Programming Language Basics

The processor in a modern computer can perform many millions of operations per second. Although as was discussed, the processor can only perform operations on numbers, specifically binary numbers. These numerical operations, when properly combined, result in everything from web pages, to video games, to Artificial Intelligence.

Even though the processors are great with numeric operations, humans generally are not. We are better with more abstract concepts. That's where programming languages come into the picture. They provide enough structure that they can be (fairly) easily converted into the series of numbers that the processor needs, yet are simple enough for us humans to work with.

Dennis Ritchie & Ken Thompson, creators of C and UNIX.

Over time, many levels and types of languages were developed. Modern programming languages are often called *high level* programming languages because most of the details of how the processors, memory, and other actual hardware in the computer need to be accessed, are separated by many levels of abstraction, i.e. the programmer doesn't need to know exactly what it takes to actually store the number 12, they just do it.
We are going to consider programming languages which are text based. While there are some graphical programming languages, and even some aural and gestural languages, they are not yet common place, and are typically reserved for simple tasks.
A tool is required to convert this high-level program into the machine language required. This tool is called a ***compiler***. There are actually many parts involved with specific names and functions, but the generic term, compiler, is typically used to represent the entire process. Of course, this tool needs to know what the processor expects, and how to interpret the language.

An additional benefit of using a high level programming language is portability. Each type of processor, either built by a different company or just from a different era, expects a different series of numbers to tell it what to do — in one processor instruction #16284 might mean 'add the next two numbers together', in another, it might mean 'go back to the start'. But if we write our programs in an abstract language, then we just need to change the part of the compiler which generates that final series of numbers, sometimes called ***machine language***, and now all of our programs can run on a different processor.

In order for the compiler to be able to understand the programming language, the language needs to have some very strict and specific rules. Just as in spoken languages these rules collectively go by the term ***Grammar***. In a spoken language, grammatical errors are considered impolite and/or potentially confusing, for example: "Let's eat Grandma" vs "Let's eat, Grandma". But in a programming language not following the grammar to the letter is usually catastrophic — the compiler cannot convert your program into the required machine language.

The compiler needs to look at your entire program and begin mapping items in the grammar that you have expressed into patterns which it can eventually break down to the specific machine language appropriate for the target system. Your program is just a bunch of letters and numbers — which the computer sees really as just numbers — so the compiler needs to look through these numbers and look for particular patterns. This is called ***parsing***.

The rules combined with the specific set of symbols and keywords that should be used make up the ***syntax*** of the language. And if your program doesn't follow that convention, when you try to compile it you will receive a ***syntax error***.

Syntax errors are commonplace. They can be a result of a simple typographical error, or a mis-understanding (or lack of understanding) of the details and specifics of the syntax. Depending on the compiler and IDE in use — as well as the nature of the error itself — it may be immediately obvious what is wrong and how to correct it. Or not. Syntax errors can be the bane of first time programmers. It is expected you will spend a fair amount of time finding and correcting syntax errors as a new programmer. Don't worry, after a short time you will have it mastered, and our Arduino IDE will help you find and correct many, if not most, of them easily.

We will be using the Arduino environment, which is built upon the programming language C++, itself a super-set of the C Programming language.

Components	**Wiring Diagram**
✓Computer ✓Arduino IDE	No wiring needed
Connection Instructions	
No connections required	
Sketch(es)	needsWork.ino
Analysis Questions	
Programming Tasks	

- Open needsWork.ino from File->Examples->dataseamClass (under Examples from Custom Libraries)
- Be sure that the board & processor type are selected correctly
- Compile the program (Verify) and validate that there were no errors.
- Fix any compilation errors
- Once it compiles, as before, edit the information in the comments appropriately.
- Assure that the program still compiles.
- Save your changed program — *note: you will need to specify where to save it (probably somewhere local to your account)*