Functions

Duplicating code in different parts of a program can cause several problems. For one thing, if some of that code needed to be changed we would have to track down all the places we’ve copied it to, and we are likely to miss something. It can also make debugging harder and can make the program more difficult for another programmer to understand.

Luckily for us, we can use functions to encapsulate some code and use it in multiple places without having to copy and paste it. We have already used some functions in earlier chapters, such as printf() and System.out.println(). Any code you have written has also been inside a special function, main(). Every C and Java program must have a single function named main which is where the program starts.

# Function Definition

A function consists of four things which together make up the function definition.

returnType name(parameters)

{

body

}

* The return type specifies the type of data that you will get from the function. If the function does not return anything, the keyword void is used for the return type.
* The name of the function must be unique, and follows the same naming requirements as variable identifiers. This name is what we use to call the function.
* The parameters are a comma-separated list of variables (such as int foo, float bar) representing the data that should be given to the function when it is called. If there are no parameters, the keyword void can be used.
* The body of the function is the code that will be executed each time it is called. If the function is specified to return a type of data, the body must include at least one return statement to say what value is being returned.

In Java, there is an additional modifier that goes before the return type, to specify whether the function is public, private, or protected. If you do not add this modifier the function will default to private. These modifiers will be covered in more detail in a future chapter: for now, we will make all our functions public. We will also make all our functions static by placing the static keyword between the modifier and the return type. Non-static functions will be covered later.

The simplest function is one which has no parameters, returns nothing, and does nothing.

|  |  |
| --- | --- |
| **C** | **Java** |
| void my\_function(void)  {  } | public static void my\_function(void)  {  } |

# Function Parameters

Parameters let us use the function’s logic with different data. Inside the function, the parameters behave just like a regular variable. The example below takes in a floating point number, squares it, and returns the result.

float square(float value)

{

return value \* value;

}

An important thing to know about function parameters is that they **copy** their data. Any variables that you use to pass data into a function will not be affected by changes the function makes to its parameters.

int my\_function(int value)

{

value \*= 2;

return value;

}

int foo = 5;

int bar = my\_function(foo); /\* bar is 10, foo is still 5 \*/

Since using a function parameter is equivalent to creating a new variable and setting it equal to the variable that is passed in, the same rules apply as when trying to set one type of variable equal to another type of variable. If the function has an int parameter and you try to pass in a float, for example, the int won’t be able to store the decimal information from the float.

There is no limit to the number of parameters you can have in a function. If you have a long list of parameters, it can help the readability of your code to format it like this:

void complicated\_function(int parameter1,

int parameter2,

int parameter3,

int parameter4,

int parameter5,

int parameter6)

{

...

}

There are many other style choices as well. The important thing is, no matter what code styles you choose, to always be consistent.

# Return Statements

A return statement is used when we want to leave, or return from, a function. What is being returned must match the return type of the function definition.

These two functions both have illegal return statements:

int function1(int a, int b)

{

return; /\* Illegal, must return an integer \*/

}

void function2(int a, int b)

{

return 0; /\* Illegal, can’t return anything \*/

}

# Scope

We described how local variables behave with block scope in the previous chapter on Conditionals. This, of course, also applies to functions. All variables declared inside a function, as well as the function parameters, are only visible inside that function.

void function1(int a)

{

int x = 1;

}

void function2(int a)

{

int y = x; /\* **error**: x in function1 is not in scope \*/

}