**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?**

There's a lot of code that was made during the older versions, and so to ensure that older apps and code compile with the new stuff, we have to know what they are.  New features are important to keep up to date with because they are the best practices, and code development will be optimized for these new features from Apple and it's SDK.

**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?**

Steve Job's company made an OS called NeXTStep which was built on Objective-C.  Apple acquired NexT and integrated their OS into Mac OSX which is the OS for their computers, which is also what the iOS layer was built on top of.  Objective-C is the language everything has been made in.

**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?**

You want to build an app with the most current type of code but want to deploy on older version because not every potential user has the most recent OS.  Many users lag behind getting their computer updated, so to ensure that the most number of people can use your program, you may deploy an older version.

**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 years = 10;

        int days = 365;

        int hours = 24;

        int minutes = 60;

        int minutesIn10Year = years\*days\*hours\*minutes;

        NSLog(@"There are %i minutes in a year.", minutesIn10Year);

**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 == 0) || (day==6)){

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

        } else {

            NSLog(@"I hope you have 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.**

**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.**

**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?**

There is the mathematical operators - add, subtract, divide, and multiply.

There is the modulus operator, which gives you the reminder of a division.

There are the logical operators - AND / OR.

There is the ternary operator, which condenses an if-else statement.

The ternary operator replaces the if-else code snippet.

**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.**

If you only want to show even outputs, this could be a case where it passes over writing out the odd cases.

**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 program that has a return type and is passed parameters to execute some code and possibly return a value.  A function prototype is the function name and it's return type without any other information.  This is used so the program knows the function exists when compiling and can access the actual implementation later.  A function can only be called if it's prototype has been defined, or if the entire function is defined before the main where it is called.

**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?**

The primitive data types are int, float, double, char, and BOOL.  Apple created a set of classes for other data types because without them it would be very inefficient, and so there are NS classes for strings and dates, for example.

**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.**

The data types are int, long, long long, and short for integer values.  For values with decimals, they are float and double.  Ints are 4 bytes, long are 4 or 8, long long are 8, and short are 2.  Float is 4 bytes, while double is 8.

Char are 1 byte.  BOOL are 1 byte.

**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 scope of a variable in Objective-C is within the curly brackets of the area it is in.  If a variable is defined in the main method, it is accessible to all loops inside of it, but not to a function outside of main.  Global variables are ones defined outside of main or other methods in the actual class itself that can be accessed and changed by all functions in the class.  Variables can be accessed while digging deeper, not by rising out, similar to inheritance in some ways.

**4.5 Enumerations 3m 35s**

**What does "enum" allow you to do?**

Enum allows you to create a data type that has a limited option of values, instead of being all ints, floats, or characters.  This is useful for naming conventions and for easier to understand data.

**4.6 Using typedef 2m 17s**

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

I would define my own data type when it will be used repeatedly or very often in my program over several different classes.  An enum would be useful for a specific instance where a new data type isn't necessary, or I want to specifically limit it's use.

**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.**

The #import is used to bring in other frameworks and libraries into the program, which is useful for importing the Math library that has math functions in it.  The #define is used to define things that cannot be changed in the program, but can be used and added on to and defined as other changeable variables.  This is useful for constants such as PI or a sum for a specific test.  The #if DEBUG is an if statement that runs if the program is in debug mode and executes it code, which is good to simply test the state that the user is in.

**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 test = "test";

NSString \*test = @"test";

For the normal C-style definition, there is a string type, a variable name, and a value.  For the NSString, NSString is the object name, test is the variable name, and "test" is the value.  The pointer is a description that test points to a type of object, NSString.  The @ symbol is how strings are printed, but it actually just indicates an object, not specifically a string.

**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.**

class mobileMakersParticipant {

NSString name;

int age;

code();

}

mobileMakersParticpant \*Kabir = [[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?**

The pointer specifics the object name to the the class name, saying it is an instance of that class.  A pointer gives the location of the object, whereas primitive types contain the actual value itself.

**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?**

Other languages use dot syntax, where it is the object name, dot, method name, followed by the parameters in parenthesis.  Objective-C, however, has a bracket notation, with an open bracket, object name, space, method name, parameter, and parameter names and parameter separated by colons.  This is an advantage because it is clear in the method name what parameters are passed and what they are for.

