Functions in Arduino Programming

Functions are one of the core concepts of programming, and understanding how to use them will save lots of time and greatly expand your capacity for building out programs. Functions can be considered as sub-routines or sub-programs that run within your larger code. Functions are essentially groupings of code that are wrapped together and given a name that can then be called by the program at another point.

If you've worked with functions in math class, you may remember that they can take inputs, then perform operations on that input, and then output another value. Functions in programming work similarly.

You've actually already been working with functions in your Arduino programming! Both **void setup()** and **void loop()** are special functions that are necessary for all Arduino sketches.

The **void setup()** function executes the code contained within its { } whenever the Arduino board boots up, and the **void loop()** function is a loop that continually executes while the Arduino has power.

Going back to the basic *Blink* example (thought you could get away from that example, didn't you?), the code that controls the behavior of the LED and actually results in the blinking effect is contained within the **void loop()** function. That is essentially a subroutine of code that Arduino knows to continually execute.

Advantages of Functions

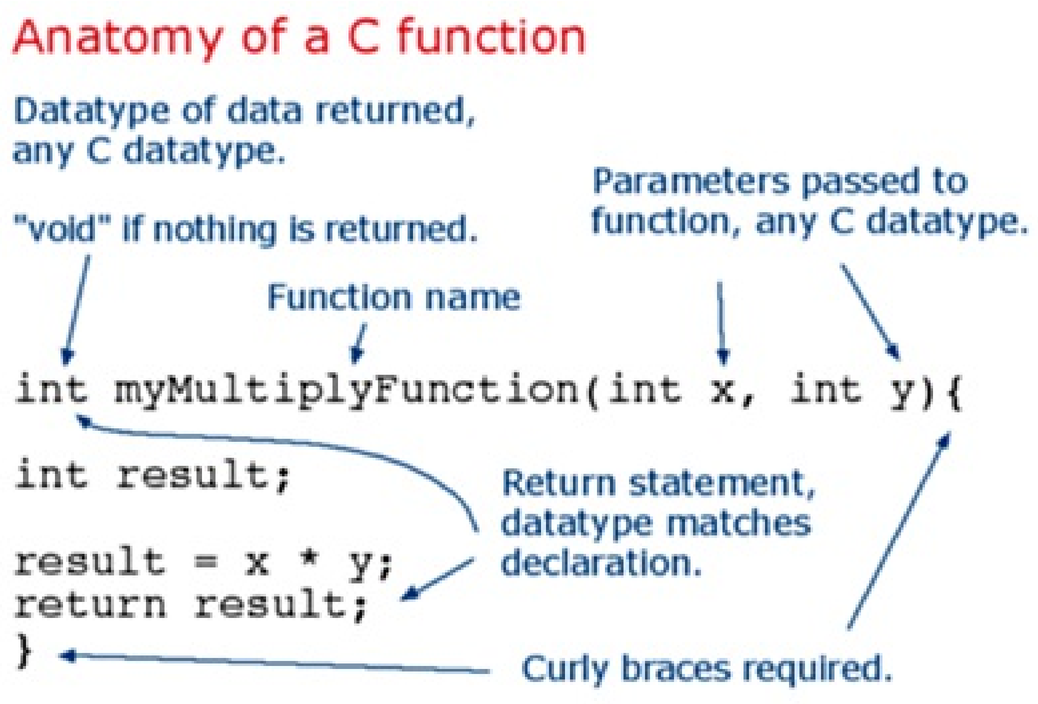
One of the most common questions from new programmers is "what is the benefit of writing a function?" One of the key benefits is that functions allow you to write modular, reusable code. If you find yourself writing the same code multiple times, it's time to consider putting that code inside a function that can then be *called* whenever you need it!

Here are some other key benefits of functions:

- Organization of code : Grouping your code into functions helps streamline and conceptualize your project.
- Clarity of code : Building out the functions that you need to accomplish the goal of your project often gives clarity to what you're doing.
- Makes your code compact : Once you get into the practice of creating and reusing functions, you'll see that your code becomes much more precise.

Structure of a Function

Just like in math class, functions often receive an input and *return* an output. This concept of returning a value is very important. Let's take a quick look at the structure of a function. Check out this image from the *Arduino Reference* documentation:


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.

```
int myMultiplyFunction(int x, int y){
int result;

result = x * y;
return result;
}
```

That image contains several important concepts and it's alright if it seems slightly overwhelming at first. As you delve into writing functions everything will become much clearer.

Here's a quick overview of how that example function would work:

1. The function is of type *int* because it's going to return an *integer* result.
2. The *name* of the function is "myMultiplyFunction" and it takes two integer parameters as input. These are the *int x* and *int y* that are inside the ( ) and before the { }
3. Inside the { } is the code that "myMultiplyFunction" executes when it's called. An integer variable named *result* is initialized and then set to be equal to the input *x* multiplied by the input *y*.
4. The function then uses *return* to output the value stored in result.
    - Since an *int* is returned, datatype of the function needs to match this returned value.

Consider the **void setup()** and **void loop()** functions that you've already worked with. These are of type *void* because they don't return a value after they're executed.

The myMultiplyFunction() would be declared somewhere else in your code, *outside* of either the **setup** or **loop** functions, and then can be called after it's created.

This function would then need to be called somewhere in your code. The following example on the *Arduino Reference* page includes the function inside the **void loop()** :

```
    void loop() {
        int i = 2;
        int j = 3;
        int k;
        k = myMultiplyFunction(i,j);
    }
```

In the above example, k would now contain the returned value of myMultiplyFunction(), which is 6.

Remember, functions can, but don't always, return a value. Some functions just execute the code. These functions would still need to be called.

Functions and Variable Scope

When working with variables inside functions, there is an important property called **scope** that needs to be considered. Scope is either *global* or *local*.

Global variables are variables that are created *outside* of any functions, whereas local variables are variables created *inside* a function. Global variables can be seen and used by every function in your code, whereas local variables are only seen by the function that they're declared in. Here's a non-programming parallel:

Your full given birth name is on your birth certificate and is seen/accessed by anyone in any of your circles, as long as you reveal it. This is your "global name." A nickname that is known to only your inner circle of friends and only applied to you when in their company is your "local name." There is no record of it anywhere else other than that particular local circle, so if someone called you by this particular nickname in an official capacity, it wouldn't be recognized.

If you need to be able to access the same variable in multiple functions, you'll generally want to create a global variable. However, there are some risks associated with this, but for right now you should use these when necessary.

Resources

- Arduino Reference Documentation : This has more details about functions in Arduino, and is where the image and example code in this lesson are sourced from.