Lesson 4: Objective-C

INTRODUCTION

This lesson provides an overview of Objective-C. It is assumed that the reader is already familiar with at least one programming language such as C or C++. This overview focuses on some of the differences and terminologies specific to Objective-C. Objective-C is a cross-platform language that can be used with Mac, Linux/UNIX, or Windows. Most often, it is used with Mac.

LESSON OBJECTIVES

By the end of this lesson, the student will be able to:

1. Discriminate among a superclass, a class and a subclass.
2. Identify in a message, the instance variable, the method and whether there are arguments included.
3. Explain why comments are important in a program.
4. Identify the two Boolean values used in Objective-C.
5. Identify the differences between a class method and an instance method.
6. Explain what a pointer is and the benefit of using pointers in iOS applications.
7. Identify primitive data types.
8. Differentiate content and code that goes in the header (.h) file, the implementation (.m) file and the xib (.xib) (or storyboard) file.
9. Identify the use of **id** as a data type.
10. Identify the purpose of the viewDidLoad method.

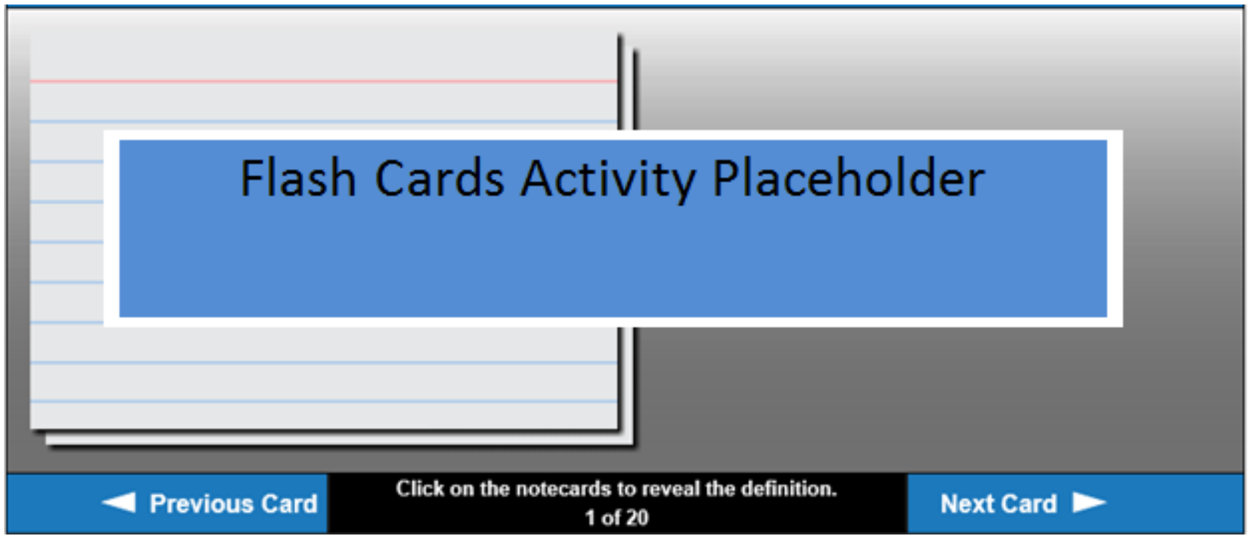
LEARNING SEQUENCE

Required Reading	Read the following: • Lesson 4: Objective-C
Resources	View the following: • *Xcode #8: Connecting the Storyboard to H Files* (4:37) • *View Xcode #9 - .H files, .M files, iBaction, Scope and more…*(6:16) • *Xcode #10 – Sprite Animatin using the AnimatedImages Class* (8:56) • *Xcode #11 – Scale Animation* (6:55) • *Xcode #12 – Random Numbers and Sounds* (5:16) Other resources: • Objective-C Classes & Objects

	• Using Objective-C Preprocessor Directives • Ry's Objective-C Tutorial > Protocols • Ry's Objective-C Tutorial > Blocks
Assignments	Complete the following: 1. Practice Exercise - Caption 2. Lesson 4 Quiz

KEY TERMS

As you read your lesson, pay close attention to the key terms and phrases listed throughout the lesson. These terms and concepts are important to your understanding of the information provided in the lesson.


