Introduction

During this activity, students will install the Arduino IDE and write some basic programs. This is meant to be a basic familiarization with the Arduino IDE. Students will need to use documentation from https://docs.arduino.cc/language-reference/ and libraries to complete future programming tasks.

Materials List

1. Computer with a USB Port
2. Arduino Mega 2560
3. USB A to USB B cable
4. USB C adapter (if needed)

Downloading and Installing the IDE

1. Installing the IDE (Steps as of Sept. 2025)
   1.1. Open your web browser of choice and go to https://www.arduino.cc/
   1.2. From the top menu, select **Products** and then **Arduino IDE**
   1.3. From the dropdown menu, select the version of the IDE that matches your OS and hit download as shown in Figure 1
   1.4. Once the installation file has completed downloading, run the installer and follow the prompts to install the IDE. If asked about installing drivers, you should select yes, these are base board files and drivers for using the serial port.
   1.5. After the installation completes, plug your Arduino in with your USB cable and open the Arduino IDE
   1.6. If you have issues installing, read the instructions at: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing/

Figure 1: Dropdown menu for selecting the Arduino IDE version.

Navigating the IDE

NOTE: Board and Library managers require an internet connection to download and install libraries. You will see a yellow offline message at the bottom of the IDE without an active connection.

1. Using the board manager
   1.1. The AVR Board files should have been installed as part of the base install.

   1.2. Open the Board Manager by clicking on the Arduino Icon below the folder on the left side of the screen.
   1.3. Arduino **AVR Boards** should be the top item on the list. If there is a version number followed by **installed** below that text and a **REMOVE** button, the board libraries are installed.
   1.4. If instead there is a green **INSTALL** button, click that and follow the prompts to install the board files.
2. Connecting your Arduino
   2.1. Look in the lower right corner of the IDE, you should see the text **Arduino Mega or Mega 2560 on COM#** (Windows) or **Arduino Mega or Mega 2560 on /dev/##** (Mac and Linux), indicating you are connected to the Arduino.
   2.2. If not, click the tools menu and then select Port and look for a port with an Arduino Listed and select that port.
   2.3. If you are still not able to connect, you can try unplugging and replugging the Arduino and restarting the IDE.
3. Uploading a blank sketch
   3.1. Once you have successfully connected to the Arduino, you should have a blank sketch (Arduino program) of just the setup() and loop() functions in the IDE window.
   3.2. You can always get a new blank sketch by selecting **New**

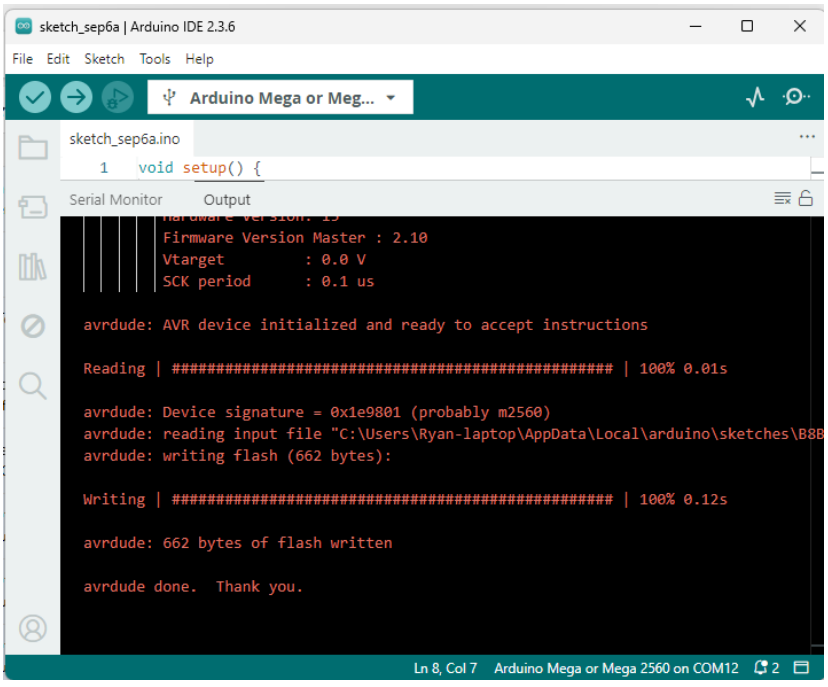

Figure 2: Upload message in the bottom portion of the IDE. Here, the size of the output should show more lines in the output window. If you see an error, verify that the Arduino and the COM port are selected in the dropdown.

Sketch from the file menu. Once code has been uploaded to the Arduino, it will run automatically when the microcontroller is powered. It is often useful to upload a blank sketch to ensure no other code is running on the microcontroller.

