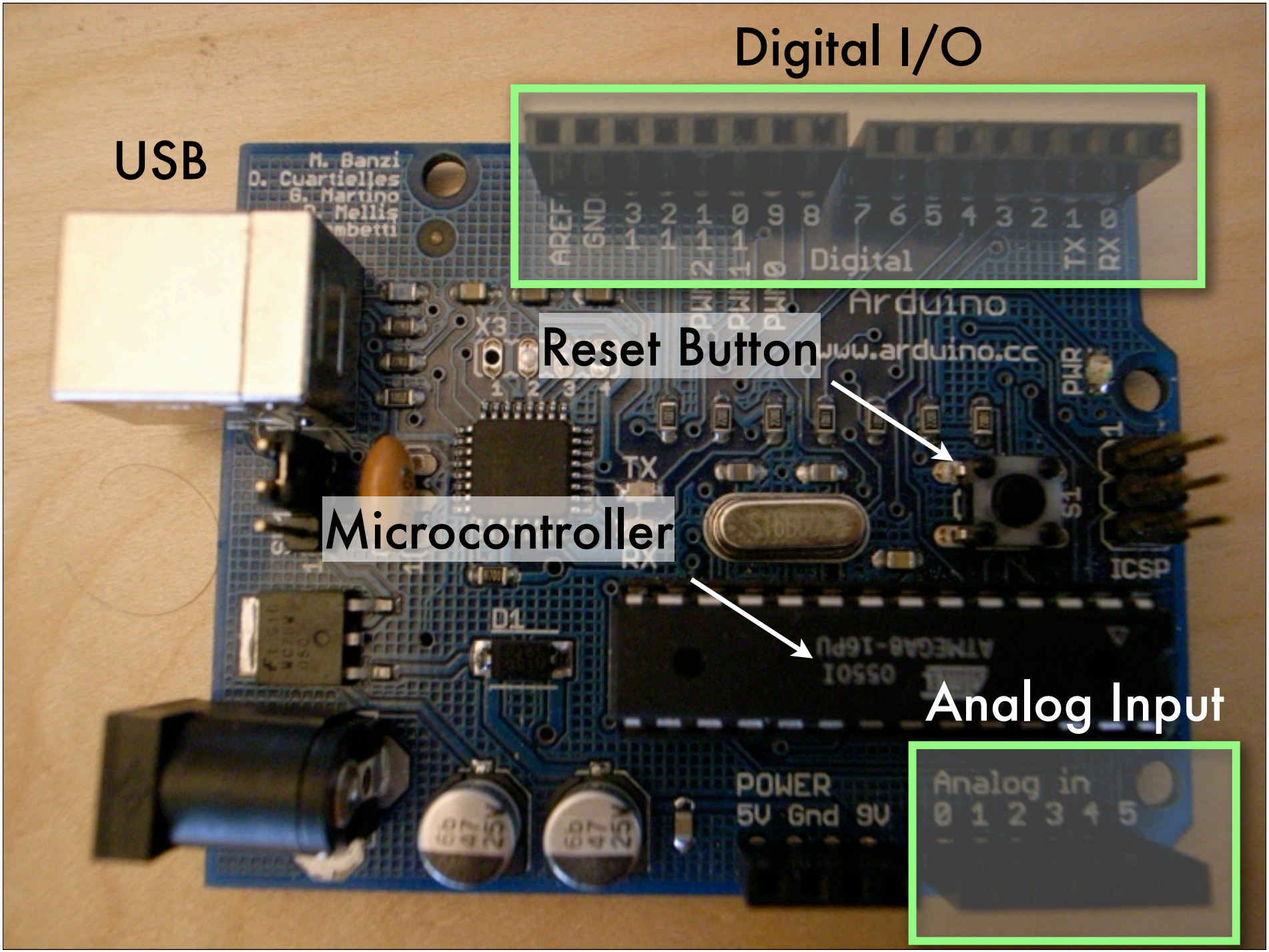
* pinMode(pin, mode)
* digitalWrite(pin, value)
* int digitalRead(pin)
* unsigned long pulseIn(pin, value)

Analog I/O

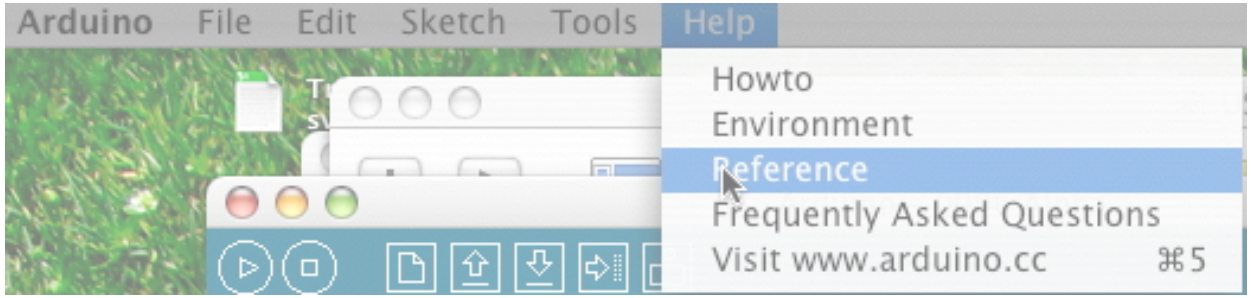
* int analogRead(pin)
* analogWrite(pin, value) - PWM

Handling Time

* unsigned long millis()
* delay(ms)
* delayMicroseconds(us)


Digital I/O
USB
Reset Button
Microcontroller
Analog Input
AREF GND 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Digital
TX RX
Arduino
www.arduino.cc
PWR
ICSP
POWER
5V Gnd 9V
Analog in
0 1 2 3 4 5

