Mobile Makers Academy

Pre-course Material, Summer 2013 iOS Immersive

All Rights Reserved. © Mobile Makers Academy

# Challenge 3: Get Your Hands Dirty

I hope everything went well with your last challenge! Be sure to send us your responses to the challenge questions if you haven't already!

Next up, find the lynda.com video series entitled: Objective-C Essential Training. Watch each video in the following list and answer to the correlated question. And again, send us your answers when you have completed them. As always, let us know if you get stuck!

## Section 1 - Getting Started

**1.1 Installing the tools 4m 42s**

Install Xcode and register as an Apple Developer.

**1.2 Creating your first application 11m 28s**

Create your first application using the same steps Simon describes in the video. Familiarize yourself with the Xcode environment, specifically notice how it can be manipulated to display different helper tools and how it will attempt to fill in your code as you type it.

**1.3 Updates to this course 3m 31s**

Why do you think it's important to be aware of the idiosyncrasies with older versions of Objective-C and to keep up with new features as they are added?

You may find yourself working in an older environment, relying on tutorials or code created in an older version or on a legacy app that requires you use older versions of code.

You also need to keep up with changes that are made to Xcode because Apple may introduce new methods or easier ways to accomplish tasks.

**Section 2 - Objective-C Basics**

**2.1 The Objective-C language 4m 11s**

How did Objective-C become the language to learn if you want to make apps for the iPhone and iPad?

Objective-C was adopted as the main language of Next Computer and they built everything including their NeXTStep operating system from it. After Apple acquired Next in 1996, they integrated NeXTStep into Mac OS X and Objective-C continues to be the base code for Apple’s computers and devices.

**2.2 The structure of an Objective-C program 6m 15s**

Create a new project. Go to the menu option Xcode/Preferences/Text\_Editing and make sure "Line Numbers" is checked in the section marked "Show." Then add comments describing the purpose of each auto-generated line in the main.m file. For example on Line 17 I would write: "NSLog(@"Hello, World!"); //instructs the console to output the phrase "Hello, World!")"

#import <Foundation/Foundation.h> // Points to the Foundation Framework

int main(int argc, const char \* argv[]) // This tells the program where to start.

// main is included in every project

{ // Opening statement block - the beginning of main

@autoreleasepool { // autoreleasepool deals with memory management

// insert code here...

NSLog(@"Hello, World!"); // Writes "Hello, World!" to the console

} // Closing statement block

return 0; // Final line of main and says "main was successful"

} // Final closing statement block

**2.3 Compiling and running your code 8m 37s**

Why might you build in one version of iOS but deploy in an older version?

You may be using the most up-to-date version but your users may be using older versions of iOS and you don’t want them to be excluded from using the app.

## Section 3 - Program Flow

**3.1 Logging messages to the command line 6m 7s**

Following the example in the video, write a program that calculates and outputs to the console the number of seconds in ten years. Copy and paste your code here.

// simple calculation for seconds. Does not accommodate leap year(s).

int secondsInAMinute = 60;

int minutesInAnHour = 60;

int hoursInADay = 24;

int daysInAYear = 365;

int secondsInTenYears = ((((secondsInAMinute \* minutesInAnHour) \* hoursInADay) \* daysInAYear) \* 10);

NSLog(@"Total seconds in 10 years: %i", secondsInTenYears);

**3.2 Writing conditional code 7m 1s**

Using Objective-C, create an integer variable called "day" that represents the days of the

week. Write an if statement that checks whether "day" is a weekend day. If the day is a

weekend day then have your program print a message saying "Have a nice weekend!" and if it's not, print a message saying "I hope you're having a good week!"

int day = 4;

if (day < 5) {

NSLog(@"I hope you're having a good week");

} else {

NSLog(@"Have a nice weekend");

}

**3.3 The switch statement 5m 58s**

Create a variable called "hurricaneCategory" and a switch statement that prints out a message describing a hurricane's category from 1-5.

int hurricaneCategory = 4;

switch (hurricaneCategory) {

case 1:

NSLog(@"The hurricane is a Category %i.", hurricaneCategory);

break;

case 2:

NSLog(@"The hurricane is a Category %i.", hurricaneCategory);

break;

case 3:

NSLog(@"The hurricane is a Category %i.", hurricaneCategory);

break;

case 4:

NSLog(@"The hurricane is a Category %i.", hurricaneCategory);

break;

case 5:

NSLog(@"The hurricane is a Category %i.", hurricaneCategory);

break;

default:

NSLog(@"This is a tropical storm.");

break;

}

**3.4 Code snippets 5m 15s**

Grab a code snippet, indent it to match the indent of your project, then add comments to it, then select the entire snippet you just modified and save it as your own code snippet. Time yourself and record how many seconds it takes you to do all this.

