OBJECTIVE-C

programming language reference

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. This is the main programming language used by Apple for the OS X and iOS operating systems and their respective APIs, Cocoa and Cocoa Touch.

This reference will take you through simple and practical approach while learning Objective-C Programming language.

Audience

This reference has been prepared for the beginners to help them understand basic to advanced concepts related to Objective-C Programming languages.

Prerequisites

Before you start doing practice with various types of examples given in this reference, I'm making an assumption that you are already aware about what is a computer program, and what is a computer programming language?

Copyright & Disclaimer

Table of Contents

1. OVERVIEW

Objective-C is general-purpose language that is developed on top of C Programming language by adding features of Small Talk programming language making it an object-oriented language. It is primarily used in developing iOS and Mac OS X operating systems as well as its applications.

Initially, Objective-C was developed by NeXT for its NeXTSTEP OS from whom it was taken over by Apple for its iOS and Mac OS X.

Object-Oriented Programming

Objective-C fully supports object-oriented programming, including the four pillars of object-oriented development:

- Encapsulation
- Data hiding
- Inheritance
- Polymorphism

Example Code

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

int main (int argc, const char * argv[])
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

   NSLog (@"hello world");
   [pool drain];
   return 0;
}
```

Foundation Framework

Foundation Framework provides large set of features and they are listed below.

- It includes a list of extended datatypes like NSArray, NSDictionary, NSSet and so on.
- It consists of a rich set of functions manipulating files, strings, etc.
- It provides features for URL handling, utilities like date formatting, data handling, error handling, etc.

Learning Objective-C

The most important thing to do when learning Objective-C is to focus on concepts and not get lost in language technical details.

The purpose of learning a programming language is to become a better programmer; that is, to become more effective at designing and implementing new systems and at maintaining old ones.

Use of Objective-C

Objective-C, as mentioned earlier, is used in iOS and Mac OS X. It has large base of iOS users and largely increasing Mac OS X users. As Apple focuses on quality first, its wonderful for those who started learning Objective-C.

2. ENVIRONMENT SETUP

Try it Option Online

You really do not need to set up your own environment to start learning Objective-C programming language. Reason is very simple, we already have set up Objective-C Programming environment online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

*Try the following example using **Try it** option available at the top right corner of the below sample code box:*

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

int main()
{
   /* my first program in Objective-C */
   NSLog(@"Hello, World! \n");

   return 0;
}
```

*For most of the examples given in this tutorial, you will find **Try it** option, so just make use of it and enjoy your learning.*

Local Environment Setup

If you are still willing to set up your own environment for Objective-C programming language, then you need to install **Text Editor** and **The GCC Compiler** on your computer.

Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

Name and version of text editor can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on windows as well as Linux or UNIX.

The files you create with your editor are called source files and contain program source code. The source files for Objective-C programs are typically named with the extension "**.m**".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, compile it and finally execute it.

The GCC Compiler

The source code written in source file is the human readable source for your program. It needs to be "compiled" to turn into machine language, so that your CPU can actually execute the program as per instructions given.

This GCC compiler will be used to compile your source code into final executable program. I assume you have basic knowledge about a programming language compiler.

GCC compiler is available for free on various platforms and the procedure to set up on various platforms is explained below.

Installation on UNIX/Linux

The initial step is install gcc along with gcc Objective-C package. This is done by:

```
$ su -
$ yum install gcc
$ yum install gcc-objc
```

The next step is to set up package dependencies using following command:

```
$ yum install make libpng libpng-devel libtiff libtiff-devel libobjc
libxml2 libxml2-devel libX11-devel libXt-devel libjpeg libjpeg-devel
```

In order to get full features of Objective-C, download and install GNUStep. This can be done by downloading the package from http://main.gnustep.org/resources/downloads.php.

Now, we need to switch to the downloaded folder and unpack the file by:

```
$ tar xvfz gnustep-startup-.tar.gz
```

Now, we need to switch to the folder gnustep-startup that gets created using:

```
$ cd gnustep-startup-
```

Next, we need to configure the build process:

```
$ ./configure
```

Then, we can build by:

```
$ make
```

We need to finally set up the environment by:

```
$ . /usr/GNUstep/System/Library/Makefiles/GNUstep.sh
```

We have a helloWorld.m Objective-C as follows:

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

int main (int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog (@"hello world");
    [pool drain];
    return 0;
}
```

Now, we can compile and run an Objective-C file say helloWorld.m by switching to folder containing the file using cd and then using the following steps:

```
$ gcc `gnustep-config --objc-flags` -L/usr/GNUstep/Local/Library/Libraries
-lgnustep-base helloWorld.m -o helloWorld
$ ./helloWorld
```

We can see the following output:

```
2013-09-07 10:48:39.772 tutorialsPoint[12906] hello world
```

Installation on Mac OS

If you use Mac OS X, the easiest way to obtain GCC is to download the Xcode development environment from Apple's web site and follow the simple installation instructions. Once you have Xcode set up, you will be able to use GNU compiler for C/C++.

Xcode is currently available at developer.apple.com/technologies/tools/.

Installation on Windows

In order to run Objective-C program on windows, we need to install MinGW and GNUStep Core. Both are available at gnustep.org/experience/Windows.html.

First, we need to install the MSYS/MinGW System package. Then, we need to install the GNUstep Core package. Both of which provide a windows installer, which is self-explanatory.

Then to use Objective-C and GNUstep by selecting Start -> All Programs -> GNUstep -> Shell

Switch to the folder containing helloWorld.m

We can compile the program by using:

```
$ gcc `gnustep-config --objc-flags` -L /GNUstep/System/Library/Libraries
hello.m -o hello -lgnustep-base -lobjc
```

We can run the program by using:

```
./hello.exe
```

We get the following output:

```
2013-09-07 10:48:39.772 tutorialsPoint[1200] hello world
```

3. PROGRAM STRUCTURE

Before we study basic building blocks of the Objective-C programming language, let us look a bare minimum Objective-C program structure so that we can take it as a reference in upcoming chapters.

Objective-C Hello World Example

An Objective-C program basically consists of the following parts:

- Preprocessor Commands
- Interface
- Implementation
- Method
- Variables
- Statements & Expressions
- Comments

Let us look at a simple code that would print the words "Hello World":

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

@interface SampleClass:NSObject
- (void)sampleMethod;
@end

@implementation SampleClass

- (void)sampleMethod{
   NSLog(@"Hello, World! \n");
}
```

```
@end

int main()
{
   /* my first program in Objective-C */
   SampleClass *sampleClass = [[SampleClass alloc]init];
   [sampleClass sampleMethod];
   return 0;
}
```

Let us look various parts of the above program:

1. The first line of the program *#import <Foundation/Foundation.h>* is a preprocessor command, which tells an Objective-C compiler to include Foundation.h file before going to actual compilation.

2. The next line *@interface SampleClass:NSObject* shows how to create an interface. It inherits NSObject, which is the base class of all objects.

3. The next line - *(void)sampleMethod;* shows how to declare a method.

4. The next line *@end* marks the end of an interface.

5. The next line *@implementation SampleClass* shows how to implement the interface SampleClass.

6. The next line - *(void)sampleMethod{}* shows the implementation of the sampleMethod.

7. The next line *@end* marks the end of an implementation.

8. The next line *int main()* is the main function where program execution begins.

9. The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.

10. The next line *NSLog(...)* is another function available in Objective-C which causes the message "Hello, World!" to be displayed on the screen.

11.The next line **return 0;** terminates main()function and returns the value 0.

Compile & Execute Objective-C Program

Now when we compile and run the program, we will get the following result.

```
2013-09-07 22:38:27.932 demo[28001] Hello, World!
```

4. BASIC SYNTAX

You have seen a basic structure of Objective-C program, so it will be easy to understand other basic building blocks of the Objective-C programming language.

Tokens in Objective-C

An Objective-C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following Objective-C statement consists of six tokens:

```
NSLog(@"Hello, World! \n");
```

The individual tokens are:

```
NSLog
@
(
"Hello, World! \n"
)
;
```

Semicolons ;

In Objective-C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

For example, following are two different statements:

```
NSLog(@"Hello, World! \n");
return 0;
```

Comments

Comments are like helping text in your Objective-C program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below:

```
/* my first program in Objective-C */
```

You cannot have comments within comments and they do not occur within a string or character literals.

Identifiers

An Objective-C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).

Objective-C does not allow punctuation characters such as @, $, and % within identifiers. Objective-C is a **case-sensitive** programming language. Thus, *Manpower* and *manpower* are two different identifiers in Objective-C. Here are some examples of acceptable identifiers:

```
mohd       zara    abc   move_name  a_123
myname50   _temp   j     a23b9      retVal
```

Keywords

The following list shows few of the reserved words in Objective-C. These reserved words may not be used as constant or variable or any other identifier names.

Auto	Else	long	Switch
Break	enum	register	Typedef
Case	extern	return	Union
char	float	short	Unsigned
const	For	signed	Void
continue	Goto	sizeof	Volatile
default	If	static	While

do	Int	struct	_Packed
double	protocol	interface	Implementation
NSObject	NSInteger	NSNumber	CGFloat
property	nonatomic;	retain	Strong
weak	unsafe_unretained;	readwrite	Readonly

Whitespace in Objective-C

A line containing only whitespace, possibly with a comment, is known as a blank line, and an Objective-C compiler totally ignores it.

Whitespace is the term used in Objective-C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement:

```
int age;
```

There must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them. On the other hand, in the following statement:

```
fruit = apples + oranges;   // get the total fruit
```

No whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

5. DATA TYPES

In the Objective-C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in Objective-C can be classified as follows:

S.N.	Types and Description
1	**Basic Types:** They are arithmetic types and consist of the two types: (a) integer types and (b) floating-point types.
2	**Enumerated types:** They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section whereas other types will be covered in the upcoming chapters.

