CS121 iOS Tutorial 1

This tutorial provides a basic introduction to Xcode, iOS development and Objective-C, and GitHub. In the process you’ll develop a simple app.

Purple bubbles give you information you’ll need to know.

Orange bubbles tell you what you’re not expected to understand yet.

Yellow Bubbles tell you what to do.

Green bubbles are questions for you to look up and answer!

9/4/13
Launch Xcode then click here to start a new project.
1. We are going to build an iOS application so click here.

2. We'll build a single view application, so click here.

Xcode creates template code based on the type of project you select.

Here is the definition of a single view app.

3. Click on the next button.

9/4/13
1. Name your product `yourinitials.tutorial1`.

2. Use `edu.hmc.cs.yourlogin` as the Company Identifier followed by your name.

Specifying a prefix for our classes can help make our code easier to understand. It’s not necessary for this tutorials, but it does help avoid conflicts.

Reversing the company’s DNS lookup string is a common convention for producing this ID: `edu.hmc.cs.mike`. It won’t cause any problems if you don’t own this domain name.

Use your initials, e.g. MAE.
1. Choose iPhone

2. Make sure Automatic Reference Counting is checked.

3. Click next.

We can create apps for iPhone, iPad or both. We won’t need much screen space for this app so iPhone is good enough.

We’ll talk more about these options later.
1. Navigate to an appropriate folder to save your project folder.

2. Make sure this is unchecked. You already have a global git repository to use.

3. Click create.

**WARNING:** If you are using a CIS laptop and are logged in as guest we suggest you save the folder to the desktop for now. But if you want to save your work, before logging off be sure to move the project to your H drive, Sakai dropbox, or some other site.
Woohoo! You’ve created your first iOS app.

Click the run button.
The app is now running in the iPhone simulator, which appears on your desktop.
Click the stop button to stop running the simulator.
We’ll add to the app but first let’s take a quick look at Xcode.

The navigation panel is on the left.

The editor occupies the center panel.

The utility panel is on the right.
The navigation panel allows you to move between the relevant project files.

The editor is where you will do the bulk of your coding.

The utility panel gives you access to ready-made objects and other resources.
1. Use this button to hide/reveal the navigator panel.

2. The editor panel has a console window that can be hidden/revealed with the center button.

3. Use this button to hide/reveal the utility panel.

What information does the editor panel display?
Xcode created these files for us.

This is the navigator toolbar.

Click this button to navigate the project folder.
Click on the ViewController.h file and it will appear in the editor.

This is a class definition *a la* Objective-C.

This is the ViewController header file. It contains public declarations of methods and properties of the class.
Click on the ViewController.xib file to display the interface builder (IB) editor.

A xib file is an XML file that describes the user interface. Xcode can generate the code for the UI based on the xib file. The Interface Builder (IB) lets us build the xib file by dragging and dropping UI elements to a screen mockup.

iPhone screen mockup.
1. Click on the cube to see the object library.

2. Drag a “label” onto the iPhone screen mockup in the IB editor.
Click here for quick help.
(Make sure the label on the screen mockup is selected.)

This describes the UILabel class, which is built into the iOS framework.
Click here for the attribute editor.

We can edit various attributes of our label in the attribute editor.
Click here for the size editor.

Other properties can be edited in the size editor.
Double click to change the label text to Hello World.

We can also edit the label directly with the mouse and keyboard, as opposed to using the editors.
Before we go any further, let’s build our app! Click on the run button.

You’ll see various build messages here. Eventually you should get a popup message saying the build succeeded.
Here is our label.

Hello World
1. Press stop.

2. Now drag a “button” onto the iPhone screen mockup in the IB editor.
Double click in the button to add the text PRESS.
Run the app. Press the button!

The IB Editor & Xcode are connected such that we can see on the simulator what was created in the Editor.

The XIB File has the specification of the Objects created using the IB Editor.

Next we will tie the IB objects we have created to program actions that we will write.

I know you are thinking this is awesome. 😊 But just wait. Next we’ll make the app count how many times the button is pressed!
1. Stop the simulation.

2. Click on ViewController.h to display the header file.

iOS uses a Model-View-Controller (MVC) architectural pattern. We'll talk more about it in the next lecture. For now, all you need to know is that the ViewController class is going to do the work of counting the button presses. For a detailed presentation on MVC, see Stanford Coding Together Lecture 1.
Class declarations in Objective-C have the following format:
@interface newClassName: parentClassName
{
    // ClassMembers
}
// ClassMethods
@end

The curly brackets are optional when there are no class members.
Add a variable to count the number of button presses.

Add a method to be called when the button is pressed.

An IBAction allows you to connect your controller and view objects in the XIB File to your code.
Method declarations are preceded by + or - symbols. A + signifies a class (static) method; a – signifies an instance method.

The method name is `buttonPressed`

The method takes one parameter, `sender`, which has type `id`.

The method type is IBAction. The compiler resolves IBAction to void. But Xcode recognizes IBAction as signifying the method will be connected to a UI element created in the IB. (We’ll make the connection later)
The syntax for calling this method is [A buttonPressed:sender] where A is a pointer to a ViewController object. The equivalent syntax in C++ would be A->buttonPressed(sender)
Xcode creates lots of template code. We’ll see what all of this does later.