40. 9 seconds

**3.5 Operators and expressions 11m 8s**

List the 6 types of operators described in this video. Provide their name, a description of their meaning, and the syntax you would use to execute them. What code snippet does the ternary operator replace?

**Arithmetic:** These operators perform math calculations on variables.

Addition (+): Adds numbers or concatenates strings: (a + b) = c;

Subtraction (-): Subtracts numbers: (a – b) = c;

Multiplication (\*): Multiplies numbers: (a \* b) = c;

Division (/): Divides numbers: (a / b) = c;

Assignment (=): Assigns the value to a variable or operation: int c = 5;

Shortcuts:

pointTotal = pointTotal + 1; can be written as pointTotal += 1;

Also note operator precedence is enforced.

**Comparison:**  Used to compare variables and returns a true or false result

Equal to: (a == b)

Does not equal: (a != b)

Greater than: (a > b)

Less than: (a < b)

Greater than or equal to: (a >= b)

Less than or equal to: (a <= b)

**Logical And/Or:** Used to determine if multiple conditions are true or false

And: if (a == b && c == d)

Or: if (a == b || c == d)

**Modulus:** Used to calculate the remainder (NOT the result)

%: int i = 21 % 4; // result is 1

Useful for determining odd/even numbers (% 2)

Only works on integers

**Increment/Decrement:** Adding 1 or subtracting 1 from a value

Prefix Increment: ++i: adds one to the value for i

Postfix Increment: i++: adds one to the value for i

Prefix Decrement: --i: subtracts one from the value for i

Postfix Decrement: i--: subtracts one from the value for i

See next operator for details on prefix/postfix

**Prefix/Postfix:** Determines when the increment or decrement should occur

++i: The increment should occur before the line is executed

i++: The increment should occur after the line is executed

--i: The decrement should occur before the line is executed

i--: The decrement should occur after the line is executed

**Ternary:** Uses three elements to return a single value or result

Condition ? true : false;

If a certain condition is true, do one thing : if the condition is false, do another

Used very similarly to an if…else statement

**3.6 Loops 8m 53s**

CHALLENGE: Think of a scenario while using a mobile app that might require you to use a "continue" statement in the middle of a loop.

When displaying the calendar you only want to display weekdays.

**3.7 Functions 10m 16s**

What is a function? What is a function prototype? What are the purposes of each? What are the rules for when and how you can call a function?

A function is a chunk of related code that is contained within a start and close block and given a name.

A function prototype allows the program to compile be describing the function early in execution. The function must still be defined, but including the function prototype early in execution allows the function to be called (in viewDidLoad, for example) prior to it being defined (at the end of the implementation file).

## Section 4 - Variables

**4.1 Data types 7m 6s**

What are the primitive data types in Objective-C? Why did Apple add a set of classes to handle other data types?

* Integer : int
* Float : float
* Double : double
* Character : char
* Boolean: BOOL

Using classes to deal with other data types makes it easier to work with that data.

**4.2 Working with numbers 9m 33s**

Make a table of Objective-C primitive data types. Add numeric data types and their properties to this table.

|  |  |  |  |
| --- | --- | --- | --- |
| Integer | int | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| 32-bit Long Integer | long int | 32 bit | 4 bytes (same as an int) |
| 64-bit Long Integer | long int | 64 bit | 8 bytes |
| Long Long Integer | long long int |  | 8 bytes |
| Short Integer | short int | 2 bytes | -32,767 to 32,768 |
| float | float |  | be ‘explicit’ and use the f (i.e. 7.2f;) |
| double | double |  |  |
| Character | char |  | Represents the ASCII character |
| Boolean | BOOL | YES/NO | Returns a true/false or yes/no result |

**4.3 Working with characters 4m 39s**

Add char and BOOL (the character data types) to your table created above.

**4.4 Variable scope 8m 6s**

Describe in your own words what the scope of a variable is in Objective-C

Scope defines where a variable is visible and accessible. By default, variable scope is limited to the statement block where the variable is defined (i.e. a local variable). To make the variable scope global (i.e. available to every function), you define the variable outside all the functions.

