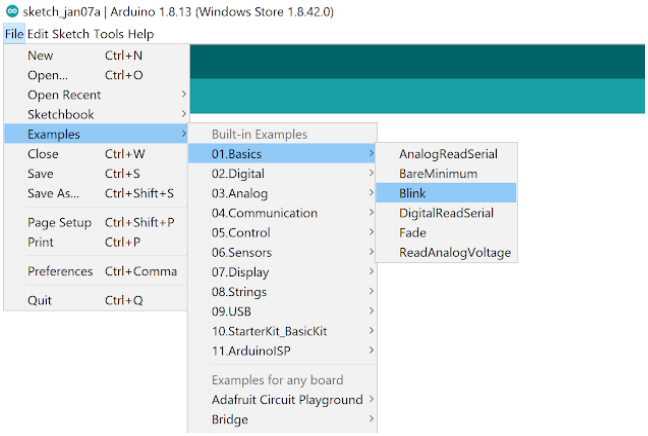
The Arduino Blink Example

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!

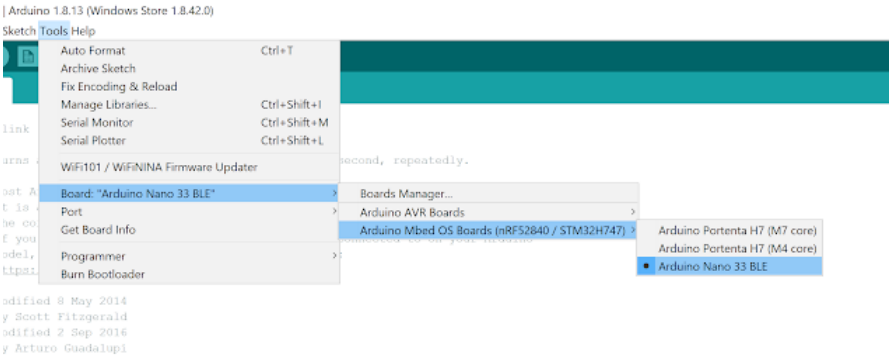
Preparing for Deployment

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.



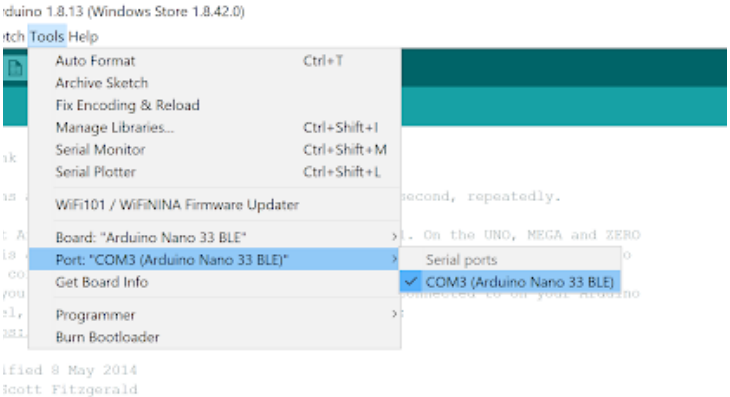
3. Use the Tools drop down menu to select appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically.

    a. 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.

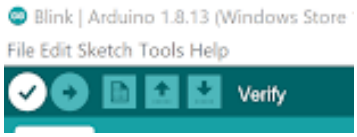


    b. Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate ‘Arduino Nano 33 BLE” in parenthesis. You can select this by going to `Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE).` Where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

        i. Windows → `COM<#>`
        ii. macOS → `/dev/cu.usbmodem<#>`
        iii. Linux → `ttyUSB<#>` or `ttyACM<#>`



4. Finally, use the checkmark button at the top left of the UI to verify that the code within the example sketch is valid.



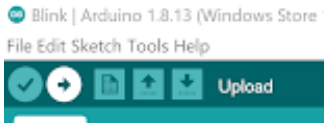
Verification will compile the code, so take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled 'verbose output during compilation' in Preferences. You should most likely at the end see 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'[1] 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 only press the 'upload' arrow.



   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.

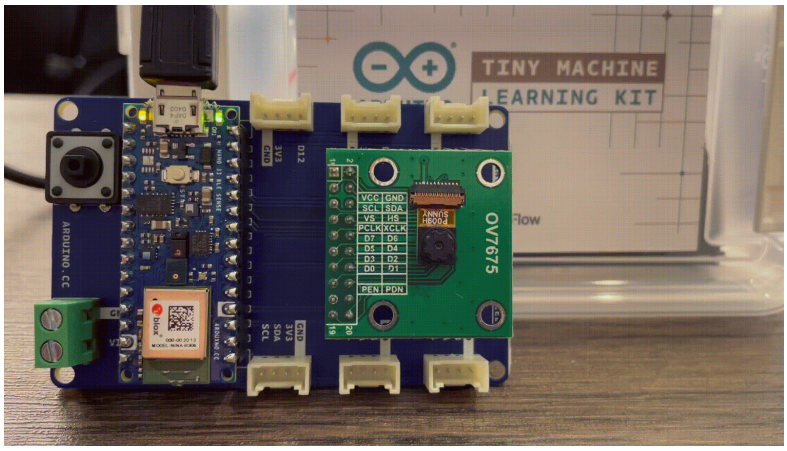
   You'll know the upload is complete when you red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

[1] This is a colloquial reference to the type of program memory MCUs call on: flash memory

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 with answers to the most common errors!

2. At this point, you'll want to look to the board itself. The orange LED opposite the green LED power indicator about the USB port should now be blinking!



As a final note if you'd like to learn more about how the process of flashing code to a microcontroller works please check out [this appendix document].

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.

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 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!