Programming For Arduino

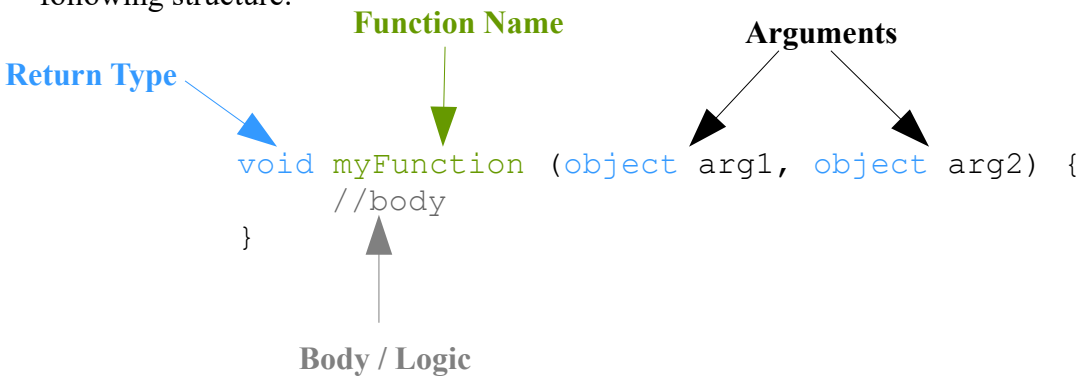
Part 1 - Functions

Lets start our tutorial on Arduino programming by looking at the most trivial Arduino program. From the Arduino IDE, just select **file-->new** to see a bare minimum sketch.

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

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

There are two main parts to any Arduino sketch: setup and loop. Each of these blocks of code is a programming construct known as a **function**. A function in a C-syntax language like Arduino has the following structure:



Return Type – The type of object that the function outputs. This could be something like a number or a string of text. It's very common for the return type to be 'void' which means this function doesn't actually output anything, and that's perfectly valid.

Function Name – This is just a word that describes what the function does. It can be pretty much anything you choose, but its best to try and pick a name that succintly describes what the function actually *does*.

Arguments – These are the inputs to the function. There can be any number of them, including zero. In the event there are no inputs, you will just see the open and close parens ( ) with nothing inside, and this is a very common case.

Body/Logic - This is the code that actually implements logic to *do* something. You'll notice that the body of the function is wrapper inside curly braces { } and this is important to mark the start and end of the function.

Lets look at the setup function from the blank sketch we loaded. First, we see that the return type is void, meaning this function doesn't return any data. Next, we see that the function is named setup. Next, we see that the function does not take any input arguments. And finally we see that the fuction's body is empty. It does contain a comment (a bit of text with the // symbol in front of it); but this is ignored by the compiler.

The function "setup" is aptly named, because (as the comment indicates) whatever code we write into the fucntion will be executed just once when the Arduino first starts up. By contrast, the second function called loop, which again return void, has no arguments, and has an empty body, get run continuously over and over while the Arduino is running.

Part 2 - Variables

Now we're going to do something new: declare a variable. A variable is a contruct where we can save some value to memory and recall it later. The declaration of a variable has the following structure:

```
type variableName = value;
```

Type – This can be a difficult concept for first time programmers. When we assign a varaible to memory, we do so with context of what *type* of data we are storing. For instance, we may be saving an integer, a list of characters, a number with some floating point precision, etc.

VariableName – This is just the name we will call this variable. It can be just about any word, but it can't include spacers or some other special characters. This is why we use a format called camel case, where we capitalize words without spaces between them instead of adding spaces.

Value – This is the actual value we are assigning to that variable. Some times we may just manually type in a value, and other times we might assign this value to be the output of some other process of function.

Here is an example where we create a new variable (of type integer), name it myNumber and give it a value of 42.

```
int myNumber = 42;
```

Part 3 – Programming Challenge

Now open the included sketch 'ArduinoProgramming.' This program is designed to generate two random numbers, add them together, and then print the results out to the terminal window for us to view. As it's currently written, it generates two random numbers between 0 and 10, called 'num1' and 'num2' respectively. Instead of adding these numbers together and assigning that value to the variable 'sum', it assigns a value of 10. Your challenge is to write a new function, called AddInts that adds together the two arguments you pass it, and use this function to generate the correct value for 'sum.' You'll do this by replacing the ?values in code using the included hints.