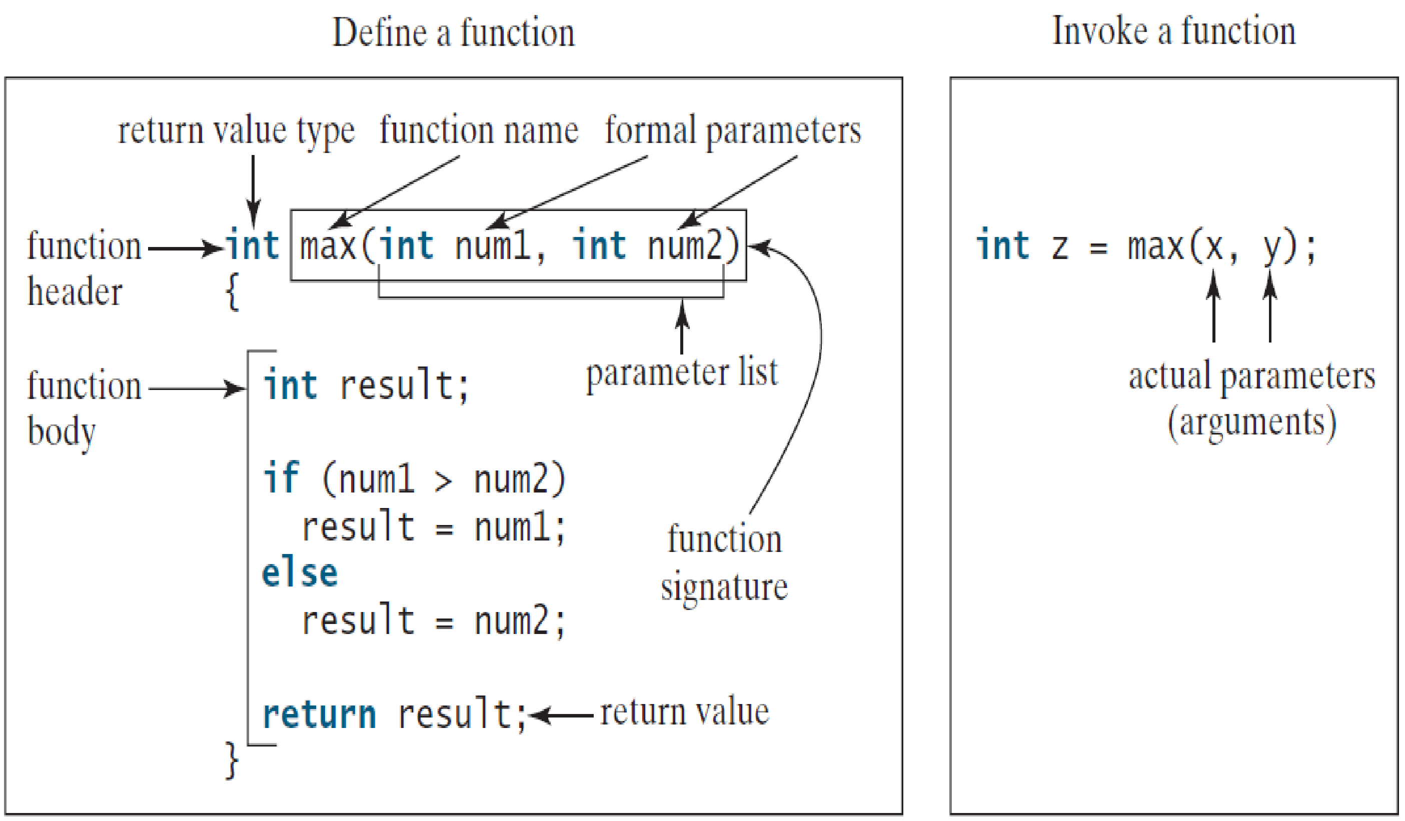
Functions are group of statements that perform a task,

They take 0 or more parameters and perform their statements and return an output (<u>except for void functions that don't return a value</u>).



```
int max(int num1, int num2)
{
    int result;

    if (num1 > num2)
      result = num1;
    else
      result = num2;

    return result;
}
```

```
int z = max(x, y);
```

- A function prototype (signature) consists of the <u>return type, the name of the function and the parameter list.</u>
- A function body is the statements that gets executed when the function is **called.**
- A function can return a value and ends it's execution with the **return** keyword (for a void function the return keywork can be used to end the execution early).

Arguments Can be passed by reference or by value:-

-if passed by value then the argument is copied and is passed to the function and the original copy is untouched

```
void add_five(int var){ //var taken by value
    var = var + 5;
}
```

```
int var = 3;
add_five(var); //var value doesn’t change

//output is “var: 3”
cout << "var: " << var;
```

-if passed by reference then the original variable is passed and any changes that happens to it in the function’s body also changes the original variable

```
void add_five(int & var){ //var taken by reference
    var = var + 5;
}
```

```
int var = 3;
add_five(var); //var value does change

//output is “var: 8”
cout << "var: " << var;
```

-constant references can be used to guaranty that no changes happen to the variable

Function overloading is creating different functions with the **same name** but with **different parameter list** <u>(changing the return type of the function doesn’t matter).</u>

```
//arguments are taken as constant references
//to guaranty the original variables are
//unmodified

//swap two ints
void swap(const int & a, const int & b){
    int tmp = a;
    a = b;
    b = tmp;
}

//swap two strings
void swap(const string & a, const string & b){
    string tmp = s1;
    s1 = s2;
    s2 = tmp;
}
```

Functions can have default arguments for their parameters

That are used when no arguments are provided by the caller.

Default parameters have to be declared in order and they can not be skipped.

```
void func1 (int x, int y = 3, int z); //Illegal
void func2 (int x, int y = 3, int z = 5);
// Legal
```

Passing arguments have also to be done in order and no argument can be skipped and the argument(s) after it given values.

```
func1(2, , 20); // illegal
func2(2) // legal as y and z use their default
values
```

C++ has three types of variable scopes and they are:-

1. Global Scope: global scopes define variables outside of any function and are accessible anywhere in the program

2. Local Scope: Local scope define variables that are visible only inside a specific region of the program and are **inaccessible outside it.**

   This region of code is usually contained withen curly brackets and they mark the **body of a function** or a body of a **loop** or an **if** statement or an **unbounded internal scope**

3. Static Scope: static scope defines variables that are local to a function and which values **aren't destroyed** after the function **returns** and **retain their values when the function is called again.**

- Static variables should be **initialized at declaration** or else they won't retain their value when the function is called again

```
//global variable accessible everywhere
int x = 5;

int function(){
    //static variable that retains it's value
    static int sum = 0;

    for (int i = 0; i < 10; i++){
        //variables value and i are local
        //to the  for loop scope
        int value = i + 5;
        sum += value;
    }
    {
        //variable z is local and accessible
        //only in this inner scope
        int z = 2 * sum;

    }
    //illegal as z is inaccessible here
    cout << z;
    return sum;
}
```

- Inline functions: are function whose statements get inserted where they are called to avoid the overhead of function calls and speed up the program

- Inline functions should be simple functions and should not contain **Loops** or **Recursion**

```
inline int mult3(int x){
    return x * x * x;
}



int x = 5;

cout << mult3(x) << endl; //output is 75
// the above will expand to this
cout << x * x * x << endl;
```