Flash Cards Activity Placeholder
Previous Card
Click on the notecards to reveal the definition.
1 of 20
Next Card

INSTRUCTION

Getting Started with Objective-C

Objective-C is a superset of C, so that means that it is built on top of C. Objective-C adds object-oriented features to C and because of that, it is possible to compile any program written in C with an Objective-C compiler and C code can be included within an Objective-C class.

Objective-C is the native language used for Mac, iPhone, and iPad development. A prerequisite for Objective-C is that Xcode has been downloaded and installed from the Mac App Store onto a Mac system.

What is Object-Oriented Programming?

Object-Oriented Programming (OOP) is a programming language model made up of **objects**. An object is an instance of a class. A **class** defines an object's properties and its capabilities. OOP allows the programmer to break up code into objects. It is also self-contained, so details are not visible outside of the object making encapsulation possible. Read Objective-C Classes & Objects for more information.

Characteristics and Symbols

Objective-C is written differently from other languages especially when dealing with **methods** and messages. A method is a function that is part of a class. With Objective-C, objects communicate with messages; an object does not call a method which is common in other programming languages.

The "@" symbol indicates a compiler directive. Objective-C has its own preprocessor that processes @ directives before the actual compilation. The preprocessor searches for these special directives written by the developer and converts them to code that can then be handled by the compiler.

The "#" symbol indicates a preprocessor directive. Both @'s and #'s are going to be handled before the code is compiled.

Boolean is indicated by YES and NO (all capitals) rather than True and False (which is used in C).

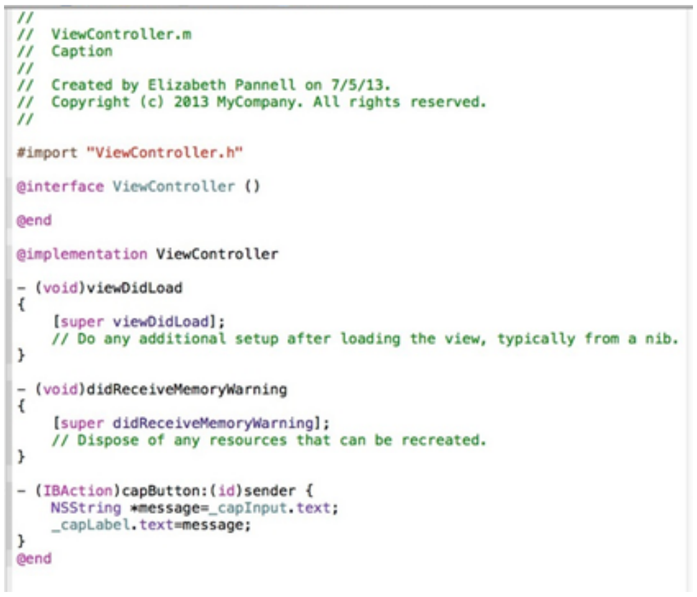
Common Compiler Directives

The @interface is used to describe an instance variable for a class in the header file and wherever methods are declared.

The @property declares variables and automates the setup of a getter/setter in the header file. To change the @property's default naming conventions, change the getter/setter method names with the getter= and setter= attributes.

The @implementation shows the compiler which class is being used followed by the code being implemented in an implementation file.

The @end is used with @interface or @implementation and indicates the end of the interface or implementation.



```
//
//  ViewController.m
//  Caption
//
//  Created by Elizabeth Pannell on 7/5/13.
//  Copyright (c) 2013 MyCompany. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)capButton:(id)sender {
    NSString *message=_capInput.text;
    _capLabel.text=message;
}
@end
```

Figure 1: Code showing common compiler directives.

Common Preprocessor Directives

Preprocessing is the first step in the compilation stage. For additional information, read Using Objective-C Preprocessor Directives.

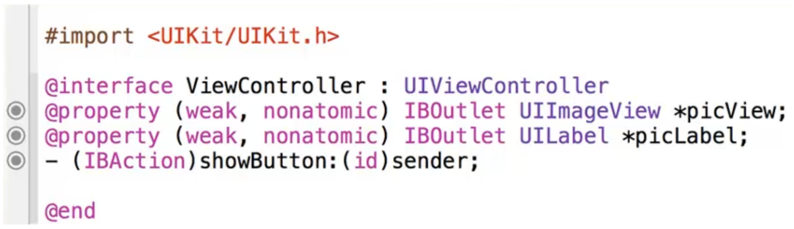
The #import <file.h> tells the program to replace a statement with the file.h contents. It only adds a header file once. Figure 1 uses the #import with ViewController.h to control and manage the View.