Integer Types

Following table gives you details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
Char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expression *sizeof(type)* yields the storage size of the object or type in bytes. Following is an example to get the size of int type on any machine:

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


int main()
{
```

```
   NSLog(@"Storage size for int : %d \n", sizeof(int));

   return 0;
}
```

When you compile and execute the above program, it produces the following result on Linux:

```
2013-09-07 22:21:39.155 demo[1340] Storage size for int : 4
```

Floating-Point Types

Following table gives you details about standard float-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. Following example will print storage space taken by a float type and its range values:

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

int main()
{
   NSLog(@"Storage size for float : %d \n", sizeof(float));

   return 0;
}
```

When you compile and execute the above program, it produces the following result on Linux:

```
2013-09-07 22:22:21.729 demo[3927] Storage size for float : 4
```

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

S.N.	Types and Description
1	**Function returns as void** There are various functions in Objective-C which do not return value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in Objective-C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**

The void type may not be understood to you at this point, so let us proceed and we will cover these concepts in upcoming chapters.

6. VARIABLES

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in Objective-C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because Objective-C is case-sensitive. Based on the basic types explained in previous chapter, there will be the following basic variable types:

Type	Description
char	Typically a single octet (one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

Objective-C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like enumeration, pointer, array, structure, union, etc. For this chapter, let us study only basic variable types.

Variable Definition in Objective-C

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid Objective-C data type including char, w_char, int, float, double, bool or any user-defined object, etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

Variable Declaration in Objective-C

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files, which will be available at the time of linking of the program. You will use **extern** keyword to declare a variable at any place. Though you can declare a variable multiple times in your Objective-C program but it can be defined only once in a file, a function or a block of code.

Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function:

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

// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main ()
{
  /* variable definition: */
  int a, b;
  int c;
  float f;

  /* actual initialization */
  a = 10;
  b = 20;

  c = a + b;
  NSLog(@"value of c : %d \n", c);

  f = 70.0/3.0;
  NSLog(@"value of f : %f \n", f);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-07 22:43:31.695 demo[14019] value of c : 30
2013-09-07 22:43:31.695 demo[14019] value of f : 23.333334
```

Same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. In the following example, it's explained using C function and as you know Objective-C supports C style functions also:

```
// function declaration
int func();

int main()
{
    // function call
    int i = func();
}

// function definition
int func()
{
    return 0;
}
```

Lvalues and Rvalues in Objective-C

There are two kinds of expressions in Objective-C:

- **lvalue:** Expressions that refer to a memory location is called "lvalue" expression. An lvalue may appear as either the left-hand or right-hand side of an assignment.

- **rvalue:** The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and cannot appear on the left-hand side. Following is a valid statement:

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error:

```
10 = 20;
```

7. CONSTANTS

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are also enumeration constants as well.

The **constants** are treated just like regular variables except that their values cannot be modified after their definition.

Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals:

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

Following are other examples of various types of Integer literals:

```
85         /* decimal */
0213       /* octal */
0x4b       /* hexadecimal */
30         /* int */
30u        /* unsigned int */
30l        /* long */
30ul       /* unsigned long */
```

Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals:

```
3.14159       /* Legal */
314159E-5L    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```

Character Constants

Character literals are enclosed in single quotes e.g., 'x' and can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C when they are proceeded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t). Here, you have a list of some of such escape sequence codes:

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character

\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh . . .	Hexadecimal number of one or more digits

Following is the example to show few escape sequence characters:

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

int main()
{
   NSLog(@"Hello\tWorld\n\n");

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-07 22:17:17.923 demo[17871] Hello      World
```

String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters. You can break a long line into multiple lines using string literals and separating them using whitespaces.

Here are some examples of string literals. All the three forms are identical strings.

```
"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"
```

Defining Constants

There are two simple ways in C to define constants:

- Using **#define** preprocessor.
- Using **const** keyword.

The #define Preprocessor

Following is the form to use #define preprocessor to define a constant:

```
#define identifier value
```

Following example explains it in detail:

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

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'
```

```
int main()
{

   int area;

   area = LENGTH * WIDTH;
   NSLog(@"value of area : %d", area);
   NSLog(@"%c", NEWLINE);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-07 22:18:16.637 demo[21460] value of area : 50
2013-09-07 22:18:16.638 demo[21460]
```

The const Keyword

You can use **const** prefix to declare constants with a specific type as follows:

```
const type variable = value;
```

Following example explains it in detail:

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

int main()
{
   const int  LENGTH = 10;
   const int  WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
```

```
   NSLog(@"value of area : %d", area);
   NSLog(@"%c", NEWLINE);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-07 22:19:24.780 demo[25621] value of area : 50
2013-09-07 22:19:24.781 demo[25621]
```

Note that it is a good programming practice to define constants in CAPITALS.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**