Arduino IDE



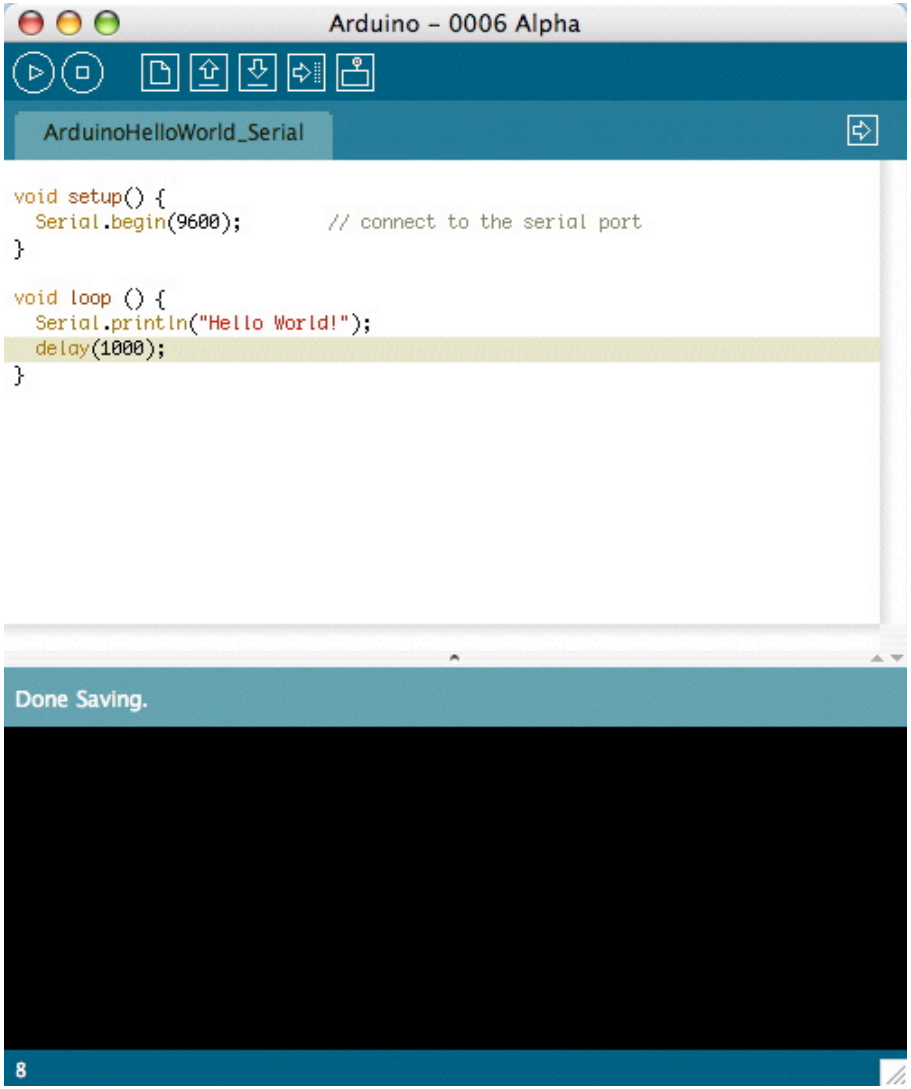
An Arduino program run in two parts:

* void setup()
* void loop()

setup() is preparation, and loop() is execution.
The loop section is the code to be executed -- reading inputs, triggering outputs, etc.
The loop section runs continuously, over-and-over so long as the Arduino is powered.

Tutorial 1: Hello World, Serial Port Edition

First, Type In Code

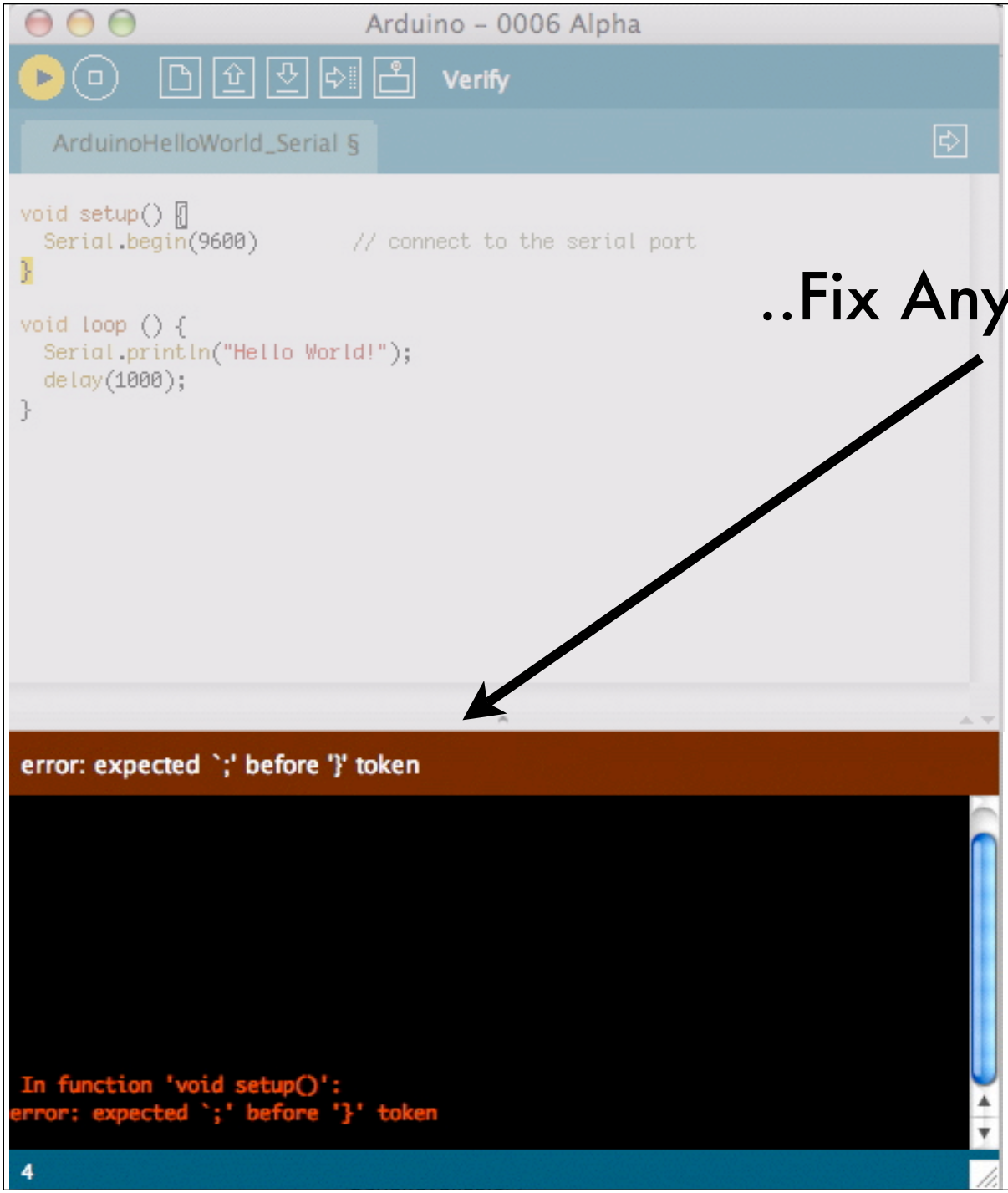



```
void setup() {
  Serial.begin(9600);
}

void loop () {
  Serial.println("Hello World!");
  delay(1000);
}
```

Arduino IDE

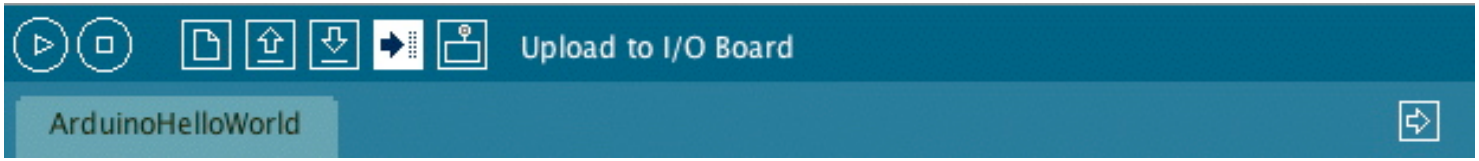
Then Verify (Compile Your Code)