The #define is a preprocessor macro used to define constants and macros.

The #pragma mark is used to create sections in the code. It is very similar to bookmarking and allows the programmer to go to different sections of code.

Objective-C Classes

There is an interface area and an implementation area which are in separate files (in C and C++, both of these are in the same file). The header file (.h) is where the class properties and actions are declared. Figure 2 shows an example header (.h) file.



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

@interface ViewController : UIViewController
@property (weak, nonatomic) IBOutlet UIImageView *picView;
@property (weak, nonatomic) IBOutlet UILabel *picLabel;
- (IBAction)showButton:(id)sender;

@end
```

Figure 2 Header (.h) File Example

The #import pulls from the indicated file before the code compiles. The UI is a framework, so the #import statement is indicating the framework for the section. The **<** and **>** symbols indicate a reference to something in the language that does not need to be copied into the project; it is handled automatically.

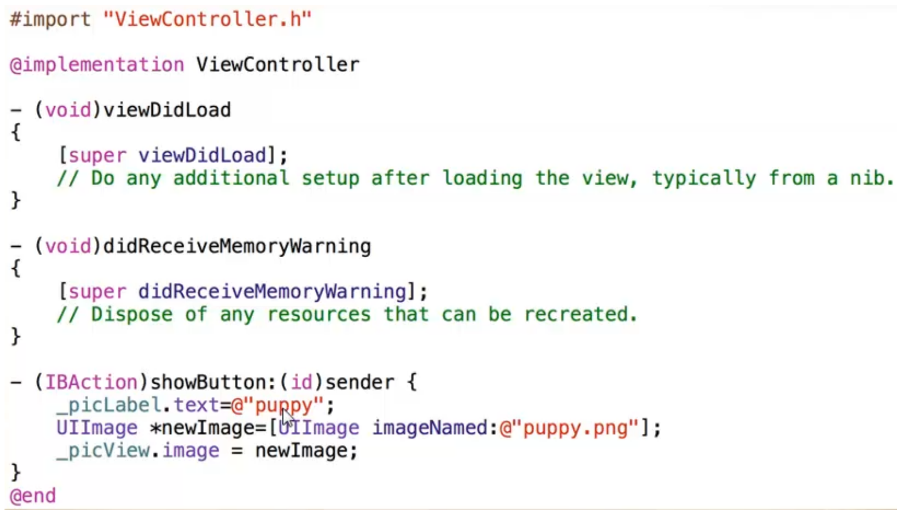
@interface indicates where the interface area starts: the name of the class is ViewController and the name of the **superclass** is UIViewController. A superclass is the parent class.

The two @property statements declare instance variables. IBOutlet tells Xcode that it will be tied to or used with something in the interface.

The – (IBAction) is an action or method which will be visible inside the Interface Builder section.

The end of the interface section is indicated by @end.

The implementation file (.m) is where the class properties and actions are implemented. Figure 3 shows an example of this file.



```
#import "ViewController.h"