3.3. Click the upload button, and you should see messages for a successful compile and upload at the bottom of the IDE window. See Figure 2 for example.

4. Installing a library

4.1. Open the Arduino Library manager by clicking on the Book Icons on the left side of the IDE

4.2. Type **adafruit gps** in the search text box. The Adafruit GPS Library should be the top search result. Click install.

4.3. You will see a message about the GPS Library having a dependence on the SD library. Select **INSTALL ALL**. This will install both the GPS library and the SD library. (We will use these libraries later when we start working with the GPS shield).

4.4. You should now see those two libraries as options at the bottom of the **Include Libraries** option under **Sketch**. You should also now have entries for the libraries under **Examples** in the **File** menu.

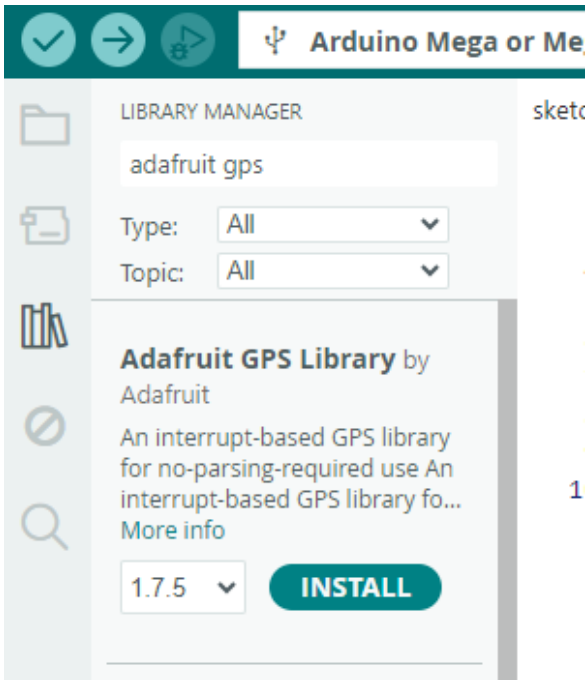

Figure 3: Adafruit GPS library inside the Arduino Library.

5. Finding your sketchbook folder.

5.1. Open the **Preferences** option under the **File** menu. Look at the text box labelled **Sketchbook Location**. This is the folder on your computer where your Arduino programs and libraries are stored.

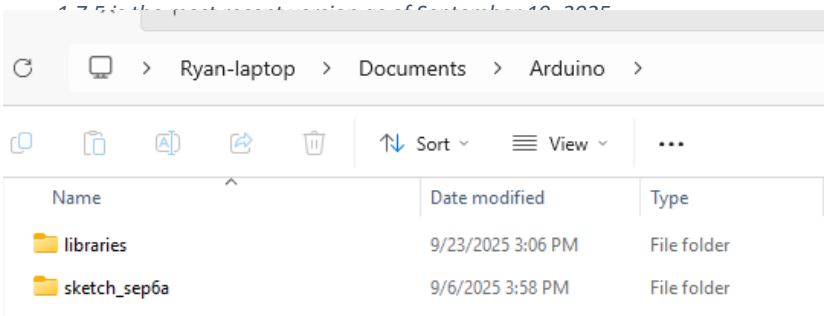

Figure 4: Default sketchbook folder location on a Windows install.

5.2. Using File Explorer(Windows) or Finder (Mac) navigate to the Sketchbook folder.

5.3. You should see a folder called libraries, open it, and you will see a folder for the Adafruit GPS Library and SD. When we download a library, a folder is created here, and the library code is copied here.

5.4. Go back to the sketchbook folder, and you should see a folder for each sketch you have created. Whenever you open a new sketch, a folder will be created. The actual sketch is an INO file inside the folder. You can change the name of your sketch by using the **Save AS** option in the file menu.

5.5. Try to ensure you do not create additional files inside the sketch folders as that can sometimes cause errors.

Using the Serial Monitor

1. Setting up Serial Communication

1.1. Now return to the blank sketch.

1.2. Save the File as Activity0303. It is recommended that for the remaining activities, you name your files based on the Activity number. You may also want to use a date or version number for the more complex files.

*Figure 5: A simple program to print the text **Hello World** to the serial port*

1.3. Change your sketch to match the code in Figure 5. Serial is a variable that is part of the built-in libraries. More complex variable types, like Serial, have functions that are written to be called on the variable. That is what is happening with the begin() and print() functions.
The documentation for this can be found here: https://docs.arduino.cc/language-reference/en/functions/communication/serial/. It is a good idea to look over the documentation for a function when you first encounter it.
The 9600 argument in Serial.begin() sets the baud rate or speed of the serial port; the Serial monitor will need to match the setting.
The print() function prints characters to the serial port.

