Add Arduino power to your projects

Get started with coding for the Arduino platform

John Wargo

@johnwargo

John is a professional software developer, writer, presenter, father, husband, and geek. He is currently a Program Manager at Microsoft, working on Visual Studio Mobile Center. You can find him at **johnwargo.com**

So you want to start programming microcontrollers and doing some cool projects with the hardware. You've selected Arduino as your starting platform, purchased a popular Arduino board, and you're ready to get started. What's next? In this short article, we'll show you how to get started coding for Arduino.

Arduino (**arduino.cc**) is a very popular hardware platform for computer-controlled hardware projects. Arduino is a small, inexpensive, programmable microcontroller that exposes a multitude of input and output (I/O) connections you can use to create computer-controlled circuits, wiring in switches, lights, sensors, and more. It's an open hardware platform, which means that the hardware specification is open source, so anyone with the means can design and distribute their own Arduino-compatible hardware. Therefore, there's a series of devices made by arduino.cc and a bevy of 'compatible' devices from other vendors as well.

To program an Arduino device, you'll code applications in a language similar to very old programming language called C; these applications are called sketches. Because the Arduino is basically a small computer system, although with limited processing speed and memory, the platform supports a subset of the capabilities of C. You'll code your Arduino applications in an integrated development environment (IDE); Arduino offers both a locally installed IDE or a cloud IDE to use for your projects. There are alternative IDEs available as well; you can find a list of options at **hsmag.cc/aQJqkJ**.

Figure 1 Arduino development architecture

When creating sketches, you'll code your sketch in an IDE, then connect your Arduino compatible device to your PC using an USB cable. With that in place, the IDE compiles your sketches into executable code, then downloads them to the Arduino device over the cable. As your sketch runs, you can pass data between the IDE and your Arduino device over a serial communication channel enabled in the IDE (shown in **Figure 1**). Once compiled code is deployed to the device, the device resets and, once the device completes initialisation, it executes the sketch.

An Arduino sketch consists, at a minimum, of two parts: code that runs once, and code that runs repeatedly. Let us show you.

In the Arduino IDE (described later), an empty Arduino sketch looks like this:

SERIAL COMMUNICATION

The serial communications capabilities of the Arduino platform expose additional capabilities for your sketches. At a minimum, you can use serial communication to send data back to the IDE while you're troubleshooting your sketches. To do this, open the IDE's Tools menu and select Serial Monitor. A new window opens, and any data written using the Serial commands (described at **arduino.cc/en/Reference/Serial**) will appear in the monitor window.

You can also use serial communications to transfer collected data (from sensors connected to the Arduino board) to another computer system like a Windows PC or a Raspberry Pi. Makers often do this since the Arduino supports analogue inputs and the Raspberry Pi does not. In this scenario, the Arduino becomes merely a data collection device, and the Raspberry Pi does whatever number crunching is appropriate for the project, potentially even displaying data on a connected screen or uploading the data to a remote server for processing.

```
/*
*/
void setup() {
}
void loop() {
}
```

The first part of the sketch is a comment block. Anything, absolutely anything you enter between the **/*** and ***/** characters is ignored by the Arduino compiler.

```
/************************************
 My First Arduino Sketch

 by John M. Wargo
  December, 2017

Meatloaf meatball pork ground round fatback
kielbasa cow porchetta pork loin ball tip. Spare
ribs picanha drumstick pork jerky cupim alcatra
meatball beef ribs. Ball tip ground round
pastrami pancetta shank kevin.
************************************/
```

In your sketches, you'll use this commenting approach when you have multiple lines of content you want displayed within the sketch. At a minimum, use a comment block at the beginning of the sketch to describe the sketch, as we've done in the example, using dummy content from the Bacon Ipsum generator (at **baconipsum.com**). You should also use block comments like this to describe important parts of your sketches.

> **Arduino is a small, inexpensive, programmable microcontroller** that exposes a multitude of input and output (I/O) connections

You can also add single-line comments to your sketches. To do this, start any line in your sketch with double forward slash characters (**//**) or after any of your code. All content following the double forward slashes is ignored by the Arduino compiler. In the following example, a single-line comment precedes the definition of the **numCols** variable. The comment and the executable code are on separate lines, so we started the comment line with the double forward slashes.

```
//Number of columns in the table
int numCols;
```

Or something like this where the comment follows the definition of the **relayStatus** variable:

```
bool relayStatus;  //The current status of
the relay (on/off)
```

The sketch's **setup** function is defined with the following code:

```
void setup() {
}
```

Any code you add to this function (you'll add your code between the curly braces **{}**) is executed by the Arduino device as soon as you power it up and the hardware finishes initialising. This function is executed only once; you'll use this function to set up your sketch and execute the things that only need to be done when the sketch starts.

You'll normally use this to define the configuration of your hardware; as many input/output (I/O) connectors on the Arduino can be used for either input or output, you'll have to tell your sketch how you intend to use them. We'll show you an example of this in a little while.