..Fix Any Bugs And Errors

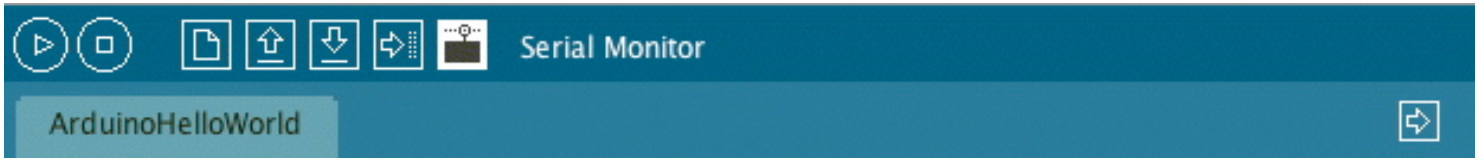
Arduino IDE

Once Your Code Is Free Of Errors, Upload Compiled Code To Your Arduino Board

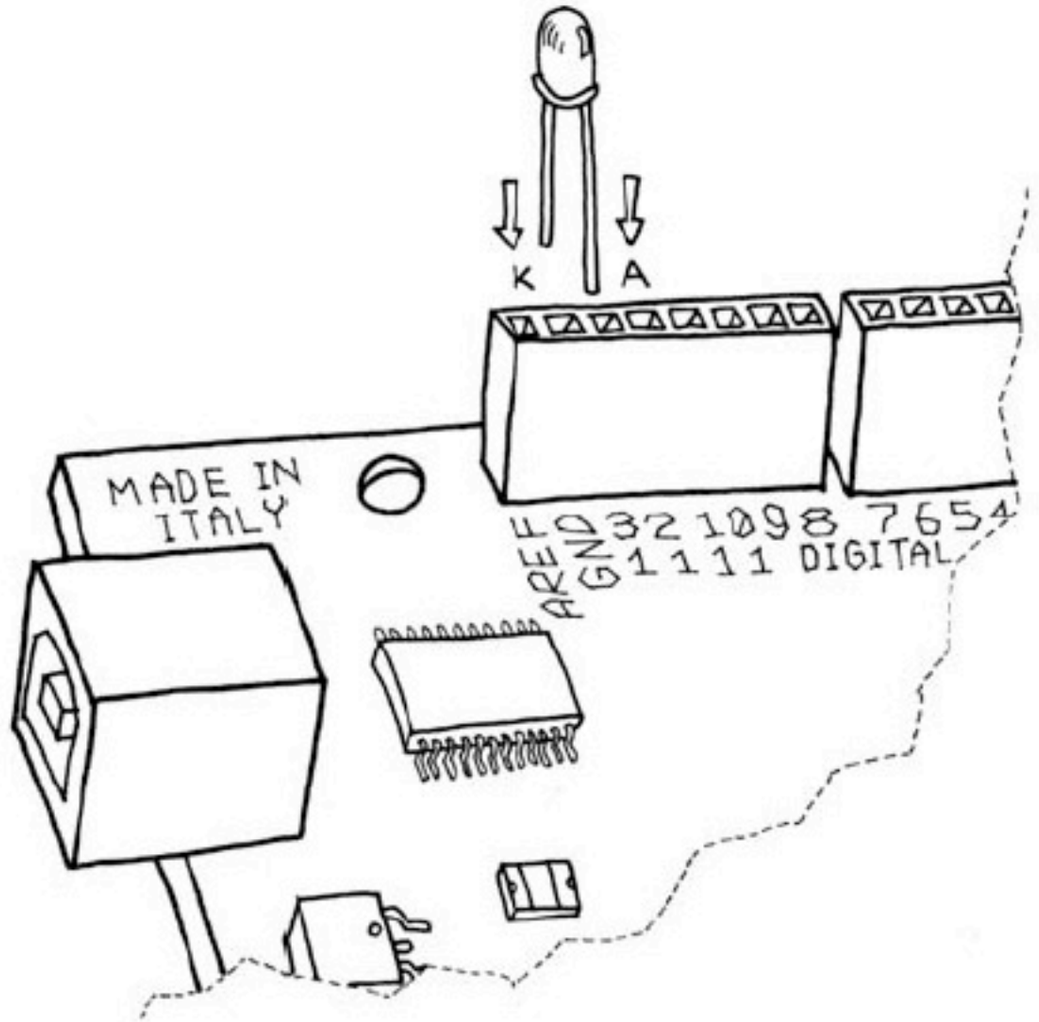


Arduino IDE

Monitor Your Program's Output Using The "Serial Monitor" Button



Blinking an LED



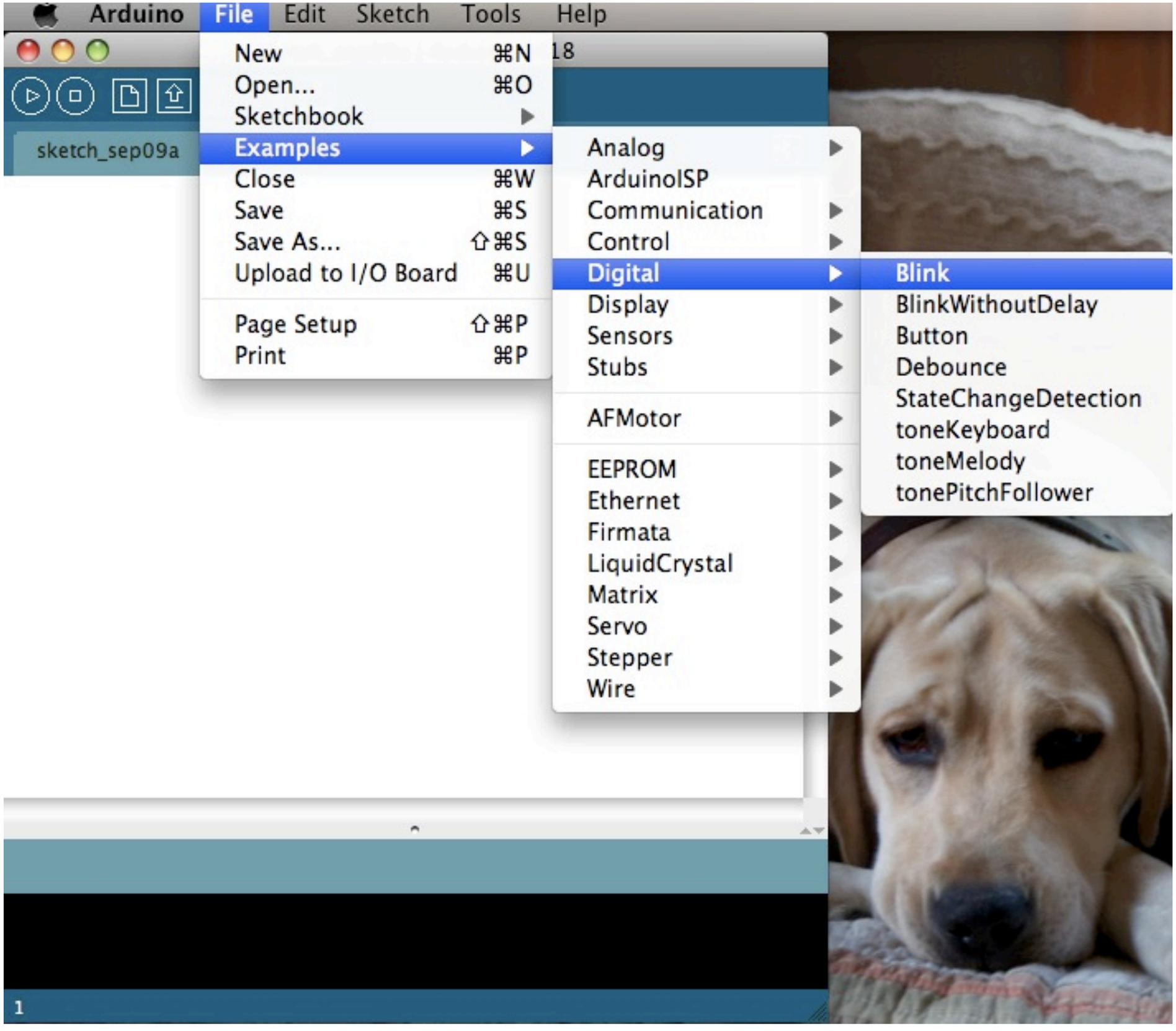
Blinking an LED

The LED blinking sketch is the first program that you should run to test whether your Arduino board is working and is configured correctly. It is alsousually the very first programming exercise someone does when learningto program a microcontroller. A light-emitting diode (LED) is a small electroniccomponent that's a bit like a light bulb, but is more efficient andrequires lower voltages to operate.