1.4. Upload the code and open the serial monitor (Magnifying glass in the upper right corner). You should see the **Hello World** printed out over and over again in the Serial Monitor. If you see a series of squares, just scroll to the right; that is just the Serial Monitor trying to display the bytes transmitted when the code was uploaded.

2. **println()** and **delay()**

2.1. Now, let's try to make the output easier to read. A giant horizontal line of text is not really useful. Change the print() function to println().
NOTE: in println that is a lowercase L (ln for line)
This print line function automatically adds the New Line and Carriage Return characters to the serial output, starting a new line after the printout.

2.2. Upload your code and open the serial monitor. You should now see it printing **Hello World** over and over vertically and automatically scrolling.

2.3. The Arduino is looping over the Serial.println() as fast as possible, much faster than is useful to us.
To slow this down, use the delay(). This will pause the program for the number of milliseconds given in the argument.
Add the line delay(1000); after the println and upload the code.

Figure 7: Serial monitor output with the incorrect Baud rate setting.

3. Changing Baud Rate

3.1. Now try changing the baud rate in the serial monitor to 19200 using the dropdown menu on the right of the serial monitor.

3.2. The output from the serial monitor should change to a bunch of strange symbols. This is because the baud rate on the Arduino (still 9600) and Serial Monitor (now 19200) no longer match.

3.3. Modify the Serial begin line to match the Serial monitor setting by changing the 9600 to 19200.

3.4. Upload the code, you should see the output return to normal,

4. Other Serial Ports

4.1. Change **Serial** to **Serial1** on both the Serial.begin line and the Serial.println line. Upload the code.

4.2. You will notice that code uploads correctly, but the output stops. The Arduino Mega has 4 Serial ports: **Serial**, **Serial1**, **Serial2**, **Serial3** but only the first is connected the USB.

4.3. If you try to use a different number, you will instead get an error because those serial ports do not exist and are not defined in the library.

4.4. Change your serial port back to Serial

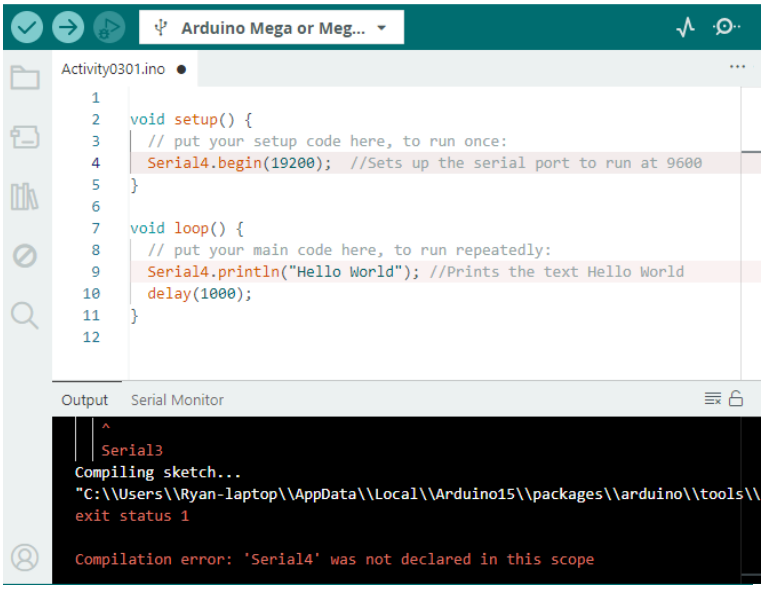

Figure 8: Error when attempting to a serial port that does not exist.

Basic Programs

5. Working with variables

5.1. Now declare an integer variable at the top of the sketch and set it equal to 0.

5.2. Replace **Hello World** with the variable and upload the code.

5.3. You will now see 0 printed out repeatedly since the value of myVar is never changing.

5.4. Add the line:
myVar = myVar + 1;
after the delay() and upload the code.

Figure 9: Printing the variable myVar out in the loop.

5.5. You will now see the number being displayed counting up.

5.6. Now move the variable declaration inside of the loop. After uploading the code what happens?
You will see the code goes back to printing out zeroes. Why?

Remember that when the loop finishes, it restarts from the beginning. This means the variable is reinitialized; in fact, the same thing will occur even if you remove the assignment by changing **int myVar = 0;** to **int myVar;**

5.7. Now try moving the declaration of myVar into setup(). When you try to compile the code, you will get an error. When a variable is declared inside a function, it can only be accessed inside that function.

