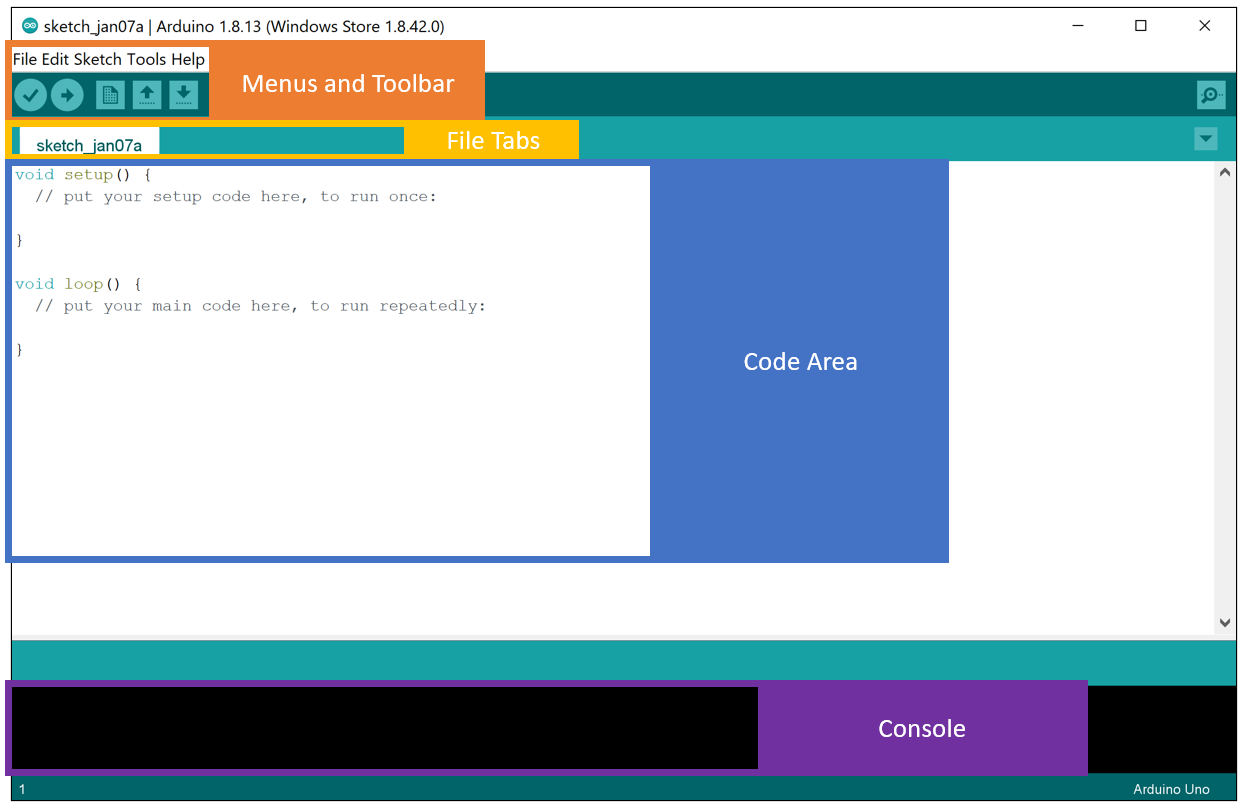
Setting up your Software 

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.

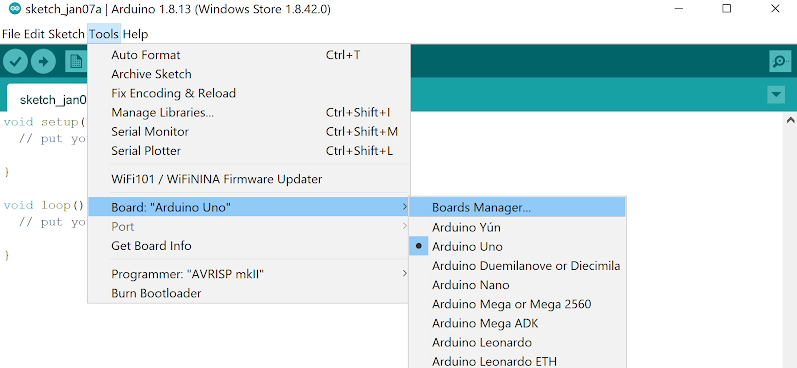
Finally, at the bottom of the screen, you’ll find the console. This is where you will see debug and error messages that result from compiling your C++ sketch and uploading it to your Arduino.

