What is a function?

A function is a block of code which only runs when it is called. You can pass data, known as parameters, 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

}
```

- **myFunction()** is the name of the function
- **void** means that the function does not have a return value.
- inside the function (the **body**), add code that defines what the function should do

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:

Example: Inside main, call myFunction()

```
// Create a function
void myFunction() {
  cout << "I just got executed!";
}

int main() {
  myFunction(); // call the function
  return 0;
}

// Outputs "I just got executed!"
```

Example: 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 return type, the name of the function, 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**:

Example:

```
#include <iostream>
using namespace std;

int main() {
  myFunction();
  return 0;
}

void myFunction() {
  cout << "I just got executed!";
}
```

output :

```
In function 'int main()':
5:3: error: 'myFunction' was not declared in
this scope
```

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:

```
// Function declaration
void myFunction();

// The main method
int main() {
  myFunction();  // call the function
  return 0;
}

// Function definition
void myFunction() {
  cout << "I just got executed!";
}
```

Parameters and Arguments

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

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:

```
//Syntax
void functionName (parameter1, parameter2, parameter3) {
  // code to be executed
}
```

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:

Example

```
void myFunction(string fname) {
  cout << fname << " Refsnes\n";
}
int main() {
  myFunction("Liam");
  myFunction("Jenny");
  myFunction("Anja");
  return 0;
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

Note: When a **parameter** is passed to the function, it is called an **argument**. So, from the example above: fname is a parameter, while Liam, Jenny and Anja are **arguments**.

Note: 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"):

Example

```
void myFunction(string country = "Norway") {
  cout << country << "\n";
}
```

```
int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}
// Sweden
// India
// Norway
// USA
```

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

Example

```
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.
```

Note: that when you are working with multiple parameters, the function call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

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:

Example:

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

Example:

This example returns the sum of a function with two parameters:

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  cout << myFunction(5, 3);
  return 0;
}
```

```
// Outputs 8 (5 + 3)
```

You can also pass arrays to a function:

Example:

```
void myFunction (int myNumbers[5]) {
  for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
  }
}
int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

The function (myFunction) takes an array as its parameter (int myNumbers[5]), and loops through the array elements with the for loop. When the function is called inside main(), we pass along the myNumbers array, which outputs the array elements.

Note: that when you call the function, you only need to use the name of the array when passing it as an argument myFunction(myNumbers). However, the full declaration of the array is needed in the function parameter (int myNumbers[5]).

Real Life Example:

create a program that converts a value from Fahrenheit to Celsius.

```
// Function to convert Fahrenheit to Celsius
float toCelsius(float fahrenheit) {
  return (5.0 / 9.0) * (fahrenheit - 32.0);
}

int main() {
  // Set a fahrenheit value
  float f_value = 98.8;

  // Call the function with the fahrenheit value
  float result = toCelsius(f_value);

  // Print the fahrenheit value
  cout << "Fahrenheit: " << f_value << "\n";

  // Print the result
  cout << "Convert Fahrenheit to Celsius: " << result << "\n";

  return 0;
}
```

Home work (1): Write a function named factorial that takes a positive integer as an argument and returns its factorial (n!).

Home work (2): Write a function that takes three floating-point numbers as input and returns their average