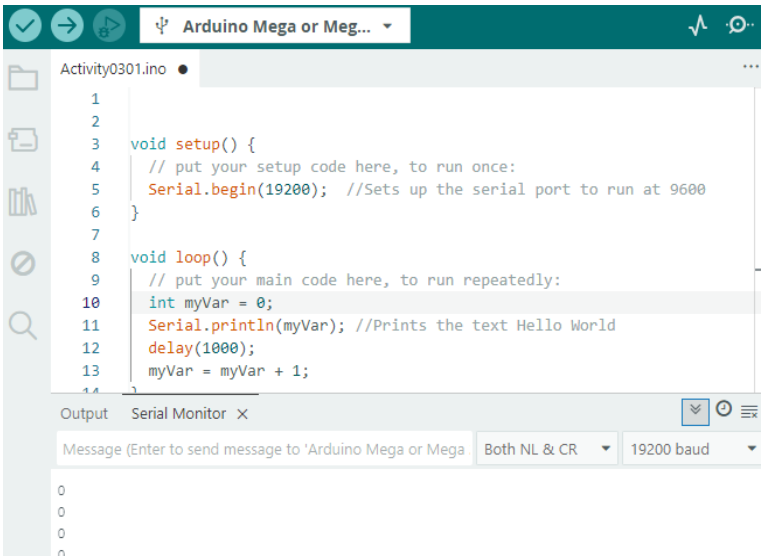

Figure 10: When myVar is declared inside the loop, it is reset to 0 at the top of every loop, so the printed out number never increases.

If we want to use a variable in multiple functions, we can declare it outside of all functions; this makes it a global variable. But as a program grows in size and complexity, you can get unintended behavior since it will be harder to keep track of everywhere the variable is used or modified.

You can pass the variable between functions using arguments and returns.

6. If Statements

6.1. Move your declaration back to the top of the program to make it global.

6.2. Now let's use if statements to print out if myVar is even or odd as it counts up

6.3. We can use the **%** operator for this since it gives the remainder of division. For odd numbers, **myVar % 2** will be 1, and 0 for even numbers.

Figure 11: Code now modified to print out even or odd depending on the variable value. Notice how we can use a print() followed by a println() to print out on a single line. Also, notice that if lines do not have semicolons.

6.4. Using two **if** statements, we can print out the text for even or odd. Don't forget to use braces **{}** to surround the code you want to be executed when the If is true.

7. Loops
   7.1. Now let's write a simple loop.
   7.2. A **for** loop combines a variable declaration, a test condition, and a variable increment
   7.3. Delete your myVar declaration from the top of the program.
   7.4. Change your loop function to have a for loop using myVar to count from 0 to 4.
   7.5. The **++** operator combines the a **+1** operation and a **=** operation.
   7.6. Notice how the count repeats, the loop() function is still being repeated, so every time it restarts the for loop starts from its initial conditions.

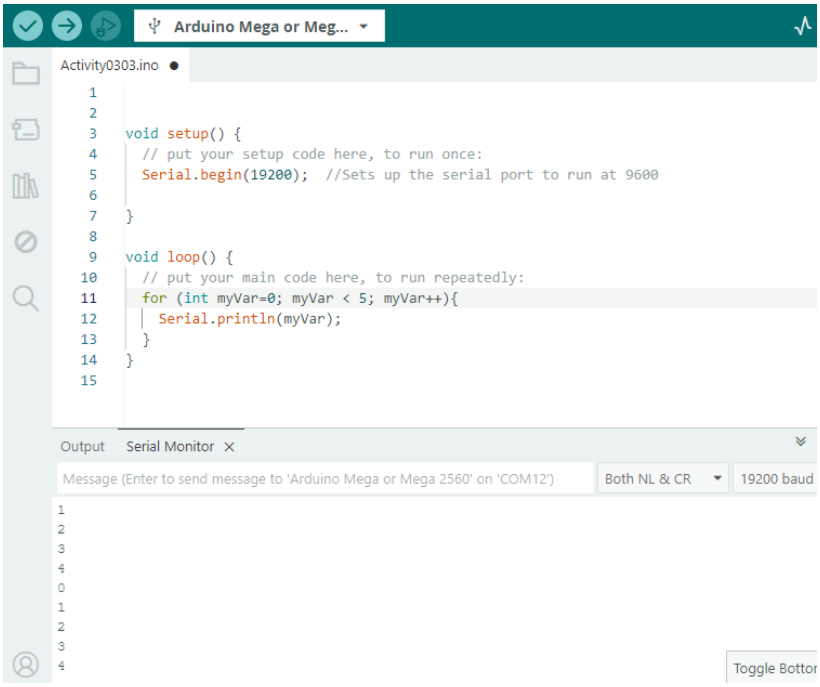

Figure 12: Code for a simple for loop. Nice how the output starts at 0 and does not print out 5. Also notice the parts of the for loop are separated by semicolons and not commas (commas would be used to separate function arguments).