# General

* Dynamically-types language. Object defined with ‘id’ can hold any type of object and change (and checked) in run-time.
* C + stuff you can do with square brackets = Objective-C!
* Objective-C maintains all the syntax from C and just adds object-oriented features as a thin layer on top of the world of C code.
* <https://learnxinyminutes.com/docs/objective-c/>

## 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.

### Imports

// Imports the Foundation headers with #import

// Use <> to import global files (in general frameworks)

// Use "" to import local files (from project)

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

***#import "MyClass.h"***

// If you enable modules for iOS >= 7.0 or OS X >= 10.9 projects in

// Xcode 5 you can import frameworks like that:

@import Foundation;

### Main

// Your program's entry point is a function called

// main with an integer return type

**int** **main** (**int** argc, **const** **char** **\*** argv[])

{..}

### IO

// Use NSLog to print lines to the console

NSLog(@"Hello World!"); // Print the string "Hello World!"

// %@ is an object

// 'description' is a convention to display the value of the Objects

NSLog(@"%@ and %@", myObject1, [myObject2 description]); // prints => "(null) and (null)"

# Types

* The @ symbol is used in a few places in Objective-C, and basically it’s a way to signal that whatever it’s attached to is special to Objective-C and not part of regular C.
* square brackets are used to differentiate Objective-C constructs from plain old C code.

## Basic Types

* **int** – An integer value, i.e. a whole number (no decimals) that includes zero and negative numbers.
* **float** – A floating point value that includes as many decimal places as it can hold. Because the decimal place can change, or float, its important to know that these values may technically be imprecise. When precise decimals are needed, like for currency, we should use the NSDecimalNumber data type.
* **BOOL** – Short for “boolean”, this is a 1-bit “true” or “false” value that can only be in one of those states. The C language (and hence, Objective-C) treat 0 as “false” and 1 as “true”. As such, the following keywords can be used to represent true/false values: YES/NO, TRUE/FALSE, true/false, 1,0.
* **char** – A single character, such as the letter A or the symbol “#”. Note that lowercase and uppercase characters are different, so “a” and “A” are two different characters.
* **NSString** – String data is a bunch of characters strung together to make text, like a banner strung up at a party.
* **NSNumber** – This class is a lightweight “wrapper” class that gives object-oriented features to the primitive number types mentioned above (among others).

## Objects

### Dyncamic-typed object

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**

### Static-types object:

NSString \*aString = nil;

## Strings

// String

NSString **\***worldString **=** @"World";

// NSMutableString is a mutable version of the NSString object

NSMutableString **\***mutableString **=** [NSMutableString **stringWithString**:@"Hello"];

[mutableString **appendString**:@" World!"];

NSLog(@"%@", mutableString); // prints => "Hello World!"

# Functions/Methods/Messages

## Functions

* Defined like in C++:

void NSLog(NSString \*text) {  
 // Logging code  
}

* Functions can be called from anywhere without any reference to the class where they are defined. They simply act on the data passed into them.

## Methods

Methods are member functions of objects. They can be class-methods (static) or instance methods (on instances)

Method C++:

class Man extends Person {

public void cleanYourRoom( Date when, Broom what ){

this.yeahSure();

etc...

}

}

Is equivalent to objective-c:

@implementation Man : Person

- ( void ) cleanYourRoom: (Date \*) when withCleaningTool: (Broom \*) what

{

[self yeahSure];

.. use when and what parameters…

etc...

}

@end

And in general form:

[+/-] (returned type) nameOfFunctionPart1: (type of Parameter 1) nameOfParameter1 nameOfFunctionPart2: (type of Parameter 2) nameOfParameter2 …. {..}

* +/-:
  + + : a class method – static methods that can only be called on the class itself (not on instances of it).
  + - : an instance method
* The name of the method will be   
  nameOfFunctionPart1: nameOfFunctionPart2…

## Messages

The difference between calling a method on an object and sending it a message:

Any message can be sent to any object. It’s up to the object to handle the message or not. If it doesn’t handle the message, then nothing bad will happen; it will just return nil to the sender, or caller. This is different than the method model because, If a method is called on an object that doesn’t define the method, we end up with compile-time or run-time errors that stop us dead in our tracks.

send *messages* to *objects* (more on these later) in this manner:

mug makeEmpty

(The makeEmpty message is sent to the mug object.) In Objective-C we simply surround such actions with square brackets:

[mug makeEmpty]

Calling the method in C++:

Man man = new Man();

Date now = new Date();

Broom broom = new Broom();

Man.cleanYourRoom(now, broom);

Or:

Man man = new Man();

Man.cleanYourRoom(new Date(), new Broom());

And in objective-c, using a message:

Man \*aMan = [Man man];

Date \*now = [Date date];

Broom \*aBroom = [Broom broom];

[aMan cleanYouRoom:now withCleaningTool:broom];

Or:

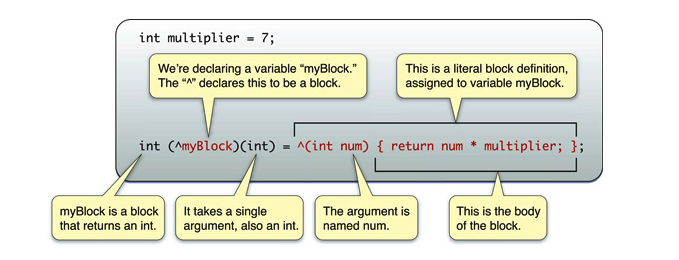
[aMan cleanYourRoom:[Date date] withCleaningTool:[Broom broom]];

Note: you can call the method directly in objective-c (like in C). However, then you don’t get the benefit of auto-resolving to nill if the method is not defined.

## Blocks == Anonymous Functions

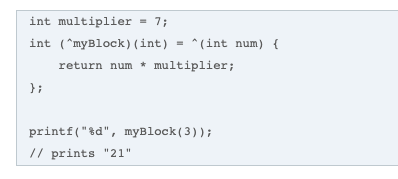
Defined inside the code (like in C#/java) with the syntax:

**^(**parameters..**){**function’s code..**}**



Notice that the block is able to make use of variables from the same scope in which it was defined.

If you declare a block as a variable, you can then use it just as you would a function:



# 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}
* To print to output  
  printf("My message!! Var value: %d", myVar);
* To print with NSLog(@”referrer params: %@”, params);
  + In xCode:   
    Product -> Scheme -> Edit Scheme -> Run -> Arguments -> Environment Variables
  + Add OS\_ACTIVITY\_MODE and set it to ‘default’