The Arduino IDE

What is an IDE?

An **IDE** is a TLA[1] for Integrated Development Environment. In order to develop software for a system the programmer needs to use several tools and follow a variety of steps. The IDE can assists the programmer by simplifying or automating many of these processes.

Although there are many, many functions an IDE may perform, the most critical are:

- Editing your program.
- Compiling your program.
- Loading the resulting binary.

Editing the program is obvious, and that is what we are going to begin with in this experiment.

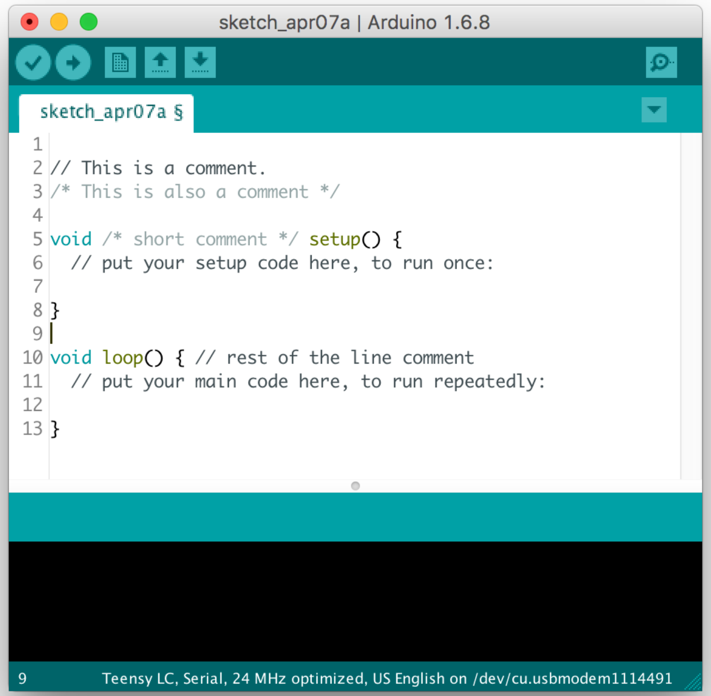
Compiling a program consists of converting the higher level programming language you are working in down to the actual machine language that the target processor understands. This may actually require the use of several tools and many steps. Creating and optimizing compilers in an entire realm of computer science.

Once the compilation phase has completed it will generate an output file. This file is commonly called a 'binary', to indicate that it is ultimately just a large sequence of binary numbers (0s & 1s) that the processor will act upon, and not easily human-readable. In order to get the processor to start using this 'binary', some additional work needs to be done with yet more tools.

Depending on the computer systems in use, it may be as simple as asking the currently running operating system to run your binary. This would be the case if we were developing an application for the Mac OS X system where we are running the Arduino IDE — however, in our case we need to run our binary on the microcontroller, and this requires that we must transfer the binary to the microcontroller, then instruct the micro controller to run our code. There are many tools and steps required for this, but fortunately the Arduino IDE takes care of all of that for us, and we simply need to tell it what type of microcontroller we have and then click an icon.

For this experiment, we just want to launch the Arduino IDE, load an existing program into the editor, configure some options to let it know what sort of micro controller we are using, then ask it to compile the program. After that we'll try to make a few simple changes, and verify that the program still compiles okay.

In programming languages, a ***comment*** is some text embedded in the program which is not part of the instructions for the system, but is simply some plain old prose with some information for other programmers to read telling them something valuable about the program at hand — a commentary. Because programming languages need to be compact, precise and easily converted into machine language, it is not always easy to look at a snippet of the program and determine exactly what is intended and why. This is where comments can play a key role. If you want another programmer (or yourself weeks, months, or years later) to re-use some or all of your program, adding comments in strategic areas will be key.



```

// This is a comment.
/* This is also a comment */

void /* short comment */ setup() {
  // put your setup code here, to run once:

}

void loop() { // rest of the line comment
  // put your main code here, to run repeatedly:

}
```

9 Teensy LC, Serial, 24 MHz optimized, US English on /dev/cu.usbmodem1114491

The changes that we will make will simply be to the content of the initial comments. The programming language the Arduino environment uses is called ***C++***, and in ***C++*** there are two ways to indicate that a section of text is a comment. Anytime the compiler sees a double slash, ***//***, it interprets everything after that until a line break as a comment. If the compiler sees a slash followed by as asterisk, ***/**** , it will consider everything after that a comment until it see it's inverse, ****/***, which closes the comment — this may be only one word, or it may span several lines.

We will look more at how the compiler interprets, or ***parses***, the contents of a program in the next few lessons.

[1] Three Letter Acronym, very popular in the tech world.

Components	**Wiring Diagram**
✓Computer ✓Arduino IDE	No wiring needed

Connection Instructions
No connections required

Sketch(es)	firstProgram.ino

Analysis Questions
Were you able to determine what the program will do by looking at the commands and comments? Did the comments help?

Programming Tasks
Open the Arduino app from your dock. Generally familiarize yourself with the various menus and icons, most have ‘tooltips’ available if you hover your mouse over them. Open firstProgram.ino from File->Examples->dataseamClass (under Examples from Custom Libraries.) Select the board type from the Tools pull-down menu from bar at the top of the screen. *(The board type will be **Arduino Nano** unless otherwise specified in class)* *Select the Processor type from the same Tools pull-down menu* *(The processor will be **ATmega328P** unless otherwise specified in class)* Compile the program by clicking the Verify (Checkmark) button and validate that there were no errors. Edit the information in the initial comment section of the code with the appropriate information. Assure that the program still compiles. Save your changed program — note that you will need to save it somewhere local to your account.