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)showButton:(id)sender {
    _picLabel.text=@"puppy";
    UIImage *newImage=[UIImage imageNamed:@"puppy.png"];
    _picView.image = newImage;
}
@end
```

Figure 3 Implementation (.m) File Example

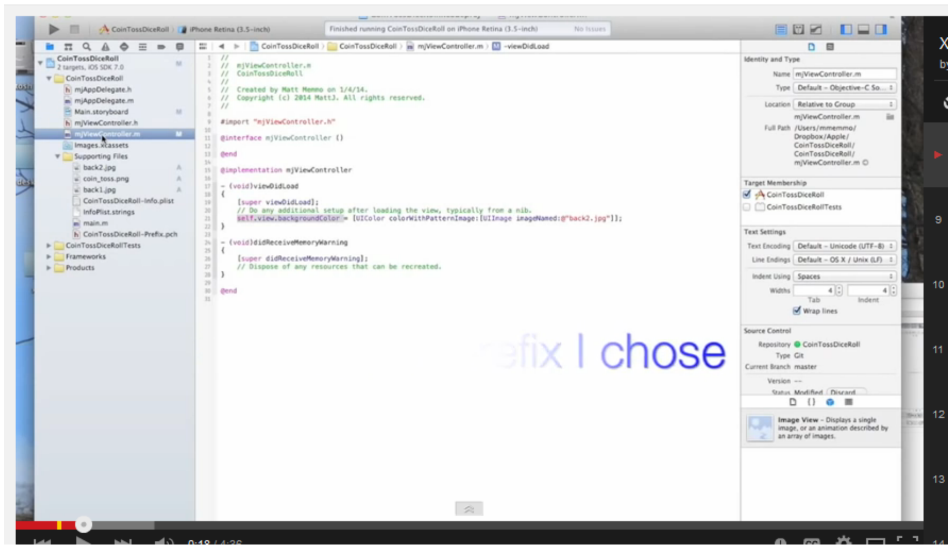
In this example, ViewController.h, the header file, is being imported. The quotations indicate that this file was created and is part of the project and not a library or framework within the language.

The implementation section starts with @implementation ViewController. ViewController is the name of the class. (void)viewDidLoad detects when the view loads and is used for setup (such as setting up an array to have it available); (void)didReceiveMemoryWarning allows tasks like saving or freeing up memory when a memory warning is received.

The (IBAction)showButton was created in the interface. The curly brace is used at the beginning and the end of the code. In this example, “puppy” is going to show as the label text and a **variable**, which stores a variable, is being created to hold the image, puppy.png. _picView loads the image into the image viewer.

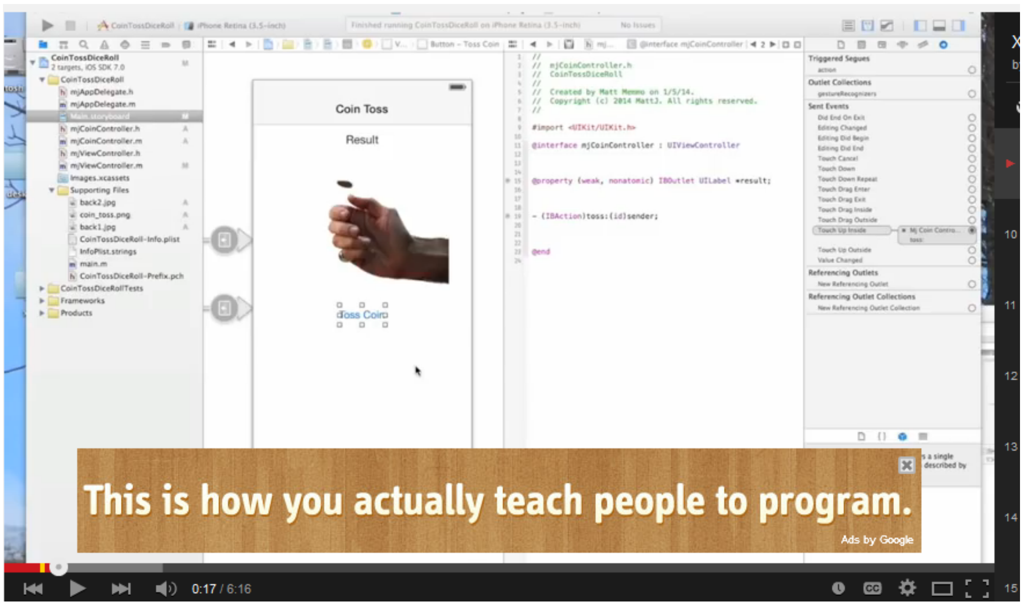
The implementation section is then ended with @end.

View *Xcode #8: Connecting the Storyboard to H Files* (4:37) to create files in Xcode and associate these files with a ViewController.



```
//
//  mjViewController.m
//  CoinTossDiceRoll
//
//  Created by Matt Memmo on 1/4/14.
//  Copyright (c) 2014 MattJ. All rights reserved.
//

#import "mjViewController.h"

@interface mjViewController ()

@end

@implementation mjViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"back2.jpg"]];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

For more information on .h, .m and IBAction, view *Xcode #9 - .H files, .M files, iBaction, Scope and more*…(6:16).



```
//
//  mjCoinController.h
//  CoinTossDiceRoll
//
//  Created by Matt Memmo on 1/5/14.
//  Copyright (c) 2014 MattJ. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface mjCoinController : UIViewController



@property (weak, nonatomic) IBOutlet UILabel *result;



- (IBAction)toss:(id)sender;



@end
```

Formatting

All statements end with a semi-colon (**;**). Statements can go over multiple lines and will ignore white space and indentions.

