Contents

[The Function of Functions 1](#_Toc57038853)

[Built-in Functions 2](#_Toc57038854)

[Declare & Define 2](#_Toc57038855)

[Void — The Point of No Return 3](#_Toc57038856)

[Return Types — Beyond the Void 4](#_Toc57038857)

[How Return Values Work 4](#_Toc57038858)

[Parameters & Arguments 5](#_Toc57038859)

[Tackling Multiple Arguments 6](#_Toc57038860)

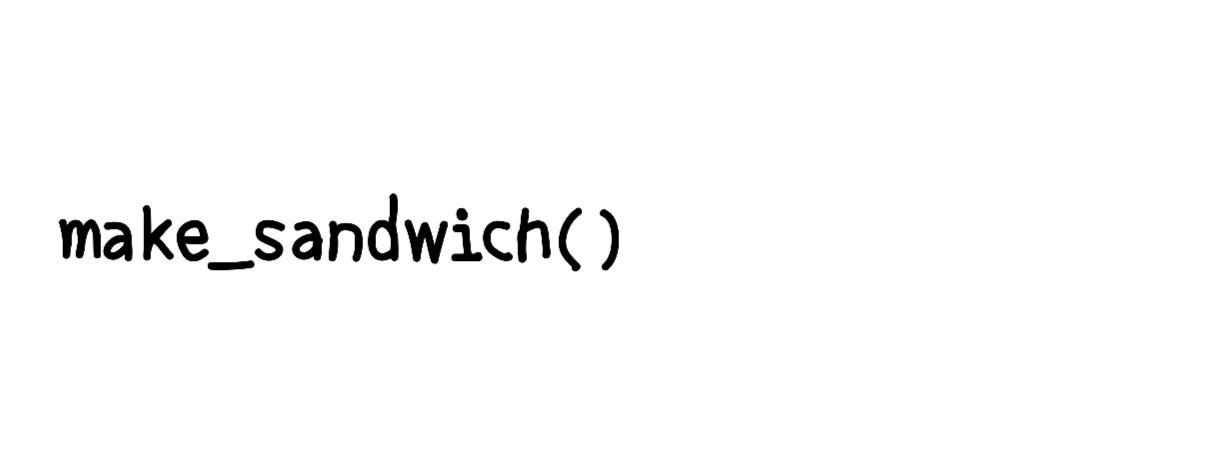
[How Parameters & Arguments Work 6](#_Toc57038861)

[Review of C++ Functions 7](#_Toc57038862)

## The Function of Functions

As a programmer, you will find yourself reusing the same blocks of code over and over throughout your program. In times like these, you can turn to functions.

Also known as a method or procedure, a *function* is a named group of code statements that accomplish something together, a bit like a factory machine.



There are some great reasons to use functions in your code:

* A single line can make all that code fire off instead of a whole bunch of lines.
* You can build DRY (Don’t Repeat Yourself) code, reusing the code you already wrote.
* Functions help make your code flexible and *modular*, meaning you can group your code more easily by task.

In fact, every C++ program has at least one function. “Hold on,” you may be thinking, “I’ve written some C++ programs, but I haven’t written any functions yet!”

Well, as it happens, main() is a function that you’ve already used! And you’ll understand it a bit more as you learn how functions work.

## Built-in Functions

Before we learn how to create functions, let’s go over some built-in functions…

C++ comes chock-full of functions that are already created as part of the standard library. But how do we access this hidden hoard of helpful functions? We gain access to various functions by including headers like <cmath> or <string>.

In fact, you may already have used a couple functions without even knowing it! With the following header:

#include <cmath>

We gain the power to call sqrt() to find the square root of any number.

Wait, “call” sqrt()?

*Calling* a function is how we get a function to take action. To call a basic function, we just need the function name followed by a pair of parentheses like sqrt(9). For example:

std::cout << sqrt(9) << "\n";  
   
// This would output 3

## Declare & Define

Often, built-in functions aren’t enough to tackle the wide array of programming challenges out there. But never fear: you can write your own functions too!

A C++ function is comprised of two distinct parts:

* *Declaration*: this includes the function’s name, what the return type is, and any parameters (if the function will accept input values, known as arguments).
* *Definition*: also known as the *body* of the function, this contains the instructions for what the function is supposed to do.

This is the overall structure:

return\_type function\_name( any, parameters, you, have ) {  
   
   // Code block here  
   
   return output\_if\_there\_is\_any;  
   
}

This is what it might look like with real code:

void make\_sandwich() {  
   
  std::cout << "bread\n";  
  std::cout << "egg\n";  
  std::cout << "cheese\n";  
  std::cout << "avocado\n";  
  std::cout << "bread\n";  
   
}

## Void — The Point of No Return

Let’s build a simple function with no input and no output. We can do that?

Enter the void specifier, which is added in the function declaration before the function name. A void function, also known as a subroutine, has no return value, making it ideally suited for situations where you just want to print stuff to the terminal.

For example:

void animal\_chat() {  
   
  std::string fav, pet;  
   
  std::cout << "What's your favorite animal?\n";  
  std::cin >> fav;  
   
  std::cout << "Do you have a " << fav << " as a pet? y/n\n";  
  std::cin >> pet;  
   
  if (pet == "y") {  
   
    std::cout << "How lucky you have a " << fav << " as a pet!\n";  
   
  } else {  
   
    std::cout << "That's too bad.\n";  
   
  }  
   
}

The above chat program is built to capture user responses and print to the terminal without returning any values.