**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.**

**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?**

A pointer says a created object is an object of a certain type.  The pointer points to a block of memory where the actual value is contained.  At that point the memory is owned.  When the instance is done being used, the developer should release it, so that way the value stored at that memory block is destroyed.

**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 the space in the heap for the object and then initializes it with every attributes and methods. We avoid this because often times objects will have several different types of init methods and we want to explicitly initialize an object with certain parameters.  An object's lifetime is unlimited until the end of the program, which is why releasing it is important.

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

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

Autorelease pool adds the object to a pool of other objects that aren't released right away, but are instead gathered in a pool.  At the end of an object cycle, the pool then calls release on all of the objects at once.  This can be used when you are returning an instance in a method and have to release the memory.  This way you can add it to the pool but the instance still exists, return the instance, and then close out of the function without a memory leak.

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

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

NARC stands for new, alloc, retain, and copy, which are 4 ways to create a new object directly.  It is important to remember this because for every object that has any value of NARC in it, those have to be released directly at the end of the program.

**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 saves us from conventional memory management, and we do not have to release or autorelease any instances since it does it for us now.  The compiler runs through our code and finds which instances aren't being used anymore and releases it for us, saving us that extra effort.

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

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

**differences advantageous?**

Garbage collection runs at run-time, requires a garbage collector that runs during the app, and is nondeterministic/random, while ARC runs at compile time, does not need a memory layer in runtime, and is predicable for when it happens.  ARC ensures the app does not slow down or get laggy if it runs in the middle of an app.

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

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

ARC does it for you, so you aren't able to do it.

**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 two different sections are the interface and the implementation files.  The interface is the .h file while the implementation is the .m.  The interface file holds the attributes and the method names, while the implementation file actually has the functionality and uses.

Challenge:

class Tweet : NSObject {

@property NSString \*username;

@property NSString \*tweetText;

@property NSString \*hashtag;

-(void)sendTweet;

-(void)shareTweet;

-(void)reTweet;

}

**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.**

sendTwet

**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?**

Before 2012 they would define instance variables inside curly brackets in the interface file.  They would then also define getter and setter methods in the interface file, as well as defining the methods in the implementation file.  Now, instance methods are defined as properties in the interface file, and so the getter/setter methods are no longer needed. There was a synthesize property in the implementation file to define the getter/setter for us, but since it ties to property those aren't needed anymore.  It's easier to get instance variables up and running, but now it is less clear what they are.

**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.**

Initializers create an object instance, and they can be used to create classes with pre-filled attributes during initialization.  If you are just creating an NSDate instance, you can just use the classic init.  But if you are creating a custom object of a person with several attributes, it may be more efficient to create a 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?**

Dealloc is already called so we can't call it manually but we can create one if it has to do other things like closing a database connection.

**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.**

The 3 constraints are: no bounds checking, arrays are fixed size, and you can't mix types.

NSString \*days[] = {@"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.**

Mutable arrays can be changed, as in objects can be removed, added, or inserted.

**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.**

I can move files, rename files, get it's attributes, find if it exists, remove an item, and a number of other things.

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

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

The 3 parts are the scheme, domain, and path.  The advantages of NSURL is that it is faster than string paths, it can trap errors, and it is used by more classes.

**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?**

If you want to store it for later use, it ensures the data is kept safely instead of accidentally destroyed on the heap.

**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?**

This ensures we can access different parts of the data as we want, and it is a safer way to store data encoded.

**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?**

Inherited methods can be seen from the documentation.  You can override a method by providing your own version with the same signature as the super class method.

**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?**

Categories add methods to classes, while inheritance pulls in all of the attributes and methods to a subclass.  The limitations are that you cannot add new instance variables, and it is not good for use in abstraction.

**10.3 Defining protocols 5m 14s**

**How are protocols useful?**

Protocols standardize interactions between objects and classes without having to formalize it as inheritance.

**10.4 Dynamic typing 11m 33s**

**What are the advantages and disadvantages to dynamic typing?**

Sometimes you don't know the object type before you process it, and so using id simply knows it is an object.  This allows you to dynamically process information.  Also during fast enum we don't have to know the object types, and can just use id instead.