**Installing the Board Files for the Nano 33 BLE Sense**

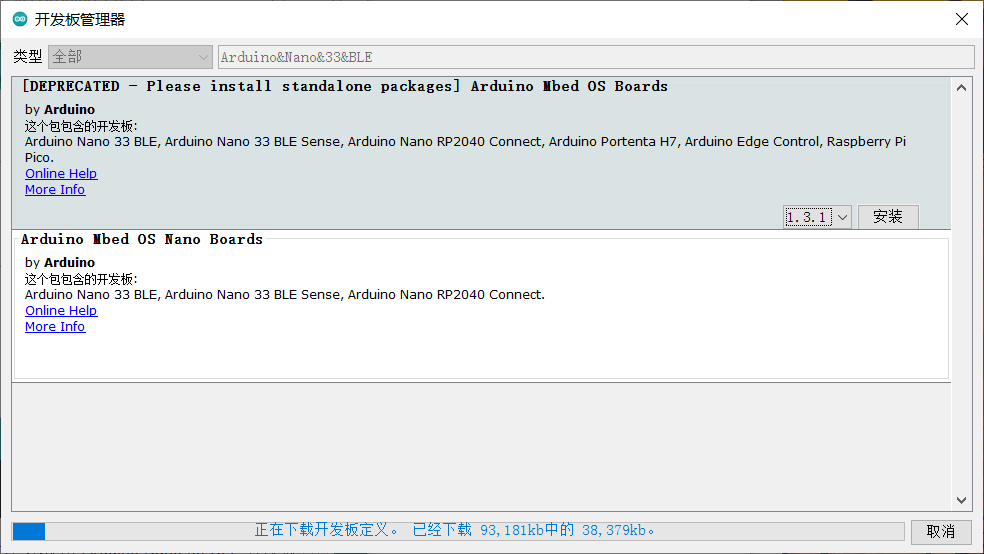
One of the primary advantages that the Arduino ecosystem affords is the portability of code you write for one or another board within their line-up or even in porting code to affiliate boards. This is made possible by the support files organized in the Boards Manager, which coordinates a download and installation of files that detail the Arduino functions (sometimes ‘core’) that are defined for that particular board (which is how hardware differences between boards are abstracted) as well as compiler or linker details specific to the given board.

To install the board files that you will need for your Arduino Nano 33 BLE Sense, please do the following:

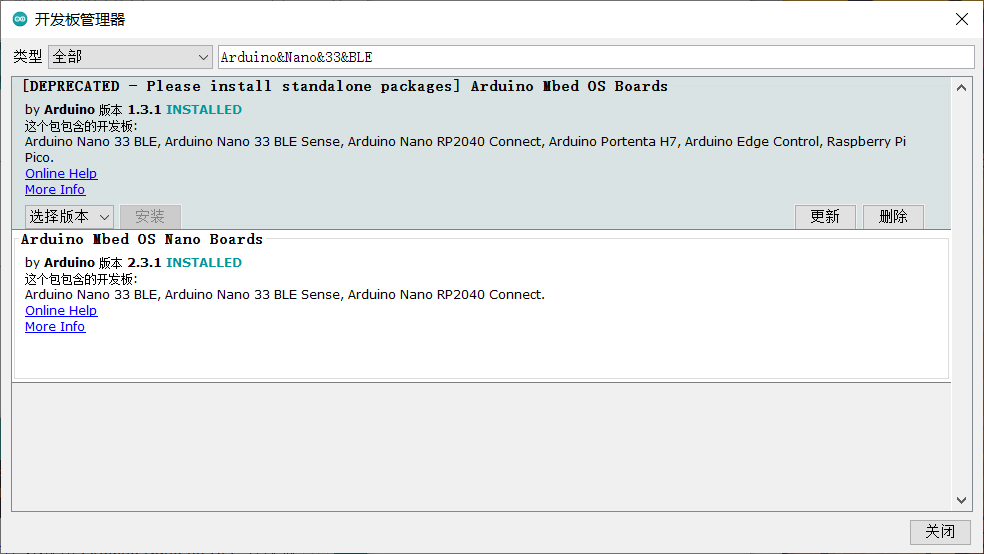
* + 1. Open the Boards Manager, which you can find via the Tools drop-down menu. Navigate, as follows: Tools → Board → Boards Manager. Note that the Board may be set to “Arduino UNO” by default.



* + 1. In the Boards Manager dialog box, use the search bar at the top right to search for “Nano 33 BLE,” which should bring up two results. We’re interested in the first result (as shown). Make sure **Version 1.3.1** is selected and then click “Install.” As the install process progresses, you will see a blue completion bar work its way across the bottom of the Board Manager window. Be patient, you may need to install USB drivers, which requires you to approve an administrator privileges popup which can take a couple of minutes to appear.



