OBJECTIVE-C FUNCTIONS

A function is a group of statements that together perform a task. Every Objective-C program has one C function, which is **main**, and all of the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Basically in Objective-C, we call the function as method.

The Objective-C foundation framework provides numerous built-in methods that your program can call. For example, method **appendString** to append string to another string.

A method is known with various names like a function or a sub-routine or a procedure, etc.

Defining a Method

The general form of a method definition in Objective-C programming language is as follows:

```
- (return_type) method_name:( argumentType1 )argumentName1
joiningArgument2:( argumentType2 )argumentName2 ...
joiningArgumentn:( argumentTypen )argumentNamen
{
   body of the function
}
```

A method definition in Objective-C programming language consists of a *method header* and a *method body*. Here are all the parts of a method:

- **Return Type:** A method may return a value. The **return_type** is the data type of the value the function returns. Some methods perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Method Name:** This is the actual name of the method. The method name and the parameter list together constitute the method signature.
- **Arguments:** A argument is like a placeholder. When a function is invoked, you pass a value to the argument. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the arguments of a method. Arguments are optional; that is, a method may contain no argument.
- **Joining Argument:** A joining argument is to make it easier to read and to make it clear while calling it.
- **Method Body:** The method body contains a collection of statements that define what the method does.

Example:

Following is the source code for a method called **max**. This method takes two parameters num1 and num2 and returns the maximum between the two:

```
/* function returning the max between two numbers */
- (int) max:(int) num1 secondNumber:(int) num2
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
   {
```

```
      result = num1;
   }
   else
   {
      result = num2;
   }

   return result;
}
```

Method Declarations:

A method **declaration** tells the compiler about a function name and how to call the method. The actual body of the function can be defined separately.

A method declaration has the following parts:

```
- (return_type) function_name:( argumentType1 )argumentName1
joiningArgument2:( argumentType2 )argumentName2 ...
joiningArgumentn:( argumentTypen )argumentNamen;
```

For the above-defined function max, following is the method declaration:

```
-(int) max:(int)num1 andNum2:(int)num2;
```

Method declaration is required when you define a method in one source file and you call that method in another file. In such case you should declare the function at the top of the file calling the function.

Calling a method:

While creating a Objective-C method, you give a definition of what the function has to do. To use a method, you will have to call that function to perform the defined task.

When a program calls a function, program control is transferred to the called method. A called method performs defined task, and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a method, you simply need to pass the required parameters along with method name, and if method returns a value, then you can store returned value. For example:

```
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
/* method declaration */
- (int)max:(int)num1 andNum2:(int)num2;
@end

@implementation SampleClass

/* method returning the max between two numbers */
- (int)max:(int)num1 andNum2:(int)num2{
/* local variable declaration */
   int result;

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

   return result;
}
```

```
@end

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   SampleClass *sampleClass = [[SampleClass alloc]init];

   /* calling a method to get max value */
   ret = [sampleClass max:a andNum2:b];

   NSLog(@"Max value is : %d\n", ret );

   return 0;
}
```

I kept max function along with main function and complied the source code. While running final executable, it would produce the following result:

```
2013-09-07 22:28:45.912 demo[26080] Max value is : 200
```

Function Arguments:

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, Objective-C uses **call by value** to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function, and above-mentioned example while calling max function used the same method.