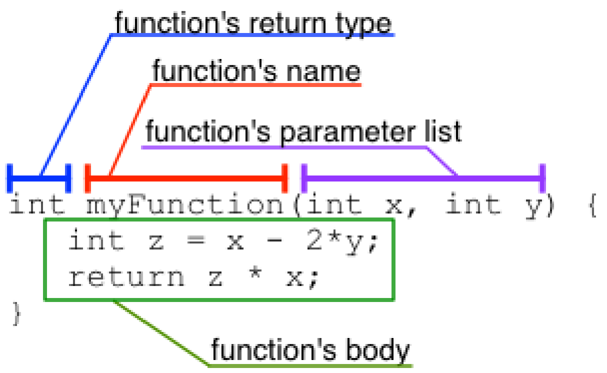
Anatomy of a Function

coursera.org/learn/programming-fundamentals/supplement/16ZXn/anatomy-of-a-function

Functions

A function gives a name to a parameterized computation—it is the implementation in code of a specific algorithm. All of the code that you will read or write (in this book) will be inside of functions. There are two sides to using functions in your programming: *declaring* a function—which provides the definition for how a function behaves—and *calling* a function—which executes the definition of the function on specific values of the parameters.



```
int myFunction(int x, int y) {
  int z = x - 2*y;
  return z * x;
}
```

The figure above shows a function declaration. The function's name may be any valid identifier, just like a variable's name. In this particular example, the function's name is **myFunction**. Immediately before the function's name is its *return type*—the type of value that this function will compute. As mentioned earlier, we will learn more about types later. For now, we will just work with ints, which are numbers. The fact that this function returns an int means that its "answer" is an int. After the function's name comes a set of parentheses, with the *parameter list* inside. The parameter list looks like a comma-separated list of variable declarations. Here, the function takes two parameters, **x** and **y**, both of which are ints. The similarity between parameters and variable declarations is not a coincidence—the parameters behave much like variables, but they are initialized by the function call (which we will discuss shortly).

The body of the function then comes between a set of curly braces, and is comprised of zero or more statements. The body of this function has two statements. The first statement in this function's body is the now-familiar declaration and initialization of a variable: **z** is declared as a variable of type int and initialized to the value of the expression **x – 2 * y**.

The second statement within the body of this function is a new type of statement which we have not seen before: a *return statement*. A return statement starts with the keyword **return**, which is then followed by an expression. The effect of this statement is to say what the "answer" is for the current function, leaving its computation and returning to the code that called it.

To understand this last concept completely, we must first see the other aspect of using a function—calling the function. A function is another kind of expression, whose value is whatever "answer" the called function comes up with when it is executed with the specified *arguments*—values of its parameters. This "answer" is more formally called the function's *return value*.