

Functions and String

Functions

A function is a named block of code which is used to perform a particular task. The power of functions lies in the capability to perform that task anywhere in the program without requiring the programmer to repeat that code many times. This also allows us to group portions of our code around concepts, making programs more organized. You can think of a function as a mini-program.

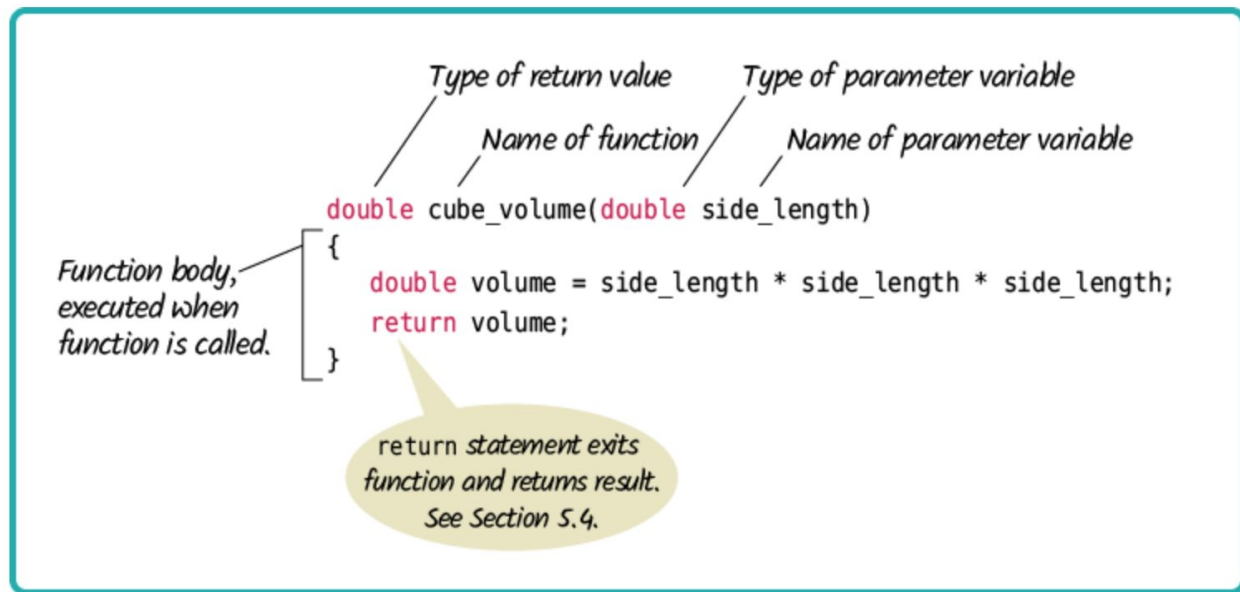
Depending on whether a function is predefined or created by programmer; there are two types of function:

1. Library Function
2. User-defined Function

Library functions are pre-written functions, grouped together in a **library** of related functions. For example, the C++ `math` library provides a `sqrt()` function to calculate the square root of a number. Libraries other than the built-in C++ libraries can be found online.

C++ allows programmers to define their own functions. These are called user-defined functions. Every valid C++ program has at least one function, the `main()` function.

We pass values to functions via **parameters**. In general, the parameters should be all the information needed for the function to do its work. When that work is complete, we would like to use the result in other code. The function can **return one value** of the specified type. A function may be used anywhere that is appropriate for an expression of that type. A function may also return nothing, in which case its return type is **void**.



Here is the syntax for the function definition:

```
return_type function_name( parameter_list )
{
    // function_body
}
```

- `return_type` is the data type of the value that the function returns.
- `function_name` is the actual name of the function.
- `parameter_list` refers to the type, order, and number of the parameters of a function. A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.
- `function_body` contains a collection of statements that define what the function does. The statements inside the function body are executed when a function is called.