Your Arduino board comes with an LED preinstalled. It's marked "L". You can also add your own LED—connect it as shown in Figure 4-2.
K indicates the cathode (negative), or shorter lead; A indicates the anode (positive), or longer lead.

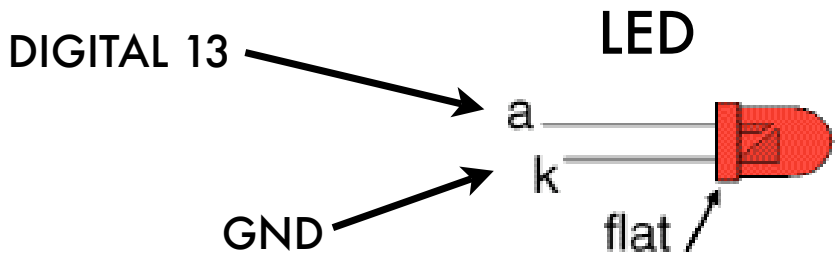
Once the LED is connected, you need to tell Arduino what to do. This is
done through code, that is, a list of instructions that we give the microcontroller to make it do what we want.

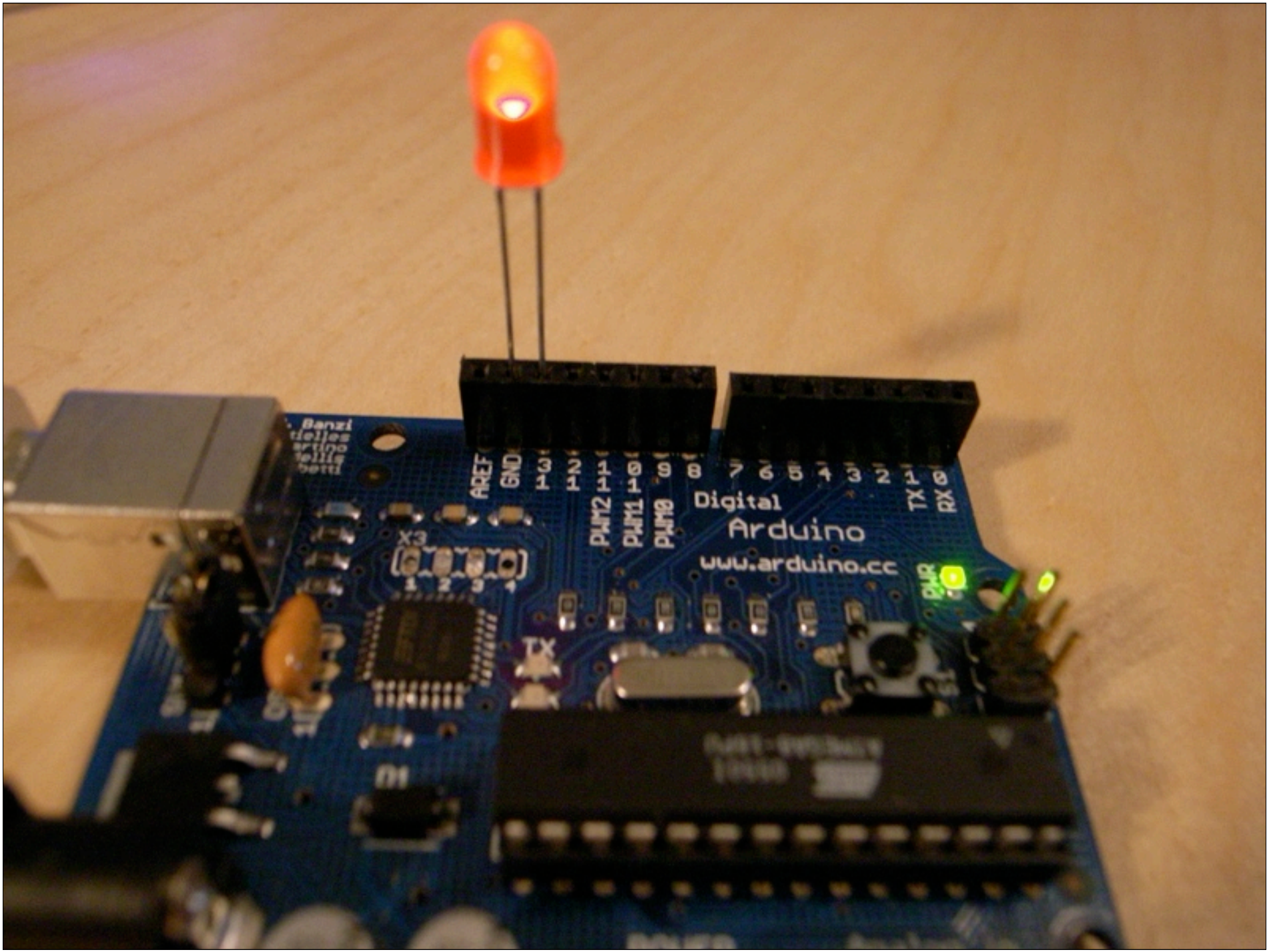
Opening our First Sketch



Tutorial 2: Hello World, LED Edition

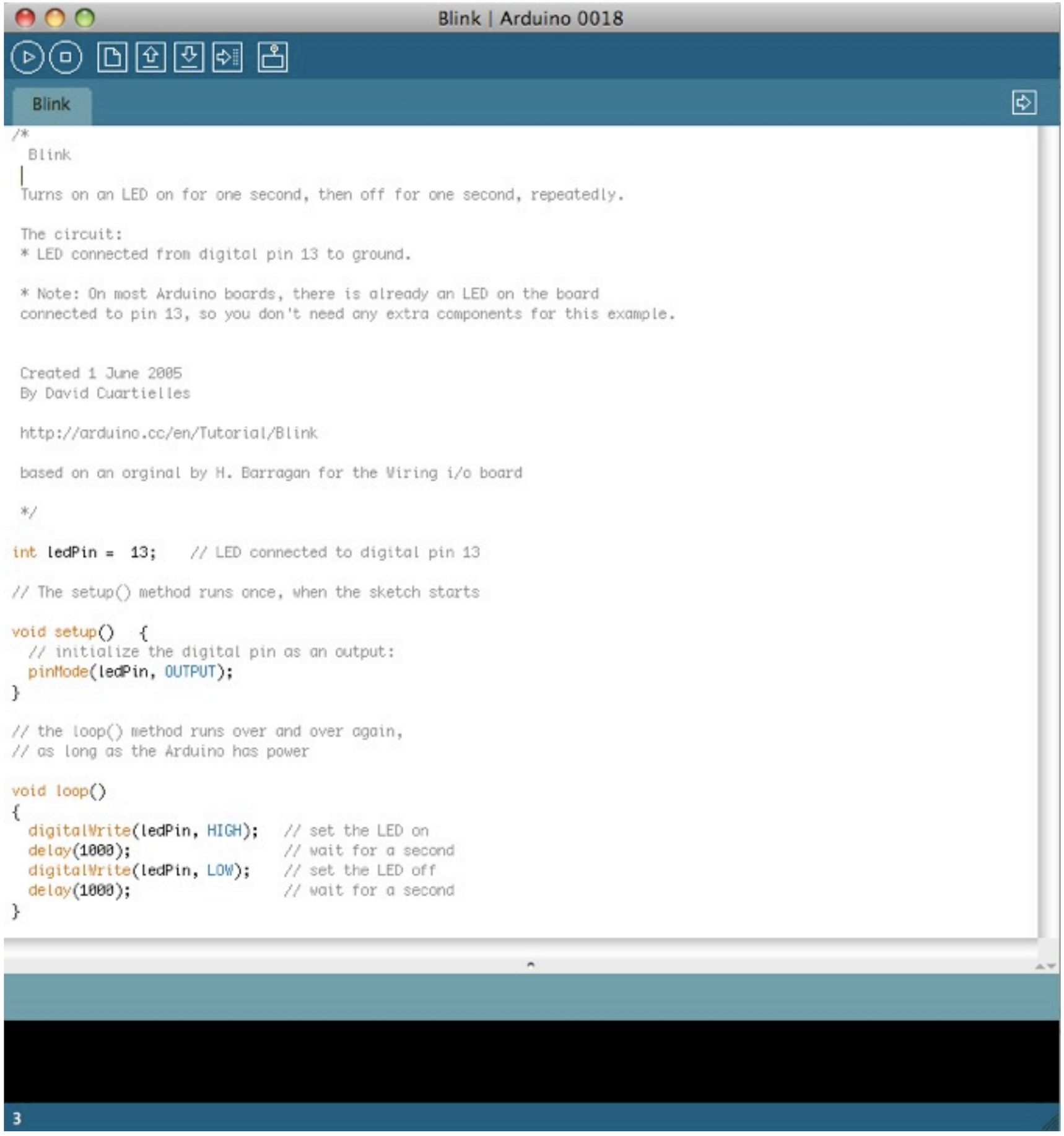
LED


LED
DIGITAL 13
a
k
GND
flat


AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM2
PWM1
PWM0
Digital
TX
RX
Arduino
www.arduino.cc
PWR
X3
1 2 3 4
TX
D1

Lets go through the code and figure out why our LED is blinking...

What is all of this?

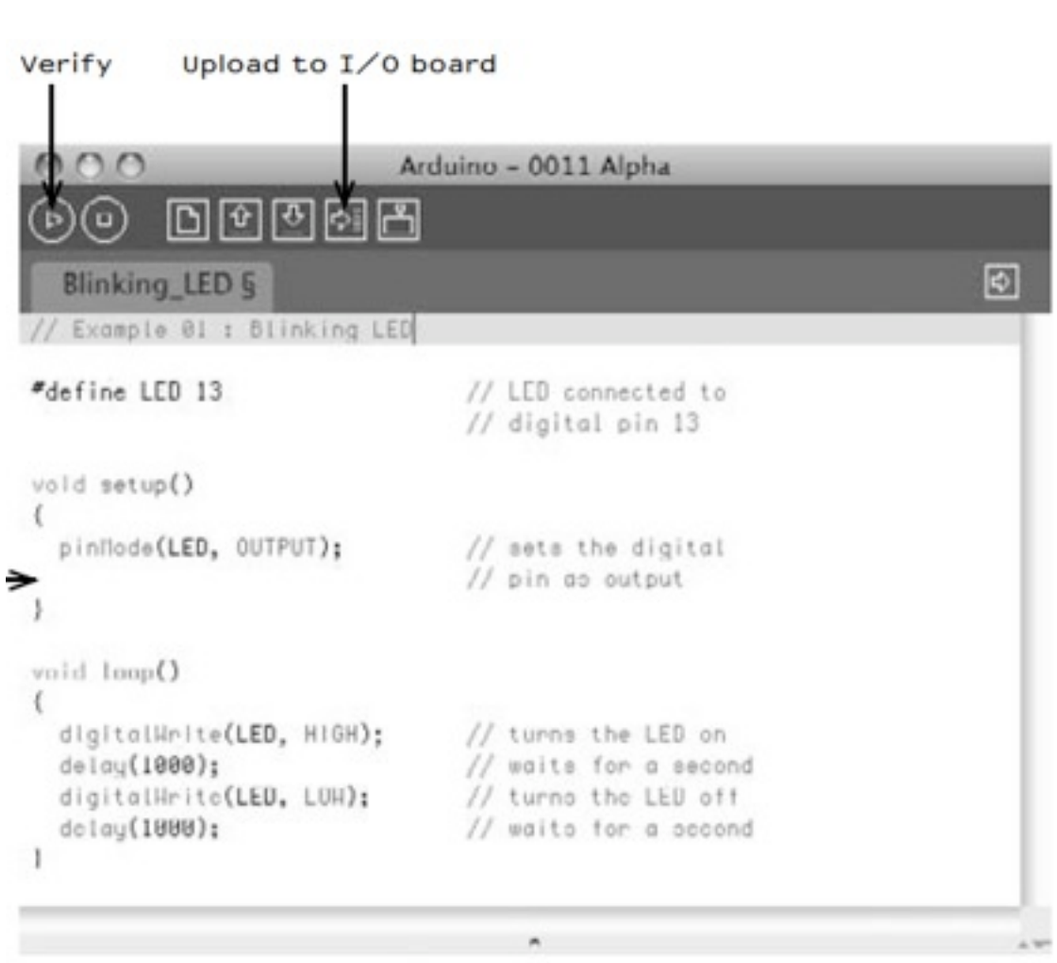


Now that the code is in your IDE, you need to verify that it is correct. Press the “Verify” button. If everything is correct,
you’ll see the message “Done compiling” appear at the bottom of the Arduino IDE. This message means that the Arduino IDE has translated your sketch into an executable program that can be run by the board, a bit like an .exe file in Windows or an .app file on a Mac.

At this point, you can upload it into the board: press the Upload to I/O Board button (see Figure 4-3). This will reset the board, forcing it to stop what it’s doing and listen for instructions coming from the USB port. The Arduino IDE sends the current sketch to the board, which will store it in its memory and eventually run it.

You will see a few messages appear in the black area at the bottom of the window, and just above that area, you’ll see the message “Done uploading” appear to let you know the process has completed correctly. There are two LEDs, marked RX and TX, on the board; these flash every time a byte is sent or received by the board. During the upload process, they keep flickering.

