iPhone Development Basics

Do You have a Mac?
Basics for iPhone

- Need -> OS X operating system.
- Either a real Mac or a system that has OS X installed on it
- Need the iPhone SDK
- This gives you the simulator and X Code installed on your computer
X Code

```swift
// FML_Category.m

// Created by mg on 3/5/09.
// Copyright 2009 GM. All rights reserved.

#import "FML_Category.h"

@implementation FML_Category

// Implementation FML_Category

@synthesize getInstance; // MARK: - FML_Category

@end

@end
```
Iphone Simulator

- Looks like an iPhone
- Runs like an iPhone

- Not an iPhone
- Limited in certain areas
- Decieving
iPhone Simulator Limitations

- No Camera
- None. You cannot use the Mac webcam to take pictures.
- You are limited to the iPhone’s photo library to process images
- Camera requires real hardware
The simulator is very fast. It runs faster than your iPhone.

Not as picky as real hardware.

Harder to process multi touches in simulator.
iPhone Hardware

- Selections between 2G (First Gen), 3G (Second Gen), and 3GS(Second ½ Gen)

- BIG differences in the hardware limitations

- What will run well on the 3GS means nothing for the previous iPhones
iPhone Hardware continued

- iPod touches have no camera. But otherwise same conditions apply.
- First Gen iPod Touches have no bluetooth or speaker
Objective-C Basics

- It’s like a cross between java and C.
- There’s objects like java but C like syntax.

- Objective-C is SUPER-SET of C
  - You can write pure C code for iPhone Apps
  - Problem: Will need to write Objective-C wrappers to do conversion between C data and Objective-C objects
  - Luckily Objective-C has many objects available for you to use
Objective-C Basics

Choose a template for your new project:

- iPhone OS
- Mac OS X

Application
- XCode iPhone Template 2
- Audio Units
- Automator Action
- Bundle
- Command Line Utility
- Dynamic Library
- Framework
- Java
- Kernel Extension
- Standard Apple Plug-ins
- Static Library
- Other

Empty OpenGL ES Application
Navigation-Based Application
OpenGL ES Application
Tab Bar Application

Utility Application
View-Based Application
Window-Based Application

Description: This template provides a starting point for any application. It provides just an application delegate and a window.
Interface Builder
Interface Builder

- Very easy to use and powerful.
- Almost all the default Apple GUI elements included.
- Easily integrated with code via ‘IBOutlet’ connections.
X Code and Interface Builder

```c
UI components
IBOutlet UIButton *btnPic;
BOOL btnPic_selected;
IBOutlet UIButton *btnDesc;
BOOL btnDesc_selected;
IBOutlet UIButton *btnSubmit;
BOOL btnSubmit_selected;
IBOutlet UIImageView *img1;
IBOutlet UIImageView *img2;
IBOutlet UIImageView *img3;
```

1. Take a picture.
2. Add a description.
3. Submit.
X Code
X Code & Objective-C

- Similar file definitions
  - .h for header files
  - .m for Objective-C files
  - Will compile *.c files as well
Creating a new project

- Open up XCode, Select File->New Project
Objective-C basics

- C’s `#include` is Objective-C’s `#import`
  - Like Java but with C syntax
  - i.e. `#import “MyheaderFile.h”`

- Classes are outlined in header files as such
  - `@interface CLASS_NAME : SUPER_CLASS`
    //fill stuff in here with variables and types
  `@end`
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

@interface MapKitAnnotation : NSObject <MKAnnotation, MKReverseGeocoderDelegate> {
    CLLocationCoordinate2D mkcoord;
    NSString *mktitle;
    MKPlacemark *mkplacemark;
}

@property (nonatomic, retain) NSString *mktitle;

-(id)initWithCoordinate:(CLLocationCoordinate2D)coordinate title:(NSString *)title;
-(void)changeCoordinate:(CLLocationCoordinate2D)coordinate;
-(NSString*) subtitle;
@end
Confused?

@interface MapKitAnnotation : NSObject <MKAnnotation, MKReverseGeocoderDelegate> {

- Note: @interface <Class_Name>
- The <Class_Name> : NSObject indicates you are subclassing NSObject
- Technically all Classes you write will subclass somethin
- NSObject <stuff?> - this indicates what delegate methods this class will implement/override
Delegates?

- Everytime you want to write your own custom subclass of something Apple already wrote it’s really easy.
- Just override the delegate methods needed by the class with your own versions
- Like Java overloading
@property?

- @property (nonatomic, retain) <Class_type> <class_name>
  - i.e. @property (nonatomic, retain) NSString *astring;
  - @property indicates you are setting properties for this variable
  - The (nonatomic, retain) indicate type of properties to set.
    - MORE in Apple documentation.
    - (nonatomic, retain) is one of the most common ones
  - By properties I mean GET/SET methods for these values.
  - Objective-C is a OOP language -> prefers you to use get/set methods to assign/read values.
@property?

