Arduino Tutorial

In this tutorial, we will be talking about and getting familiar with the Arduino hardware and code!

What is Arduino?

Arduino describes itself as “an open-source electronics platform based on easy-to-use hardware and software.” In large part, this description is fitting, as the company designs and sells a collection of microcontroller development boards that simplify deployment of embedded hardware alongside a software framework that abstracts away all, but the most relevant considerations for your application. Perhaps, what’s under-represented by this description is the role that the surrounding community plays in enabling many plug-and-play experiences given the number and quality of auxiliary hardware modules, support libraries, and tutorials that have and continue to be produced within Arduino’s ecosystem.

One thing we’d like to acknowledge here is that Arduino’s mission of creating easy-to-use hardware and software has the necessary tradeoff of limiting the feature set of its development environment to the essentials.

What is an Integrated Development Environment (IDE)?

As is true for all forms of programming, an integrated development environment, or IDE, is an application with a feature set that facilitates software development generally or within a particular niche. The Arduino Desktop IDE is highly specific in the sense that it is intended to facilitate software development for a specific set of microcontroller boards, in C++.

Within the niche of IDEs for embedded software, there is a noticeable dichotomy between light-weight applications, (like Arduino’s IDE) that minimize functionality and abstract away details in the name of simplicity and full-featured IDEs (like Keil µVision and Visual Studio Code) that exist to support industry professionals.

If you’re interested in finding a happy medium to use in future embedded projects, we have had good experience using VS Code with the hardware specific extension PlatformIO, that together enable nice-to-have features like line completion, reference tracking, etcetera, without all of the complexity that more advanced IDEs introduce. Having said this, this particular combination of VS Code + PlatformIO will not be officially supported within this course.

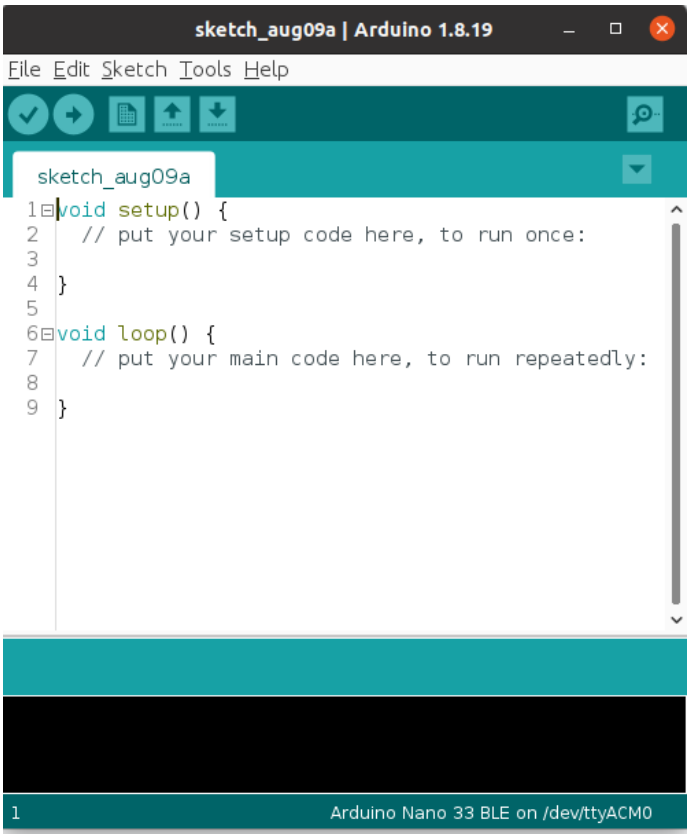
Download the OS specific Arduino IDE

What is the Arduino IDE?

As you might expect given the description above, the Arduino IDE is a light-weight development environment with features that permit you to very quickly manipulate microcontroller development boards. While there is a cloud-based offering (the Arduino Create Web Editor) as

well as a so-called Arduino Pro IDE that is in development (specifically in the alpha stage at the time of writing), we are going to use the standard Arduino Desktop IDE in this course.

A Quick Tour of the IDE



<-Menus and Toolbars

<-Coding Area

<- Console

IDE stands for *Integrated Development Environment*. The IDE is a text editor-like program that allows you to write Arduino code. When you save a file in Arduino, the file is called a *sketch* – a sketch is where you save the computer code you have written.

When you open the Arduino program, you are opening the IDE. Click on File and open a blank sketch. When you first open the IDE, you will see a screen that looks something like the above screenshot.

Up at the top of the IDE, you will find the menus and the toolbar, which we will explore as we work through the rest of this document and when we run the tests later in this section.

Below that, you will find the file tabs. For now, there will only be one file open that is most likely named sketch_<date>, however, later in the course when we work through more complicated examples, you will see all of the various files that make up the project across the file tab area. In “Arduino speak,” a sketch is a simple project/application.

In the middle of the screen, you will see the large code area. Each sketch can consist of many files and use many libraries, but at its core, each sketch is made of two main functions, “setup” and “loop”, which you can see pre-populated in the code area. We’ll explore what those functions are used for through the Blink example in a future video. For now, let's continue orienting ourselves with and setting up the IDE.

Finally, at the bottom of the screen, you’ll find the console. This is where you will see debug and error messages that result from compiling your C++ sketch and uploading it to your Arduino.

Coding in the Arduino Environment

The coding language that Arduino uses is very much like C++ (“see plus plus”), which is a common language in the world of computing. The code you learn to write for Arduino will be very similar to the code you write in any other computer language – all the basic concepts remain the same – it is just a matter of learning a new dialect should you pursue other programming languages.

The code you write should be “human readable” and will be organized for a human to follow. In the following assignments/applications you will be seeing a lot of code organized into different files. This preserves readability and you can trace functions to their parent/header files(.hpp). The IDE translates the code into machine-readable code to be executed by the Arduino. This process is called *compiling.*

If you have errors in your computer code, the compiler will display an error message at the bottom of the IDE and highlight the line of code that seems to be the issue.

Let’s look at some syntax,

A semicolon needs to follow every statement written in the Arduino programming language. For example,

int LEDpin = 9;

In the above statement, We declare an integer variable named ”LEDpin” (see what i did here, human readable) and set it to tell the microcontroller that it is connected to pin 9 (could be a digital or an analog pin).

Comments are what you use to annotate code. Good code should be commented on well.

//This is the pin on the Arduino that the LED is plugged into
int LEDpin = 9

Comments will be ignored by the compiler – so you can write whatever you like in them.

We use “//” for starting a one line comment and

/* The multi-line comment */

Now, let's talk about the two functions used in nearly EVERY Arduino program.

void setup ( )

The function, setup(), as the name implies, is used to set up the Arduino board. The Arduino executes all the code that is contained between the curly braces of setup() only once. Typical things that happen in setup() are setting the modes of pins, starting

void setup( ) { //the code between the curly braces is only run once for setup()

You might be wondering what void means before the function setup(). **Void** means that the function does not return information.

1. setup() only runs once.

2. setup() needs to be the first function in your Arduino sketch.

3. setup() must have opening and closing curly braces.

void loop( )

You have to love the Arduino developers because the function names are so telling. As the name implies, all the code between the curly braces in loop() is repeated over and over again – in a loop. The loop() function is where the body of your program will reside.

As with setup(), the function loop() does not return any values, therefore the word void precedes it.

void loop( ) { //whatever code you put here is executed over and over