Assuming that the sketch has been uploaded correctly, you will see the LED “L” turn on for a second and then turn off for a second. If you installed a separate LED as shown back in Figure 4-2, that LED will blink, too. What you have just written and ran is a “computer program”, or sketch, as Arduino programs are called. Arduino, as I’ve mentioned before, is a small computer, and it can be programmed to do what you want. This is done using a programming language to type a series of instructions in the Arduino IDE, which turns it into an executable for your Arduino board.



Brackets, Set Up, & Loops

Notice the presence of curly brackets, which are used to group together lines of code. These are particularly useful when you want to give a name to a group of instructions. If you're at dinner and you ask somebody, "Please pass me the Parmesan cheese," this kicks off a series of actions that are summarized by the small phrase that you just said.

As we're humans, it all comes naturally, but all the individual tiny actions required to do this must be spelled out to the Arduino, because it's not as powerful as our brain. So to group together a number of instructions, you stick a { before your code and an } after.

You can see that there are two blocks of code that are defined in this way here. Before each one of them there is a strange command:
void setup()

This line gives a name to a block of code. If you were to write a list of instructions that teach Arduino how to pass the Parmesan, you would write *void passTheParmesan()* at the beginning of a block, and this block would become an instruction that you can call from anywhere in the Arduino code.

These blocks are called functions. If after this, you write *passTheParmesan()* anywhere in your code, Arduino will execute those instructions and continue where it left off.

The Code Step by Step

```
/*
  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 orginal by H. Barragan for the Wiring i/o board

*/
```

COMMENTS

/* insert text here */

This is a paragraph comment. This text is not actually ‘read’ by your arduino. This is helpful to label your sketch to remind yourself or other users (if you share it), what the code below actually does

// Example 01 : Blinking LED

A comment is a useful way for us to write little notes. The preceding title
comment just reminds us that this program, Example 01, blinks an LED. This is just like the above however, when you use the // you are only commenting a line of code rather than an entire paragraph

Brackets, Set Up, & Loops

Arduino expects two functions to exists—one called *setup()* and one called ***loop()***.

setup() is where you put all the code that you want to execute once at the beginning of your program and *loop()* contains the core of your program, which is executed over and over again.

This is done because Arduino is not like your regular computer—it cannot run multiple programs at the same time and programs can't quit. When you power up the board, the code runs; when you want to stop, you just turn it off.

The Code Step by Step

```
int ledPin =  13;    // LED connected to digital pin 13

// The setup() method runs once, when the sketch starts

void setup()   {
```

int ledPin = 13 **// LED connected to digital pin 13**

int is like an automatic search and replace for your code; in this case, it's telling Arduino to write the number 13 every time the word *LED* appears. The replacement is the first thing done when you click Verify or Upload to I/O Board (you never see the results of the replacement as it's done behind the scenes). We are using this command to specify that the LED we're blinking is connected to the Arduino pin 13.

void setup()

This line tells Arduino that the next block of code will be called *setup()*. The loop**()** method runs whatever is inside of the brackets **{ }** over and over again as long as the Arduino has power

{

With this opening curly bracket, a block of code or messages to the arduino begins.

The Code Step by Step

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

pinMode(ledPin, OUTPUT); // sets the digital pin as output

Finally, a really interesting instruction. *pinMode* tells Arduino how to configure a certain pin. Digital pins can be used either as INPUT or OUTPUT.
In this case, we need an output pin to control our LED, so we place the number of the pin and its mode inside the parentheses. *pinMode* is a function, and the words (or numbers) specified inside the parentheses are arguments. INPUT and OUTPUT are constants in the Arduino language.
(Like variables, constants are assigned values, except that constant values are predefined and never change).

}

This closing curly bracket signifies the end of the *setup()* function.

The Code Step by Step

```
void loop()
{
  digitalWrite(ledPin, HIGH);   // set the LED on
```

void loop()
{

loop() is where you specify the main behaviour of your interactive device. It will be repeated over and over again until you switch the board off.

digitalWrite(ledPin, HIGH); // turns the LED on

As the comment says, *digitalWrite()* is able to turn on (or off) any pin that has been configured as an OUTPUT. The first argument (in this case, *LED*) specifies which pin should be turned on or off (remember that *LED* is a constant value that refers to pin 13, so this is the pin that's switched). The second argument can turn the pin on (HIGH) or off (LOW).

Imagine that every output pin is a tiny power socket, like the ones you have on the walls of your apartment. European ones are 230 V, American ones are 110 V, and Arduino works at a more modest 5 V. The magic here is when software becomes hardware. When you write *digitalWrite(LED, HIGH)*, it turns the output pin to 5 V, and if you connect an LED, it will light up. So at this point in your code, an instruction in software makes something happen in the physical world by controlling the flow of electricity to the pin. Turning on and off the pin at will now let us translate these into something more visible for a human being; the LED is our actuator.

The Code Step by Step

```
delay(1000);                    // wait for a second
```

delay(1000); // waits for a second

Arduino has a very basic structure. Therefore, if you want things to happen with a certain regularity, you tell it to sit quietly and do nothing until it is time to go to the next step. *delay()* basically makes the processor sit there and do nothing for the amount of milliseconds that you pass as an argument. Milliseconds are thousandths of seconds; therefore, 1000 milliseconds equals 1 second. So the LED stays on for one second here.

The Code Step by Step

```
  digitalWrite(ledPin, LOW);    // set the LED off
  delay(1000);                  // wait for a second
}
```

digitalWrite(ledPin, LOW); // turns the LED off

This instruction now turns off the LED that we previously turned on. Why do we use HIGH and LOW? Well, it's an old convention in digital electronics. HIGH means that the pin is on, and in the case of Arduino, it will be set at 5 V. LOW means 0 V. You can also replace these arguments mentally with ON and OFF.

