# Functions & Function Prototypes in C++ | C++ Tutorials for Beginners #15

In this tutorial, we will discuss functions and functions prototype in C++

## Functions in C++

Functions are the main part of top-down structured programming. We break the code into small pieces and make functions of that code. Functions help us to reuse the code easily. An example program for the function is shown in Code Snippet 1.

```
int sum(int a, int b){
 int c = a+b;
    return c;
}
```

***Code Snippet 1: Function example***

As shown in Code Snippet 1, we created an integer function with the name of sum, which takes two parameters “int a” and “int b”. In the function, body addition is performed on the values of variable “a” and variable “b” and the result is stored in variable “c”. In the end, the value of variable “c” is returned to the function. We have seen how this function works now we will see how to pass values to the function parameters. An example program for passing the values to the function is shown in Code Snippet 2.

```
int main(){
    int num1, num2;
    cout<<"Enter first number"<<endl;
    cin>>num1;
    cout<<"Enter second number"<<endl;
    cin>>num2;
    cout<<"The sum is "<<sum(num1, num2);
    return 0;
}
```

***Code Snippet 2: Passing Value to Function Parameters***

As shown in Code Snippet 2, we have declared two integer variables “num1” and “num2”, we will take their input at run time. In the end, we called the “sum” function and passed both variables “num1” and “num2” into sum function. “sum” function will perform the addition and returns the value at the same location from where it was called. The output of the following program is shown in figure 1.

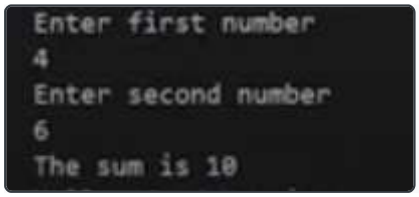


***Figure 1: Function Output***

## Function Prototype in C++

The function prototype is the template of the function which tells the details of the function e.g(name, parameters) to the compiler. Function prototypes help us to define a function after the function call. An example of a function prototype is shown in Code Snippet 3.

```
// Function prototype
    int sum(int a, int b);
```

***Code Snippet 3: Function Prototype***

As shown in Code Snippet 3, we have made a function prototype of the function “sum”, this function prototype will tell the compiler that the function “sum” is declared somewhere in the program which takes two integer parameters and returns an integer value. Some examples of acceptable and not acceptable prototypes are shown below:

- int sum(int a, int b); //Acceptable
- int sum(int a, b); // Not Acceptable
- int sum(int, int); //Acceptable

### Formal Parameters

The variables which are declared in the function are called a formal parameter. For example, as shown in Code Snippet 1, the variables “a” and “b” are the formal parameters.

### Actual Parameters

The values which are passed to the function are called actual parameters. For example, as shown in Code Snippet 2, the variables “num1” and “num2” are the actual parameters.

The function doesn't need to have parameters or it should return some value. An example of the void function is shown in Code Snippet 4.

```
void g(){
    cout<<"\nHello, Good Morning";
}
```

***Code Snippet 4: Void Function***

As shown in Code Snippet 4, void as a return type means that this function will not return anything, and this function has no parameters. Whenever we will call this function it will print “Hello, Good Morning”

## Code as described/written in the video

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

// Function prototype
// type function-name (arguments);
// int sum(int a, int b); //--> Acceptable
// int sum(int a, b); //--> Not Acceptable
int sum(int, int); //--> Acceptable
// void g(void); //--> Acceptable
void g(); //--> Acceptable

int main(){
    int num1, num2;
    cout<<"Enter first number"<<endl;
    cin>>num1;
    cout<<"Enter second number"<<endl;
    cin>>num2;
    // num1 and num2 are actual parameters
    cout<<"The sum is "<<sum(num1, num2);
    g();
    return 0;
}
```