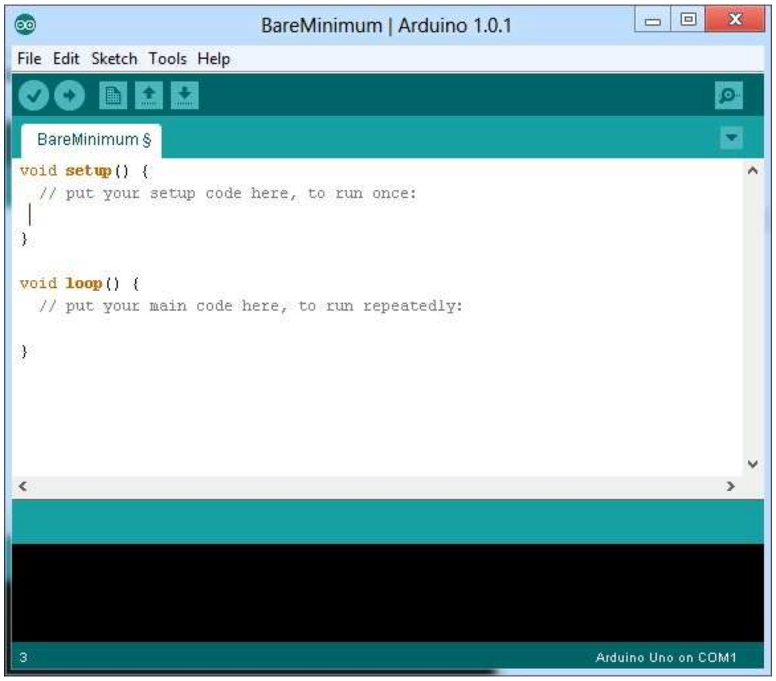
Variables and Functions in Arduino

Programming in Arduino's C language is very different than the majority of programming that you have done so far in the engineering curriculum, which heavily utilizes Matlab. This may create a steep learning curve, but once you gain a hold of the syntax, programming the Arduino will not be that difficult. One of the biggest differences between C and Matlab is how functions and variables are declared and used. This document will highlight the differences in program structure, functions, and variables to help get you started.

Basic Arduino Set-Up

A *sketch* is the name Arduino uses for programs. It is the unit of code that is uploaded to and run on an Arduino board. All sketches **must** have 2 void type functions, `setup()` and `loop()`. "Void" simply means that the function doesn't return any value.

The `setup()` method is run once just after the Arduino is powered on, and the `loop()` is run continuously afterwards. The `setup()` is where you want to do any initialization steps, and in `loop()`, you want to run the code you want to run over and over again. Your basic sketch should look something like this:



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 1: Bare minimum code required for an Arduino sketch

Variables

A variable is a place to store a piece of date. It has a name, a type, and a value.

Name: You can name a variable any word that is not already a keywords in Arduino. You should also avoid beginning variable names with numeral characters

Type: Unlike Matlab, you must declare all variable types before using them when coding in Arduino. In choosing a variable type, it is important to consider the size of the numbers you want to store. Variables will roll on when the value stored exceeds the space assigned to store it. Here are some common variable types:

Name	**Description**	**Range of Values**	**Size**
`int`	Integer, primary data-type for number storage	-32,768 – 32,767	2 bytes
`long`	Extended size variable for number storage	-2,147,483,648 – 2,147,483,647	4 bytes
`float`	Floating point numbers, used for numbers that have a decimal point. Floating numbers allow for 6-7 points of data precision. However, math with float variable is significantly slower.	-3.402835E38 – 3.402835E38	4 bytes
`double`	Double precision floating number. These take up 4-bytes on the Arduino UNO - the same as float variables - so even with double implementation, there is no gain in precision.	-3.402835E38 – 3.402835E38	4 bytes
`byte`	Stores an 8-bit unsigned number	0 – 255	1 byte
`boolean`	Holds one of two values, true or false	true/false	1 byte

Variables can also be `unsigned`. This is helpful when dealing solely with positive numbers, as it can essentially double the range of useful values. For example `unsigned int` can hold values from 0 to 65,535.

When declaring variables, the location of variable declaration matters, and variables can be *global* or *local*.

A global variable is declared outside of a function (e.g. `setup()`, `loop()`, etc.), and can be seen by every function in the program.

A local variable is declared within a function and can only be seen by that function. Local variables are useful when programs start to become larger and complex. They insure that only one function has access to its own variables and prevents programming errors when one function may accidentally modify same-name variables used by another function.

Value: When declaring a variable, setting an initial value is *optional*. For example:

```
unsigned int Variable1;
unsigned int Variable2=0;
```

Both are correct ways to declare variables. It is often to useful to initialize variables, but it will be dependent on the information that the variable is actually storing.

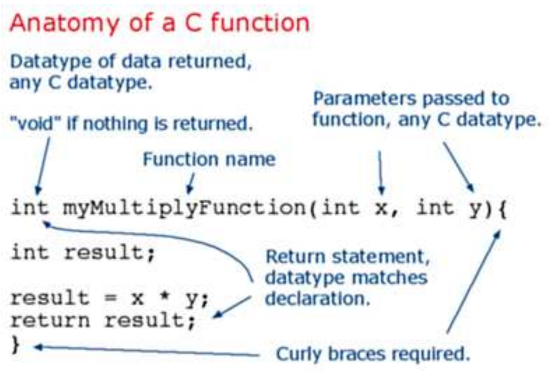
Functions

Creating Modular Code

When coding in Arduino, it is best to use functions and make your code as modular as possible, for a number of reasons:

- Helps the keep the program organize and often helps to conceptualize the program
- Makes the code more readable and easier to debug by codifying the one action in one place
- Makes the whole sketch smaller when sections of code are reused many times
- Makes it easier to reuse code in other programs

As you know, Arduino sketches require 2 functions: `setup()` and `loop()`. All other functions must be created outside the brackets of those two functions. In making your own function, it is helpful to designate a **New Tab** in your Arduino sketch to the function. The Arduino website provides a helpful diagram of the basic structure of functions:



```
int myMultiplyFunction(int x, int y){
int result;
result = x * y;
return result;
}
```

Anatomy of a C function
Datatype of data returned, any C datatype.
"void" if nothing is returned.
Function name
Parameters passed to function, any C datatype.
Return statement, datatype matches declaration.
Curly braces required.

Figure 2: Anatomy of a C Function.

To ”call” on this simple multiplication function, we pass it the parameters of interest.

```
    ans = myMultiplyFunction(4,6); // ans now contains the value 24
```