delay(1000); // waits for a second

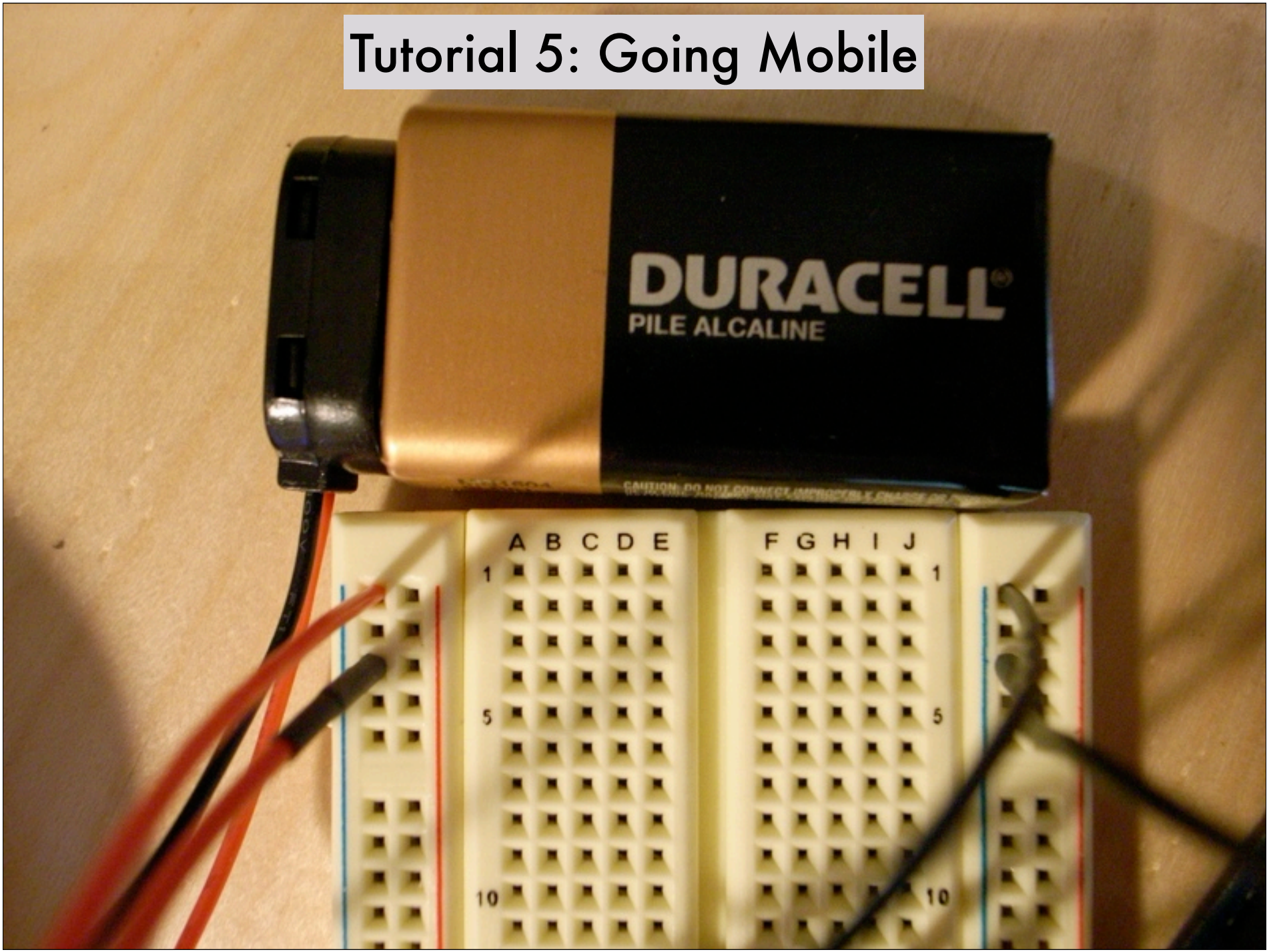
Here, we delay for another second. The LED will be off for one second.