## Return Types — Beyond the Void

When you do in fact want your function to return something and pass information back to the rest of your program, C++ has you covered. Just like there are many variable types, there are many different return types for functions.

A function can return most data types we’ve covered, including double, int, bool, char, std::string, and std::vector.

std::string always\_blue() {  
   
  return "blue!\n";  
   
}

**Note:** The return statement is the last line of code that a function will execute. For example:

std::string always\_blue() {  
   
  return "blue!\n";  
   
  std::cout << "Returned blue!";  
   
}

The final line will not execute because a value has already been returned. So "Returned blue!" won’t be printed to the terminal.

## How Return Values Work

When functions have a return type other than void, the function has two new requirements:

* There must be a value returned from the function.
* The return value must be the same type as the function’s return type.

But where does the return value get returned to?

It gets returned to the place where the function is called. For example, if you have the following function:

std::string feed\_the\_cat() {  
   
  return "Cat is fed!";  
   
}

And then print the function call inside of main():

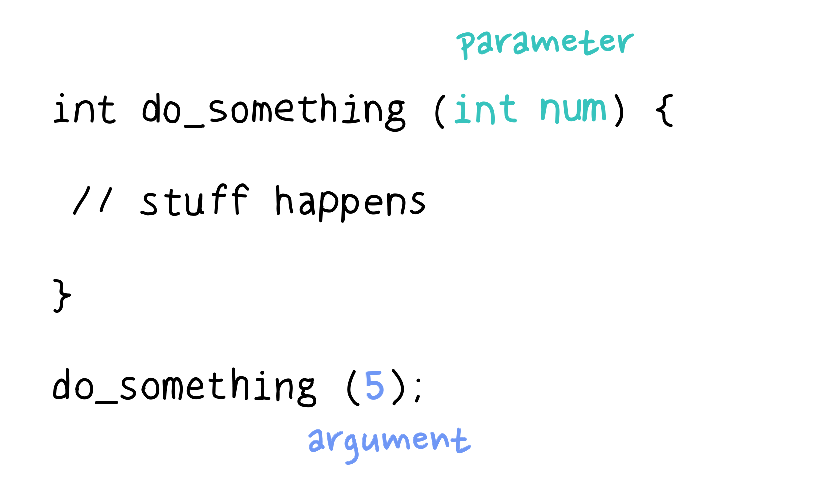
int main() {  
   
  std::string cat\_message = feed\_the\_cat();  
  std::cout << cat\_message;  
   
}

The return value of the function is what gets printed to the terminal.

## Parameters & Arguments

Returning data is all well and good, but let’s say you’re visiting NYC and you’ve been told that New Yorkers usually add a 20% tip for restaurants and taxis. It would be really convenient if you could just build a function that accepted different prices as input and figured out how much you should tip.

As it happens, you can do that with parameters. *Parameters* (sometimes called *formal parameters*) are variables used in a function’s definition. They act as placeholders for the input values you’ll use during your function call.



In the function below, price is the function’s parameter and it’s a double. It is declared between the parentheses and then used in the body of the function.

double get\_tip(double price) {  
   
  return price \* 0.2;  
   
}

Then, when you’re ready to use your function, the value you pass to the function is called an *argument* (also known as an *actual parameter*). In this case, 15.75 is the argument that is passed to the function:

double tip = get\_tip(15.75);  
// tip would be 3.15

## Tackling Multiple Arguments

Hang on, you may be thinking, are you limited to one parameter per function? Not at all! You can add as many as you like, but you will have to remember their order when you call the function.

double get\_tip(double price, double tip, bool total\_included) {  
   
  if (total\_included) {  
   
    return price \* tip + price;  
   
  } else {  
   
    return price \* tip;  
   
  }  
   
}

So here we have three parameters:

* double price
* double tip
* bool total\_included

When calling get\_tip(), it’s important to call it with price first, tip second, and return\_total last:

get\_tip(0.25, true, 45.50);  
// this code will not work

get\_tip(45.50, 0.25, true);  
// this code results in 56.875, which you could round up to 56.88

## How Parameters & Arguments Work

A function with parameters has a couple of requirements:

* The function call must include the **same number** of arguments as there are parameters.
* The corresponding arguments must be passed in the **same order**.

By calling a function with arguments, you are telling a function, “Hey function, when you execute, use these values where you have parameters in your definition.”

While it executes, anywhere the function comes across a parameter, it replaces the parameter with the corresponding argument you gave it.

## Review of C++ Functions

Check out all you’ve learned about C++ functions:

* A function is a named group of statements that do something together.
* Functions allow you to create more flexible, modular, and DRY code.
* C++ has many built-in functions that you can use.
* Functions are called like function\_name();
* A function has a declaration with a return type and possible parameters.
* A function has a definition (or body) with a group of statements and a possible return value.
* void functions do not have return values.
* Functions with a return value have return statements.
* Parameters are variables used as placeholders for function input values.
* Arguments are a function’s actual input values.

You now know enough C++ to create some pretty cool projects on your own. But, as you’ll see, there are still many ways to improve your code.