Comments are indicated by a double slash (**//**) for a single line. Look at the implementation file in Figure 3 to see an example of a single line comment. Use a slash star (**/***) to begin a multiline comment followed by a star slash (***/**) to end it.

The name of a variable or function is case sensitive and may NOT include special characters or spaces. A variable name can only have letters, numbers, underscores. A variable name cannot start with a number. For example, George and GeOrGe are different variables. An example of a valid name is _Puppy, but _Puppy& is invalid because of the special character, and 1Puppy is invalid because it begins with a number.

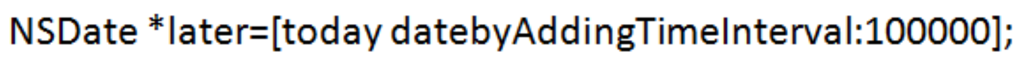
Pointers and Variable Creation

A **pointer** points to a memory location and is used in Objective-C to point to **instance variables**, or instances which are variables inside a class that holds values specific to that class. An object will have a pointer to it. The star (*) is used to indicate a pointer and it must be used when an instance variable is declared. After the instance variable is declared, it is not necessary to use the * any longer.

An **id** is a generic data type that is required when there is an instance variable that has a pointer. The data type for arguments and parameters on a method also needs to be indicated. By assigning an id, different types of objects can use the same method. An id does not require that the star (*) be used in front of it.

Messages

Instead of calling a function, Objective-C uses messages. The message must be enclosed in brackets **([ ])**. There must also be a pointer to the object receiving the message and the name of the method. Figure 4 provides an example.



```
NSDate *later=[today datebyAddingTimeInterval:100000];
```

Figure 4 Message Example

The *later indicates an instance variable. The class is NSDate. today is an instance variable (which was previously created) receiving the message datebyAddingTimeInterval:100000. The method is datebyAddingTimeInterval and the colon (:) indicates that there is one argument which is the value 100000.

Nesting messages will execute the message inside the inner bracket first and then proceed outward. The Apple-approved way of initializing is to nest. For example, the following code is not nested:

```
NSDate *now=[NSDate alloc];
[now init];
```

The preferred, nested version is as follows:

```
NSDate *now=[[NSDate alloc]init];
```

Protocols

A **protocol** focuses on creating a set of methods that perform a specific role. Different objects can use the protocol to fulfill that role.

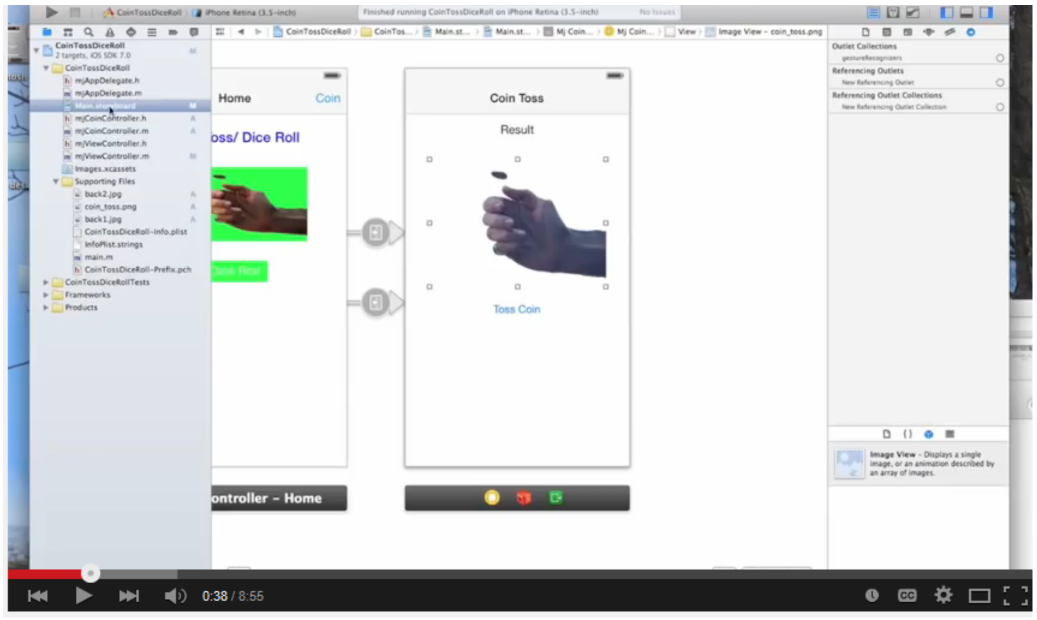
Use the @protocol to create a list of method declarations. Some of the methods will be required, others will be optional. If a class adopts the protocol, then all of the required methods must be used. Read the Protocols section of Ry's Objective-C Tutorial for an explanation on how protocols can be used to reduce redundant code.