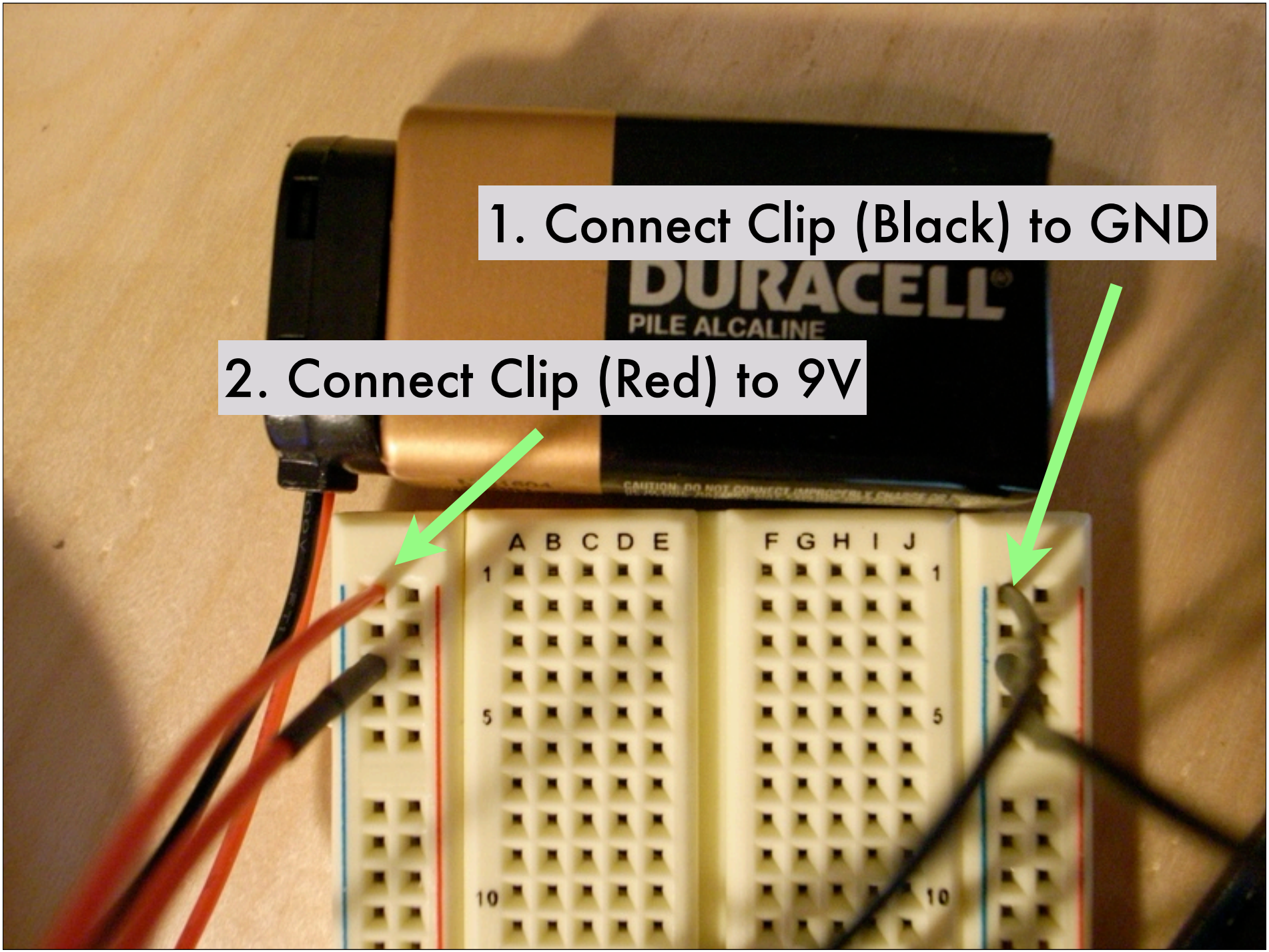
}

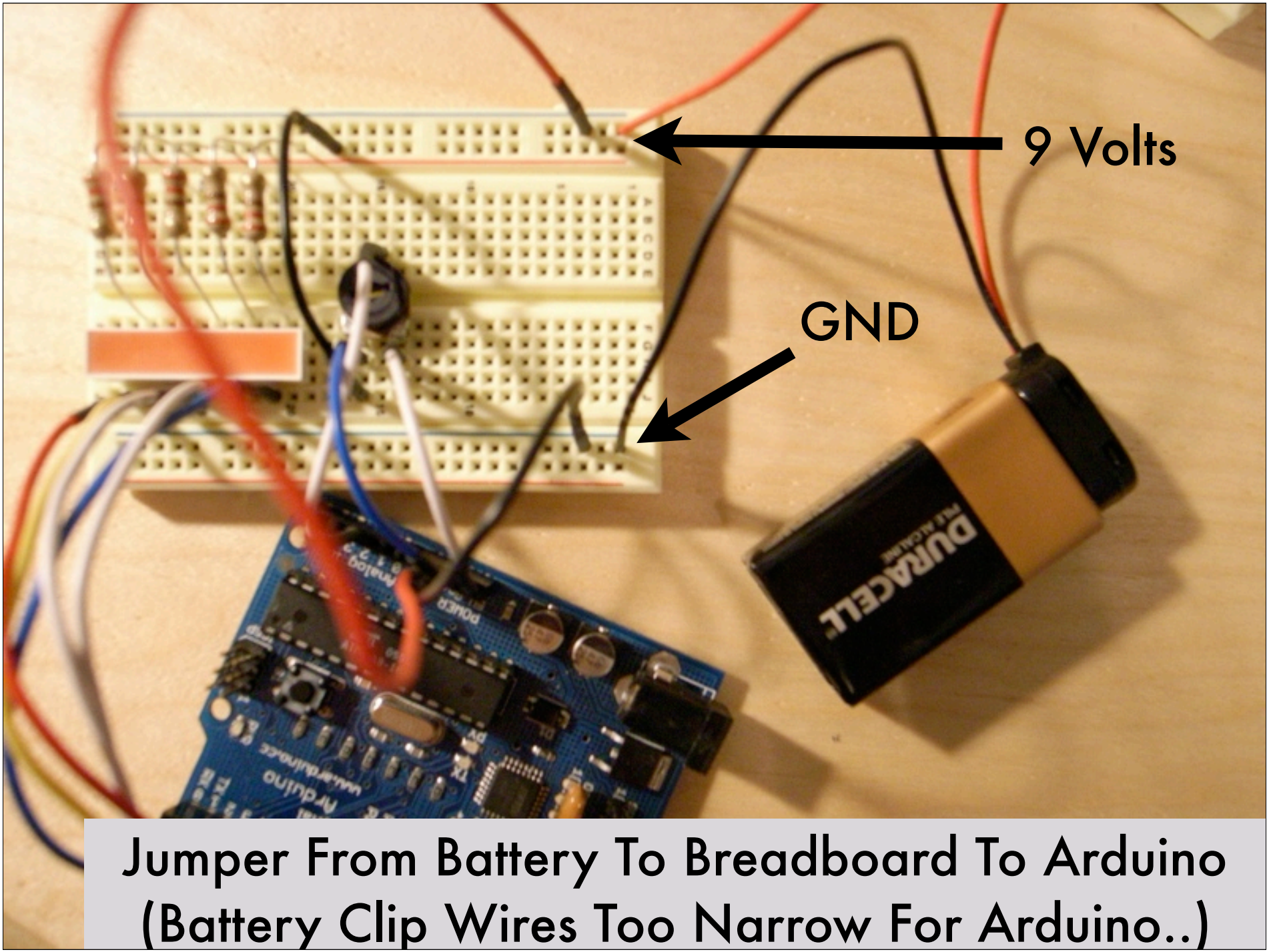
This closing curly bracket marks end of the loop function.

Challenge: Change how fast your LED blinks

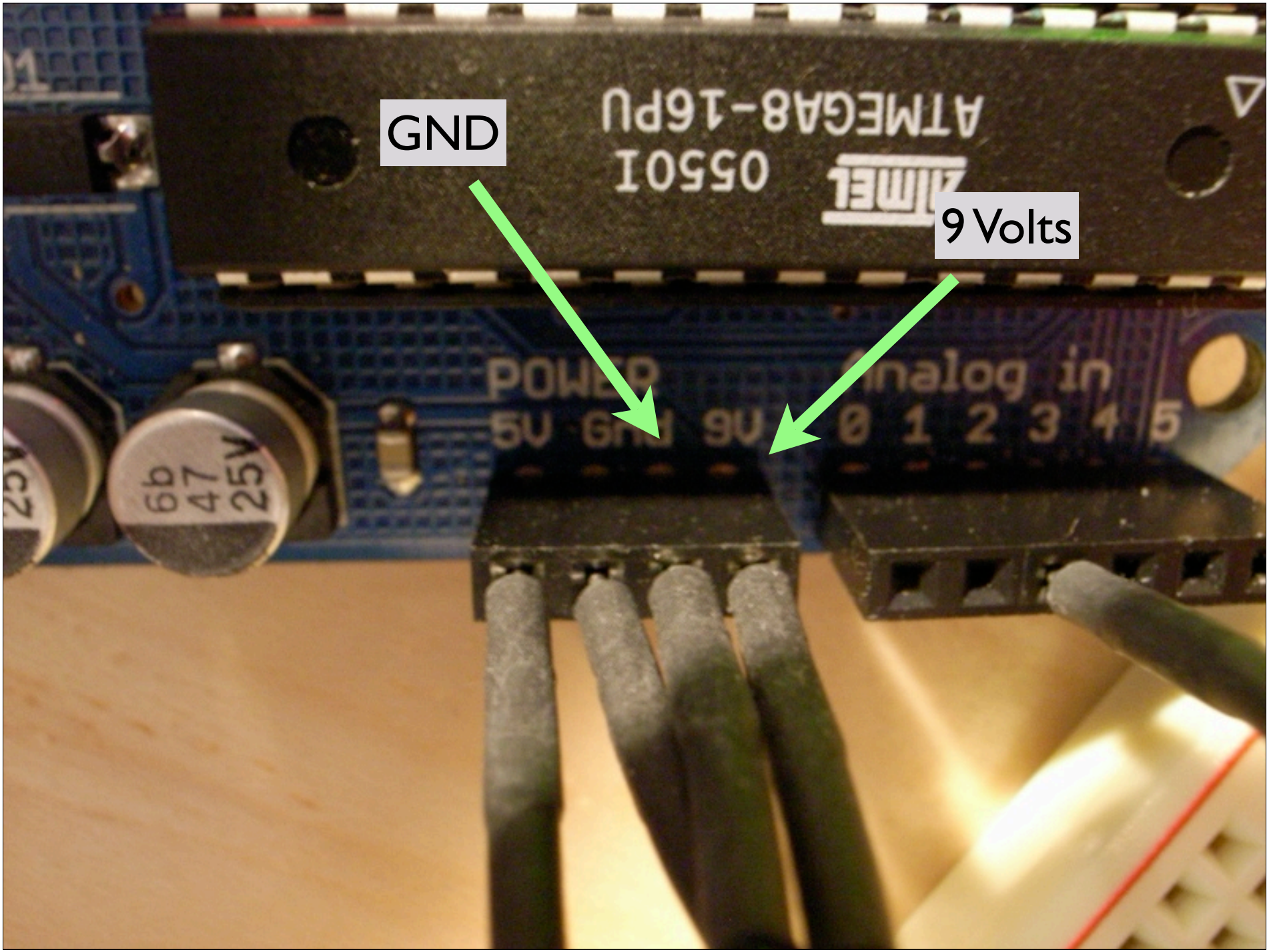
Tutorial 5: Going Mobile




1. Connect Clip (Black) to GND
2. Connect Clip (Red) to 9V
DURACELL
PILE ALCALINE
A B C D E
F G H I J
1
5
10



Jumper From Battery To Breadboard To Arduino
(Battery Clip Wires Too Narrow For Arduino..)


GND
9 Volts
ATMEGA8-16PU
0550I
POWER
5V GND 9V
Analog in
0 1 2 3 4 5
6b
47
25V