After you have successfully installed the board, if you exit and re-open the Board Manager and search again for “Nano 33 BLE,” you will now see a green INSTALLED next to the library and the option to “Remove” the library or install a different version.

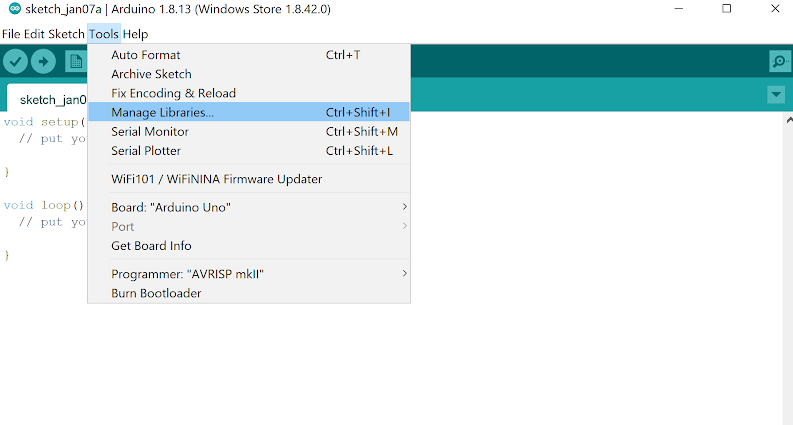


**Installing the Libraries Needed for this Course**

Another advantage of the Arduino ecosystem is the availability of a wide array of libraries for performing various tasks, such as interfacing with a sensor module or manipulating data using common algorithms. There are many libraries that can be accessed from within the Library Manager in the Arduino IDE as described below. Check [here](https://www.arduino.cc/reference/en/libraries/) for a complete list.

For this course, we are going to need four libraries. To install the libraries, please do the following and **make sure to install the version specified in the reading below or the tinyML applications will not work**:

* 1. Open the Library Manager, which you can find via the Tools drop-down menu. Navigate, as follows: Tools → Manage Libraries.

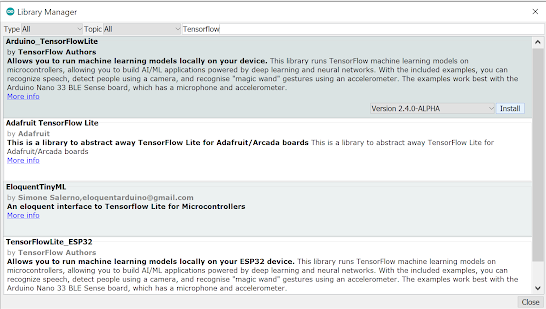


* 1. Then, much like for the Boards Manager, in the Library Manager dialog box, use the search bar at the top right to search for the following libraries, one at a time. Note that like with the Board manager, a blue completion bar will appear across the bottom of the Library Manager window.
     + - The Tensorflow Lite Micro Library:

Search Term: Tensorflow

Library Name: Arduino\_TensorFlowLite

Version: 2.4.0-ALPHA



* + - * The Harvard\_TinyMLx Library we put together for this course!

Search Term: TinyMLx

Library Name: Harvard\_TinyMLx

Version: 1.0.1

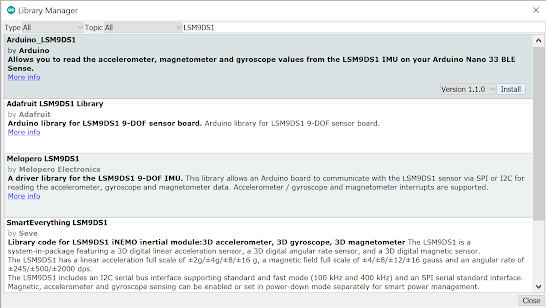


* + - * The library that supports the accelerometer, magnetometer, and gyroscope on the Nano 33 BLE sense:

Search Term: LSM9DS1

Library Name: Arduino\_LSM9DS1

Version: 1.1.0

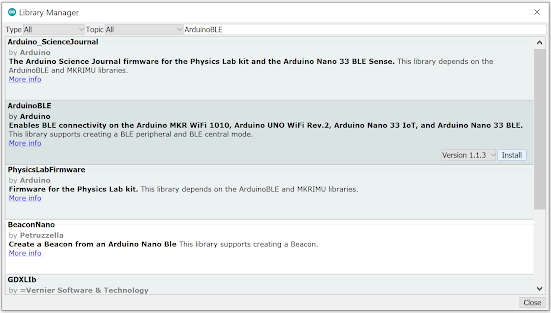


* + - * ArduinoBLE:

Search Term: ArduinoBLE

Library Name: ArduinoBLE

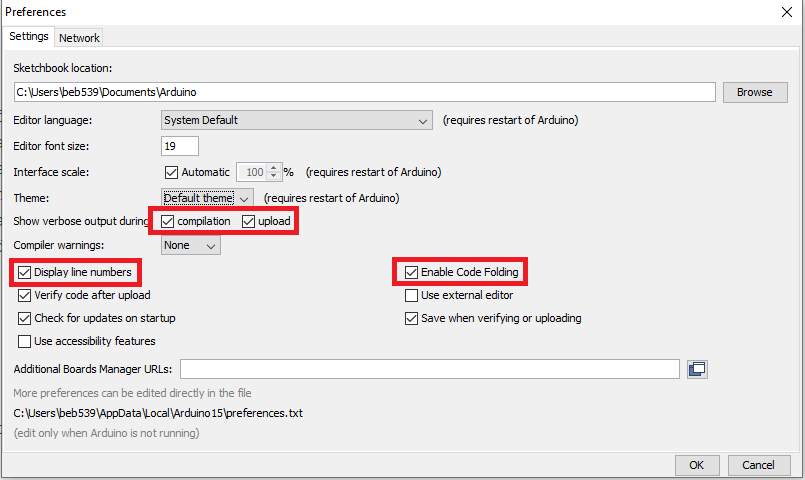
Version: 1.1.3



**Setting your Preferences**

You can adjust the preferences set for the Arduino Desktop IDE via the File drop-down menu, File → Preferences. There are a few preferences that we recommend enabling to make the Arduino IDE a little easier to use, namely:

* 1. Show verbose output during: compilation and upload.
  2. Enable code folding.
  3. Display line numbers.



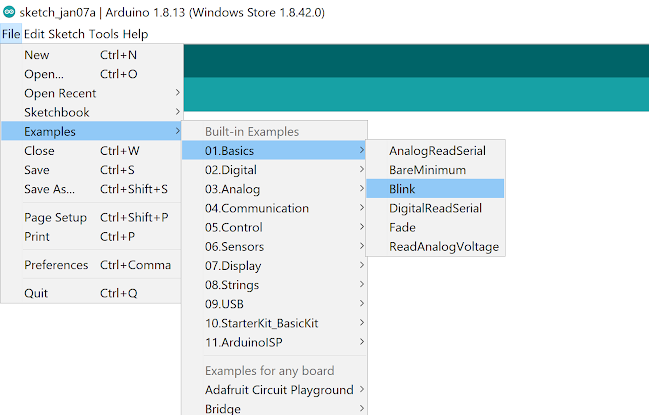
On final note, if you don’t like the default theme for the Arduino Desktop IDE, there is a nice tutorial for a [dark theme you can find here](https://create.arduino.cc/projecthub/konradhtc/one-dark-arduino-modern-dark-theme-for-arduino-ide-2fca81). Also, if you would like to learn more about the IDE, check out [Arduino's documentation](https://www.arduino.cc/en/guide/environment).

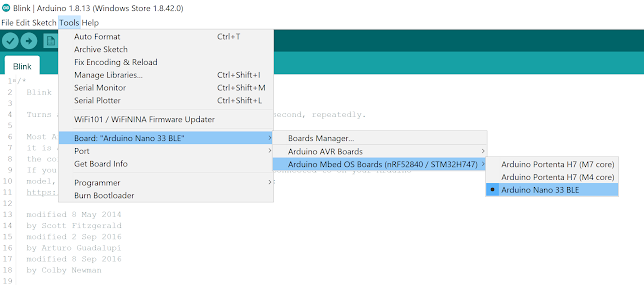
And that’s it! Your Arduino IDE should be all configured for this course. Now that you have all of the necessary board files and libraries installed, it’s time to explore more of the features of the IDE available under the “Tools” menu and start to test out your Arduino by deploying the Blink example!

The Arduino Blink Example

In this reading, we will deploy the Arduino Blink example to make sure everything is working properly and to give you your first experience deploying code to your Arduino!

**Preparing for Deployment**

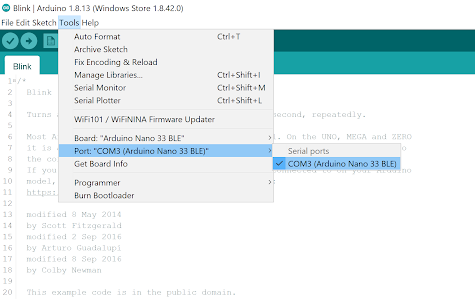
* + 1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see a green LED power indicator come on when the board first receives power.
    2. Open the Blink.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → 01.Basics → Blink. You’ll notice that Arduino has provided a wealth of examples to choose from should you like to explore the board more on your own outside of the course material. There is [great documentation about those examples on the Arduino website.](https://www.arduino.cc/en/Tutorial/BuiltInExamples)
    3. Use the Tools drop-down menu to select appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically, but if not, you’ll want to select:
       - * Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems, the exact name of the board may vary but/and, it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.



* + - * + Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate ‘Arduino Nano 33 BLE” in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.

Windows → COM<#>

macOS → /dev/cu.usbmodem<#>

Linux → ttyUSB<#> or ttyACM<#>  
  


* + 1. Finally, use the checkmark button at the top left of the UI to verify that the code within the example sketch is valid.   
         
       A screenshot of the Arduino IDE noting the Verify button.

Verification will compile the code, so take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled ‘verbose output during compilation’ in Preferences. You should most likely at the end see a final output indicating how much memory the sketch will take on the Arduino once it is uploaded. Something like: “Sketch uses 86568 bytes (8%) of program storage space. Maximum is 983040 bytes. Global variables use 44696 bytes (17%) of dynamic memory, leaving 217448 bytes for local variables. Maximum is 262144 bytes.” As you can see, this is a very simple example and does not take up much space. While, as you’ll see in a moment, uploading your code will also verify your code automatically, it is often helpful to verify your code first as you can iron out any compilation errors without having any hardware on hand.

**Deploying (Uploading) the Sketch**

Once we know that the code at hand is valid, we can ‘flash’ it to the MCU:

* + 1. Use the rightward arrow next to the ‘compile’ checkmark to upload / flash the code.

Note that pragmatically, this step will re-compile the sketch before flashing the code, so that in the future if you intend to sequentially compile and flash a program, you need to only press the ‘upload’ arrow.

A screenshot of the Arduino IDE noting the Upload button.

As before, take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled ‘verbose output during compilation’ in Preferences, accessible via the File drop-down menu in the IDE.

You’ll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like “Done in <#.#> seconds.” Again, if this is the first time you are uploading a sketch to an Arduino, the upload may hang for a little while until you get another administrator approval popup and approve it. Don’t worry, this is just a one time thing.

If you receive an error, you will see an orange error bar appear and a red error message in the console (as shown below). Don’t worry -- there are many common reasons this may have occurred. To help you debug, please check out [our FAQ appendix](https://github.com/tinyMLx/appendix/blob/main/ArduinoFAQ.md) with answers to the most common errors!

**Understanding the Code in the Blink Example**

Now that you have gotten the blink example deployed to your microcontroller, let's explore the code as shown below.

You’ll notice that it consists of two functions: setup, and loop. As we mentioned before, this is the standard setup for an Arduino sketch. This is because when the Arduino turns on it runs the setup() function ONCE to initialize (aka setup) the sketch. Then it runs the loop() function infinitely many times (aka it runs as an infinite loop) to execute the sketch. This works well for most tinyML applications as they are designed to respond to continuous sensor input. You can imagine in the case of Keyword spotting that we need to initialize the neural network and the microphone and then in a loop we want to listen to audio and trigger (or not) depending upon the output of the neural network!

// 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);

}

// the loop function runs over and over again forever

void loop(){

  digitalWrite(LED\_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)

  delay(1000);                     // wait for a second

  digitalWrite(LED\_BUILTIN, LOW);  // turn the LED off by making the voltage LOW

  delay(1000);                     // wait for a second

}

You’ll also notice that both functions have the void return type as they do not ever return anything but instead have side effects.

For example, in the setup function the LED\_BUILTIN (which is a shortcut name for the pin that controls the voltage to the LED) is set to be an output for the duration of the loop. In general, you will need to set all of the pins you use as either inputs or outputs during the setup function. If you wired up the camera yourself, you have already explored a lot of the special names reserved for the pins as shown on the [pinout diagram](https://content.arduino.cc/assets/Pinout-NANOsense_latest.pdf).

In the loop function, you’ll notice that we are alternating between writing a HIGH (aka turning on the LED) and writing a LOW (aka turning off the LED). The delay of 1000 milliseconds (1 second) between each step is crucial as otherwise the light would turn on and off so fast that it would be imperceptible. In fact, if you make the delay too short the light will simply seem to be dim. This is a trick called [Pulse Width Modulation](https://en.wikipedia.org/wiki/Pulse-width_modulation) that is actually used often in industry to e.g., control motors. If you’d like, feel free to experiment with modifying these delays and redeploying the code to see its effect!