# General

* Dynamically-types language. Object defined with ‘id’ can hold any type of object and change (and checked) in run-time.

## Namespaces

Objective-C doesn’t support name spaces so instead, many build-in libraries use a 2-letters prefix to prevent name collisions.

The most common prefixed are:

* UI – UIKit classes and types
* CG – Core Graphics
* NS – Foundation and AppKit

It is a good practice to you add a prefix to your classes and types too. Even if you are not developing a framework but a normal app (which is often the case), just use letters from the app name to be sure to avoid collisions to code you might add later from a different source (like some open source code you might use in your project). There is no strict check from the compiler, so you can omit it, but it’s better not to.

## Files

* Has header files like C and C++ that define the interface.

# Objects

id myObject = nil;

The **id** type means that the variable can contain any type of object. The **nil** value is equivalent to what is called **null**

Specific-types object:

NSString \*aString = nil;

# Operators

## Ternary Operator

variable = [ setFunction: <condition> ? <value is condition is true> : <value if condition is false>];

Note: if the <condition> is verifying that an object is not nil and the <value if condition is true> is the same object – the <value if condition is true> can be removed to give:

Variable = [ setFunction: <condition> ?: <value if condition is false>];

(void)displayErrorMessage:(NSString \*)msg

{

NSString \*msgFullString;

if (msg != nil)

msgFullString = [NSString stringWithFormat:@"Message: %@", msg];

else

msgFullString = @"Message: Unknown error 1";

NSLog(@"%@", msgFullString);

}

# xCode (IDE)

build (command + b), and run (command + r).

## Debugging

* To see the value of a variable on the watch panel -> right click -> print description of variable {var name}