- @property ( #properties you set# ) sets the properties for the variable.
- Objective-C allows get/set methods to be generated automatically for variables
- Such as: NSString *mystring;
- Set property: @property (nonatomic, retain) NSString* mystring
- This says to the compiler: set GET/SET methods for mystring and keep the variable in memory until I release it
- Require a corresponding @synthesize mystring method
@synthesize?

- When you have a `@property` you must have a corresponding `@synthesize` in the `*.m` file to INIT the properties of the value.

- **NOTE:** `@synthesize` only inits the properties, not the value itself.

- Still must allocate memory for the value or assign a initial value.
Like this:

- Header.h will be like so:
  - #import “someother headerfile.h”

    @interface Test_Class : NSObject
    {
        NSString *mystring;
    }
    @property (nonatomic, retain) NSString *mystring;
    @end
@implementation Test_Class
@synthesize mystring

//other stuff here
@end

Simple right?
How to initialize the string?
Most Objective-C objects will require the following initializing code:
object = [[object_class alloc] init];
Now if you want to set the value you can just do this

[self setMystring:@"Hello World"]

Self is Java’s ‘this’ in reference to the current object class

If another class holds the object, reference that object with that class’s object name instead of self

Same principles apply to function calling

[self functionToCall:Parameters];
wait? What? The?

- `[object_owner setMystring:@”Hello World”]`
- Confused? This is how Objective-C calls functions, in a [ ] fashion.
- Always `[object_owner function:<parameters>]`;

- `Object_owner` is the owner of the function/value you want to call/set
- Same type of call is used for return values:
  - `New_value = [object_owner <value_name>]` will return that value
  - Can be kind of annoying and useful at the same time
Functions

- Header.h will be like so:
  - #import “someother headerfile.h”

    @interface Test_Class : NSObject
    {
        NSString *mystring;
    }
    @property (nonatomic, retain) NSString *mystring;

    - (void) letsprint : (NSString*)astring;
    @end
Within Header.c between @implementation and @end you fill out the function prototype just like C

- (void) printstring : (NSString *)astring
  {
    //print the string to the ‘console’
    NSLog(@”%@”, astring);
  }
**NSLog? %@?**

- NSLog is the function that prints to the Xcode console
- iPhone apps are GUI apps so this is the only way to log out information to the GDB console
- %@ is the %s equivalent in Objective-C
- **NOTE:** ALL STRINGS MUST HAVE A ‘@’ in front
Calling the function

- The function can be called somewhere in another function like thus `[self letsprint:@"hey"];`

- And the GDB console will print “hey”:
Whenever a new project is created.

- A XXX_XXXAppDelegate.h & *.m file are created.
- These are executed by the apps main.c (which you will never have to modify hopefully) and displays the initial screen.
- The loading function for every app is:
  -(void)applicationDidFinishLaunching
Objects are created thus:
new_object = [[A_CLASS alloc] init];

To release this later you will call [new_object release];
PLEASE be careful about memory leaks. On an iPhone can severly impact performance.
You can usually release everything in an objects
-(void)dealloc() method. This is the default method called when object is closed or ‘released’
Objective C is not too picky about memory initialization. Won’t always crash but your application might not work. Make sure you always alloc and init your objects
The MVC model

- Model-View-Control model
- Makes it so all elements are separate from each other

- The view (GUI), the data (Model) and the interaction between (Control)

- May fit you may not. Do what is best and doesn’t break your code easily or frustrate you
Basic Apple GUI elements to note

- **UIView** – basic view: shown to user; add gui elements to the objects view to show
- **UIViewController** - Controls a ‘view’. Think of this as literally the guardian of the UIView. Used with navigation controls
- **UINavigationController** - that navigation bar on top with a ‘Back’ button

- **UITabBarController** - bottom row tab bar with buttons
More GUI elements

- UIWebView -> similar to safari html browser
- UITableViewController -> lists data in a cell like fashion
- UITextField
- UITextViewArea
- UILabel
- UIButton
- Etc…
Any questions?

- Hmm.
- Google is your friend.
- www.iphonedevsdk.com is a good resource.

http://www.iphonedevcentral.org/home.php
Site with lots of video tutorials. Interface Builder is confusing at first, so all the basic tutorials here can get you up to speed.

http://www.iponedevsdk.com
great site for beginners with a forum for questions.

http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html
basic introduction from apple to application programming

basic objective-C introduction; useful for the little introduction stuff and a primer

where the above links came from; Don't need an account to download the SDK or view the documents. Accounts only good for real iphone device development