Click on ViewController.m to display the source file.

What is a nib?
Create the `buttonPressed` method as shown.

Initialize our `numPresses` variable here.
Compilation warnings and errors will be highlighted in the source code.

Run the app to make sure it builds successfully before moving on.
The IB editor has created IB code for our objects on screen. We have also written code to manipulate our data. Next we need to “connect” our PRESS button in the UI to our buttonPressed method.
1. Click on .xib file to display the IB editor.

2. If the Placeholders panel is not visible, click on this arrow.

Make sure you can see the Placeholders panel before proceeding.
1. Click on the button.

2. Click the Connections Inspector icon.

The action we want to recognize is when the user lifts her/his finger off the button.
1. Click on the “touch up inside” action.

2. Holding down the mouse button, drag the cursor over to the box labeled File’s Owner. A blue line should trail the cursor.

“The xib file is “owned by” the ViewController object. We’ll talk more about the system architecture later.
The bubble reveals all IBAction methods in the ViewController class. These are the methods that can be called when the button is pressed. In this case there is only one such method: buttonPressed.
The connection inspector now shows that buttonPressed will be called when a “Touch Up Inside” event occurs on our button.
We have now linked the `buttonPressed` method to an action that occurs via the “Button Press” on our screen. At this point the easiest way to check that we are counting button presses is in the debugger. (It is also a great time to introduce you to the debugger!)
1. Open the ViewController.m file.

2. Click here. A blue mark will appear.

The blue mark indicates a break point.
Press the button!
1. If the console is not visible at the bottom of the editor panel, click here.

Execution stops at the breakpoint.

2. Click here to step past the breakpoint.

Note: numPresses = 0.
Now numPresses = 1.
Next we will display the number of button presses in the label field.

Our viewController needs to provide a UILabel reference to the “screen.”
1. Open the ViewController.h file.

2. Add this variable to store a pointer to our UILabel.

3. Add this getter function for the UILabel.

Notice the use of IBOutlet
The IBOutlet keyword is ignored by the compiler. Similar to IBAction, the IBOutlet keyword signifies to Xcode that the variable can be connected to a UI element created in the IB.

Objects cannot be statically allocated in Objective-C! All objects are accessed via pointers and allocated at run time. (Try making numPressesLabel a UILabel rather than a UILabel*)
1. Open the ViewController.m file.

2. Create the getter function.

3. Update the label when the button is pressed.

4. Initialize the label.
numPressesLabel.text = [[NSString alloc] initWithFormat: @”%d”, numPresses;]

We are going to set the text field of our UILabel object.

In C++ we’d say `numPressesLabel->text = ...` because `numPressesLabel` is a pointer.

But the only way to access an object in Objective-C is through a pointer and the dot notation is used.

The good news is you don’t have to remember whether you have an object or a pointer to an object; in Objective-C it will always be the latter.
numPressesLabel.text = [[NSString alloc] initWithFormat: @"%@d", numPresses;]

text is a data member of the UILabel class. It is a pointer to a string; more specifically a pointer to an NSString, which is iOS’s string class.

NSString is immutable. (There is another string class that is mutable.) If we want to change the string pointed to by text we have to allocate new space and initialize it.
numPressesLabel.text = [[NSString alloc] initWithFormat: @"%@d", numPresses;]

Object creation is a two step process. We first allocate space for the string with [NSString alloc].
numPressesLabel.text = [[NSString alloc] initWithFormat: @”%d”, numPresses;]

The second step is to initialize the newly allocated space.

NSString has a variety of initialization methods; here we use initWithFormat.

Note an @ before a string means it is an NSString as opposed to a C-style string.
Now we’ll connect `numPressesLabel` to the label on the screen.

1. Open the `ViewController.xib` file.
2. Select File’s Owner.
1. Holding down the command and control keys, drag the cursor from File Owner to the label; a blue line will trail the cursor.

Click on `numPressesLabel`.

The bubble reveals all IBOutlet variables in the ViewController class. In this case there is only one such variable: `numPressesLabel`.

9/4/13
The connection editor now shows that the label field is connected to our `numPressesLabel` variable.

Run the app.
Click on PRESS

You should see this!
Next you are going to modify the app so that there are two buttons labeled 0 and 1. The text field of the UILabel should say “The last button pressed was n”, where n is either 0 or 1.

Before modifying your tutorial 1, upload the current version to your Sudoku Team GitHub repository. Thus, It is time to do the GitHub tutorials.

There are two things you need to know.
This “sender” is a pointer to the button object that was pressed. We can query that object. For example [sender currentTitle] returns the title of the button. In our current program this would be the NSString @”PRESS”.
The second is how to construct the NSString @”The last button pressed was n” where n is either 0 or 1. See if you can figure out! Help will be provided on request.
Upload your modified tutorial 1 to your GitHub Sudoku Team repository – This does not need to be a new project but rather just an update to your tutorial 1. Be sure that you have differentiated your Tutorial1 from your partner’s.