Collection Class

A collection class holds a group of pointers. Examples of collection classes are NSArray, NSMutableArray, NSSet, NSMutableSet, NSDictionary, and NSMutableDictionary.

NSArry is the default in which all objects are assigned when created. NSMutableArray can create and assign objects at different points in the program which allows for much more manipulation.

In the video, *Xcode #10 – Sprite Animation using the AnimatedImages Class* (8:56), an array is used to create the animation.



Callbacks

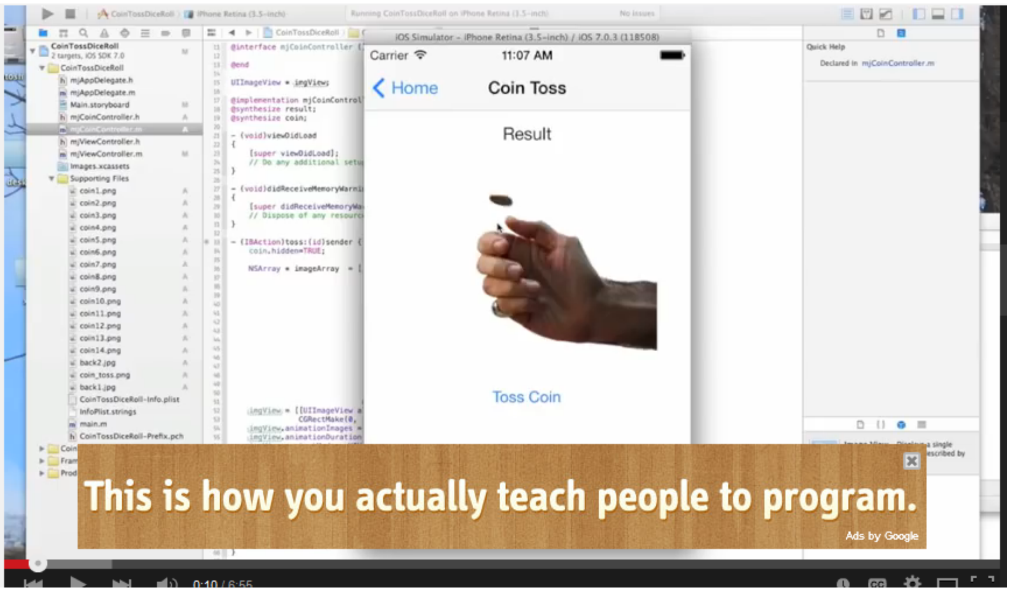
A **callback** is executable code that is passed as an argument to other code which will eventually return, or be called back. There are three different types of callbacks:

- Target-action: if something happens, a message is sent to the target object (this is the simplest of the three)
- Helper Objects: a delegate or data source is available if something occurs (a protocol is adopted an is waiting for something to happen)