**4.5 Enumerations 3m 35s**

What does "enum" allow you to do?

Using an enumerated data type (i.e. enum) allows you to define a restricted range of possible values for a variable with words and then use the enum value as an integer (You can also assign a numeric value to each word within the enum definition: { red = 5, green = 10, blue = 15 };). You can use the enum as a data type in conjunction with an integer variable and set it equal to one of the words defined in the enum. For example:

enum primaryColors { red, green, blue ];

enum primaryColors bradsFavoriteColor = blue;

enum primaryColors daniellesFavoriteColor = green;

Using an enum is useful in programming because you don’t have to remember what value you assigned to each color (for example), you simply need to enumerate the variable and assign it the more intuitive (and easier to remember) word descriptor.

Enums are stored as integers so you can use %i to write it out.

**4.6 Using typedef 2m 17s**

When would you define your own data type versus use an enum?

typedef allows you to assign a new name to an existing data type and then create variables of that type. It is mostly used to shorten code that is otherwise clunky to write.

**4.7 Preprocessor directives 5m 56s**

Describe the three common preprocessor directives, #import, #define, and #if DEBUG. Come up with one example where you would use each.

Causes something simple to happen before our code is executed.

#import: Tells the compiler to find the referenced file and include that (plus any files that referenced file imports) when compiling.

#define: Allows us to create a shortcut or macro to a value and give it a name. You can then reference that value using its name instead of writing out the value. You cannot change the defined value within the program, it must be manually changed. The advantage of using define is it gives you one place to store values and it makes it easier to find if the value needs to be changed.

#if DEBUG: Used in conjunction with the #endif closing remark. This essentially asks the application if it is being compiled in debug mode. If true, then execute the code that is contained within the remarks.

**4.8 Working with strings 7m 52s**

Define the same string using both NSString and C-style string syntax. Describe the purpose behind each part of your definition.

C:

string message = “Hello”;

type name value

Objective-C:

NSString \* myString = @”Hello”;

type pointer name value

## Section 5 - Classes

**5.1 Introduction to object orientation 7m 36s**

Create an encapsulated (including generalized attributes and behavior) description of a "mobileMakersParticipant" class. Instantiate a single object representing yourself as a member of this class.

mobileMakersParticipant \*Brad = [[mobileMakersParticipant alloc] init];

**5.2 Using objects and pointers 6m 38s**

What is the pointer's role in instantiating an object from a class? How is a pointer different than a primitive?

A pointer references the memory address where the object exists. With primitives, the required memory is automatically allocated and the variable holds the memory directly. This can be done because the memory requirements for primitives are already known whereas the program doesn’t know how much memory is required to hold the object.

**5.3 Messages and methods 6m 44s**

What is the main difference between Objective-C's messages and method calls in other languages? How can this difference be seen as an advantage while programming?

Objective-C doesn’t use dot syntax for messages and method calls. Instead it relies on square brackets. This can be an advantage because it lends legibility to the code and makes the program easier to read.

**5.4 Using existing classes in the foundation framework 8m 40s**

What's the difference between a class method and an instance method? EXPLORE: Try typing "NSD…" into your code window. Use the autofill feature and select a single class name that starts with those three letters. Once the name has been auto-completed, use the handy shortcut (Option + click) and investigate the class whose name just got printed to the screen. Examine the task list for this class. Do this a few more times until you're familiar with the process, or until you've exhausted your curiosity, whichever comes last.

Class method: indicated with a plus (+) sign. When calling the method, Objective-C works on the entire class.

Instance method: indicated with a minus (-) sign. When calling the method, Objective-C will work on an instance of the object and not the class.

## Section 6 - Memory Management

**6.1 What's new with memory management? 1m 45s**

Let it soak in. No questions for this one.

**6.2 Memory management in Objective-C 6m 58s**

What is the relationship between a pointer to an object, a block of memory, and the owning and releasing process. Can you come up with an analogy for this relationship?

The pointer references a block a memory that the device will reserve for the object. When you create an object, you own it and have reserved a space in memory for it. If you are no longer using that object, you must release it so the device can then clear that memory address and make that space available for another object.

An analogy might be buying a condo. Early in the purchasing process, you may put down earnest money to reserves your right to buy that condo (the pointer). If you buy, you occupy that unit (memory) and own it. If you decide not to buy or eventually sell the condo, the unit is once again free and can be owned by someone else.