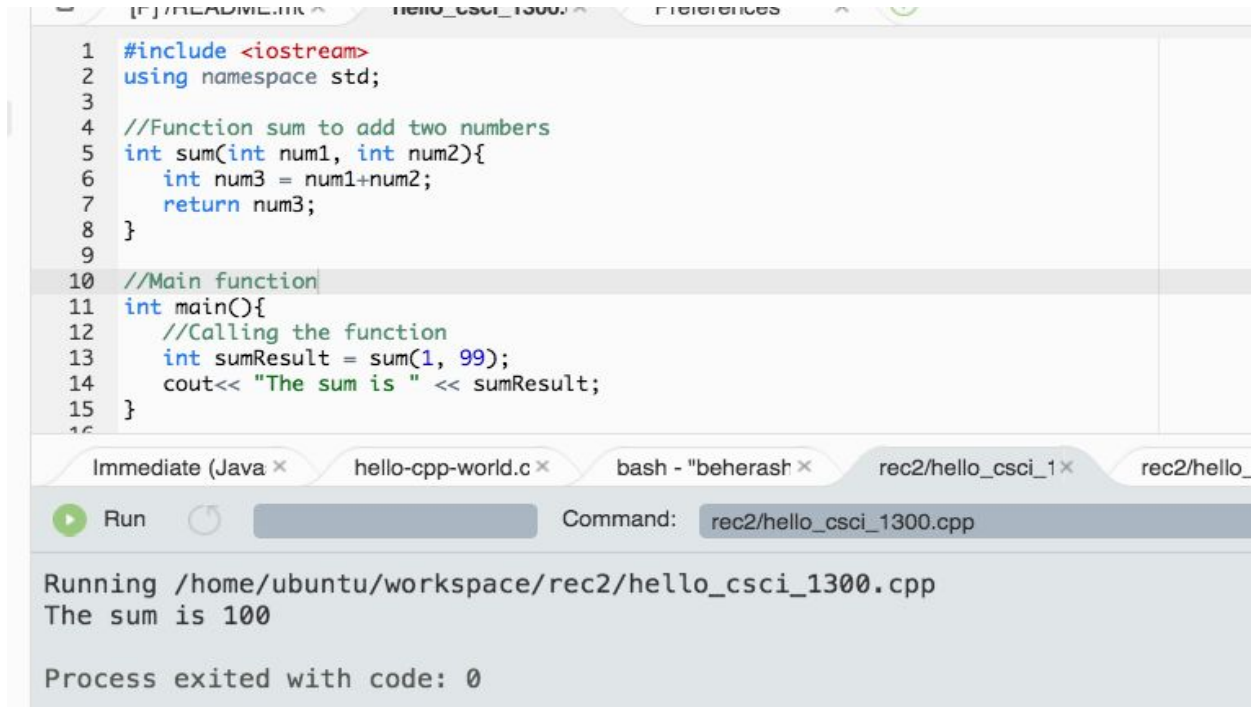
When we want to run (invoke) a function, we **call** it using syntax like this:

```
variable = function_name( parameters )
```

A function has its own **scope**. That means that the parameters of the function, as well as **local variables** declared *within* the function body, are not accessible outside the function. This is useful because it allows us to solve a small problem in a self-contained way. Parameter values and local variables disappear from memory when the function completes its execution.

When we call a function, the **flow of execution** changes. Let's follow in the example below:

When a program begins running, the system calls the `main()` function. When control of the program reaches the `sum()` function inside the `main()` (in the example below, at line **13**), it moves to function `sum()` and all code inside the function is executed (lines **6** and **7**). After the `sum()` function finishes, the execution returns in the `main()` function and the following statement is executed (line **14**).



```
1 #include <iostream>
2 using namespace std;
3
4 //Function sum to add two numbers
5 int sum(int num1, int num2){
6     int num3 = num1+num2;
7     return num3;
8 }
9
10 //Main function
11 int main(){
12     //Calling the function
13     int sumResult = sum(1, 99);
14     cout<< "The sum is " << sumResult;
15 }
16
```

Immediate (Java ×) hello-cpp-world.c × bash - "beherash × rec2/hello_csci_1× rec2/hello_

Run Command: rec2/hello_csci_1300.cpp

Running /home/ubuntu/workspace/rec2/hello_csci_1300.cpp
The sum is 100

Process exited with code: 0

Strings

In C++, `string` is a data type that represents sequences of characters instead of a numeric value (such as `int` or `float`). A string literal can be defined using double quotes. So `"Hello, world!"`, `"3.1415"`, and `"int i"` are all strings. We can access the character at a particular location within a string by using square brackets, which enclose an **index** which you can think of as the **address** of the character within the string. Importantly, strings in C++ are indexed starting from zero. This means that the first character in a string is located at index 0, the second character has index 1, and so on. For example:

```
string s = "Hello, world!";
cout << s[0] << endl; //prints the character 'H' to the screen
cout << s[4] << endl; //prints the character 'o' to the screen
```

```
cout << s[6] << endl; //prints the character ' ' to the screen
cout << s[12] << endl; //prints the character '!' to the screen
```

There are many useful standard functions available in C++ to manipulate strings. One of the simplest is `length()`. We can use this function to determine the number of characters in a string. This allows us to loop over a string character by character (i.e. *traverse the string*):

```
string s = "Hello, world!";
cout << s.length() << endl; //This will print 13
for (int i = 0; i < s.length(); i++)
{
    cout << s[i] << endl;
}
```

This will print each character in the string "Hello, world!" to the screen one per line. Notice how the `length` function is called.

The correct way:

- `s.length()`

Common mistakes:

- `length(s)`
- `s.length`
- `s.size()`

This is a special kind of function associated with objects, usually called a *method*, which we will discuss later in the course.