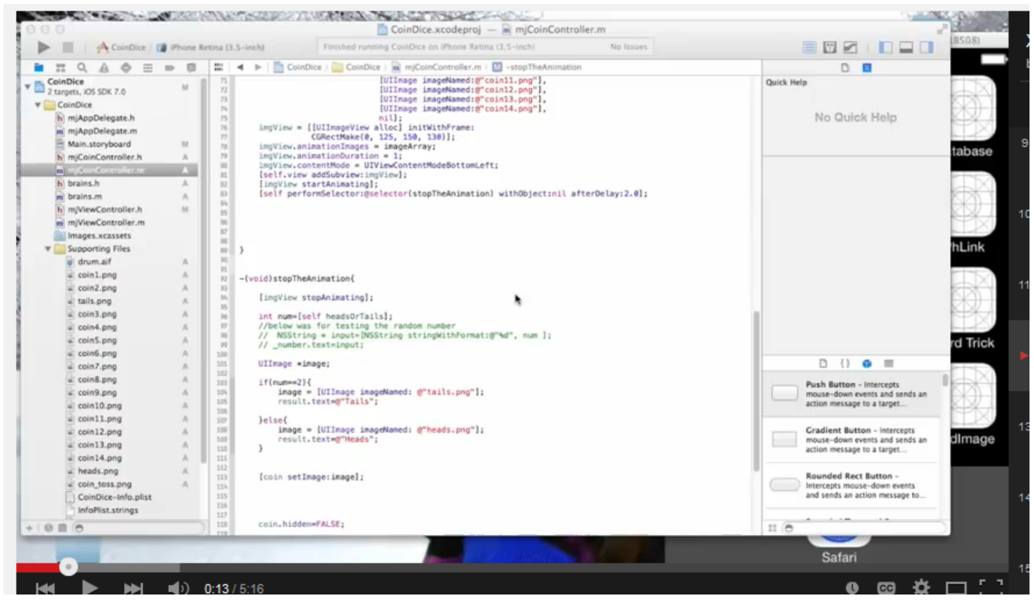
- Notifications: a notification center object is waiting for an event to occur. When the event happens, the notification center passes on the message.

Blocks are used to handle a problem with callbacks that are spread apart in the code (in other languages, this is known as anonymous functions, closures, or lambdas). A block allows the programmer to treat a chunk of code (like a function, for example) as if it was data. A block is indicated with a carat symbol (**^**). Read the section on Blocks at Ry's Objective-C Tutorial.

View *Xcode #11 – Scale Animation* (6:55) which demonstrates how to make an object slowly increase in size and *Xcode #12 – Random Numbers and Sounds* (5:16) to finish the coin toss app by adding in the sound effects.



```
@interface mjCoinController (
@end

UIImageView * imgView;

@implementation mjCoinControl
@synthesize result;
@synthesize coin;

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setu
}

- (void)didReceiveMemoryWarni
{
    [super didReceiveMemoryWa
    // Dispose of any resourc
}

- (IBAction)toss:(id)sender {
    coin.hidden=TRUE;

    NSArray * imageArray  = [

    imgView = [[UIImageView a
                 CGRectMake(0,
    imgView.animationImages =
    imgView.animationDuration
```



```
                              [UIImage imageNamed:@"coin11.png"],
                              [UIImage imageNamed:@"coin12.png"],
                              [UIImage imageNamed:@"coin13.png"],
                              [UIImage imageNamed:@"coin14.png"],
                              nil];
    imgView = [[UIImageView alloc] initWithFrame:
               CGRectMake(0, 125, 150, 130)];
    imgView.animationImages = imageArray;
    imgView.animationDuration = 1;
    imgView.contentMode = UIViewContentModeBottomLeft;
    [self.view addSubview:imgView];
    [imgView startAnimating];
    [self performSelector:@selector(stopTheAnimation) withObject:nil afterDelay:2.0];




}


-(void)stopTheAnimation{

    [imgView stopAnimating];

    int num=[self headsOrTails];
    //below was for testing the random number
    //  NSString * input=[NSString stringWithFormat:@"%d", num ];
    // _number.text=input;

    UIImage *image;

    if(num==2){
        image = [UIImage imageNamed: @"tails.png"];
        result.text=@"Tails";

    }else{
        image = [UIImage imageNamed: @"heads.png"];
        result.text=@"Heads";
    }


    [coin setImage:image];



    coin.hidden=FALSE;
```

SUMMARY

Objective-C is a superset of C, and is the language used for Mac, iPhone, and iPad development. Object-oriented programming breaks code into objects and encapsulation keeps details of the objects invisible outside the object. This lesson discussed the different components within Objective-C and how those components function.

ASSIGNMENTS

1. Practice Exercise - Caption
2. Lesson Quiz 4