**6.3 Object creation 7m 31s**

What does the new method do when used to create an object instance of a class? Why do we avoid using this method? How long is an object's lifetime?

The “new” method allocates memory and initializes the object. We often don’t allocate and initialize with the “new” method because there are multiple ways to initialize an object. For example, initWithName or initWithContentsOfFile.

An object’s lifetime is the life of the program unless it is released at some time during the execution of the program (i.e. release).

**6.4 Using autorelease pools 5m 14s**

How does the autorelease pool work? How and when can you use it deliberately?

The autorelease pool is a list of objects that need release called on them at some point. Autorelease tells the application to release the memory at the end of an event loop like touching a button.

Autorelease can be used deliberately when you create an object. Immediately after you create the object you autorelease and then return the object. This way, the program knows that at some point in the future it will be releasing the object.

**6.5 Apple autoreleased objects 3m 39s**

What does NARC stand for? Why is it important to remember this?

New Alloc Retain Copy

If you scan your code for any of those key words, that will tell you how many release statements you need.

**6.6 Introduction to Automatic Reference Counting (ARC) 4m 43s**

What does ARC save us from having to do? How does it keep us from having to make this extra effort?

ARC manages memory within the application and releases objects as they are no longer needed. The compiler (LLVM) scans through your code and synthesizes the retain and release calls as they are needed.

**6.7 What ARC manages 2m 42s**

What are the differences between ARC and garbage collection? What makes these differences advantageous?

ARC is predictable and runs at compile time not at runtime like garbage collection. Running at compile time is more advantageous because you can avoid the slow downs that some apps experience due the garbage collector cleaning memory at random during operation.

**6.8 The rules of ARC 4m 20s**

Why can you not release or dealloc memory when working with ARC?

Because ARC manages memory usage within the app, you cannot retain, release, autorelease or dealloc in programs that utilize ARC. Xcode will immediately complain and throw an error if you attempt to use any of these methods.

## Section 7 - Custom Classes

**7.1 Creating your own classes 14m 1s**

What are the two different sections used to create a class? What do they hold and what files are they placed in?

**CHALLENGE:** Create a Tweet class for a twitter style app.

**7.2 Defining methods 8m 36s**

**CHALLENGE:** Define what should get passed in and what should get returned by each of your methods in your Tweet class above.

**7.3 Defining properties 7m 21s**

How did Objective-C programmers handle instance variables before 2012? How are they handled now? What got easier and what got obscured?

**7.4 Defining initializers 12m 30s**

What are initializers and why do we need to use them? Describe a situations when you can rely on the standard init method and when you have to create your own custom initializer.

**7.5 Using dealloc 5m 33s**

Why can we have a dealloc method in a class when using ARC, but we can't call dealloc manually oursevles when using ARC?

## Section 8 - Collections

**8.1 Working with C-style arrays 7m 12s**

What are the three constraints when using C-style arrays? Create a C-style array that holds the days of the week.

**8.2 Working with Objective-C array objects 8m 0s**

What is the difference between a mutable and an immutable array? CHALLENGE: Create an immutable array containing the days of the week. Create a mutable array that contains the days of the week that you will be at Mobile Makers. Add the days of the week from the immutable array to the mutable array.

**8.3 Using dictionaries 5m 55s**

Create a dictionary that lists five or more events in your life and the accompanying year (or date if you want to get fancy) of the event.

**8.4 Fast enumeration 3m 27s**

Use fast enumeration to log the timeline of the life events you described above to the console.

## Section 9 - File Management

**9.1 Introduction to file management in Objective-C 6m 44s**

What can you do with files using the methods you are aware of that are available in Objective C's Foundation class.

**9.2 Working with paths and URLs 7m 17s**

What are the three parts of a URL? What are the advantages to using NSURL?

**9.3 Reading and writing strings 4m 38s**

What would be a reason you would want to write a string to disk instead of just keeping it memory?

**9.4 Archiving objects 12m 41s**

Why would you want to archive an object instead of writing the data to disk using the techniques discussed previously?

## Section 10 - More Complex Classes

**10.1 Inheritance and NSObject 8m 13s**

How can you determine what methods you're inheriting from a super class? How do you overide a method inherited from a super class?

**10.2 Extending classes with categories 6m 31s**

What is the difference between a category and an inheritance? What are the limitations of using a category?

**10.3 Defining protocols 5m 14s**

How are protocols useful?

**10.4 Dynamic typing 11m 33s**

What are the advantages and disadvantages to dynamic typing?