The final component of a minimalist sketch is the **loop** function:

```
void loop() {
}
```

In this function, put any code that you want to run repeatedly on the Arduino. The Arduino executes the **setup** function once, then executes the **loop** function over, and over, and over again until either the Arduino explodes (it won't, we're just kidding) or you disconnect power from the device. You can put all your code in the loop, or break your code into smaller functions and call those functions from the **loop** function.

To see all of this in action, look at the following example. By default, the Arduino developer tools include a simple sample sketch called Blink.

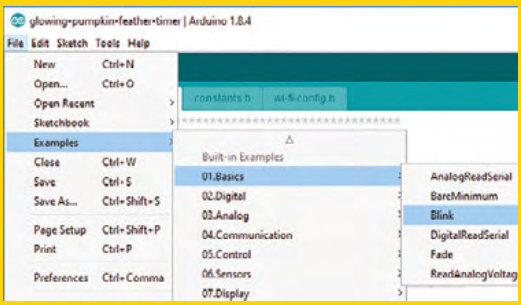
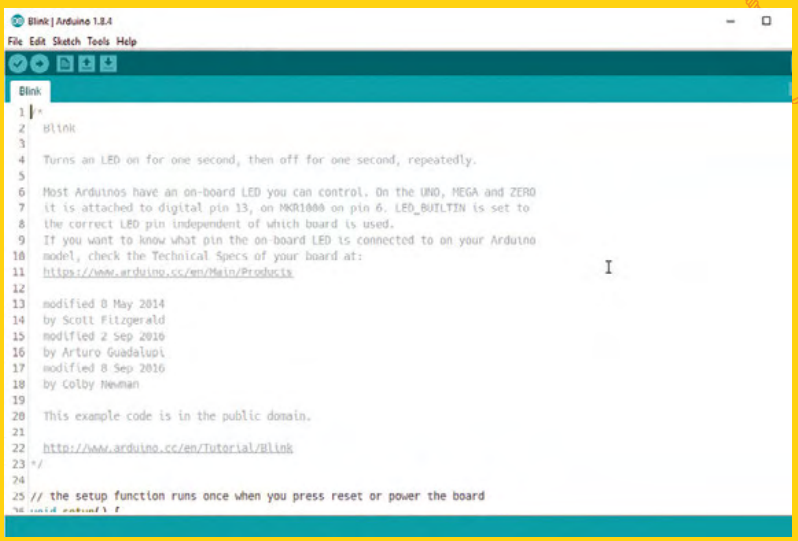


Figure 2 Opening the Arduino Blink sketch

YOU'LL NEED

- **An Arduino-compatible board** An actual Arduino device is preferred as there's extra setup required for many Arduino compatible boards. The recommended starter board is the Arduino Uno (hsmag.cc/QKaKXM) or the newer, and more capable, Arduino Zero (hsmag.cc/KGJbVd)
- **Microsoft Windows, Apple macOS, or Linux**
- **Universal serial bus (USB) cable** To connect the Arduino device to your computer system. Arduino on-device connectors vary; most use a micro-USB connector, but the Uno uses a USB A/B cable



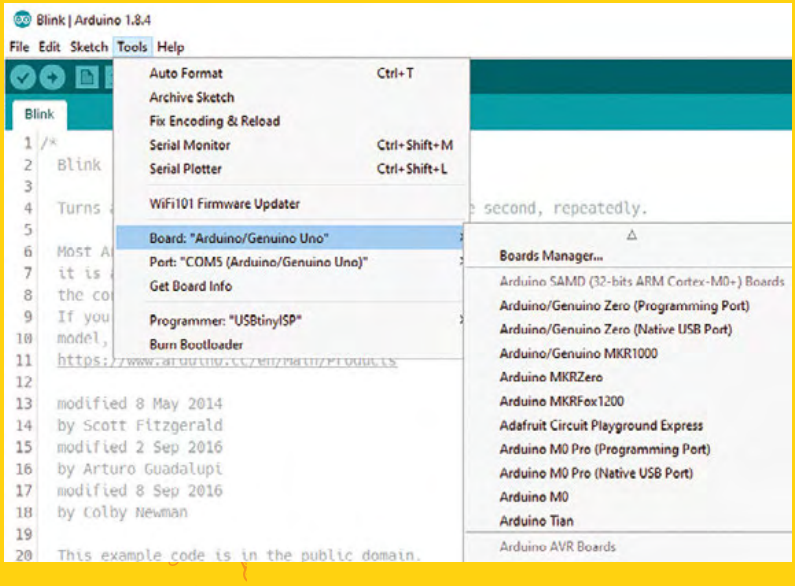
Above
The Arduino Blink sketch

Most Arduino devices include an LED on board, hard-wired into one of the Arduino's I/O ports. The included Blink sketch enables you to quickly accomplish something with the Arduino – turning that on-board LED on and off repeatedly.

