# TUTORIAL 5: C++ FUNCTION

**Function Declaration and Definition**

A C++ function consist of two parts:

1) Declaration: the return type, the name of the function, and parameters (if any)
2) Definition: the body of the function (code to be executed).

**NOTE: Before using /calling function, you have to define it. Otherwise it will generate error. To avoid it programmer just declared function before main function body. See below example.**

If a user-defined function, such as myFunction() is declared after the main() function, an error will occur:

**Example (ERROR WILL GENRATE)**

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

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

However, it is possible to separate the declaration and the definition of the function - for code optimization.

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:

**Example**

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

```
// The main method
int main() {
  myFunction();  // call the function
  return 0;
}
// Function definition
void myFunction() {
  cout << "I just got executed!";
}
```

# DEFAULT PARAMETERS IN FUNCTIONS

**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"):

**Example**

```
void myFunction(string country = "Norway") {
  cout << country << "\n";
}
int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();      //CALL WITHOUT PASSING  ARGUMENT
  myFunction("USA");
  return 0;
}
```

**OUTPUT:**

Sweden

India

Norway

USA

A parameter with a default value, is often known as an "optional parameter". From the example above, country is an optional parameter and "Norway" is the default value.

# PASS BY REFERENCE:

Function only return one value (as per return datatype), but the time when you want to change multiple parameters value during function call at that time this concept "**PASS REFERENCE**" is very useful. C++ is supports pass by reference concept.(Original concept create reference)

**EXAMPLE:**

```
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;
}
```

## Pass Arrays as Function Parameters

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;
}
```

**Example Explained**

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]).

# C++ FUNCTION OVERLOADING:

C++ supports function overloading.(OOP FEATURE – POLYMORPHISM)

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

Note: Multiple functions can have the same name as long as the number and/or type of parameters are different.

**EXAMPLE:**

```
#include<iostream>

using namespace std;

//Function overloading

void print_ur_arg(int x)
{

   cout << "You are passing integer : "<<x<<endl;
}

void print_ur_arg(double x)
{

   cout << "You are passing float : "<<x<<endl;
}

void print_ur_arg(string x)
{
```

```
    cout << "You are passing string : "<<x<<endl;
}

int main()
{
 print_ur_arg(10);      //pass int
 print_ur_arg(10.0);    //pass float
 print_ur_arg("TEN");   //pass string
    return 0;
}
```

# Recursion

**Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.**

**NOTE: The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.**

Breaking of recursive function is important.

```
int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}

int main() {
  int result = sum(10);
```

```
  cout << result;
  return 0;
}
```