**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 idiosyncracies with older versions of

Objective-C and to keep up with new features as they are added?

To be able to read/understand/update old code, and to take advantage of/implement/understand new features.

**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?

Apple bought Next, and rolled their Obj-C-based OS into Mac OS X, the basis for their OS

**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!")"

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

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

The iDevice may not be able to run newer versions

**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.

int seconds = 60;

int minutes = 60;

int hours = 24;

int days = 365;

int secondsIn10Years = seconds \* minutes \* hours \* days \* 10;

NSLog(@"There are %i seconds in 10 years", secondsIn10Years);

**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 = 0;

if (day < 1 || day > 5) {

NSLog(@"Have a nice weekend!");

} else {

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

}

**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(@"Meh");

break;

case 2:

NSLog(@"OK");

break;

case 3:

NSLog(@"Now we're talking!");

break;

case 4:

NSLog(@"Yikes!");

break;

case 5:

NSLog(@"Hellllppp!");

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.

150

**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: a+b, a/b

Assignment operator-assigns value to variable: var = 5

Comparison operators-compares 2 things, evaluated T/F: 6 >= 7

Logical And/Or-used in evaluation of multiple conditions: a > b && c == d

Modulus-provides remainder after division: 10 % 3 = 1

Increment: a+=b <=> a = a + b

Ternary-replaces if/else statement: a > 5 ? b : c

**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.

Playing song playlist but want to skip anything by Yanni

**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?

Function is a chunk of related code wrapped-up and named. It is simpler and more organized to work with these reusable modular code groupings.

A function prototype is a description of what a function returns and any parameters it may accepts. It is described as very useful in the object-oriented side of C.

Function should be described (declared?) before main, defined before or after main, and then called from main.

**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?

int. float, double, char, BOOL

Apple added classes to make it easier to work with strings, dates, images etc.

**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.

|  |  |
| --- | --- |
| Int | Whole numbers, signed or unsigned to adjust range, 4 bytes  long long 8 bytes; short 2 bytes |
| float | Decimals, 4 bytes, good practice to write as 4.33f |
| double | Decimals, 8 bytes, in literal will be assumed double unless ‘f’ included as above |
| char | Represented as ‘c’ <<any character in ASSCI character sets, 1 byte |
| BOOL | Represented as YES or NO |

**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

The range over which it has dominion.

**4.5 Enumerations 3m 35s**

What does "enum" allow you to do?

Define a data type that can restrict a variable to a set of values

**4.6 Using typedef 2m 17s**

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

If you’re using the enum a lot or it much clearer to move it into a typedef

**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.

#import is a way to include the contents of another file in yours, such as when you include the obj-C Foundation.h file in your code.

#define allows you to create predefined macro shortcuts such as a static value like pi that will be used in your code.

#if DEBUG allows you to have conditional code that is only included when running your program in debug mode. So you can have diagnostic messages that are automatically stripped-out when running a release version of your code

**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.

string x = “yes”

NSString \* x = @”yes”

NSString(type) \*(makes variable x a pointer) a = @(lets compiler know is NSString) ”(represent a string)yes”

**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 memberof this class.

MobileMakersParticipant Class: private: name, skillLevel, iDevice, learn(), create(), teach()

MobileMakersParticipant \* name = @”Norm”;

**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 is a variable that holds the address of the object.

A primitive has a fixed memory allocation and the variable holds the value directly, the pointer simply holds the memory address of 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?

With Obj-C, in calls to methods with parameters the method name is split so that each parameter is associated with a specific part of the method name. This affords better clarity and understanding of what is in the script.

**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.

They are designed to work with a class or with an instance.

**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?

Pointer references memory block, and if you create or use object in memory you need to release it when done or problems may ensue.

If you manage an apartment building and someone moves out and you don’t denote that you will attempt to interact with that person but they are no longer there, and you think the unit is occupied so you can’t use it for rental.