Note: The Blink sketch starts with a long and thorough introductory comment block that we're omitting here for brevity's sake. We'll show you how to open the sketch soon, so you'll be able to study the whole sketch in detail.

```
// the setup function runs once when you press reset or
// power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

Below
Configuring the IDE for the connected Arduino board




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

In the **setup** function, there's only one executable line:

```
pinMode(LED_BUILTIN, OUTPUT);
```

Calling **pinMode** sets the configuration for one of the Arduino's I/O pins. In this case, its configuring the I/O pin defined in **LED_BUILTIN** for output mode. Remember, most Arduino boards have an on-board LED; the Arduino team has preconfigured the Arduino development environment to store the I/O pin associated with each Arduino board in a variable called **LED_BUILTIN**. Any time the sketch references **LED_BUILTIN**, the compiler replaces the reference with the actual pin number to which the LED is connected. The Arduino Zero has its LED wired to I/O pin 13, so for the Zero, the code is essentially:

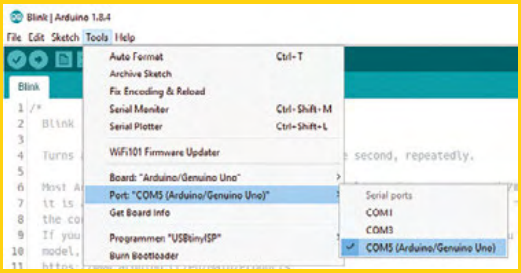
```
pinMode(13, OUTPUT);
```

With this in place, the sketch knows that when working with pin 13, it will be outputting (sending a voltage) to the pin, not receiving input.

In the `loop` function, the code completes the following steps:

- Uses the **digitalWrite** method to set the output voltage on the **LED_BUILTIN** pin to **HIGH**. This means that the pin gets a voltage equivalent to the current operating voltage of the Arduino. Some Arduino devices operate at 3V and others at 5V; all that's important to know here is that with execution of this code, the Arduino is now powering the LED connected to the I/O pin at its brightest.
- Waits for 1000 milliseconds (1 second) using the **delay()** method.
- Uses the **digitalWrite** method to set the output voltage on the **LED_BUILTIN** pin to LOW. This translates to no voltage (0), essentially turning the LED off.
- Waits for 1000 milliseconds (1 second) using the **delay()** method.

Below
Setting the IDE's communication port



Below
Compile and Deploy buttons

When the code runs, it will turn the LED on for 1 second, then off for 1 second, repeating the process until you remove power from the device or deploy a different sketch.

Now it's time to see the sketch in operation. To do this, you'll start by installing the Arduino IDE on your computer system. Open your browser of choice and navigate to **arduino.cc**. On the site's top menu, click the Software link, then, on the page that opens, download the latest version of the Arduino IDE for your system's operating system. Once the download completes, launch the downloaded file to begin the software installation.

> "To program an Arduino device, you'll code applications in a language similar to **an old language called C;** these applications are called sketches"

Once the installation completes, start the Arduino IDE. In the Arduino IDE, open the File menu, select Examples, then 01.basics, then Blink, as shown in **Figure 2** (page 83).

```
Archiving built core (caching) in: C:\
Users\JOHNWARGO\AppData\Local\Temp\arduino_
cache_950966\core\core_arduino_avr_uno_
c3bfe3f79ffbeab93536a1a484b588d9.a
Sketch uses 928 bytes (2%) of program storage
space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic
memory, leaving 2039 bytes for local variables.
Maximum is 2048 bytes.
```

If the verification fails, the IDE will display information about any errors and reference the sketch line number where the error was found. You'll need to fix any errors before continuing to the next step.

Finally, click the Upload button; the IDE will repeat the verification step, then deploy the compiled sketch to the connected Arduino device. When the upload process completes, the Arduino device will immediately reset, then begin executing the new sketch. In this example, the Arduino will turn its on-board LED on and off repeatedly until power is removed from the board or a different sketch is uploaded.

Now it's time to play around with the code. If you remember from earlier, the sketch uses delay statements to control the amount of time the LED is on and off. Right now, they're coded to pause 1 second (1000 milliseconds); modify the code so the LED stays on for half a second (500 milliseconds) and pauses for two seconds (2000 milliseconds) in between. Upload the modified code to the board and see what happens. □

NEXT STEPS

We've only lightly brushed the surface of what you can do with the Arduino platform. To make it easier for Arduino developers to get started, the IDE includes a whole catalogue of example applications you can study and use to expand your skills. To access these examples, in the Arduino IDE, open the File menu, select Examples, then look for a sketch category that appeals to you. The Basics category offers some simple sketches you can use to expand from where we've started here. There's a simple sketch to fade the on-board LED up and down (instead of turning it on and off, as in the Blink example). There are also sketches for reading analogue or digital signals; you'd use these with the appropriate analogue or digital output device connected to the Arduino. The other sketch categories offer more sophisticated sketches that work with different hardware devices and more.