Unit:3

C++ Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters/arguments, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

## Create a Function

C++ provides some pre-defined functions, such as main ( ), which is used to execute code. But you can also create your own functions to perform certain actions.

To create (often referred to **as declare**) a function, specify the name of the function, followed by parentheses **()**:

void myFunction() {  
  // code to be executed  
}

## Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.

To call a function, write the function's name followed by two parentheses () and a semicolon ;

In the following example, myFunction() is used to print a text (the action), when it is called:

void myFunction()

{  
  cout << "I just got executed!";  
}  
  
int main()

{  
  **myFunction();** // call the function  
  return 0;  
}  
  
// Outputs "I just got executed!"

A function can be called multiple times:

void myFunction()

{  
  cout << "I just got executed!\n";  
}  
  
int main() {  
  **myFunction();**  
  **myFunction();**  
  **myFunction();**  
  return 0;  
}  
  
// I just got executed!  
// I just got executed!  
// I just got executed!

## Function Declaration and Definition

A C++ function consist of two parts:

* **Declaration:** the function's name, return type, and parameters (if any)
* **Definition:** the body of the function (code to be executed)

void **myFunction()** { // **declaration**  
  // the body of the function (**definition**)  
}

**Note:** If a user-defined function, such as myFunction() is declared after the main() function, **an error will occur**. It is because C++ works from top to bottom; which means that if the function is not declared above main().

You will often see C++ programs that have function declaration above main(), and function definition below main(). This will make the code better organized and easier to read:

// **1.** **Function declaration**  
void myFunction();  
  
**// The main method**int main() {  
  myFunction();  // **2. call the function**  
  return 0;  
}  
  
// **3.Function definition**  
void myFunction()

{  
  cout << "I just got executed!";  
}

## Parameters and Arguments

Information/data can be passed to functions as a parameter. Parameters act as variables inside the function.

**Terminology**

* **Formal Parameter :** A variable and its type as they appear in the prototype/declare of the function or method.
* **Actual Parameter :** The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

void functionName(parameter1, parameter2, parameter3)

{  
  // code to be executed  
}

C++ Functions - Pass By Value

The following example has a function that takes a string called **fname** as parameter. When the function is called, we pass along a first name, which is used inside the function to print the full name:

void myFunction(**string fname**)

{  
  cout << fname << " Refsnes\n";  
}  
  
int main() {  
  myFunction(**"Liam"**);  
  myFunction(**"Jenny"**);  
  myFunction(**"Anja"**);

  myFunction(**"Lalit"**);  
  
  return 0;  
}  
  
// Liam Refsnes  
// Jenny Refsnes  
// Anja Refsnes

//Lalit Refsnes

## Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

If we call the function without an argument, it uses the default value ("Norway"):

void myFunction(**string country = "Norway"**)

{  
  cout << country << "\n";  
}  
  
int main() {  
  myFunction("Sweden");  
  myFunction("India");  
  **myFunction();**  
  myFunction("USA");  
  return 0;  
}  
  
// Sweden  
// India  
// Norway  
// USA

## Multiple Parameters

Inside the function, you can add as many parameters as you want:

void myFunction(**string fname, int age**) {  
  cout << fname << " Refsnes. " << age << " years old. \n";  
}  
  
int main() {  
  myFunction(**"Liam", 3**);  
  myFunction(**"Jenny", 14**);  
  myFunction(**"Anja", 30**);  
  return 0;  
}  
  
// Liam Refsnes. 3 years old.  
// Jenny Refsnes. 14 years old.  
// Anja Refsnes. 30 years old.

## Return Values

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, string, etc.) instead of void, and use the return keyword inside the function:

**int** myFunction(int x) {  
  **return** 5 + x;  
}

int main() {  
  cout << myFunction(3);  
  return 0;  
}  
  
// Outputs 8 (5 + 3)

You can also store the result in a variable:

int myFunction(int x, int y) {  
  return x + y;  
}  
  
int main() {  
  int z = myFunction(5, 3);  
  cout << z;  
  return 0;  
}  
// Outputs 8 (5 + 3)

# C++ Functions - Pass By Reference

In the examples from the previous page, we used normal variables when we passed parameters to a function. You can also pass a [reference](https://www.w3schools.com/cpp/cpp_references.asp) to the function. This can be useful when you need to change the value of the arguments:

void swapNums(int &x, int &y) {  
  int z = x;  
  x = y;  
  y = z;  
}  
  
int main() {  
  int firstNum = 10;  
  int secondNum = 20;  
  
  cout << "Before swap: " << "\n";  
  cout << firstNum << secondNum << "\n";  
  
  // Call the function, which will change the values of firstNum and secondNum  
  swapNums(firstNum, secondNum);  
  
  cout << "After swap: " << "\n";  
  cout << firstNum << secondNum << "\n";  
  
  return 0;  
}

## Function Overloading

With**function overloading**, multiple functions can have the same name with different parameters:

int myFunction(int x)  
float myFunction(float x)  
double myFunction(double x, double y)

Consider the following example, which have two functions that add numbers of different type:

int plusFuncInt(int x, int y) {  
  return x + y;  
}  
double plusFuncDouble(double x, double y) {  
  return x + y;  
}  
int main() {  
  int myNum1 = plusFuncInt(8, 5);  
  double myNum2 = plusFuncDouble(4.3, 6.26);  
  cout << "Int: " << myNum1 << "\n";  
  cout << "Double: " << myNum2;  
  return 0;

[Run example »](https://www.w3schools.com/cpp/showcpp.asp?filename=demo_functions)