**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?

Allocates memory, returns address and puts in pointer, initializes instance

Lifetime is until you release object and retained count is 0, or obj-C runtime ascertains idleness and reclaims space.

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

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

Takes all objects in pool and reduces retain counts to 0 and releases memory when drained. Use it by calling autorelease on object instead of release; need to use when method returns the object cause “release” won’t work when used either before or after the return

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

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

New, alloc, retain, copy. Objects created with these methods must be released from memory.

**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?

Keeping track of retain counts and what needs to be released. The compiler now keeps track of this.

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

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

differences advantageous?

Garbage collection is done during runtime, and adds a process that occurs nondeterministically. ARC is code added in compiler and replaces code that would’ve been added manually, so you don’t slow down runtime.

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

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

ARC is already managing this and you’d interfere with this.

**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.

The interface is placed in the .h file tells what properties and methods are available.

The implementation is in the .m file contains the code that describes how everything is done.

Tweet.h

@property NSString \*acctName;

@property NSString \*acctFollowing;

@property NSString \*acctFollower;

@property NSString \*message;

@property static int charLimit;

sendMessage;

receiveMessage;

**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.

-(void) sendMessage: (NSString \*) message;

-(void) receiveMessage: (NSString \*) message;

**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?

They used to be defined in interface header file, along with accessor methods, which were then defined in implementation file. Then @properties took care of all this in interface, and until XCode 4.4 needed @synthesize to handle implementation. Now you just need to define properties in interface file. Now the getter and setter methods, and their implementations, have been 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.

Methods that ensure new objects are created in a valid state, and if desired, it’s internal values are set to specified amounts. You can rely on standard init when all objects are instantiated with same property values, but when you want to be able to customize these you’ll need custom inits.

**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?

You may need to ensure your object is no longer holding on to a resource, and you’d then need to write dealloc method. But ARC handles the call to this method during runtime, and you’d screw this up if you did it yourself.

**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.

Don’t try to access array elements not allocated to array; can’t change size of array; can’t mix types.

NSString \*daysWeek[] = {@”Sunday”, @"Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday", @"Saturday"};

**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.

You can add elements to a mutable array.

NSArray \*daysWeek = [NSArray arrayWithObjects:@"Sunday", @"Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday", @"Saturday", nil];

NSMutableArray \*daysWeekMM = [NSMutableArray arrayWithObjects:@"Monday", @"Tuesday", @"Wednesday", @"Thursday", nil];

[daysWeekMM addObject:daysWeek];

**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.

NSDictionary \*events = [NSDictionary dictionaryWithObjectsAndKeys:

@"walked on Moon", @"1969",

@"babysat for Mark Zuckerberg", @"1990",

@"was resurrected", @"33",

@"shot bin Laden", @"2011",

@"baked 1st loaf no-knead bread", @"2005",

nil];

**8.4 Fast enumeration 3m 27s**

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

console.

        for (NSString \*date in events) {  
            NSLog(@"In %@, I %@.\n", date, [events objectForKey:date]);  
        }

**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.

Ensure it exists, copy, move, delete, check attributes.

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

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

Scheme, domain, path. Many more classes use URL methods, URL object operations are faster than string paths, and NSURL formation can trap errors

**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?

So it is saved for future reference, not just during runtime.

**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?

It would be very tedious, inefficient, disorganized and error-prone to do the latter.

**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?

Check the Reference Library for that class.

In implementation file for your class, write method with same signature as one in superclass.

**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?

A category adds methods to an existing class; inheritance provides methods from one class to another.

A category can only add methods, not add instance variables.

**10.3 Defining protocols 5m 14s**

How are protocols useful?

By following them you gain access to a boatload of functionalities

**10.4 Dynamic typing 11m 33s**

What are the advantages and disadvantages to dynamic typing?

Advantages: lets you work with objects without knowing in advance of runtime what their type is

Disadvantages: need more script; compiler won’t catch issues as well as with static typing